Online Courseware for B.TechComputer Science and Engineering Program(Autonomy)
Paper Name: Web Technology
Paper Code: CS702C

Name of the Paper: Web Technology
Paper Code: CS702C
Contact (Periods/Week): 3L/Week
Credit Point: 3
No. of Lectures: 35

Course Objective(s):

- To impart the design, development and implementation of Static and Dynamic Web Pages.
- To develop programs for Web using Scripting Languages and .net framework.
- To give an overview of Server Side Programming in Web

Course Outcome(s) (CO):

CS702C.1 To understand the notions of World Wide Web(www), Internet, HTTP Protocol, Web Browsers, Client-Server etc.
CS702C.2 To develop interactive web pages using HTML, DHTML and CSS.
CS702C.3 To procure the knowledge of different information interchange formats like XML.
CS702C.4 To design web applications using scripting languages like JavaScript, CGI, PHP.
CS702C.5 To acquire the server side programming concepts using servlet, JSP and .Net framework.

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C01	1	-	2	-	-	-	-	-	-	-	-	-
C02	-	2	3	2	-	-	-	-	-	-	-	-
C03	-	2	2	-	-	-	-	-	-	-	-	-
C04	1	2	3	2	-	-	-	-	-	-	-	-
C05	1	-	3	2	-	-	-	-	-	-	-	-

PROPOSED THEORY SYLLABUS

Module 1: [4L]
Introduction to Web [4L]: Concept of World Wide Web (www), Internet and the relation with www **[1L];** The Internet - Basic Internet Protocols, HTTP Protocol - Request and Response, Web browser **[1L];** Web clients and Web servers, Dynamic IP **[1L];** Clients, Servers, and Communication, Web site design principles, Planning the site and navigation **[1L].**

Module -2: [9L]
HTML, DHTML & CSS [6L]: Introduction, Elements, Attributes, Heading, Paragraph. Formatting **[1L];** Link, Table, List, Block, Layout, Html Forms and input **[1L];** Iframe, Colors, Image Maps and attributes of image area **[2L];**Introduction to CSS, basic syntax and structure of

CSS, different types- internal, external and inline CSS **[1L];** Basic Introduction of DHTML, Difference between HTML and DHTML, Documentary Object Model (DOM) **[1L].**
Extended Markup Language (XML) [3L]: Introduction, Difference between HTML & XML, XML-Tree **[1L];** Syntax, Elements, Attributes, Validation and parsing, DTD **[2L].**

Module 3: [8L]
Java Scripts [3L]: Basic Introduction, Statements, comments, variable, operators, data types**[1L];** condition, switch, loop, break **[1L];** Java script functions, objects and events**[1L].**
CGI Scripts [1L]: Introduction, Environment Variable, GET and POST Methods.
PHP Scripting [4L]: Introduction, Syntax, Variables, Output, Data types, String, Constants**[1L];** Operator, Decision Control statements**[1L];** switch-case, Loop, PHP function**[1L];** array, Form Handling**[1L].**

Module-4: [14L]
Java Server Page (JSP) [8L]:
JSP Architecture **[1L];** JSP Servers, JSP Life Cycle **[1L];** Understanding the layout of JSP, JSP Scriptlet Tag **[1L];** JSP implicit object (request and response) **[1L];** Variable declaration, methods in JSP **[1L];**JSP directive (Taglib and Include), JavaBean- inserting JavaBean in JSP **[1L];**JSP Action tags (Forward & Include) **[1L];** Creating ODBC data source name, Introduction to JDBC, prepared statement and callable statement **[1L].**
Java Servlet [3L]: Servlet environment and role, Servlet life cycle **[1L];** Servlet methods- Request, Response, Get and post **[1L];** Cookies and Session **[1L].**
.NET Framework [3L]:ASP.Net with MVC introduction, MVC Architecture, MVC routing, controller, Action method, Action Selector and Action verb, Model and View **[1L];**.net framework, C#.net introduction, environment variable, basic syntax of conditional statement, loop and function**[2L].**

Text Books:

1. “Web Technology: A Developer's Perspective”, N.P. Gopalan and J. Akilandeswari, PHI Learning, Delhi, 2013. **(Topics covered: html, CSS, imagemap, xml)**
2. “Learning PHP, MySQL & JavaScript”, Robin Nixon, O’Reilly Publication.(**Topics covered: PHP, Java Script)**
3. “Head First Servlet’s & JSP”, Bryan Basham, Kathy Sterra, Bert Bates, O’Reilly Publication. (**Topics covered: Servlet, JSP)**
4. ASP.NET Core 2.0 MVC And Razor Pages For Beginners:” Jonas Frajerberg, O’Reilly Publication.(**Topics covered: MVC, ASP.Net, C#)**

Recommended books:

1. "Programming the World Wide Web", Robert. W. Sebesta, Fourth Edition, Pearson Education, 2007.
2. ”Core Web Programming”- Second Edition-Volume I and II, Marty Hall and Larry Brown, Pearson Education, 2001.

MODULE 1: Introduction to Web

Lecture 1: World Wide Web and Internet

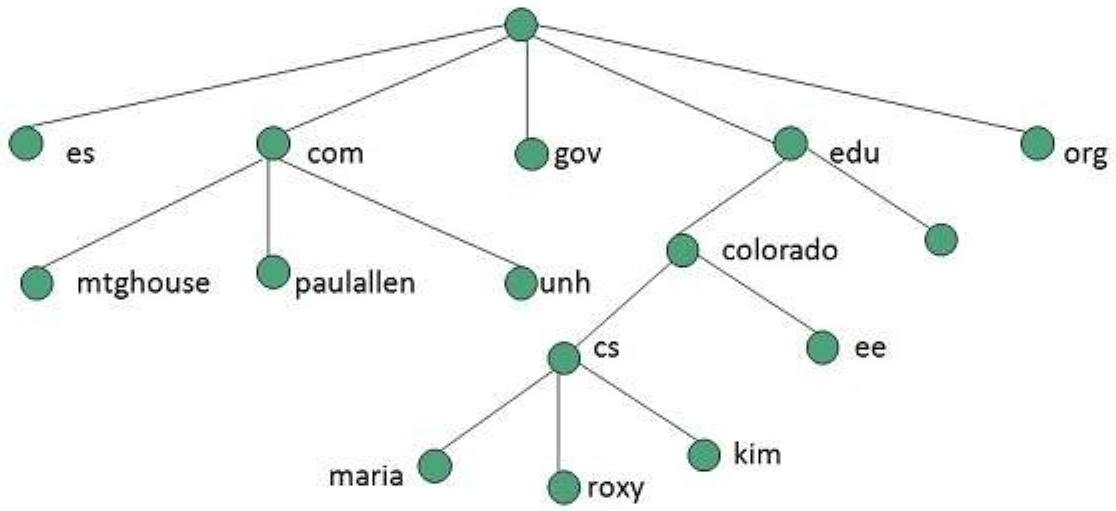
1.1. Concept of World Wide Web (www):

WWW stands for World Wide Web. A technical definition of the World Wide Web is: all the resources and users on the Internet that are using the Hypertext Transfer Protocol (HTTP).

A broader definition comes from the organization that Web inventor Tim Berners-Lee helped found, the World Wide Web Consortium (W3C).

“The World Wide Web is the universe of network-accessible information, an embodiment of human knowledge.”

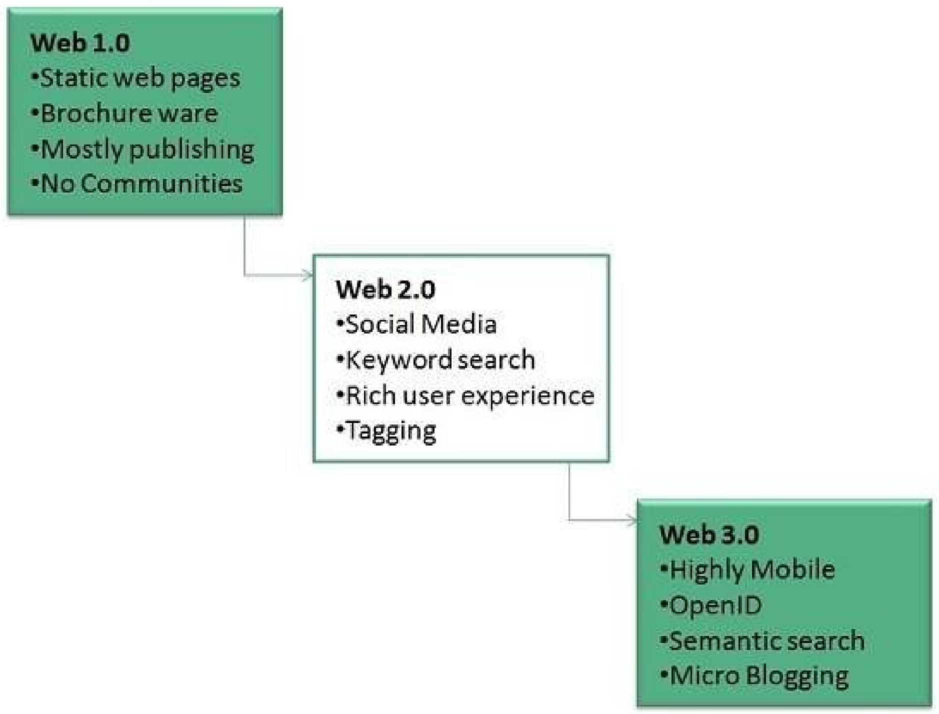
In simple terms, The World Wide Web is a way of exchanging information between computers on the Internet, tying them together into a vast collection of interactive multimedia resources.



1.1.1. Evolution:

World Wide Web was created by Timothy Berners Lee in 1989 at CERN in Geneva. World Wide Web came into existence as a proposal by him, to allow researchers to work together effectively and efficiently at CERN. Eventually it became World Wide Web.

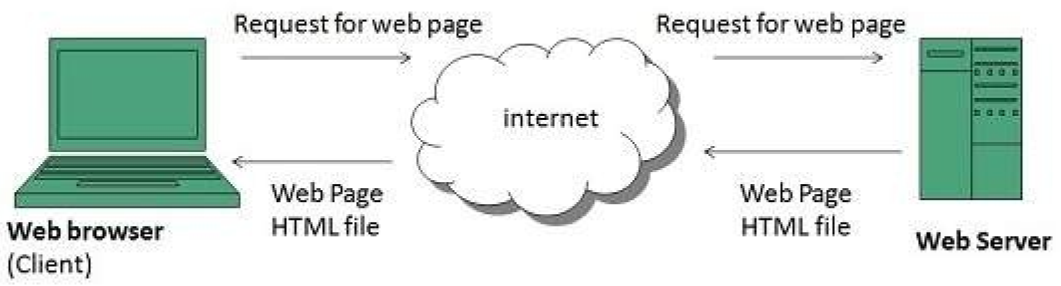
The following diagram briefly defines evolution of World Wide Web:



1.1.2. WWW Operation:

WWW works on client- server approach. Following steps explains how the web works:

1. User enters the URL (say, http://www.google.co.in/) of the web page in the address bar of web browser.
2. Then browser requests the Domain Name Server for the IP address corresponding to www.google.co.in.
3. After receiving IP address, browser sends the request for web page to the web server using HTTP protocol which specifies the way the browser and web server communicates.
4. Then web server receives request using HTTP protocol and checks its search for the requested web page. If found it returns it back to the web browser and close the HTTP connection.
5. Now the web browser receives the web page, it interprets it and display the contents of web page in web browser's window.



1.2. Internet and the relation with WWW:

The Web, as it's commonly known, is often confused with the internet. Although the two are intricately connected, they are different things. WWW is all the Web pages, pictures, videos and other online content that can be accessed via a Web browser. The Web is a communications model that, through HTTP, enables the exchange of information over the internet.The Internet, in contrast, is the underlying network connection that allows us to send email and access the World Wide Web.As its name implies, Internet is a network -- a vast, global network that incorporates a multitude of lesser networks. As such, the internet consists of supporting infrastructure and other technologies.

Tim Berners-Lee is the inventor of the Web and the director of the W3C, the organization that oversees its development. Berners-Lee developed hypertext, the method of instant cross-referencing that supports communications on the Web, making it easy to link content on one web page to content located elsewhere. The introduction of hypertext revolutionized the way people used the internet.

In 1989, Berners-Lee began work on the first World Wide Web server at CERN. He called the server "httpd" and dubbed the first client "WWW." Originally, WWW was just a WYSIWYG hypertext browser/editor that ran in the NeXTStep environment.

The World Wide Web has been widely available since 1991.

Lecture 2: Basic Internet Protocols, HTTP Protocol and Web Browser

2.1. The Internet - Basic Internet Protocols:

A means of connecting a computer to any other computer anywhere in the world via dedicated routers and servers. When two computers are connected over the Internet, they can send and receive all kinds of information such as text, graphics, voice, video, and computer programs.

No one owns Internet, although several organizations the world over collaborate in its functioning and development. The high-speed, fiber-optic cables (called backbones) through which the bulk of the Internet data travels are owned by telephone companies in their respective countries.

The Internet grew out of the Advanced Research Projects Agency's Wide Area Network (then called ARPANET) established by the US Department Of Defense in 1960s for collaboration in military research among business and government laboratories.

Later universities and other US institutions connected to it. This resulted in ARPANET growing beyond everyone's expectations and acquiring the name 'Internet.'

The development of hypertext based technology (called World Wide Web, WWW, or just the Web) provided means of displaying text, graphics, and animations, and easy search and navigation tools that triggered Internet's explosive worldwide growth.

2.1.1. Basic Internet Protocols:

When most people talk about "the Internet" what they are really referring to is the World Wide Web. The Internet is actually composed of many different components. Some of the components are widely known, such as FTP, while others are not so familiar, such as Gopher and Telnet.

Several protocols are used on the Internet, including Electronic Mail (e-mail), File Transfer Protocol (FTP), HTTP (World Wide Web), News (or Usenet), Gopher and Telnet. Each of these has its own standard and usage.

2.1.1.1. Electronic Mail:

Included in the email protocol are three distinct protocols. SMTP (Simple Mail Transfer Protocol), IMAP (Internet Message Access Protocol) and POP3 (Post Office Protocol 3). SMTP is a protocol used for sending mail, while IMAP and POP3 are used for receiving. Almost all Internet service providers support all three protocols. However the most popular setup for most providers is to use SMTP for sending mail while using POP3 for receiving.

2.1.1.2. File Transfer Protocol:

File Transfer Protocol, or FTP, is a means of transferring a file from one computer to another. FTP is commonly used for uploading a web page to a web server so that it may be seen on the World Wide Web. A special program, called a client, is usually needed to use FTP.

2.1.1.3. News (or Usenet):

Network News Transfer Protocol (NNTP) is used for serving Usenet posts Usenet is similar to the forums that many web sites have. Usenet has forums that are dedicated to specific companies as well as forums that have a wide range of topics. Usenet is divided into several areas. Some of the forums that are included in Usenet are comp. for discussion of computer-related topics, sci. for discussion of scientific subjects, rec. for discussion of recreational activities (e.g. games and hobbies) and talk. for discussion of contentious issues such as religion and politics.

2.1.1.4. Gopher:

Another tool of the Internet is Gopher, a menu-based program that enables you to browse for information without knowing where the material is located. It lets you search a list of resources and then sends the material to you.

2.1.1.5. Telnet:

Telnet lets you log in to a remote computer just as you would if you were there. So any commands that you would be able to run from the remote computer if you were sitting in front of it, you would be able to run from the computer you logged in from.

2.2. HTTP Protocol - Request and Response:

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

Basically, HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

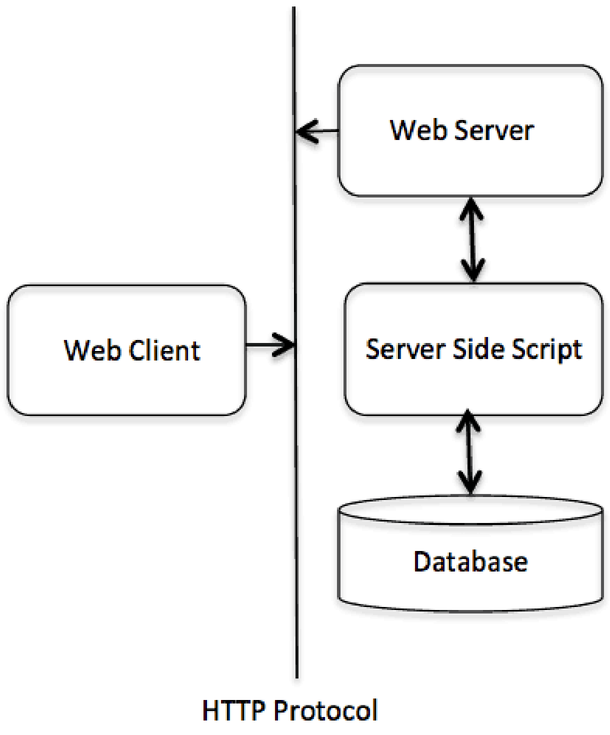
2.2.1. Basic Features:

There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connectionless:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.
- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
- **HTTP is stateless:** As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

2.2.2. Basic Architecture:

The following diagram shows a very basic architecture of a web application and depicts where HTTP sits:



The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

2.2.3. Client:

The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.

2.2.4. Server:

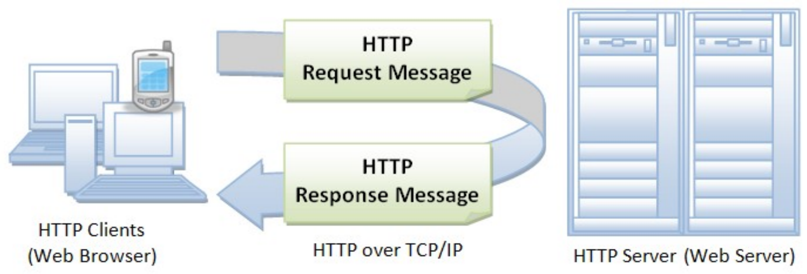
The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.

2.2.5. Request and Response:

The operation of Hypertext Transfer Protocol (HTTP) involves the communication between a Hypertext Transfer Protocol (HTTP) client application (Usually web browser) and a Hypertext Transfer Protocol (HTTP) server application (Web servers like IIS). Hypertext Transfer Protocol (HTTP) uses Transmission Control Protocol (TCP) as the Transport Layer Protocol at Well Known port number 80. Once the TCP connection is established, the two steps in Hypertext Transfer Protocol (HTTP) communication are:

1. **HTTP Client Request:** Hypertext Transfer Protocol (HTTP) client sends an Hypertext Transfer Protocol (HTTP) Request to the Hypertext Transfer Protocol (HTTP) Server according to the HTTP standard, specifying the information the client like to retrieve from the Hypertext Transfer Protocol (HTTP) Server.

2. **HTTP Server Response:** Once the Hypertext Transfer Protocol (HTTP) Request arrived at the Hypertext Transfer Protocol (HTTP) server, it will process the request and creates a Hypertext Transfer Protocol (HTTP) Response message. The Hypertext Transfer Protocol (HTTP) response message may contain the resource the Hypertext Transfer Protocol (HTTP) Client requested or information why the Hypertext Transfer Protocol (HTTP) request failed.



2.2. Web Browser:

Web Browser is an application software that allows us to view and explore information on the web. User can request for any web page by just entering a URL into address bar.

Web browser can show text, audio, video, animation and more. It is the responsibility of a web browser to interpret text and commands contained in the web page.

Earlier the web browsers were text-based while now a days graphical-based or voice-based web browsers are also available. Following are the most common web browser available today:

Browser	**Vendor**
Internet Explorer	Microsoft
Google Chrome	Google
Mozilla Firefox	Mozilla
Netscape Navigator	Netscape Communications Corp.
Opera	Opera Software
Safari	Apple
Sea Monkey	Mozilla Foundation
K-meleon	K-meleon

Lecture 3: Web clients, Web servers and Dynamic IP

3.1. Web clients and Web servers:

3.1.1. Web clients:

The Web client is a client-side component, a distributed multi-tiered application model used for building and developing enterprise applications. Client-side components are typically computer applications running on a user's computer and connect to a server. These components perform client-side operations as they might need access to information available only on the client side, like user input, or because the server lacks the processing power necessary in such operations.

A Web client contains two parts: dynamic Web pages and the Web browser. Dynamic Web pages are produced by components that run in the Web tier, and a Web browser delivers Web pages received from the server.

A Web client is also known as a thin client because it does not execute heavy-duty operations such as querying databases, performing complex business tasks, or connecting to legacy applications.

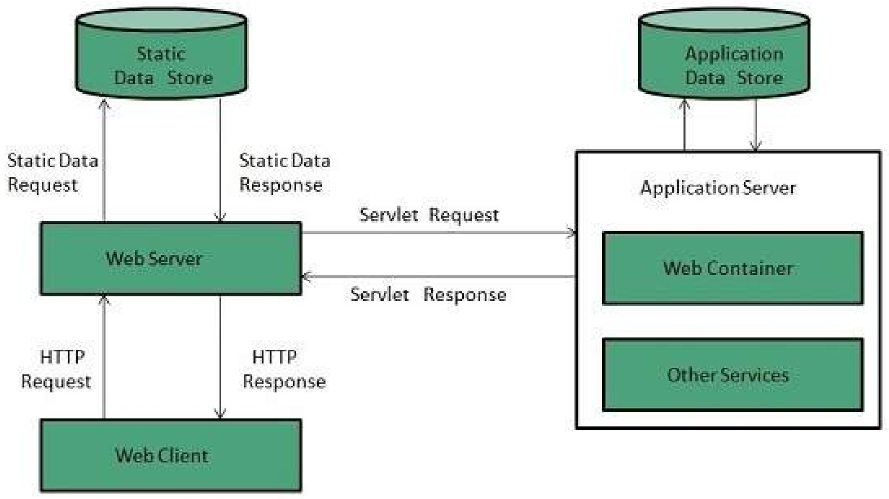
3.1.2. Web server:

Web server is a computer where the web content is stored. Basically web server is used to host the web sites but there exists other web servers also such as gaming, storage, FTP, email etc.

3.1.2.1. Web Server Working:

Web server respond to the client request in either of the following two ways:

- Sending the file to the client associated with the requested URL.
- Generating response by invoking a script and communicating with database.



3.1.2.2. Key Points:

- When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response.
- If the requested web page is not found, web server will the send an HTTP response:Error 404 Not found.
- If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

3.1.2.3. Examples:

Following table describes the most leading web servers available today:

Sl.No.	Web Server Description
1	**Apache HTTP Server** This is the most popular web server in the world developed by the Apache Software Foundation. Apache web server is an open source software and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more. About 60% of the web server machines run the Apache Web Server.
2.	**Internet Information Services (IIS)** The Internet Information Server (IIS) is a high performance Web Server from Microsoft. This web server runs on Windows NT/2000 and 2003 platforms (and may be on upcoming new Windows version also). IIS comes bundled with Windows NT/2000 and 2003; Because IIS is tightly integrated with the operating system so it is relatively easy to administer it.
3.	**Lighttpd** The lighttpd, pronounced lighty is also a free web server that is distributed with the

	FreeBSD operating system. This open source web server is fast, secure and consumes much less CPU power. Lighttpd can also run on Windows, Mac OS X, Linux and Solaris operating systems.
4.	**Sun Java System Web Server** This web server from Sun Microsystems is suited for medium and large web sites. Though the server is free it is not open source. It however, runs on Windows, Linux and UNIX platforms. The Sun Java System web server supports various languages, scripts and technologies required for Web 2.0 such as JSP, Java Servlets, PHP, Perl, Python, and Ruby on Rails, ASP and Coldfusion etc.
5.	**Jigsaw Server** Jigsaw (W3C's Server) comes from the World Wide Web Consortium. It is open source and free and can run on various platforms like Linux, UNIX, Windows, and Mac OS X Free BSD etc. Jigsaw has been written in Java and can run CGI scripts and PHP programs.

3.2. Dynamic IP:

A dynamic Internet Protocol address (dynamic IP address) is a temporary IP address that is assigned to a computing device or node when it’s connected to a network. A dynamic IP address is an automatically configured IP address assigned by a DHCP server to every new network node.

Dynamic IP addresses are generally implemented by Internet service providers and networks that have a large number of connecting clients or end-nodes. Unlike static IP addresses, dynamic IP addresses are not permanent. A dynamic IP is assigned to a node until it’s connected to the network; therefore, the same node may have a different IP address every time it reconnects with the network.

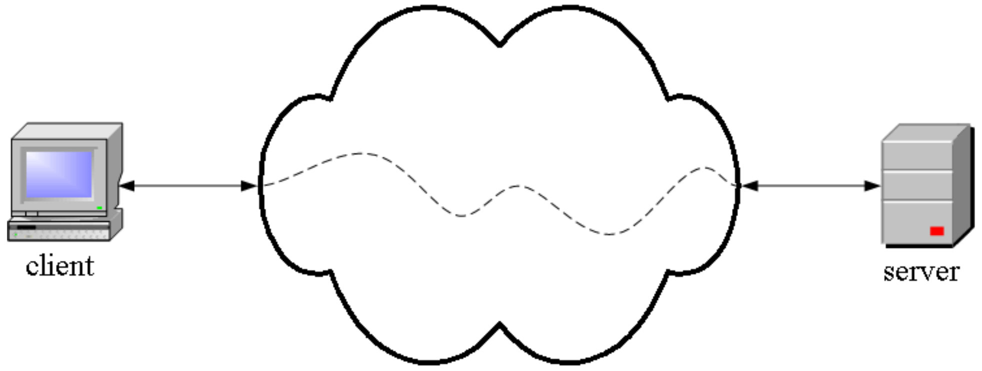
The assigning, reassigning and modification of dynamic IP addresses is managed by a Dynamic Host Configuration Protocol (DHCP) server. One of the primary reasons behind having dynamic IP addresses is the shortage of static IP address on IPv4. Dynamic IP addresses allow a single IP address to be shuffled between many different nodes to circumvent this problem.

Lecture 4: Clients-ServersCommunication, Website design principles, Site planning and navigation

4.1. Clients, Servers, and Communication:

In network **applications**, a **client** and **server** work together to get useful work done.

Clients and servers communicate with each other across the Internet or any other network. (**Client-server applications** are also common on **local area networks**). The client requests some **service** and the server performs it (or returns an error message if there is a problem).



The Web is a familiar client-server application. To see a Web page, you run a Web client like Internet Explorer or Firefox. When you enter a **URL** and hit enter, the client encloses it in a Get message which it sends across the network to a Web server. If the document is on that Web server, it sends it back to the client. If not, it sends an error message (formatted as a Web page). When the client receives the page (pr error message), it formats and displays it.

1. the user enters a URL and hits enter
2. the client program sends a Get message containing the URL to the waiting server

3. if the server has the requested document, it sends it back to the client
4. if the server does not have the document, it sends an error message back to the client
5. the client displays whatever was returned
6. the server waits for the next request

This dialog between the client and the server follows a specific pattern or **protocol**. The protocol specifies the format of messages that can be sent and their meanings. The Web protocol is called the **Hypertext Transfer Protocol** (HTTP).

Note that when information is transferred from a server to a client as in this case, we speak of **downloading** and when we transfer information to a server we speak of **uploading**.

Surfing the Web with a Web client is a popular application, but there are many other application protocols, for example:

Protocol	Application
HTTP, Hypertext Transfer	retrieving and viewing web pages
FTP, File Transfer	copy files from client to server or from server to client
SMTP, Simple Mail Transport	send email
POP, Post Office	read email

In some applications, a computer is programmed to be both a client and a server. These are called **peer to peer** applications.

Note that the terms "client" and "server" refer both to the computer hardware and the program it is running. When you are looking at things on the Internet, your computer is the client hardware and it is running a client program.

In the case of the Web, people also refer to a client program as a **browser** since it is used to browse through the contents of Web sites.

4.2. Web site design principles:

The success of any website entirely depends on how its web design is. Whether the designers have taken utmost care or not while developing it. Your nicely-designed site which includes usability and utility, determines the success, and not the visual design. Since your site is the face of your business and most potential customers will visit your site before they ever look in on your store, it becomes inevitable to get your website designed cautiously. Lacking in any aspect could end up demolishing your brand impression.

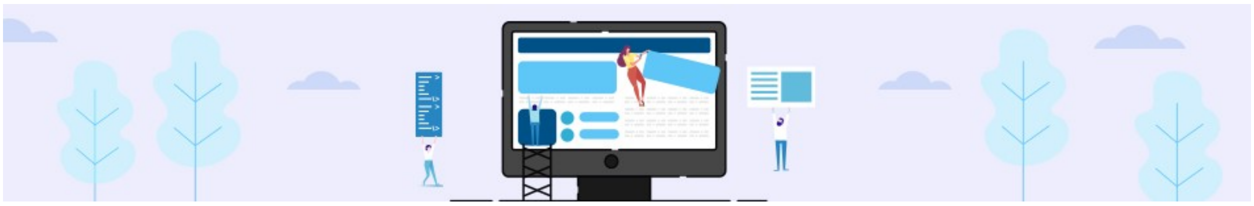
Web page design is more critical for conversions than you think. Despite using a great conversion boosting tactic, you may not do much if it looks poor in quality. In fact, website design doesn’t necessarily mean how it looks like and feels like, but is how it works. Even a simple looking website with exceptional usability and well-structured, typically performs amazingly on Google. User views of such websites are also higher than those with poor user experience. The performance entirely depends on the effectiveness of the website.

So, how to design a good website?

Good website design needs a wide range of professionals having expertise in different areas. Their collective efforts need to put in when there is a critical decision to take place. Here in this article, we'll outline the 8 essential principles of good website design which must be pondered while developing a website. These design principles will definitely help web designers to develop awe-inspiring designs and to enhance the usability of a website.

Here is the list of 8-good design principles which will make your website aesthetic, user-friendly, effective, and engaging:

4.2.1. Simple is the best:



Over-designed website may not work. Putting too many elements on the page may lead to distract visitors from the main purpose of your website. Simplicity always works in an effective web page design. Clean and fresh design of your website not only makes the website appealing, but also help user to navigate from one page to another seamlessly. Loading a website having design features which do not serve the purpose may be frustrating. Keep your design as simple as possible so that the visitors can feel it easy-to-use and can find their ways easily.

4.2.2. Consistency:

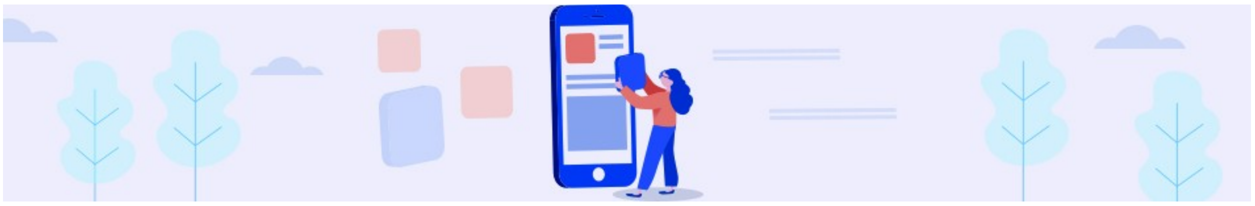
Consistency in website design matter a lot. Give your attention to match design elements throughout each of the pages. It can be understood that your fonts, sizes, headings, sub-headings, and button styles must be the same throughout the website. Plan everything in advance. Finalize the fonts and the right colors for your texts, buttons etc., and stick to them throughout the development. CSS (Cascading Style Sheets) would come in handy to keep the complete information about design styles and elements.

4.2.3. Typography & Readability:

No matter how good your design is text still rules the website as it provides users the desired information. Since search engine crawlers are very much familiar with this data, it becomes an integral part of SEO activities. You should keep your typography visually appealing and readable for visitors, along with tricky use of keywords, meta-data, and other SEO-sensitive elements.

Consider using fonts that are easier to read. The modern sans-serif fonts as Arial, Helvetica etc. can be used for the body texts. Make proper combinations of typefaces for each and every design elements such as headlines, body texts, buttons etc.

4.2.4. Mobile compatibility:



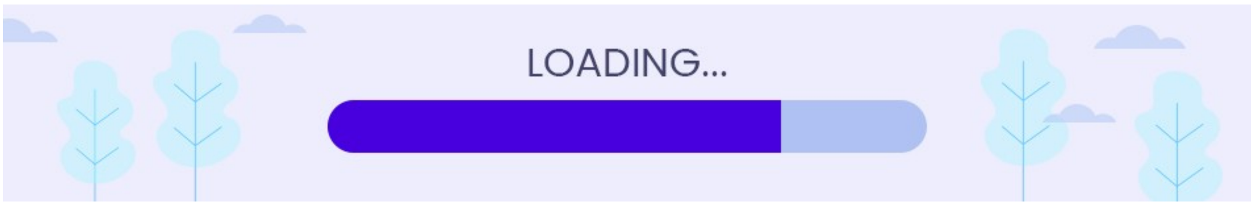
Keeping in mind the ever-growing usage of smartphones, tablets and phablets, web design must be effective for various screens. If your website design doesn't support all screen sizes, chance is that you'll lose the battle to your competitors. There are a number of web design studios or service points from where you can turn your desktop design into a responsive and adaptive one for all screen sizes.

4.2.5. Color palette and imagery:

A perfect color combination attracts users while a poor combination can lead to distraction. This necessitates you to pick a perfect color palette for your website which can create a pleasing atmosphere, thus leaving a good impact on visitors. Enhance users experience by selecting complementary color palette to give a balanced-look to your website design. Remember to white space use as it avoids your website from visual clutter and mess. Also avoid using too many colors. 3 or 4 tones for the whole websites are ample to give appealing and clear design.

Same is the case with images. Don't use multiple vibrant images.

4.2.6. Easy loading:



No one likes the website that takes too much time to load. So take care of it by optimizing image sizes, combing code into a central CSS or JavaScript file as it reduces HTTP requests. Also, compress HTML, JavaScript and CSS for enhanced loading speed.

4.2.7. Easy Navigation:

Study shows that visitors stay more time on the websites having easy navigation. For effective navigation, you may consider creating a logical page hierarchy, using bread scrums, and designing clickable buttons. You should follow the “three-click-rule” so that visitors can get the required information within three clicks.

4.2.8. Communication:

The ultimate purpose of the visitors is to get information, and if your website is able to communicate your visitors efficiently, most probably they would spend more time on your website. Tricks that may work to establish effortless communication with the visitors are – organizing information by making good use of headlines and sub-headlines, cutting the waffle, and using bullet points, rather than long gusty sentences.

To wrap it up

Keeping the aforementioned principles of good website design, you can easily develop an aesthetic and functional website. Without this base, it would be difficult to travel a long path. Only with a clean and user-friendly design, you can think to succeed.

4.3. Planning the site and navigation:

Web navigation refers to the process of navigating a network of information resources in the World Wide Web, which is organized as hypertext or hypermedia. The user interface that is used to do so is called a web browser.

A central theme in web design is the development of a web navigation interface that maximizes usability.

A website's overall navigational scheme includes several navigational pieces such as global, local, supplemental, and contextual navigation; all of these are vital aspects of the broad topic of web navigation. Hierarchical navigation systems are vital as well since it is the primary navigation system. It allows for the user to navigate within the site using levels alone, which is often seen as restricting and requires additional navigation systems to better structure the website. The global navigation of a website, as another segment of web navigation, serves as the outline and template in order to achieve an easy maneuver for the users accessing the site, while local navigation is often used to help the users within a specific section of the site. All these navigational pieces fall under the categories of various types of web navigation, allowing for further development and for more efficient experiences upon visiting a webpage.

4.3.1. History:

Web navigation came about with the introduction of the World Wide Web, in 1989 when Timothy Berners-Lee invented it. Once the World Wide Web was available, web navigation increasingly became a major aspect and role in jobs and everyday lives. With one-third of the world’s population now using the internet, web navigation maintains a global use in today’s ever evolving international society. Web navigation is not restricted to just computers, either, as mobile phones and tablets have added avenues for access to the ever-growing information on the web today. The most recent wave of technology which has affected web navigation is the introduction and growth of the smartphone. As of January 2014, 58% of American adults owned a smart phone, and that number is on the rise from previous years. Web navigation has evolved from a restricted action, to something that many people across the world now do on a daily basis.

4.3.2. Types of web navigation:

The use of website navigation tools allow for a website's visitors to experience the site with the most efficiency and the least incompetence. A website navigation system is analogous to a road map which enables webpage visitors to explore and discover different areas and information contained within the website.

There are many different types of website navigation:

1. **Hierarchical website navigation:**The structure of the website navigation is built from general to specific. This provides a clear, simple path to all the web pages from anywhere on the website.

2. **Global website navigation:**Global website navigation shows the top level sections/pages of the website. It is available on each page and lists the main content sections/pages of the website.

3. **Local website navigation:**Local navigation is the links within the text of a given web page, linking to other pages within the website.

4.3.3. Styles of web navigation:

Web navigations vary in styles between different website as well as within a certain site. The availability of different navigational styles allows for the information in the website to be delivered easily and directly. This also differentiates between categories and the sites themselves to indicate what the vital information is and to enable the user's access to more information and facts discussed within the website. Across the globe, different cultures prefer certain styles for web navigations, allowing for a more enjoyable and functional experience as navigational styles expand and differentiate.

- **Navigation bar:** A navigation bar or (navigation system) is a section of a website or online page intended to aid visitors in travelling through the online document.

- **Sitemap:** A site map (or sitemap) is a list of pages of a web site accessible to crawlers or users. It can be either a document in any form used as a planning tool for Web design, or a Web page that lists the pages on a Web site, typically organized in hierarchical fashion.

- **Dropdown menu:** In computing with graphical user interfaces, a dropdown menu or drop-down menu or drop-down list is a user interface control GUI element ("widget" or "control"), similar to a list box, which allows the user to choose one value from a list.
- **Flyout menu:** In computing with graphical user interfaces, a menu that flies out (either down or to the side) when you click or hover (mouseover) some GUI element.

- **Named anchor:** An anchor element is called an anchor because web designers can use it to anchor a URL to some text on a web page. When users view the web page in a browser, they can click the text to activate the link and visit the page whose URL is in the link.

4.3.4. Design of web navigation:

What makes Web design navigation difficult to work with is that it can be so versatile. Navigation varies in design through the presence of a few main pages in comparison to multi-level architecture. Content can also vary between logged-in users and logged-out users and more. Because navigation has so many differences between websites, there are no set guidelines or to-do lists for organizing navigation. Designing navigation is all about using good information architecture, and expressing the model or concept of information used in activities requiring explicit details of complex systems.

4.3.5. Future of web navigation:

4.3.5.1. Adaptive website navigation:

Adaptive web navigation describes the process of real-time changes in a website's navigation links and layout according to individual user preferences as they browse the site. Innovative websites are increasingly attempting to automatically personalize web sites based on a user's browsing pattern in order to find relevant information more quickly and efficiently. The usage of data analysis allows website creators to track behavior patterns of a user as they navigate a site. Adding shortcut links between two pages, rearranging list items on a page, and omitting irrelevant navigation links are all examples of adaptive changes that can be implemented in real-time. The advantage of utilizing adaptive technologies in web navigation is it reduces the time and navigational effort required for users to find information. A possible disadvantage of this is that users may get disoriented from global and local navigational changes from page to page.

MODULE 2: HTML, DHTML, CSS& XML

Lecture 1: Introduction, Elements, Attributes, Heading, Paragraph, Formatting

1.1. Introduction:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

Example Explained

- The DOCTYPE declaration defines the document type
- The text between <html> and </html> describes the web page
- The text between <body> and </body> is the visible page content
- The text between <h1> and </h1> is displayed as a heading
- The text between <p> and </p> is displayed as a paragraph

1.1.1. What is HTML?

HTML is a language for describing web pages.

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage

- HTML is a **markup** language
- A markup language is a set of markup **tags**
- The tags **describe** document content
- HTML documents contain HTML **tags** and plain **text**
- HTML documents are also called **web pages**

1.1.2. HTML Tags:

HTML markup tags are usually called HTML tags.

- HTML tags are keywords (tag names) surrounded by **angle brackets** like <html>
- HTML tags normally **come in pairs** like <p> and </p>
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, with a **slash** before the tag name
- Start and end tags are also called **opening tags** and **closing tags**<tagname>content</tagname>

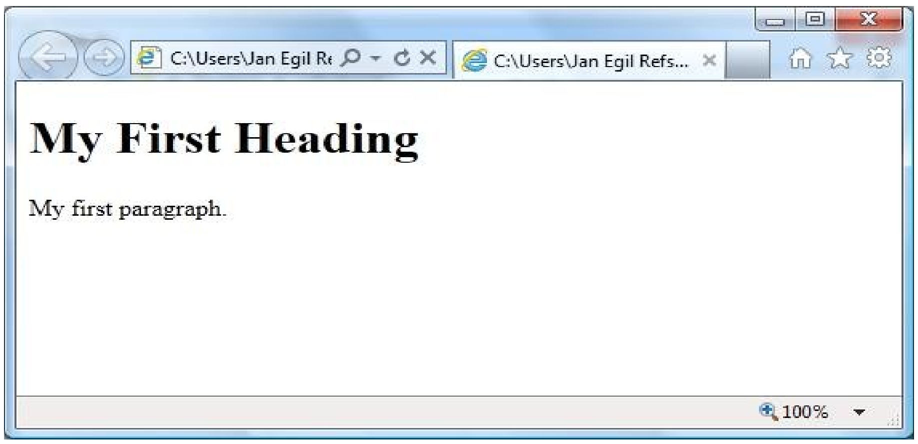
1.1.3. HTML Elements:

In HTML, most elements are written with a start tag (e.g. <p>) and an end tag (e.g. </p>), with the content in between:
<p>This is a paragraph.</p>

1.1.4. Web Browsers:

The purpose of a web browser (such as Google Chrome, Internet Explorer, Firefox, Safari) is to read HTML documents and display them as web pages.

The browser does not display the HTML tags, but uses the tags to determine how the content of the HTML page is to be presented/displayed to the user:



My First Heading

My first paragraph.

1.1.5. HTML Page Structure:

Below is a visualization of an HTML page structure:

```
<html>
<body>
<h1>This a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

1.1.6. The <!DOCTYPE> Declaration:

The <!DOCTYPE> declaration helps the browser to display a web page correctly. There are many different documents on the web, and a browser can only display an HTML page 100% correctly if it knows the HTML version and type used.

Common Declarations

HTML5

```
<!DOCTYPE html>
```

1.1.7. Write HTML Using Notepad or TextEdit:

HTML can be edited by using a professional HTML editor like:

- Adobe Dreamweaver
- Microsoft Expression Web
- Coffee Cup HTML Editor

However, for learning HTML we recommend a text editor like Notepad (PC) or TextEdit (Mac).

We believe using a simple text editor is a good way to learn HTML.

Follow the 4 steps below to create your first web page with Notepad.

Step 1: Open Notepad

To open Notepad in Windows 7 or earlier:

Click **Start** (bottom left on your screen). Click **All Programs**. Click **Accessories**. Click **Notepad**.

To open Notepad in Windows 8 or later:

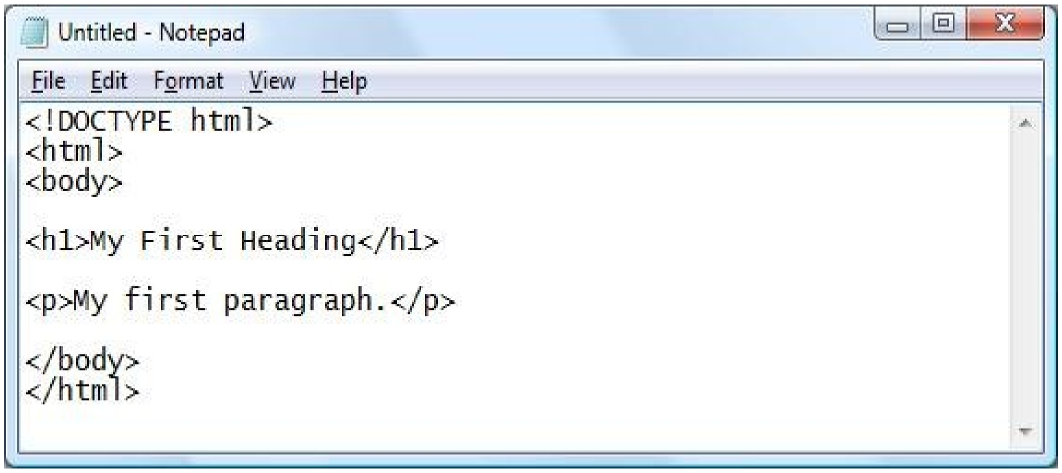
Open the **Start Screen** (the window symbol at the bottom left on your screen). Type **Notepad**.

Step 2: Write Some HTML

Write or copy some HTML into Notepad.

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Step 3: **Save the HTML Page**

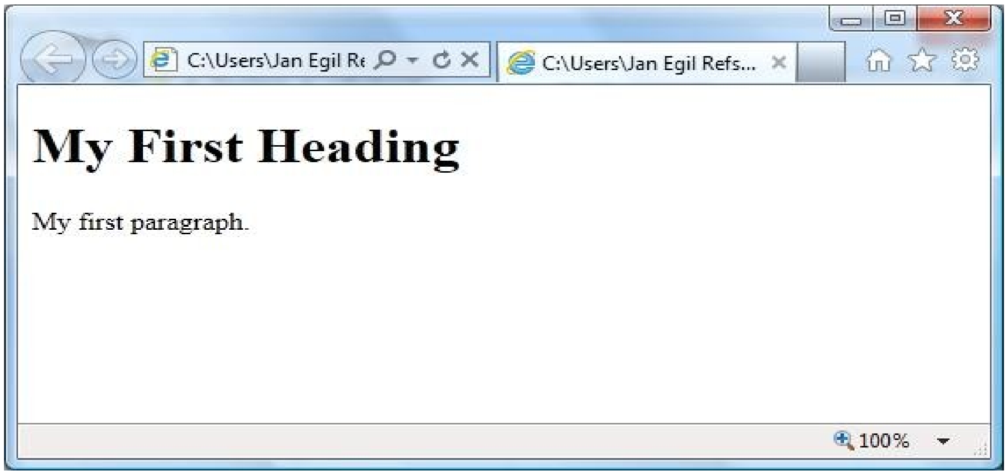
Save the file on your computer.

Select **File -> Save as** in the Notepad menu.

When saving an HTML file, use either the .htm or the .html file extension. There is no difference; it is entirely up to you.

Step 4: View HTML Page in Your Browser

Double-click your saved HTML file, and the result will look much like this:



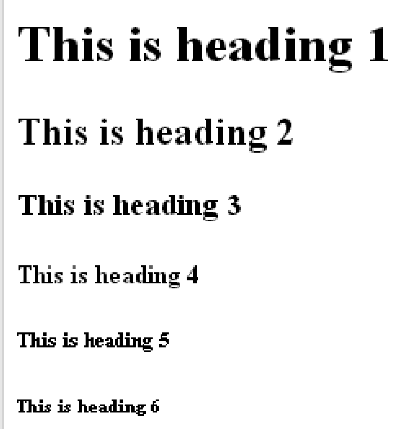
My First Heading

My first paragraph.

1.1.8. HTML Headings:

HTML headings are defined with the <h1> to <h6> tags.

```
<!DOCTYPE html>
<html>
<body>

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>

</body>
</html>
```



This is heading 1

This is heading 2

This is heading 3

This is heading 4

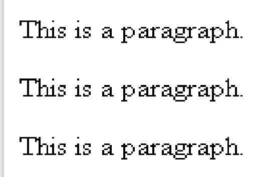
This is heading 5

This is heading 6

1.1.9. HTML Paragraphs:

HTML paragraphs are defined with the <p> tag.

```
<!DOCTYPE html>
<html>
<body>

<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>

</body>
</html>
```



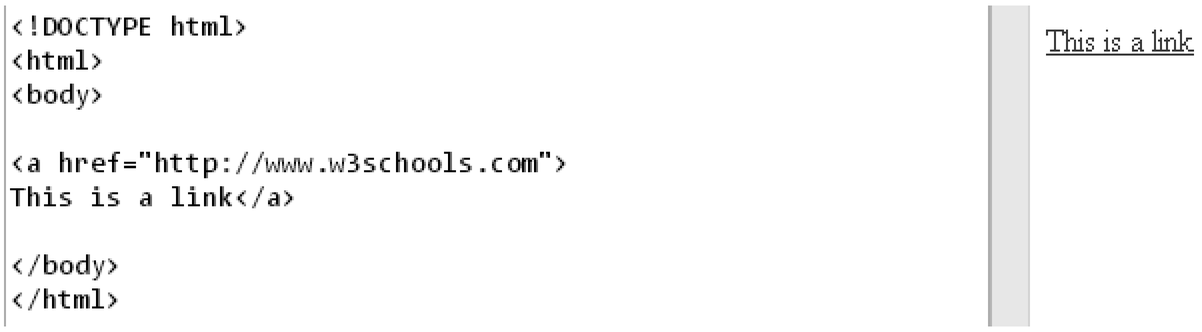
This is a paragraph.

This is a paragraph.

This is a paragraph.

1.1.10. HTML Links:

HTML links are defined with the <a> tag.

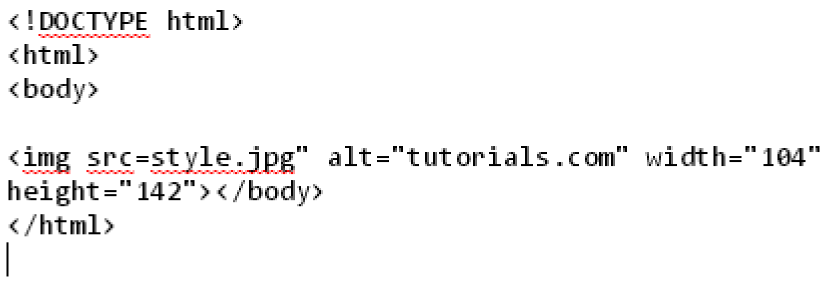


```
<!DOCTYPE html>
<html>
<body>

<a href="http://www.w3schools.com">
This is a link</a>

</body>
</html>
```

This is a link

1.1.11. HTML Images:

HTML images are defined with the <img> tag.



```
<!DOCTYPE html>
<html>
<body>

<img src=style.jpg" alt="tutorials.com" width="104"
height="142"></body>
</html>
```

1.2. HTML Elements:

The start tag is often called the **opening tag**. The end tag is often called the **closing tag**.

1.2.1. HTML Element Syntax:

- An HTML element starts with a **start tag / opening tag**
- An HTML element ends with an **end tag / closing tag**
- The **element content** is everything between the start and the end tag
- Some HTML elements have **empty content**
- Empty elements are **closed in the start tag**
- Most HTML elements can have **attributes**

1.2.2. Nested HTML Elements:

Most HTML elements can be nested (can contain other HTML elements).HTML documents consist of nested HTML elements.

1.2.3. HTML Document Example:

```
<!DOCTYPE html>
<html>
<body>
<p>This is my first paragraph.</p>
</body>
</html>
```

The example above contains 3 HTML elements.

1.2.4. HTML Example Explained:

The <p> element:

```
<p>This is my first paragraph.</p>
```

The <p> element defines a paragraph in the HTML document.
The element has a start tag <p> and an end tag </p>.
The element content is: This is my first paragraph.

The <body> element:

```
<body>
<p>This is my first paragraph.</p>
</body>
```

The <body> element defines the body of the HTML document.
The element has a start tag <body> and an end tag </body>.
The element content is another HTML element (a p element).

The <html> element:

```
<html>
<body>
<p>This is my first paragraph.</p>
</body>
</html>
```

The <html> element defines the whole HTML document.
The element has a start tag <html> and an end tag </html>.
The element content is another HTML element (the body element).

Don't Forget the End Tag:

Some HTML elements might display correctly even if you forget the end tag:

```
<p>This is a paragraph
<p>This is a paragraph
```

The example above works in most browsers, because the closing tag is considered optional.
Never rely on this. Many HTML elements will produce unexpected results and/or errors if you forget the end tag.

1.2.5. Empty HTML Elements:

HTML elements with no content are called empty elements.

<br> is an empty element without a closing tag (the <br> tag defines a line break).

Tip: In XHTML, all elements must be closed. Adding a slash inside the start tag, like <br/>, is the proper way of closing empty elements in XHTML (and XML).

1.3. HTML Attributes:

- HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes come in name/value pairs like: **name="value"**

1.3.1. Attribute Example:

HTML links are defined with the <a> tag. The link address is specified in the **href attribute**.

Example

```
<a href="http://www.gnit.ac.in">This is a link</a>
```

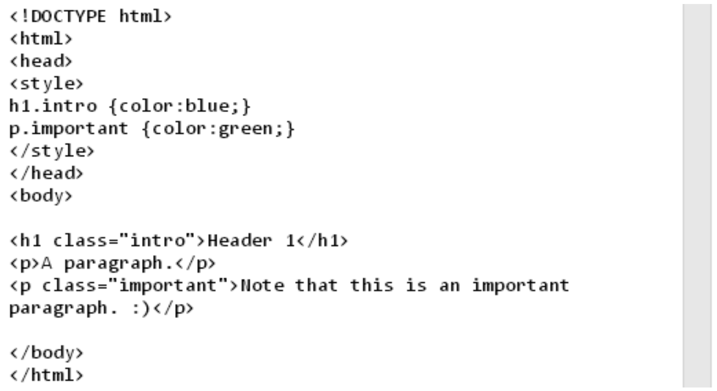
Always Quote Attribute Values
Attribute values should always be enclosed in quotes.
Double style quotes are the most common, but single style quotes are also allowed.

1.3.2. HTML Attributes Reference:

Below is a list of some attributes that can be used on any HTML element:

class	Specifies one or more classnames for an element (refers to a class in a style sheet)
id	Specifies a unique id for an element
style	Specifies an inline CSS style for an element
title	Specifies extra information about an element (displayed as a tool tip)

1.3.3. HTML class Attribute:



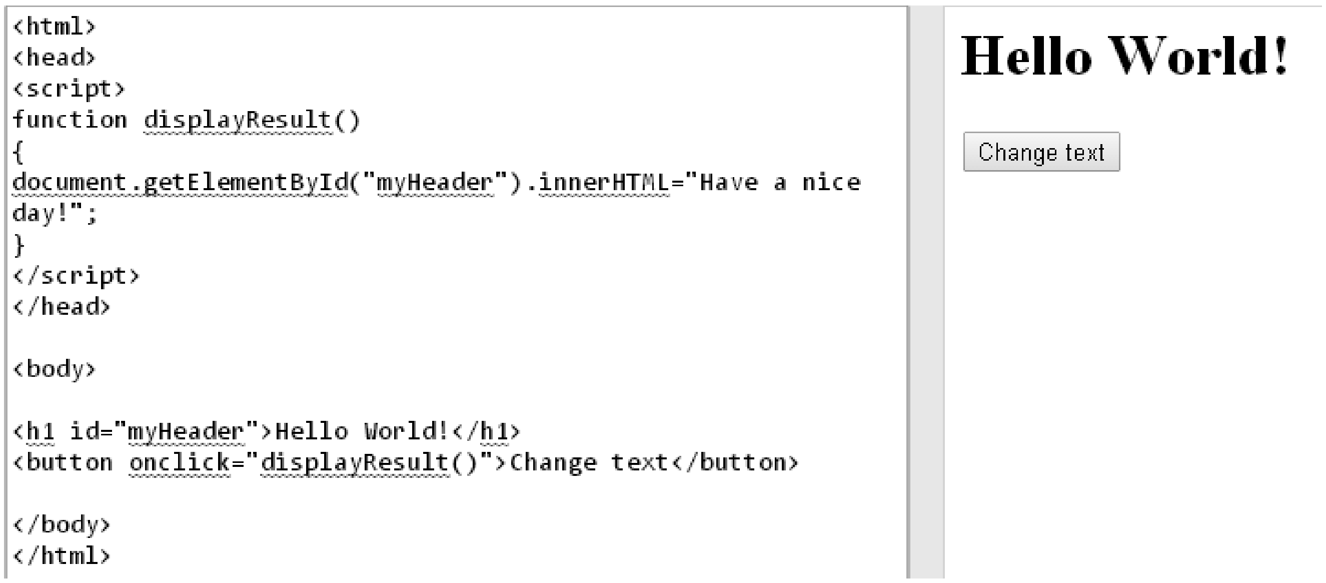
```
<!DOCTYPE html>
<html>
<head>
<style>
h1.intro {color:blue;}
p.important {color:green;}
</style>
</head>
<body>

<h1 class="intro">Header 1</h1>
<p>A paragraph.</p>
<p class="important">Note that this is an important
paragraph. :)</p>

</body>
</html>
```

Header 1

A paragraph.

Note that this is an important paragraph. :)

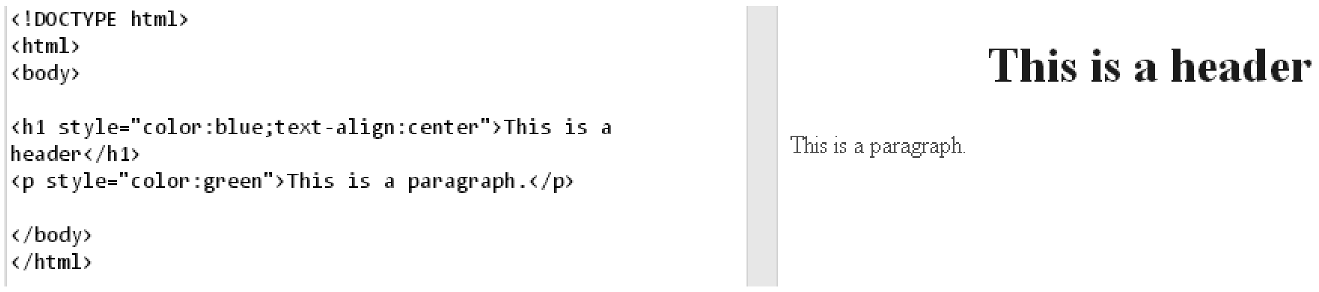
1.3.4. HTML id Attribute:



```
<html>
<head>
<script>
function displayResult()
{
document.getElementById("myHeader").innerHTML="Have a nice
day!";
}
</script>
</head>

<body>

<h1 id="myHeader">Hello World!</h1>
<button onclick="displayResult()">Change text</button>

</body>
</html>
```

Hello World!

1.3.5. HTML style Attribute:

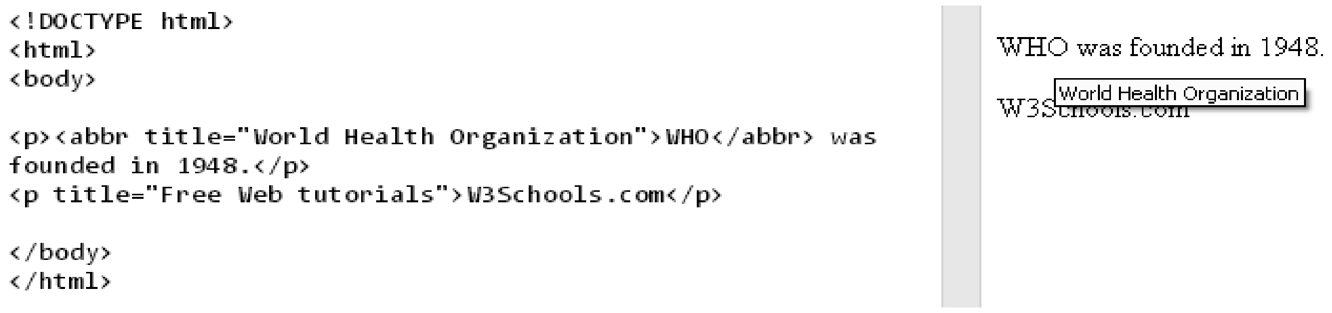


```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue;text-align:center">This is a
header</h1>
<p style="color:green">This is a paragraph.</p>

</body>
</html>
```

This is a header

This is a paragraph.

1.3.6. HTML title Attribute:



```
<!DOCTYPE html>
<html>
<body>

<p><abbr title="World Health Organization">WHO</abbr> was
founded in 1948.</p>
<p title="Free Web tutorials">W3Schools.com</p>

</body>
</html>
```

WHO was founded in 1948.

W3Schools.com

Tag	Description
<!--...-->	Defines a comment
<!DOCTYPE>	Defines the document type
<a>	Defines a hyperlink
<b>	Defines bold text
<blockquote>	Defines a section that is quoted from another source
<body>	Defines the document's body
<br>	Defines a single line break
<button>	Defines a clickable button
<center>	Not supported in HTML5. Use CSS instead. Defines centered text
<div>	Defines a section in a document

<font>	Not supported in HTML5. Use CSS instead. Defines font, color, and size for text
<footer>	Defines a footer for a document or section
<form>	Defines an HTML form for user input
<frame>	Not supported in HTML5. Defines a window (a frame) in a frameset
<h1> to <h6>	Defines HTML headings
<head>	Defines information about the document
<hr>	Defines a thematic change in the content
<html>	Defines the root of an HTML document
<i>	Defines a part of text in an alternate voice or mood
<link>	Defines the relationship between a document and an external resource (most used to link to style sheets)
<map>	Defines a client-side image-map
<meta>	Defines metadata about an HTML document
<object>	Defines an embedded object
<ol>	Defines an ordered list
<option>	Defines an option in a drop-down list
<p>	Defines a paragraph
<param>	Defines a parameter for an object
<script>	Defines a client-side script
<section>	Defines a section in a document
<style>	Defines style information for a document
<table>	Defines a table
<td>	Defines a cell in a table

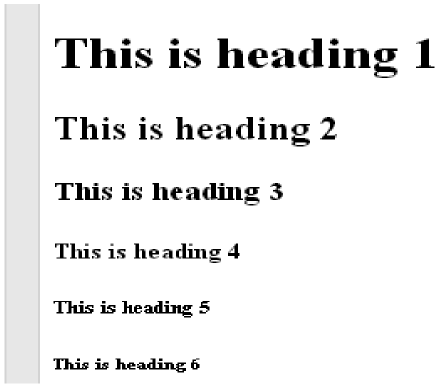
<textarea>	Defines a multiline input control (text area)
<th>	Defines a header cell in a table
<title>	Defines a title for the document
<tr>	Defines a row in a table

1.4. HTML Heading:

Headings are defined with the <h1> to <h6> tags.

<h1> defines the most important heading. <h6> defines the least important heading.

```
<!DOCTYPE html>
<html>
<body>

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>

</body>
</html>
```

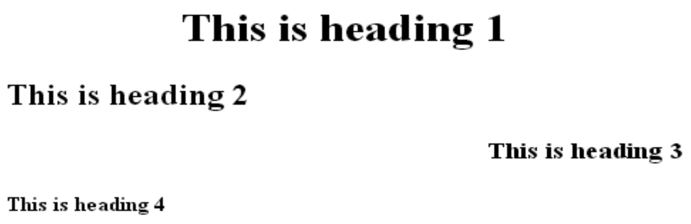


```
This is heading 1
This is heading 2
This is heading 3
This is heading 4
This is heading 5
This is heading 6
```

Note: Browsers automatically add some empty space (a margin) before and after each heading.

1.4.1. HTML Headings with align attributes:

```
<!DOCTYPE html>
<html>
<body>

<h1 align="center">This is heading 1</h1>
<h2 align="left">This is heading 2</h2>
<h3 align="right">This is heading 3</h3>
<h4 align="justify">This is heading 4</h4>

</body>
</html>
```



This is heading 1

This is heading 2

This is heading 3

This is heading 4

Headings Are Important

Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold**. Search engines use your headings to index the structure and content of your web pages. Since users may skim your pages by its headings, it is important to use headings to show the document structure.

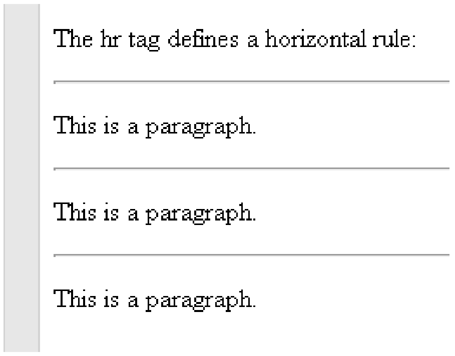
H1 headings should be used as main headings, followed by H2 headings, then the less important H3 headings, and so on.

1.4.2. HTML Lines:

The <hr> tag creates a horizontal line in an HTML page.

The hr element can be used to separate content:

```
<!DOCTYPE html>
<html>
<body>
<p>The hr tag defines a horizontal rule:</p>
<hr>
<p>This is a paragraph.</p>
<hr>
<p>This is a paragraph.</p>
<hr>
<p>This is a paragraph.</p>
</body>
</html>
```



The hr tag defines a horizontal rule:

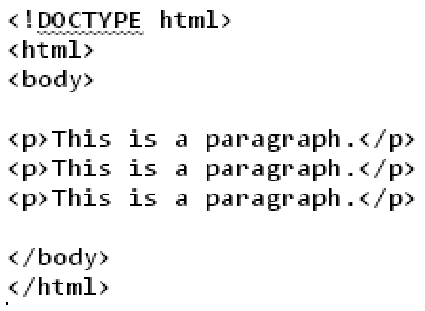
This is a paragraph.

This is a paragraph.

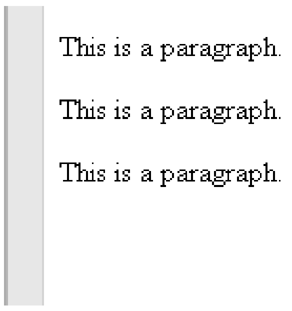
This is a paragraph.

1.5. HTML Paragraphs:

The <p> tag defines a paragraph. Browsers automatically add some space (margin) before and after each <p> element. The margins can be modified with CSS (with the margin properties).



```
<!DOCTYPE html>
<html>
<body>

<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>

</body>
</html>
```

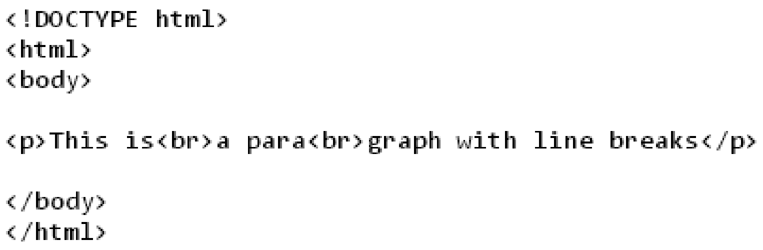


This is a paragraph.

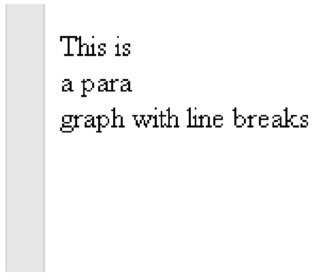
This is a paragraph.

This is a paragraph.

1.5.1. HTML line breaks:

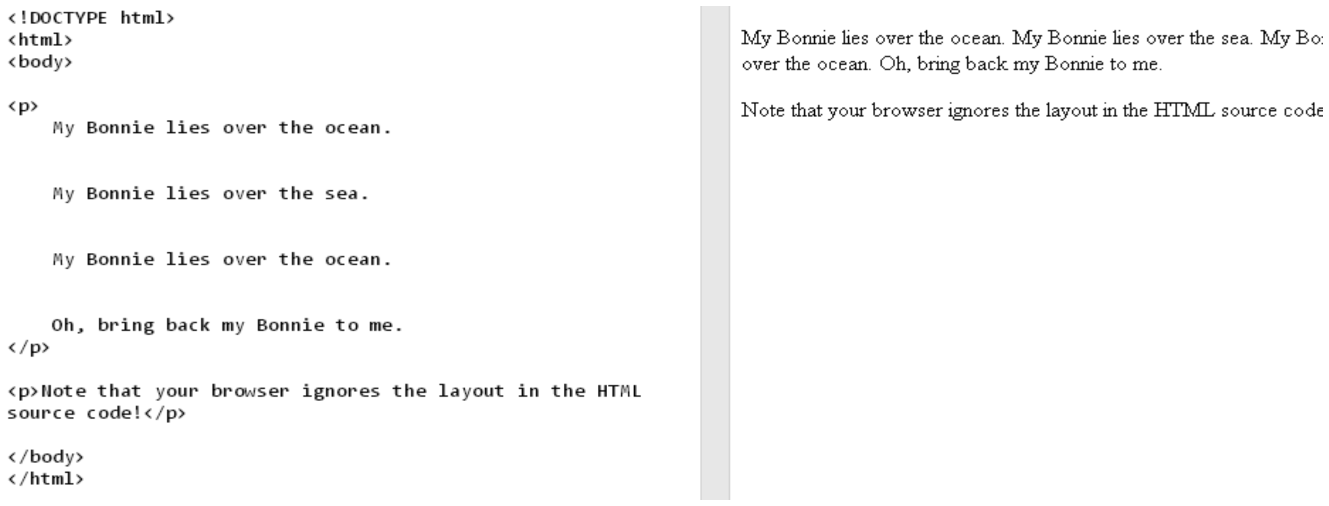


```
<!DOCTYPE html>
<html>
<body>

<p>This is<br>a para<br>graph with line breaks</p>

</body>
</html>
```



This is
a para
graph with line breaks

Another example

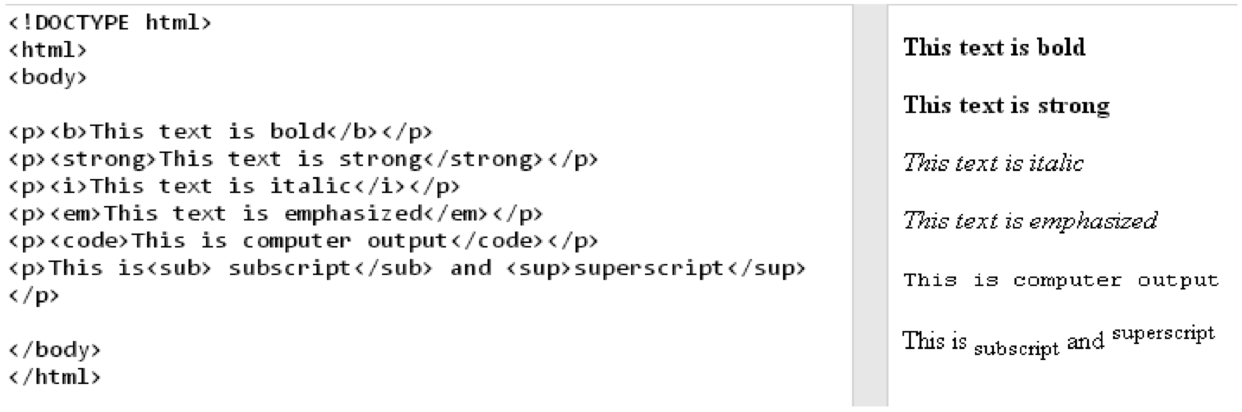


```
<!DOCTYPE html>
<html>
<body>

<p>
    My Bonnie lies over the ocean.


    My Bonnie lies over the sea.


    My Bonnie lies over the ocean.


    Oh, bring back my Bonnie to me.
</p>

<p>Note that your browser ignores the layout in the HTML
source code!</p>

</body>
</html>
```

```
My Bonnie lies over the ocean. My Bonnie lies over the sea. My Bo
over the ocean. Oh, bring back my Bonnie to me.

Note that your browser ignores the layout in the HTML source code
```

1.6. HTML Formatting:

1.6.1. Text Formatting:



```
<!DOCTYPE html>
<html>
<body>

<p><b>This text is bold</b></p>
<p><strong>This text is strong</strong></p>
<p><i>This text is italic</i></p>
<p><em>This text is emphasized</em></p>
<p><code>This is computer output</code></p>
<p>This is<sub> subscript</sub> and <sup>superscript</sup>
</p>

</body>
</html>
```

This text is bold

This text is strong

This text is italic

This text is emphasized

`This is computer output`

This is $_{\text{subscript}}$ and $^{\text{superscript}}$

1.6.2. HTML Formatting Tags:

HTML uses tags like <b> and <i> for formatting output, like bold or *italic* text.

These HTML tags are called formatting tags (look at the bottom of this page for a complete reference).

Often <strong> renders as <b>, and <em> renders as <i>.

However, there is a difference in the meaning of these tags:

<b> or <i> defines bold or italic text only.

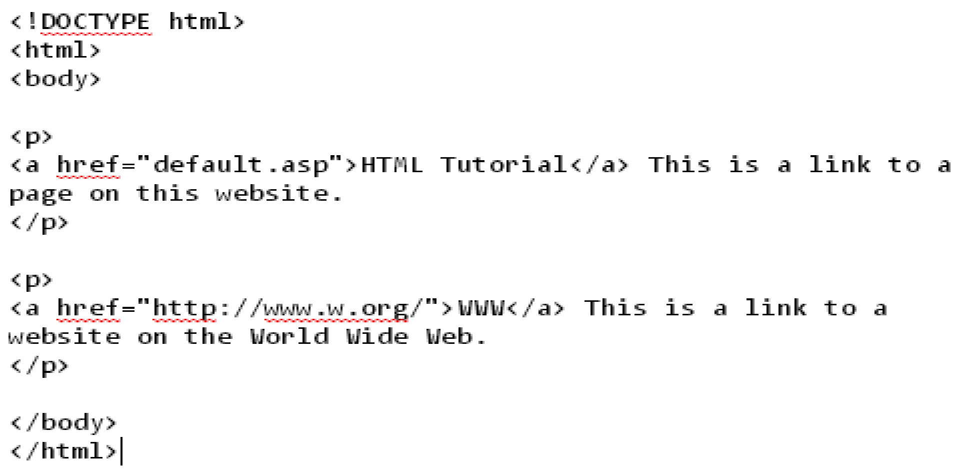
<strong> or <em> means that you want the text to be rendered in a way that the user understands

as "important". Today, all major browsers render strong as bold and em as italics. However, if a browser one day wants to make a text highlighted with the strong feature, it might be cursive for example and not bold!

1.6.3. HTML Text Formatting Tags:

Tag	Description
<b>	Defines bold text
<em>	Defines emphasized text
<i>	Defines a part of text in an alternate voice or mood
<small>	Defines smaller text
<strong>	Defines important text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text
<mark>	Defines marked/highlighted text

1.6.4. HTML Links:



```
<!DOCTYPE html>
<html>
<body>

<p>
<a href="default.asp">HTML Tutorial</a> This is a link to a
page on this website.
</p>

<p>
<a href="http://www.w.org/">WWW</a> This is a link to a
website on the World Wide Web.
</p>

</body>
</html>
```

HTML Tutorial This is a link to a page on this website.

WWW This is a link to a website on the World Wide Web.

1.6.4.1. HTML Link Syntax:

The HTML code for a link is simple. It looks like this:

```
<a href="url">Link text</a>
```

The href attribute specifies the destination of a link.

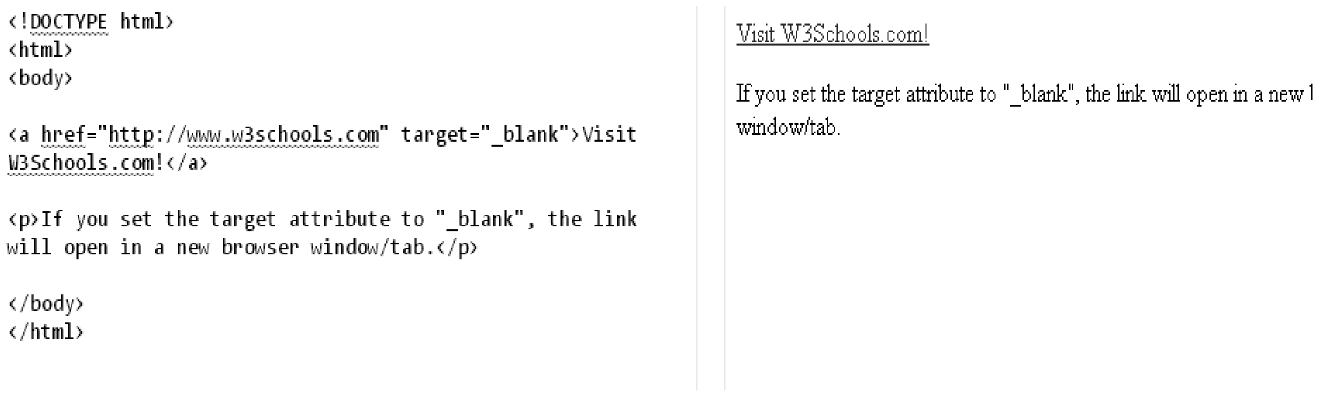
Example

<a href="http://www.yahoo.com/">Visit yahoo</a>

1.6.4.1.1. HTML Links - The target attribute:

The target attribute specifies where to open the linked document.

The example below will open the linked document in a new browser window or a new tab:



```
<!DOCTYPE html>
<html>
<body>

<a href="http://www.w3schools.com" target="_blank">Visit
W3Schools.com!</a>

<p>If you set the target attribute to "_blank", the link
will open in a new browser window/tab.</p>

</body>
</html>
```

```
Visit W3Schools.com!

If you set the target attribute to "_blank", the link will open in a new l
window/tab.
```

1.6.4.1.2. HTML Links - The id attribute:

The id attribute can be used to create a bookmark inside an HTML document.

Tip: Bookmarks are not displayed in any special way. They are invisible to the reader.

Example

An anchor with an id inside an HTML document:

<a id="tips">Useful Tips Section</a>

Create a link to the "Useful Tips Section" inside the same document:

<a href="#tips">Visit the Useful Tips Section</a>

Or, create a link to the "Useful Tips Section" from another page:

<a href="http://www.w3schools.com/html_links.htm#tips">

Visit the Useful Tips Section</a>

1.6.5. The HTML <head> Element:

The <head> element is a container for all the head elements. Elements inside <head> can include scripts, instruct the browser where to find style sheets, provide meta information, and more.

The following tags can be added to the head section: <title>, <style>, <meta>, <link>, <script>, <noscript>, and <base>.

1.6.6. The HTML <title> Element:

The <title> tag defines the title of the document.

The <title> element is required in all HTML/XHTML documents.

The <title> element:

- defines a title in the browser toolbar
- provides a title for the page when it is added to favorites
- displays a title for the page in search-engine results

A simplified HTML document:

```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
</head>
<body>
The content of the document......
</body>
</html>
```

1.6.7. The HTML <base> Element:

The <base> tag specifies the base URL/target for all relative URLs in a page:

```
<head>
<base href="http://www.gnit.com/images/" target="_blank">
</head>
```

1.6.8. The HTML <link> Element:

The <link> tag defines the relationship between a document and an external resource.

The <link> tag is most used to link to style sheets:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

1.6.9. The HTML <style> Element:

The <style> tag is used to define style information for an HTML document.

Inside the <style> element you specify how HTML elements should render in a browser:

```
<head>
<style type="text/css">
body {background-color:yellow;}
p {color:blue;}
</style>
</head>
```

1.6.10. The HTML <meta> Element:

Metadata is data (information) about data.

The <meta> tag provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parsable.

Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata.

The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services.

<meta> tags always go inside the <head> element.

<meta> Tags - Examples of Use

Define keywords for search engines:

```
<meta name="keywords" content="HTML, CSS, XML, XHTML, JavaScript">
```

Define a description of your web page:

```
<meta name="description" content="Free Web tutorials on HTML and CSS">
```

Define the author of a page:

```
<meta name="author" content="Hege Refsnes">
```

Refresh document every 30 seconds:

```
<meta http-equiv="refresh" content="30">
```

1.6.11. The HTML <script> Element:

The <script> tag is used to define a client-side script, such as a JavaScript.

The <script> element will be explained in a later chapter.

1.6.12. HTML head Elements:

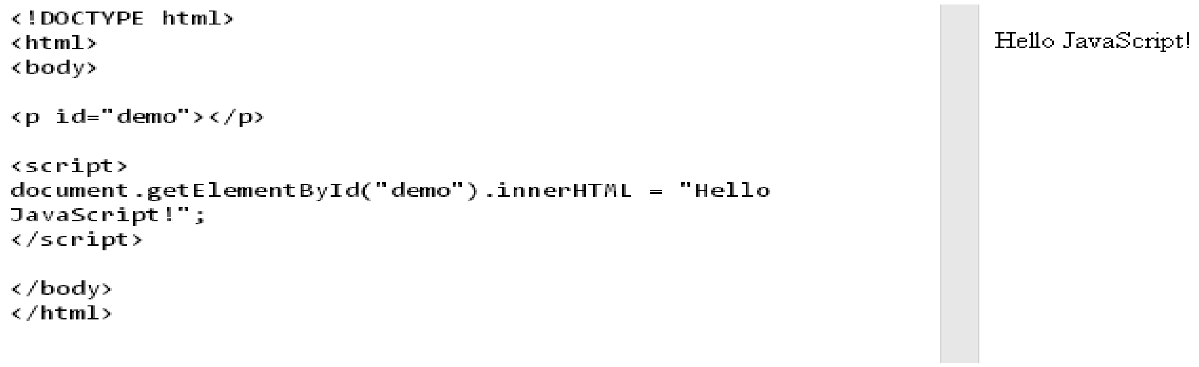
Tag	Description
<head>	Defines information about the document
<title>	Defines the title of a document
<base>	Defines a default address or a default target for all links on a page
<link>	Defines the relationship between a document and an external resource
<meta>	Defines metadata about an HTML document
<script>	Defines a client-side script
<style>	Defines style information for a document

1.6.12.1. <meta>:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="Hege Refsnes">
</head>
<body>
```

```
<p>All meta information goes in the head section...</p>
</body>
</html>
```

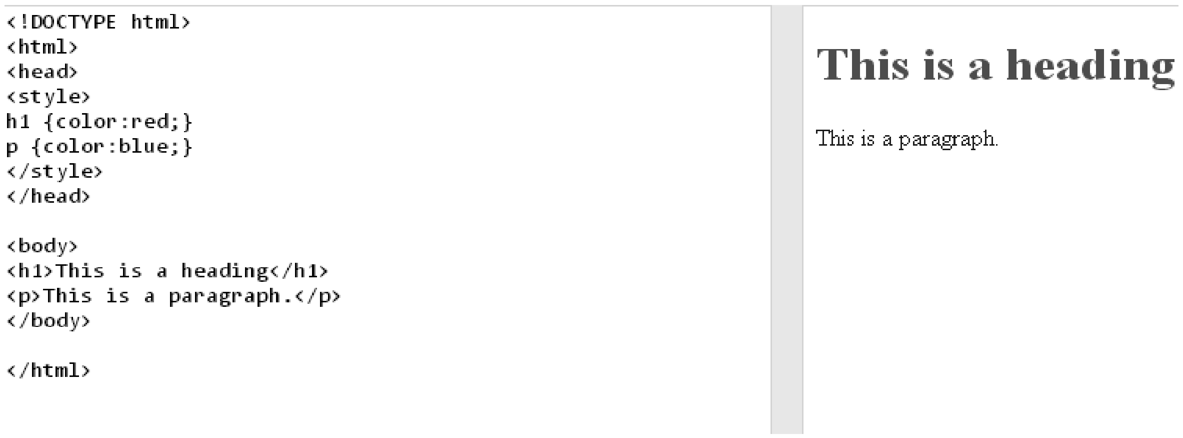
1.6.12.2. <script>:



```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello
JavaScript!";
</script>

</body>
</html>
```

Hello JavaScript!

1.6.12.3. <style>:



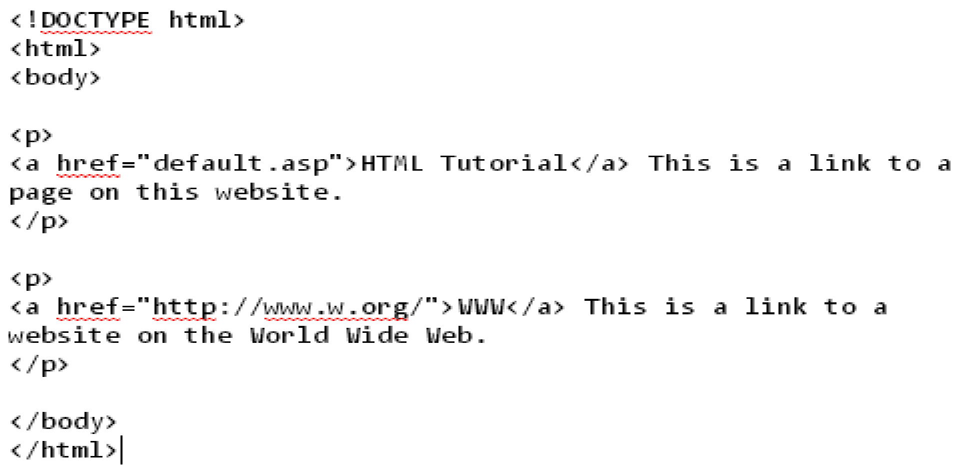
```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {color:red;}
p {color:blue;}
</style>
</head>

<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>

</html>
```

This is a heading

This is a paragraph.

Lecture 2: Link, Table, List, Block, Layout, Html Forms and input

2.1. HTML Links:



```
<!DOCTYPE html>
<html>
<body>

<p>
<a href="default.asp">HTML Tutorial</a> This is a link to a
page on this website.
</p>

<p>
<a href="http://www.w.org/">WWW</a> This is a link to a
website on the World Wide Web.
</p>

</body>
</html>
```

HTML Tutorial This is a link to a page on this website.

WWW This is a link to a website on the World Wide Web.

2.1.1. HTML Link Syntax:

The HTML code for a link is simple. It looks like this:

```
<a href="url">Link text</a>
```

The href attribute specifies the destination of a link.

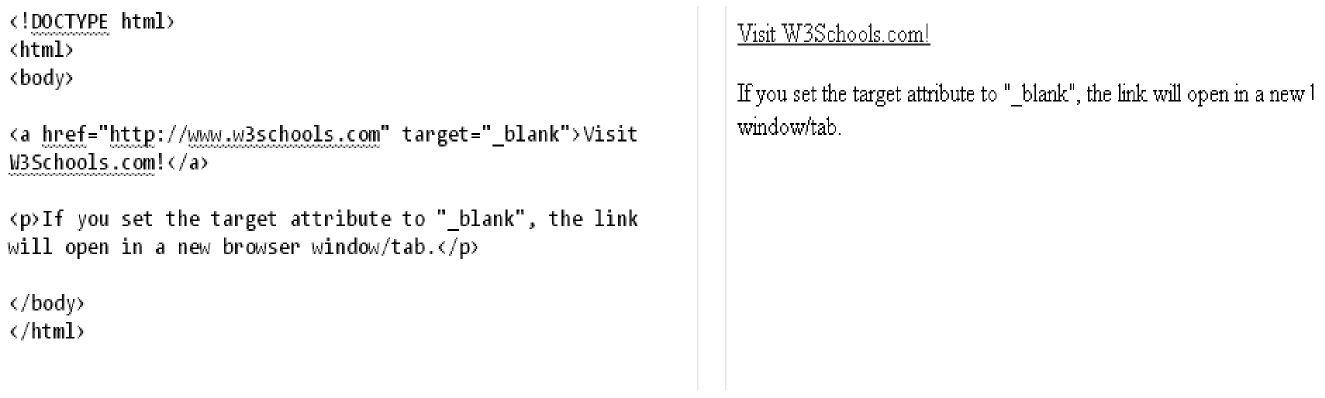
Example

<a href="http://www.yahoo.com/">Visit yahoo</a>

2.1.1.1. HTML Links - The target attribute:

The target attribute specifies where to open the linked document.

The example below will open the linked document in a new browser window or a new tab:



```
<!DOCTYPE html>
<html>
<body>

<a href="http://www.w3schools.com" target="_blank">Visit
W3Schools.com!</a>

<p>If you set the target attribute to "_blank", the link
will open in a new browser window/tab.</p>

</body>
</html>
```

Visit W3Schools.com!

If you set the target attribute to "_blank", the link will open in a new
window/tab.

2.1.1.2. HTML Links - The id attribute:

The id attribute can be used to create a bookmark inside an HTML document.

Tip: Bookmarks are not displayed in any special way. They are invisible to the reader.

Example

An anchor with an id inside an HTML document:

<a id="tips">Useful Tips Section</a>

Create a link to the "Useful Tips Section" inside the same document:

<a href="#tips">Visit the Useful Tips Section</a>

Or, create a link to the "Useful Tips Section" from another page:

<a href="http://www.w3schools.com/html_links.htm#tips">

Visit the Useful Tips Section</a>

2.2. HTML Table:

Tables are defined with the **<table>** tag.

A table is divided into rows with the **<tr>** tag. (tr stands for table row)

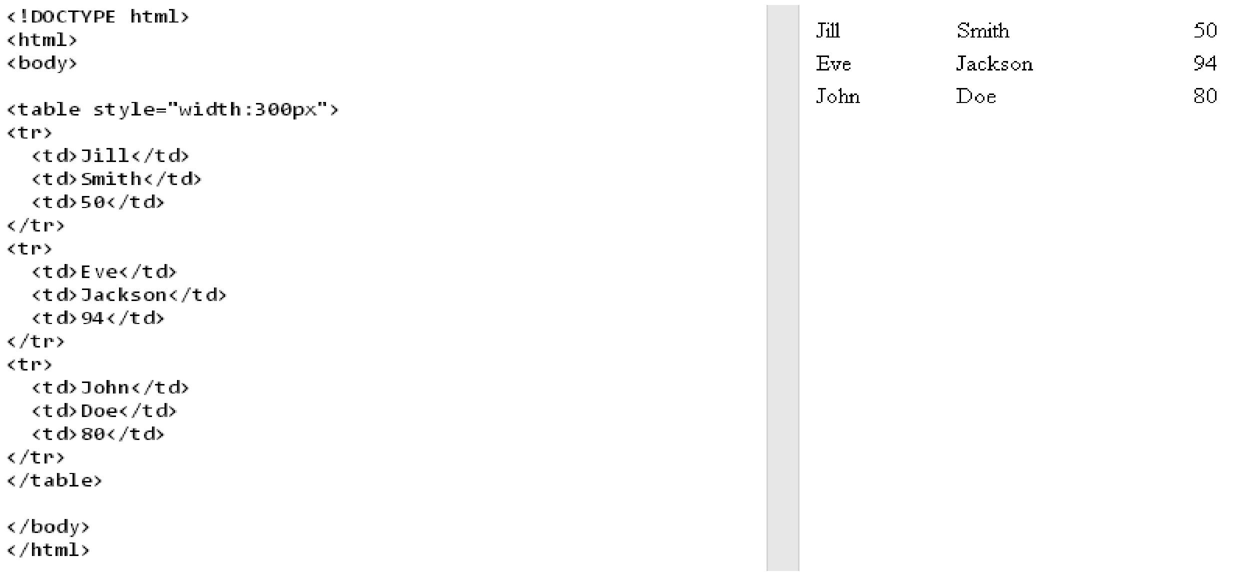
A row is divided into data cells with the **<td>** tag. (td stands for table data)

A row can also be divided into headings with the **<th>** tag. (th stands for table heading)

The <td> elements are the data containers in the table.

The <td> elements can contain all sorts of HTML elements like text, images, lists, other tables, etc.

The width of a table can be defined using CSS.



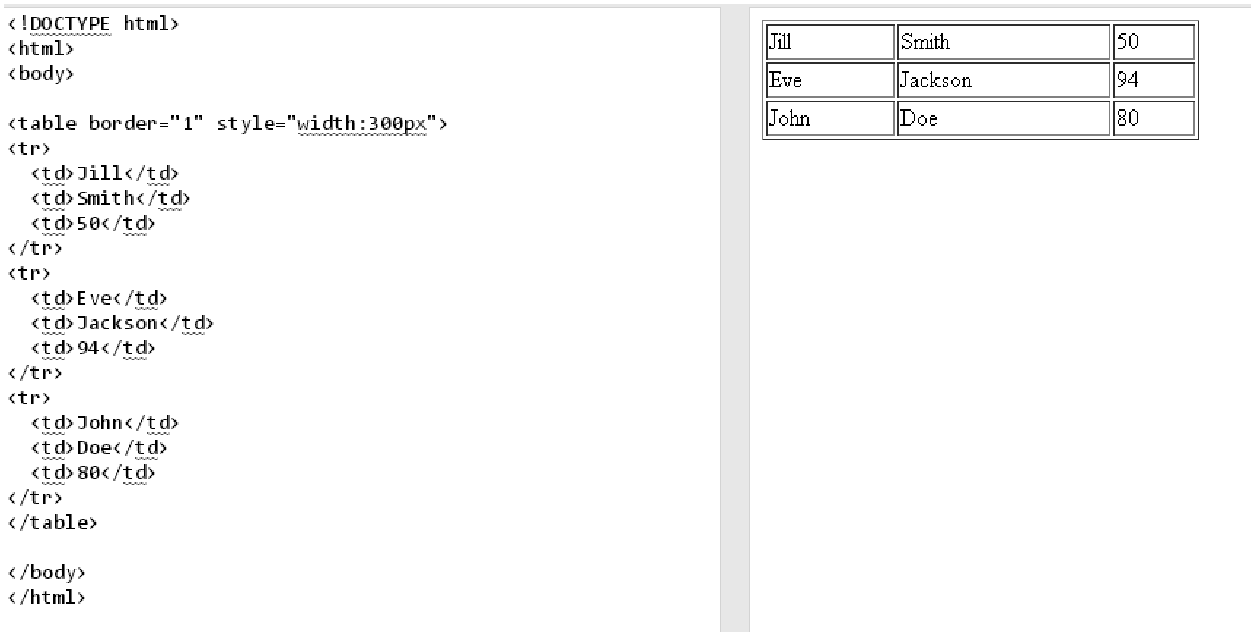
```
<!DOCTYPE html>
<html>
<body>

<table style="width:300px">
<tr>
  <td>Jill</td>
  <td>Smith</td>
  <td>50</td>
</tr>
<tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>94</td>
</tr>
<tr>
  <td>John</td>
  <td>Doe</td>
  <td>80</td>
</tr>
</table>

</body>
</html>
```

Jill	Smith	50
Eve	Jackson	94
John	Doe	80

2.2.1. An HTML Table with a Border Attribute:

If you do not specify a border for the table, it will be displayed without borders.

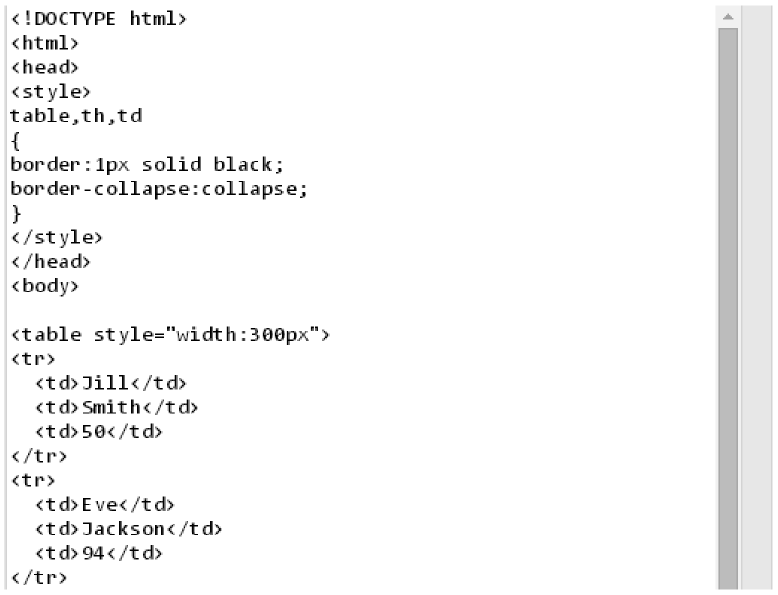
A border can be added using the border attribute.



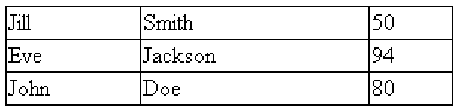
```
<!DOCTYPE html>
<html>
<body>

<table border="1" style="width:300px">
<tr>
  <td>Jill</td>
  <td>Smith</td>
  <td>50</td>
</tr>
<tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>94</td>
</tr>
<tr>
  <td>John</td>
  <td>Doe</td>
  <td>80</td>
</tr>
</table>

</body>
</html>
```

Jill	Smith	50
Eve	Jackson	94
John	Doe	80

2.2.2. An HTML Table with Collapsed Borders:



```
<!DOCTYPE html>
<html>
<head>
<style>
table,th,td
{
border:1px solid black;
border-collapse:collapse;
}
</style>
</head>
<body>

<table style="width:300px">
<tr>
  <td>Jill</td>
  <td>Smith</td>
  <td>50</td>
</tr>
<tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>94</td>
</tr>
```



Jill	Smith	50
Eve	Jackson	94
John	Doe	80

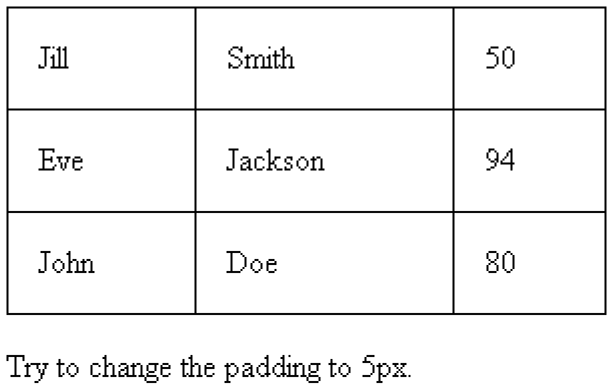
2.2.3. An HTML Table with Cell Padding:

Cell padding specifies the space between the cell content and its borders.

If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS padding property.

```
<html>
<head>
<style>
        table, th, td
        {
                border-collapse:collapse;
                border:1px solid black;
        }
        th, td
        {
                padding:15px;
        }
</style>
</head>
<body>
<table style="width:300px">
<tr>
        <td>Jill</td>
        <td>Smith</td>
        <td>50</td>
</tr>
<tr>
        <td>Eve</td>
        <td>Jackson</td>
        <td>94</td>
</tr>
<tr>
        <td>John</td>
        <td>Doe</td>
        <td>80</td>
</tr>
</table>
<p>Try to change the padding to 5px.</p>
</body>
</html>
```



Jill	Smith	50
Eve	Jackson	94
John	Doe	80

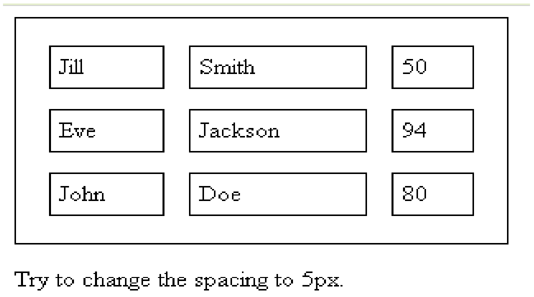
Try to change the padding to 5px.

2.2.4. An HTML Table with Cell Spacing:

Cell spacing specifies the space between the cells.

To set the cell spacing for the table, use the CSS border-spacing property.

```
<html>
<head>
<style>
        table, th, td
        {
                border:1px solid black;
                padding:5px;
        }
        table
        {
                border-spacing:15px;
        }
</style>
</head>
<body>

<table style="width:300px">
<tr>
        <td>Jill</td>
        <td>Smith</td>
        <td>50</td>
</tr>
<tr>
        <td>Eve</td>
        <td>Jackson</td>
        <td>94</td>
</tr>
<tr>
        <td>John</td>
        <td>Doe</td>
        <td>80</td>
</tr>
</table>

<p>Try to change the spacing to 5px.</p>

</body>
</html>
```



Jill	Smith	50
Eve	Jackson	94
John	Doe	80

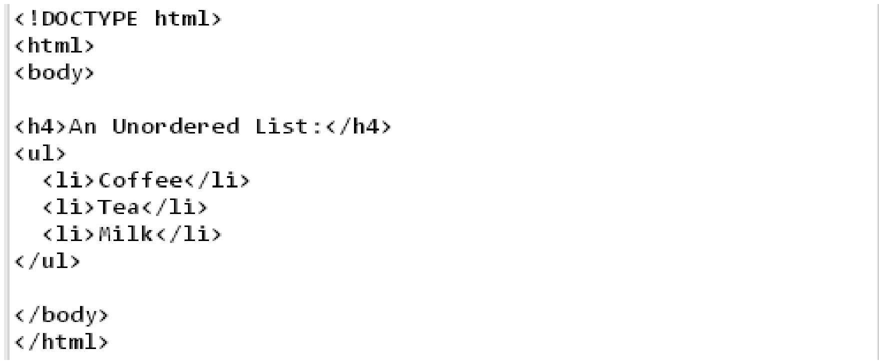
Try to change the spacing to 5px.

2.3. HTML Lists:

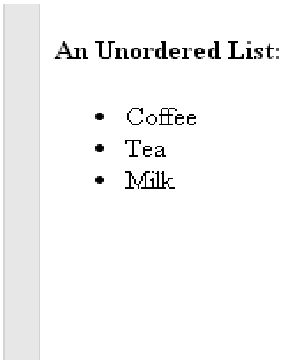
2.3.1. An unordered list:

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.The list items are marked with bullets (typically small black circles).

- List item
- List item
- List item



```
<!DOCTYPE html>
<html>
<body>

<h4>An Unordered List:</h4>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>

</body>
</html>
```

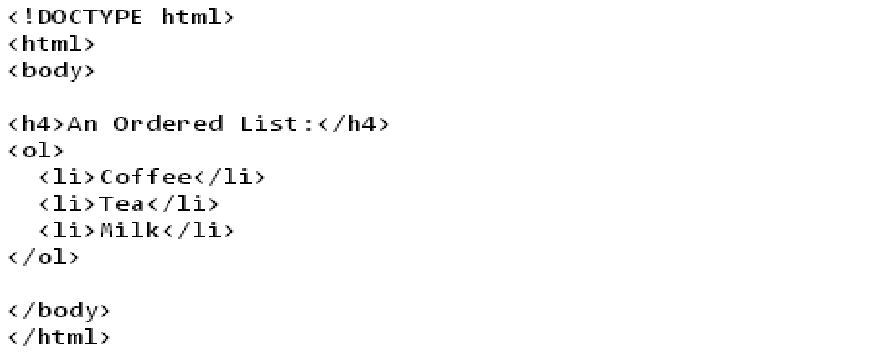


An Unordered List:

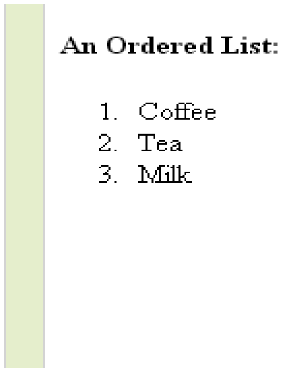
- Coffee
- Tea
- Milk

2.3.2. An ordered list:

1. The first list item
2. The second list item
3. The third list item



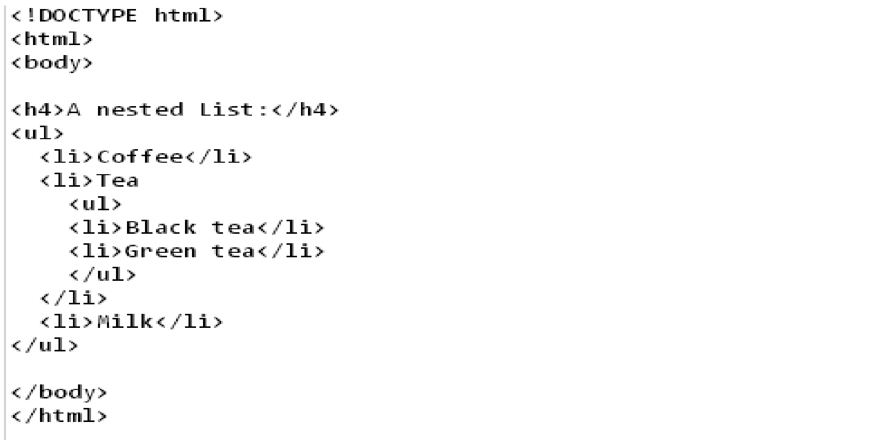
```
<!DOCTYPE html>
<html>
<body>

<h4>An Ordered List:</h4>
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>

</body>
</html>
```



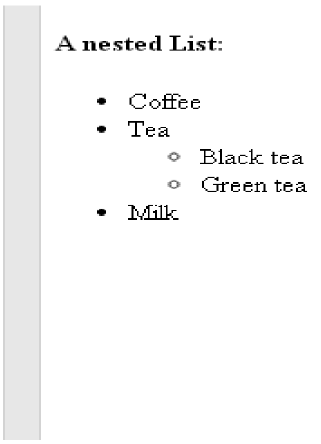
An Ordered List:

1. Coffee
2. Tea
3. Milk

2.3.3. A nested list:



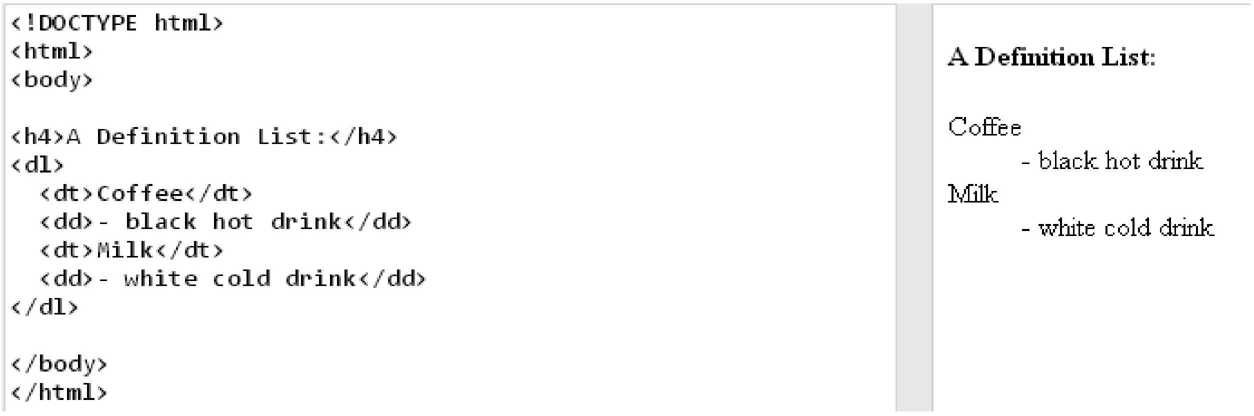
```
<!DOCTYPE html>
<html>
<body>

<h4>A nested List:</h4>
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
    <li>Black tea</li>
    <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>

</body>
</html>
```



A nested List:

- Coffee
- Tea
  - Black tea
  - Green tea
- Milk

2.3.4. A Description List:



```
<!DOCTYPE html>
<html>
<body>

<h4>A Definition List:</h4>
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>

</body>
</html>
```

```
A Definition List:

Coffee
        - black hot drink
Milk
        - white cold drink
```

2.4. HTML Blocks:

HTML elements can be grouped together with <div> and <span>.

2.4.1. HTML Block Elements:

Most HTML elements are defined as **block level** elements or as **inline** elements.

Block level elements normally start (and end) with a new line when displayed in a browser.
Examples: <h1>, <p>, <ul>, <table>

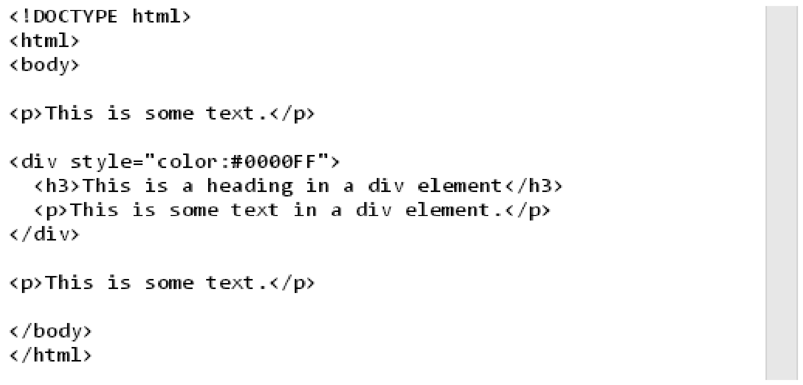
2.4.2. HTML Inline Elements:

Inline elements are normally displayed without starting a new line.

Examples: <b>, <td>, <a>, <img>

2.4.3. The HTML <div> Element:

The HTML <div> element is a block level element that can be used as a container for grouping other HTML elements. The<div> element has no special meaning. Except that, because it is a block level element, the browser will display a line break before and after it.When used together with CSS, the <div> element can be used to set style attributes to large blocks of content. Another common use of the <div> element, is for document layout. It replaces the "old way" of defining layout using tables. Using <table> elements for layout is not the correct use of <table>. The purpose of the <table> element is to display tabular data.



```
<!DOCTYPE html>
<html>
<body>

<p>This is some text.</p>

<div style="color:#0000FF">
  <h3>This is a heading in a div element</h3>
  <p>This is some text in a div element.</p>
</div>

<p>This is some text.</p>

</body>
</html>
```

This is some text.

This is a heading in a div element

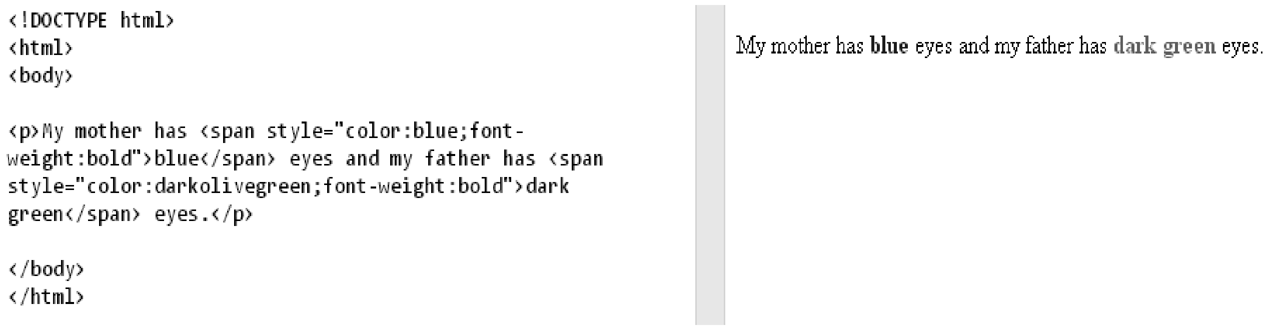
This is some text in a div element.

This is some text.

2.4.4. The HTML <span> Element:

The HTML <span> element is an inline element that can be used as a container for text. The <span> element has no special meaning.

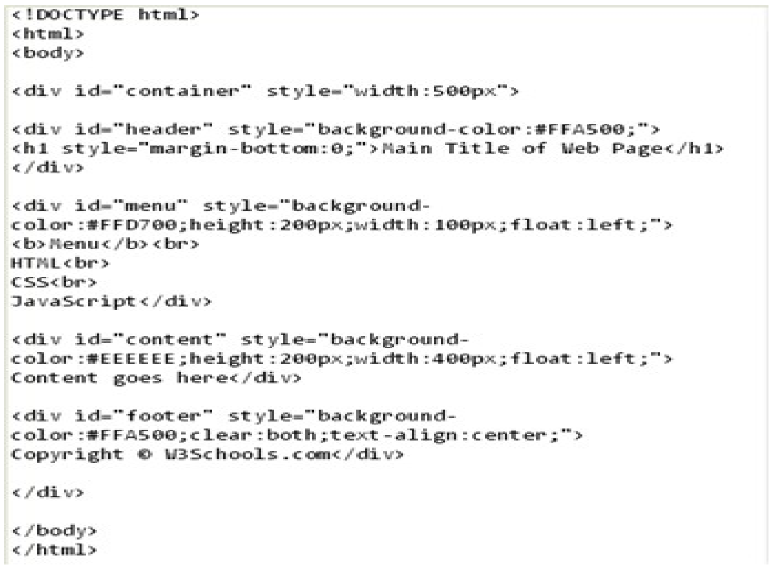
When used together with CSS, the <span> element can be used to set style attributes to parts of the text.



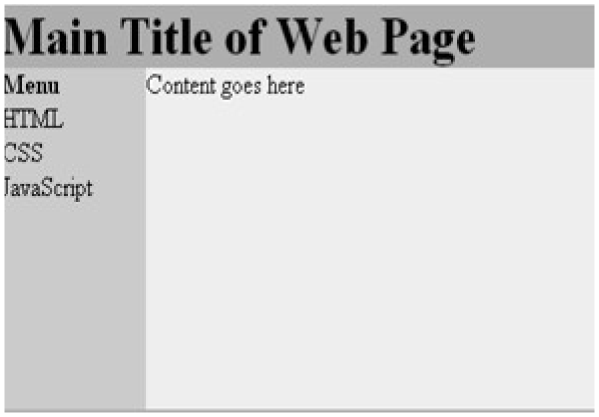
```
<!DOCTYPE html>
<html>
<body>

<p>My mother has <span style="color:blue;font-
weight:bold">blue</span> eyes and my father has <span
style="color:darkolivegreen;font-weight:bold">dark
green</span> eyes.</p>

</body>
</html>
```

My mother has **blue** eyes and my father has **dark green** eyes.

2.5. HTML Layouts:



```
<!DOCTYPE html>
<html>
<body>

<div id="container" style="width:500px">

<div id="header" style="background-color:#FFA500;">
<h1 style="margin-bottom:0;">Main Title of Web Page</h1>
</div>

<div id="menu" style="background-
color:#FFD700;height:200px;width:100px;float:left;">
<b>Menu</b><br>
HTML<br>
CSS<br>
JavaScript</div>

<div id="content" style="background-
color:#EEEEEE;height:200px;width:400px;float:left;">
Content goes here</div>

<div id="footer" style="background-
color:#FFA500;clear:both;text-align:center;">
Copyright © W3Schools.com</div>

</div>

</body>
</html>
```



2.6. HTML Formsand Inputs:

2.6.1. HTML Forms:

HTML forms are used to pass data to a server.An HTML form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select lists, textarea, fieldset, legend, and label elements.

The <form> tag is used to create an HTML form:

```
<form>
        .
        input elements
        .
</form>
```

2.6.2. HTML Forms - The Input Element:

The most important form element is the <input> element.The <input> element is used to select user information. An <input> element can vary in many ways, depending on the type attribute. An <input> element can be of type text field, checkbox, password, radio button, submit button, and more.The most common input types are described below.

2.6.2.1. Text Fields:

<input type="text"> defines a one-line input field that a user can enter text into:

```
<form>
        First name: <input type="text" name="firstname"><br>
        Last name: <input type="text" name="lastname">
</form>
```

How the HTML code above looks in a browser:

First name:

Last name:

Note: The form itself is not visible. Also note that the default width of a text field is 20 characters.

2.6.2.2. Password Field:

<input type="password"> defines a password field:

```
<form>
    Password: <input type="password" name="pwd">
</form>
```

How the HTML code above looks in a browser:

Password:

Note: The characters in a password field are masked (shown as asterisks or circles).

2.6.2.3. Radio Buttons:

<input type="radio"> defines a radio button. Radio buttons let a user select ONLY ONE of a limited number of choices:

```
<form>
    <input type="radio" name="sex" value="male">Male<br>
    <input type="radio" name="sex" value="female">Female
</form>
```

How the HTML code above looks in a browser:

○ Male

○ Female

2.6.2.4. Checkboxes:

<input type="checkbox"> defines a checkbox. Checkboxes let a user select ZERO or MORE options of a limited number of choices:

```
<form>
    <input type="checkbox" name="vehicle" value="Bike">I have a bike<br>
```

```
        <input type="checkbox" name="vehicle" value="Car">I have a car
</form>
```

How the HTML code above looks in a browser:

☐ I have a bike

☐ I have a car

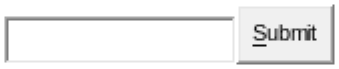
2.6.2.5. Submit Button:

<input type="submit"> defines a submit button:

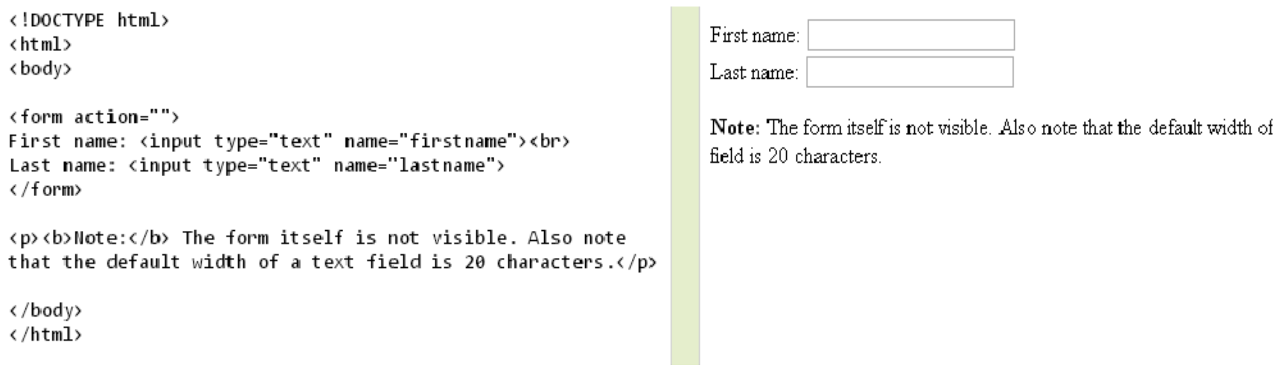
A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input.

```
<form name="input" action="demo_form_action.asp" method="get">
        Username: <input type="text" name="user">
        <input type="submit" value="Submit">
</form>
```

2.6.2.6. How the HTML code above looks in a browser:

Username: 

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "demo_form_action.asp". The page will show you the received input.



```
<!DOCTYPE html>
<html>
<body>

<form action="">
First name: <input type="text" name="firstname"><br>
Last name: <input type="text" name="lastname">
</form>

<p><b>Note:</b> The form itself is not visible. Also note
that the default width of a text field is 20 characters.</p>

</body>
</html>
```

First name:
Last name:

Note: The form itself is not visible. Also note that the default width of field is 20 characters.

Lecture 3, 4: Iframe, Colors, Image Maps and attributes of image area

3.1. HTML Iframes:

An iframe is used to display a web page within a web page.

Syntax for adding an iframe:

<iframe src="URL"></iframe>

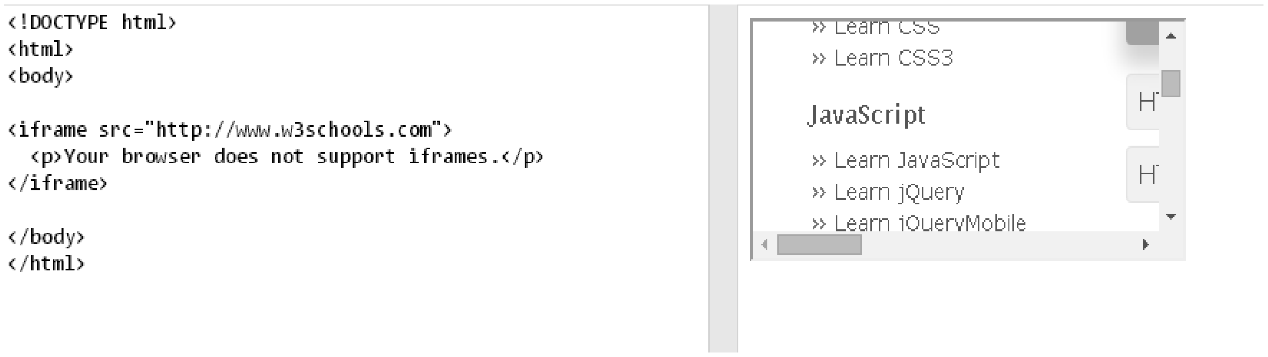
The URL points to the location of the separate page.

3.1.1. Iframe - Set Height and Width:

The height and width attributes are used to specify the height and width of the iframe.
The attribute values are specified in pixels by default, but they can also be in percent (like "80%").

Example

<iframe src="demo_iframe.htm" width="200" height="200"></iframe>



```
<!DOCTYPE html>
<html>
<body>

<iframe src="http://www.w3schools.com">
  <p>Your browser does not support iframes.</p>
</iframe>

</body>
</html>
```

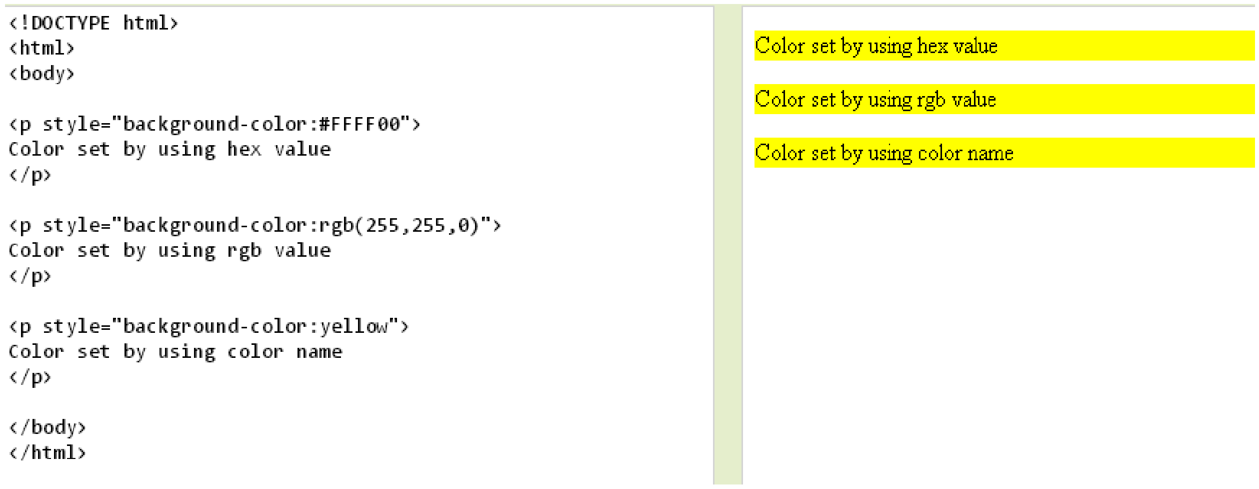
3.2. HTML Colors:

Colors are displayed combining RED, GREEN, and BLUE light.

3.2.1. Color Values:

CSS colors are defined using a hexadecimal (hex) notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00). The highest value is 255 (hex FF).Hex values are written as 3 double digit numbers, starting with a # sign.

	#000000	rgb(0,0,0)
	#FF0000	rgb(255,0,0)
	#00FF00	rgb(0,255,0)
	#0000FF	rgb(0,0,255)
	#FFFF00	rgb(255,255,0)
	#00FFFF	rgb(0,255,255)
	#FF00FF	rgb(255,0,255)
	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

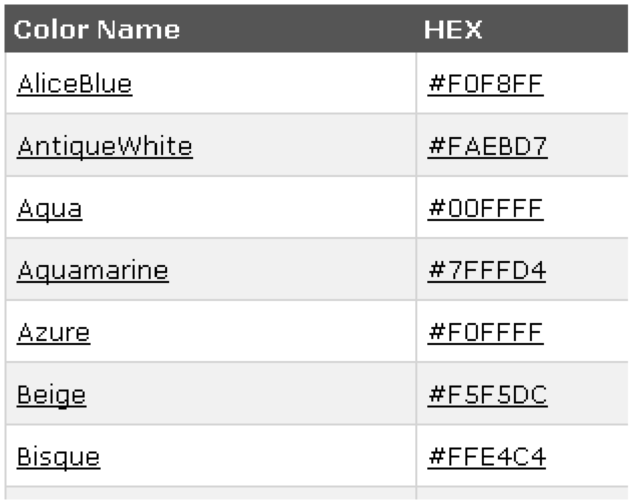


```
<!DOCTYPE html>
<html>
<body>

<p style="background-color:#FFFF00">
Color set by using hex value
</p>

<p style="background-color:rgb(255,255,0)">
Color set by using rgb value
</p>

<p style="background-color:yellow">
Color set by using color name
</p>

</body>
</html>
```

```
Color set by using hex value

Color set by using rgb value

Color set by using color name
```

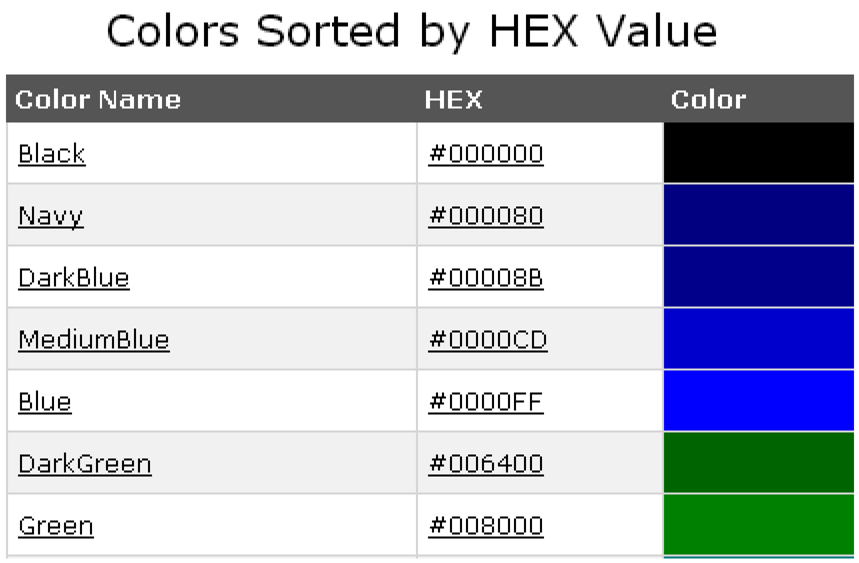
3.2.2. HTML Color Name:

Color Names Supported by All Browsers.140 color names are defined in the HTML and CSS color specification (17 standard colors plus 123 more). The table below lists them all, along with their hexadecimal values. Tip: The 17 standard colors are: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow.



Color Name	HEX
AliceBlue	#F0F8FF
AntiqueWhite	#FAEBD7
Aqua	#00FFFF
Aquamarine	#7FFFD4
Azure	#F0FFFF
Beige	#F5F5DC
Bisque	#FFE4C4

3.2.3. HTML Color Values:



Colors Sorted by HEX Value

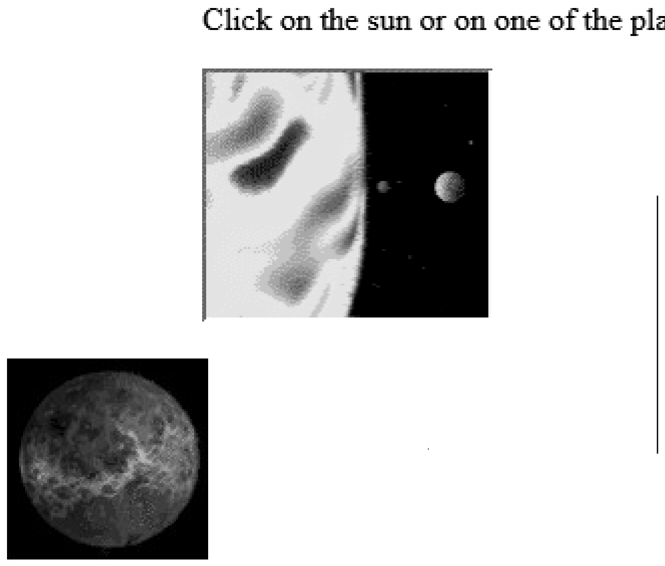
Color Name	HEX	Color
Black	#000000	⬛
Navy	#000080	🟦
DarkBlue	#00008B	🟦
MediumBlue	#0000CD	🟦
Blue	#0000FF	🟦
DarkGreen	#006400	🟩
Green	#008000	🟩

4. Image Map:

4.1. HTML <map> Tag:

The <map> tag is used to define a client-side image-map. An image-map is an image with clickable areas.The required name attribute of the <map> element is associated with the <img>'s usemap attribute and creates a relationship between the image and the map. The <map> element contains a number of <area> elements that defines the clickable areas in the image map.

```
<!DOCTYPE html>
<html>
<body>
<p>Click on the sun or on one of the planets to watch it closer:</p>
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap">
<map name="planetmap">
<area shape="rect" coords="0,0,82,126" alt="Sun" href="sun.htm">
<area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm">
<area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm">
</map>
</body>
</html>
```


Click on the sun or on one of the planets to watch it closer:

1. <map>:- name(attribute)
2. <img>:- usemap , src, alt(attribute)
3. <area>:- shape, cords, alt, href (attribute)

4.2. HTML <map> name Attribute:

The required name attribute specifies the name of an image-map.The name attribute is associated with the <img>'s usemap attribute and creates a relationship between the image and the map.

4.3. HTML <img> usemap Attribute:

The usemap attribute specifies an image as a client-side image-map (an image-map is an image with clickable areas).The usemap attribute is associated with a <map> element's name or id attribute, and creates a relationship between the <img> and the <map>.

Note: The usemap attribute cannot be used if the <img> element is a descendant of an <a> or <button> element.

4.4. HTML Images - The <img> Tag and the src Attribute:

In HTML, images are defined with the <img> tag.The <img> tag is empty, which means that it contains attributes only, and has no closing tag.To display an image on a page, you need to use the src attribute. src stands for "source". The value of the src attribute is the URL of the image you want to display.

4.4.1. Syntax for defining an image:

```
<img src="url" alt="some_text">
```

The URL points to the location where the image is stored. An image named "boat.gif", located in the "images" directory on "www.abc.com" has the URL: http://www.abc.com/images/boat.gif.

The browser displays the image where the <img> tag occurs in the document. If you put an image tag between two paragraphs, the browser shows the first paragraph, then the image, and then the second paragraph.

4.5. HTML Images - The Alt Attribute:

The required alt attribute specifies an alternate text for an image, if the image cannot be displayed.

The value of the alt attribute is an author-defined text:

```
<img src="smiley.gif" alt="Smiley face">
```

The alt attribute provides alternative information for an image if a user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

4.6. HTML <area> Tag:

The <area> tag defines an area inside an image-map (an image-map is an image with clickable areas).The <area> element is always nested inside a <map> tag.

Note: The usemap attribute in the <img> tag is associated with the <map> element's name attribute, and creates a relationship between the image and the map.

Attribute	Value	Description
alt	text	Specifies an alternate text for the area. Required if the href attribute is present
text	coordinates	Specifies the coordinates of the area
coords	URL	Specifies the hyperlink target for the area
coordinates	default rect circle poly	Specifies the shape of the area

4.6.1. <area coords="value">:

Value	Description
x1,y1,x2,y2	Specifies the coordinates of the left, top, right, bottom corner of the rectangle (for shape="rect")
x,y,radius	Specifies the coordinates of the circle center and the radius (for shape="circle")
x1,y1,x2,y2,..,xn,yn	Specifies the coordinates of the edges of the polygon. If the first and last coordinate pairs are not the same, the browser will add the last coordinate pair to close the polygon (for shape="poly")

4.6.2. <img usemap="#mapname">:

Value	Description
#mapname	A hash character ("#") plus the name or id of the <map> element to use

4.6.3. <img alt="text">:

Value	Description
text	Specifies an alternate text for an image. Guidelines for the alt text: • The text should describe the image if the image contains information • The text should explain where the link goes if the image is inside an <a> element • Use alt="" if the image is only for decoration

4.6.4. <area shape="default|rect|circle|poly">:

The shape attribute specifies the shape of an area.The shape attribute is used together with the coords attribute to specify the size, shape, and placement of an area.

Value	Description
default	Specifies the entire region
rect	Defines a rectangular region
circle	Defines a circular region
poly	Defines a polygonal region

Lecture 5: Introduction to CSS, basic syntax and structure of CSS, different types- internal, external and inline CSS

5.1. Cascading Style Sheet (CSS):

Cascading **S**tyle **S**heets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, and variations in display for different devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

5.2. Advantages of CSS:

- **CSS saves time** − You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
- **Pages load faster** − If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- **Easy maintenance** − To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- **Superior styles to HTML** − CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- **Multiple Device Compatibility** − Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- **Global web standards** − Now HTML attributes are being deprecated and it is being recommended to use CSS. So it’s a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

5.3. Internal Style Sheet:

An internal style sheet can be used if one single document has a unique style. Internal styles are defined in the <head> section of an HTML page, by using the <style> tag, like this:

```
<head>
<style>
body {background-color:yellow;}
p {color:blue;}
</style>
```

```
</head>
```

5.4. External Style Sheet:

An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the <head> section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

File name: - b.css

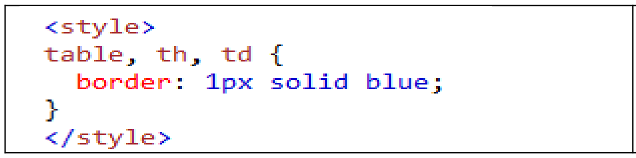
```
body {
   background-color: #d0e4fe;
}
h1 {
   color: orange;
   text-align: center;
}
p {
   font-family: "Times New Roman";
   font-size: 20px;
}
```

File name:- a.html

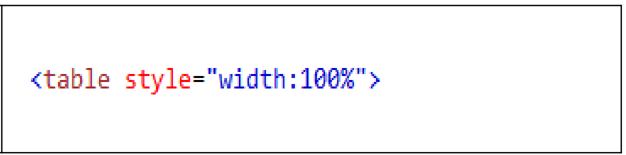
```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="b.css">

<body>
<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>
</body>

</html>
```

5.5. Syntax of Internal and External Style sheet:

Internal Style Sheet	Inline Style Sheet

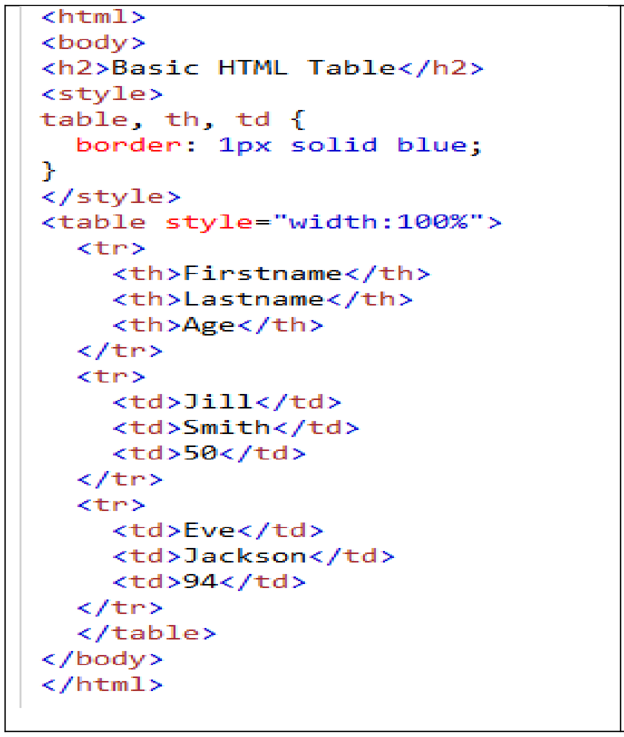


```
<style>
table, th, td {
  border: 1px solid blue;
}
</style>
```

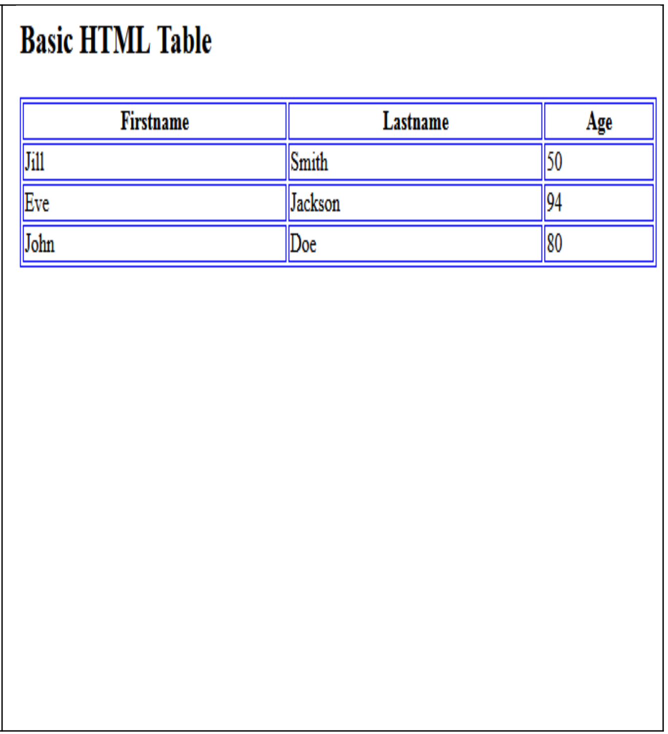


```
<table style="width:100%">
```

Internal Style Sheet + Inline Style Sheet



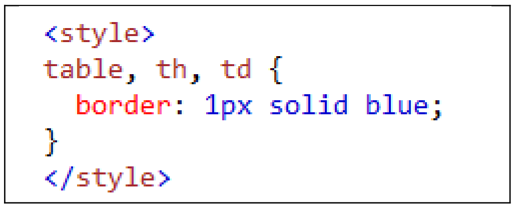
```
<html>
<body>
<h2>Basic HTML Table</h2>
<style>
table, th, td {
  border: 1px solid blue;
}
</style>
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  </table>
</body>
</html>
```



Basic HTML Table

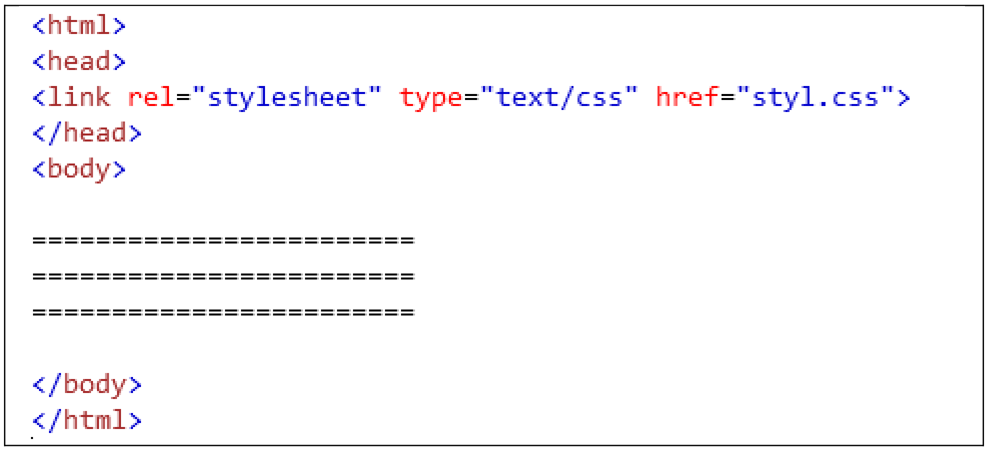
Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94
John	Doe	80

External Style Sheet



```
<style>
table, th, td {
  border: 1px solid blue;
}
</style>
```

External Style sheet File Name: Styl.css



```
<html>
<head>
<link rel="stylesheet" type="text/css" href="styl.css">
</head>
<body>

=========================
=========================
=========================

</body>
</html>
```

Html File Name: Example.html

Lecture 6: Basic Introduction of DHTML, Difference between HTML and DHTML, Documentary Object Model (DOM)

6.1. Dynamic HTML or DHTML:

Dynamic HTML or DHTML refers to a combination of regular HTML (Hypertext Markup Language) and various programming languages to increase a web page's interactivity. Good examples are drag and drop features and drop-down menus that appear when a user moves their mouse over a certain section of the page.

6.2. How it works:

DHTML is accomplished through the use of client-side scripting such as JavaScriptand CSS, as well as server-side languages such as Perl and PHP, in conjunction with static HTML. DHTML is beneficial in that it can be used to create web pages that react to user input without sending requests to a web server.

DHTML stands for Dynamic HTML, it is totally different from HTML. The browsers which support the dynamic HTML are some of the versions of Netscape Navigator and Internet Explorer of version higher than 4.0. The DHTML is based on the properties of the HTML, JavaScript, CSS, and DOM (Document Object Model which is used to access individual elements of a document) which helps in making dynamic content. It is the combination of HTML, CSS, JS, and DOM. The DHTML make use of Dynamic object model to make changes in settings and also in properties and methods. It also makes uses of Scripting and it is also part of earlier computing trends.

DHTML allows different scripting languages in a web page to change their variables, which enhance the effects, looks and many others functions after the whole page have been fully loaded or under a view process, or otherwise static HTML pages on the same. But in true ways, there is noting that as dynamic in DHTML, there is only the enclosing of different technologies like CSS, HTML, JS, DOM, and different sets of static languages which make it as dynamic.

DHTML is used to create interactive and animated web pages that are generated in real-time, also known as dynamic web pages so that when such a page is accessed, the code within the page is analyzed on the web server and the resulting HTML is sent to the client’s web browser.

HTML

HTML stands for Hypertext Markup Language and it is a client-side markup language. It is used to build the block of web pages.

JavaScript

It is a Client-side Scripting language. JavaScript is supported by most of the browser, also have cookies collection to determine the user needs.

CSS

The abbreviation of CSS is Cascading Style Sheet. It helps in the styling of the web pages and helps in designing of the pages. The CSS rules for DHTML will be modified at different levels using JS with event handlers which adds a significant amount of dynamism with very little code.

DOM

It is known as a dynamic Object Model which act as the weakest links in it. The only defect in it is that most of the browser does not support DOM. It is a way to manipulate the static contents.

Note: Many times DHTML is confused with being a language like HTML but it is not. It must be kept in mind that it is an interface or browsers enhancement feature which makes it possible to access the object model through JavaScript language and hence make the webpage more interactive.

6.3. Why Use DHTML?

DHTML makes a webpage dynamic but JavaScript also does, the question arises that what different does DHTML do? So the answer is that DHTML has the ability to change a webpages look, content and style once the document has loaded on our demand without changing or deleting everything already existing on the browser’s webpage. DHTML can change the content of a webpage on demand without the browser having to erase everything else, i.e. being able to alter changes on a webpage even after the document has completely loaded.

6.4. Advantages:

- Size of the files are compact in compared to other interactional media like Flash or Shockwave, and it downloads faster.
- It is supported by big browser manufacturers like Microsoft and Netscape.
- Highly flexible and easy to make changes.
- Viewer requires no extra plug-ins for browsing through the webpage that uses DHTML; they do not need any extra requirements or special software to view it.

- User time is saved by sending less number of requests to the server. As it is possible to modify and replace elements even after a page is loaded, it is not required to create separate pages for changing styles which in turn saves time in building pages and also reduces the number of requests that are sent to the server.
- It has more advanced functionality than a static HTML. It is capable of holding more content on the web page at the same time.

6.5. Disadvantages:

- It is not supported by all the browsers. It is supported only by recent browsers such as Netscape 6, IE 5.5, and Opera 5 like browsers.
- Learning of DHTML requires a lot of pre-requisites languages such as HTML, CSS, JS, etc should be known to the designer before starting with DHTML which is a long and time-consuming in itself.
- Implementation of different browsers are different. So if it worked in one browser, it might not necessarily work the same way in another browser.
- Even after being great with functionality, DHTML requires a few tools and utilities that are some expensive. For example, the DHTML text editor, Dreamweaver. Along with it the improvement cost of transferring from HTML to DHTML makes cost rise much higher.

6.6. Difference between HTML and DHTML:

- HTML is a markup language while DHTML is a collection of technologies.
- HTML is used to create static webpages while DHTML is capable of creating dynamic webpages.
- DHTML is used to create animations and dynamic menus but HTML not used.
- HTML sites are slow upon client-side technologies whereas DHTML sites are comparatively faster.
- Web pages created using HTML are rather simple and have no styling as it uses only one language whereas DHTML uses HTML, CSS, and JavaScript which results in a much better and way more presentable webpage.
- HTML cannot be used as server side code but DHTML used as server side code.
- DHTML needs database connectivity but not in case of HTML.
- Files in HTML are stored using .htm or .html extension while DHTML uses .dhtm extension.
- HTML requires no processing from the browser but DHTML does.

6.7. DOM (Documentary Object Model):

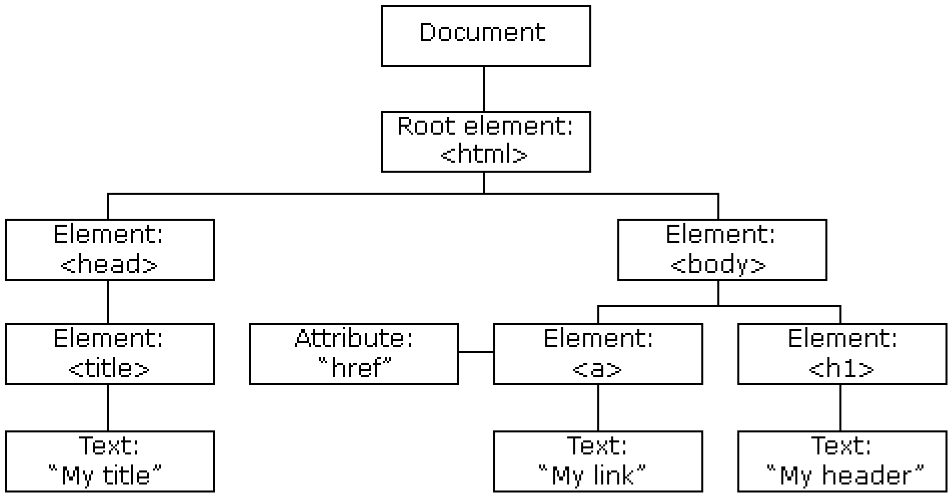
The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Lecture 7: Introduction to XML, Difference between HTML & XML, XML-Tree

7.1. What is XML?

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

7.2. The Difference between XML and HTML:

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is.
- HTML was designed to display data, with focus on how data looks.

HTML is about displaying information, while XML is about carrying information.

7.3. XML Does Not DO Anything:

Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information.

The following example is a note to Tove, from Jani, stored as XML:

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The note above is quite self-descriptive. It has sender and receiver information, it also has a heading and a message body.

But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

7.4. With XML You Invent Your Own Tags:

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

That is because the XML language has no predefined tags.

The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

7.5. XML is Not a Replacement for HTML:

XML is a complement to HTML.

It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.

My best description of XML is this:

XML is a software- and hardware-independent tool for carrying information.

7.6. USE OF XML:

XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.With XML, data can be stored in separate XML files. This way you can concentrate on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

XML Simplifies Data Sharing

In the real world, computer systems and databases contain data in incompatible formats.XML data is stored in plain text format. This provides a software and hardwareindependent way of storing data.This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Your Data More Available

Different applications can access your data, not only in HTML pages, but also from XML data sources.With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc.), and make it more available for blind people, or people with other disabilities.

7.7. XML is used to Create New Internet Languages:

A lot of new Internet languages are created with XML.
Here are some examples:

- XHTML
- WSDL for describing available web services
- WAP and WML as markup languages for handheld devices
- RSS languages for news feeds
- RDF and OWL for describing resources and ontology
- SMIL for describing multimedia for the web

7.8. Tree in XML:

XML documents form a tree structure that starts at "the root" and branches to "the leaves".
XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0).The next line describes the **root element** of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 **child elements** of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element :</note>

You can assume, from this example, that the XML document contains a note to Tove from Jani.Don't you agree that XML is pretty self-descriptive?

7.8.1. XML Documents Form a Tree Structure:

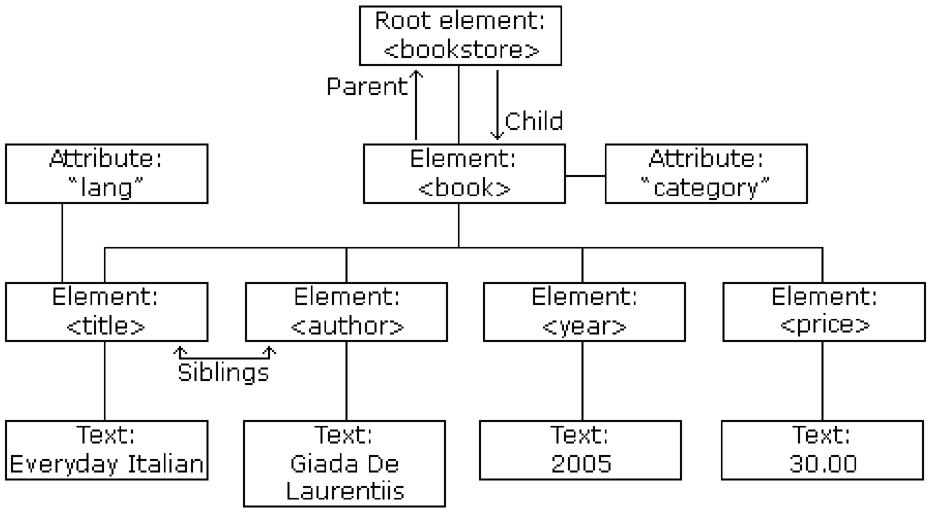
XML documents must contain a **root element**. This element is "the parent" of all other elements.The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements):

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).All elements can have text content and attributes (just like in HTML).

Example



The image above represents one book in the XML below:

```
<bookstore>
<book category="COOKING">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="CHILDREN">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="WEB">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

The root element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>.The <book> element has 4 children: <title>, < author>, <year>, <price>.

Lecture 8, 9: XML Syntax, Elements, Attributes, Validation and parsing, DTD

8.1. XML Syntax:

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

8.1.1. All XML Elements Must Have a Closing Tag:

In HTML, some elements do not have to have a closing tag:

<p>This is a paragraph.
<br>
In XML, it is illegal to omit the closing tag. All elements must have a closing tag:
<p>This is a paragraph.</p>
<br />

Note: You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

8.1.2. XML Tags are Case Sensitive:

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
Opening and closing tags must be written with the same case:

<Message>This is incorrect</message>
<message>This is correct</message>

Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

8.1.3. XML Elements Must be Properly Nested:

In HTML, you might see improperly nested elements:

<b><i>This text is bold and italic</b></i>

In XML, all elements **must** be properly nested within each other:

<b><i>This text is bold and italic</i></b>

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

8.1.4. XML Documents Must Have a Root Element:

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root**element.

```
<root>
<child>
<subchild>.....</subchild>
</child>
</root>
```

8.1.5. XML Attribute Values Must be quoted:

XML elements can have attributes in name/value pairs just like in HTML.In XML, the attribute values must always be quoted.Study the two XML documents below. The first one is incorrect, the second is correct:

```
<note date=12/11/2007>
<to>Tove</to>
<from>Jani</from>
</note>

<note date="12/11/2007">
<to>Tove</to>
<from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

8.1.6. Entity References:

Some characters have a special meaning in XML.If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 predefined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

8.1.7. Comments in XML:

The syntax for writing comments in XML is similar to that of HTML.

<!-- This is a comment -->

8.1.8. White-space is preserved in XML:

HTML truncates multiple white-space characters to one single white-space:

HTML:	Hello Tove
Output:	Hello Tove

With XML, the white-space in a document is not truncated.

8.2. XML Elements:

8.2.1. What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

- other elements
- text
- attributes
- or a mix of all of the above...

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <book> also has an **attribute** (category="CHILDREN"). <title>, <author>, <year>, and <price> have **text content** because they contain text.

```
<bookstore>
<book category="CHILDREN">
<title>Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="WEB">
<title>Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

8.2.2. Empty XML Elements:

An alternative syntax can be used for XML elements with no content:
Instead of writing a book element (with no content) like this:

```
<book></book>
```

It can be written like this:

```
<book />
```

This sort of element syntax is called self-closing.

8.2.3. XML Naming Rules:

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

Any name can be used, no words are reserved.

8.2.4. Best Naming Practices:

Make names descriptive: <first_name>, <last_name>.

Make names short and simple, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid "-". If you name something "first-name," some software may think you want to subtract name from first.

Avoid ".". If you name something "first.name," some software may think that "name" is a property of the object "first."

Avoid ":". Colons are reserved to be used for something called namespaces (more later).

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them.

8.2.5. Naming Styles:

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

8.2.6. XML Elements are Extensible:

XML elements can be extended to carry more information.

Look at the following XML example:

```
<note>
<to>Tove</to>
<from>Jani</from>
<body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

MESSAGE

To: Tove
From: Jani

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
<date>2008-01-10</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output.One of the beauties of XML, is that it can be extended without breaking applications.

8.3. XML Attributes:

XML elements can have attributes, just like HTML.Attributes provide additional information about an element.

In HTML, attributes provide additional information about elements:

```
<img src="computer.gif">
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

8.3.1. XML Attributes Must be quoted:

Attribute values must always be quoted. Either single or double quotes can be used. For a person's sex, the person element can be written like this:

```
<person sex="female">
```

or like this:

```
<person sex=’female’>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

8.3.2. XML Elements vs. Attributes:

Take a look at these examples:

```
<person sex="female">
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>

<person>
<sex>female</sex>
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

In the first example sex is an attribute. In the last, sex is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead.

8.3.3. My Favorite Way:

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="10/01/2008">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!</body>
</note>
```

A date element is used in the second example:

```
<note>
<date>10/01/2008</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

An expanded date element is used in the third: (THIS IS MY FAVORITE):

```
<note>
<date>
<day>10</day>
<month>01</month>
<year>2008</year>
</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

8.3.4. Avoid XML Attributes?

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

Don't end up like this:

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
```

```
</note>
```

8.3.5. XML Attributes for Metadata:

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

The id attributes above are for identifying the different notes. It is not a part of the note itself.

What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

```
<messages>
<note id="501">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
<note id="502">
<to>Jani</to>
<from>Tove</from>
<heading>Re: Reminder</heading>
<body>I will not</body>
</note>
</messages>
```

9.1. XML Validator:

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are –

- Well-formed XML document
- Valid XML document

9.1.1. Well-formed XML Document:

An XML document is said to be **well-formed** if it adheres to the following rules –

- Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self-ending tag. (<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**entities other than these must be declared.

Example

Following is an example of a well-formed XML document –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address
[
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
]>

<address>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</address>
```

The above example is said to be well-formed as −

- It defines the type of document. Here, the document type is **element** type.
- It includes a root element named as **address**.
- Each of the child elements among name, company and phone is enclosed in its self-explanatory tag.
- Order of the tags is maintained.

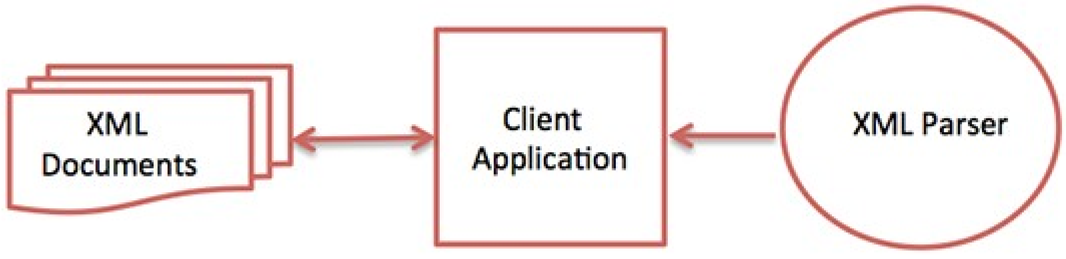
9.1.2. Valid XML Document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document.

9.2. XML – Parsers:

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

Following diagram shows how XML parser interacts with XML document −



The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below –

- **MSXML (Microsoft Core XML Services)** − This is a standard set of XML tools from Microsoft that includes a parser.
- **System.Xml.XmlDocument** − This class is part of .NET library, which contains a number of different classes related to working with XML.
- **Java built-in parser** − The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
- **Saxon** − Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces** − Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.

9.3. XML - DTDs:

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows –

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  ........
]>
```

In the above syntax,

- The DTD starts with <!DOCTYPE delimiter.
- An element tells the parser to parse the document from the specified root element.
- DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called External Subset.
- The square brackets [ ] enclose an optional list of entity declarations called Internal Subset.

9.3.1. Internal DTD:

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to yes. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD –

```
<!DOCTYPE root-element [element-declarations]>
```

Where root-element is the name of root element and element-declarations is where you declare the elements.

Example

Following is a simple example of internal DTD –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
]>
```

```
<address>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code –

Start Declaration − Begin the XML declaration with the following statement.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
```

DTD − Immediately after the XML header, the document type declaration follows, commonly referred to as the DOCTYPE −

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element name to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

9.3.1.1. Rules:

- The document type declaration must appear at the start of the document (preceded only by the XML header) − it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

9.3.2. External DTD:

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD –

```
<!DOCTYPE root-element SYSTEM "file-name">
```

Where file-name is the file with .dtd extension.

Example

The following example shows external DTD usage –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file address.dtd is as shown –

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

9.3.2.1. Types:

You can refer to an external DTD by using either system identifiers or public identifiers.

9.3.2.1.1. System Identifiers:

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

9.3.2.1.2. Public Identifiers:

Public identifiers provide a mechanism to locate DTD resources and is written as follows –

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called Formal Public Identifiers, or FPIs.

MODULE 3: JavaScript, CGI Script &PHP Script

Lecture 1: Basic Introduction of JavaScript, Statements, Comments, Variable, Operators, and Data Types

1. Basic Introduction:

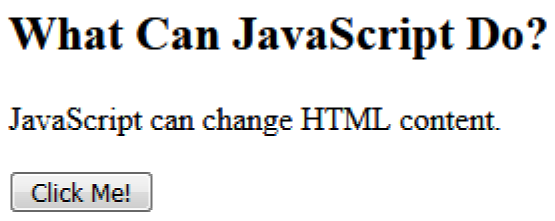
JavaScript(JS) is a scripting language, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code.

1.1. JavaScript Can Change HTML Content:

One of many JavaScript HTML methods is getElementById().

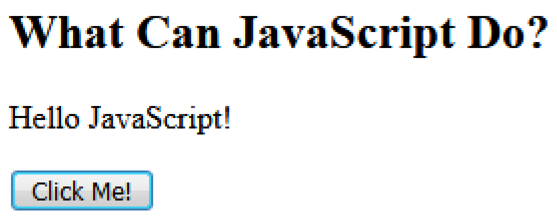
This example uses the method to "find" an HTML element (with id="demo") and changes the element content (innerHTML) to "Hello JavaScript":

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick='document.getElementById("demo").innerHTML =
"Hello JavaScript!"'>Click Me!</button>
</body>
</html>
```


What Can JavaScript Do?

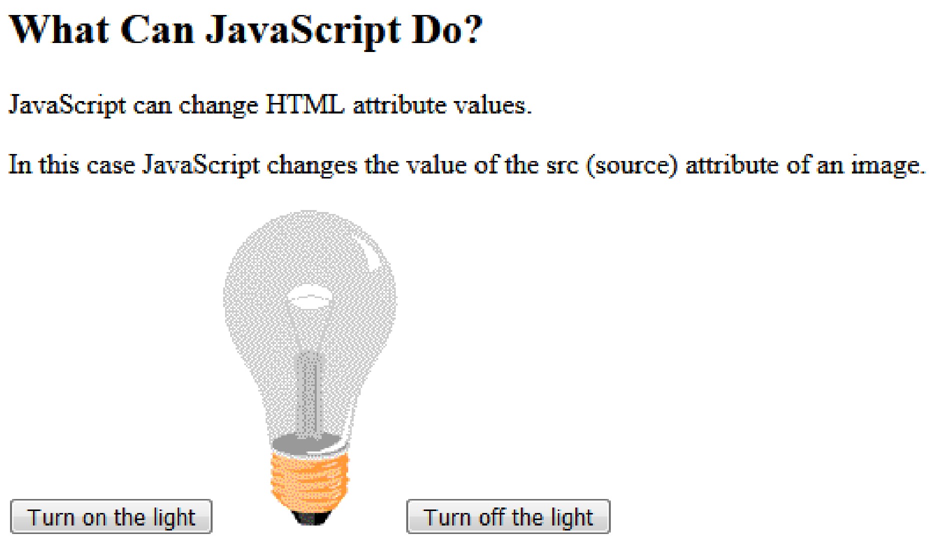
JavaScript can change HTML content.

After Clicking over the command button –


What Can JavaScript Do?

Hello JavaScript!

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p>JavaScript can change HTML attribute values.</p>
<p>In this case JavaScript changes the value of the src (source) attribute of an
image.</p>
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on
the light</button>
<img id="myImage" src="pic_bulboff.gif" style="width:100px">
<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off
the light</button>
</body>
</html>
```


What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.

1.2. JavaScript Statements:

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello GNIT." inside an HTML element with id="demo":

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Statements</h2>
<p>In HTML, JavaScript statements are executed by the browser.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello GNIT.";
</script>
</body>
</html>
```

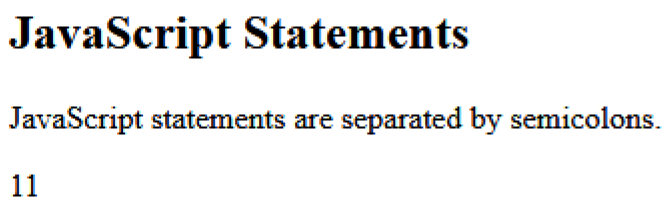
Output

```
JavaScript Statements
In HTML, JavaScript statements are executed by the browser.
Hello GNIT
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Statements</h2>
<p>JavaScript statements are separated by semicolons.</p>
<p id="demo1"></p>
<script>
var a, b, c;
a = 5;
b = 6;
c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>
```

```
</body>
</html>
```

Output



JavaScript Statements

JavaScript statements are separated by semicolons.

11

1.3. JavaScript Keywords:

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

Keyword	**Description**
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

1.4. JavaScript Comments:

- JavaScript comments can be used to explain JavaScript code, and to make it more readable.
- JavaScript comments can also be used to prevent execution, when testing alternative code.

1.4.1. Single Line Comments:

Single line comments start with //.
Any text between // and the end of the line will be ignored by JavaScript (will not be executed).
This example uses a single-line comment before each code line:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="myH"></h1>
<p id="myP"></p>

<script>
// Change heading:

document.getElementById("myH").innerHTML = "JavaScript Comments";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>

</body>
</html>
```

1.4.2. Multi-line Comments:

Multi-line comments start with /* and end with */.
Any text between /* and */ will be ignored by JavaScript.
This example uses a multi-line comment (a comment block) to explain the code:

```
<!DOCTYPE html>
<html>
<body>

<h1 id="myH"></h1>
<p id="myP"></p>
```

```
<script>
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
*/
document.getElementById("myH").innerHTML = "JavaScript Comments";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
</body>
</html>
```

1.5. JavaScript Variables:

1.5.1. Identifiers:

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names.

1.5.2. Variables:

Creating a variable in JavaScript is called "declaring" a variable.
You declare a JavaScript variable with the var keyword:

```
var carName;
```

After the declaration, the variable has no value (technically it has the value of undefined).
To assign a value to the variable, use the equal sign:

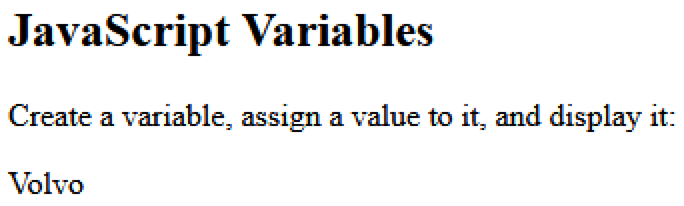
```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:
var carName = "Volvo";

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Variables</h2>
<p>Create a variable, assign a value to it, and display it:</p>
<p id="demo"></p>
<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
</body>
</html>
```



```
JavaScript Variables

Create a variable, assign a value to it, and display it:

Volvo
```

1.5.3. One Statement, Many Variables:

You can declare many variables in one statement.Start the statement with var and separate the variables by **comma**:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Variables</h2>
<p>You can declare many variables in one statement.</p>
<p id="demo"></p>
<script>
var person = "John Doe", carName = "Volvo", price = 200;
document.getElementById("demo").innerHTML = carName;
</script>
</body>
```

</html>

1.6. JavaScript Operators:

1.6.1. JavaScript Arithmetic Operators:

Arithmetic operators perform arithmetic on numbers (literals or variables).

Operator	**Description**
+	Addition
-	Subtraction
*****	Multiplication
******	Exponentiation
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

1.6.2. JavaScript Assignment Operators:

Assignment operators assign values to JavaScript variables.

Operator	**Example**	**Same As**
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

1.6.3. JavaScript Comparison Operators:

Operator	**Description**
= =	equal to

= = =	equal value and equal type
!=	not equal
!= =	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

1.6.4. JavaScript Logical Operators:

Operator	**Description**
&&	logical and
||	logical or
!	logical not

1.6.5. JavaScript Bitwise Operators:

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	**Description**	**Example**	**Same as**	**Result**	**Decimal**
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.
~00000000000000000000000000000101 will return 11111111111111111111111111111010

1.7. JavaScript Data Types:

JavaScript variables can hold many data types: numbers, strings, objects and more:

```
var length = 16;                    // Number
var lastName = "Johnson";                    // String
var x = {firstName:"John", lastName:"Doe"};    // Object
```

In programming, data type is an important concept.To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
var x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
var x = "16" + "Volvo";
```

Lecture 2: JavaScript condition, switch, loop, and break

2.1. JavaScript – if,if-else,if-else-if Statement:

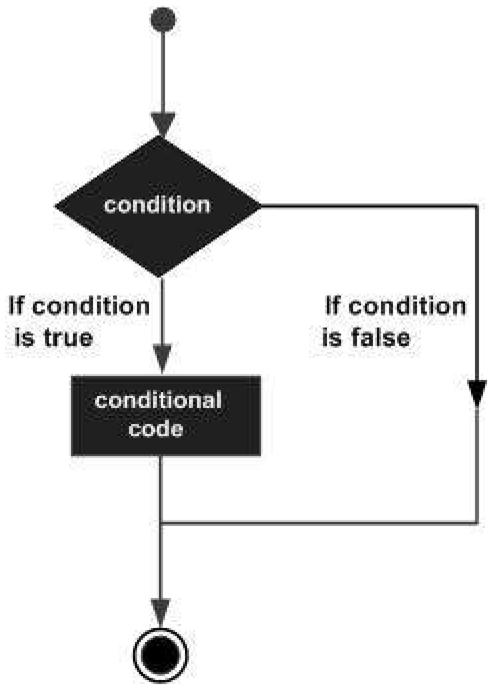
While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..else statement.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.

Decision Making



JavaScript supports the following forms of if..else statement –

- if statement
- if...else statement
- if...else if... statement

2.1.1. if statement:

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows −

```
if (expression) {
  Statement(s) to be executed if expression is true
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

Try the following example to understand how the if statement works.

```
<html>
<body>
<script type = "text/javascript">
<!--
        var age = 20;
        if( age > 18 ) {
          document.write("<b>Qualifies for driving</b>");
        }
      //-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Qualifies for driving
Set the variable to different value and then try...
```

2.1.2. if...else statement:

The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression) {
  Statement(s) to be executed if expression is true
} else {
  Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
<body>
<script type = "text/javascript">
<!--
        var age = 15;

        if( age > 18 ) {
          document.write("<b>Qualifies for driving</b>");
        } else {
          document.write("<b>Does not qualify for driving</b>");
        }
      //-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Does not qualify for driving
Set the variable to different value and then try...
```

2.1.3. if...else if... statement:

The if...else if... statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows −

```
if (expression 1) {
  Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
  Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
  Statement(s) to be executed if expression 3 is true
} else {
  Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>
<body>
<script type = "text/javascript">
<!--
        var book = "maths";
        if( book == "history" ) {
          document.write("<b>History Book</b>");
        } else if( book == "maths" ) {
          document.write("<b>Maths Book</b>");
        } else if( book == "economics" ) {
          document.write("<b>Economics Book</b>");
        } else {
          document.write("<b>Unknown Book</b>");
        }
      //-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
<html>
```

Output

```
Maths Book
Set the variable to different value and then try...
```

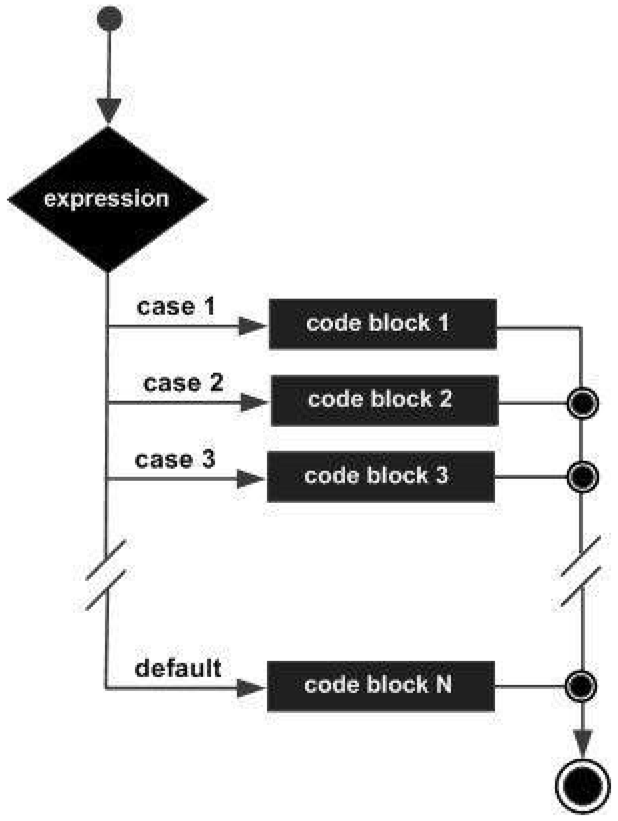
2.2. JavaScript - Switch Case:

You can use multiple if...else…if statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

Flow Chart

The following flow chart explains a switch-case statement works.



The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
switch (expression) {
  case condition 1: statement(s)
  break;

  case condition 2: statement(s)
  break;
  ...

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Example

Try the following example to implement switch-case statement.

```
<html>
<body>
<script type = "text/javascript">
<!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          break;

          case 'B': document.write("Pretty good<br />");
          break;

          case 'C': document.write("Passed<br />");
          break;

          case 'D': document.write("Not so good<br />");
          break;

          case 'F': document.write("Failed<br />");
          break;

          default:  document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...
```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```
<html>
<body>
<script type = "text/javascript">
<!--
        var grade = 'A';
        document.write("Entering switch block<br />");
        switch (grade) {
          case 'A': document.write("Good job<br />");
          case 'B': document.write("Pretty good<br />");
          case 'C': document.write("Passed<br />");
          case 'D': document.write("Not so good<br />");
          case 'F': document.write("Failed<br />");
          default: document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...
```

2.3. JavaScript Loop:

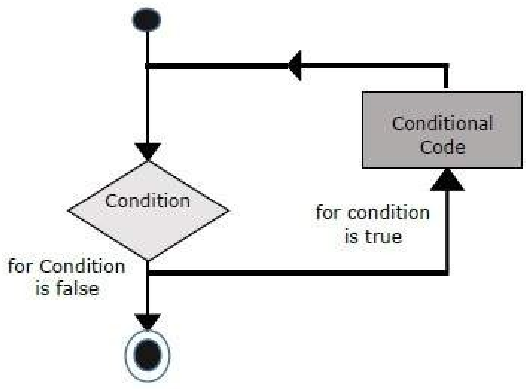
The '**for**' loop is the most compact form of looping. It includes the following three important parts –

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a **for** loop in JavaScript would be as follows –



```
<html>
<body>
<script type = "text/javascript">
<!--
        var count;
        document.write("Starting Loop" + "<br />");

        for(count = 0; count < 10; count++) {
          document.write("Current Count : " + count );
          document.write("<br />");
        }
       document.write("Loop stopped!");
      //-->
```

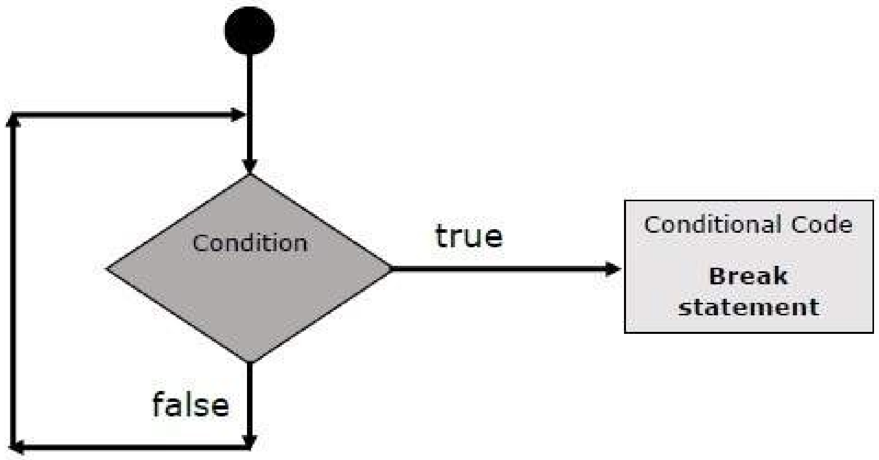
```
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

2.4. The break Statement:

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows −



Example

The following example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once **x** reaches 5 and reaches to **document.write (..)** statement just below to the closing curly brace –

```
<html>
<body>
<script type = "text/javascript">
<!--
      var x = 1;
      document.write("Entering the loop<br /> ");

      while (x < 20) {
            if (x == 5) {
              break;   // breaks out of loop completely
            }
            x = x + 1;
            document.write( x + "<br />");
          }
          document.write("Exiting the loop!<br /> ");
          //-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop
2
3
4
5
Exiting the loop!
Set the variable to different value and then try...
```

We already have seen the usage of break statement inside a switch statement.

2.4.1. The continue Statement:

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

This example illustrates the use of a **continue** statement with a while loop. Notice how the **continue** statement is used to skip printing when the index held in variable **x** reaches 5 –

```
<html>
<body>
```

```
<scripttype="text/javascript">
<!--
var x =1;
        document.write("Entering the loop<br /> ");

while(x <10){
          x = x +1;

if(x ==5){
continue;// skip rest of the loop body
}
          document.write( x +"<br />");
}
        document.write("Exiting the loop!<br /> ");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
Set the variable to different value and then try...
```

Lecture 3: JavaScript functions, objects and events

3.1. JavaScript functions:

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

3.1.1. Function Definition:

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type = "text/javascript">
<!--
    function functionname(parameter-list) {
      statements
    }
  //-->
</script>
```

Example

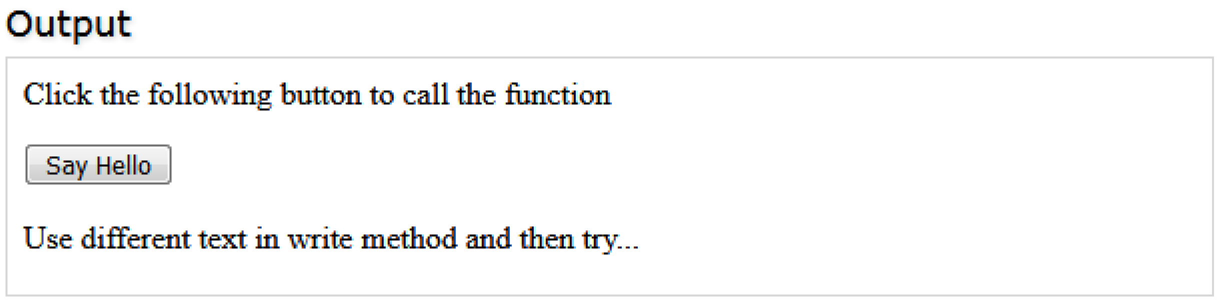
Try the following example. It defines a function called sayHello that takes no parameters −

```
<scripttype="text/javascript">
<!--
function sayHello(){
      alert("Hello there");
}
//-->
</script>
```

3.1.2. Calling a Function:

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<scripttype="text/javascript">
function sayHello(){
        document.write ("Hello there!");
}
</script>

</head>

<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="sayHello()"value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```



```
Output

Click the following button to call the function

[Say Hello]

Use different text in write method and then try...
```

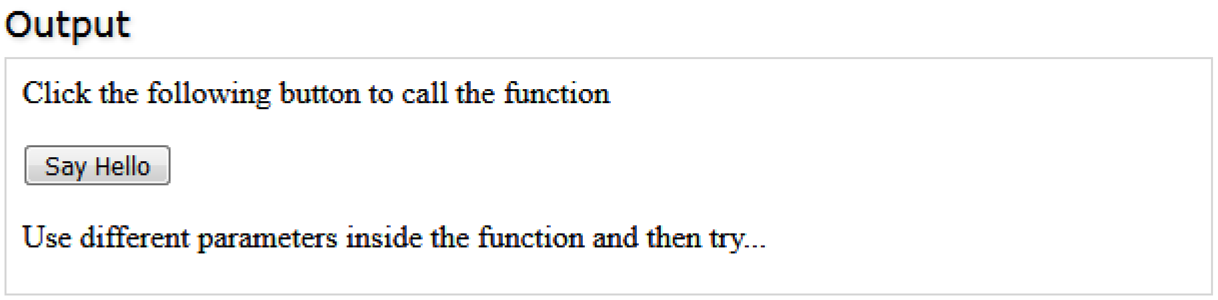
3.1.3. Function Parameters:

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
<head>
<scripttype="text/javascript">
function sayHello(name, age){
        document.write (name +" is "+ age +" years old.");
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="sayHello('Zara',7)"value="Say Hello">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```



```
Output
Click the following button to call the function

Say Hello

Use different parameters inside the function and then try...
```

3.1.4. The return Statement:

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

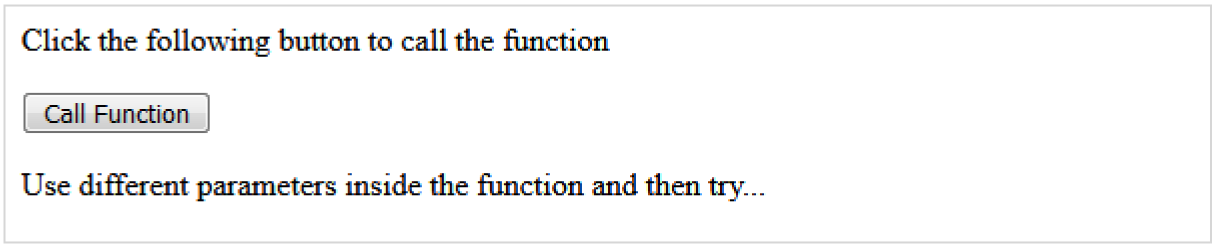
For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
<head>
<scripttype="text/javascript">
function concatenate(first, last){
var full;
        full = first + last;
return full;
}
function secondFunction(){
var result;
        result = concatenate('Zara','Ali');
        document.write (result );
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<inputtype="button"onclick="secondFunction()"value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output



```
Click the following button to call the function

Call Function

Use different parameters inside the function and then try...
```

3.2. JavaScript objects and events:

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** − the capability to store related information, whether data or methods, together in an object.
- **Aggregation** − the capability to store one object inside another object.
- **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.

- **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

3.2.1. Object Properties:

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

```
objectName.objectProperty = propertyValue;
```

Example

The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

3.2.2. Object Methods:

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

Example

Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

3.2.3. User-Defined Objects:

All user-defined objects and built-in objects are descendants of an object called **Object**.

3.2.4. The new Operator:

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee =newObject();
var books =newArray("C++","Perl","Java");
var day =newDate("August 15, 1947");
```

3.3. What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

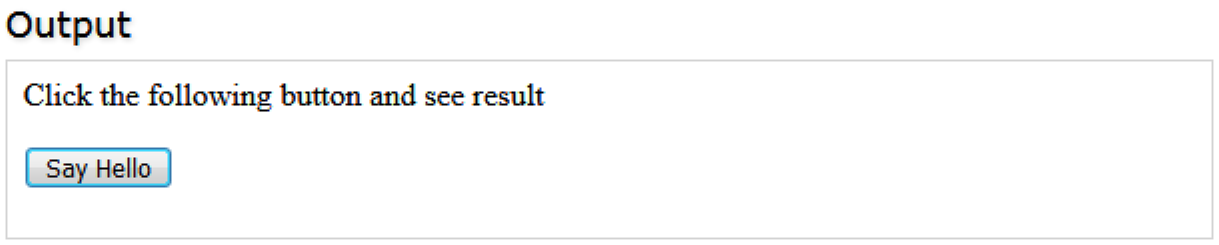
3.3.1. onclick Event Type:

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

```
<html>
<head>
<scripttype="text/javascript">
<!--
function sayHello(){
        alert("Hello World")
}
//-->
</script>
```

```
</head>

<body>
<p>Click the following button and see result</p>
<form>
<inputtype="button"onclick="sayHello()"value="Say Hello"/>
</form>
</body>
</html>
```


Output

Click the following button and see result

The standard HTML 5 events are listed here for your reference. Here script indicates a JavaScript function to be executed against that event.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
onblur	script	Triggers when the window loses focus
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click
oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged
ondragend	script	Triggers at the end of a drag operation
ondragenter	script	Triggers when an element has been dragged to a valid drop

		target
ondragleave	script	Triggers when an element is being dragged over a valid drop target
ondragover	script	Triggers at the start of a drag operation
ondragstart	script	Triggers at the start of a drag operation
ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
onemptied	script	Triggers when a media resource element suddenly becomes empty.
onended	script	Triggers when media has reach the end
onerror	script	Triggers when an error occur
onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes
onforminput	script	Triggers when a form gets user input
onhaschange	script	Triggers when the document has change
oninput	script	Triggers when an element gets user input
oninvalid	script	Triggers when an element is invalid
onkeydown	script	Triggers when a key is pressed
onkeypress	script	Triggers when a key is pressed and released
onkeyup	script	Triggers when a key is released
onload	script	Triggers when the document loads
onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
onloadstart	script	Triggers when the browser starts to load the media data
onmessage	script	Triggers when the message is triggered
onmousedown	script	Triggers when a mouse button is pressed
onmousemove	script	Triggers when the mouse pointer moves
onmouseout	script	Triggers when the mouse pointer moves out of an element
onmouseover	script	Triggers when the mouse pointer moves over an element
onmouseup	script	Triggers when a mouse button is released
onmousewheel	script	Triggers when the mouse wheel is being rotated
onoffline	script	Triggers when the document goes offline
onoine	script	Triggers when the document comes online
ononline	script	Triggers when the document comes online
onpagehide	script	Triggers when the window is hidden
onpageshow	script	Triggers when the window becomes visible

onpause	script	Triggers when media data is paused
onplay	script	Triggers when media data is going to start playing
onplaying	script	Triggers when media data has start playing
onpopstate	script	Triggers when the window's history changes
onprogress	script	Triggers when the browser is fetching the media data
onratechange	script	Triggers when the media data's playing rate has changed
onreadystatechange	script	Triggers when the ready-state changes
onredo	script	Triggers when the document performs a redo
onresize	script	Triggers when the window is resized
onscroll	script	Triggers when an element's scrollbar is being scrolled
onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	script	Triggers when an element is selected
onstalled	script	Triggers when there is an error in fetching media data
onstorage	script	Triggers when a document loads
onsubmit	script	Triggers when a form is submitted
onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	script	Triggers when media changes its playing position
onundo	script	Triggers when a document performs an undo
onunload	script	Triggers when the user leaves the document
onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	script	Triggers when media has stopped playing, but is expected to resume

Lecture 4: Introduction to CGI Script, Environment Variable, GET and POST Methods

4.1. What is CGI?

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script. The CGI specs are currently maintained by the NCSA and NCSA defines CGI is as follows-*The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.* The current version is CGI/1.1 and CGI/1.2 is under progress.

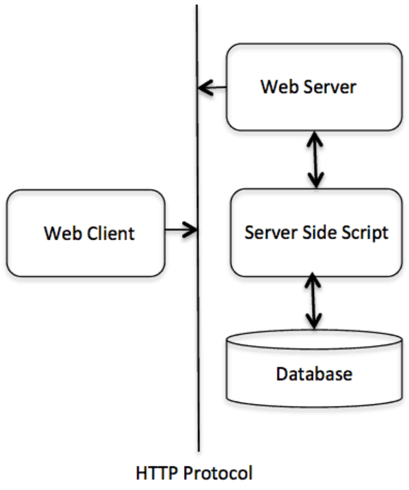
4.1.1. Web Browsing:

To understand the concept of CGI, let’s see what happens when we click a hyper link to browse a particular web page or URL.

- Your browser contacts the HTTP web server and demand for the URL ie. filename.
- Web Server will parse the URL and will look for the filename in if it finds that file then sends back to the browser otherwise sends an error message indicating that you have requested a wrong file.

- Web browser takes response from web server and displays either the received file or error message.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a PERL Script, Shell Script, C or C++ program etc.



4.1.2. First CGI Program:

```
#!/usr/bin/perl

print"Content-type:text/html\r\n\r\n";
print'<html>';
print'<head>';
print'<title>Hello Word - First CGI Program</title>';
print'</head>';
print'<body>';
print'<h2>Hello Word! This is my first CGI program</h2>';
print'</body>';
print'</html>';

1;
```

Output

```
Hello Word! This is my first CGI program
```

4.1.3. CGI Environment Variables:

All the CGI program will have access to the following environment variables. These variables play an important role while writing any CGI program.

Sl.No.	Variable Name & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example file upload etc.
2	**CONTENT_LENGTH** The length of the query information. It's available only for POST requests.
3	**HTTP_COOKIE** Return the set cookies in the form of key & value pair.
4	**HTTP_USER_AGENT** The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser.
5	**PATH_INFO** The path for the CGI script.
6	**QUERY_STRING** The URL-encoded information that is sent with GET method request.
7	**REMOTE_ADDR** The IP address of the remote host making the request. This can be useful for logging or for authentication purpose.
8	**REMOTE_HOST** The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR address.
9	**REQUEST_METHOD** The method used to make the request. The most common methods are GET and POST.
10	**SCRIPT_FILENAME** The full path to the CGI script.
11	**SCRIPT_NAME** The name of the CGI script.
12	**SERVER_NAME** The server's hostname or IP Address.
13	**SERVER_SOFTWARE** The name and version of the software the server is running.

4.1.4. GET and POST Methods:

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

4.1.4.1. Passing Information using GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows − **http://www.test.com/cgi-bin/hello.cgi?key1=value1&key2=value2**

The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be in a request string.

This information is passed using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable.

You can pass information by simply concatenating key and value pairs along with any URL or you can use HTML <FORM> tags to pass information using GET method.

4.1.4.2. Simple URL Example: Get Method

Here is a simple URL which will pass two values to hello_get.cgi program using GET method. Below is hello_get.cgi script to handle input given by web browser.

```
#!/usr/bin/perl
local($buffer,@pairs, $pair, $name, $value,%FORM);
# Read in text
$ENV{'REQUEST_METHOD'}=~ tr/a-z/A-Z/;

if($ENV{'REQUEST_METHOD'} eq "GET"){
  $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs= split(/&/, $buffer);

foreach $pair (@pairs){
($name, $value)= split(/=/, $pair);
  $value =~ tr/+/ /;
  $value =~ s/%(..)/pack("C", hex($1))/eg;
  $FORM{$name}= $value;
}

$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print"Content-type:text/html\r\n\r\n";
print"<html>";
print"<head>";
print"<title>Hello - Second CGI Program</title>";
print"</head>";
print"<body>";
print"<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print"</body>";
print"</html>";

1;
```

Output

```
Hello  ZARA ALI .....
```

4.1.4.3. Passing Information using POST method:

A generally more reliable method of passing information to a CGI program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes into the CGI script in the form of the standard input.

Below is hello_post.cgi script to handle input given by web browser. This script will handle GET as well as POST method.

```
#!/usr/bin/perl

local($buffer,@pairs, $pair, $name, $value,%FORM);
# Read in text
$ENV{'REQUEST_METHOD'}=~ tr/a-z/A-Z/;

if($ENV{'REQUEST_METHOD'} eq "POST"){
  read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else{
  $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs= split(/&/, $buffer);

foreach $pair (@pairs){
($name, $value)= split(/=/, $pair);
  $value =~ tr/+/ /;
  $value =~ s/%(..)/pack("C", hex($1))/eg;
  $FORM{$name}= $value;
}

$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print"Content-type:text/html\r\n\r\n";
print"<html>";
print"<head>";
print"<title>Hello - Second CGI Program</title>";
print"</head>";
print"<body>";
print"<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print"</body>";
print"</html>";

1;
```

Lecture 5: Introduction to PHP Script, Syntax, Variables, Output, Data types, String, Constants

5.1. Introduction:

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the UNIX side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.

5.2. Characteristics of PHP:

Five important characteristics make PHP's practical nature possible –

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this −

```
<html>

<head>
<title>Hello World</title>
</head>

<body>
<?php echo "Hello, World!";?>
</body>

</html>
```

5.3. Syntax:

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP'. There are four ways to do this –

Canonical PHP tags

The most universally effective PHP tag style is −

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

Short-open (SGML-style) tags

Short or short-open tags look like this −

```
<?...?>
```

Short tags are, as one might expect, the shortest option. You must do one of two things to enable PHP to recognize the tags –

- Choose the – enable-short-tags configuration option when you're building PHP.
- Set the short_open_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

ASP-style tags

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this −

```
<%...%>
```

To use ASP-style tags, you will need to set the configuration option in your php.ini file.

HTML script tags

HTML script tags look like this −

```
<script language = "PHP">...</script>
```

5.4. Commenting PHP Code:

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP −

Single-line comments

They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
# This is a comment, and
# This is the second line of the comment

// This is a comment too. Each style comments only
print"An example with single line comments";
?>
```

Multi-lines comments

They are generally used to provide pseudocode algorithms and moredetailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?
/* This is a comment with multiline
    Author : Mohammad Mohtashim
    Purpose: Multiline Comments Demo
    Subject: PHP
  */

print"An example with multi line comments";
?>
```

PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row.one whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable $four is equivalent −

```
$four =2+2;// single spaces
$four <tab>=<tab2<tab>+<tab>2;// spaces and tabs
$four =
2+
2;// multiple lines
```

PHP is case sensitive

Yeah it is true that PHP is a case sensitive language. Try out following example −

```
<html>
<body>

<?php
      $capital =67;
print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");
```

```
?>

</body>
</html>
```

This will produce the following result −

```
Variable capital is 67
Variable CaPiTaL is
```

Statements are expressions terminated by semicolons

A *statement* in PHP is any expression that is followed by a semicolon (;).Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called $greeting −

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth.

Braces make blocks

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent −

```
if(3==2+1)
print("Good - I haven't totally lost my mind.<br>");

if(3==2+1)
{
print("Good - I haven't totally");
print("lost my mind.<br>");
```

}

5.5. Variables:

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

5.6. Data types:

PHP has a total of eight data types which we use to construct our variables **–**

- **Integers** − are whole numbers, without a decimal point, like 4195.
- **Doubles** − are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** − have only two possible values either true or false.
- **NULL** − is a special type that only has one value: NULL**.**
- **Strings** − are sequences of characters, like 'PHP supports string operations.

- **Arrays** − are named and indexed collections of other values.

- **Objects** − are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.

- **Resources** − are special variables that hold references to resources external to PHP (such as database connections).

5.7. String:

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?php
  $variable ="name";
  $literally ='My $variable will not print!\\n';
print($literally);
print"<br />";
  $literally ="My $variable will print!\\n";
print($literally);
?>
```

This will produce the following result −

```
My $variable will not print!\n
My name will print!\n
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP –

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are –

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character

- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

5.7.1. String Concatenation Operator:

To concatenate two string variables together, use the dot (.) operator –

```
<?php
  $string1="Hello World";
  $string2="1234";

  echo $string1 ." ". $string2;
?>
```

This will produce the following result −

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

5.7.2. Using the strlen() function:

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!" −

```
<?php
  echo strlen("Hello world!");
?>
```

This will produce the following result −

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

5.7.3. Using the strops() function:

The strpos() function is used to search for a string or character within a string.
If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string −

```
<?php
  echo strpos("Hello world!","world");
?>
```

This will produce the following result −

```
 6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

5.8. Constants:

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
A valid constant name starts with a letter or underscore (no $ sign before the constant name).
Note: Unlike variables, constants are automatically global across the entire script.

5.8.1. Create a PHP Constant:

To create a constant, use the define() function.

Syntax

define(*name*, *value*, *case-insensitive*)

Parameters

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

The example below creates a constant with a **case-sensitive** name:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

The example below creates a constant with a **case-insensitive** name:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```

5.8.2. Constants are Global:

Constants are automatically global and can be used across the entire script.

The example below uses a constant inside a function, even if it is defined outside the function:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
```

```
function myTest() {
   echo GREETING;
}
 myTest();
?>
```

Lecture 6: Operator, Decision Control statements

6.1. Operator:

Operators are used to perform operations on some values. In other words, we can describe operators as something that takes some values, performs some operation on them and gives a result. From example, “1 + 2 = 3” in this expression ‘+’ is an operator. It takes two values 1 and 2, performs addition operation on them to give 3.

Just like any other programming language, PHP also supports various types of operations like the arithmetic operations (addition, subtraction, etc), logical operations (AND, OR etc), Increment/Decrement Operations etc. Thus, PHP provides us with many operators to perform such operations on various operands or variables or values. These operators are nothing but symbols needed to perform operations of various types. Given below are the various groups of operators:

- Arithmetic Operators
- Logical or Relational Operators
- Comparison Operators
- Conditional or Ternary Operators
- Assignment Operators
- Spaceship Operators (Introduced in PHP 7)
- Array Operators
- Increment/Decrement Operators
- String Operators

Let us now learn about each of these operators in details:

6.1.1. Arithmetic Operators:

The arithmetic operators are used to perform simple mathematical operations like addition, subtraction, multiplication etc. Below is the list of arithmetic operators along with their syntax and operations that PHP provides us:

OPERATOR	**NAME**	**SYNTAX**	**OPERATION**
+	Addition	$x + $y	Sum the operands
–	Subtraction	$x – $y	Differences the operands
*	Multiplication	$x * $y	Product of the operands
/	Division	$x / $y	Quotient of the operands
**	Exponentiation	$x ** $y	$x raised to the power $y
%	Modulus	$x % $y	Remainder of the operands

Note: The exponentiation has been introduced in PHP 5.6.

Example

```
<?php

// variable 1
$x = 29;

// variable 2
$y = 4;
```

```
// some arithmetic operations on
// these two variables
echo ($x + $y), "\n";
echo($x - $y), "\n";
echo($x * $y), "\n";
echo($x / $y), "\n";
echo($x % $y), "\n";

?>
```

Output

```
33
25
116
7.25
1
```

6.1.2. Logical or Relational Operators:

These are basically used to operate with conditional statements and expressions. Conditional statements are based on conditions. Also, a condition can either be met or cannot be met so the result of a conditional statement can either be true or false. Here are the logical operators along with their syntax and operations that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
and	Logical AND	$x and $y	True if both the operands are true else false
or	Logical OR	$x or $y	True if either of the operand is true else false
xor	Logical XOR	$x xor $y	True if either of the operand is true and false if both are true
&&	Logical AND	$x && $y	True if both the operands are true else false
||	Logical OR	$x || $y	True if either of the operand is true else false
!	Logical NOT	!$x	True if $x is false

Example

```
<?php
```

```
$x = 50;
$y = 30;

if ($x == 50 and $y == 30)
        echo "and Success \n";

if ($x == 50 or $y == 20)
        echo "or Success \n";

if ($x == 50 xor $y == 20)
        echo "xor Success \n";

if ($x == 50 && $y == 30)
        echo "&& Success \n";

if ($x == 50 || $y == 20)
        echo "|| Success \n";

if (!$z)
        echo "! Success \n";

?>
```

Output

```
and Success
or Success
xor Success
&& Success
|| Success
! Success
```

6.1.3. Comparison Operators:

These operators are used to compare two elements and outputs the result in boolean form. Here are the comparison operators along with their syntax and operations that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
==	Equal To	$x == $y	Returns True if both the operands are equal
!=	Not Equal To	$x != $y	Returns True if both the operands are not equal

<>	Not Equal To	$x <> $y	Returns True if both the operands are unequal
===	Identical	$x === $y	Returns True if both the operands are equal and are of the same type
!==	Not Identical	$x == $y	Returns True if both the operands are unequal and are of different types
<	Less Than	$x < $y	Returns True if $x is less than $y
>	Greater Than	$x > $y	Returns True if $x is greater than $y
<=	Less Than or Equal To	$x <= $y	Returns True if $x is less than or equal to $y
>=	Greater Than or Equal To	$x >= $y	Returns True if $x is greater than or equal to $y

Example

```
<?php

$a = 80;
$b = 50;
$c = "80";

// Here var_dump function has been used to
// display structured information. We will learn
// about this function in complete details in further
// articles.
var_dump($a == $c) + "\n";
var_dump($a != $b) + "\n";
var_dump($a <> $b) + "\n";
var_dump($a === $c) + "\n";
var_dump($a !== $c) + "\n";
var_dump($a < $b) + "\n";
var_dump($a > $b) + "\n";
var_dump($a <= $b) + "\n";
var_dump($a >= $b);

?>
```

Output

```
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
```

6.1.4. Conditional or Ternary Operators:

These operators are used to compare two values and take either of the result simultaneously, depending on whether the outcome is TRUE or FALSE. These are also used as shorthand notation for if…else statement that we will read in the article on decision making.

Syntax

```
$var = (condition)? value1 : value2;
```

Here, condition will either evaluate to true or false. If the condition evaluates to True, then value1 will be assigned to the variable $var otherwise value2 will be assigned to it.

OPERATOR	NAME	OPERATION
?:	Ternary	If condition is true ? then $x : or else $y. This means that if condition is true then left result of the colon is accepted otherwise the result on right.

Example:

```
<?php
$x = -12;
echo ($x > 0) ? 'The number is positive' : 'The number is negative';
?>
```

Output

```
The number is negative
```

6.1.5. Assignment Operators:

These operators are used to assign values to different variable, with or without mid-operations. Here are the assignment operators along with their syntax and operations, that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
=	Assign	$x = $y	Operand on the left obtains the value of the operand on right
+=	Add then Assign	$x += $y	Simple Addition same as $x = $x + $y
-=	Subtract then Assign	$x -= $y	Simple subtraction same as $x = $x – $y
***=**	Multiply then Assign	$x *= $y	Simple product same as $x = $x * $y
/=	Divide then Assign (quotient)	$x /= $y	Simple division same as $x = $x / $y
%=	Divide then Assign (remainder)	$x %= $y	Simple division same as $x = $x % $y

Example

```
<?php

// simple assign operator
$y = 75;
echo $y, "\n";

// add then assign operator
$y = 100;
$y += 200;
echo $y, "\n";

// subtract then assign operator
$y = 70;
$y -= 10;
echo $y, "\n";

// multiply then assign operator
$y = 30;
$y *= 20;
echo $y, "\n";

// Divide then assign(quotient) operator
$y = 100;
$y /= 5;
echo $y, "\n";
```

```
// Divide then assign(remainder) operator
$y = 50;
$y %= 5;
echo $y;

?>
```

Output

```
75
300
60
600
20
0
```

6.1.6. Array Operators:

These operators are used in case of arrays. Here are the array operators along with their syntax and operations that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
+	Union	$x + $y	Union of both i.e., $x and $y
==	Equality	$x == $y	Returns true if both has same key-value pair
!=	Inequality	$x != $y	Returs True if both are unequal
===	Identity	$x === $y	Returns True if both has same key-value pair in the same order and of same type
!==	Non-Identity	$x !== $y	Returns True if both are not identical to each other
<>	Inequality	$x <> $y	Returns True if both are unequal

Example

```
<?php

$x = array("k" => "Car", "l" => "Bike");
$y = array("a" => "Train", "b" => "Plane");

var_dump($x + $y);
var_dump($x == $y) + "\n";
var_dump($x != $y) + "\n";
var_dump($x <> $y) + "\n";
var_dump($x === $y) + "\n";
var_dump($x !== $y) + "\n";

?>
```

Output

```
array(4) {
  ["k"]=>
  string(3) "Car"
  ["l"]=>
  string(4) "Bike"
  ["a"]=>
  string(5) "Train"
  ["b"]=>
  string(5) "Plane"
}
bool(false)
bool(true)
bool(true)
bool(false)
bool(true)
```

6.1.7. Increment/Decrement Operators:

These are called the unary operators as it work on single operands. These are used to increment or decrement values.

OPERATOR	NAME	SYNTAX	OPERATION
++	Pre-Increment	++$x	First increments $x by one, then return $x
—	Pre-Decrement	–$x	First decrements $x by one, then return $x
++	Post-Increment	$x++	First returns $x, then increment it by one
—	Post-Decrement	$x–	First returns $x, then decrement it by one

Example

```
<?php

$x = 2;
echo ++$x, " First increments then prints \n";
echo $x, "\n";

$x = 2;
echo $x++, " First prints then increments \n";
echo $x, "\n";

$x = 2;
echo --$x, " First decrements then prints \n";
echo $x, "\n";

$x = 2;
echo $x--, " First prints then decrements \n";
echo $x;

?>
```

Output

```
3 First increments then prints
3
2 First prints then increments
3
1 First decrements then prints
1
2 First prints then decrements
1
```

6.1.8. String Operators:

These are implemented over strings.

OPERATOR	NAME	SYNTAX	OPERATION
.	Concatenation	$x.$y	Concatenated $x and $y
.=	Concatenation and assignment	$x.=$y	First concatenates then assigns, same as $x = $x.$y

Example

```
<?php

$x = "AB";
$y = "c";
```

```
$z = "DE!!!";
echo $x . $y . $z, "\n";

$x .= $y . $z;
echo $x;

?>
```

Output

```
ABcDE
ABcDE
```

6.1.9. Spaceship Operators (Introduced in PHP 7):

PHP 7 has introduced a new kind of operator called spaceship operator (). These operators are used to compare values but instead of returning boolean result, it returns integer values. If both the operands are equal, it returns 0. If the right operand is greater, it returns -1. If the left operand is greater, it returns 1. The following table shows how it works in detail:

OPERATOR	SYNTAX	OPERATION
$x < $y	$x <=> $y	Identical to -1 (right is greater)
$x > $y	$x <=> $y	Identical to 1 (left is greater)
$x <= $y	$x <=> $y	Identical to -1 (right is greater) or identical to 0 (if both are equal)
$x >= $y	$x <=> $y	Identical to 1 (if left is greater) or identical to 0 (if both are equal)
$x == $y	$x <=> $y	Identical to 0 (both are equal)
$x != $y	$x <=> $y	Not Identical to 0

Example

```
<?php

$x = 50;
$y = 50;
$z = 25;

echo $x <=> $y;
echo "\n";

echo $x <=> $z;
echo "\n";
echo $z <=> $y;
echo "\n";

// We can do the same for Strings
$x = "Ram";
$y = "Krishna";

echo $x <=> $y;
echo "\n";

echo $x <=> $y;
echo "\n";

echo $y <=> $x;

?>
```

Output

```
0
1
-1
1
1
-1
```

6.2. Decision Control statements:

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif....else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

6.2.1. The if Statement:

The if statement executes some code if one condition is true.

Syntax

```
if (condition) {
   code to be executed if condition is true;
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php
$t = date("H");

if ($t <"20") {
   echo"Have a good day!";
}
?>
```

6.2.2. The if...else Statement:

The if....else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {
   code to be executed if condition is true;
} else {
   code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php
$t = date("H");

if ($t <"20") {
  echo"Have a good day!";
} else {
   echo"Have a good night!";
}
?>
```

6.2.3. The if...elseif....else Statement:

The if....elseif...else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
   code to be executed if this condition is true;
} elseif (condition) {
   code to be executed if this condition is true;
} else {
   code to be executed if all conditions are false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
$t = date("H");

if ($t <"10") {
   echo"Have a good morning!";
} elseif ($t <"20") {
   echo"Have a good day!";
} else {
   echo"Have a good night!";
}
?>
```

Lecture 7: switch-case in PHP, Loop, and PHP function

7.1. The PHP switch Statement:

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example

```
<?php
$favcolor = "red";

switch ($favcolor){
    case"red":
        echo"Your favorite color is red!";
      break;
  case"blue":
        echo"Your favorite color is blue!";
      break;
  case"green":
        echo"Your favorite color is green!";
      break;
  default:
        echo"Your favorite color is neither red, blue, nor green!";
}
?>
```

7.2. Loop:

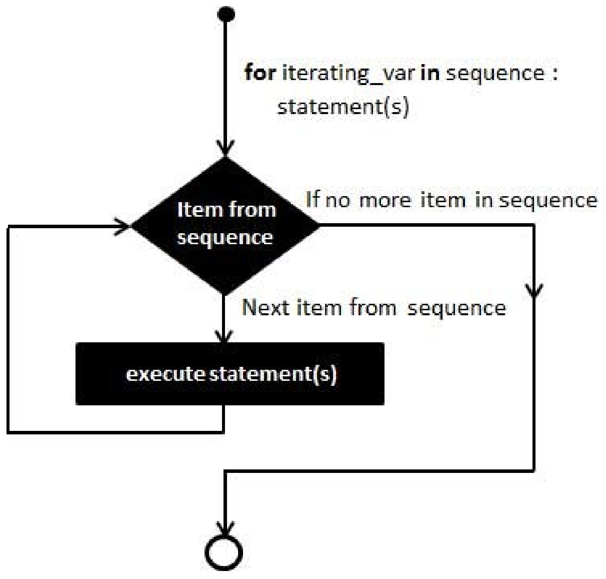
Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** − loops through a block of code a specified number of times.
- **while** − loops through a block of code if and as long as a specified condition is true.
- **do...while** − loops through a block of code once, and then repeats the loop as long as a special condition is true.
- **for-each** − loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

7.2.1. The for loop statement:

The for statement is used when you know how many times you want to execute a statement or a block of statements.



Syntax

for (*initialization*; *condition*; *increment*){
code to be executed;
}

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop −

```
<html>
<body>

<?php
   $a =0;
   $b =0;

for( $i =0; $i<5; $i++){
    $a +=10;
    $b +=5;
}

   echo ("At the end of the loop a = $a and b = $b");
?>

</body>
</html>
```

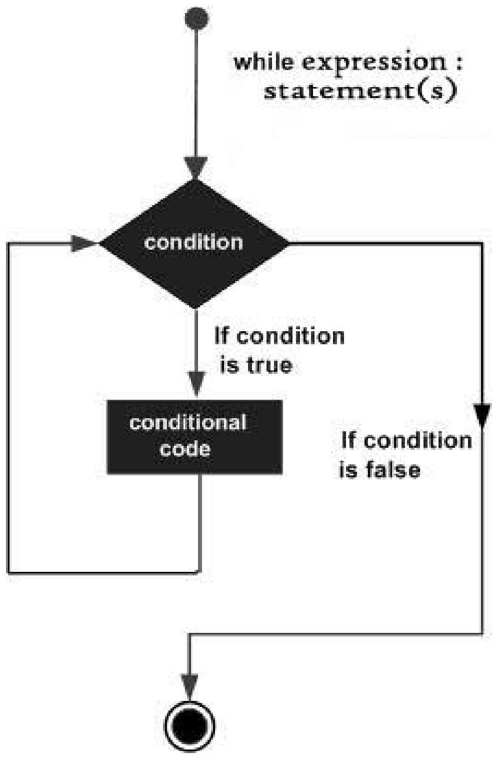
This will produce the following result –

```
At the end of the loop a = 50 and b = 25
```

7.2.2. The while loop statement:

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.



Syntax

```
while (condition) {
code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
<body>

<?php
      $i =0;
      $num =50;

while( $i <10){
        $num--;
        $i++;
}

    echo ("Loop stopped at i = $i and num = $num");
?>

</body>
</html>
```

This will produce the following result –

```
Loop stopped at I = 10 and num = 40
```

7.2.3. The do...while loop statement:

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do {
code to be executed;
}
while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 −

```
<html>
<body>

<?php
    $i =0;
    $num =0;

do{
     $i++;
}

while( $i <10);
```

```
      echo ("Loop stopped at i = $i");
?>

</body>
</html>
```

This will produce the following result –

```
Loop stopped at I = 10
```

7.2.4. The for-each loop statement:

The for-each statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value) {
code to be executed;
}
```

Example

Try out following example to list out the values of an array.

```
<html>
<body>

<?php
      $array = array(1,2,3,4,5);

foreach( $array as $value ){
        echo "Value is $value <br />";
}
?>

</body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

7.3.PHP function:

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you –

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

7.3.1. Creating PHP Function:

It’s very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below −

```
<html>

<head>
<title>Writing PHP Function</title>
</head>

<body>

<?php
```

```
/* Defining a PHP Function */
function writeMessage(){
        echo "You are really a nice person, Have a nice time!";
}

/* Calling a PHP Function */
      writeMessage();
?>

</body>
</html>
```

This will display following result –

```
You are really a nice person, Have a nice time!
```

7.3.2. PHP Functions with Parameters:

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>

<head>
<title>Writing PHP Function with Parameters</title>
</head>

<body>

<?php
function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        echo "Sum of the two numbers is : $sum";
      }

      addFunction(10, 20);
    ?>

</body>
```

```
</html>
```

This will display following result –

```
Sum of the two numbers is : 30
```

7.3.3. Passing Arguments by Reference:

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

```
<html>

<head>
<title>Passing Argument by Reference</title>
</head>

<body>

<?php
function addFive($num) {
    $num += 5;
  }

function addSix(&$num) {
    $num += 6;
  }

  $orignum = 10;
  addFive( $orignum );

  echo "Original Value is $orignum<br />";

  addSix( $orignum );
  echo "Original Value is $orignum<br />";
```

```
   ?>

</body>
</html>
```

This will display following result −

```
Original Value is 10
Original Value is 16
```

7.3.4. PHP Functions returning value:

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program.

Note that **return** keyword is used to return a value from a function.

```
<html>

<head>
<title>Writing PHP Function which returns value</title>
</head>

<body>

<?php
function addFunction($num1, $num2){
        $sum = $num1 + $num2;
return $sum;
}
      $return_value = addFunction(10,20);

      echo "Returned value from the function : $return_value";
?>

</body>
</html>
```

This will display following result –

```
Returned value from the function : 30
```

Lecture 8: PHP array, Form Handling

8.1. PHP Array:

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables it’s easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** − an array with a numeric index. Values are stored and accessed in linear fashion.
- **Associative array** − an array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** − an array containing one or more arrays and values are accessed using multiple indices.

8.1.1. Numeric Array:

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
<body>

<?php
/* First method to create array. */
      $numbers = array(1,2,3,4,5);

foreach( $numbers as $value ){
        echo "Value is $value <br />";
}

/* Second method to create array. */
      $numbers[0]="one";
      $numbers[1]="two";
      $numbers[2]="three";
      $numbers[3]="four";
      $numbers[4]="five";

foreach( $numbers as $value ){
      echo "Value is $value <br />";
}
?>

</body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

8.1.2. Associative Arrays:

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

Example

```
<html>
<body>

<?php
/* First method to associate create array. */
      $salaries = array("mohammad"=>2000,"qadir"=>1000,"zara"=>500);
      echo "Salary of mohammad is ". $salaries['mohammad']."<br />";
      echo "Salary of qadir is ".  $salaries['qadir']."<br />";
      echo "Salary of zara is ".  $salaries['zara']."<br />";

/* Second method to create array. */
      $salaries['mohammad']="high";
      $salaries['qadir']="medium";
      $salaries['zara']="low";

      echo "Salary of mohammad is ". $salaries['mohammad']."<br />";
      echo "Salary of qadir is ".  $salaries['qadir']."<br />";
      echo "Salary of zara is ".  $salaries['zara']."<br />";
?>

</body>
</html>
```

This will produce the following result −

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

8.1.3. Multidimensional Arrays:

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects −

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
```

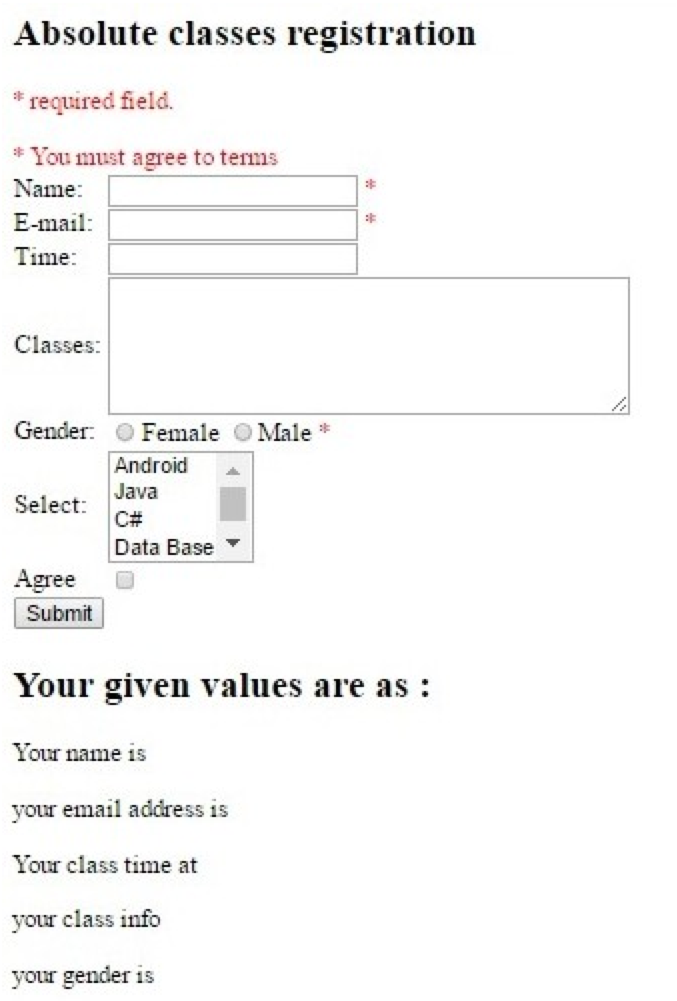
```
<body>

<?php
        $marks = array(
"mohammad"=> array (
"physics"=>35,
"maths"=>30,
"chemistry"=>39
),

"qadir"=> array (
"physics"=>30,
"maths"=>32,
"chemistry"=>29
),

"zara"=> array (
"physics"=>31,
"maths"=>22,
"chemistry"=>39
)
);

/* Accessing multi-dimensional array values */
        echo "Marks for mohammad in physics : ";
        echo $marks['mohammad']['physics']."<br />";

        echo "Marks for qadir in maths : ";
        echo $marks['qadir']['maths']."<br />";

        echo "Marks for zara in chemistry : ";
        echo $marks['zara']['chemistry']."<br />";
?>

</body>
</html>
```

This will produce the following result −

```
Marks for mohammad in physics : 35
```

Marks for qadir in maths : 32
Marks for zara in chemistry : 39

8.2. PHP Form Handling:



Absolute classes registration

* required field.

* You must agree to terms

Name: *
E-mail: *
Time:
Classes:
Gender: Female Male *
Select: Android Java C# Data Base
Agree
Submit

Your given values are as :

Your name is

your email address is

Your class time at

your class info

your gender is

8.2.1. PHP Code:

```
<html>
<head>
<style>
.error {color:#FF0000;}
</style>
</head>
```

```
<body>
<?php
// define variables and set to empty values
        $nameErr = $emailErr = $genderErr = $websiteErr ="";
        $name = $email = $gender = $class = $course = $subject ="";

if($_SERVER["REQUEST_METHOD"]=="POST"){
if(empty($_POST["name"])){
            $nameErr ="Name is required";
}else{
            $name = test_input($_POST["name"]);
}

if(empty($_POST["email"])){
            $emailErr ="Email is required";
}else{
            $email = test_input($_POST["email"]);

// check if e-mail address is well-formed
if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
              $emailErr ="Invalid email format";
}
}

if(empty($_POST["course"])){
            $course ="";
}else{
            $course = test_input($_POST["course"]);
}

if(empty($_POST["class"])){
            $class ="";
}else{
            $class = test_input($_POST["class"]);
}

if(empty($_POST["gender"])){
            $genderErr ="Gender is required";
}else{
            $gender = test_input($_POST["gender"]);
}

if(empty($_POST["subject"])){
            $subjectErr ="You must select 1 or more";
}else{
            $subject = $_POST["subject"];
}
}

function test_input($data){
          $data = trim($data);
          $data = stripslashes($data);
          $data = htmlspecialchars($data);
return $data;
}
?>

<h2>Absolute classes registration</h2>

<p><spanclass="error">* required field.</span></p>

          <form method = "POST" action = "<?php echo
          htmlspecialchars($_SERVER["PHP_SELF"]);?>">
<table>
<tr>
<td>Name:</td>
<td><inputtype="text"name="name">
<spanclass="error">* <?php echo $nameErr;?></span>
</td>
</tr>

<tr>
<td>E-mail: </td>
<td><inputtype="text"name="email">
<spanclass="error">* <?php echo $emailErr;?></span>
</td>
</tr>

<tr>
<td>Time:</td>
```

```
<td><inputtype="text"name="course">
<spanclass="error"><?php echo $websiteErr;?></span>
</td>
</tr>

<tr>
<td>Classes:</td>
<td><textareaname="class"rows="5"cols="40"></textarea></td>
</tr>

<tr>
<td>Gender:</td>
<td>
<inputtype="radio"name="gender"value="female">Female
<inputtype="radio"name="gender"value="male">Male
<spanclass="error">* <?php echo $genderErr;?></span>
</td>
</tr>

<tr>
<td>Select:</td>
<td>
<selectname="subject[]"size="4"multiple>
<optionvalue="Android">Android</option>
<optionvalue="Java">Java</option>
<optionvalue="C#">C#</option>
<optionvalue="Data Base">Data Base</option>
<optionvalue="Hadoop">Hadoop</option>
<optionvalue="VB script">VB script</option>
</select>
</td>
</tr>

<tr>
<td>Agree</td>
<td><inputtype="checkbox"name="checked"value="1"></td>
<?php if(!isset($_POST['checked'])){?>
<spanclass="error">* <?php echo "You must agree to terms";?></span>
<?php }?>
```

```
</tr>

<tr>
<td>
<inputtype="submit"name="submit"value="Submit">
</td>
</tr>

</table>
</form>

<?php
        echo "<h2>Your given values are as :</h2>";
        echo ("<p>Your name is $name</p>");
        echo ("<p> your email address is $email</p>");
        echo ("<p>Your class time at $course</p>");
        echo ("<p>your class info $class </p>");
        echo ("<p>your gender is $gender</p>");

for($i =0; $i < count($subject); $i++){
            echo($subject[$i]." ");
}
?>
</body>
</html>
```

MODULE 4: Java Server Page, Java Servlet&.NET Framework

Lecture 1: Introduction to JSP, JSP Architecture

1.1. What is Java Server Pages?

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

1.2. Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- Java Server Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP, etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

1.3. Advantages of JSP:

Following table lists out the other advantages of using JSP over other technologies −

vs. Active Server Pages (ASP)

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI)

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML

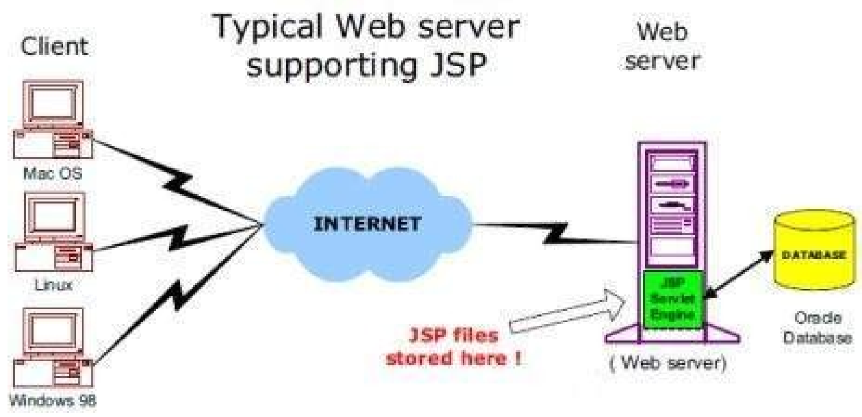
Regular HTML, of course, cannot contain dynamic information.

1.4. JSP – Architecture:

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.



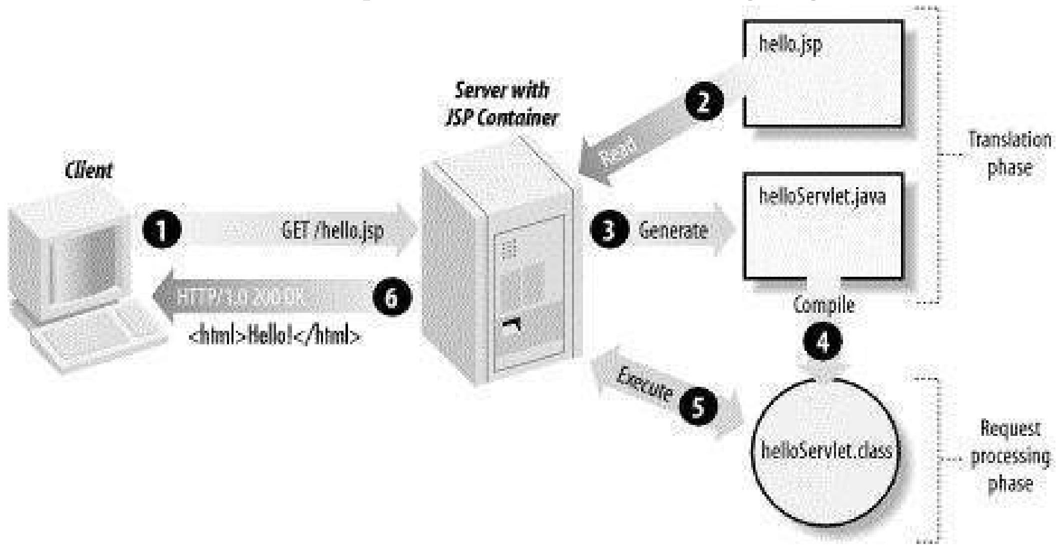
1.4.1. JSP Processing:

The following steps explain how the web server creates the Webpage using JSP –

- As with a normal page, your browser sends an HTTP request to the web server.

- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behaviour of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram –



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

Lecture 2: JSP Servers, JSP Life Cycle, JSP Page Creation and Execution

2.1. JSP Servers:

If you do not have a JSP capable web-server or application server, the first step is to download one. There are many such servers available, most of which can be downloaded for free evaluation and/or development. Some of them are:

- Blazix from Desiderata Software (1.5 Megabytes, JSP, Servlets and EJBs)
- TomCat from Apache (Approx 6 Megabytes)
- WebLogic from BEA Systems (Approx 40 Megabytes, JSP, Servlets and EJBs)
- WebSphere from IBM (Approx 100 Megabytes, JSP, Servlets and EJBs)

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server. Here are the steps to set up Tomcat on your machine –

- Download the latest version of Tomcat from https://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example, in **C:\apache-tomcat-5.5.29 on windows**, or **/usr/local/apache-tomcat-5.5.29 on Linux/Unix** and create **CATALINA_HOME** environment variable pointing to these locations.

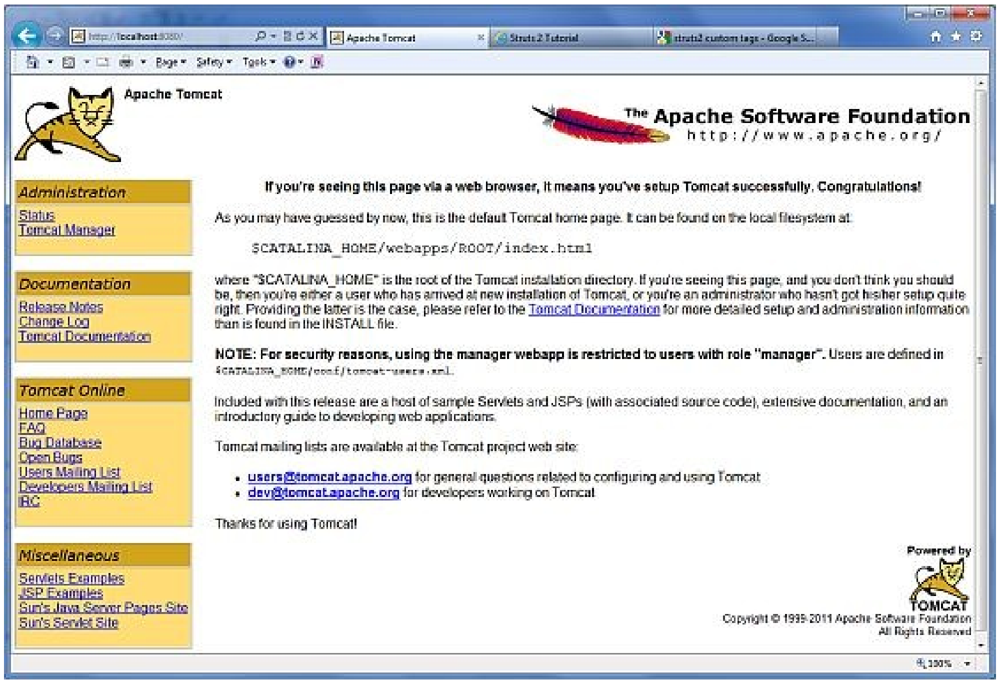
Tomcat can be started by executing the following commands on the Windows machine –

%CATALINA_HOME%\bin\startup.bat
or
C:\apache-tomcat-5.5.29\bin\startup.bat

Tomcat can be started by executing the following commands on the Unix (Solaris, Linux, etc.) machine –

$CATALINA_HOME/bin/startup.sh
or
/usr/local/apache-tomcat-5.5.29/bin/startup.sh

After a successful startup, the default web-applications included with Tomcat will be available by visiting **http://localhost:8080/**.
Upon execution, you will receive the following output –



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- **users@tomcat.apache.org** for general questions related to configuring and using Tomcat
- **dev@tomcat.apache.org** for developers working on Tomcat

Thanks for using Tomcat!

Copyright © 1999-2011 Apache Software Foundation
All Rights Reserved

Tomcat can be stopped by executing the following commands on the Windows machine –

%CATALINA_HOME%\bin\shutdown
or
C:\apache-tomcat-5.5.29\bin\shutdown

Tomcat can be stopped by executing the following commands on UNIX (Solaris, Linux, etc.) machine −

```
$CATALINA_HOME/bin/shutdown.sh
or
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

2.2. JSP Life Cycle:

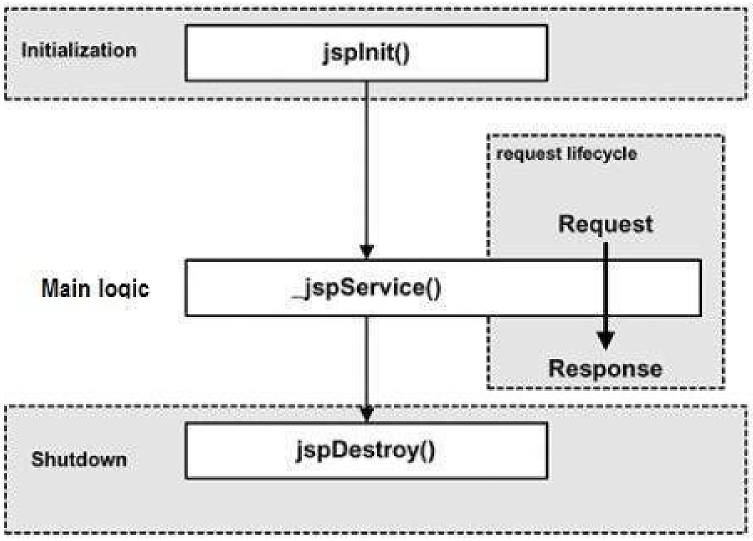
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

2.2.1. Paths Followed By JSP:

The following are the paths followed by a JSP –

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



2.2.2. JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

2.2.3. JSP Initialization:

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method –

```
public void jspInit(){
  // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

2.2.4. JSP Execution:

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET**, **POST**, **DELETE**, etc.

2.2.5. JSP Cleanup:

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form –

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

2.3. Creating a simple JSP Page:

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

It will print **10** on the browser.

2.4. How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

Lecture 3: JSP scriptlet tag, Variable (field) declaration, Methods in JSP, JSP Comments, Decision-Making Statements, Loop Statements, Operators, Literals

3.1. JSP scriptlet tag (Scripting elements):

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what the scripting elements are first.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

3.1.1. JSP scriptlet tag:

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<%  java source code %>
```

Example

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

3.1.2.JSP expression tag:

The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax

```
<%=  statement %>
```

Example

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
```

```
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

3.1.3.JSP Declaration Tag:

The JSP declaration tag is used to declare fields and methods.

The code written inside the jsp declaration tag is placed outside the service () method of auto generated servlet.

So it doesn't get memory at each request.

Syntax

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

3.1.3.1. Difference between JSP scriptlet tag and declaration tag:

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
```

```
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

3.2. JSP Comments:

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.
Following is the syntax of the JSP comments –

```
<%-- This is JSP comment --%>
```

Following example shows the JSP Comments –

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</body>
</html>
```

The above code will generate the following result −

A Test of Comments

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary –

Sl. No.	Syntax & Purpose
1	<%-- comment --%> A JSP comment. Ignored by the JSP engine.
2	<!-- comment --> An HTML comment. Ignored by the browser.
3	<\% Represents static <% literal.
4	%\> Represents static %> literal.
5	\' A single quote in an attribute that uses single quotes.
6	\" A double quote in an attribute that uses double quotes.

3.3. Control-Flow Statements:

You can use all the APIs and building blocks of Java in your JSP programming including decision-making statements, loops, etc.

3.3.1. Decision-Making Statements:

The if...else block starts out like an ordinary scriptlet, but the scriptlet is closed at each line with HTML text included between the scriptlet tags.

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>

<body>
<% if (day == 1 || day == 7) { %>
<p> Today is weekend</p>
<% } else { %>
<p> Today is not weekend</p>
<% } %>
</body>
```

```
</html>
```

The above code will generate the following result –

```
Today is not weekend
```

Now look at the following switch...case block which has been written a bit differentlty using out.println() and inside scriptlets –

```
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
      switch(day) {
        case 0:
          out.println("It\'s Sunday.");
          break;
        case 1:
          out.println("It\'s Monday.");
          break;
        case 2:
          out.println("It\'s Tuesday.");
          break;
        case 3:
          out.println("It\'s Wednesday.");
          break;
        case 4:
          out.println("It\'s Thursday.");
          break;
        case 5:
          out.println("It\'s Friday.");
          break;
        default:
          out.println("It's Saturday.");
      }
    %>
</body>
</html>
```

The above code will generate the following result –

It's Wednesday.

3.3.2. Loop Statements:

You can also use three basic types of looping blocks in Java: for, while, and do…while blocks in your JSP programming.

Let us look at the following for loop example –

```
<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body>
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
<font color = "green" size = "<%= fontSize %>">
        JSP Tutorial
</font><br />
<%}%>
</body>
</html>
```

The above code will generate the following result –

```
JSP Tutorial


JSP Tutorial



JSP Tutorial
```

Above example can be written using the while loop as follows –

```
<%! int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>

<body>
<%while ( fontSize <= 3){ %>
<font color = "green" size = "<%= fontSize %>">
        JSP Tutorial
</font><br />
<%fontSize++;%>
<%}%>
</body>
</html>
```

The above code will generate the following result −

```
JSP Tutorial

JSP Tutorial

JSP Tutorial
```

3.4. JSP Operators:

JSP supports all the logical and arithmetic operators supported by Java. Following table lists out all the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>>>>>><<	Left to right
Relational	>>= <<=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

3.5. JSP Literals:

The JSP expression language defines the following literals –

- Boolean − true and false
- Integer − as in Java
- Floating point − as in Java
- String − with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\
- Null − null

Lecture 4: JSP implicit objects (request and response)

4.1. JSP - Implicit Objects:

These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

Following table lists out the nine Implicit Objects that JSP supports −

Sl. No.	Object & Description
1	**request** This is the **HttpServletRequest** object associated with the request.
2	**response** This is the **HttpServletResponse** object associated with the response to the client.
3	**out** This is the **PrintWriter** object used to send output to the client.
4	**session** This is the **HttpSession** object associated with the request.
5	**application** This is the **ServletContext** object associated with the application context.
6	**config** This is the **ServletConfig** object associated with the page.

7	**pageContext** This encapsulates use of server-specific features like higher performance **JspWriters**.
8	**page** This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
9	**Exception** The **Exception** object allows the exception data to be accessed by designated JSP.

4.1.1. The request object:

The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example

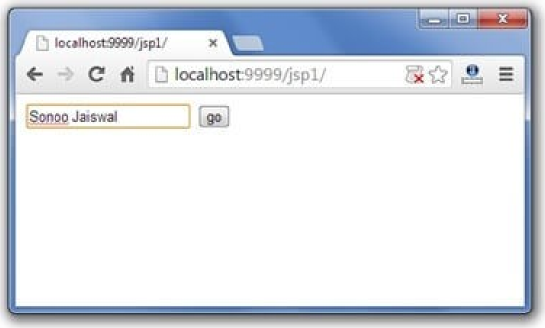
index.html

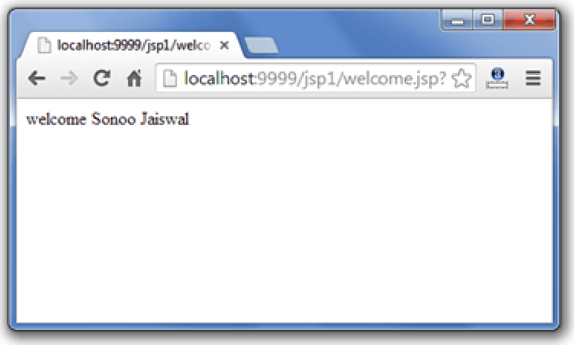
```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

Output





welcome Sonoo Jaiswal

4.1.2. The response object:

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example

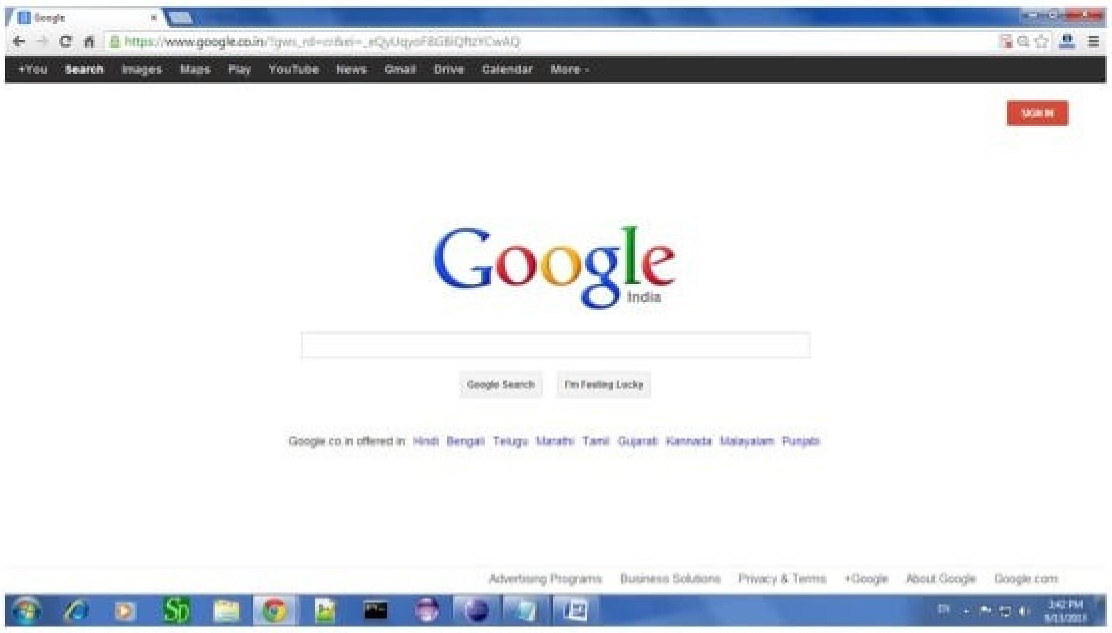
index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

Output



Lecture 5: JSP directive (Taglib and Include), JSP Action tags (Forward & Include)

5.1. JSP directives:

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive

- taglib directive

Syntax

<%@ directive attribute="value" %>

5.1.1. JSP page directive:

The page directive defines attributes that apply to an entire JSP page.

Syntax

<%@ page attribute="value" %>

5.1.1.1. Attributes of JSP page directive:

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

5.1.2. JSP Include Directive:

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage

Code Reusability

Syntax

```
<%@ include file="resourceName" %>
```

Example

In this example, we are including the content of the header.html file. To run this example you must create a header.html file.

```
<html>
<body>
<%@ include file="header.html" %>
 Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Note: The include directive includes the original content, so the actual page size grows at runtime.

5.1.3. JSP Taglib directive:

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

Syntax

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Example

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

5.2. JSP Action Tags:

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The JSP action tags are given below.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for JavaBean development.

5.2.1. jsp:forward action tag:

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Syntax of jsp:forward action tag with parameter

```
<jsp:forward page="relativeURL | <%= expression %>">

<jsp:param name="parametername" value="parametervalue | <%=expression%>" />

</jsp:forward>
```

Example of jsp:forward action tag without parameter

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>
```

```
<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Example of jsp:forward action tag with parameter

In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="javatpoint.com" />
</jsp:forward>

</body>
</html>
```

printdate.jsp

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```

5.2.2. jsp:forward action tag:

The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet.

The jsp include action tag includes the resource at request time so it is better for dynamic pages because there might be changes in future.

The jsp:include tag can be used to include static as well as dynamic pages.

Advantage of jsp:include action tag

Code reusability: We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Difference between jsp include directive and include action

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

Syntax of jsp:include action tag without parameter

```
<jsp:include page="relativeURL | <%= expression %>" />
```

Syntax of jsp:include action tag with parameter

```
<jsp:include page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:include>
```

Example of jsp:include action tag without parameter

In this example, index.jsp file includes the content of the printdate.jsp file.

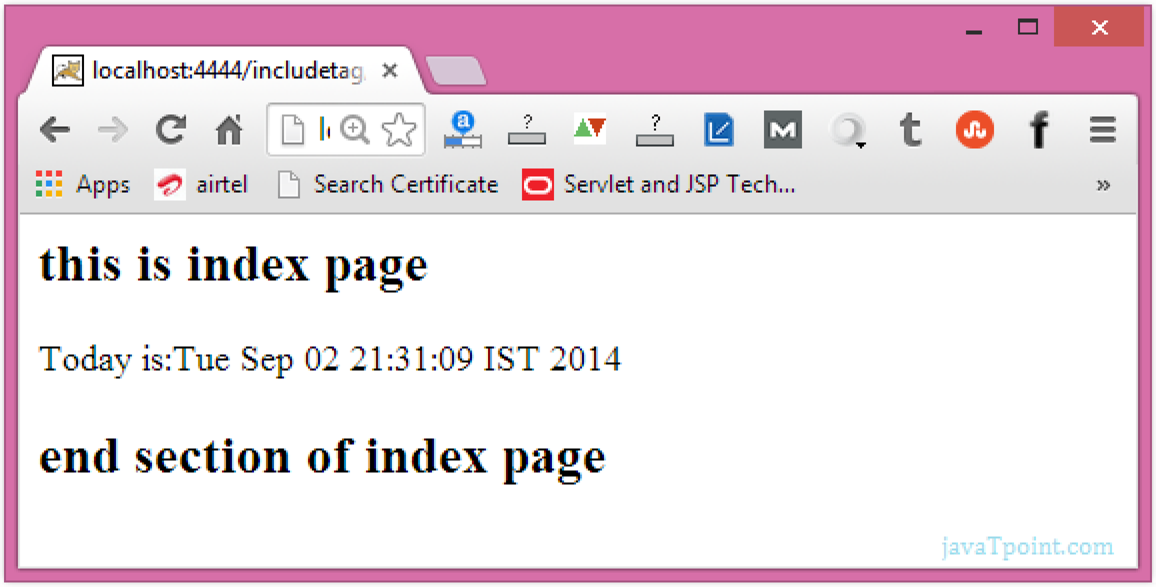
File: index.jsp

```
<h2>this is index page</h2>

<jsp:include page="printdate.jsp" />

<h2>end section of index page</h2>
```

File: printdate.jsp

```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
```

Output



```
this is index page

Today is:Tue Sep 02 21:31:09 IST 2014

end section of index page
```

Lecture 6: Javabean- inserting javabean in JSP

6.1. JSP – JavaBeans:

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes –

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the Serializable interface.

- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

6.1.1. JavaBeans Properties:

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class –

Sl. No.	Method & Description
1	get**PropertyName**() For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

A read-only attribute will have only a getPropertyName() method, and a write-only attribute will have only a setPropertyName() method.

6.1.2. JavaBeans Example:

Consider a student class with few properties –

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable {
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

6.1.3. Accessing JavaBeans:

The useBean action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows –

```
<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>
```

Here values for the scope attribute can be a **page, request, session** or **application based** on your requirement. The value of the id attribute may be any value as a long as it is a unique name among other **useBean declarations** in the same JSP.

Following example shows how to use the useBean action –

```
<html>
<head>
<title>useBean Example</title>
</head>

<body>
<jsp:useBean id = "date" class = "java.util.Date" />
<p>The date/time is <%= date %>
</body>
</html>
```

You will receive the following result − −

```
The date/time is Thu Sep 30 11:18:11 GST 2010
```

6.1.4. Accessing JavaBeans Properties:

Along with **<jsp:useBean...>** action, you can use the **<jsp:getProperty/>** action to access the get methods and the **<jsp:setProperty/>** action to access the set methods. Here is the full syntax –

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">
<jsp:setProperty name = "bean's id" property = "property name"
    value = "value"/>
<jsp:getProperty name = "bean's id" property = "property name"/>
  ...........
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the **get** or the **set**methods that should be invoked.

Following example shows how to access the data using the above syntax –

```
<html>
<head>
<title>get and set properties Example</title>
</head>

<body>
<jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">
<jsp:setProperty name = "students" property = "firstName" value = "Zara"/>
<jsp:setProperty name = "students" property = "lastName" value = "Ali"/>
<jsp:setProperty name = "students" property = "age" value = "10"/>
</jsp:useBean>

<p>Student First Name:
<jsp:getProperty name = "students" property = "firstName"/>
</p>

<p>Student Last Name:
<jsp:getProperty name = "students" property = "lastName"/>
</p>

<p>Student Age:
<jsp:getProperty name = "students" property = "age"/>
</p>
```

```
</body>
</html>
```

Let us make the **StudentsBean.class** available in CLASSPATH. Access the above JSP. The following result will be displayed –

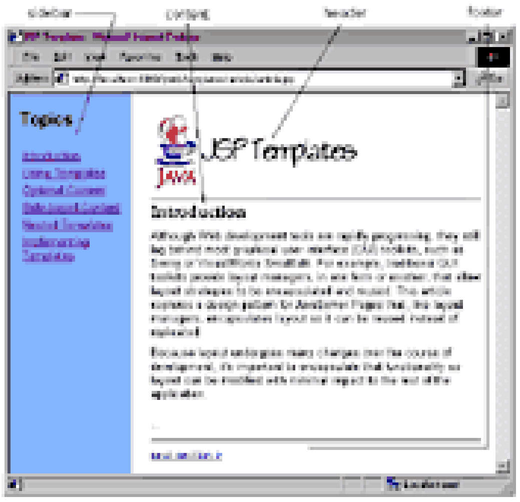
```
Student First Name: Zara


Student Last Name: Ali


Student Age: 10
```

Lecture 7: Understanding the layout of JSP

Although Web development tools are rapidly progressing, they still lag behind most graphical user interface (GUI) toolkits such as Swing or VisualWorks Smalltalk. For example, traditional GUI toolkits provide layout managers, in one form or another, that allow layout algorithms to be encapsulated and reused. Here a template mechanism for JavaServer Pages (JSP) will be shown, that, like layout managers, encapsulates layout so it can be reused instead of replicated.

Because layout undergoes many changes over the course of development, it's important to encapsulate that functionality so it can be modified with minimal impact to the rest of the application. In fact, layout managers demonstrate an example of one of the tenets of object-oriented design: encapsulate the concept that varies, which is also a fundamental theme for many design patterns.

JSP does not provide direct support for encapsulating layout, so Webpages with identical formats usually replicate layout code; for example, the following figure shows a Webpage containing header, footer, sidebar, and main content sections.



Webpage layout

The layout of the page shown in the above figure is implemented with HTML table tags:

Example 1. Including content

```
<html><head><title>JSP Templates</title></head>
<body background='graphics/background.jpg'>
<table>
<tr valign='top'><td><%@include file='sidebar.html'%></td>
<td><table>
<tr><td><%@include file='header.html'%></td></tr>
<tr><td><%@include file='introduction.html'%></td></tr>
<tr><td><%@include file='footer.html'%></td></tr>
</table>
</td>
</tr>
</table>
</body></html>
```

In the example listed above, content is included with the JSP include directive, which allows the content of the page to vary -- by changing the included files -- without modifying the page itself. However, because layout is hard coded, layout changes require modifications to the page. If a Website has multiple pages with identical formats, which is common, even simple layout changes require modifications to all of the pages.

To minimize the impact of layout changes, we need a mechanism for including layout in addition to content; that way, both layout and content can vary without modifying files that use them. That mechanism is JSP templates.

7.1. Using templates:

Templates are JSP files that include parameterized content. The templates discussed in this article are implemented with a set of custom tags: template:get, template:put, and template:insert. The template:get tag accesses parameterized content, as illustrated in Example 2.a, which produces Webpages with the format shown in the above figure.

Example 2.a. A template

```
<%@ taglib uri='/WEB-INF/tlds/template.tld' prefix='template' %>
<html><head><title><template:get name='title'/></title></head>
<body background='graphics/background.jpg'>
<table>
<tr valign='top'><td><template:get name='sidebar'/></td>
<td><table>
<tr><td><template:get name='header'/></td></tr>
<tr><td><template:get name='content'/></td></tr>
<tr><td><template:get name='footer'/></td></tr>
</table>
</td>
</tr>
</table>
</body></html>
```

Example 2.a is nearly identical to Example 1, except we use template:get instead of the include directive. Let's examine how template:get works.

template:get retrieves a Java bean with the specified name from request scope. The bean contains the URI (Uniform Resource Identifier) of a Web component that's included by template:get. For example, in the template listed in Example 2.a, template:get obtains a URI -- header.html -- from a bean named header in request scope. Subsequently, template:get includes header.html.

template:put puts the beans in request scope that are subsequently retrieved by template:get. The template is included with template:insert. Example 2.b illustrates the use of the put and insert tags:

Example 2.b. Using the template from Example 2.a

```
<%@ taglib uri='/WEB-INF/tlds/template.tld' prefix='template' %>
<template:insert template='/articleTemplate.jsp'>
<template:put name='title' content='Templates' direct='true'/>
<template:put name='header' content='/header.html' />
<template:put name='sidebar' content='/sidebar.jsp' />
<template:put name='content' content='/introduction.html'/>
<template:put name='footer' content='/footer.html' />
</template:insert>
```

The insert start tag specifies the template to be included, in this case the template listed in Example 2.a. Each put tag stores a bean in request scope and the insert end tag includes the template. The template subsequently accesses the beans as described above.

A direct attribute can be specified for template:put; if direct is set to true, the content associated with the tag is not included by template:get, but is printed directly to the implicit out variable. In Example 2.b, for example, the title content -- JSP Templates -- is used for the window title.

Websites containing multiple pages with identical formats have one template, such as the one listed in Example 2.a, and many JSP pages, such as Example 2.b, that use the template. If the format is modified, changes are restricted to the template.

Another benefit of templates and including content in general is modular design. For example, the JSP file listed in Example 2.b ultimately includes header.html, listed in Example 2.c.

Example 2.c. header.html

```
<table>
<tr>
<td><img src='graphics/java.jpg'/></td>
<td><img src='graphics/templates.jpg'/></td>
</tr>
</table><hr>
```

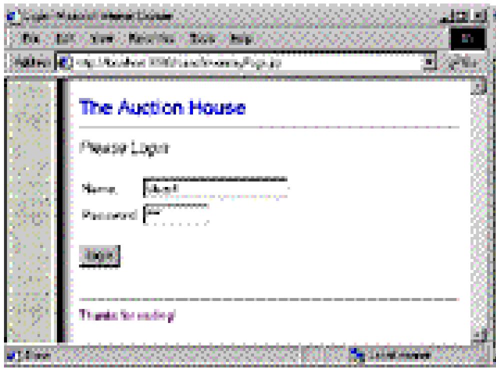
Because header.html is included content, it does not have to be replicated among pages that display a header. Also, although header.html is an HTML file, it does not contain the usual preamble of HTML tags such as <html> or <body> because those tags are defined by the template. That is, because the template includes header.html, those tags should not be repeated in header.html.

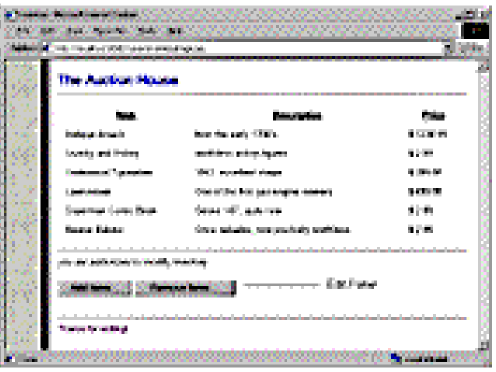
Note: JSP provides two ways to include content: statically, with the include directive, and dynamically, with the include action. The include directive includes the source of the target page at compile time and is equivalent to C's #include or Java's import. The include action includes the target's response generated at runtime.

Like the JSP include action, templates include content dynamically. So, although the JSP pages in Example 1 and Example 2.b are functionally identical, the former statically includes content, whereas the latter dynamically includes it.

7.2. Optional content:

All template content is optional, which makes a single template useful to more Webpages. For example, the following two figures show two pages -- login and inventory -- that use the same template. Both pages have a header, footer, and main content. The inventory page has an edit panel (which the login page lacks) for making inventory changes.





A login form An inventory page

Below, you'll find the template shared by the login and inventory pages:

```
<%@ taglib uri='template.tld' prefix='template' %>
...
<table width='670'>
<tr><td width='60'></td>
<td><template:get name='header'/></td></tr>
<tr><td width='60'></td>
<td><template:get name='main-content'/></td></tr>
<tr><td width='60'></td>
<td><template:get name='editPanel'/></td></tr>
<tr><td width='60'></td>
<td><template:get name='footer'/></td></tr>
</table>
...
```

The inventory page uses the template listed above and specifies content for the edit panel:

```
<%@ taglib uri='template.tld' prefix='template' %>
<%@ taglib uri='security.tld' prefix='security' %>
<template:insert template='/template.jsp'>
   ...
<template:put name='editPanel'
                    content='/editPanelContent.jsp'/>
   ...
</template:insert>
```

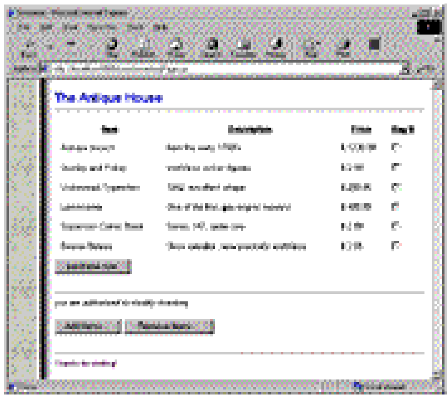
In contrast, the login page does not specify content for the edit panel:

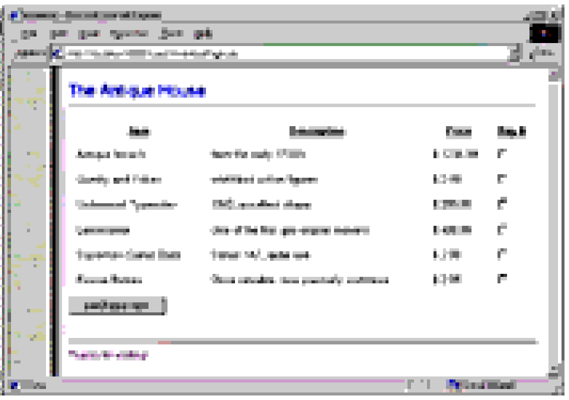
```
<%@ taglib uri='template.tld' prefix='template' %>
<template:insert template='/template.jsp'>
<template:put name='title' content='Login' direct='true'/>
<template:put name='header' content='/header.jsp'/>
<template:put name='main-content'
                 content='/login.jsp'/>
<template:put name='footer' content='/footer.jsp'/>
</template:insert>
```

Because the login page does not specify content for the edit panel, it's not included.

7.3. Role-based content:

Web applications often discriminate content based on a user's role. For example, the same JSP template, which includes the edit panel only when the user's role is curator, produces the two pages shown in the following two figures.





Inventory page for curators   Inventory page for other users

The template used in the above figures uses template:get's role attribute:

```
<%@ taglib uri='template.tld' prefix='template' %>
...
<table>
  ...
<td><template:get name='editPanel' role='curator'/></td></tr>
  ...
</table>
...
```

The get tag includes content only if the user's role matches the role attribute. Let's look at how the tag handler for template:get uses the role attribute:

```
public class GetTag extends TagSupport {
  private String name = null, role = null;
  ...
  public void setRole(String role) { this.role = role; }
  ...
  public int doStartTag() throws JspException {
    ...
    if(param != null) {
      if(roleIsValid()) {
        // include or print content ...
      }
    }
  ...
  }
  private boolean roleIsValid() {
    return role == null || // valid if role isn't set
      ((javax.servlet.http.HttpServletRequest)
       pageContext.getRequest()).isUserInRole(role);
  }
}
```

7.4. Implementing templates:

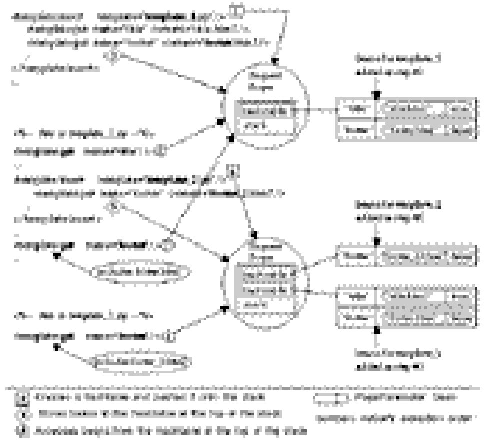
The templates discussed in this article are implemented with three custom tags:

- template:insert
- template:put
- template:get

The insert tag includes a template, but before it does, put tags store information -- a name, URI, and Boolean value specifying whether content should be included or printed directly -- about the content the template includes. template:get, which includes (or prints) the specified content, subsequently accesses the information.

template:put stores beans in request scope but not directly because if two templates use the same content names, a nested template could overwrite the enclosing template's content.

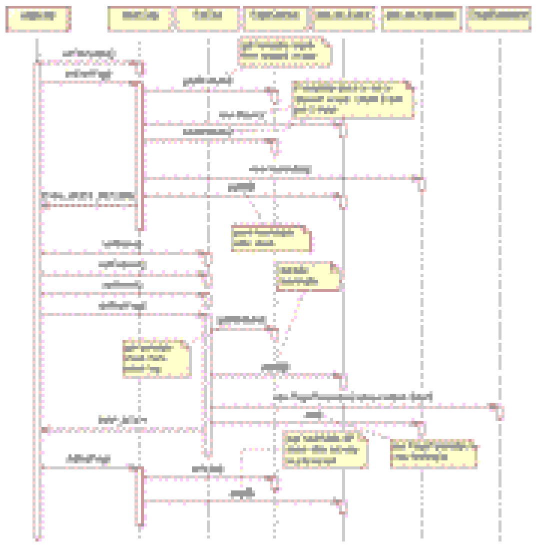
To ensure that each template has access only to its own information, template:insert maintains a stack of hashtables. Each insert start tag creates a hashtable and pushes it on the stack. The enclosed put tags create beans and store them in the newly created hashtable. Subsequently, get tags in the included template access the beans in the hashtable. The following figure shows how the stack is maintained for nested templates.


Storing template parameters in request scope

Each template in the above figure accesses the correct footer; footer.html for template_1.jsp and footer_2.html for template_2.jsp. If the beans were stored directly in request scope, step 5 in Figure 4 would overwrite the footer bean specified in step 2.

7.4.1. Template tag implementations:

Now, we will examine the implementation of the three template tags: insert, put, and get. We begin with sequence diagrams, starting with the following figure. It illustrates the sequence of events for the insert and put tags when a template is used.


Sequence diagrams for the put and insert tags

If a template stack does not already exist, the insert start tag creates one and places it in request scope. A hashtable is subsequently created and pushed on the stack.

Each put start tag creates a PageParameter bean, stored in the hashtable created by the enclosing insert tag.

The insert end tag includes the template. The template uses get tags to access the beans created by put tags. After the template is processed, the hashtable created by the insert start tag is popped off the stack.

The following figure shows the sequence diagram for template:get.

Sequence diagrams for the get tag

7.4.2. Template tag listings:

Tag handler implementations for the template tags prove straightforward. Example 3.a lists the InsertTag class -- the tag handler for template:insert.

Example 3.a. InsertTag.java

```
package tags.templates;
import java.util.Hashtable;
import java.util.Stack;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;
public class InsertTag extends TagSupport {
  private String template;
  private Stack stack;
  // setter method for template attribute
  public void setTemplate(String template) {
    this.template = template;
  }
  public int doStartTag() throws JspException {
    stack = getStack(); // obtain a reference to the template stack
    stack.push(new Hashtable()); // push new hashtable onto stack
    return EVAL_BODY_INCLUDE;  // pass tag body through unchanged
  }
  public int doEndTag() throws JspException {
    try {
      pageContext.include(template); // include template
    }
    catch(Exception ex) { // IOException or ServletException
      throw new JspException(ex.getMessage()); // recast exception
    }
    stack.pop(); // pop hashtable off stack
    return EVAL_PAGE; // evaluate the rest of the page after the tag
  }
  // tag handlers should always implement release() because
  // handlers can be reused by the JSP container
  public void release() {
    template = null;
    stack = null;
  }
  public Stack getStack() {
    // try to get stack from request scope
    Stack s = (Stack)pageContext.getAttribute(
                "template-stack",
                PageContext.REQUEST_SCOPE);
    // if the stack's not present, create a new one and
```

```
      // put it into request scope
      if(s == null) {
        s = new Stack();
        pageContext.setAttribute("template-stack", s,
                  PageContext.REQUEST_SCOPE);
      }
      return s;
    }
  }
```

Example 3.b lists the PutTag class, the tag handler for template:put:

Example 3.b. PutTag.java

```
package tags.templates;
import java.util.Hashtable;
import java.util.Stack;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
import beans.templates.PageParameter;
public class PutTag extends TagSupport {
  private String name, content, direct="false";
  // setter methods for Put tag attributes
  public void setName(String s) { name = s; }
  public void setContent(String s) {content = s; }
  public void setDirect(String s) { direct = s; }
  public int doStartTag() throws JspException {
    // obtain a reference to enclosing insert tag
    InsertTag parent = (InsertTag)getAncestor(
                "tags.templates.InsertTag");
    // put tags must be enclosed in an insert tag
    if(parent == null)
      throw new JspException("PutTag.doStartTag(): " +
                "No InsertTag ancestor");
    // get template stack from insert tag
    Stack template_stack = parent.getStack();
    // template stack should never be null
    if(template_stack == null)
      throw new JspException("PutTag: no template stack");
    // peek at hashtable on the stack
```

```
      Hashtable params = (Hashtable)template_stack.peek();
      // hashtable should never be null either
      if(params == null)
         throw new JspException("PutTag: no hashtable");
      // put a new PageParameter in the hashtable
      params.put(name, new PageParameter(content, direct));
      return SKIP_BODY; // not interested in tag body, if present
   }
   // tag handlers should always implement release() because
   // handlers can be reused by the JSP container
   public void release() {
      name = content = direct = null;
   }
   // convenience method for finding ancestor names with
   // a specific class name
   private TagSupport getAncestor(String className)
                              throws JspException {
      Class klass = null; // can't name variable "class"
      try {
         klass = Class.forName(className);
      }
      catch(ClassNotFoundException ex) {
         throw new JspException(ex.getMessage());
      }
      return (TagSupport)findAncestorWithClass(this, klass);
   }
}
```

PutTag.doStartTag creates a PageParameter bean -- listed in Example 3.c. -- subsequently stored in request scope.

Example 3.c. PageParameter.java

```
package beans.templates;
public class PageParameter {
   private String content, direct;
   public void setContent(String s) {content = s; }
   public void setDirect(String s) { direct = s; }
```

```
  public String getContent() { return content;}
  public boolean isDirect() { return Boolean.valueOf(direct).booleanValue(); }
  public PageParameter(String content, String direct) {
    this.content = content;
    this.direct = direct;
  }
}
```

PageParameter serves as a simple placeholder for the content and direct attributes set in the template:put tag. We see the GetTag class, the tag handler for template:get, in Example 3.d:

Example 3.d. GetTag.java

```
package tags.templates;
import java.util.Hashtable;
import java.util.Stack;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;
import beans.templates.PageParameter;
public class GetTag extends TagSupport {
  private String name;
  // setter method for name attribute
  public void setName(String name) {
    this.name = name;
  }
  public int doStartTag() throws JspException {
    // obtain reference to template stack
    Stack stack = (Stack)pageContext.getAttribute(
          "template-stack", PageContext.REQUEST_SCOPE);
    // stack should not be null
    if(stack == null)
      throw new JspException("GetTag.doStartTag(): " +
                      "NO STACK");
    // peek at hashtable
    Hashtable params = (Hashtable)stack.peek();
    // hashtable should not be null
    if(params == null)
      throw new JspException("GetTag.doStartTag(): " +
                      "NO HASHTABLE");
```

```
        // get page parameter from hashtable
        PageParameter param = (PageParameter)params.get(name);
        if(param != null) {
          String content = param.getContent();
          if(param.isDirect()) {
            // print content if direct attribute is true
            try {
              pageContext.getOut().print(content);
            }
            catch(java.io.IOException ex) {
              throw new JspException(ex.getMessage());
            }
          }
          else {
            // include content if direct attribute is false
            try {
              pageContext.getOut().flush();
              pageContext.include(content);
            }
            catch(Exception ex) {
              throw new JspException(ex.getMessage());
            }
          }
        }
        return SKIP_BODY; // not interested in tag body, if present
      }
      // tag handlers should always implement release() because
      // handlers can be reused by the JSP container
      public void release() {
        name = null;
      }
    }
```

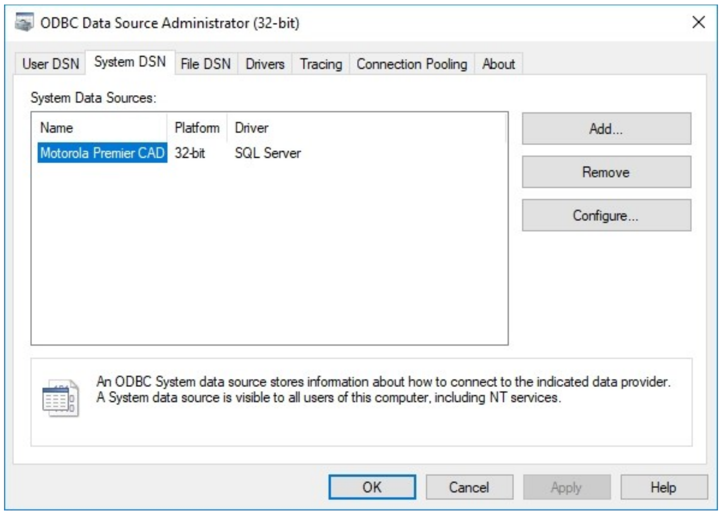
GetTag.doStartTag retrieves the page parameter bean from request scope and obtains the content and direct properties from the bean. Subsequently, the content is either included or printed, depending on the value of the direct property.

Lecture 8: Creating ODBC data source name, Introduction to JDBC, prepared statement and callable statement

8.1. Creating ODBC data source name (for 64-bit computers):

An ODBC (Open Database Connectivity) driver is a software standard that lets different applications share data. There are ODBC drivers for Microsoft® SQL Server™, Microsoft Access, Oracle® and many others. The ODBC driver lets you define information for connecting to databases in SQL Server, including the directory source name (DSN), directory, driver of the database, and the ID and password of the user.

1. In Windows Explorer, navigate to C:\Windows\sysWOW64\.
2. Double-click odbcad32.exe.
   The ODBC Data Source Administrator dialog box appears.
3. Click the System DSN tab.

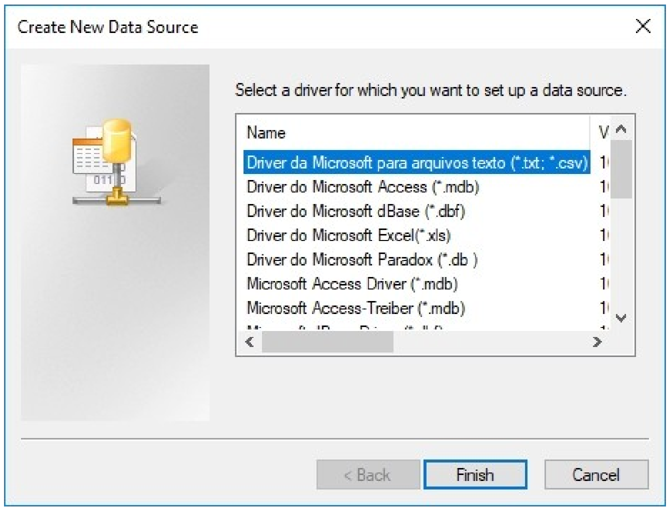


System Data Sources:

Name	Platform	Driver
Motorola Premier CAD	32-bit	SQL Server

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users of this computer, including NT services.

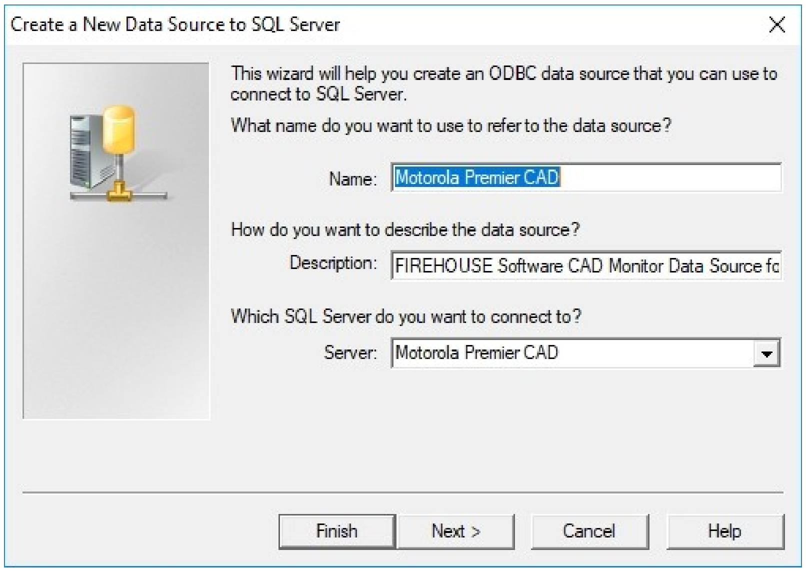
4. Click Add.
   The Create New Data Source dialog box appears.



Select a driver for which you want to set up a data source.

Name	V
Driver da Microsoft para arquivos texto (*.txt; *.csv)	1
Driver do Microsoft Access (*.mdb)	1
Driver do Microsoft dBase (*.dbf)	1
Driver do Microsoft Excel(*.xls)	1
Driver do Microsoft Paradox (*.db)	1
Microsoft Access Driver (*.mdb)	1
Microsoft Access-Treiber (*.mdb)	1

5. Scroll down through the list and select SQL Server, and then click Finish.
   The Create a New Data Source to SQL Server wizard appears.



This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: Motorola Premier CAD

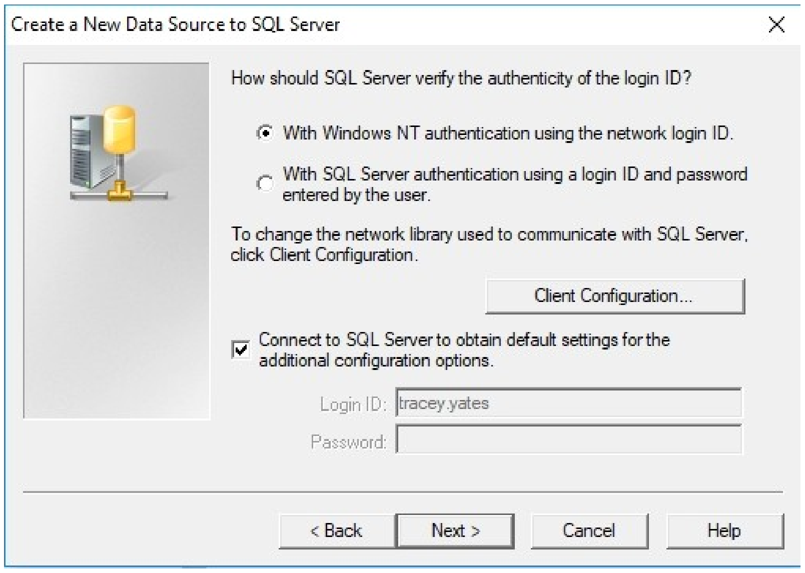
How do you want to describe the data source?

Description: FIREHOUSE Software CAD Monitor Data Source fc

Which SQL Server do you want to connect to?

Server: Motorola Premier CAD

6. In Name and Description, type the name and a description for the ODBC data source you are creating.
   Example: ODBC data source
7. From Server, select the name of the SQL server where your database software exists.
   Caution: Include the full name of the SQL instance where the database software exists.
   Example: SERVER1\SQLEXPRESS
8. Click Next >.
   The next screen of the Create a New Data Source to SQL Server wizard appears.



How should SQL Server verify the authenticity of the login ID?

● With Windows NT authentication using the network login ID.

○ With SQL Server authentication using a login ID and password entered by the user.

To change the network library used to communicate with SQL Server, click Client Configuration.

✓ Connect to SQL Server to obtain default settings for the additional configuration options.

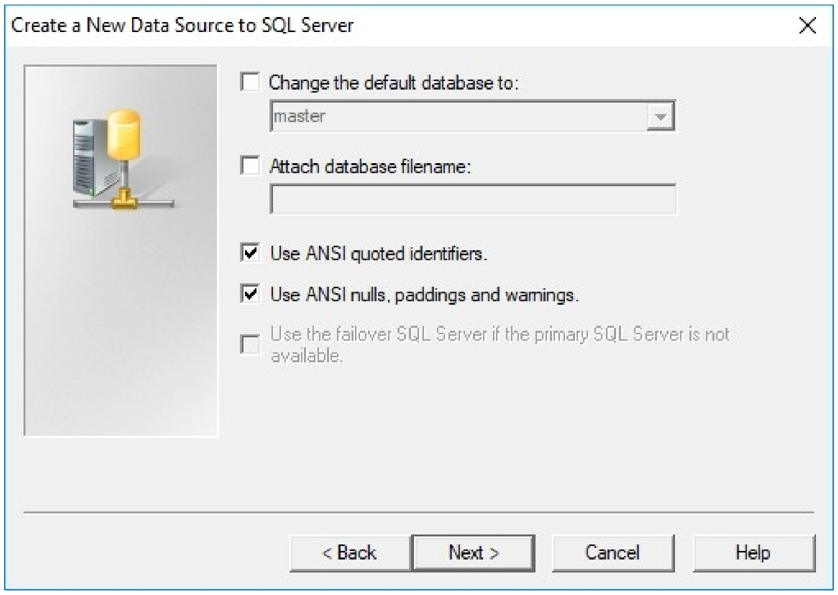
Login ID: tracey.yates

Password:

9. Define which SQL Server login credentials to use in one of the following ways:

a) The current or active Windows user account
b) Unique SQL Server login credentials

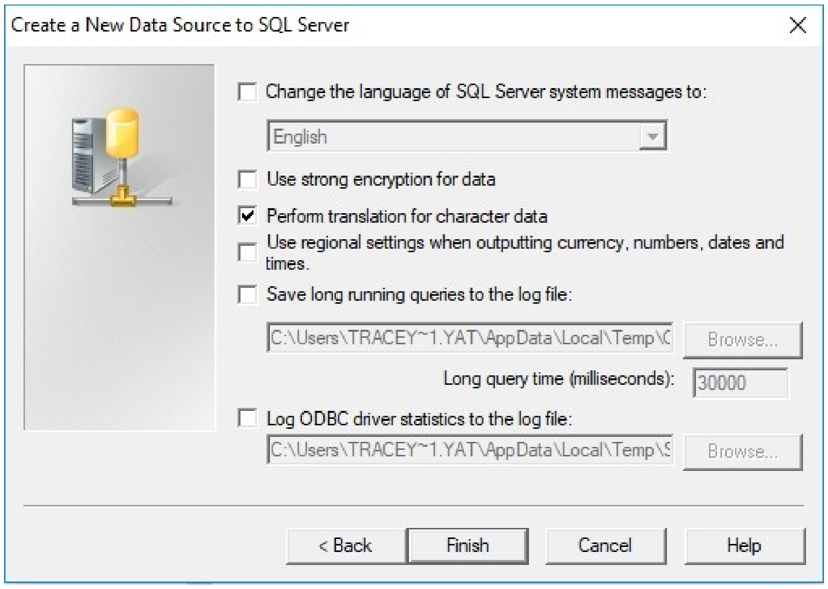
10. Click Next >.
    The next screen of the Create a New Data Source to SQL Server wizard appears.



Create a New Data Source to SQL Server

- [ ] Change the default database to:
  master
- [ ] Attach database filename:
- [x] Use ANSI quoted identifiers.
- [x] Use ANSI nulls, paddings and warnings.
- [ ] Use the failover SQL Server if the primary SQL Server is not available.

11. Select Change the default database to, and then choose the name of your database from the menu below it.
    Warning: Do not change any of the other settings on this screen of the wizard.
12. Click Next >.
    The next screen of the Create a New Data Source to SQL Server wizard appears.

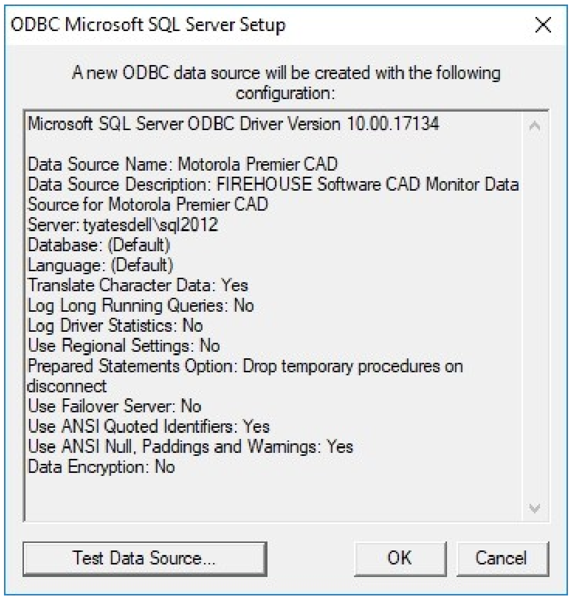


Create a New Data Source to SQL Server

- [ ] Change the language of SQL Server system messages to:
  English
- [ ] Use strong encryption for data
- [x] Perform translation for character data
- [ ] Use regional settings when outputting currency, numbers, dates and times.
- [ ] Save long running queries to the log file:
  C:\Users\TRACEY~1.YAT\AppData\Local\Temp\C
  Long query time (milliseconds): 30000
- [ ] Log ODBC driver statistics to the log file:
  C:\Users\TRACEY~1.YAT\AppData\Local\Temp\S

Warning: Do not change any settings on this screen of the wizard.

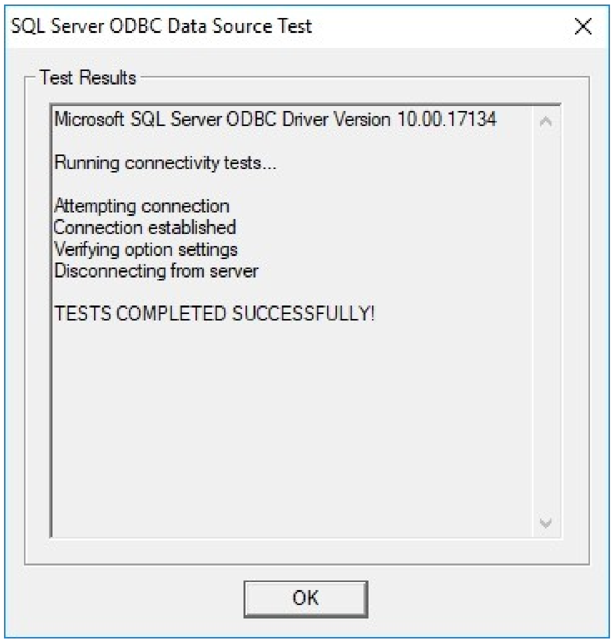
13. Click Finish.
    The ODBC data source is created, and the ODBC Microsoft SQL Server Setup dialog box appears.



A new ODBC data source will be created with the following configuration:

```
Microsoft SQL Server ODBC Driver Version 10.00.17134

Data Source Name: Motorola Premier CAD
Data Source Description: FIREHOUSE Software CAD Monitor Data
Source for Motorola Premier CAD
Server: tyatesdell\sql2012
Database: (Default)
Language: (Default)
Translate Character Data: Yes
Log Long Running Queries: No
Log Driver Statistics: No
Use Regional Settings: No
Prepared Statements Option: Drop temporary procedures on
disconnect
Use Failover Server: No
Use ANSI Quoted Identifiers: Yes
Use ANSI Null, Paddings and Warnings: Yes
Data Encryption: No
```

14. Click Test Data Source.
    The SQL Server ODBC Data Source Test dialog box appears.



```
Microsoft SQL Server ODBC Driver Version 10.00.17134

Running connectivity tests...

Attempting connection
Connection established
Verifying option settings
Disconnecting from server

TESTS COMPLETED SUCCESSFULLY!
```

15. Click OK.
16. In the ODBC Microsoft SQL Server Setup dialog box, click OK.
17. In the ODBC Data source Administrator dialog box, click OK.

8.2. What is JDBC?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as –

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs)

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

8.2.1. JDBC Architecture:

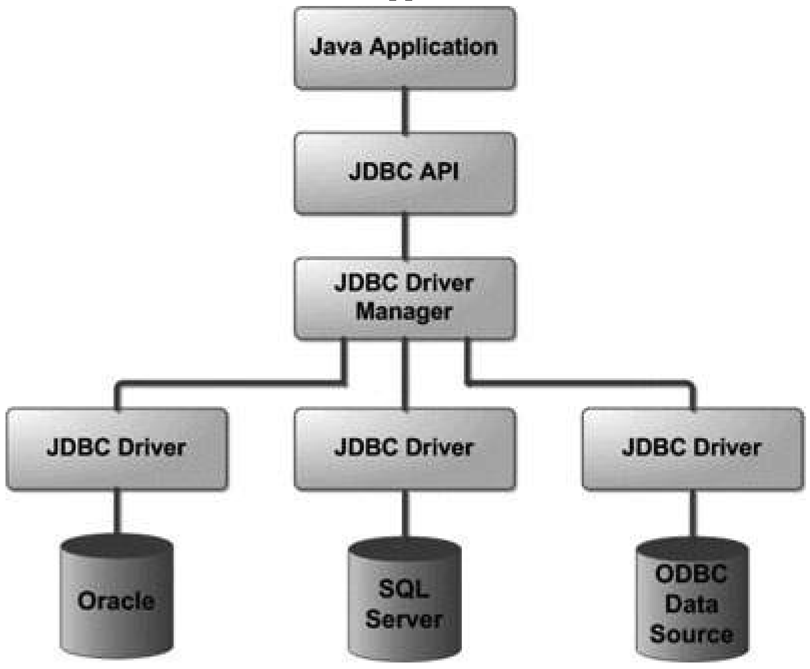
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –

- JDBC API: This provides the application-to-JDBC Manager connection.
- JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application –



8.2.2. Common JDBC Components:

The JDBC API provides the following interfaces and classes –

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

8.2.3. JDBC - Statements, PreparedStatement and CallableStatement:

The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

They also define methods that help bridge data type differences between Java and SQL data types used in a database.

The following table provides a summary of each interface's purpose to decide on the interface to use.

Interfaces	Recommended Use
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

8.2.3.1. The Statement Objects:

Creating Statement Object

Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method, as in the following example −

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
```

```
finally {
  . . .
}
```

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

- **boolean execute (String SQL):** Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL):** Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL):** Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

Closing Statement Object

Just as you close a Connection object to save database resources, for the same reason you should also close the Statement object.

A simple call to the close() method will do the job. If you close the Connection object first, it will close the Statement object as well. However, you should always explicitly close the Statement object to ensure proper cleanup.

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  stmt.close();
}
```

8.2.3.2. The PreparedStatement Objects:

The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.

This statement gives you the flexibility of supplying arguments dynamically.

Creating PreparedStatement Object

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

All parameters in JDBC are represented by the ? symbol, which is known as the parameter marker. You must supply values for every parameter before executing the SQL statement.

The setXXX() methods bind values to the parameters, where XXX represents the Java data type of the value you wish to bind to the input parameter. If you forget to supply the values, you will receive a SQLException.

Each parameter marker is referred by its ordinal position. The first marker represents position 1, the next position 2, and so forth. This method differs from that of Java array indices, which starts at 0.

All of the Statement object's methods for interacting with the database (a) execute(), (b) executeQuery(), and (c) executeUpdate() also work with the PreparedStatement object. However, the methods are modified to use SQL statements that can input the parameters.

Closing PreparedStatement Object

Just as you close a Statement object, for the same reason you should also close the PreparedStatement object.

A simple call to the close() method will do the job. If you close the Connection object first, it will close the PreparedStatement object as well. However, you should always explicitly close the PreparedStatement object to ensure proper cleanup.

```
PreparedStatement pstmt = null;
```

```
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   pstmt.close();
}
```

8.2.3.3. The CallableStatement Objects:

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.

Creating CallableStatement Object

Suppose, you need to execute the following Oracle stored procedure −

```
CREATE OR REPLACE PROCEDURE getEmpName
   (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END;
```

NOTE: Above stored procedure has been written for Oracle, but we are working with MySQL database so, let us write same stored procedure for MySQL as follows to create it in EMP database –

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
```

```
END $$
DELIMITER ;
```

Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

Here are the definitions of each –

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

The following code snippet shows how to employ the Connection.prepareCall() method to instantiate a CallableStatement object based on the preceding stored procedure –

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

The String variable SQL, represents the stored procedure, with parameter placeholders.

Using the CallableStatement objects is much like using the PreparedStatement objects. You must bind values to all the parameters before executing the statement, or you will receive an SQLException.

If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.

When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type, to the data type that the stored procedure is expected to return.

Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

Closing CallableStatement Object

Just as you close other Statement object, for the same reason you should also close the CallableStatement object.

A simple call to the close() method will do the job. If you close the Connection object first, it will close the CallableStatement object as well. However, you should always explicitly close the CallableStatement object to ensure proper cleanup.

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  cstmt.close();
}
```

Lecture 9: Java Servlet Environment and Role, Servlet life cycle

9.1. Servlet Environment and Role:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

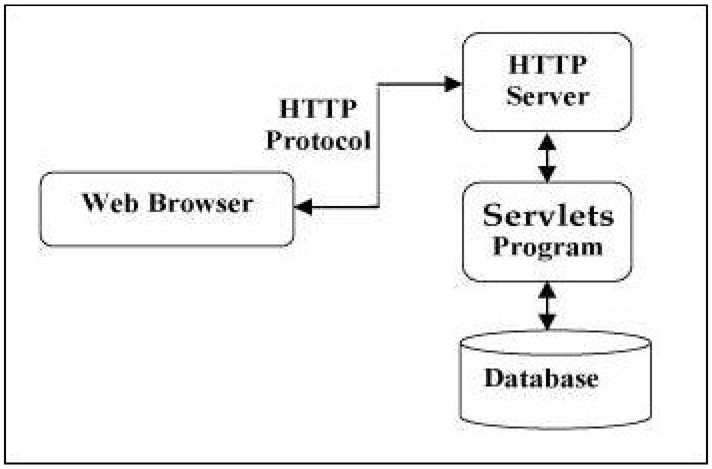
Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.

- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

9.2. Servlets Architecture:

The following diagram shows the position of Servlets in a Web Application.



9.3. Servlets Tasks:

Servlets perform the following major tasks –

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.

- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

9.4. Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http**packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

9.5. Servlets - Life Cycle:

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this –

```
public void init() throws ServletException {
  // Initialization code...
}
```

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
Here is the signature of this method –

```
public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException {
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
   throws ServletException, IOException {
   // Servlet code
}
```

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

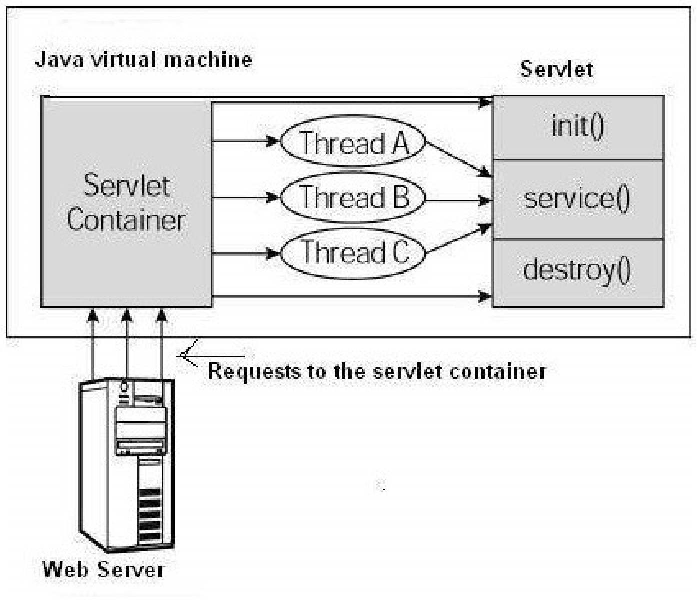
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this –

```
public void destroy() {
   // Finalization code...
}
```

Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



9.6. Servlets – Examples:

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code

Following is the sample source code structure of a servlet example to show Hello World −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException {
    // Do required initialization
    message = "Hello World";
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message + "</h1>");
  }

  public void destroy() {
    // do nothing.
  }
}
```

Compiling a Servlet

Let us create a file with name HelloWorld.java with the code shown above. Place this file at C:\ServletDevel (in Windows) or at /usr/ServletDevel (in Unix). This path location must be added to CLASSPATH before proceeding further.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows –

```
$ javac HelloWorld.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce **HelloWorld.class**file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of com.myorg.MyServlet, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

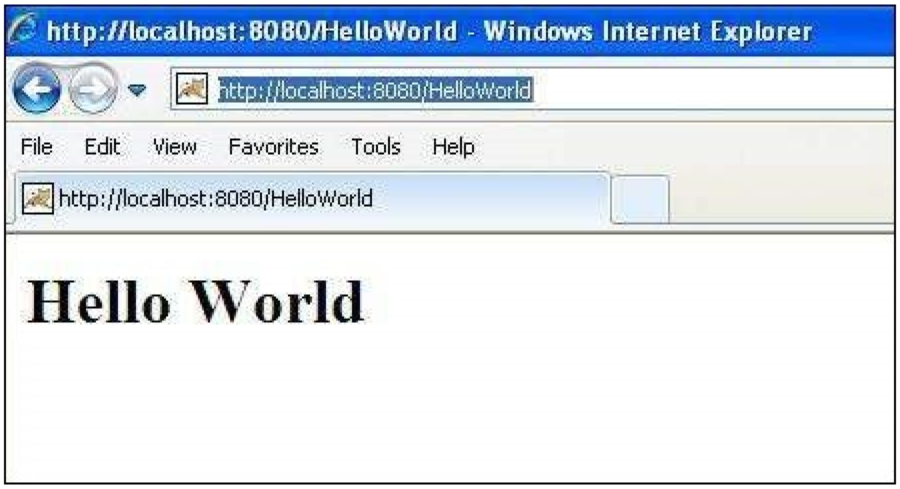
For now, let us copy HelloWorld.class into <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>
```

```
<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installationdirectory>\bin\startup.bat (on Windows) or <Tomcat-installationdirectory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in the browser's address box. If everything goes fine, you would get the following result.


http://localhost:8080/HelloWorld - Windows Internet Explorer
http://localhost:8080/HelloWorld
File Edit View Favorites Tools Help
http://localhost:8080/HelloWorld

Hello World

Lecture 10: Servlet methods- Request, Response, Get and Post

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

10.1. GET Method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? (question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.

10.2. POST Method:

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using doPost() method.

10.3. Reading Form Data using Servlet:

Servlets handles form data parsing automatically using the following methods depending on the situation −

- getParameter() − You call request.getParameter() method to get the value of a form parameter.
- getParameterValues() − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- getParameterNames() − Call this method if you want a complete list of all parameters in the current request.

10.4. GET Method Example using URL:

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

Given below is the HelloForm.java servlet program to handle input given by web browser. We are going to use getParameter() method which makes it very easy to access passed information −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
// Extend HttpServlet class
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
          "</ul>\n" +
        "</body>" +
      "</html>"
    );
  }
```

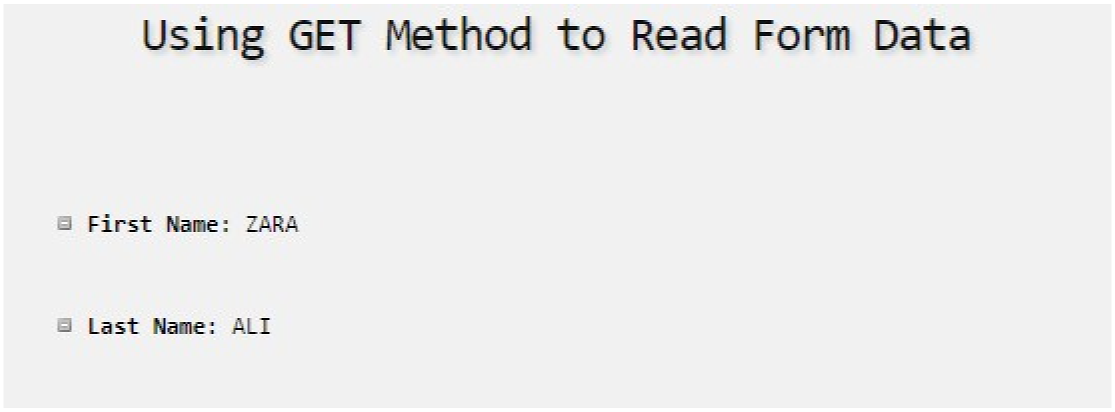
Assuming your environment is set up properly, compile HelloForm.java as follows −

```
$ javac HelloForm.java
```

If everything goes fine, above compilation would produce HelloForm.class file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloForm</servlet-name>
<servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloForm</servlet-name>
<url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Now type http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result –



```
Using GET Method to Read Form Data

• First Name: ZARA

• Last Name: ALI
```

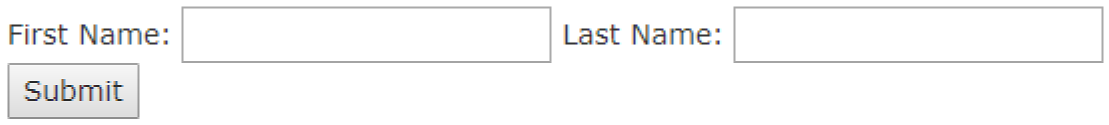
10.5. GET Method Example Using Form:

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
<body>
<form action = "HelloForm" method = "GET">
    First Name: <input type = "text" name = "first_name">
<br />
    Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
```

```
</body>
</html>
```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would accesshttp://localhost:8080/Hello.htm, here is the actual output of the above form.



Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

10.6. POST Method Example Using Form:

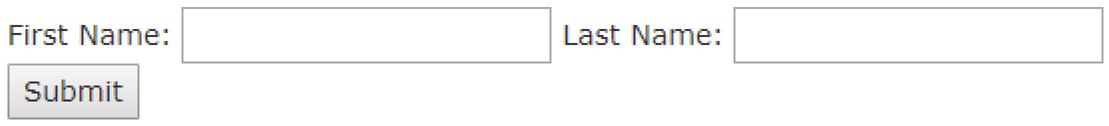
Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>"
         "</html>"
      );
   }
   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows −

```
<html>
<body>
<form action = "HelloForm" method = "POST">
      First Name: <input type = "text" name = "first_name">
<br />
      Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
</body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.



Based on the input provided, it would generate similar result as mentioned in the above examples.

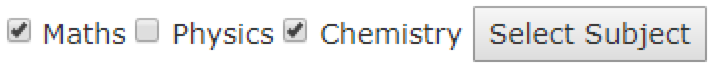
10.7. Passing Checkbox Data to Servlet Program:

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
<body>
<form action = "CheckBox" method = "POST" target = "_blank">
<input type = "checkbox" name = "maths" checked = "checked" /> Maths
<input type = "checkbox" name = "physics"  /> Physics
<input type = "checkbox" name = "chemistry" checked = "checked" />
                                    Chemistry
<input type = "submit" value = "Select Subject" />
</form>
</body>
</html>
```

The result of this code is the following form

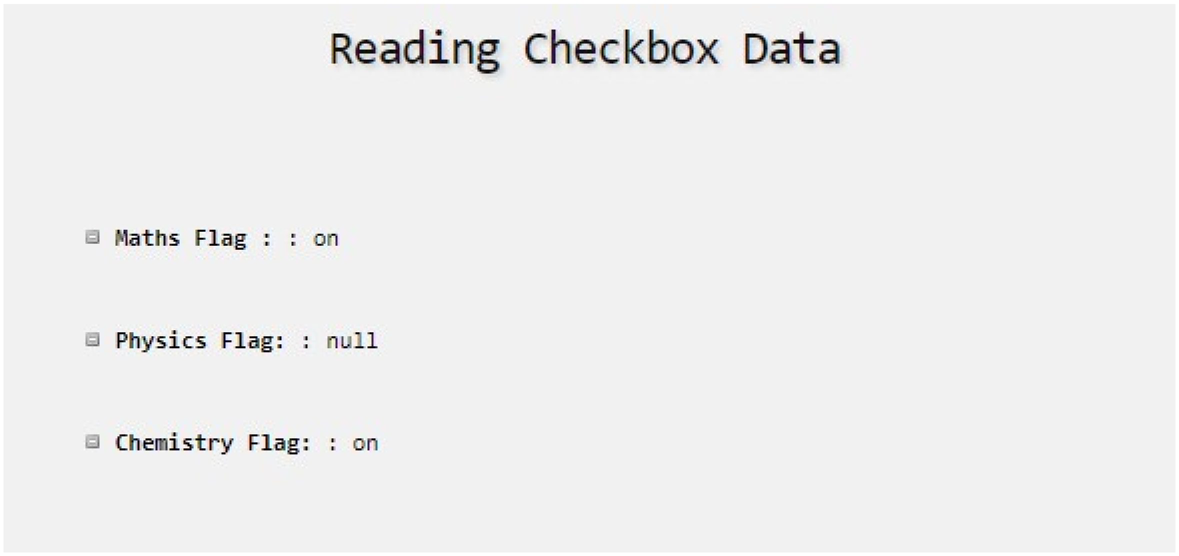


Given below is the CheckBox.java servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class CheckBox extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading Checkbox Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>Maths Flag : </b>: "
                  + request.getParameter("maths") + "\n" +
                  "  <li><b>Physics Flag: </b>: "
                  + request.getParameter("physics") + "\n" +
                  "  <li><b>Chemistry Flag: </b>: "
                  + request.getParameter("chemistry") + "\n" +
               "</ul>\n" +
            "</body>"
         "</html>"
      );
   }
   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
```

For the above example, it would display following result –



Reading Checkbox Data

- Maths Flag : : on
- Physics Flag: : null
- Chemistry Flag: : on

10.8. Reading All Form Parameters:

Following is the generic example which uses getParameterNames() method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.

Once we have an Enumeration, we can loop down the Enumeration in standard way by, using hasMoreElements() method to determine when to stop and using nextElement() method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
```

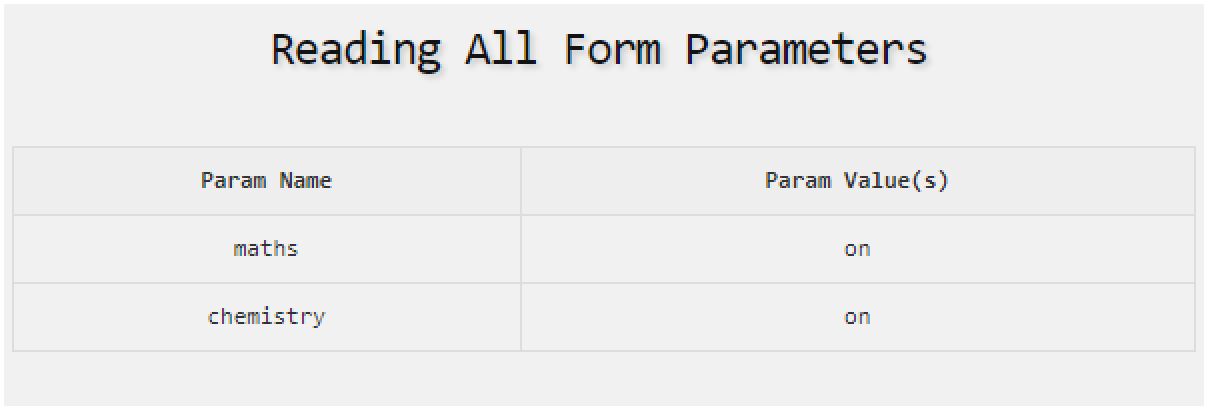
```
String title = "Reading All Form Parameters";
String docType =
  "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

out.println(docType +
  "<html>\n" +
  "<head><title>" + title + "</title></head>\n" +
  "<body bgcolor = \"#f0f0f0\">\n" +
  "<h1 align = \"center\">" + title + "</h1>\n" +
  "<table width = \"100%\" border = \"1\" align = \"center\">\n" +
  "<tr bgcolor = \"#949494\">\n" +
    "<th>Param Name</th>"
    "<th>Param Value(s)</th>\n"+
  "</tr>\n"
);

Enumeration paramNames = request.getParameterNames();

while(paramNames.hasMoreElements()) {
  String paramName = (String)paramNames.nextElement();
  out.print("<tr><td>" + paramName + "</td>\n<td>");
  String[] paramValues = request.getParameterValues(paramName);

  // Read single valued data
  if (paramValues.length == 1) {
    String paramValue = paramValues[0];
    if (paramValue.length() == 0)
      out.println("<i>No Value</i>");
      else
      out.println(paramValue);
  } else {
    // Read multiple valued data
    out.println("<ul>");

    for(int i = 0; i < paramValues.length; i++) {
      out.println("<li>" + paramValues[i]);
    }
    out.println("</ul>");
```

```
        }
      }
      out.println("</tr>\n</table>\n</body></html>");
    }

    // Method to handle POST method request.
    public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
    }
}
```

Now, try the above servlet with the following form −

```
<html>
<body>
<form action = "ReadParams" method = "POST" target = "_blank">
<input type = "checkbox" name = "maths" checked = "checked" /> Maths
<input type = "checkbox" name = "physics"  /> Physics
<input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
<input type = "submit" value = "Select Subject" />
</form>
</body>
</html>
```

Now calling servlet using the above form would generate the following result –



Reading All Form Parameters

Param Name	Param Value(s)
maths	on
chemistry	on

You can try the above servlet to read any other form's data having other objects like text box, radio button or drop down box etc.

10.9. Servlets - Client HTTP Request:

When a browser requests for a web page, it sends lot of information to the web server which cannot be read directly because this information travel as a part of header of HTTP request. You can check HTTP Protocol for more information on this.

Following is the important header information which comes from browser side and you would use very frequently in web programming –

Sr.No.	Header & Description
1	**Accept** This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or **image/jpeg** are the two most common possibilities.
2	**Accept-Charset** This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.
3	**Accept-Encoding** This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or **compress** are the two most common possibilities.
4	**Accept-Language** This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc
5	**Authorization** This header is used by clients to identify themselves when accessing password-protected Web pages.
6	**Connection** This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used.
7	**Content-Length** This header is applicable only to POST requests and gives the size of the POST data in bytes.
8	**Cookie** This header returns cookies to servers that previously sent them to the browser.
9	**Host** This header specifies the host and port as given in the original URL.
10	**If-Modified-Since** This header indicates that the client wants the page only if it has been changed

	after the specified date. The server sends a code, 304 which means **Not Modified** header if no newer result is available.
11	**If-Unmodified-Since** This header is the reverse of If-Modified-Since; it specifies that the operation should succeed only if the document is older than the specified date.
12	**Referer** This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referrer header when the browser requests Web page 2.
13	**User-Agent** This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

10.9.1. Methods to read HTTP Header:

There are following methods which can be used to read HTTP header in your servlet program. These methods are available with HttpServletRequest object.

Sr.No.	Method & Description
1	**Cookie[] getCookies()** Returns an array containing all of the Cookie objects the client sent with this request.
2	**Enumeration getAttributeNames()** Returns an Enumeration containing the names of the attributes available to this request.
3	**Enumeration getHeaderNames()** Returns an enumeration of all the header names this request contains.
4	**Enumeration getParameterNames()** Returns an Enumeration of String objects containing the names of the parameters contained in this request
5	**HttpSession getSession()** Returns the current session associated with this request, or if the request does not have a session, creates one.
6	**HttpSession getSession(boolean create)** Returns the current HttpSession associated with this request or, if if there is no current session and value of create is true, returns a new session.
7	**Locale getLocale()** Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.

8	**Object getAttribute(String name)** Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
9	**ServletInputStream getInputStream()** Retrieves the body of the request as binary data using a ServletInputStream.
10	**String getAuthType()** Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or null if the JSP was not protected.
11	**String getCharacterEncoding()** Returns the name of the character encoding used in the body of this request.
12	**String getContentType()** Returns the MIME type of the body of the request, or null if the type is not known.
13	**String getContextPath()** Returns the portion of the request URI that indicates the context of the request.
14	**String getHeader(String name)** Returns the value of the specified request header as a String.
15	**String getMethod()** Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
16	**String getParameter(String name)** Returns the value of a request parameter as a String, or null if the parameter does not exist.
17	**String getPathInfo()** Returns any extra path information associated with the URL the client sent when it made this request
18	**String getProtocol()** Returns the name and version of the protocol the request.
19	**String getQueryString()** Returns the query string that is contained in the request URL after the path.
20	**String getRemoteAddr()** Returns the Internet Protocol (IP) address of the client that sent the request.
21	**String getRemoteHost()** Returns the fully qualified name of the client that sent the request.
22	**String getRemoteUser()** Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
23	**String getRequestURI()**

	Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
24	**String getRequestedSessionId()** Returns the session ID specified by the client.
25	**String getServletPath()** Returns the part of this request's URL that calls the JSP.
26	**String[] getParameterValues(String name)** Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
27	**boolean isSecure()** Returns a Boolean indicating whether this request was made using a secure channel, such as HTTPS.
28	**int getContentLength()** Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
29	**int getIntHeader(String name)** Returns the value of the specified request header as an int.
30	**int getServerPort()** Returns the port number on which this request was received.

10.9.2. HTTP Header Request Example:

Following is the example which uses getHeaderNames() method of HttpServletRequest to read the HTTP header information. This method returns an Enumeration that contains the header information associated with the current HTTP request.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using hasMoreElements() method to determine when to stop and using nextElement() method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class DisplayHeader extends HttpServlet {
```

```
// Method to handle GET method request.
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

  // Set response content type
  response.setContentType("text/html");

  PrintWriter out = response.getWriter();
  String title = "HTTP Header Request Example";
  String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

  out.println(docType +
    "<html>\n" +
    "<head><title>" + title + "</title></head>\n"+
    "<body bgcolor = \"#f0f0f0\">\n" +
    "<h1 align = \"center\">" + title + "</h1>\n" +
    "<table width = \"100%\" border = \"1\" align = \"center\">\n" +
    "<tr bgcolor = \"#949494\">\n" +
    "<th>Header Name</th><th>Header Value(s)</th>\n"+
    "</tr>\n"
  );

  Enumeration headerNames = request.getHeaderNames();

  while(headerNames.hasMoreElements()) {
    String paramName = (String)headerNames.nextElement();
    out.print("<tr><td>" + paramName + "</td>\n");
    String paramValue = request.getHeader(paramName);
    out.println("<td> " + paramValue + "</td></tr>\n");
  }
  out.println("</table>\n</body></html>");
}

// Method to handle POST method request.
public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Now calling the above servlet would generate the following result –

HTTP Header Request Example

Header Name	**Header Value(s)**
accept	*/*
accept-language	en-us
user-agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; InfoPath.2; MS-RTC LM 8)
accept-encoding	gzip, deflate
host	localhost:8080
connection	Keep-Alive
cache-control	no-cache

10.10. Servlets - Server HTTP Response:

As discussed in the previous chapter, when a Web server responds to an HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document. A typical response looks like this −

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
```

```
<!doctype ...>
<html>
<head>...</head>
<body>
   ...
</body>
</html>
```

The status line consists of the HTTP version (HTTP/1.1 in the example), a status code (200 in the example), and a very short message corresponding to the status code (OK in the example).

Following is a summary of the most useful HTTP 1.1 response headers which go back to the browser from web server side and you would use them very frequently in web programming –

Sr.No.	Header & Description
1	**Allow** This header specifies the request methods (GET, POST, etc.) that the server supports.
2	**Cache-Control** This header specifies the circumstances in which the response document can safely be cached. It can have values **public**, **private**or **no-cache** etc. Public means document is cacheable, Private means document is for a single user and can only be stored in private (non-shared) caches and nocache means document should never be cached.
3	**Connection** This header instructs the browser whether to use persistent in HTTP connections or not. A value of **close** instructs the browser not to use persistent HTTP connections and **keepalive** means using persistent connections.
4	**Content-Disposition** This header lets you request that the browser ask the user to save the response to disk in a file of the given name.
5	**Content-Encoding** This header specifies the way in which the page was encoded during transmission.
6	**Content-Language** This header signifies the language in which the document is written. For example en, en-us, ru, etc
7	**Content-Length**

	This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection.
8	**Content-Type** This header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
9	**Expires** This header specifies the time at which the content should be considered out-of-date and thus no longer be cached.
10	**Last-Modified** This header indicates when the document was last changed. The client can then cache the document and supply a date by an **If-Modified-Since** request header in later requests.
11	**Location** This header should be included with all responses that have a status code in the 300s. This notifies the browser of the document address. The browser automatically reconnects to this location and retrieves the new document.
12	**Refresh** This header specifies how soon the browser should ask for an updated page. You can specify time in number of seconds after which a page would be refreshed.
13	**Retry-After** This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.
14	**Set-Cookie** This header specifies a cookie associated with the page.

10.10.1. Methods to Set HTTP Response Header:

There are following methods which can be used to set HTTP response header in your servlet program. These methods are available with HttpServletResponse object.

Sr.No.	Method & Description
1	**String encodeRedirectURL(String url)** Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
2	**String encodeURL(String url)** Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

3	**boolean containsHeader(String name)** Returns a Boolean indicating whether the named response header has already been set.
4	**boolean isCommitted()** Returns a Boolean indicating if the response has been committed.
5	**void addCookie(Cookie cookie)** Adds the specified cookie to the response.
6	**void addDateHeader(String name, long date)** Adds a response header with the given name and date-value.
7	**void addHeader(String name, String value)** Adds a response header with the given name and value.
8	**void addIntHeader(String name, int value)** Adds a response header with the given name and integer value.
9	**void flushBuffer()** Forces any content in the buffer to be written to the client.
10	**void reset()** Clears any data that exists in the buffer as well as the status code and headers.
11	**void resetBuffer()** Clears the content of the underlying buffer in the response without clearing headers or status code.
12	**void sendError(int sc)** Sends an error response to the client using the specified status code and clearing the buffer.
13	**void sendError(int sc, String msg)** Sends an error response to the client using the specified status.
14	**void sendRedirect(String location)** Sends a temporary redirect response to the client using the specified redirect location URL.
15	**void setBufferSize(int size)** Sets the preferred buffer size for the body of the response.
16	**void setCharacterEncoding(String charset)** Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.
17	**void setContentLength(int len)** Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
18	**void setContentType(String type)** Sets the content type of the response being sent to the client, if the response has

	not been committed yet.
19	**void setDateHeader(String name, long date)** Sets a response header with the given name and date-value.
20	**void setHeader(String name, String value)** Sets a response header with the given name and value.
21	**void setIntHeader(String name, int value)** Sets a response header with the given name and integer value
22	**void setLocale(Locale loc)** Sets the locale of the response, if the response has not been committed yet.
23	**void setStatus(int sc)** Sets the status code for this response

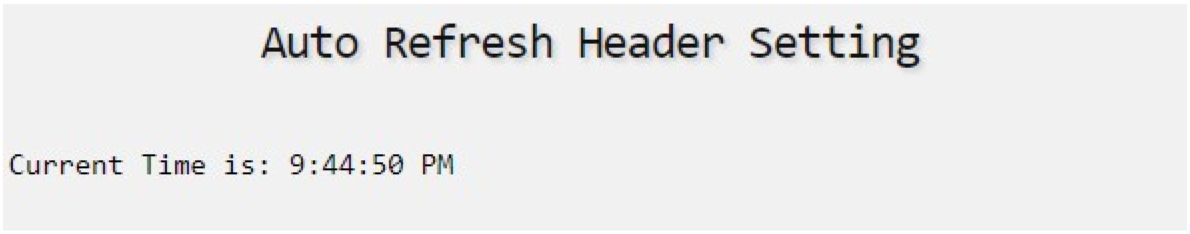
10.10.2. HTTP Header Response Example:

You already have seen setContentType() method working in previous examples and following example would also use same method, additionally we would use setIntHeader() method to set Refresh header.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class Refresh extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set refresh, autoload time as 5 seconds
    response.setIntHeader("Refresh", 5);

    // Set response content type
    response.setContentType("text/html");

    // Get current time
    Calendar calendar = new GregorianCalendar();
    String am_pm;
```

```
      int hour = calendar.get(Calendar.HOUR);
      int minute = calendar.get(Calendar.MINUTE);
      int second = calendar.get(Calendar.SECOND);

      if(calendar.get(Calendar.AM_PM) == 0)
         am_pm = "AM";
      else
         am_pm = "PM";

      String CT = hour+":"+ minute +":"+ second +" "+ am_pm;

      PrintWriter out = response.getWriter();
      String title = "Auto Refresh Header Setting";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n"+
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<p>Current Time is: " + CT + "</p>\n"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now calling the above servlet would display current system time after every 5 seconds as follows. Just run the servlet and wait to see the result –


```
Auto Refresh Header Setting

Current Time is: 9:44:50 PM
```

Lecture 11: Cookies and Session

11.1. Servlets - Cookies Handling:

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This lecture will teach you how to set or reset cookies, how to access them and how to delete them.

11.1.1. The Anatomy of a Cookie:

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A servlet that sets a cookie might send headers that look something like this −

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT;
   path = /; domain = tutorialspoint.com
Connection: close
Content-Type: text/html
```

`As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this −

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name = xyz
```

A servlet will then have access to the cookie through the request method request.getCookies() which returns an array of Cookie objects.

11.1.2. Servlet Cookies Methods:

Following is the list of useful methods which you can use while manipulating cookies in servlet.

Sr.No.	Method & Description
1	**public void setDomain(String pattern)** This method sets the domain to which cookie applies, for example tutorialspoint.com.
2	**public String getDomain()** This method gets the domain to which cookie applies, for example tutorialspoint.com.
3	**public void setMaxAge(int expiry)** This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
4	**public int getMaxAge()** This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
5	**public String getName()** This method returns the name of the cookie. The name cannot be changed after creation.
6	**public void setValue(String newValue)** This method sets the value associated with the cookie
7	**public String getValue()** This method gets the value associated with the cookie.
8	**public void setPath(String uri)** This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
9	**public String getPath()** This method gets the path to which this cookie applies.
10	**public void setSecure(boolean flag)** This method sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
11	**public void setComment(String purpose)** This method specifies a comment that describes a cookie's purpose. The

	comment is useful if the browser presents the cookie to the user.
12	**public String getComment()** This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

11.1.3. Setting Cookies with Servlet:

Setting cookies with servlet involves three steps –

Creating a Cookie object

You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

Cookie cookie = new Cookie("key","value");

Keep in mind, neither the name nor the value should contain white space or any of the following characters −

[ ] ( ) = , " / ? @ : ;

Setting the maximum age

You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

cookie.setMaxAge(60 * 60 * 24);

Sending the Cookie into the HTTP response headers

You use response.addCookie to add cookies in the HTTP response header as follows −

response.addCookie(cookie);

Example

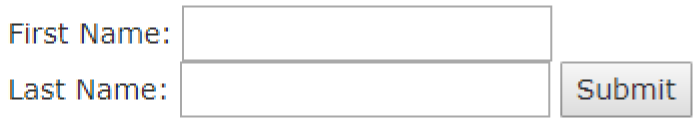
Let us modify our Form Example (mentioned in the previous lecture) to set the cookies for first and last name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Create cookies for first and last names.
    Cookie firstName = new Cookie("first_name", request.getParameter("first_name"));
    Cookie lastName = new Cookie("last_name", request.getParameter("last_name"));

    // Set expiry date after 24 Hrs for both the cookies.
    firstName.setMaxAge(60*60*24);
    lastName.setMaxAge(60*60*24);

    // Add both the cookies in the response header.
    response.addCookie( firstName );
    response.addCookie( lastName );

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Setting Cookies Example";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head>
<title>" + title + "</title>
</head>\n" +

        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
          "</ul>\n" +
            "</body>
</html>"
      );
   }
}
```

Compile the above servlet HelloForm and create appropriate entry in web.xml file and finally try following HTML page to call servlet.

```
<html>
<body>
<form action = "HelloForm" method = "GET">
      First Name: <input type = "text" name = "first_name">
<br />
      Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
</body>
</html>
```

Keep above HTML content in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access http://localhost:8080/Hello.htm, here is the actual output of the above form.



Try to enter First Name and Last Name and then click submit button. This would display first name and last name on your screen and same time it would set two cookies firstName and lastName which would be passed back to the server when next time you would press Submit button.

Next section would explain you how you would access these cookies back in your web application.

11.1.4. Reading Cookies with Servlet:

To read cookies, you need to create an array of javax.servlet.http.Cookie objects by calling the getCookies() method of HttpServletRequest. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

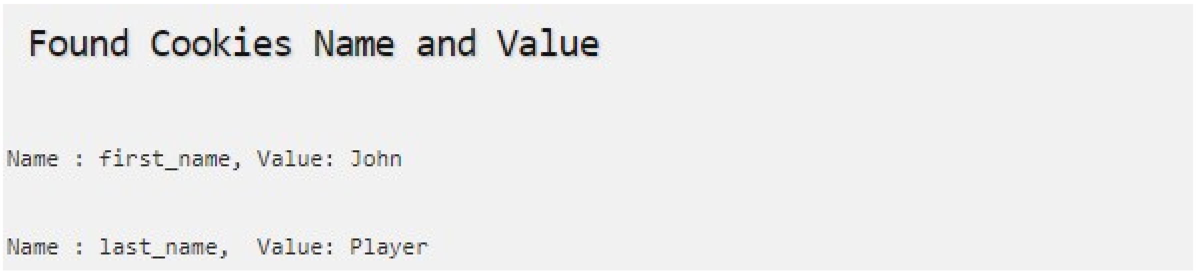
Example

Let us read cookies which we have set in previous example −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class ReadCookies extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    Cookie cookie = null;
    Cookie[] cookies = null;

    // Get an array of Cookies associated with this domain
    cookies = request.getCookies();

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Reading Cookies Example";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor = \"#f0f0f0\">\n" );

    if( cookies != null ) {
      out.println("<h2> Found Cookies Name and Value</h2>");

      for (int i = 0; i < cookies.length; i++) {
        cookie = cookies[i];
        out.print("Name : " + cookie.getName( ) + ",  ");
```

```
            out.print("Value: " + cookie.getValue( ) + " <br/>");
          }
        } else {
          out.println("<h2>No cookies founds</h2>");
        }
        out.println("</body>");
        out.println("</html>");
      }
    }
```

Compile above servlet ReadCookies and create appropriate entry in web.xml file. If you would have set first_name cookie as "John" and last_name cookie as "Player" then running http://localhost:8080/ReadCookies would display the following result –



```
Found Cookies Name and Value

Name : first_name, Value: John

Name : last_name,  Value: Player
```

11.1.5. Delete Cookies with Servlet:

To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps −

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using setMaxAge() method to delete an existing cookie
- Add this cookie back into response header.

Example

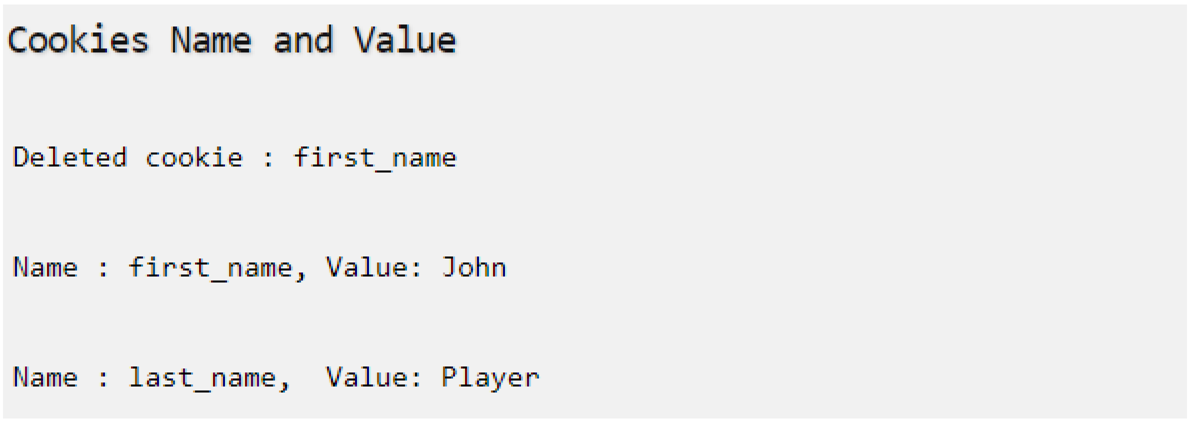
The following example would delete and existing cookie named "first_name" and when you would run ReadCookies servlet next time it would return null value for first_name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class DeleteCookies extends HttpServlet {
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

  Cookie cookie = null;
  Cookie[] cookies = null;

  // Get an array of Cookies associated with this domain
  cookies = request.getCookies();

  // Set response content type
  response.setContentType("text/html");

  PrintWriter out = response.getWriter();
  String title = "Delete Cookies Example";
  String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

  out.println(docType +
    "<html>\n" +
    "<head><title>" + title + "</title></head>\n" +
    "<body bgcolor = \"#f0f0f0\">\n" );

  if( cookies != null ) {
    out.println("<h2> Cookies Name and Value</h2>");

    for (int i = 0; i < cookies.length; i++) {
      cookie = cookies[i];

      if((cookie.getName( )).compareTo("first_name") == 0 ) {
        cookie.setMaxAge(0);
        response.addCookie(cookie);
        out.print("Deleted cookie : " + cookie.getName( ) + "<br/>");
      }
      out.print("Name : " + cookie.getName( ) + ",  ");
      out.print("Value: " + cookie.getValue( )+" <br/>");
    }
  } else {
```

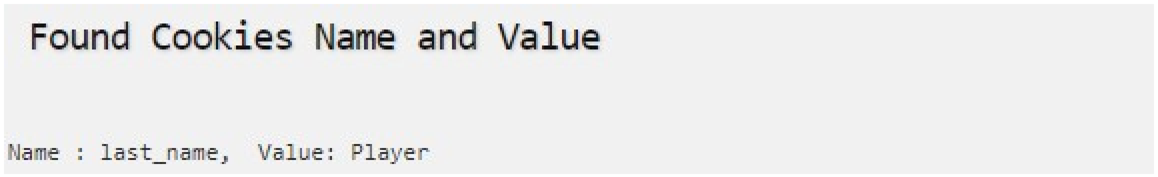
```
            out.println("<h2>No cookies founds</h2>");
          }
          out.println("</body>");
          out.println("</html>");
        }
      }
```

Compile above servlet DeleteCookies and create appropriate entry in web.xml file. Now running http://localhost:8080/DeleteCookies would display the following result –



```
Cookies Name and Value

Deleted cookie : first_name

Name : first_name, Value: John

Name : last_name,  Value: Player
```

Now try to run http://localhost:8080/ReadCookies and it would display only one cookie as follows –



```
Found Cookies Name and Value

Name : last_name,  Value: Player
```

You can delete your cookies in Internet Explorer manually. Start at the Tools menu and select Internet Options. To delete all cookies, press Delete Cookies.

11.2. Servlets - Session Tracking:

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server −

11.2.1. Cookies:

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way because many time browser does not support a cookie, so I would not recommend to use this procedure to maintain the sessions.

11.2.2. Hidden Form Fields:

A web server can send a hidden HTML form field along with a unique session ID as follows −

```
<input type = "hidden" name = "sessionid" value = "12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

11.2.3. URL Rewriting:

You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.

For example, with http://tutorialspoint.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies. The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

11.2.4. The HttpSession Object:

Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

You would get HttpSession object by calling the public method getSession() of HttpServletRequest, as below –

```
HttpSession session = request.getSession();
```

You need to call request.getSession() before you send any document content to the client. Here is a summary of the important methods available through HttpSession object –

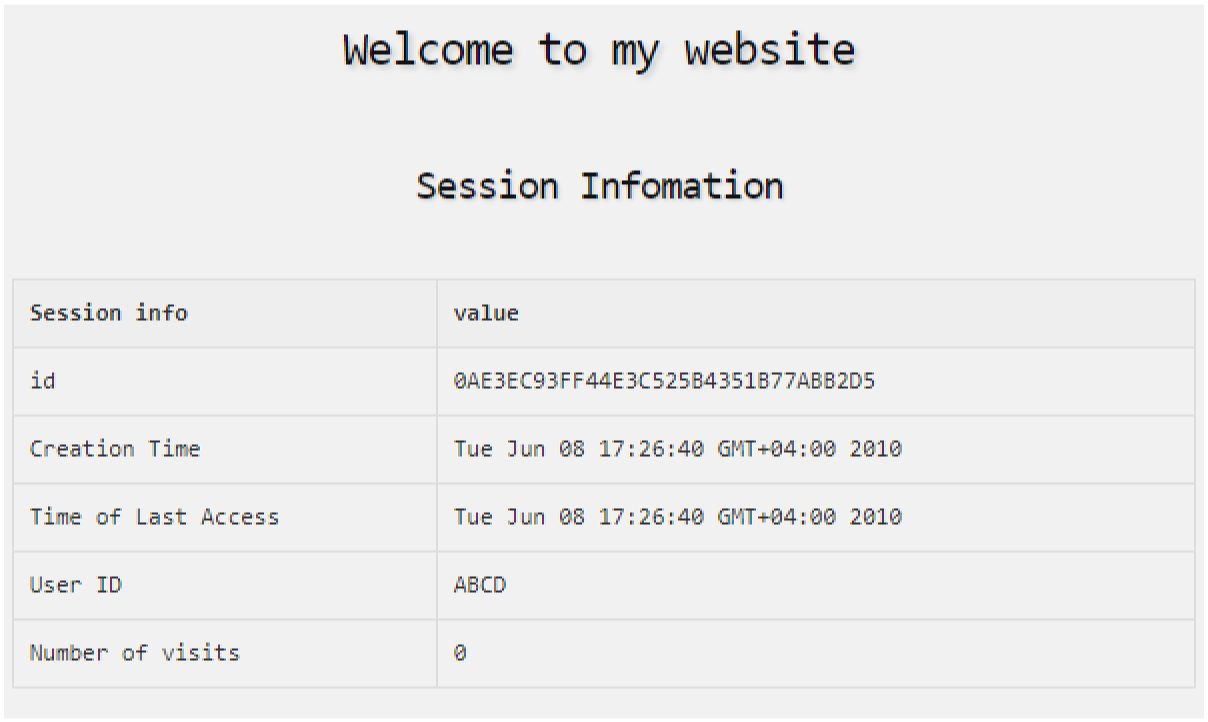
Sr.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval (seconds), that the servlet container will keep the session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew(** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

11.2.5. Session Tracking Example:

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class SessionTrack extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Create a session object if it is already not  created.
    HttpSession session = request.getSession(true);

    // Get session creation time.
    Date createTime = new Date(session.getCreationTime());

    // Get last access time of this web page.
    Date lastAccessTime = new Date(session.getLastAccessedTime());

    String title = "Welcome Back to my website";
    Integer visitCount = new Integer(0);
    String visitCountKey = new String("visitCount");
    String userIDKey = new String("userID");
    String userID = new String("ABCD");

    // Check if this is new comer on your web page.
    if (session.isNew()) {
      title = "Welcome to my website";
      session.setAttribute(userIDKey, userID);
    } else {
      visitCount = (Integer)session.getAttribute(visitCountKey);
      visitCount = visitCount + 1;
      userID = (String)session.getAttribute(userIDKey);
    }
    session.setAttribute(visitCountKey,  visitCount);

    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +

        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<h2 align = \"center\">Session Infomation</h2>\n" +
          "<table border = \"1\" align = \"center\">\n" +

            "<tr bgcolor = \"#949494\">\n" +
              "  <th>Session info</th><th>value</th>
</tr>\n" +

            "<tr>\n" +
              "  <td>id</td>\n" +
              "  <td>" + session.getId() + "</td>
</tr>\n" +

            "<tr>\n" +
              "  <td>Creation Time</td>\n" +
              "  <td>" + createTime + "  </td>
</tr>\n" +

            "<tr>\n" +
              "  <td>Time of Last Access</td>\n" +
              "  <td>" + lastAccessTime + "  </td>
</tr>\n" +
                "<tr>\n" +
                  "  <td>User ID</td>\n" +
                  "  <td>" + userID + "  </td>
</tr>\n" +

                "<tr>\n" +
                  "  <td>Number of visits</td>\n" +
                  "  <td>" + visitCount + "</td>
</tr>\n" +
              "</table>\n" +
            "</body>
</html>"
  );
}
}
```

Compile the above servlet SessionTrack and create appropriate entry in web.xml file. Now running http://localhost:8080/SessionTrack would display the following result when you would run for the first time –

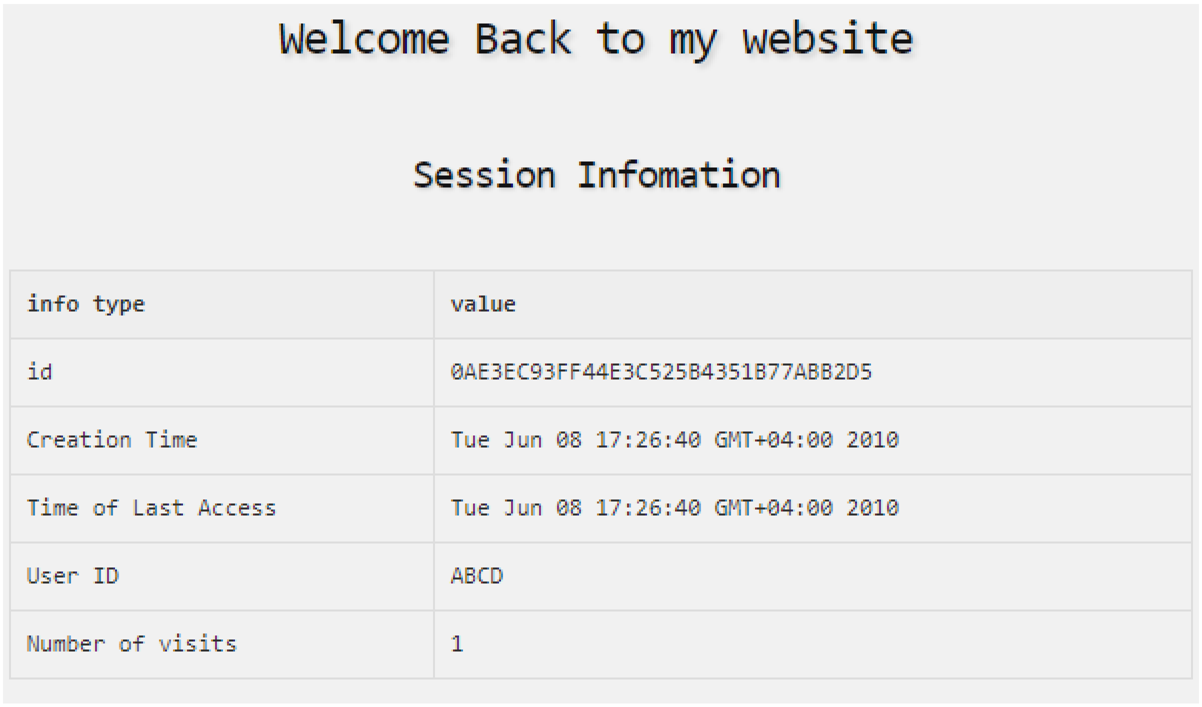


Welcome to my website

Session Infomation

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

Now try to run the same servlet for second time, it would display following result.



Welcome Back to my website

Session Infomation

info type	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1

11.2.6. Deleting Session Data:

When you are done with a user's session data, you have several options –

- Remove a particular attribute − You can call public void removeAttribute(String name) method to delete the value associated with a particular key.
- Delete the whole session − You can call public void invalidate() method to discard an entire session.
- Setting Session timeout − You can call public void setMaxInactiveInterval(int interval) method to set the timeout for a session individually.
- Log the user out − The servers that support servlets 2.4, you can call logout to log the client out of the Web server and invalidate all sessions belonging to all the users.
- web.xml Configuration − If you are using Tomcat, apart from the above mentioned methods, you can configure session time out in web.xml file as follows.

```
<session-config>
<session-timeout>15</session-timeout>
</session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

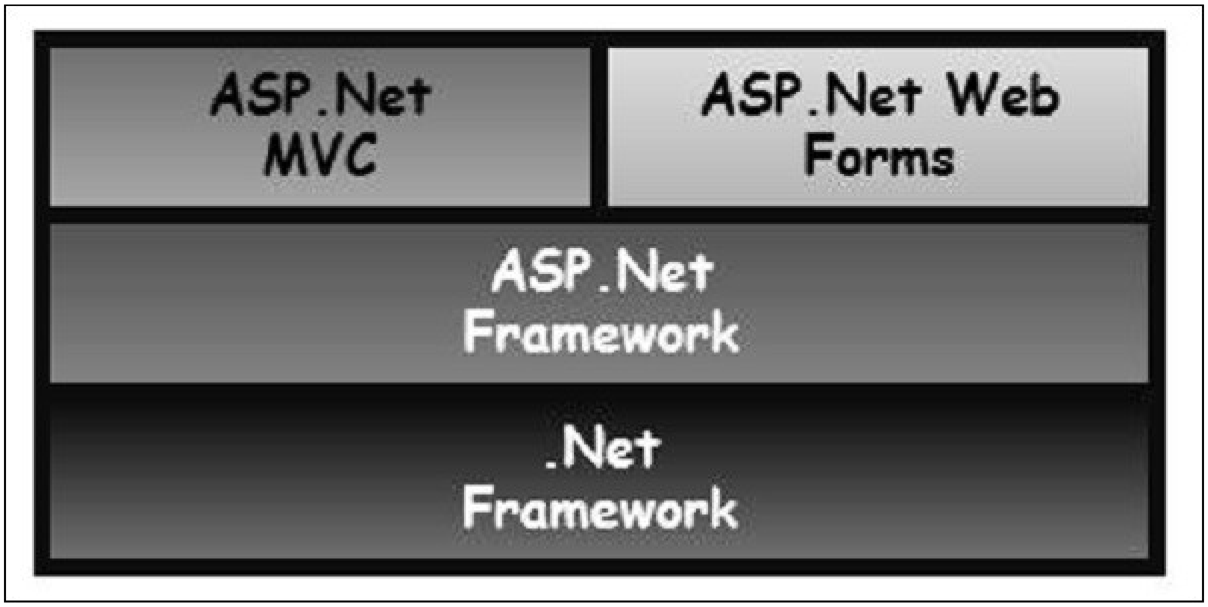
The getMaxInactiveInterval( ) method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, getMaxInactiveInterval( ) returns 900.

Lecture 12: ASP.Net with MVC introduction, MVC Architecture, MVC routing, controller, Action method, Action Selector and Action verb, Model and View

12.1. ASP.Net with MVC introduction:

ASP.NET MVC is basically a web development framework from Microsoft, which combines the features of MVC (Model-View-Controller) architecture, the most up-to-date ideas and techniques from Agile development, and the best parts of the existing ASP.NET platform.

ASP.NET MVC is not something, which is built from ground zero. It is a complete alternative to traditional ASP.NET Web Forms. It is built on the top of ASP.NET, so developers enjoy almost all the ASP.NET features while building the MVC application.



History

ASP.NET 1.0 was released on January 5, 2002, as part of .Net Framework version 1.0. At that time, it was easy to think of ASP.NET and Web Forms as one and the same thing. ASP.NET has however always supported two layers of abstraction −

- System.Web.UI − The Web Forms layer, comprising server controls, ViewState, and so on.
- System.Web − It supplies the basic web stack, including modules, handlers, the HTTP stack, etc.

By the time ASP.NET MVC was announced in 2007, the MVC pattern was becoming one of the most popular ways of building web frameworks.

In April 2009, the ASP.NET MVC source code was released under the Microsoft Public License (MS-PL). "ASP.NET MVC framework is a lightweight, highly testable presentation framework that is integrated with the existing ASP.NET features.

Some of these integrated features are master pages and membership-based authentication. The MVC framework is defined in the System.Web.Mvc assembly.

In March 2012, Microsoft had released part of its web stack (including ASP.NET MVC, Razor and Web API) under an open source license (Apache License 2.0). ASP.NET Web Forms was not included in this initiative.

Why ASP.NET MVC?

Microsoft decided to create their own MVC framework for building web applications. The MVC framework simply builds on top of ASP.NET. When you are building a web application with ASP.NET MVC, there will be no illusions of state, there will not be such a thing as a page load and no page life cycle at all, etc.

Another design goal for ASP.NET MVC was to be extensible throughout all aspects of the framework. So when we talk about views, views have to be rendered by a particular type of view engine. The default view engine is still something that can take an ASPX file. But if you don't like using ASPX files, you can use something else and plug in your own view engine.

There is a component inside the MVC framework that will instantiate your controllers. You might not like the way that the MVC framework instantiates your controller, you might want to handle that job yourself. So, there are lots of places in MVC where you can inject your own custom logic to handle tasks.

The whole idea behind using the Model View Controller design pattern is that you maintain a separation of concerns. Your controller is no longer encumbered with a lot of ties to the ASP.NET runtime or ties to the ASPX page, which is very hard to test. You now just have a class with regular methods on it that you can invoke in unit tests to find out if that controller is going to behave correctly.

Benefits of ASP.NET MVC

Following are the benefits of using ASP.NET MVC −

- Makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- Enables full control over the rendered HTML and provides a clean separation of concerns.
- Direct control over HTML also means better accessibility for implementing compliance with evolving Web standards.
- Facilitates adding more interactivity and responsiveness to existing apps.
- Provides better support for test-driven development (TDD).
- Works well for Web applications that are supported by large teams of developers and for Web designers who need a high degree of control over the application behavior.

12.2. ASP.NET MVC – Architecture:

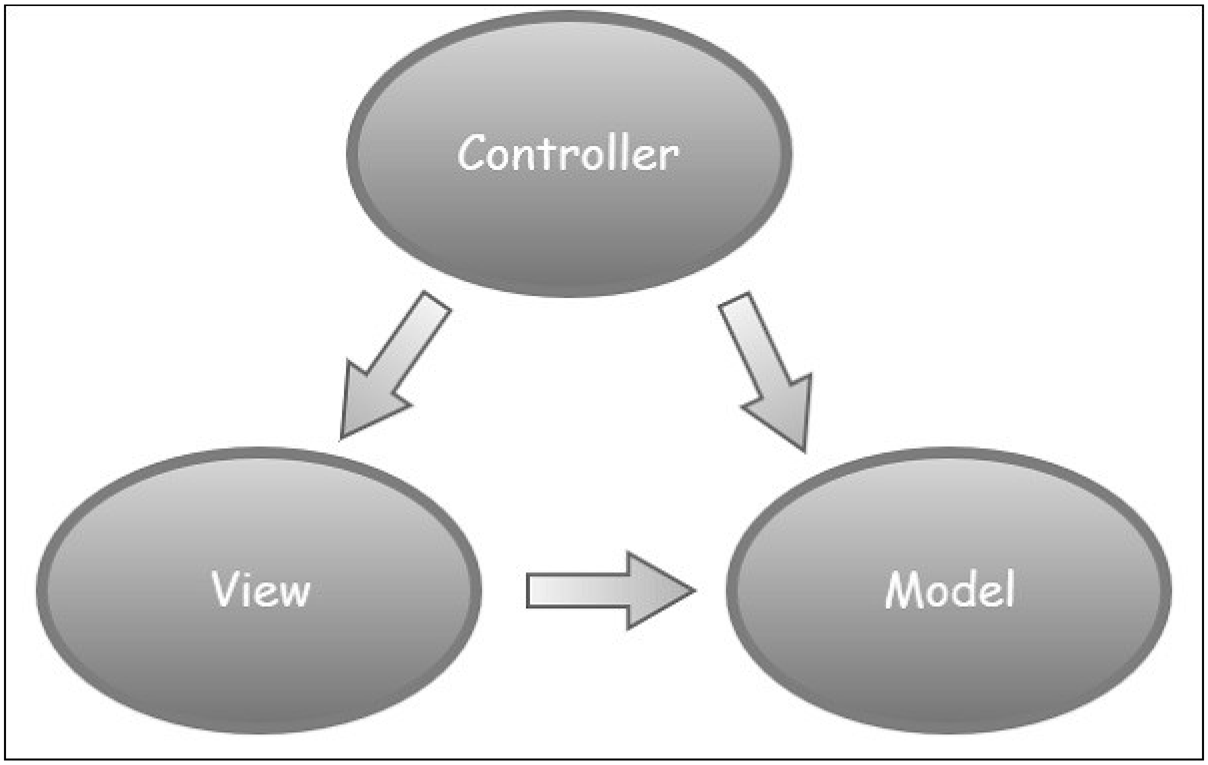
The MVC (Model-View-Controller) design pattern has actually been around for a few decades, and it's been used across many different technologies. Everything from Smalltalk to C++ to Java, and now C Sharp and .NET use this design pattern to build a user interface.

Following are some salient features of the MVC pattern −

- Originally it was named Thing-Model-View-Editor in 1979, and then it was later simplified to Model- View-Controller.

- It is a powerful and elegant means of separating concerns within an application (for example, separating data access logic from display logic) and applies itself extremely well to web applications.
- Its explicit separation of concerns does add a small amount of extra complexity to an application's design, but the extraordinary benefits outweigh the extra effort.

The MVC architectural pattern separates the user interface (UI) of an application into three main parts.



- **The Model** − A set of classes that describes the data you are working with as well as the business logic.
- **The View** − Defines how the application's UI will be displayed. It is a pure HTML, which decides how the UI is going to look like.
- **The Controller** − A set of classes that handles communication from the user, overall application flow, and application-specific logic.

Idea behind MVC

The idea is that you'll have a component called the view, which is solely responsible for rendering this user interface whether that be HTML or whether it actually be UI widgets on a desktop application.

The view talks to a model, and that model contains all of the data that the view needs to display. Views generally don't have much logic inside of them at all.

In a web application, the view might not have any code associated with it at all. It might just have HTML and then some expressions of where to take pieces of data from the model and plug them into the correct places inside the HTML template that you've built in the view.

The controller that organizes is everything. When an HTTP request arrives for an MVC application, that request gets routed to a controller, and then it's up to the controller to talk to either the database, the file system, or the model.

12.3. ASP.NET MVC - Getting Started:

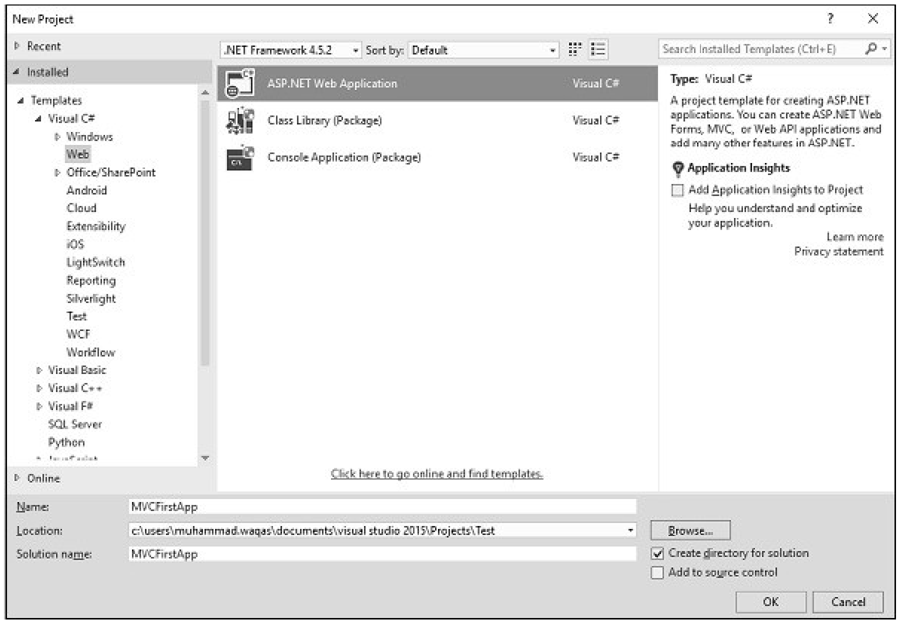
In this lecture, we will look at a simple working example of ASP.NET MVC. We will be building a simple web app here. To create an ASP.NET MVC application, we will use Visual Studio 2015, which contains all of the features you need to create, test, and deploy an MVC Framework application.

Create ASP.Net MVC Application

Following are the steps to create a project using project templates available in Visual Studio.

Step 1

Open the Visual Studio. Click File → New → Project menu option.



.NET Framework 4.5.2 Sort by: Default

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to Project
Help you understand and optimize your application.

Name: MVCFirstApp
Location: c:\users\muhammad.waqas\documents\visual studio 2015\Projects\Test
Solution name: MVCFirstApp

☑ Create directory for solution
☐ Add to source control

Step 2

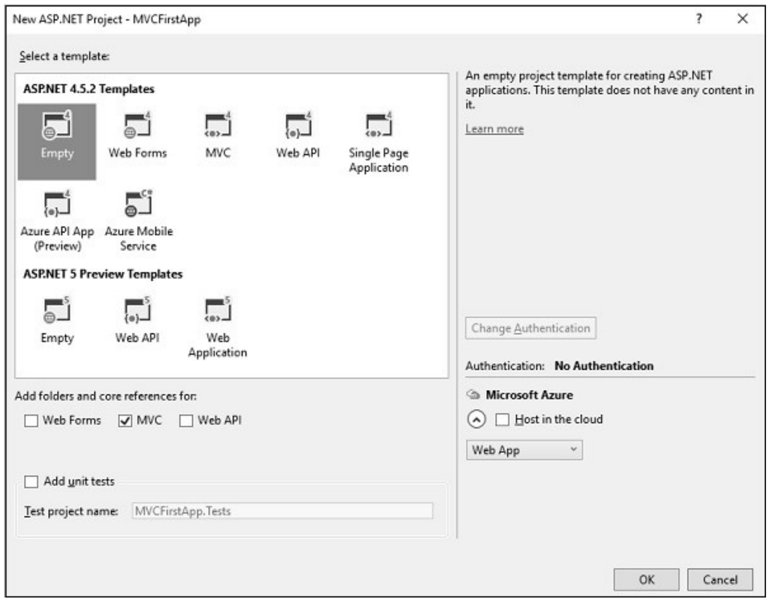
From the left pane, select Templates → Visual C# → Web.

Step 3

In the middle pane, select ASP.NET Web Application.

Step 4

Enter the project name, MVCFirstApp, in the Name field and click ok to continue. You will see the following dialog which asks you to set the initial content for the ASP.NET project.



New ASP.NET Project - MVCFirstApp

Select a template:

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Add folders and core references for: ☐ Web Forms ☑ MVC ☐ Web API

☐ Add unit tests

Test project name: MVCFirstApp.Tests

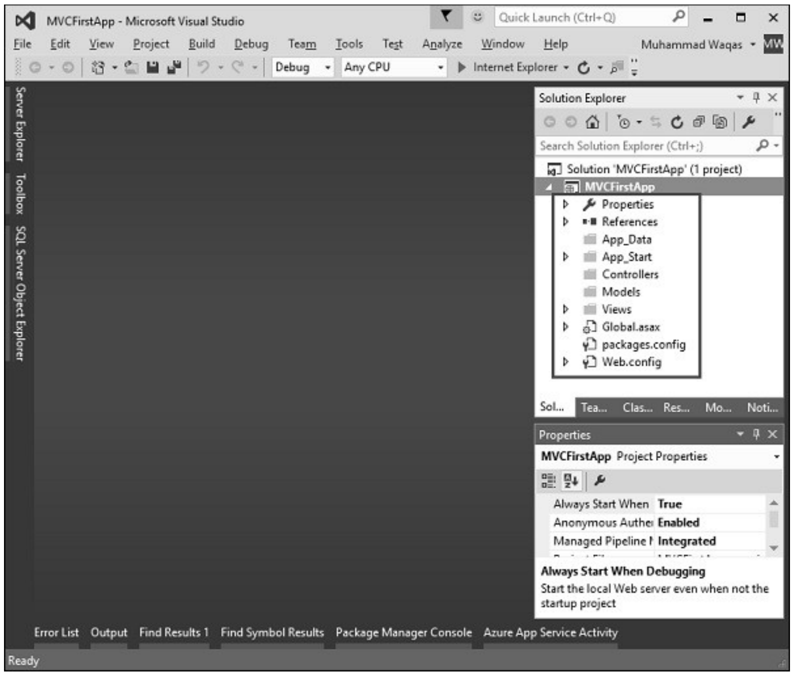
☐ Host in the cloud

Step 5

To keep things simple, select the 'Empty' option and check the MVC checkbox in the Add folders and core references section. Click Ok.

It will create a basic MVC project with minimal predefined content.

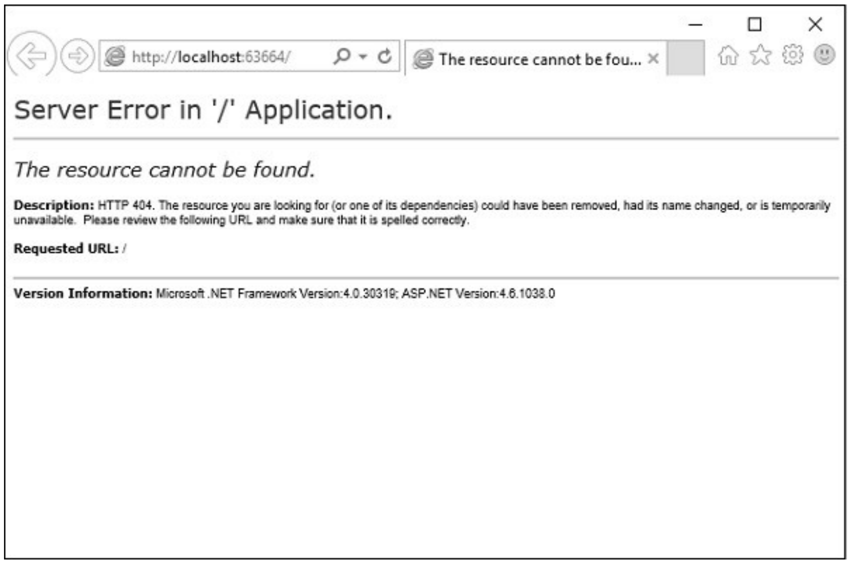
Once the project is created by Visual Studio, you will see a number of files and folders displayed in the Solution Explorer window.


MVCFirstApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug Any CPU Internet Explorer
Solution Explorer
Solution 'MVCFirstApp' (1 project)
MVCFirstApp
Properties
References
App_Data
App_Start
Controllers
Models
Views
Global.asax
packages.config
Web.config
Properties
MVCFirstApp Project Properties
Always Start When True
Anonymous Authe Enabled
Managed Pipeline Integrated
Always Start When Debugging
Start the local Web server even when not the startup project
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Ready

As you know that we have created ASP.Net MVC project from an empty project template, so for the moment the application does not contain anything to run.

Step 6

Run this application from Debug → Start Debugging menu option and you will see a 404 Not Found Error.



Server Error in '/' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, had its name changed, or is temporarily unavailable. Please review the following URL and make sure that it is spelled correctly.

Requested URL: /

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.6.1038.0

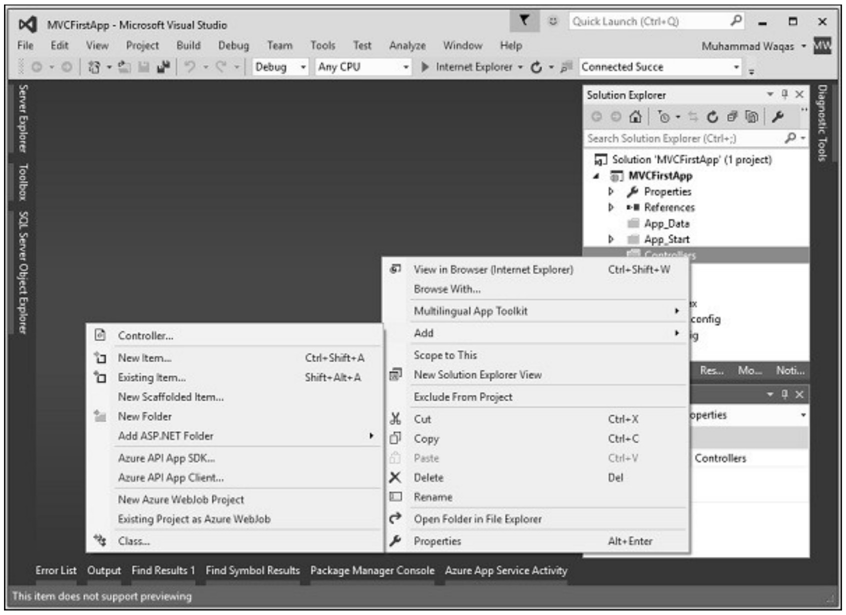
The default browser is, Internet Explorer, but you can select any browser that you have installed from the toolbar.

Add Controller

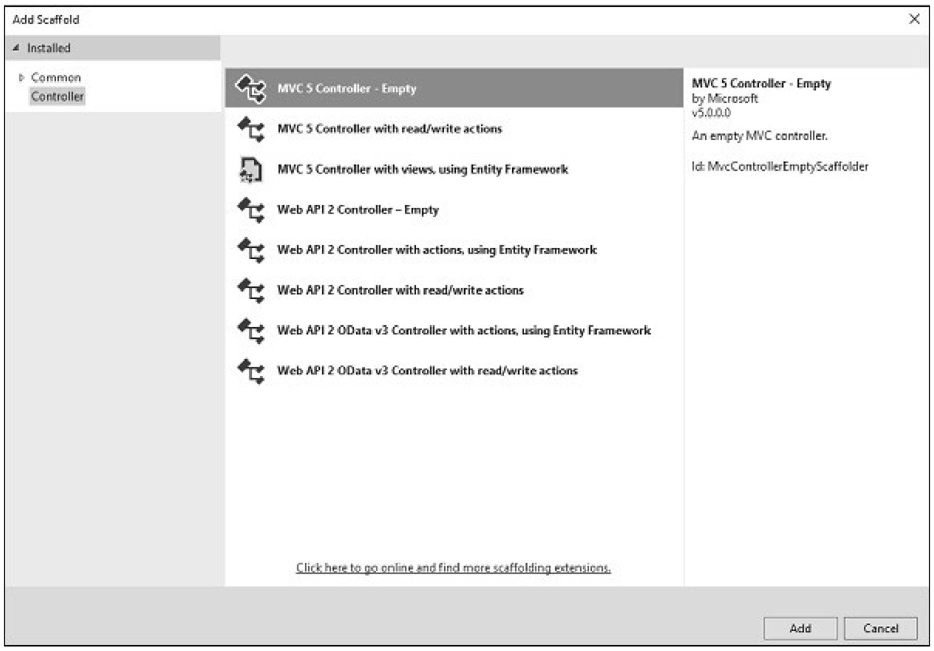
To remove the 404 Not Found error, we need to add a controller, which handles all the incoming requests.

Step 1

To add a controller, right-click on the controller folder in the solution explorer and select Add → Controller.

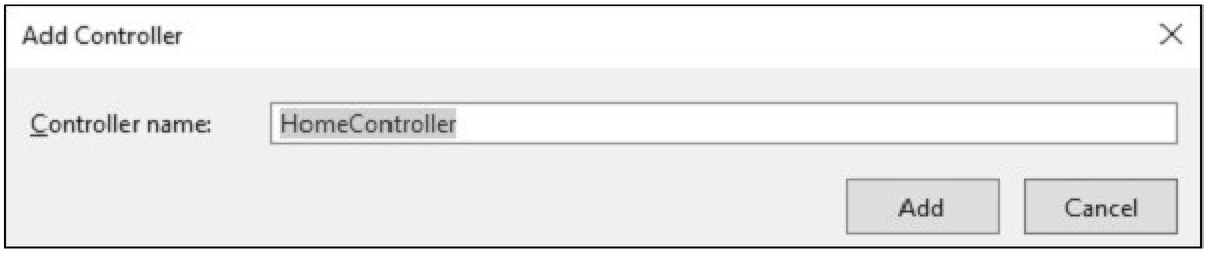

MVCFirstApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Solution Explorer
Solution 'MVCFirstApp' (1 project)
MVCFirstApp
Properties
References
App_Data
App_Start
View in Browser (Internet Explorer) Ctrl+Shift+W
Browse With...
Multilingual App Toolkit
Add
Scope to This
New Solution Explorer View
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Rename
Open Folder in File Explorer
Properties Alt+Enter
Controller...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Azure API App SDK...
Azure API App Client...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
This item does not support previewing

It will display the Add Scaffold dialog.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel

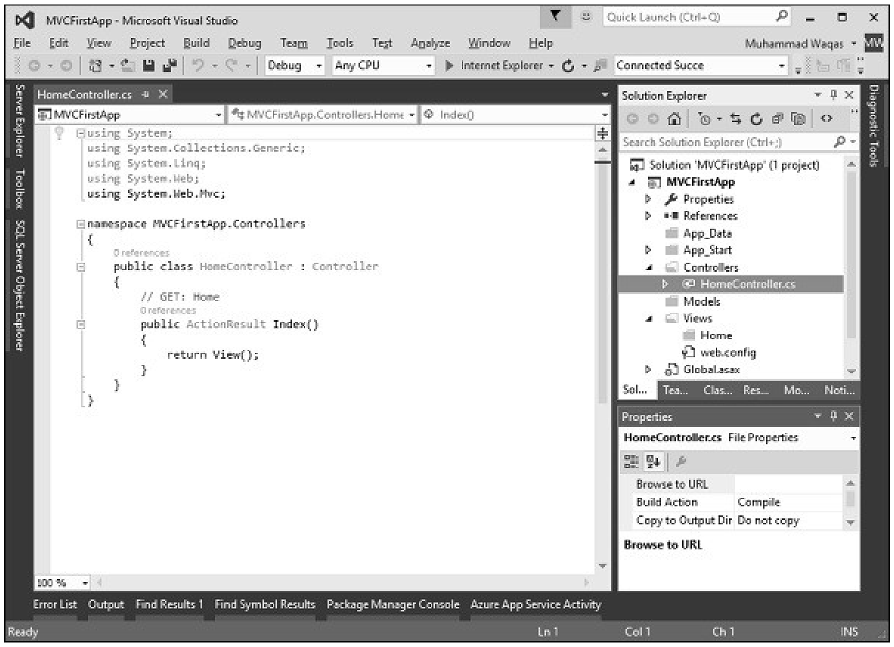
Step 2

Select the MVC 5 Controller – Empty option and click 'Add' button.
The Add Controller dialog will appear.o



Step 3

Set the name to HomeController and click the Add button.
You will see a new C# file HomeController.cs in the Controllers folder, which is open for editing in Visual Studio as well.


MVCFirstApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Internet Explorer
HomeController.cs
Solution Explorer
Properties
Ready

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCFirstApp.Controllers
{
    0 references
    public class HomeController : Controller
    {
        // GET: Home
        0 references
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

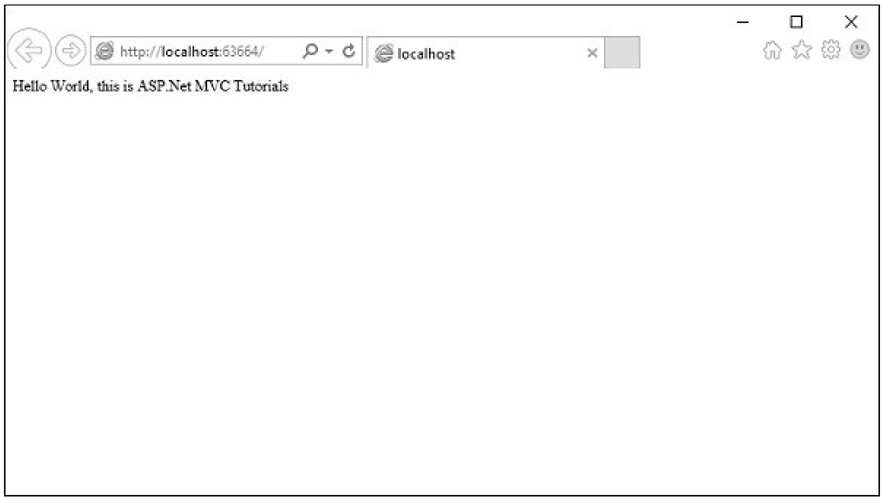
Step 4

To make this a working example, let’s modify the controller class by changing the action method called Index using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCFirstApp.Controllers {
   public class HomeController : Controller {
      // GET: Home
      public string Index(){
         return "Hello World, this is ASP.Net MVC Tutorials";
      }
   }
}
```

Step 5

Run this application and you will see that the browser is displaying the result of the Index action method.



```
http://localhost:63664/
Hello World, this is ASP.Net MVC Tutorials
```

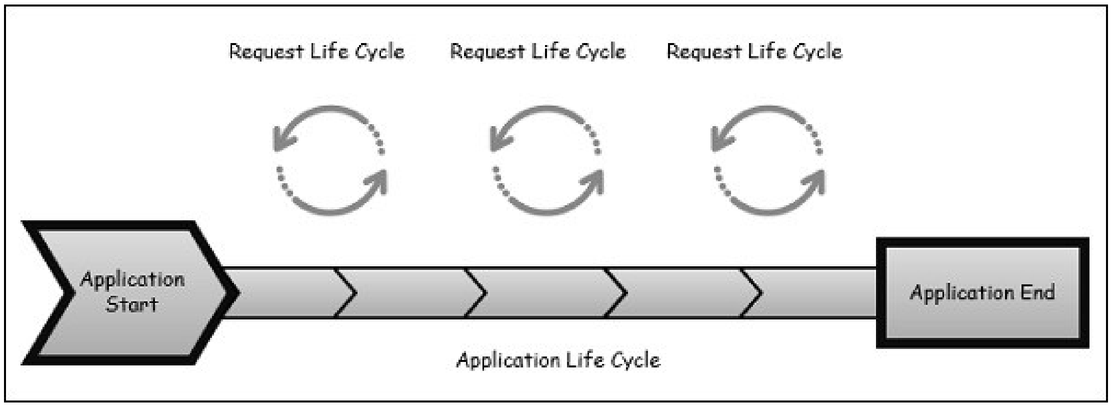
12.4. ASP.NET MVC - Life Cycle:

In this lecture, we will discuss the overall MVC pipeline and the life of an HTTP request as it travels through the MVC framework in ASP.NET. At a high level, a life cycle is simply a series of steps or events used to handle some type of request or to change an application state. You may already be familiar with various framework life cycles, the concept is not unique to MVC.

For example, the ASP.NET webforms platform features a complex page life cycle. Other .NET platforms, like Windows phone apps, have their own application life cycles. One thing that is true for all these platforms regardless of the technology is that understanding the processing pipeline can help you better leverage the features available and MVC is no different.

MVC has two life cycles −

- The application life cycle
- The request life cycle



12.4.1. The Application Life Cycle:

The application life cycle refers to the time at which the application process actually begins running IIS until the time it stops. This is marked by the application start and end events in the startup file of your application.

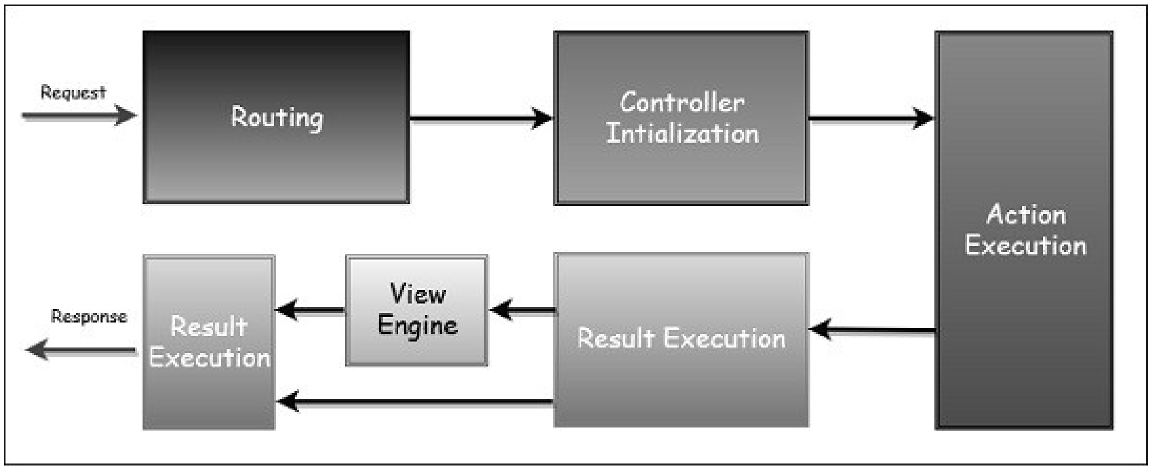
12.4.2. The Request Life Cycle:

It is the sequence of events that happen every time an HTTP request is handled by our application.

The entry point for every MVC application begins with routing. After the ASP.NET platform has received a request, it figures out how it should be handled through the URL Routing Module.

Modules are .NET components that can hook into the application life cycle and add functionality. The routing module is responsible for matching the incoming URL to routes that we define in our application.

All routes have an associated route handler with them and this is the entry point to the MVC framework.



The MVC framework handles converting the route data into a concrete controller that can handle requests. After the controller has been created, the next major step is Action Execution. A component called the action invoker finds and selects an appropriate Action method to invoke the controller.

After our action result has been prepared, the next stage triggers, which is Result Execution. MVC separates declaring the result from executing the result. If the result is a view type, the View Engine will be called and it's responsible for finding and rending our view.

If the result is not a view, the action result will execute on its own. This Result Execution is what generates an actual response to the original HTTP request.

12.5. ASP.NET MVC - Routing:

Routing is the process of directing an HTTP request to a controller and the functionality of this processing is implemented in System.Web.Routing. This assembly is not part of ASP.NET MVC. It is actually part of the ASP.NET runtime, and it was officially released with the ASP.NET as a .NET 3.5 SP1.

System.Web.Routing is used by the MVC framework, but it's also used by ASP.NET Dynamic Data. The MVC framework leverages routing to direct a request to a controller. The Global.asax file is that part of your application, where you will define the route for your application.

This is the code from the application start event in Global.asax from the MVC App which we created in the previous lecture.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MVCFirstApp {
  public class MvcApplication : System.Web.HttpApplication {
    protected void Application_Start(){
      AreaRegistration.RegisterAllAreas();
      RouteConfig.RegisterRoutes(RouteTable.Routes);
    }
  }
}
```

Following is the implementation of RouteConfig class, which contains one method RegisterRoutes.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MVCFirstApp {
   public class RouteConfig {
      public static void RegisterRoutes(RouteCollection routes){
         routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
         routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
                       defaults: new{ controller = "Home", action = "Index", id =
UrlParameter.Optional});
      }
   }
}
```

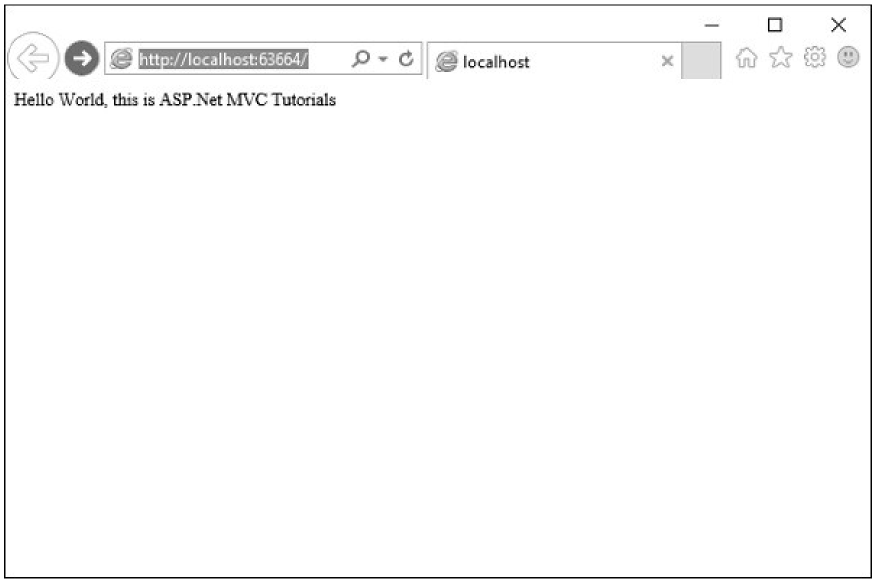
You will define the routes and those routes will map URLs to a specific controller action. An action is just a method on the controller. It can also pick parameters out of that URL and pass them as parameters into the method.

So this route that is defined in the application is the default route. As seen in the above code, when you see a URL arrive in the form of (something)/(something)/(something), then the first piece is the controller name, second piece is the action name, and the third piece is an ID parameter.

12.5.1. Understanding Routes:

MVC applications use the ASP.NET routing system, which decides how URLs map to controllers and actions.

When Visual Studio creates the MVC project, it adds some default routes to get us started. When you run your application, you will see that Visual Studio has directed the browser to port 63664. You will almost certainly see a different port number in the URL that your browser requests because Visual Studio allocates a random port when the project is created.



```
Hello World, this is ASP.Net MVC Tutorials
```

In the last example, we have added a HomeController, so you can also request any of the following URLs, and they will be directed to the Index action on the HomeController.

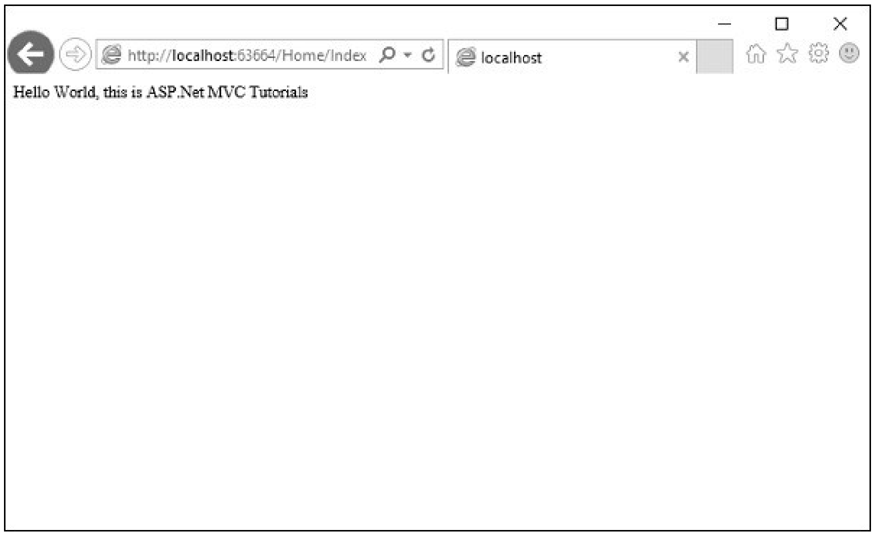
http://localhost:63664/Home/

http://localhost:63664/Home/Index

When a browser requests http://mysite/ or http://mysite/Home, it gets back the output from HomeController's Index method.

You can try this as well by changing the URL in the browser. In this example, it is http://localhost:63664/, except that the port might be different.

If you append /Home or /Home/Index to the URL and press 'Enter' button, you will see the same result from the MVC application.



```
Hello World, this is ASP.Net MVC Tutorials
```

As you can see in this case, the convention is that we have a controller called HomeController and this HomeController will be the starting point for our MVC application.

The default routes that Visual Studio creates for a new project assumes that you will follow this convention. But if you want to follow your own convention then you would need to modify the routes.

12.5.2. Custom Convention:

You can certainly add your own routes. If you don't like these action names, if you have different ID parameters or if you just in general have a different URL structure for your site, then you can add your own route entries.

Let’s take a look at a simple example. Consider we have a page that contains the list of processes. Following is the code, which will route to the process page.

```
routes.MapRoute(
  "Process",
  "Process/{action}/{id}",
  defaults: new{
    controller = "Process", action = "List ", id = UrlParameter.Optional}
);
```

When someone comes in and looks for a URL with Process/Action/Id, they will go to the Process Controller. We can make the action a little bit different, the default action, we can make that a List instead of Index.

Now a request that arrives looks like localhosts/process. The routing engine will use this routing configuration to pass that along, so it's going to use a default action of List.

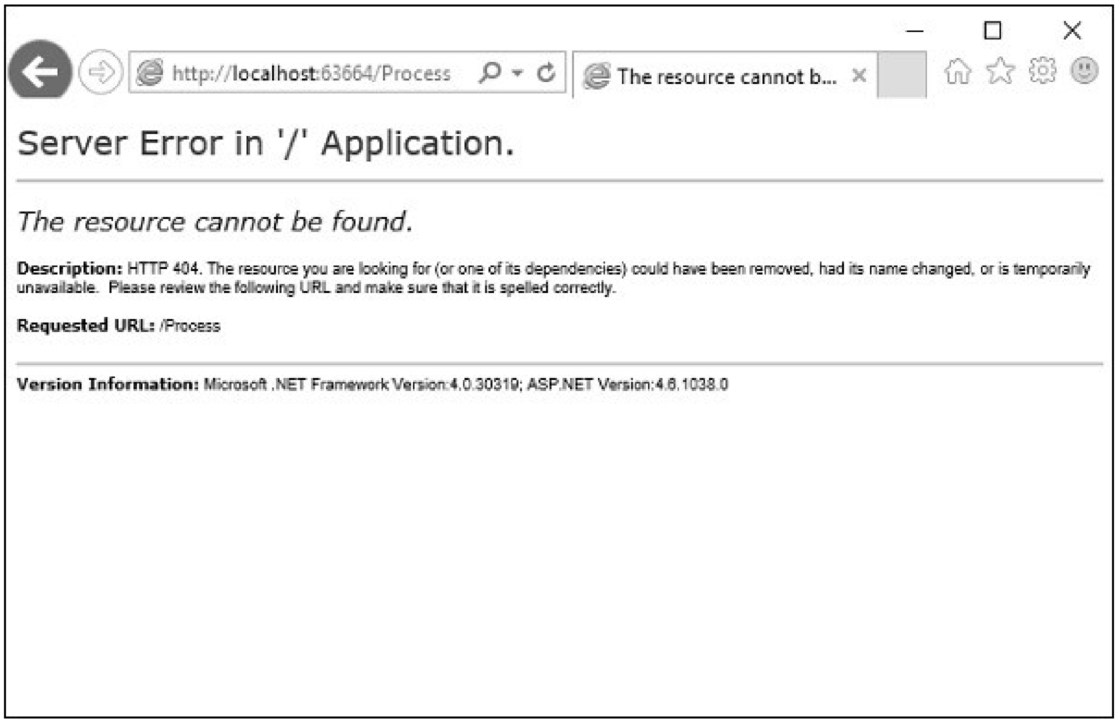
Following is the complete class implementation.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MVCFirstApp{
  public class RouteConfig{
    public static void RegisterRoutes(RouteCollection routes){
      routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

```
            routes.MapRoute(
              "Process", "Process/{action}/{id}",
              defaults: new{
                controller = " Process", action = "List ", id =
                UrlParameter.Optional});

            routes.MapRoute(
              name: "Default", url: "{controller}/{action}/{id}",
              defaults: new{
                controller = "Home", action = "Index", id =
                UrlParameter.Optional});
          }
        }
      }
```

Step 1

Run this and request for a process page with the following URL http://localhost:63664/Process



http://localhost:63664/Process

The resource cannot b...

Server Error in '/' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, had its name changed, or is temporarily unavailable. Please review the following URL and make sure that it is spelled correctly.

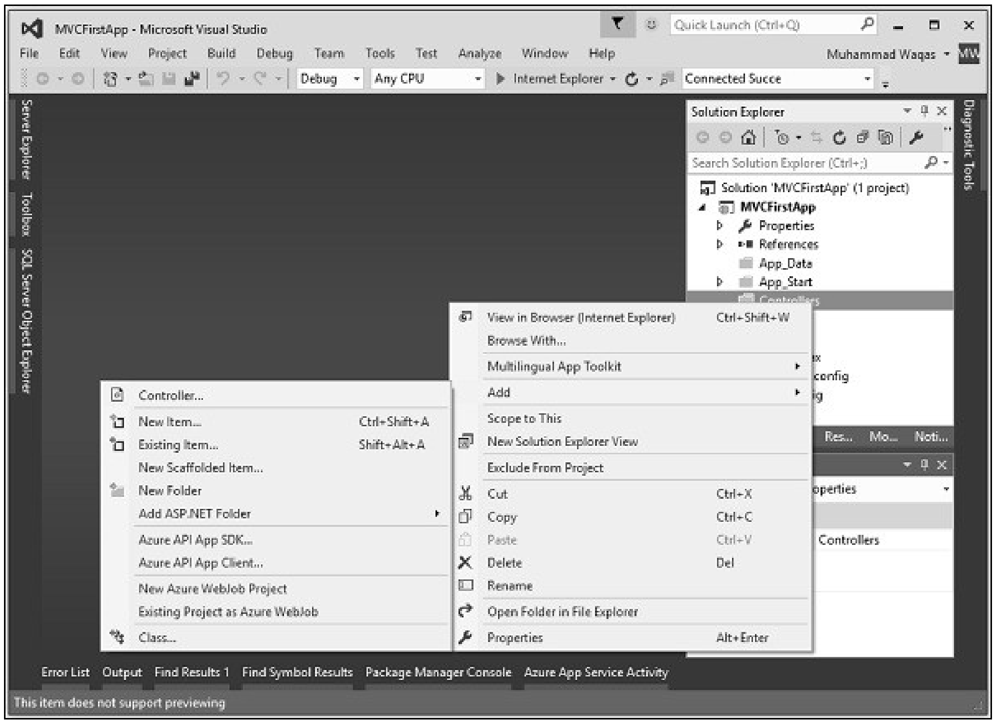
Requested URL: /Process

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.6.1038.0

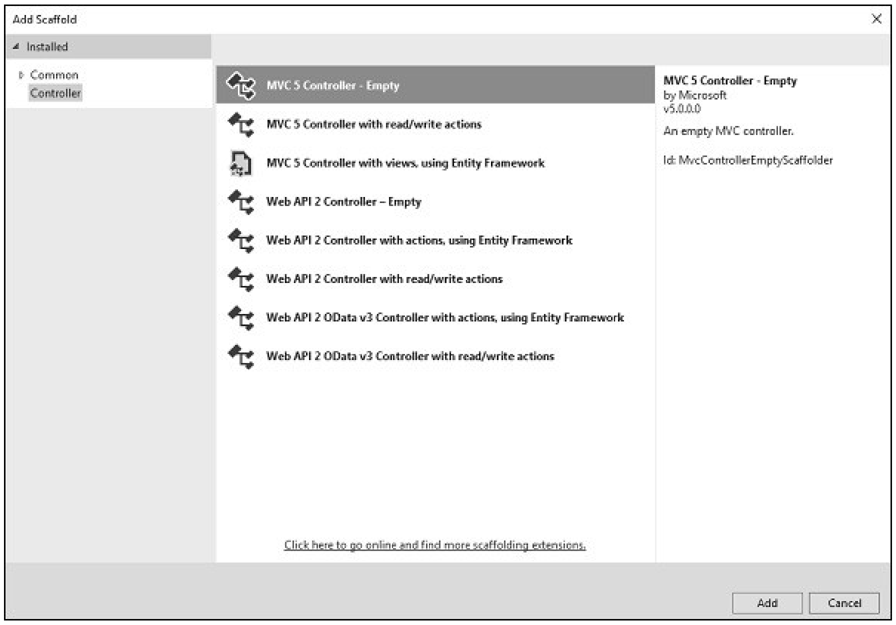
You will see an HTTP 404, because the routing engine is looking for ProcessController, which is not available.

Step 2

Create ProcessController by right-clicking on Controllers folder in the solution explorer and select Add → Controller.

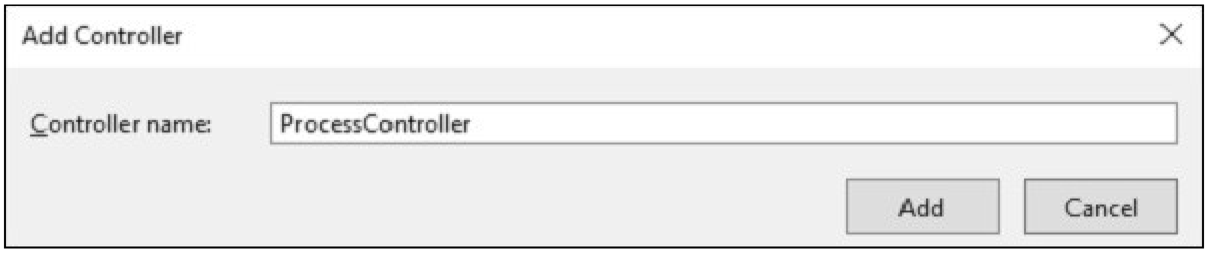

MVCFirstApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Solution Explorer
Solution 'MVCFirstApp' (1 project)
MVCFirstApp
Properties
References
App_Data
App_Start
Controllers
View in Browser (Internet Explorer) Ctrl+Shift+W
Browse With...
Multilingual App Toolkit
Add
Scope to This
New Solution Explorer View
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Rename
Open Folder in File Explorer
Properties Alt+Enter
Controller...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Azure API App SDK...
Azure API App Client...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
This item does not support previewing

It will display the Add Scaffold dialog.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel

Step 3

Select the MVC 5 Controller – Empty option and click 'Add' button.
The Add Controller dialog will appear.

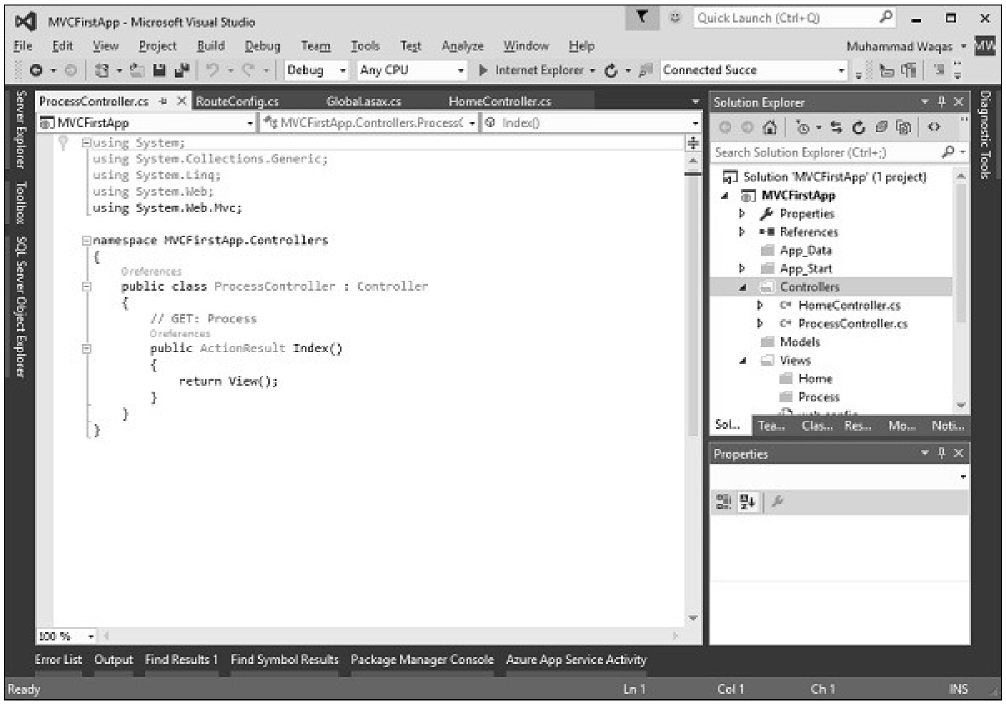

Add Controller

Controller name: ProcessController

Step 4

Set the name to ProcessController and click 'Add' button.

Now you will see a new C# file ProcessController.cs in the Controllers folder, which is open for editing in Visual Studio as well.


MVCFirstApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Internet Explorer
ProcessController.cs
RouteConfig.cs
Global.asax.cs
HomeController.cs
Solution Explorer
Solution 'MVCFirstApp' (1 project)
MVCFirstApp
Properties
References
App_Data
App_Start
Controllers
HomeController.cs
ProcessController.cs
Models
Views
Home
Process
Properties
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Ready

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCFirstApp.Controllers
{
    public class ProcessController : Controller
    {
        // GET: Process
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Now our default action is going to be List, so we want to have a List action here instead of Index.

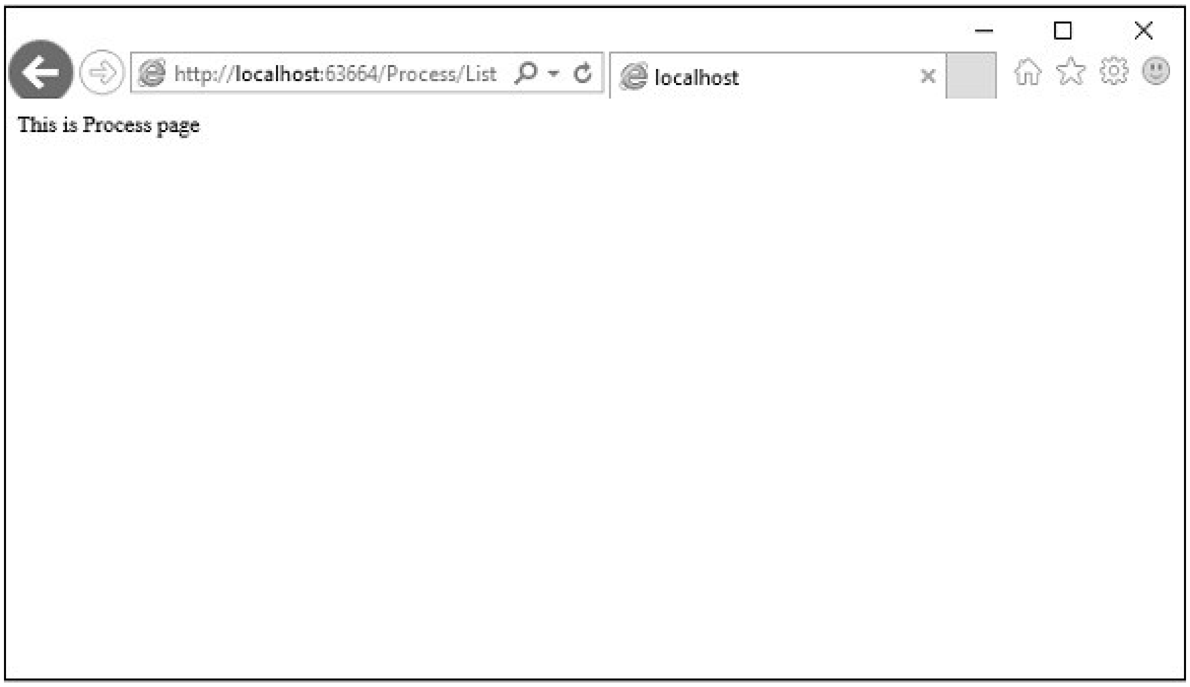
Step 5

Change the return type from ActionResult to string and also return some string from this action method using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCFirstApp.Controllers{
  public class ProcessController : Controller{
    // GET: Process
    public string List(){
      return "This is Process page";
    }
  }
}
```

Step 6

When you run this application, again you will see the result from the default route. When you specify the following URL, http://localhost:63664/Process/List, then you will see the result from the ProcessController.



```
This is Process page
```

12.6. ASP.NET MVC – Controllers:

Controllers are essentially the central unit of your ASP.NET MVC application. It is the 1st recipient, which interacts with incoming HTTP Request. So, the controller decides which model will be selected, and then it takes the data from the model and passes the same to the respective view, after that view is rendered. Actually, controllers are controlling the overall flow of the application taking the input and rendering the proper output.

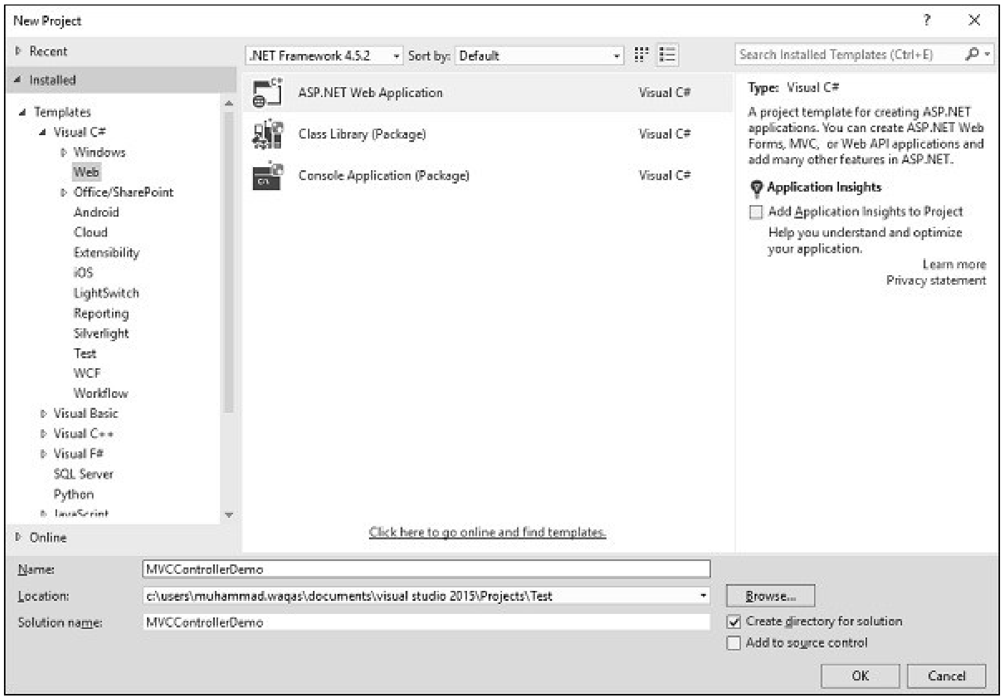
Controllers are C# classes inheriting from System.Web.Mvc.Controller, which is the builtin controller base class. Each public method in a controller is known as an action method, meaning you can invoke it from the Web via some URL to perform an action.

The MVC convention is to put controllers in the Controllers folder that Visual Studio created when the project was set up.

Let's take a look at a simple example of Controller by creating a new ASP.Net MVC project.

Step 1

Open the Visual Studio and click on File → New → Project menu option.
A new Project dialog opens.



New Project

.NET Framework 4.5.2 Sort by: Default

ASP.NET Web Application	Visual C#
Class Library (Package)	Visual C#
Console Application (Package)	Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

Add Application Insights to Project

Help you understand and optimize your application.

Name: MVCControllerDemo

Location: c:\users\muhammad.waqas\documents\visual studio 2015\Projects\Test

Solution name: MVCControllerDemo

Create directory for solution ✓

Add to source control

Step 2

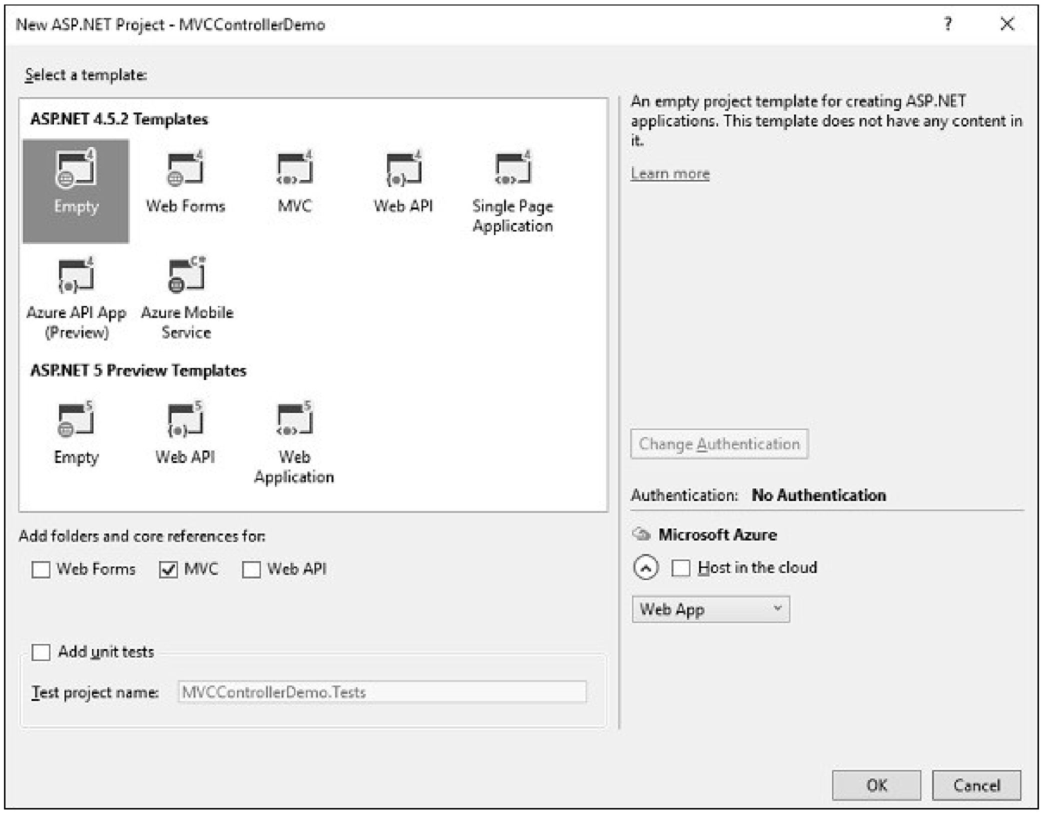
From the left pane, select Templates → Visual C# → Web.

Step 3

In the middle pane, select ASP.NET Web Application.

Step 4

Enter the project name ‘MVCControllerDemo’ in the Name field and click ok to continue. You will see the following dialog, which asks you to set the initial content for the ASP.NET project.



An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Add folders and core references for:

- [ ] Web Forms
- [✓] MVC
- [ ] Web API

- [ ] Add unit tests

Test project name: MVCControllerDemo.Tests

- [ ] Host in the cloud

Web App

Step 5

To keep things simple, select the Empty option and check the MVC checkbox in the ‘Add folders and core references for’ section and click Ok.

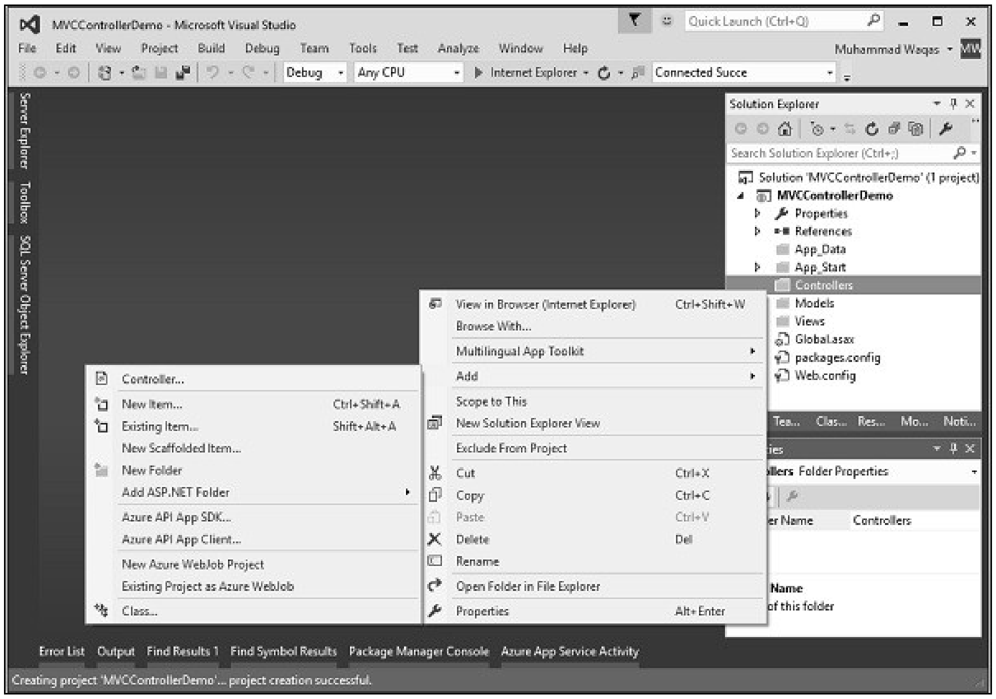
It will create a basic MVC project with minimal predefined content.

Once the project is created by Visual Studio you will see a number of files and folders displayed in the Solution Explorer window.

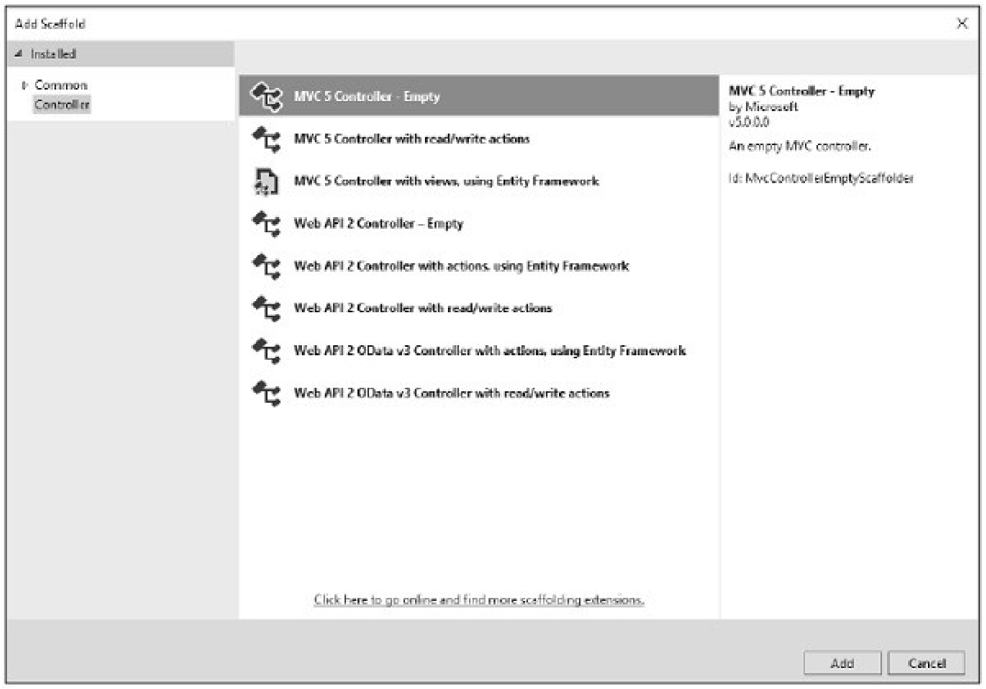
Since we have created ASP.Net MVC project from an empty project template, so at the moment, the application does not contain anything to run.

Step 6

Add EmployeeController by right-clicking on Controllers folder in the solution explorer. Select Add → Controller.


MVCControllerDemo - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Solution Explorer
Solution 'MVCControllerDemo' (1 project)
MVCControllerDemo
Properties
References
App_Data
App_Start
Controllers
Models
Views
Global.asax
packages.config
Web.config
View in Browser (Internet Explorer) Ctrl+Shift+W
Browse With...
Multilingual App Toolkit
Add
Scope to This
New Solution Explorer View
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Rename
Open Folder in File Explorer
Properties Alt+Enter
Controller...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Azure API App SDK...
Azure API App Client...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Creating project 'MVCControllerDemo'... project creation successful.

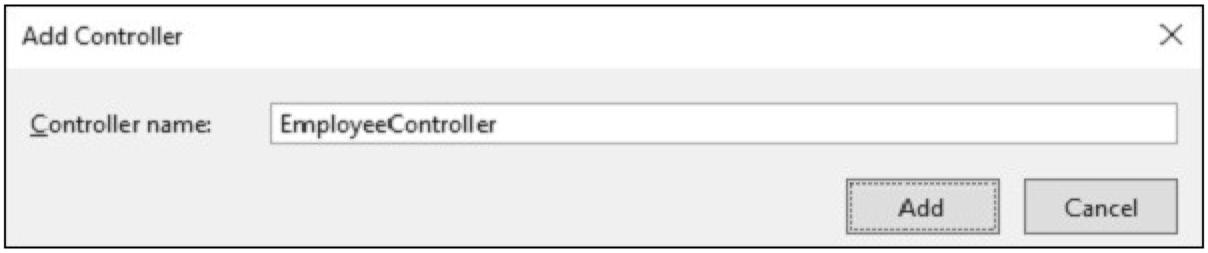
It will display the Add Scaffold dialog.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel

Step 7

Select the MVC 5 Controller – Empty option and click 'Add' button.

The Add Controller dialog will appear.

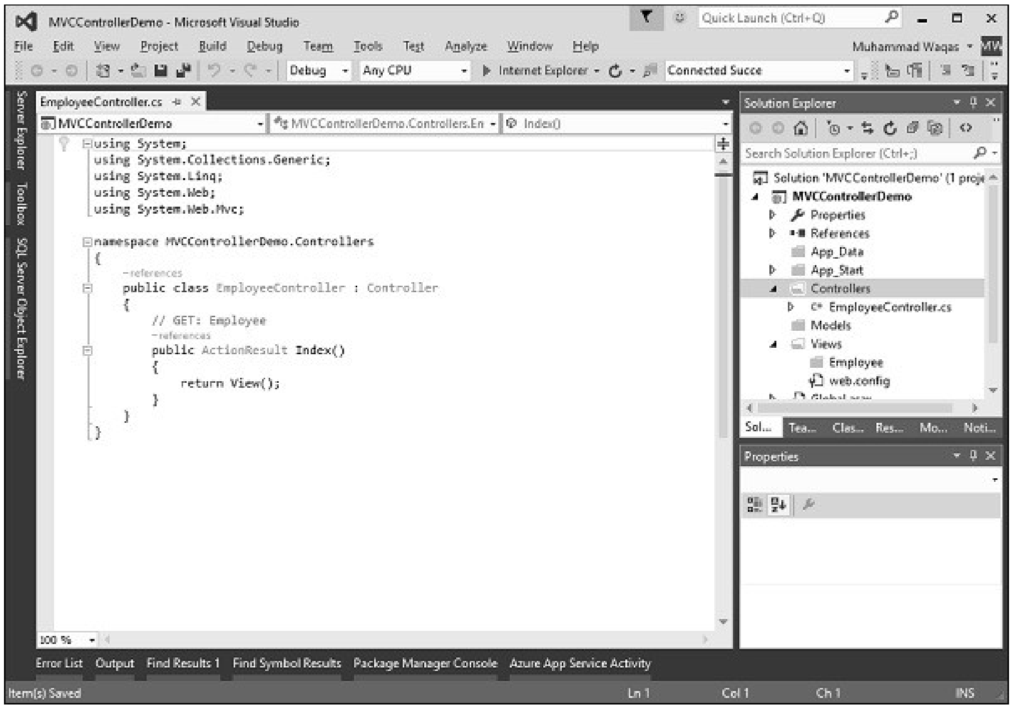

Add Controller

Controller name: EmployeeController

Step 8

Set the name to EmployeeController and click 'Add' button.

You will see a new C# file EmployeeController.cs in the Controllers folder, which is open for editing in Visual Studio as well.


MVCControllerDemo - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Internet Explorer
EmployeeController.cs
MVCControllerDemo
MVCControllerDemo.Controllers.En
Index()
Solution Explorer
Solution 'MVCControllerDemo'
MVCControllerDemo
Properties
References
App_Data
App_Start
Controllers
EmployeeController.cs
Models
Views
Employee
web.config
Properties
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Item(s) Saved
Ln 1 Col 1 Ch 1 INS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers
{
    public class EmployeeController : Controller
    {
        // GET: Employee
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Now, in this application we will add a custom route for Employee controller with the default Route.

Step 1

Go to "RouteConfig.cs" file under "App_Start" folder and add the following route.

```
routes.MapRoute(
  "Employee", "Employee/{name}", new{
    controller = "Employee", action = "Search", name =
    UrlParameter.Optional });
```

Following is the complete implementation of RouteConfig.cs file.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MVCControllerDemo {
  public class RouteConfig {
    public static void RegisterRoutes(RouteCollection routes){
      routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

      routes.MapRoute(
        "Employee", "Employee/{name}", new{
          controller = "Employee", action = "Search", name = UrlParameter.Optional });

      routes.MapRoute(
        name: "Default", url: "{controller}/{action}/{id}", defaults: new{
          controller = "Home", action = "Index", id = UrlParameter.Optional });
    }
  }
}
```

Consider a scenario wherein any user comes and searches for an employee, specifying the URL “Employee/Mark”. In this case, Mark will be treated as a parameter name not like Action method. So in this kind of scenario our default route won’t work significantly.

To fetch the incoming value from the browser when the parameter is getting passed, MVC framework provides a simple way to address this problem. It is by using the parameter inside the Action method.

Step 2

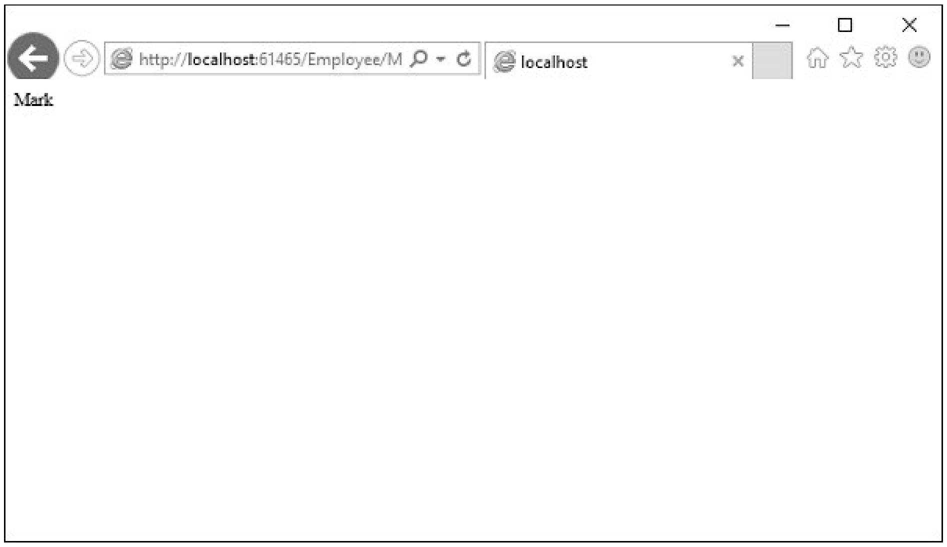
Change the EmployeeController class using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers  {
  public class EmployeeController : Controller {
    // GET: Employee
    public ActionResult Search(string name){
      var input = Server.HtmlEncode(name);
      return Content(input);
    }
  }
}
```

If you add a parameter to an action method, then the MVC framework will look for the value that matches the parameter name. It will apply all the possible combination to find out the parameter value. It will search in the Route data, query string, etc.

Hence, if you request for /Employee/Mark", then the MVC framework will decide that I need a parameter with "UserInput", and then Mark will get picked from the URL and that will get automatically passed.

Server.HtmlEncode will simply convert any kind of malicious script in plain text. When the above code is compiled and executed and requests the following URL http://localhost:61465/Employee/Mark, you will get the following output.



```
Mark
```

As you can see in the above screenshot, Mark is picked from the URL.

12.7. ASP.NET MVC – Action Methods:

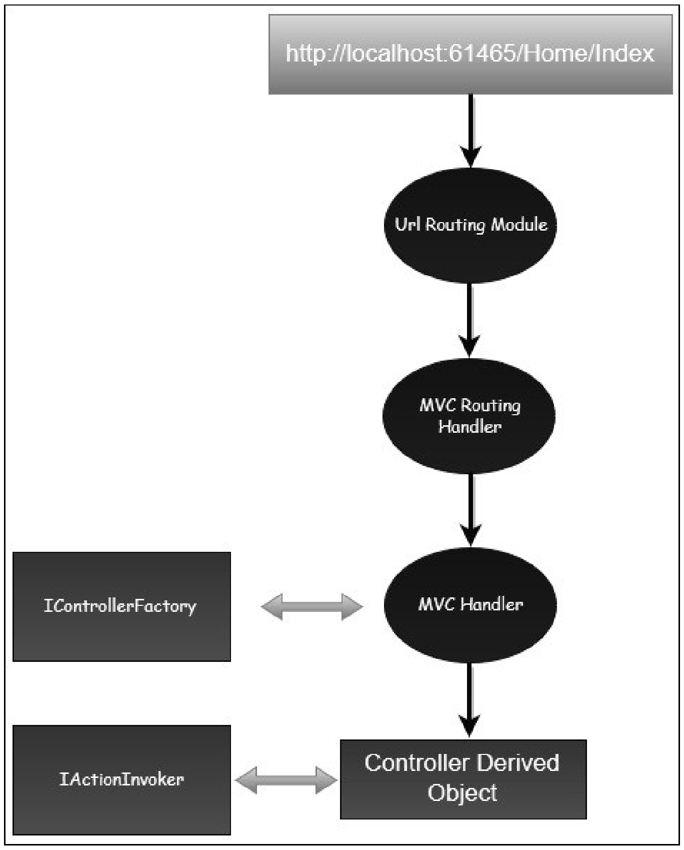
ASP.NET MVC Action Methods are responsible to execute requests and generate responses to it. By default, it generates a response in the form of ActionResult. Actions typically have a one-to-one mapping with user interactions.

For example, enter a URL into the browser, click on any particular link, and submit a form, etc. Each of these user interactions causes a request to be sent to the server. In each case, the URL of the request includes information that the MVC framework uses to invoke an action method. The one restriction on action method is that they have to be instance method, so they cannot be static methods. Also there is no return value restrictions. So you can return the string, integer, etc.

12.7.1. Request Processing:

Actions are the ultimate request destination in an MVC application and it uses the controller base class. Let's take a look at the request processing.

- When a URL arrives, like /Home/index, it is the UrlRoutingModule that inspects and understands that something configured within the routing table knows how to handle that URL.



- The UrlRoutingModule puts together the information we've configured in the routing table and hands over control to the MVC route handler.
- The MVC route handler passes the controller over to the MvcHandler which is an HTTP handler.
- MvcHandler uses a controller factory to instantiate the controller and it knows what controller to instantiate because it looks in the RouteData for that controller value.
- Once the MvcHandler has a controller, the only thing that MvcHandler knows about is IController Interface, so it simply tells the controller to execute.
- When it tells the controller to execute, that's been derived from the MVC's controller base class. The Execute method creates an action invoker and tells that action invoker to go and find a method to invoke, find an action to invoke.
- The action invoker, again, looks in the RouteData and finds that action parameter that's been passed along from the routing engine.

12.7.2. Types of Action:

Actions basically return different types of action results. The ActionResult class is the base for all action results. Following is the list of different kind of action results and its behavior.

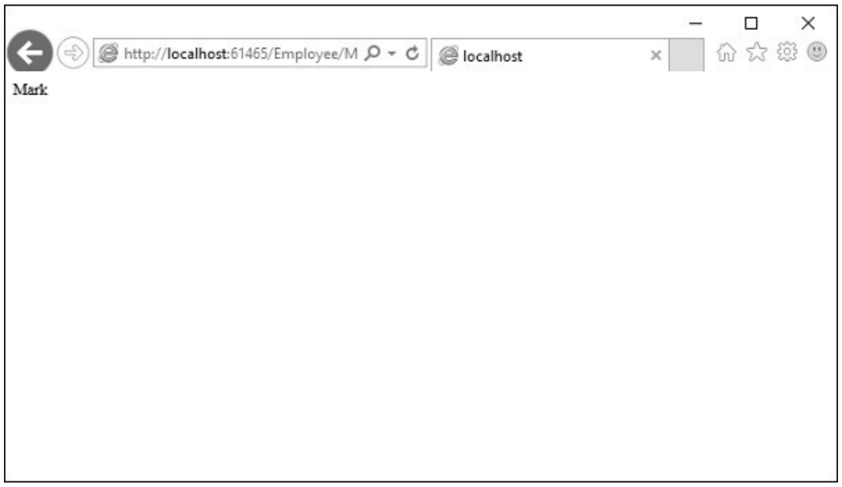
Sr.No.	Name and Behavior
1	**ContentResult** Returns a string
2	**FileContentResult** Returns file content
3	**FilePathResult** Returns file content
4	**FileStreamResult** Returns file content
5	**EmptyResult** Returns nothing
6	**JavaScriptResult** Returns script for execution
7	**JsonResult** Returns JSON formatted data
8	**RedirectToResult** Redirects to the specified URL
9	**HttpUnauthorizedResult** Returns 403 HTTP Status code
10	**RedirectToRouteResult** Redirects to different action/different controller action
11	**ViewResult** Received as a response for view engine
12	**PartialViewResult** Received as a response for view engine

Let’s have a look at a simple example from the previous chapter in which we have created an EmployeeController.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {
   public class EmployeeController : Controller{
      // GET: Employee
```

```
        public ActionResult Search(string name){
          var input = Server.HtmlEncode(name);
          return Content(input);
        }
      }
    }
```

When you request the following URL http://localhost:61465/Employee/Mark, then you will receive the following output as an action.



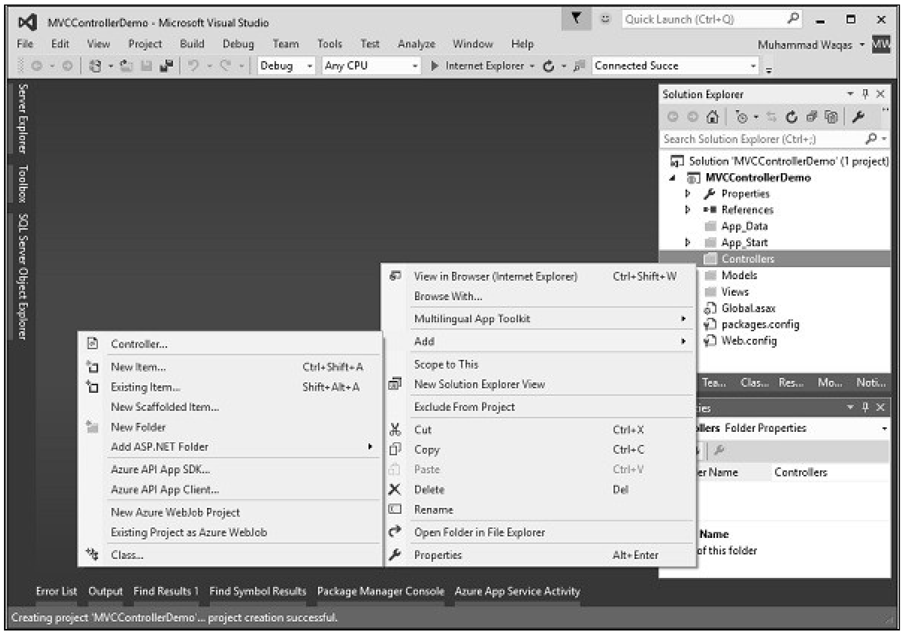
Mark

12.7.3. Add Controller:

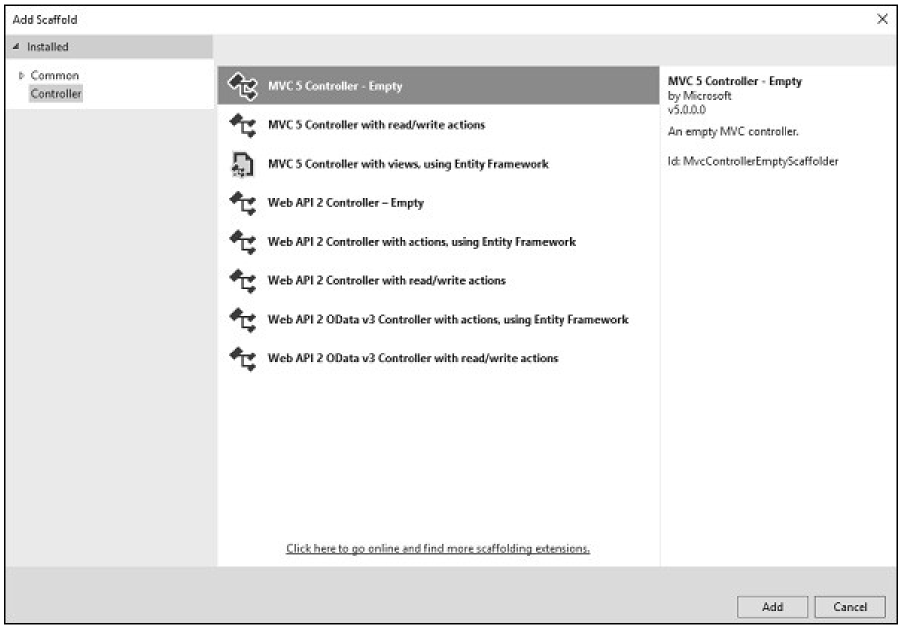
Let us add one another controller.

Step 1

Right-click on Controllers folder and select Add → Controller.

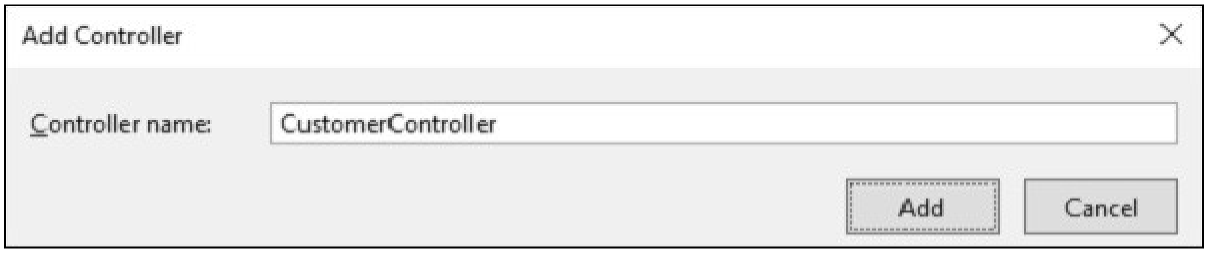

MVCControllerDemo - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Solution Explorer
Solution 'MVCControllerDemo' (1 project)
MVCControllerDemo
Properties
References
App_Data
App_Start
Controllers
Models
Views
Global.asax
packages.config
Web.config
Controller...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
Add ASP.NET Folder
Azure API App SDK...
Azure API App Client...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
View in Browser (Internet Explorer) Ctrl+Shift+W
Browse With...
Multilingual App Toolkit
Add
Scope to This
New Solution Explorer View
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Rename
Open Folder in File Explorer
Properties Alt+Enter
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Creating project 'MVCControllerDemo'... project creation successful.

It will display the Add Scaffold dialog.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel

Step 2

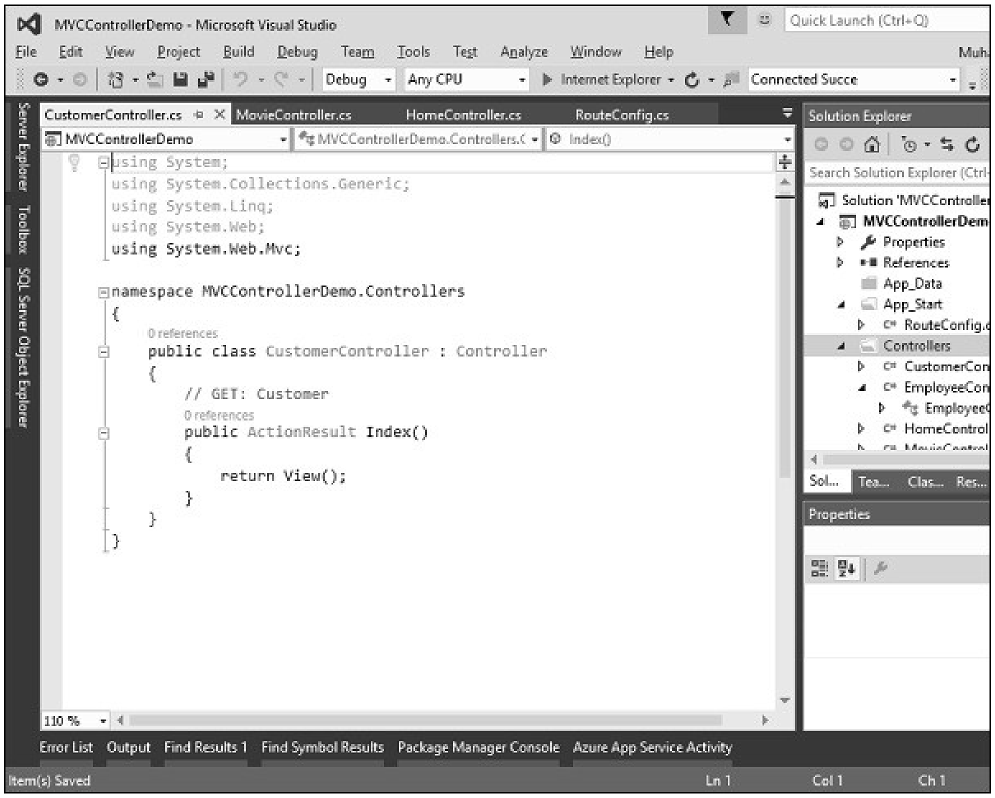
Select the MVC 5 Controller – Empty option and click ‘Add’ button.
The Add Controller dialog will appear.



Step 3

Set the name to CustomerController and click ‘Add’ button.

Now you will see a new C# file ‘CustomerController.cs’ in the Controllers folder, which is open for editing in Visual Studio as well.



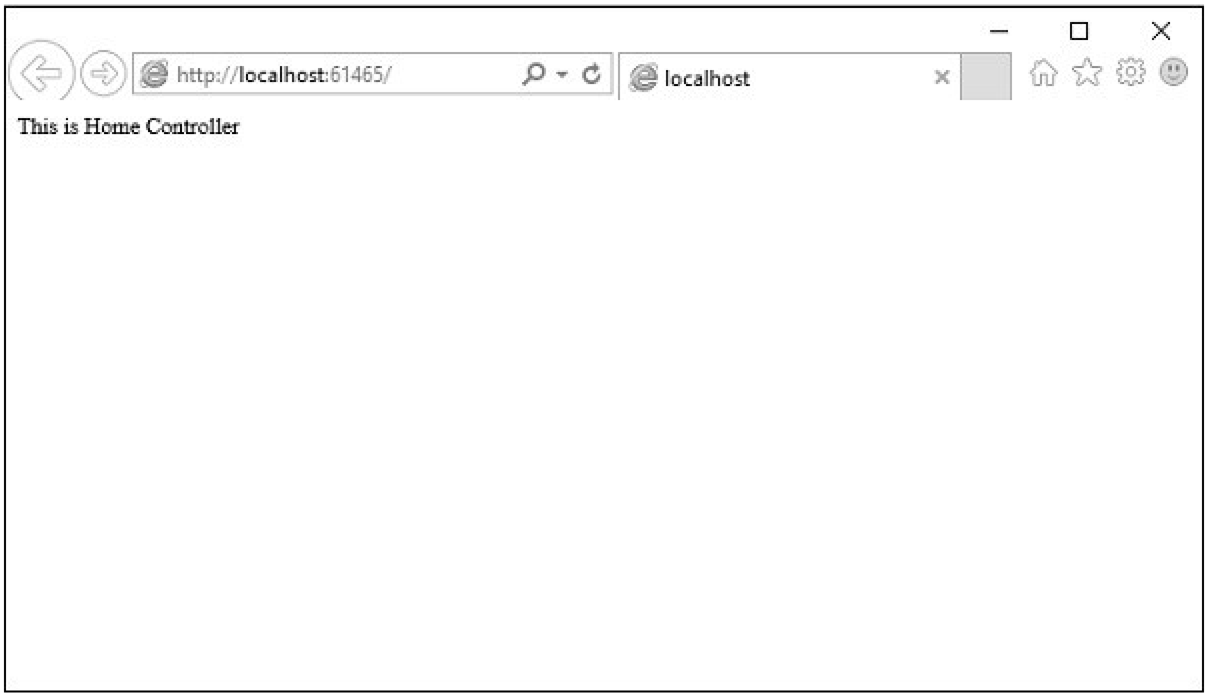
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers
{
    0 references
    public class CustomerController : Controller
    {
        // GET: Customer
        0 references
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Similarly, add one more controller with name HomeController. Following is the HomeController.cs class implementation.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {
   public class HomeController : Controller{
      // GET: Home
      public string Index(){
         return "This is Home Controller";
      }
   }
}
```

Step 4

Run this application and you will receive the following output.



http://localhost:61465/

localhost

This is Home Controller

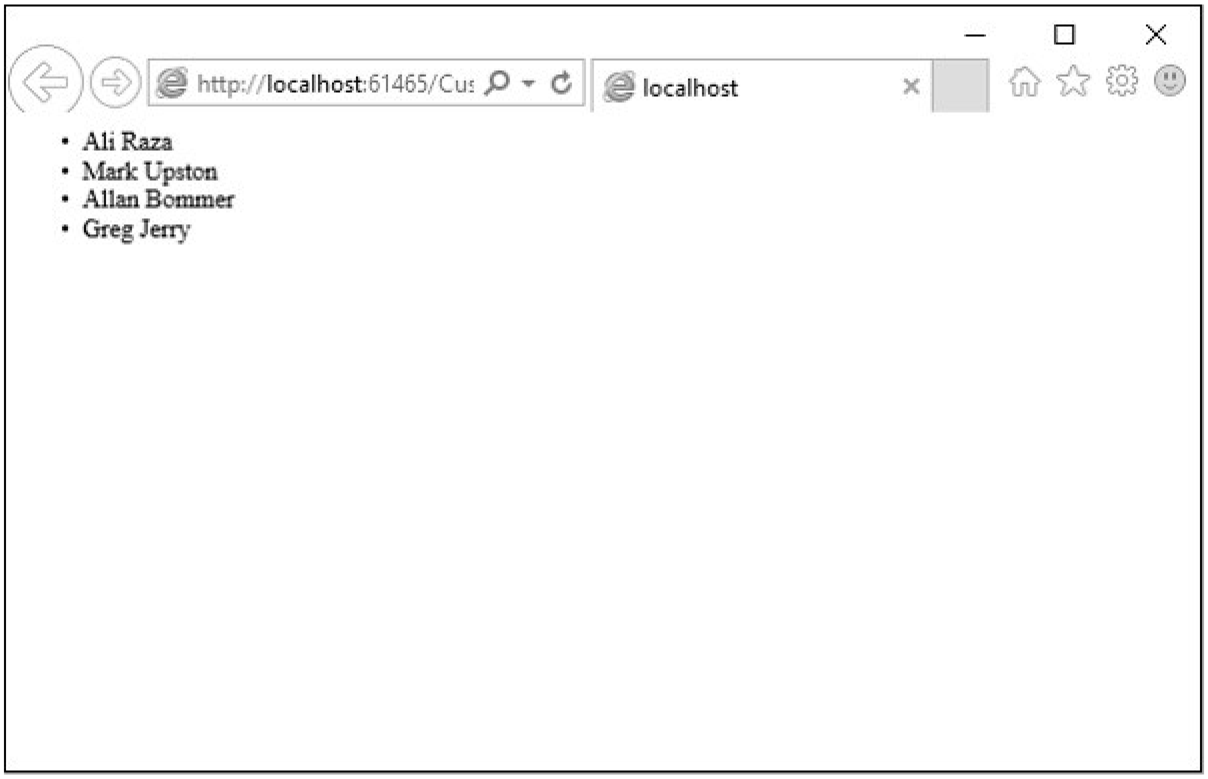
Step 5

Add the following code in Customer controller, which we have created above.

```
public string GetAllCustomers(){
   return @"<ul>
<li>Ali Raza</li>
<li>Mark Upston</li>
<li>Allan Bommer</li>
<li>Greg Jerry</li>
</ul>";
}
```

Step 6

Run this application and request for **http://localhost:61465/Customer/GetAllCustomers**. You will see the following output.



- Ali Raza
- Mark Upston
- Allan Bommer
- Greg Jerry

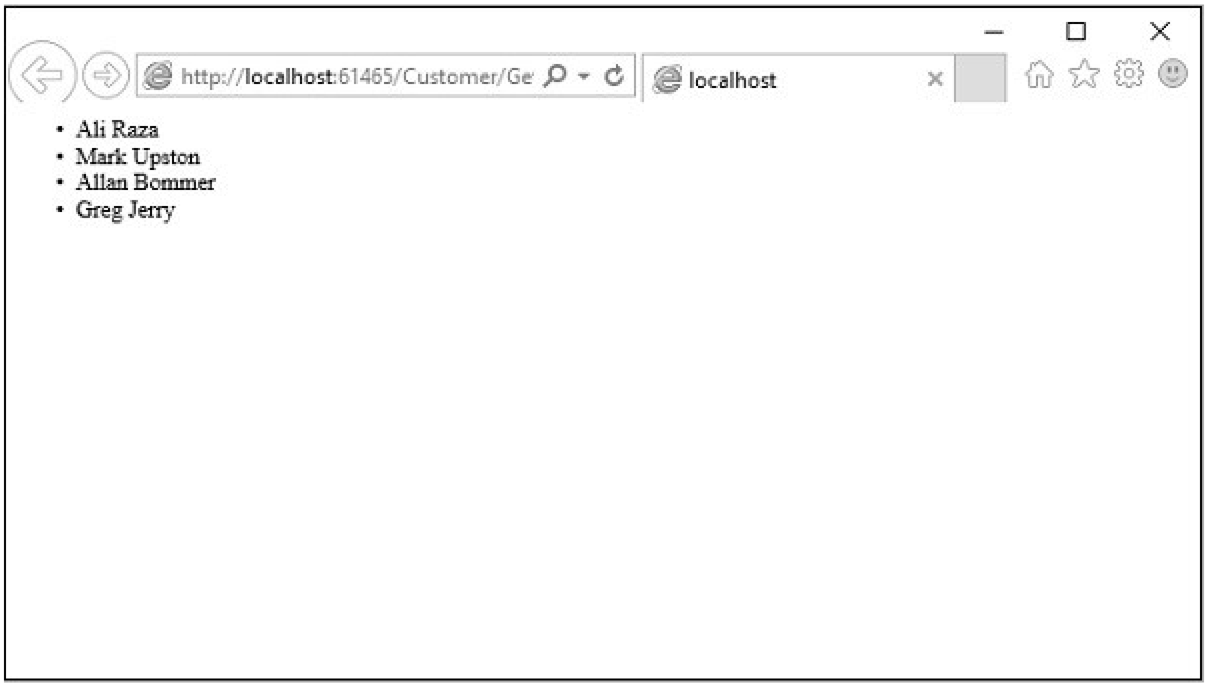
You can also redirect to actions for the same controller or even for a different controller.

Following is a simple example in which we will redirect from HomeController to Customer Controller by changing the code in HomeController using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers{
  public class HomeController : Controller{
        // GET: Home
        public ActionResult Index(){
          return RedirectToAction("GetAllCustomers","Customer");
        }
      }
    }
```

As you can see, we have used the RedirectToAction() method ActionResult, which takes two parameters, action name and controller name.

When you run this application, you will see the default route will redirect it to /Customer/GetAllCustomers



- Ali Raza
- Mark Upston
- Allan Bommer
- Greg Jerry

12.8. ASP.NET MVC – Filters:

In ASP.NET MVC, controllers define action methods that usually have a one-to-one relationship with possible user interactions, but sometimes you want to perform logic either before an action method is called or after an action method runs.

To support this, ASP.NET MVC provides filters. Filters are custom classes that provide both a declarative and programmatic means to add pre-action and post-action behavior to controller action methods.

12.8.1. Action Filters:

An action filter is an attribute that you can apply to a controller action or an entire controller that modifies the way in which the action is executed. The ASP.NET MVC framework includes several action filters −

- OutputCache − Caches the output of a controller action for a specified amount of time.
- HandleError − Handles errors raised when a controller action is executed.
- Authorize − Enables you to restrict access to a particular user or role.

12.8.2. Types of Filters:

The ASP.NET MVC framework supports four different types of filters −

- Authorization Filters − Implements the IAuthorizationFilter attribute.
- Action Filters − Implements the IActionFilter attribute.
- Result Filters − Implements the IResultFilter attribute.
- Exception Filters − Implements the IExceptionFilter attribute.

Filters are executed in the order listed above. For example, authorization filters are always executed before action filters and exception filters are always executed after every other type of filter.

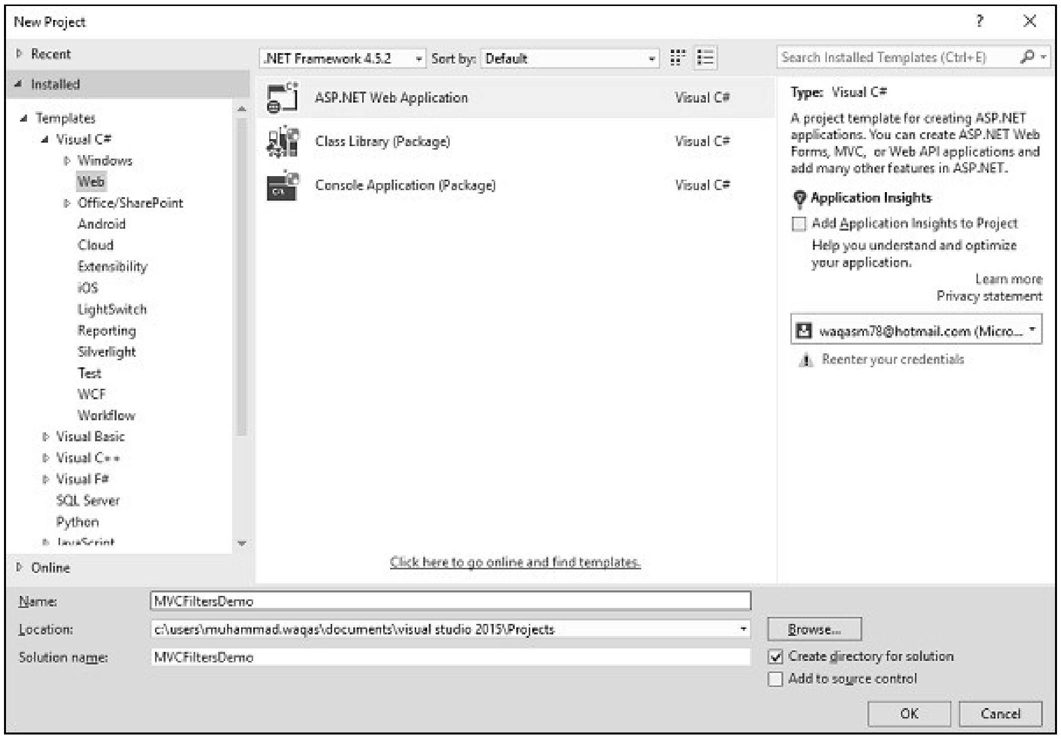
Authorization filters are used to implement authentication and authorization for controller actions. For example, the Authorize filter is an example of an Authorization filter.

Let’s take a look at a simple example by creating a new ASP.Net MVC project.

Step 1

Open the Visual Studio and click File → New → Project menu option.

A new Project dialog opens.



.NET Framework 4.5.2 Sort by: Default

ASP.NET Web Application	Visual C#
Class Library (Package)	Visual C#
Console Application (Package)	Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to Project

Help you understand and optimize your application.

Learn more

Privacy statement

waqasm78@hotmail.com (Micro...

Reenter your credentials

Name: MVCFiltersDemo

Location: c:\users\muhammad.waqas\documents\visual studio 2015\Projects

Solution name: MVCFiltersDemo

☑ Create directory for solution

☐ Add to source control

Step 2

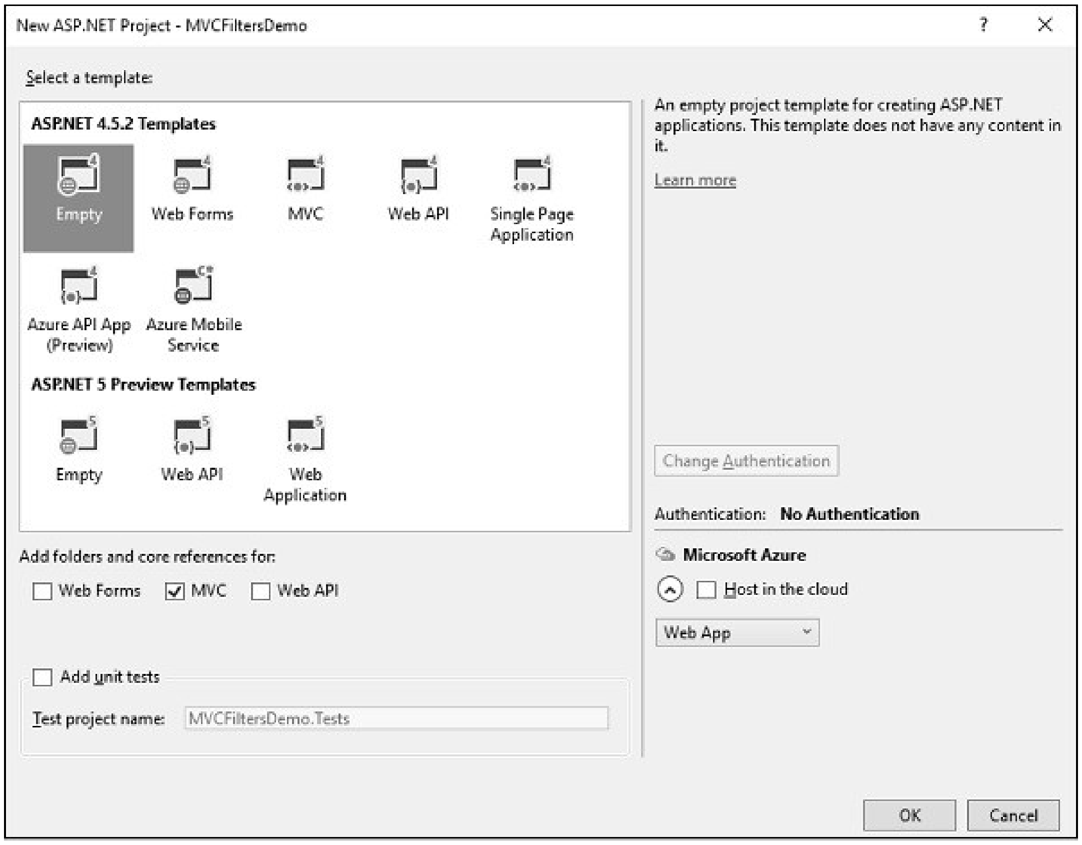
From the left pane, select Templates → Visual C# → Web.

Step 3

In the middle pane, select ASP.NET Web Application.

Step 4

Enter project name MVCFiltersDemo in the Name field and click ok to continue and you will see the following dialog which asks you to set the initial content for the ASP.NET project.


New ASP.NET Project - MVCFiltersDemo

Select a template:

ASP.NET 4.5.2 Templates

Empty, Web Forms, MVC, Web API, Single Page Application, Azure API App (Preview), Azure Mobile Service

ASP.NET 5 Preview Templates

Empty, Web API, Web Application

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Learn more

Change Authentication

Authentication: **No Authentication**

Microsoft Azure

☐ Host in the cloud

Web App

Add folders and core references for:

☐ Web Forms ☑ MVC ☐ Web API

☐ Add unit tests

Test project name: MVCFiltersDemo.Tests

OK Cancel

Step 5

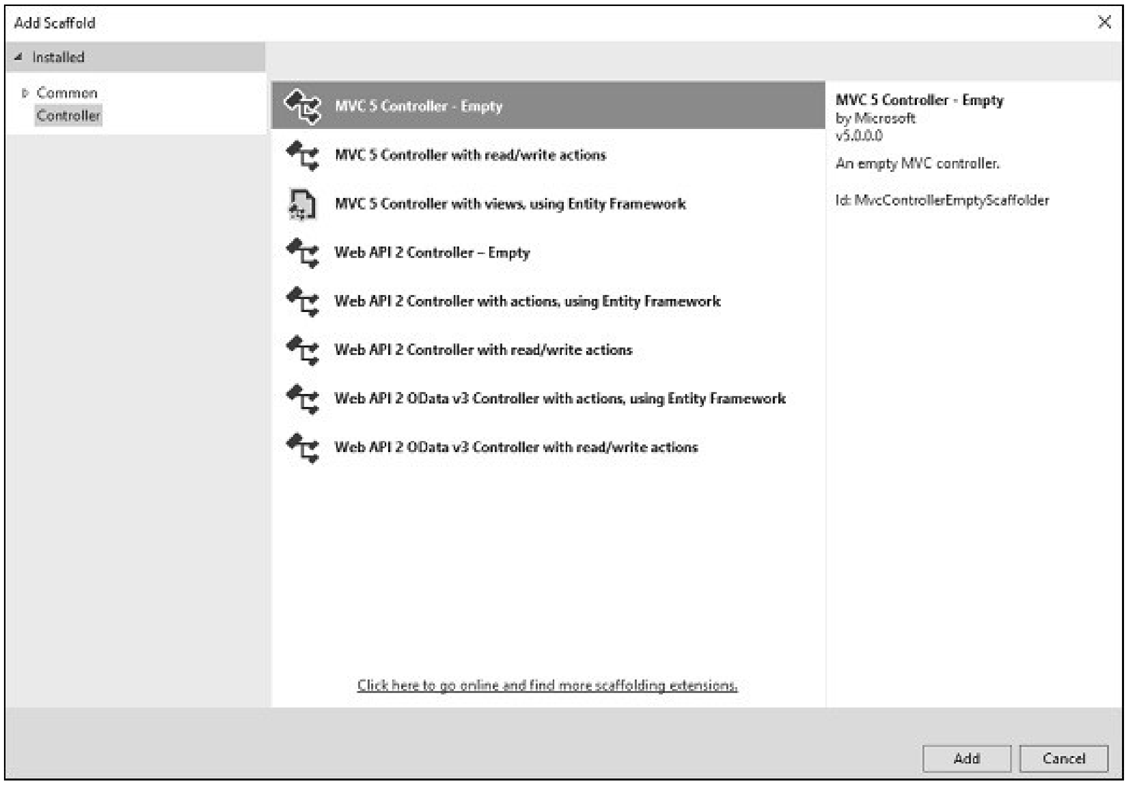
To keep things simple, select the Empty option and check the MVC checkbox in the ‘Add folders and core references for’ section and click Ok.

It will create a basic MVC project with minimal predefined content.

Step 6

To add a controller, right-click on the controller folder in the solution explorer and select Add → Controller.

It will display the Add Scaffold dialog.



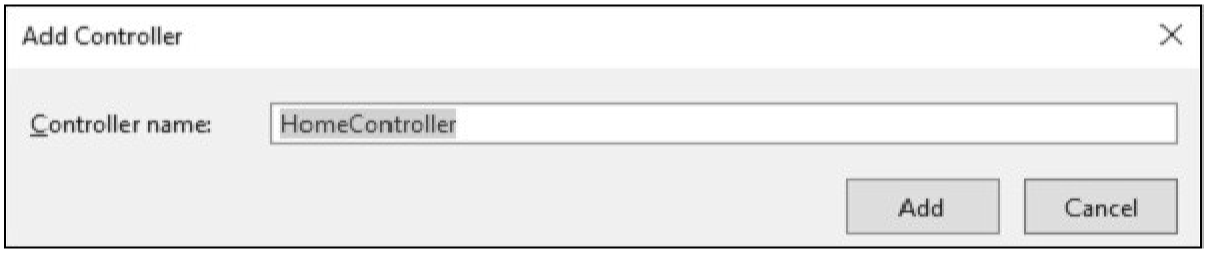
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions

MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder

Click here to go online and find more scaffolding extensions.

Step 7

Select the MVC 5 Controller – Empty option and click ‘Add’ button.
The Add Controller dialog will appear.



Controller name: HomeController

Step 8

Set the name to HomeController and click ‘Add’ button.

You will see a new C# file ‘HomeController.cs’ in the Controllers folder, which is open for editing in Visual Studio as well.

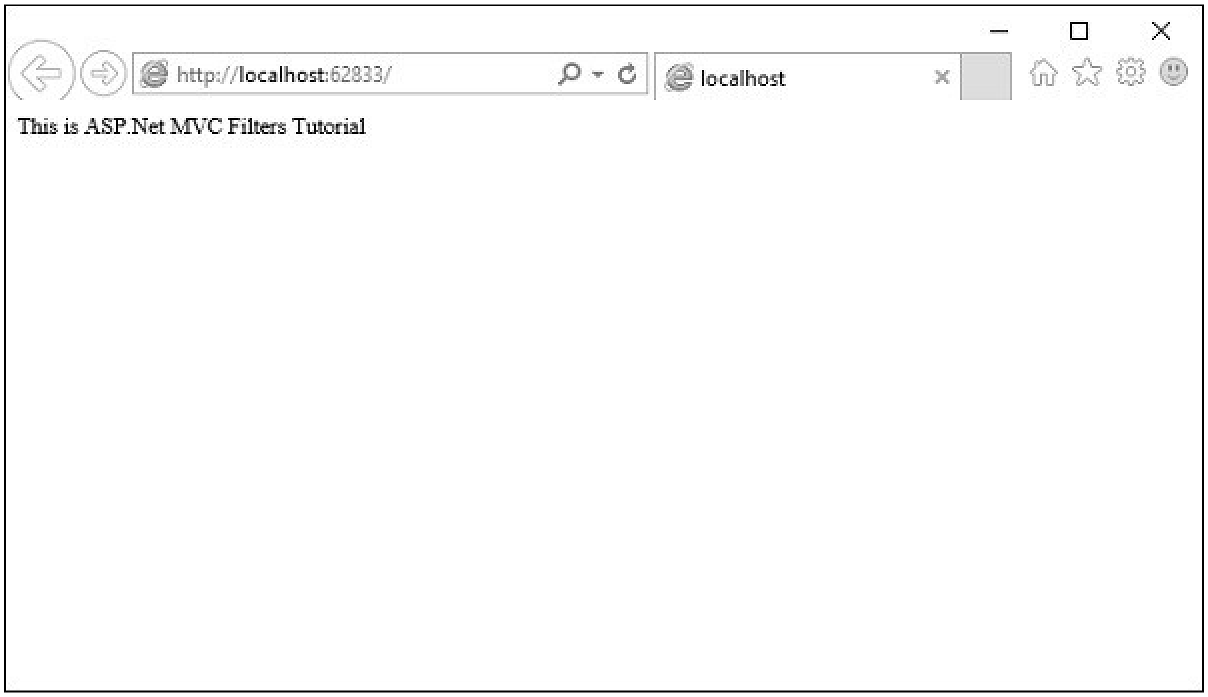
12.8.3. Apply Action Filter:

An action filter can be applied to either an individual controller action or an entire controller. For example, an action filter OutputCache is applied to an action named Index() that returns the string. This filter causes the value returned by the action to be cached for 15 seconds.

To make this a working example, let's modify the controller class by changing the action method called Index using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCFiltersDemo.Controllers {
  public class HomeController : Controller{
    // GET: Home
    [OutputCache(Duration = 15)]

    public string Index(){
      return "This is ASP.Net MVC Filters Tutorial";
    }
  }
}
```

When you run this application, you will see that the browser is displaying the result of the Index action method.

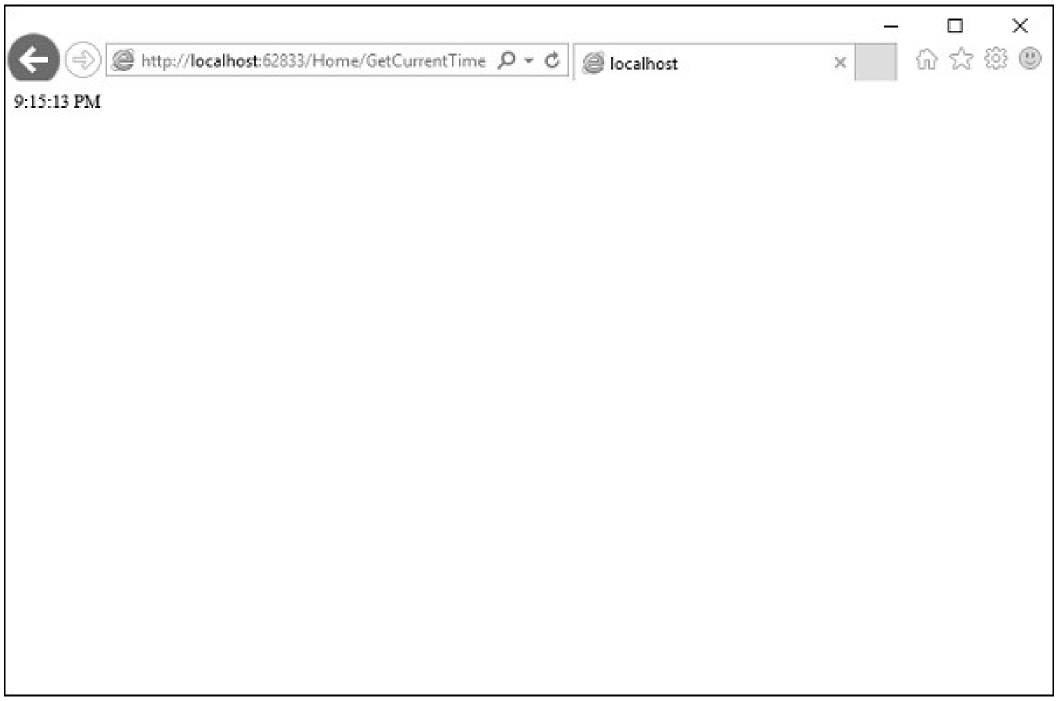


```
This is ASP.Net MVC Filters Tutorial
```

Let’s add another action method, which will display the current time.

```
namespace MVCFiltersDemo.Controllers{
  public class HomeController : Controller{
    // GET: Home

    [OutputCache(Duration = 15)]
    public string Index(){
      return "This is ASP.Net MVC Filters Tutorial";
    }

    [OutputCache(Duration = 20)]
    public string GetCurrentTime(){
      return DateTime.Now.ToString("T");
    }
  }
}
```

Request for the following URL, **http://localhost:62833/Home/GetCurrentTime**, and you will receive the following output.



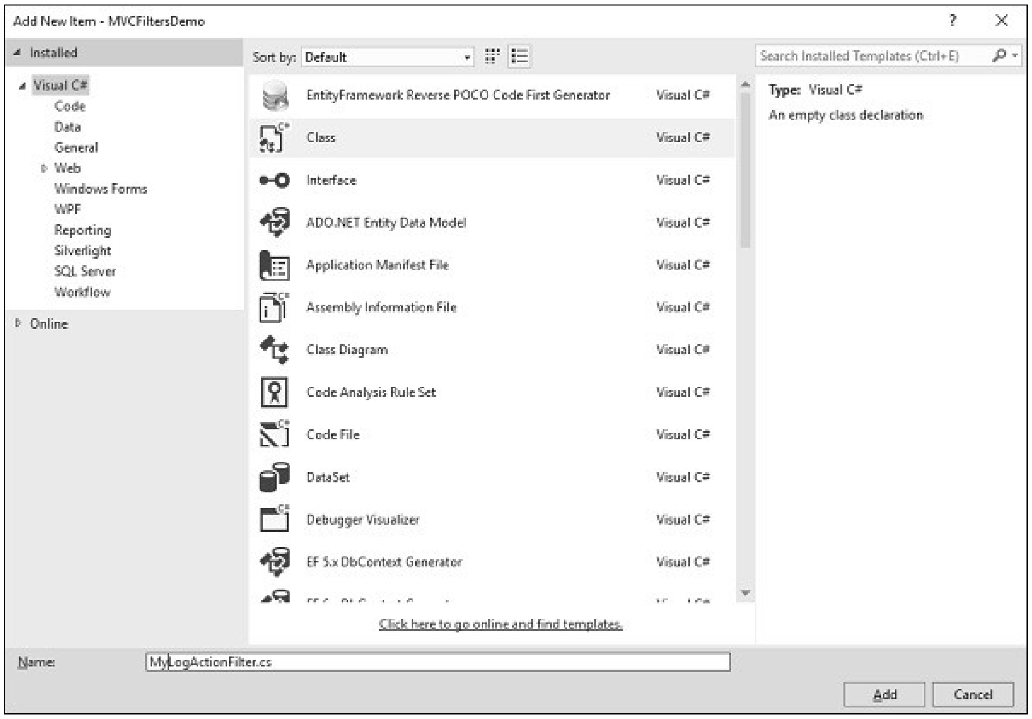
```
9:15:13 PM
```

If you refresh the browser, you will see the same time because the action is cached for 20 seconds. It will be updated when you refresh it after 20 seconds.

12.8.4. Custom Filters:

To create your own custom filter, ASP.NET MVC framework provides a base class which is known as ActionFilterAttribute. This class implements both IActionFilter and IResultFilter interfaces and both are derived from the Filter class.

Let's take a look at a simple example of custom filter by creating a new folder in your project with ActionFilters. Add one class for which right-click on ActionFilters folder and select Add → Class.


Add New Item - MVCFiltersDemo
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Workflow
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
EntityFramework Reverse POCO Code First Generator Visual C#
Class Visual C#
Interface Visual C#
ADO.NET Entity Data Model Visual C#
Application Manifest File Visual C#
Assembly Information File Visual C#
Class Diagram Visual C#
Code Analysis Rule Set Visual C#
Code File Visual C#
DataSet Visual C#
Debugger Visualizer Visual C#
EF 5.x DbContext Generator Visual C#
Type: Visual C#
An empty class declaration
Click here to go online and find templates.
Name: MyLogActionFilter.cs
Add
Cancel

Enter 'MyLogActionFilter' in the name field and click 'Add' button.

This class will be derived from the **ActionFilterAttribute**, which is a base class and overrides the following method. Following is the complete implementation of MyLogActionFilter.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;

using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
```

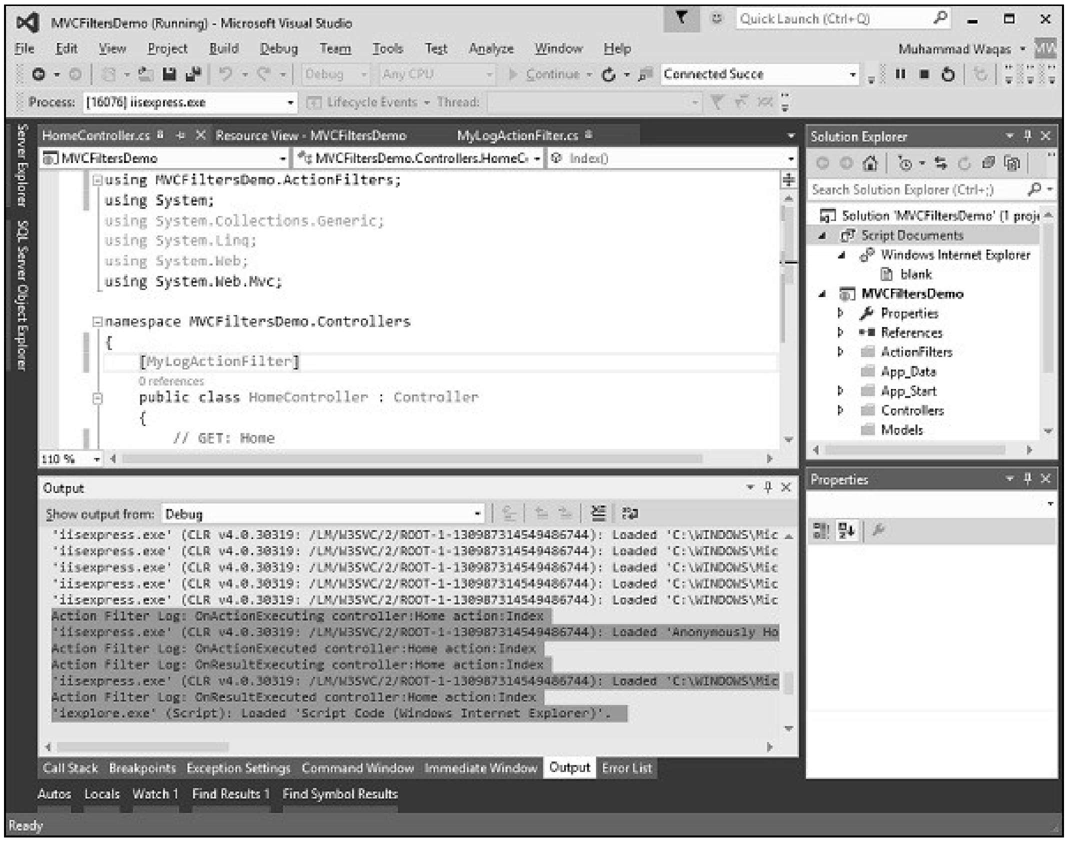
```
namespace MVCFiltersDemo.ActionFilters {
  public class MyLogActionFilter : ActionFilterAttribute{
    public override void OnActionExecuting(ActionExecutingContext filterContext){
      Log("OnActionExecuting", filterContext.RouteData);
    }

    public override void OnActionExecuted(ActionExecutedContext filterContext){
      Log("OnActionExecuted", filterContext.RouteData);
    }

    public override void OnResultExecuting(ResultExecutingContext filterContext){
      Log("OnResultExecuting", filterContext.RouteData);
    }

    public override void OnResultExecuted(ResultExecutedContext filterContext){
      Log("OnResultExecuted", filterContext.RouteData);
    }

    private void Log(string methodName, RouteData routeData){
      var controllerName = routeData.Values["controller"];
      var actionName = routeData.Values["action"];

      var message = String.Format(
        "{0} controller:{1} action:{2}", methodName, controllerName, actionName);

      Debug.WriteLine(message, "Action Filter Log");
    }
  }
}
```

Let us now apply the log filter to the HomeController using the following code.

```
using MVCFiltersDemo.ActionFilters;
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;
```

```
namespace MVCFiltersDemo.Controllers {
  [MyLogActionFilter]
  public class HomeController : Controller{
    // GET: Home

    [OutputCache(Duration = 10)]
    public string Index(){
      return "This is ASP.Net MVC Filters Tutorial";
    }

    [OutputCache(Duration = 10)]
    public string GetCurrentTime(){
      return DateTime.Now.ToString("T");
    }
  }
}
```

Run the application and then observe the output window.



```
using MVCFiltersDemo.ActionFilters;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCFiltersDemo.Controllers
{
    [MyLogActionFilter]
    0 references
    public class HomeController : Controller
    {
        // GET: Home
```

```
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'C:\WINDOWS\Mic
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'C:\WINDOWS\Mic
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'C:\WINDOWS\Mic
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'C:\WINDOWS\Mic
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'C:\WINDOWS\Mic
Action Filter Log: OnActionExecuting controller:Home action:Index
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'Anonymously Ho
Action Filter Log: OnActionExecuted controller:Home action:Index
Action Filter Log: OnResultExecuting controller:Home action:Index
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-130987314549486744): Loaded 'C:\WINDOWS\Mic
Action Filter Log: OnResultExecuted controller:Home action:Index
'iexplore.exe' (Script): Loaded 'Script Code (Windows Internet Explorer)'.
```

MVCFiltersDemo (Running) - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Muhammad Waqas
Debug Any CPU Continue Connected Succe
Process: [16076] iisexpress.exe Lifecycle Events Thread:
HomeController.cs Resource View - MVCFiltersDemo MyLogActionFilter.cs
MVCFiltersDemo MVCFiltersDemo.Controllers.HomeC Index()
Server Explorer SQL Server Object Explorer
110 %
Output
Show output from: Debug
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MVCFiltersDemo' (1 proj
Script Documents
Windows Internet Explorer
blank
MVCFiltersDemo
Properties
References
ActionFilters
App_Data
App_Start
Controllers
Models
Properties
Call Stack Breakpoints Exception Settings Command Window Immediate Window Output Error List
Autos Locals Watch 1 Find Results 1 Find Symbol Results
Ready

As seen in the above screenshot, the stages of processing the action are logged to the Visual Studio output window.

12.9. ASP.NET MVC – Action Selectors:

Action selectors are attributes that can be applied to action methods and are used to influence which action method gets invoked in response to a request. It helps the routing engine to select the correct action method to handle a particular request.

It plays a very crucial role when you are writing your action methods. These selectors will decide the behavior of the method invocation based on the modified name given in front of the action method. It is usually used to alias the name of the action method.

There are three types of action selector attributes –

- ActionName
- NonAction
- ActionVerbs

12.9.1. ActionName:

This class represents an attribute that is used for the name of an action. It also allows developers to use a different action name than the method name.

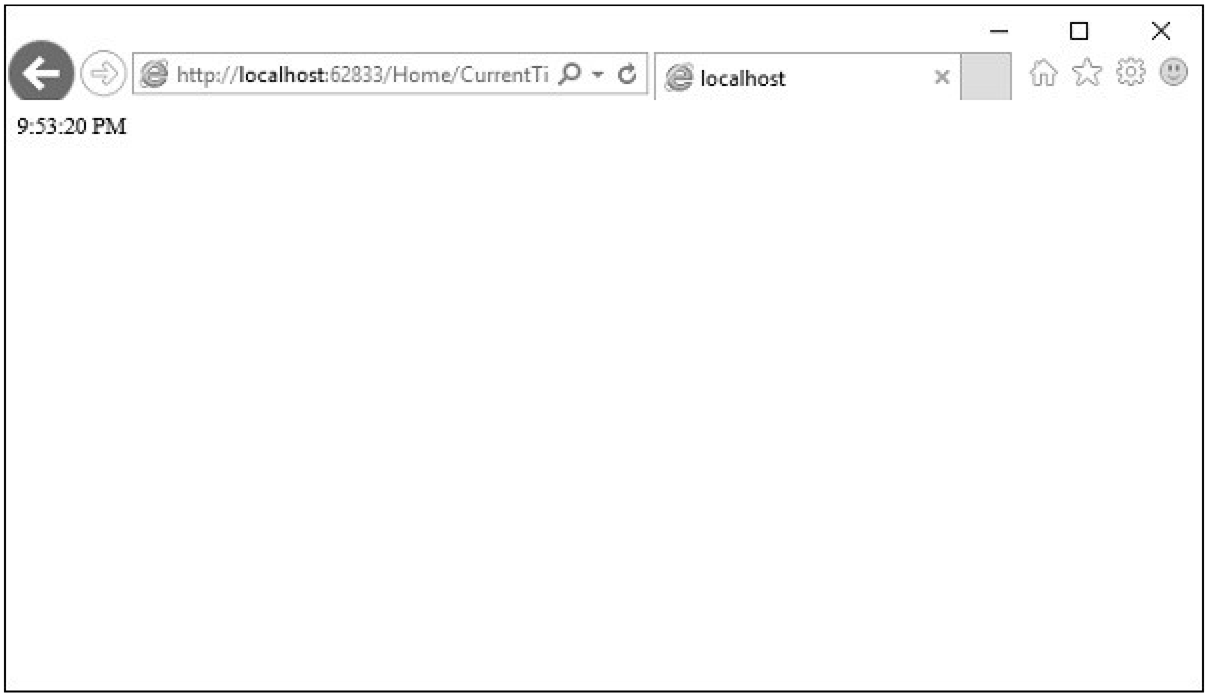
Let’s take a look at a simple example from the last chapter in which we have HomeController containing two action methods.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCFiltersDemo.Controllers {
  public class HomeController : Controller{
    // GET: Home
    public string Index(){
      return "This is ASP.Net MVC Filters Tutorial";
    }

    public string GetCurrentTime(){
      return DateTime.Now.ToString("T");
    }
  }
}
```

Let’s apply the the ActionName selector for GetCurrentTime by writing [ActionName("CurrentTime")] above the GetCurrentTime() as shown in the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCFiltersDemo.Controllers {
  public class HomeController : Controller{
    // GET: Home
    public string Index(){
      return "This is ASP.Net MVC Filters Tutorial";
    }

    [ActionName("CurrentTime")]
    public string GetCurrentTime(){
      return DateTime.Now.ToString("T");
    }
  }
}
```

Now run this application and enter the following URL in the browser **http://localhost:62833/Home/CurrentTime**, you will receive the following output.



9:53:20 PM

You can see that we have used the CurrentTime instead of the original action name, which is GetCurrentTime in the above URL.

12.9.2. NonAction:

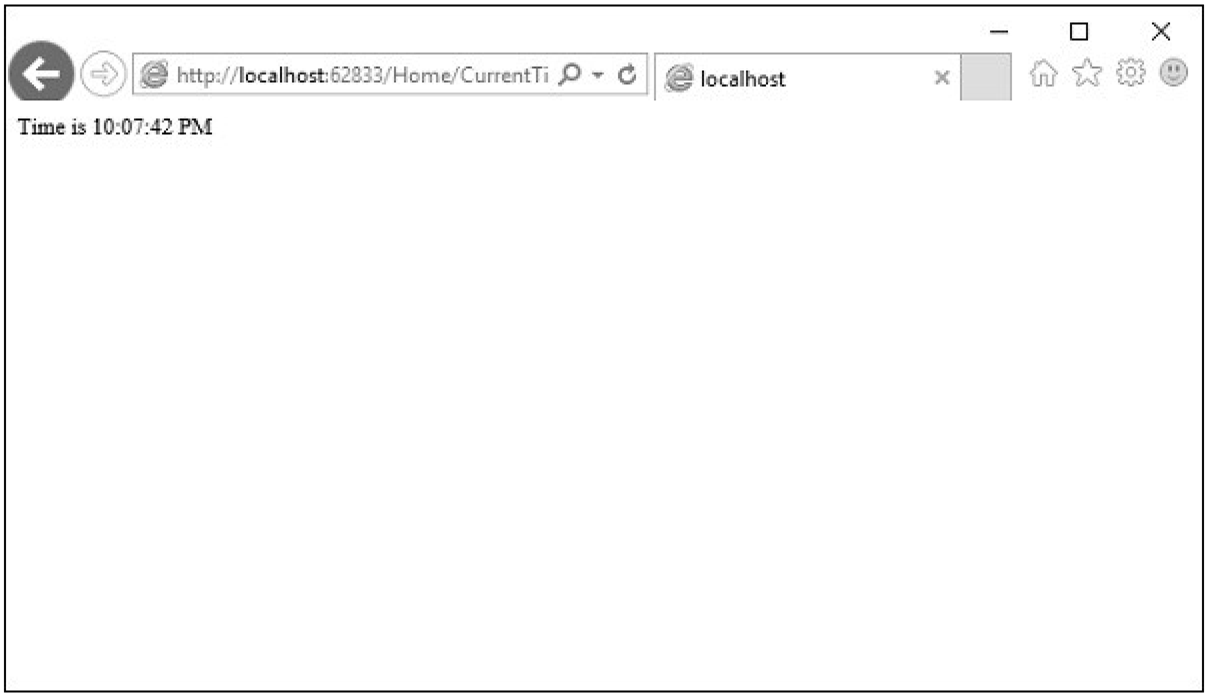
NonAction is another built-in attribute, which indicates that a public method of a Controller is not an action method. It is used when you want that a method shouldn't be treated as an action method.

Let's take a look at a simple example by adding another method in HomeController and also apply the NonAction attribute using the following code.

```
using MVCFiltersDemo.ActionFilters;
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCFiltersDemo.Controllers {
  public class HomeController : Controller{
    // GET: Home
    public string Index(){
       return "This is ASP.Net MVC Filters Tutorial";
    }

    [ActionName("CurrentTime")]
    public string GetCurrentTime(){
       return TimeString();
    }

    [NonAction]
    public string TimeString(){
       return "Time is " + DateTime.Now.ToString("T");
    }
  }
}
```

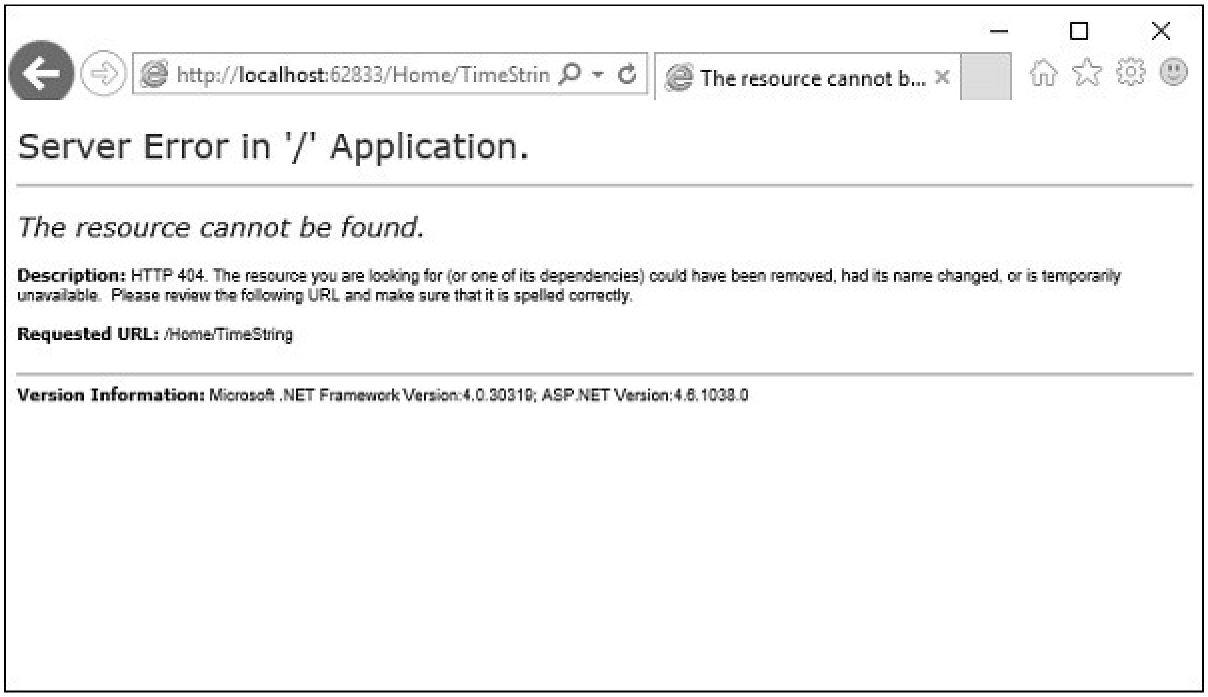
The new method TimeString is called from the GetCurrentTime() but you can’t use it as action in URL.

Let’s run this application and specify the following URL **http://localhost:62833/Home/CurrentTime** in the browser. You will receive the following output.



```
Time is 10:07:42 PM
```

Let us now check the /TimeString as action in the URL and see what happens.



Server Error in '/' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, had its name changed, or is temporarily unavailable. Please review the following URL and make sure that it is spelled correctly.

Requested URL: /Home/TimeString

Version Information: Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.6.1038.0

You can see that it gives '404—Not Found' error.

12.9.3. ActionVerbs:

Another selector filter that you can apply is the ActionVerbs attributes. So this restricts the indication of a specific action to specific HttpVerbs. You can define two different action methods with the same name but one action method responds to an HTTP Get request and another action method responds to an HTTP Post request.

MVC framework supports the following ActionVerbs.

- HttpGet
- HttpPost
- HttpPut
- HttpDelete
- HttpOptions
- HttpPatch

Let's take a look at a simple example in which we will create EmployeeController.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {
  public class EmployeeController : Controller{
    // GET: Employee
    public ActionResult Search(string name = “No name Entered”){
      var input = Server.HtmlEncode(name);
      return Content(input);
    }
  }
}
```

Now let’s add another action method with the same name using the following code.

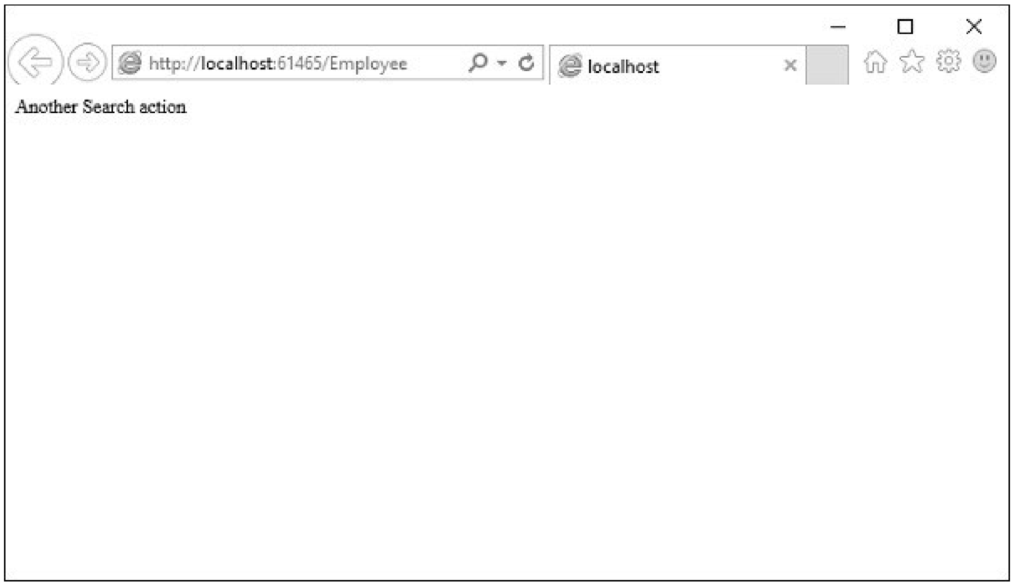
```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {
  public class EmployeeController : Controller{
    // GET: Employee
    //public ActionResult Index()
    //{
      // return View();
    //}

    public ActionResult Search(string name){
      var input = Server.HtmlEncode(name);
      return Content(input);
    }

    public ActionResult Search(){
      var input = "Another Search action";
      return Content(input);
    }
  }
}
```

When you run this application, it will give an error because the MVC framework is unable to figure out which action method should be picked up for the request.

Let us specify the HttpGet ActionVerb with the action you want as response using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {
  public class EmployeeController : Controller{
    // GET: Employee
    //public ActionResult Index()
    //{
      // return View();
    //}

    public ActionResult Search(string name){
      var input = Server.HtmlEncode(name);
      return Content(input);
    }

    [HttpGet]
    public ActionResult Search(){
      var input = "Another Search action";
      return Content(input);
    }
  }
}
```

When you run this application, you will receive the following output.



Another Search action

12.10. ASP.NET MVC – Views:

In an ASP.NET MVC application, there is nothing like a page and it also doesn't include anything that directly corresponds to a page when you specify a path in URL. The closest thing to a page in an ASP.NET MVC application is known as a **View**.

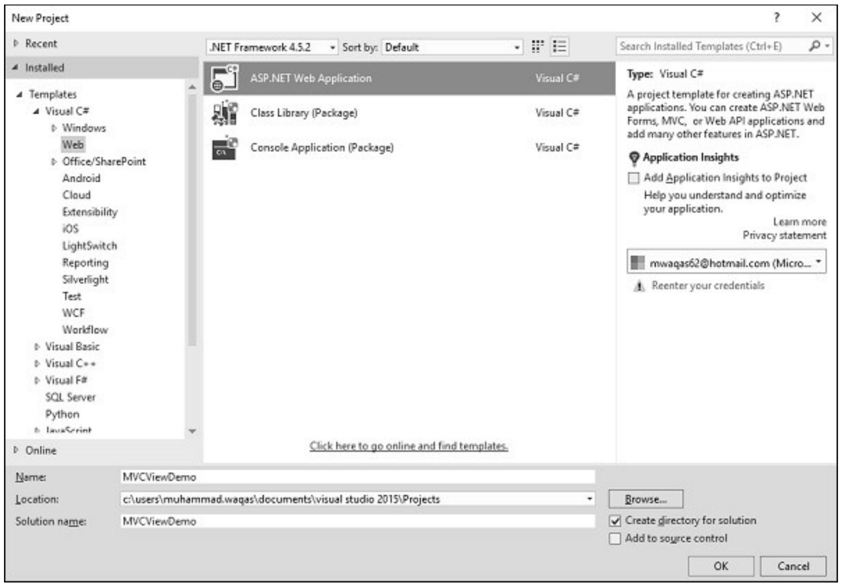
In ASP.NET MVC application, all incoming browser requests are handled by the controller and these requests are mapped to controller actions. A controller action might return a view or it might also perform some other type of action such as redirecting to another controller action.

Let's take a look at a simple example of View by creating a new ASP.NET MVC project.

Step 1

Open the Visual Studio and click File → New → Project menu option.

A new Project dialog opens.



.NET Framework 4.5.2 Sort by: Default

ASP.NET Web Application	Visual C#
Class Library (Package)	Visual C#
Console Application (Package)	Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to Project

Help you understand and optimize your application.

Learn more

Privacy statement

mwaqas62@hotmail.com (Micro...

Reenter your credentials

Name: MVCViewDemo

Location: c:\users\muhammad.waqas\documents\visual studio 2015\Projects

Solution name: MVCViewDemo

☑ Create directory for solution

☐ Add to source control

Step 2

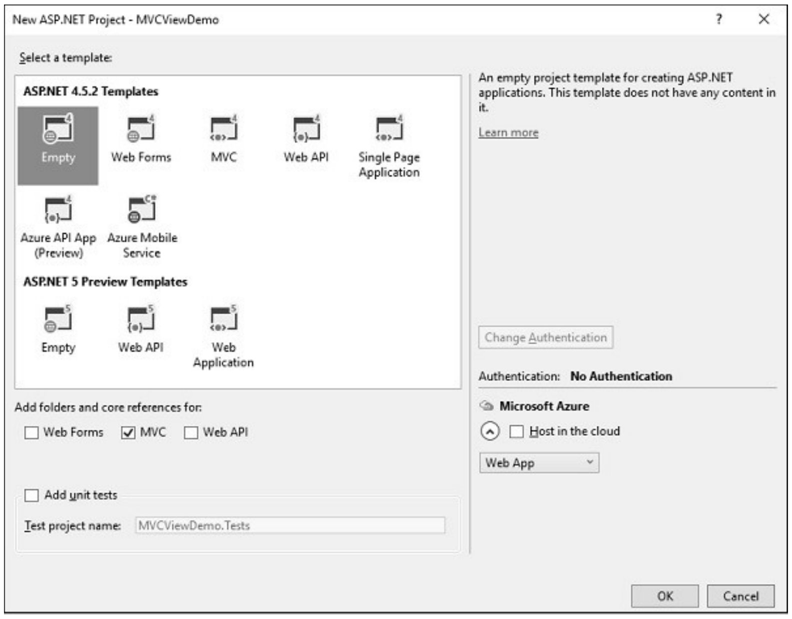
From the left pane, select Templates → Visual C# → Web.

Step 3

In the middle pane, select ASP.NET Web Application.

Step 4

Enter the project name 'MVCViewDemo' in the Name field and click Ok to continue. You will see the following dialog which asks you to set the initial content for the ASP.NET project.



An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Add folders and core references for:

- [ ] Web Forms
- [x] MVC
- [ ] Web API

- [ ] Host in the cloud

Web App

- [ ] Add unit tests

Test project name: MVCViewDemo.Tests

Step 5

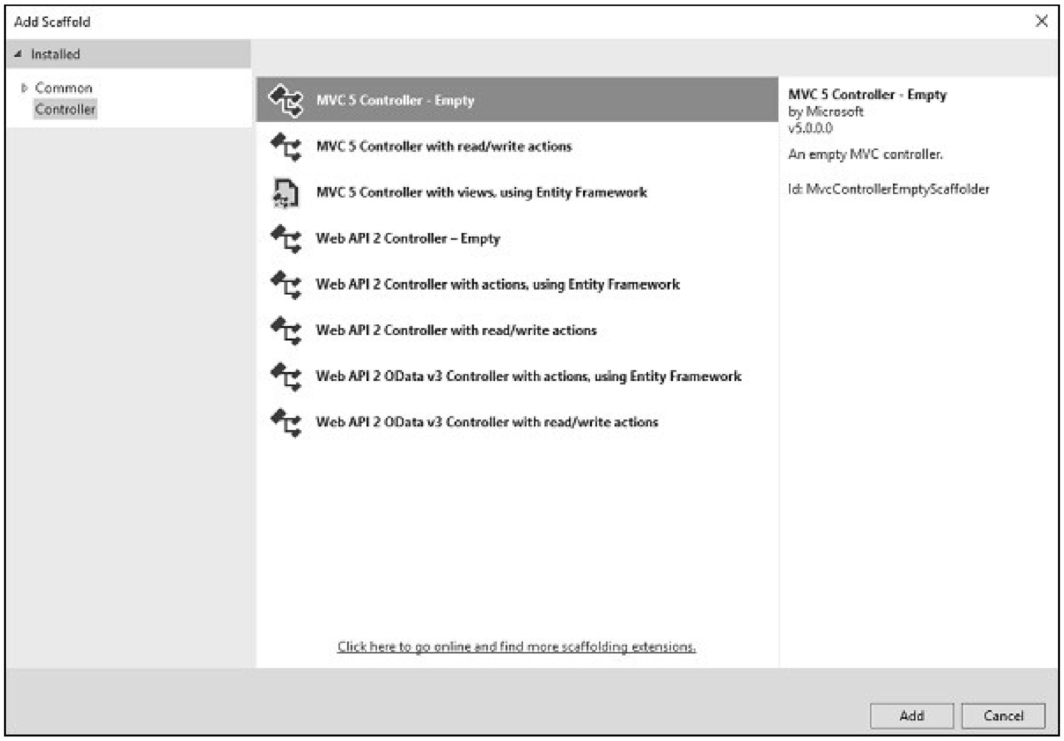
To keep things simple, select the Empty option and check the MVC checkbox in the ‘Add folders and core references for’ section and click Ok.

It will create a basic MVC project with minimal predefined content. We now need to add controller.

Step 6

Right-click on the controller folder in the solution explorer and select Add → Controller.

It will display the Add Scaffold dialog.



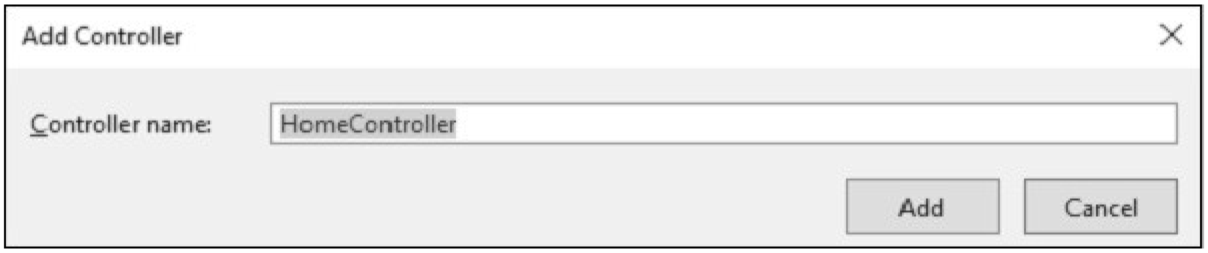
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions

Click here to go online and find more scaffolding extensions.

MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder

Step 7

Select the MVC 5 Controller – Empty option and click ‘Add’ button.
The Add Controller dialog will appear.



Controller name: HomeController

Step 8

Set the name to HomeController and click ‘Add’ button.

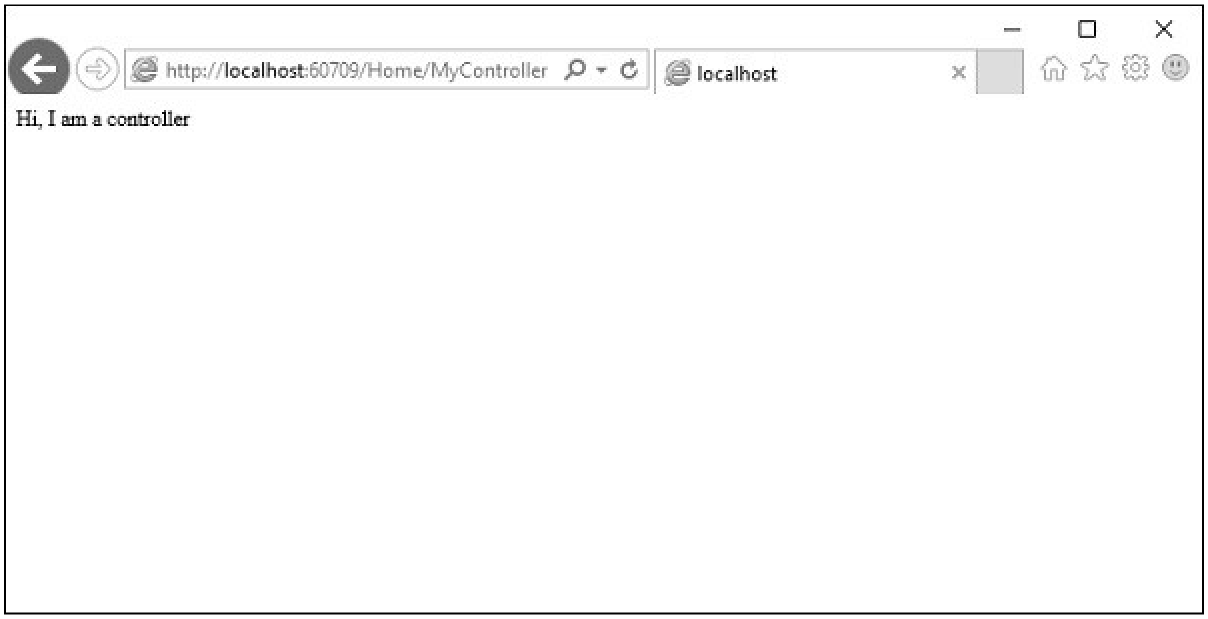
You will see a new C# file ‘HomeController.cs’ in the Controllers folder which is open for editing in Visual Studio as well.

Let’s update the HomeController.cs file, which contains two action methods as shown in the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCViewDemo.Controllers {
  public class HomeController : Controller{
    // GET: Home
    public ActionResult Index(){
      return View();
    }

    public string Mycontroller(){
      return "Hi, I am a controller";
    }
  }
}
```

Step 9

Run this application and apend /Home/MyController to the URL in the browser and press enter. You will receive the following output.


http://localhost:60709/Home/MyController

localhost

```
Hi, I am a controller
```

As MyController action simply returns the string, to return a View from the action we need to add a View first.

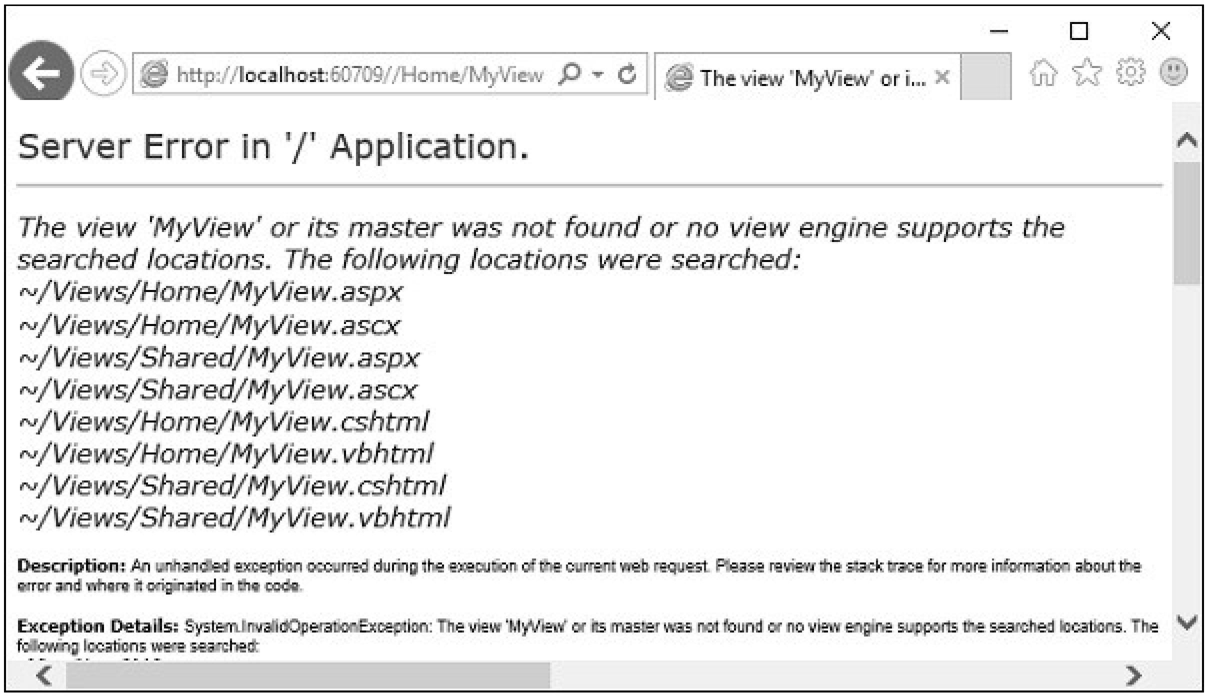
Step 10

Before adding a view let’s add another action, which will return a default view.

```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCViewDemo.Controllers {
  public class HomeController : Controller{
    // GET: Home
    public ActionResult Index(){
      return View();
    }

    public string Mycontroller(){
      return "Hi, I am a controller";
    }

    public ActionResult MyView(){
      return View();
    }
  }
}
```

Step 11

Run this application and apend /Home/MyView to the URL in the browser and press enter. You will receive the following output.


http://localhost:60709//Home/MyView
The view 'MyView' or i...

Server Error in '/' Application.

The view 'MyView' or its master was not found or no view engine supports the searched locations. The following locations were searched:
~/Views/Home/MyView.aspx
~/Views/Home/MyView.ascx
~/Views/Shared/MyView.aspx
~/Views/Shared/MyView.ascx
~/Views/Home/MyView.cshtml
~/Views/Home/MyView.vbhtml
~/Views/Shared/MyView.cshtml
~/Views/Shared/MyView.vbhtml

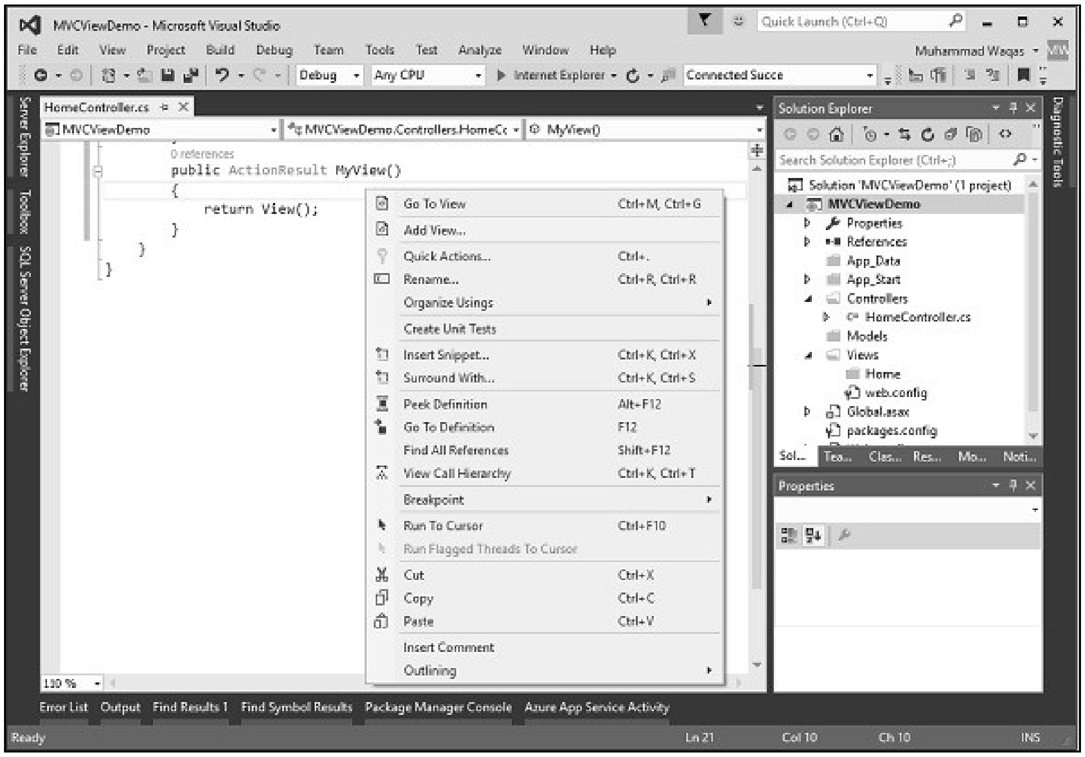
Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.InvalidOperationException: The view 'MyView' or its master was not found or no view engine supports the searched locations. The following locations were searched:

You can see here that we have an error and this error is actually quite descriptive, which tells us it can't find the MyView view.

Step 12

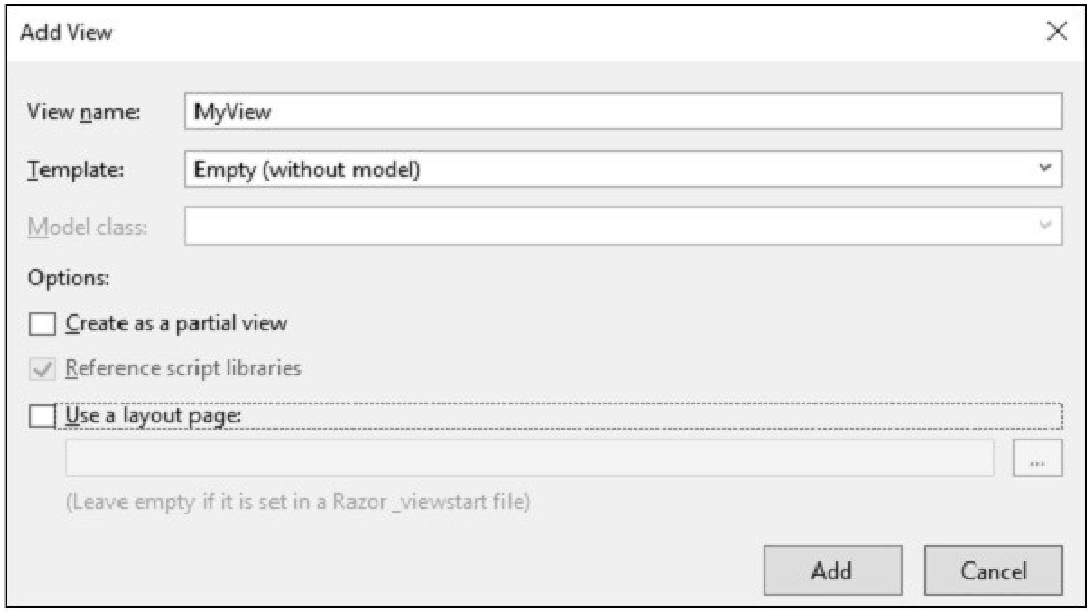
To add a view, right-click inside the MyView action and select Add view.



```
        0 references
        public ActionResult MyView()
        {
            return View();
        }
    }
}
```

MVCViewDemo - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
HomeController.cs
Go To View Ctrl+M, Ctrl+G
Add View...
Quick Actions... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Organize Usings
Create Unit Tests
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Peek Definition Alt+F12
Go To Definition F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Insert Comment
Outlining
Solution Explorer
Solution 'MVCViewDemo' (1 project)
MVCViewDemo
Properties
References
App_Data
App_Start
Controllers
HomeController.cs
Models
Views
Home
web.config
Global.asax
packages.config
Properties
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Ready Ln 21 Col 10 Ch 10 INS

It will display the Add View dialog and it is going to add the default name.



Add View

View name: MyView

Template: Empty (without model)

Model class:

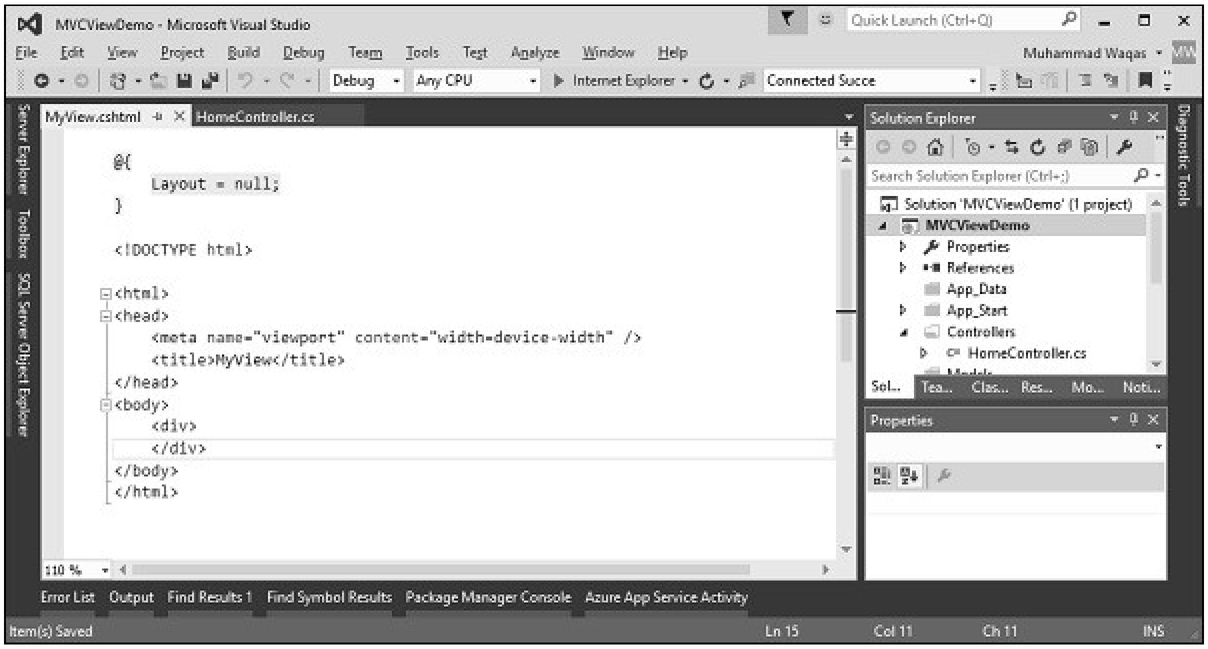
Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [ ] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Step 13

Uncheck the 'Use a layout page' checkbox and click 'Add' button.
We now have the default code inside view.


MVCViewDemo - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Muhammad Waqas
Debug
Any CPU
Internet Explorer
Connected Succe
MyView.cshtml
HomeController.cs
Server Explorer
Toolbox
SQL Server Object Explorer
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MVCViewDemo' (1 project)
MVCViewDemo
Properties
References
App_Data
App_Start
Controllers
HomeController.cs
Sol... Tea... Clas... Res... Mo... Noti...
Properties
Diagnostic Tools
110 %
Error List Output Find Results 1 Find Symbol Results Package Manager Console Azure App Service Activity
Item(s) Saved
Ln 15 Col 11 Ch 11 INS

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>MyView</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

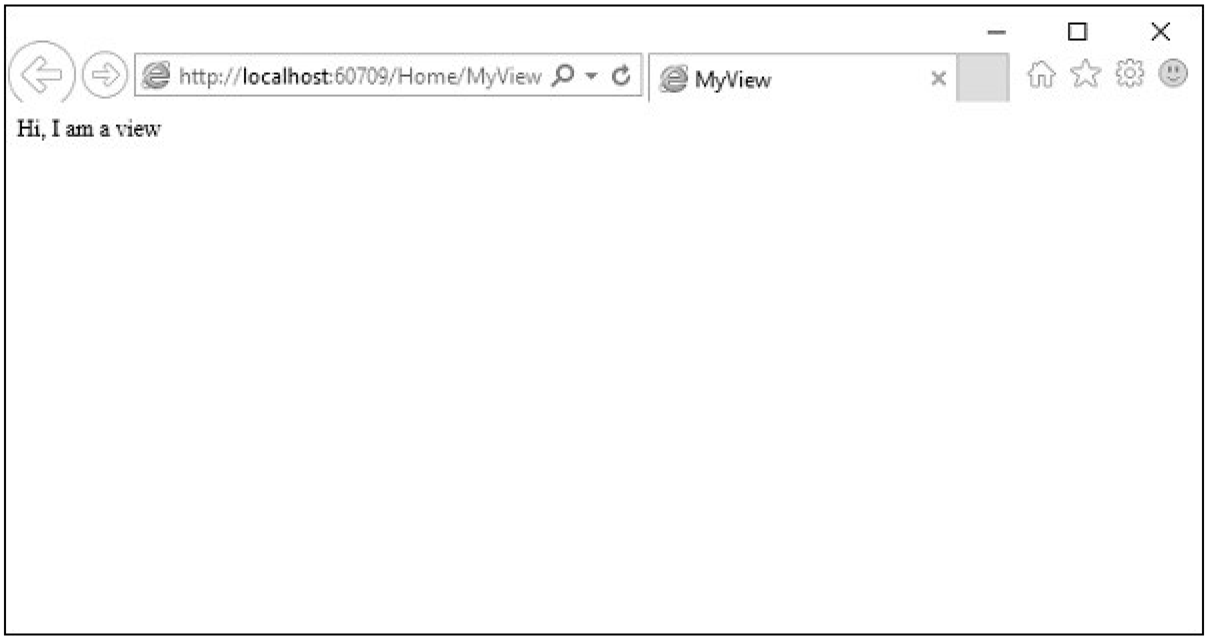
Step 14

Add some text in this view using the following code.

```
@{
  Layout = null;
}

<!DOCTYPE html>
<html>
<head>
<meta name = "viewport" content = "width = device-width" />
<title>MyView</title>
</head>

<body>
<div>
      Hi, I am a view
</div>
</body>

</html>
```

Step 15

Run this application and apend /Home/MyView to the URL in the browser. Press enter and you will receive the following output.



Hi, I am a view

You can now see the text from the View.

12.11. ASP.NET MVC – Model:

In this lecture, we will discuss about building models in an ASP.NET MVC Framework application. A **model** stores data that is retrieved according to the commands from the Controller and displayed in the View.

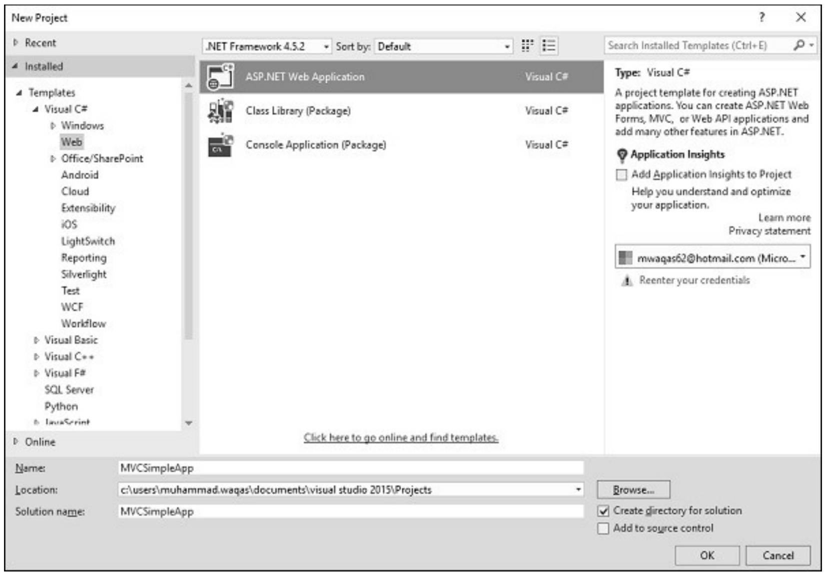
Model is a collection of classes wherein you will be working with data and business logic. Hence, basically models are business domain-specific containers. It is used to interact with database. It can also be used to manipulate the data to implement the business logic.

Let’s take a look at a simple example of View by creating a new ASP.Net MVC project.

Step 1

Open the Visual Studio. Click File → New → Project menu option.

A new Project dialog opens.



.NET Framework 4.5.2 Sort by: Default

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to Project
Help you understand and optimize your application.

Name: MVCSimpleApp
Location: c:\users\muhammad.waqas\documents\visual studio 2015\Projects
Solution name: MVCSimpleApp

☑ Create directory for solution
☐ Add to source control

Step 2

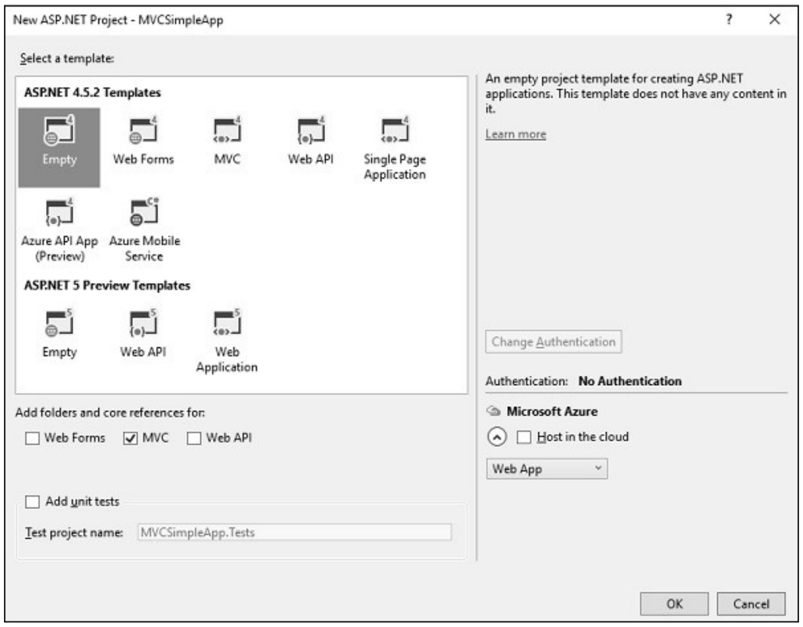
From the left pane, select Templates → Visual C# → Web.

Step 3

In the middle pane, select ASP.NET Web Application.

Step 4

Enter the project name ‘MVCSimpleApp’ in the Name field and click Ok to continue. You will see the following dialog which asks you to set the initial content for the ASP.NET project.



An empty project template for creating ASP.NET applications. This template does not have any content in it.

Add folders and core references for:
- [ ] Web Forms
- [x] MVC
- [ ] Web API

- [ ] Add unit tests

Test project name: MVCSimpleApp.Tests

Authentication: **No Authentication**

- [ ] Host in the cloud

Step 5

To keep things simple, select the Empty option and check the MVC checkbox in the 'Add folders and core references for' section and click Ok.

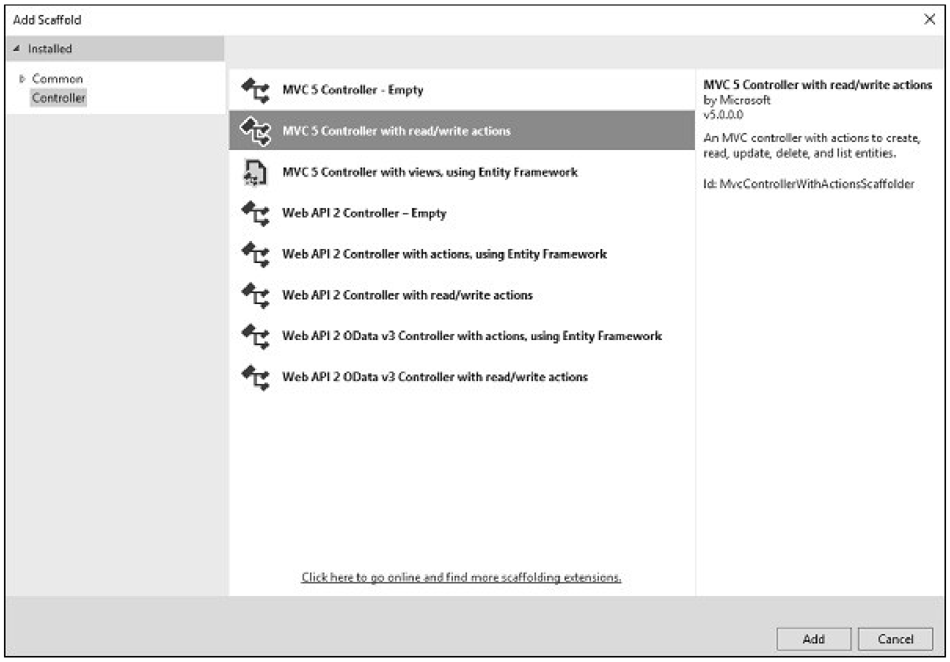
It will create a basic MVC project with minimal predefined content.

We need to add a controller now.

Step 6

Right-click on the controller folder in the solution explorer and select Add → Controller.

It will display the Add Scaffold dialog.


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
Click here to go online and find more scaffolding extensions.
Add
Cancel

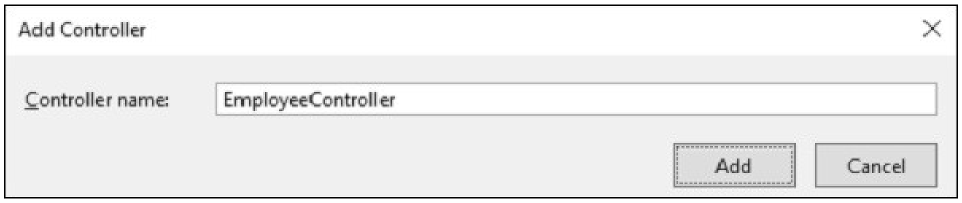
MVC 5 Controller with read/write actions
by Microsoft
v5.0.0.0
An MVC controller with actions to create, read, update, delete, and list entities.
Id: MvcControllerWithActionsScaffolder

Step 7

Select the MVC 5 Controller – with read/write actions option. This template will create an Index method with default action for Controller. This will also list other methods like Edit/Delete/Create as well.

Step 8

Click ‘Add’ button and Add Controller dialog will appear.



Controller name: EmployeeController

Step 9

Set the name to EmployeeController and click the ‘Add’ button.

Step 10

You will see a new C# file ‘EmployeeController.cs’ in the Controllers folder, which is open for editing in Visual Studio with some default actions.

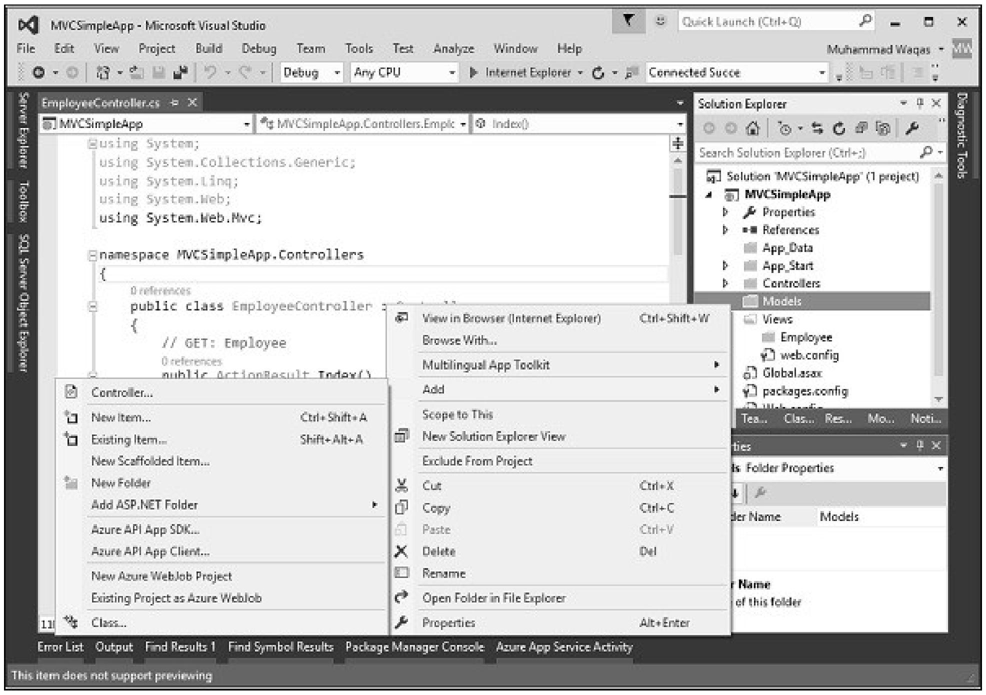
```
using System;
using System.Collections.Generic;
using System.Linq;

using System.Web;
using System.Web.Mvc;

namespace MVCSimpleApp.Controllers {
  public class EmployeeController : Controller{
    // GET: Employee
    public ActionResult Index(){
      return View();
    }

    // GET: Employee/Details/5
    public ActionResult Details(int id){
      return View();
    }

    // GET: Employee/Create
    public ActionResult Create(){
      return View();
    }

    // POST: Employee/Create
    [HttpPost]
    public ActionResult Create(FormCollection collection){
      try{
        // TODO: Add insert logic here
        return RedirectToAction("Index");
      }catch{
        return View();
      }
    }

    // GET: Employee/Edit/5
    public ActionResult Edit(int id){
      return View();
    }
```

```
      // POST: Employee/Edit/5
      [HttpPost]
      public ActionResult Edit(int id, FormCollection collection){
        try{
          // TODO: Add update logic here
          return RedirectToAction("Index");
        }catch{
          return View();
        }
      }

      // GET: Employee/Delete/5
      public ActionResult Delete(int id){
        return View();
      }

      // POST: Employee/Delete/5
      [HttpPost]
      public ActionResult Delete(int id, FormCollection collection){
        try{
          // TODO: Add delete logic here
          return RedirectToAction("Index");
        }catch{
          return View();
        }
      }
    }
  }
```

Let’s add a model.

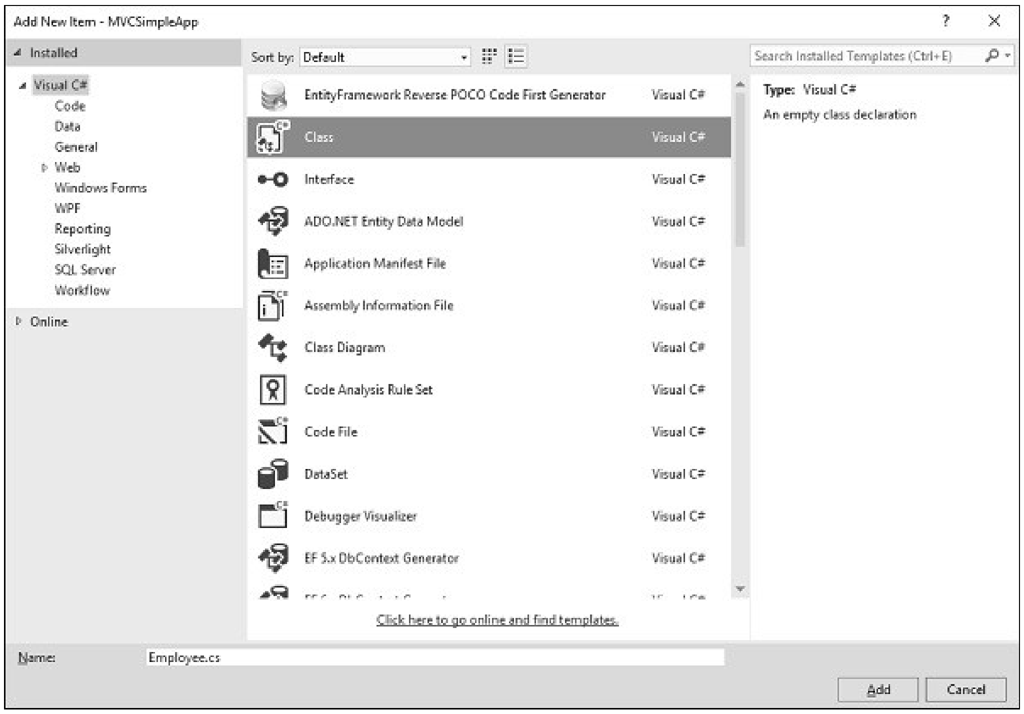
Step 11

Right-click on the Models folder in the solution explorer and select Add → Class.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCSimpleApp.Controllers
{
    public class EmployeeController : Controller
    {
        // GET: Employee
        public ActionResult Index()
```

You will see the Add New Item dialog.


Add New Item - MVCSimpleApp
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Workflow
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
EntityFramework Reverse POCO Code First Generator Visual C#
Class Visual C#
Interface Visual C#
ADO.NET Entity Data Model Visual C#
Application Manifest File Visual C#
Assembly Information File Visual C#
Class Diagram Visual C#
Code Analysis Rule Set Visual C#
Code File Visual C#
DataSet Visual C#
Debugger Visualizer Visual C#
EF 5.x DbContext Generator Visual C#
Type: Visual C#
An empty class declaration
Click here to go online and find templates.
Add
Cancel

Name: Employee.cs

Step 12

Select Class in the middle pan and enter Employee.cs in the name field.

Step 13

Add some properties to Employee class using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MVCSimpleApp.Models {
  public class Employee{
    public int ID { get; set; }
    public string Name { get; set; }
    public DateTime JoiningDate { get; set; }
    public int Age { get; set; }
  }
}
```

Let’s update the EmployeeController.cs file by adding one more method, which will return the list of employee.

```
[NonAction]
public List<Employee> GetEmployeeList(){
  return new List<Employee>{
    new Employee{
      ID = 1,
      Name = "Allan",
      JoiningDate = DateTime.Parse(DateTime.Today.ToString()),
      Age = 23
    },

    new Employee{
      ID = 2,
      Name = "Carson",
      JoiningDate = DateTime.Parse(DateTime.Today.ToString()),
      Age = 45
    },

    new Employee{
      ID = 3,
      Name = "Carson",
      JoiningDate = DateTime.Parse(DateTime.Today.ToString()),
      Age = 37
    },

    new Employee{
      ID = 4,
      Name = "Laura",
      JoiningDate = DateTime.Parse(DateTime.Today.ToString()),
      Age = 26
    },
  };
}
```

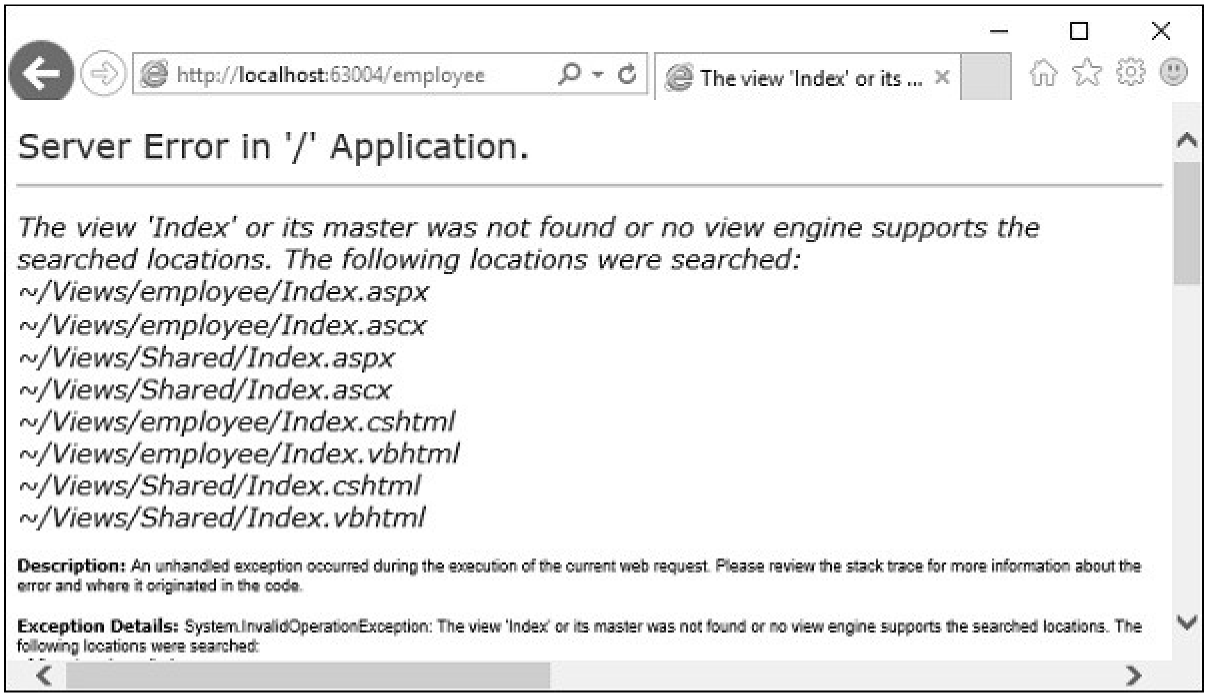
Step 14

Update the index action method as shown in the following code.

```
public ActionResult Index(){
  var employees = from e in GetEmployeeList()
  orderby e.ID
  select e;
  return View(employees);
}
```

Step 15

Run this application and append /employee to the URL in the browser and press Enter. You will see the following output.



Server Error in '/' Application.

The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:
~/Views/employee/Index.aspx
~/Views/employee/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
~/Views/employee/Index.cshtml
~/Views/employee/Index.vbhtml
~/Views/Shared/Index.cshtml
~/Views/Shared/Index.vbhtml

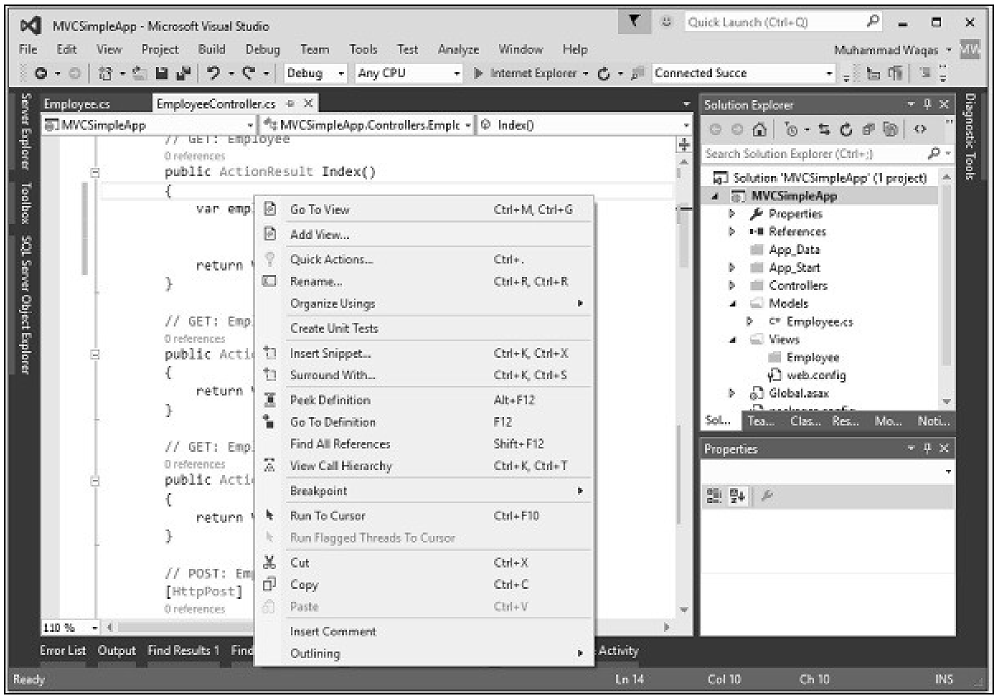
Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.InvalidOperationException: The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:

As seen in the above screenshot, there is an error and this error is actually quite descriptive which tells us it can't find the Index view.

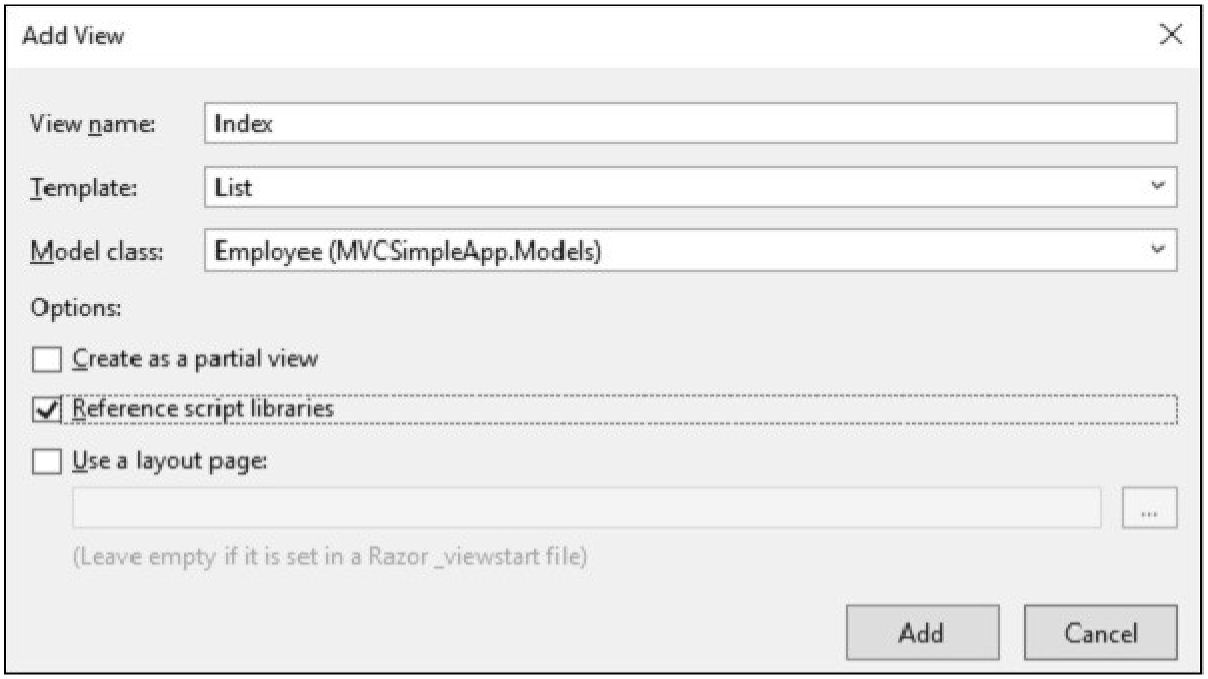
Step 16

Hence to add a view, right-click inside the Index action and select Add view.



```
// GET: Employee
public ActionResult Index()
{
    var emp
    return
}
```

It will display the Add View dialog and it is going to add the default name.


Add View

View name: Index

Template: List

Model class: Employee (MVCSimpleApp.Models)

Options:

- [ ] Create as a partial view
- [✓] Reference script libraries
- [ ] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Step 17

Select the List from the Template dropdown and Employee in Model class dropdown and also uncheck the ‘Use a layout page’ checkbox and click ‘Add’ button.

It will add some default code for you in this view.

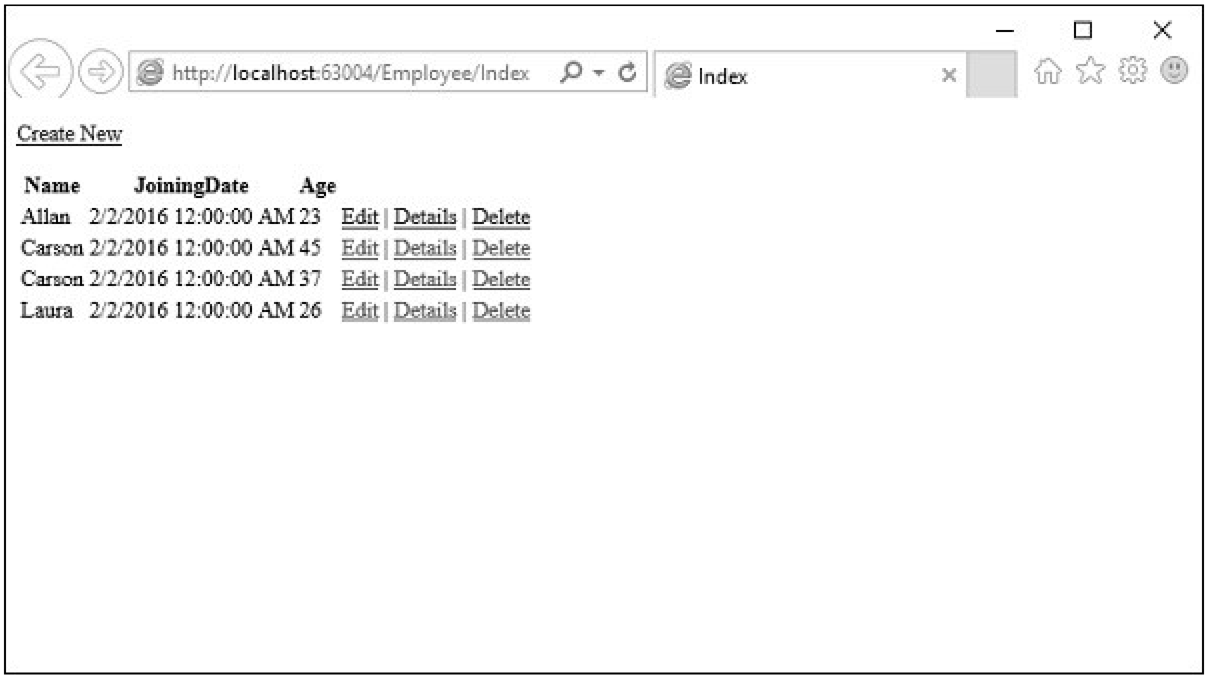
```
@model IEnumerable<MVCSimpleApp.Models.Employee>
@{
  Layout = null;
}

<!DOCTYPE html>
<html>
<head>
<meta name = "viewport" content = "width = device-width" />
<title>Index</title>
</head>

<body>
<p>@Html.ActionLink("Create New", "Create")</p>
<table class = "table">
```

```
<tr>
<th>
          @Html.DisplayNameFor(model => model.Name)
</th>

<th>
          @Html.DisplayNameFor(model => model.JoiningDate)
</th>

<th>
          @Html.DisplayNameFor(model => model.Age)
</th>

<th></th>
</tr>

      @foreach (var item in Model) {
<tr>
<td>
            @Html.DisplayFor(modelItem => item.Name)
</td>

<td>
            @Html.DisplayFor(modelItem => item.JoiningDate)
</td>

<td>
            @Html.DisplayFor(modelItem => item.Age)
</td>

<td>
            @Html.ActionLink("Edit", "Edit", new { id = item.ID }) |
            @Html.ActionLink("Details", "Details", new { id = item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id = item.ID })
</td>

</tr>
      }
```

```
</table>
</body>
</html>
```

Step 18

Run this application and you will receive the following output.


http://localhost:63004/Employee/Index

Index

Create New

```
 Name      JoiningDate        Age
Allan  2/2/2016 12:00:00 AM 23  Edit | Details | Delete
Carson 2/2/2016 12:00:00 AM 45  Edit | Details | Delete
Carson 2/2/2016 12:00:00 AM 37  Edit | Details | Delete
Laura  2/2/2016 12:00:00 AM 26  Edit | Details | Delete
```

A list of employees will be displayed.

Lecture 13: .net framework, C#.net introduction, Program Structure, Basic Syntax, Data Types, Variables

13.1. Introduction to .NET Framework:

.NET is a software framework which is designed and developed by Microsoft. The first version of .Net framework was 1.0 which came in the year 2002. In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#, VB.Net etc.It is used to develop Form-based applications, Web-based applications, and Web services. There is a variety of programming languages available on the .Net platform, VB.Net and C# being the most common ones. It is used to build applications for Windows, phone, web etc. It provides a lot of functionalities and also supports industry standards.

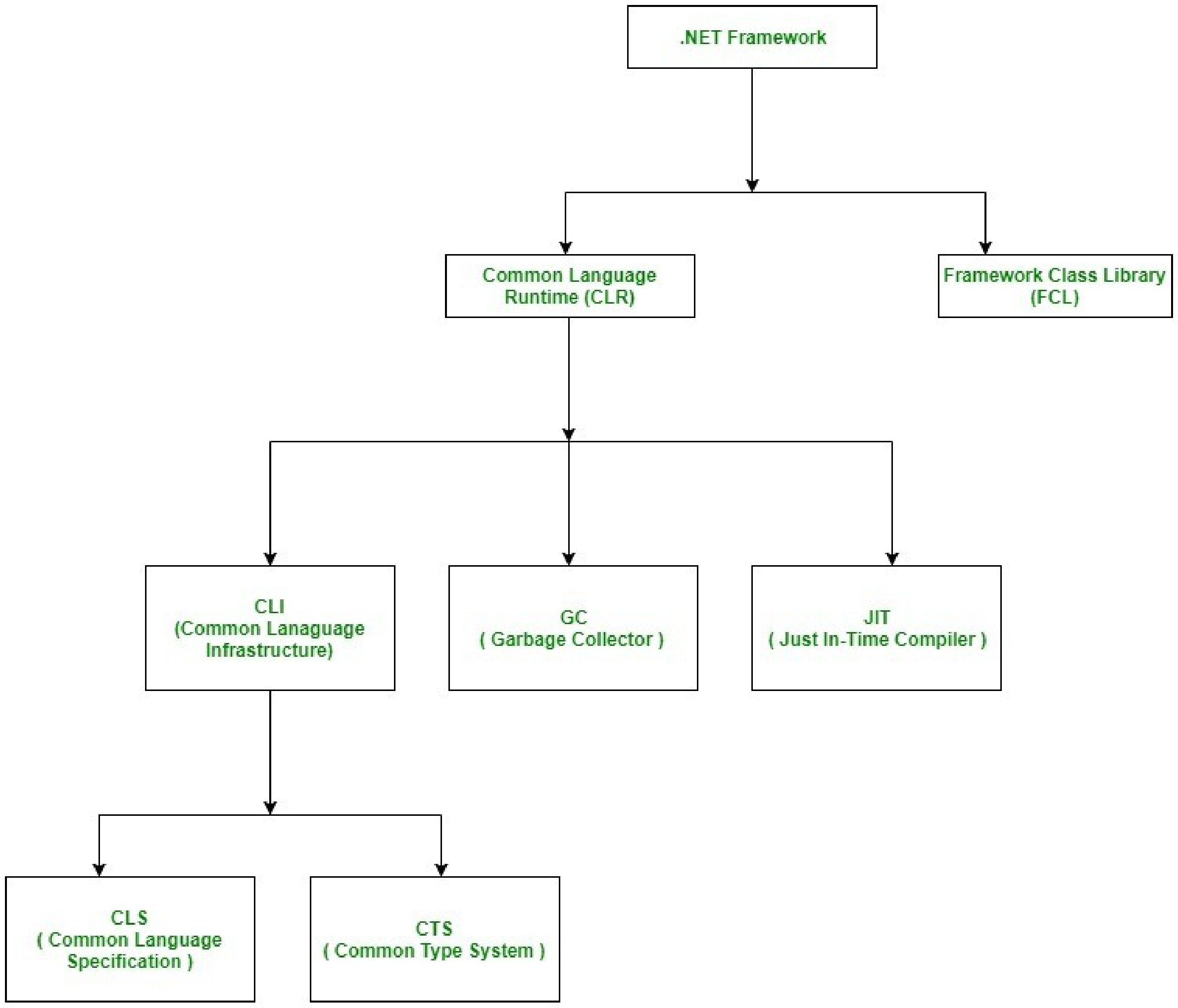
.NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft. The remaining **Non-Microsoft Languages** which are supported by .NET Framework but not designed and developed by Microsoft.
11 Programming Languages which are designed and developed by Microsoft are –

- C#.NET
- VB.NET
- C++.NET
- J#.NET
- F#.NET
- JSCRIPT.NET
- WINDOWS POWERSHELL
- IRON RUBY
- IRON PYTHON
- C OMEGA
- ASML(Abstract State Machine Language)

13.2. Main Components of .NET Framework:

Common Language Runtime(CLR): CLR is the basic and Virtual Machine component of the .NET Framework. It is the run-time environment in the .NET Framework that runs the codes and helps in making the development process easier by providing the various services such as remoting, thread management, type-safety, memory management, robustness etc. Basically, it is responsible for managing the execution of .NET programs regardless of any .NET programming language. It also helps in the management of code, as code that targets the runtime is known as the Managed Code and code doesn't target to runtime is known as Unmanaged code.

Framework Class Library(FCL): It is the collection of reusable, object-oriented class libraries and methods etc that can be integrated with CLR. Also called the Assemblies. It is just like the header files in C/C++ and packages in the java. Installing .NET framework basically is the installation of CLR and FCL into the system. Below is the overview of .NET Framework.


.NET Framework
Common Language Runtime (CLR)
Framework Class Library (FCL)
CLI (Common Lanaguage Infrastructure)
GC (Garbage Collector)
JIT (Just In-Time Compiler)
CLS (Common Language Specification)
CTS (Common Type System)

13.3. Is .NET application platform dependent or platform independent?

The combination of *Operating System Architecture and CPU Architecture* is known as the platform. Platform dependent means the programming language code will run only on particular Operating System. A *.NET application is platform dependent* because of the .NET framework which is only able to run on the Windows-based operating system. The .Net application is platform independent also because of *Mono framework*. Using Mono framework the .Net application can run on any Operating System including windows. Mono framework is a third party software developed by **Novell Company** which is now a part of **Micro Focus Company**. It is a paid framework.

13.4. Release History of .NET Framework and its compatibility with the different Windows version:

.NET VERSION	CLR VERSION	DEVELOPMENT TOOL	WINDOWS SUPPORT
1.0	1.0	Visual Studio .NET	XP SP1
1.1	1.1	Visual Studio .NET 2003	XP SP2, SP3
2.0	2.0	Visual Studio 2005	N/A
3.0	2.0	Expression Blend	Vista
3.5	2.0	Visual Studio 2008	7, 8, 8.1, 10
4.0	4	Visual Studio 2010	N/A
4.5	4	Visual Studio 2012	8
4.5.1	4	Visual Studio 2013	8.1
4.5.2	4	N/A	N/A
4.6	4	Visual Studio 2015	10 v1507
4.6.1	4	Visual Studio 2015 Update 1	10 v1511
4.6.2	4	N/A	10 v1607
4.7	4	Visual Studio 2017	10 v1703
4.7.1	4	Visual Studio 2017	10 v1709
4.7.2	4	Visual Studio 2017	10v 1803

13.5. Important Points:

- Visual Studio is the development tool which is used to design and develop the .NET applications. For using Visual Studio, the user has to first install the .NET framework on the system.
- In the older version of Windows OS like XP SP1, SP2 or SP3, .NET framework was integrated with the installation media.
- Windows 8, 8.1 or 10 do not provide a pre-installed version 3.5 or later of .NETFramework. Therefore, a version higher than 3.5 must be installed either from a Windows installation media or from the Internet on demand. Windows update will give recommendations to install the .NET framework.

13.6. C#.net Introduction:

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

The following reasons make C# a widely used professional language −

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

13.7. Strong Programming Features of C#:

Although C# constructs closely follow traditional high-level languages, C and C++ and being an object-oriented programming language. It has strong resemblance with Java, it has numerous strong programming features that make it endearing to a number of programmers worldwide.

Following is the list of few important features of C# −

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
- Assembly Versioning
- Properties and Events
- Delegates and Events Management
- Easy-to-use Generics
- Indexers
- Conditional Compilation
- Simple Multithreading
- LINQ and Lambda Expressions
- Integration with Windows

13.8. C# - Environment:

In this lecture, we will discuss the tools required for creating C# programming. We have already mentioned that C# is part of .Net framework and is used for writing .Net applications. Therefore, before discussing the available tools for running a C# program, let us understand how C# relates to the .Net framework.

13.9. The .Net Framework:

The .Net framework is a revolutionary platform that helps you to write the following types of applications −

- Windows applications
- Web applications
- Web services

The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: C#, C++, Visual Basic, Jscript, COBOL, etc. All these languages can access the framework as well as communicate with each other.

The .Net framework consists of an enormous library of codes used by the client languages such as C#. Following are some of the components of the .Net framework –

- Common Language Runtime (CLR)
- The .Net Framework Class Library
- Common Language Specification
- Common Type System
- Metadata and Assemblies
- Windows Forms
- ASP.Net and ASP.Net AJAX
- ADO.Net
- Windows Workflow Foundation (WF)
- Windows Presentation Foundation
- Windows Communication Foundation (WCF)
- LINQ

13.10. Integrated Development Environment (IDE) for C#:

Microsoft provides the following development tools for C# programming –

- Visual Studio 2010 (VS)
- Visual C# 2010 Express (VCE)
- Visual Web Developer

The last two are freely available from Microsoft official website. Using these tools, you can write all kinds of C# programs from simple command-line applications to more complex applications. You can also write C# source code files using a basic text editor, like Notepad, and compile the code into assemblies using the command-line compiler, which is again a part of the .NET Framework.

Visual C# Express and Visual Web Developer Express edition are trimmed down versions of Visual Studio and has the same appearance. They retain most features of Visual Studio. In this tutorial, we have used Visual C# 2010 Express.

13.11. Writing C# Programs on Linux or Mac OS:

Although the.NET Framework runs on the Windows operating system, there are some alternative versions that work on other operating systems. **Mono** is an open-source version of the .NET Framework which includes a C# compiler and runs on several operating systems, including various flavors of Linux and Mac OS.

The stated purpose of Mono is not only to be able to run Microsoft .NET applications cross-platform, but also to bring better development tools for Linux developers. Mono can be run on many operating systems including Android, BSD, iOS, Linux, OS X, Windows, Solaris, and UNIX.

13.12. C# - Program Structure:

Before we study basic building blocks of the C# programming language, let us look at a bare minimum C# program structure so that we can take it as a reference in upcoming lectures.

13.13. Creating Hello World Program:

A C# program consists of the following parts −

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

Let us look at a simple code that prints the words "Hello World" –

```
using System;

namespace HelloWorldApplication {
   class HelloWorld {
      static void Main(string[] args) {
         /* my first program in C# */
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

When this code is compiled and executed, it produces the following result −

```
Hello World
```

Let us look at the various parts of the given program –

- The first line of the program using System; - the using keyword is used to include the System namespace in the program. A program generally has multiple using statements.
- The next line has the namespace declaration. A namespace is a collection of classes. The HelloWorldApplication namespace contains the class HelloWorld.
- The next line has a class declaration, the class HelloWorld contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the HelloWorld class has only one method Main.
- The next line defines the Main method, which is the entry point for all C# programs. The Main method states what the class does when executed.
- The next line /*...*/ is ignored by the compiler and it is put to add comments in the program.
- The Main method specifies its behavior with the statement Console.WriteLine("Hello World");
- WriteLine is a method of the Console class defined in the System namespace. This statement causes the message "Hello, World!" to be displayed on the screen.
- The last line Console.ReadKey(); is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

It is worth to note the following points −

- C# is case sensitive.
- All statements and expression must end with a semicolon (;).
- The program execution starts at the Main method.
- Unlike Java, program file name could be different from the class name.

13.14. Compiling and Executing the Program:

If you are using Visual Studio.Net for compiling and executing C# programs, take the following steps −

- Start Visual Studio.
- On the menu bar, choose File -> New -> Project.
- Choose Visual C# from templates, and then choose Windows.
- Choose Console Application.
- Specify a name for your project and click OK button.
- This creates a new project in Solution Explorer.
- Write code in the Code Editor.

- Click the Run button or press F5 key to execute the project. A Command Prompt window appears that contains the line Hello World.

You can compile a C# program by using the command-line instead of the Visual Studio IDE −

- Open a text editor and add the above-mentioned code.
- Save the file as helloworld.cs
- Open the command prompt tool and go to the directory where you saved the file.
- Type csc helloworld.cs and press enter to compile your code.
- If there are no errors in your code, the command prompt takes you to the next line and generates helloworld.exe executable file.
- Type helloworld to execute your program.
- You can see the output Hello World printed on the screen.

13.15. C# - Basic Syntax:

C# is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, are said to be in the same class.

For example, let us consider a Rectangle object. It has attributes such as length and width. Depending upon the design, it may need ways for accepting the values of these attributes, calculating the area, and displaying details.

Let us look at implementation of a Rectangle class and discuss C# basic syntax –

```
using System;

namespace RectangleApplication {
   class Rectangle {

      // member variables
      double length;
      double width;

      public void Acceptdetails() {
         length = 4.5;
         width = 3.5;
      }
      public double GetArea() {
         return length * width;
      }
      public void Display() {
         Console.WriteLine("Length: {0}", length);
         Console.WriteLine("Width: {0}", width);
         Console.WriteLine("Area: {0}", GetArea());
      }
   }
   class ExecuteRectangle {
      static void Main(string[] args) {
         Rectangle r = new Rectangle();
         r.Acceptdetails();
         r.Display();
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result –

```
Length: 4.5
Width: 3.5
Area: 15.75
```

13.15.1. The *using* Keyword:

The first statement in any C# program is

```
using System;
```

The **using** keyword is used for including the namespaces in the program. A program can include multiple using statements.

13.15.2. The *class* Keyword:

The **class** keyword is used for declaring a class.

13.15.3. Comments in C#:

Comments are used for explaining code. Compilers ignore the comment entries. The multiline comments in C# programs start with /* and terminates with the characters */ as shown below −

```
/* This program demonstrates
The basic syntax of C# programming
```

Language */

Single-line comments are indicated by the '//' symbol. For example,

}//end class Rectangle

13.15.4. Member Variables:

Variables are attributes or data members of a class, used for storing data. In the preceding program, the *Rectangle* class has two member variables named *length* and *width*.

13.15.5. Member Functions:

Functions are set of statements that perform a specific task. The member functions of a class are declared within the class. Our sample class Rectangle contains three member functions: *AcceptDetails*, *GetArea* and *Display*.

13.15.6. Instantiating a Class:

In the preceding program, the class *ExecuteRectangle* contains the *Main()*method and instantiates the *Rectangle* class.

13.15.7. Identifiers:

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in C# are as follows –

- A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
- It must not contain any embedded space or symbol such as? - + ! @ # % ^ & * ( ) [ ] { } . ; : " ' / and \. However, an underscore (_) can be used.
- It should not be a C# keyword.

13.15.8. C# Keywords:

Keywords are reserved words predefined to the C# compiler. These keywords cannot be used as identifiers. However, if you want to use these keywords as identifiers, you may prefix the keyword with the @ character.

In C#, some identifiers have special meaning in context of code, such as get and set are called contextual keywords.

The following table lists the reserved keywords and contextual keywords in C# −

Reserved Keywords						
abstract	as	base	bool	break	byte	case

catch	char	checked	class	const	continue	decimal
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	in	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					
Contextual Keywords						
add	alias	ascending	descending	dynamic	from	get
global	group	into	join	let	orderby	partial (type)
partial (method)	remove	select	set			

13.16. C# - Data Types:

The variables in C#, are categorized into the following types −

- Value types
- Reference types
- Pointer types

13.16.1. Value Type:

Value type variables can be assigned a value directly. They are derived from the class System.ValueType.

The value types directly contain data. Some examples are int, char, and float, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an int type, the system allocates memory to store the value.

The following table lists the available value types in C# 2010 –

Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal	(-7.9 x 10^{28} to 7.9 x 10^{28}) / $10^{0}$to 28	0.0M

	values with 28-29 significant digits		
double	64-bit double-precision floating point type	(+/-)5.0 x 10^{-324} to (+/-)1.7 x 10^{308}	0.0D
float	32-bit single-precision floating point type	-3.4 x 10^{38} to + 3.4 x 10^{38}	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

To get the exact size of a type or a variable on a particular platform, you can use the sizeof method. The expression sizeof (type) yields the storage size of the object or type in bytes. Following is an example to get the size of int type on any machine –

```
using System;

namespace DataTypeApplication {
   class Program {
      static void Main(string[] args) {
         Console.WriteLine("Size of int: {0}", sizeof(int));
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
Size of int: 4
```

13.16.2. Reference Type:

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.

In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value. Example of **built-in** reference types are: **object**, **dynamic**, and **string**.

13.16.3. Object Type:

The Object Type is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System.Object class. The object types can be assigned values of any other types, value types, reference types, predefined or user-defined types. However, before assigning values, it needs type conversion.

When a value type is converted to object type, it is called **boxing** and on the other hand, when an object type is converted to a value type, it is called **unboxing**.

```
object obj;
obj = 100; // this is boxing
```

13.16.4. Dynamic Type:

You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

Syntax for declaring a dynamic type is −

```
dynamic <variable_name> = value;
```

For example,

```
dynamic d = 20;
```

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

13.16.5. String Type:

The String Type allows you to assign any string values to a variable. The string type is an alias for the System.String class. It is derived from object type. The value for a string type can be assigned using string literals in two forms: quoted and @quoted.

For example,

```
String str = "Guru Nanak";
```

A @quoted string literal looks as follows −

```
@"Guru Nanak";
```

The user-defined reference types are: class, interface, or delegate.

13.16.6. Pointer Type:

Pointer type variables store the memory address of another type. Pointers in C# have the same capabilities as the pointers in C or C++.
Syntax for declaring a pointer type is −

```
type* identifier;
```

For example,

```
char* cptr;
int* iptr;
```

13.17. C# - Variables:

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.
The basic value types provided in C# can be categorized as –

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

C# also allows defining other value types of variable such as enum and reference types of variables such as class.

13.17.1. Defining Variables:

Syntax for variable definition in C# is −

```
<data_type><variable_list>;
```

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.
Some valid variable definitions are shown here −

```
int i, j, k;
char c, ch;
```

```
float f, salary;
double d;
```

You can initialize a variable at the time of definition as −

```
int i = 100;
```

13.17.2. Initializing Variables:

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is −

```
variable_name = value;
```

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as −

```
<data_type><variable_name> = value;
```

Some examples are −

```
int d = 3, f = 5;    /* initializing d and f. */
byte z = 22;         /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';        /* the variable x has the value 'x'. */
```

It is a good programming practice to initialize variables properly, otherwise sometimes program may produce unexpected result.

The following example uses various types of variables –

```
using System;
namespace VariableDefinition {
   class Program {
      static void Main(string[] args) {
         short a;
         int b ;
         double c;
         /* actual initialization */
         a = 10;
         b = 20;
         c = a + b;
         Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
         Console.ReadLine();
      }
```

```
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
a = 10, b = 20, c = 30
```

13.17.3. Accepting Values from User:

The Console class in the System namespace provides a function ReadLine() for accepting input from the user and store it into a variable.
For example,

```
int num;
num = Convert.ToInt32(Console.ReadLine());
```

The function Convert.ToInt32() converts the data entered by the user to int data type, because Console.ReadLine() accepts the data in string format.

13.17.4. Lvalue and Rvalue Expressions in C#:

There are two kinds of expressions in C# −

- lvalue − An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
- rvalue − An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and hence they may appear on the left-hand side of an assignment. Numeric literals are rvalues and hence they may not be assigned and can not appear on the left-hand side. Following is a valid C# statement −

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error −

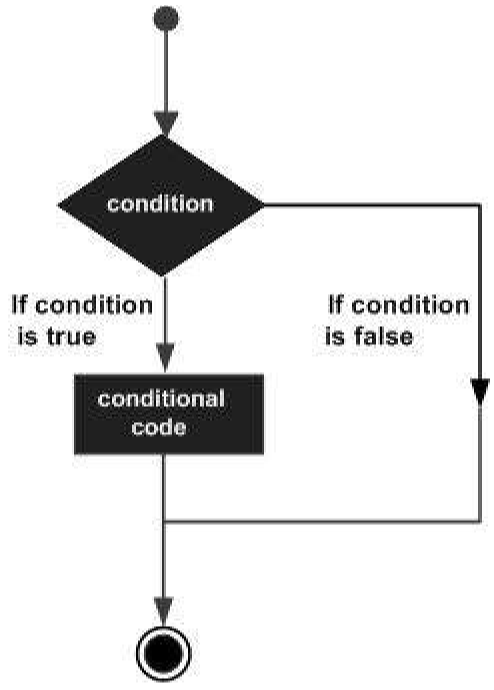
```
10 = 20;
```

Lecture 14: Basic Syntax of Conditional Statements in C#, Loops and Methods

14.1. C# - Decision Making:

Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages –



C# provides following types of decision making statements.

Sr. No.	Statement & Description
1	**if statement** An **if statement** consists of a boolean expression followed by one or more statements.
2	**if...else statement** An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
3	**nested if statements** You can use one **if** or **else if** statement inside another **if** or **else if**statement(s).
4	**switch statement** A **switch** statement allows a variable to be tested for equality against a list of values.
5	**nested switch statements** You can use one **switch** statement inside another **switch**statement(s).

14.1.1. C# - if Statement:

An if statement consists of a boolean expression followed by one or more statements.

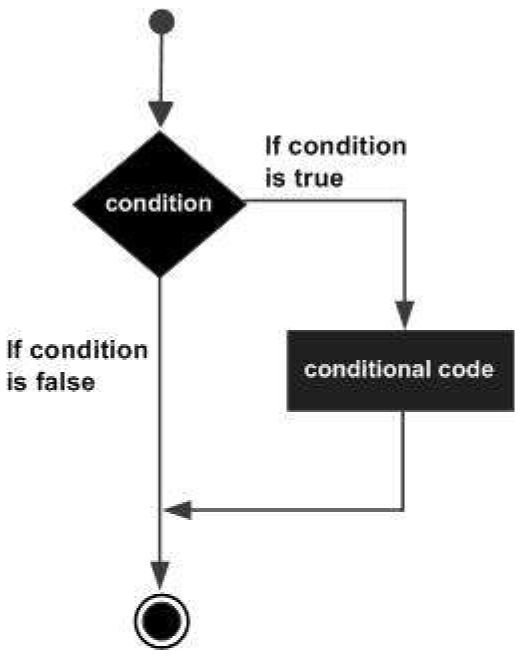
Syntax

The syntax of an if statement in C# is −

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
}
```

If the boolean expression evaluates to true, then the block of code inside the if statement is executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement(after the closing curly brace) is executed.

Flow Diagram



Example

```
using System;

namespace DecisionMaking {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 10;
         /* check the boolean condition using if statement */
         if (a < 20) {
           /* if condition is true then print the following */
           Console.WriteLine("a is less than 20");
         }
         Console.WriteLine("value of a is : {0}", a);
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
a is less than 20;
value of a is : 10
```

14.1.2. C# - if...else Statement:

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

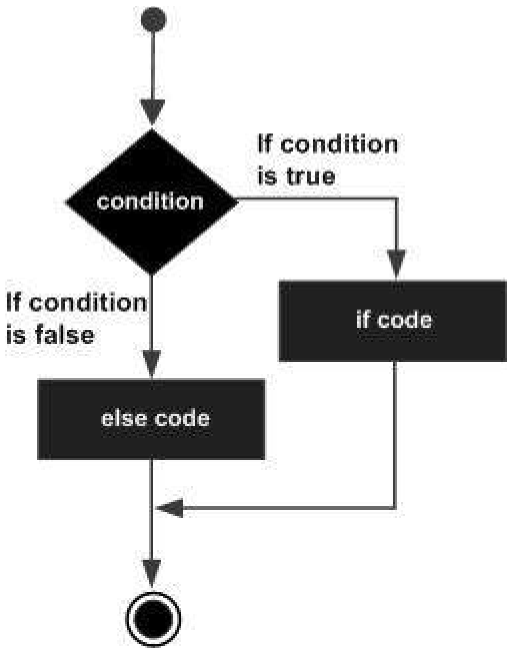
Syntax

The syntax of an if...else statement in C# is −

```
if(boolean_expression) {
  /* statement(s) will execute if the boolean expression is true */
} else {
  /* statement(s) will execute if the boolean expression is false */
}
```

If the boolean expression evaluates to true, then the if block of code is executed, otherwise else block of code is executed.

Flow Diagram



Example

```
using System;

namespace DecisionMaking {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 100;

         /* check the boolean condition */
         if (a < 20) {
            /* if condition is true then print the following */
            Console.WriteLine("a is less than 20");
         } else {
            /* if condition is false then print the following */
            Console.WriteLine("a is not less than 20");
         }
         Console.WriteLine("value of a is : {0}", a);
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
a is not less than 20;
value of a is : 100
```

14.1.3. The if...else if...else Statement:

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if, else if, else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax

The syntax of an if...else if...else statement in C# is −

```
if(boolean_expression 1) {
  /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2) {
  /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3) {
  /* Executes when the boolean expression 3 is true */
} else {
  /* executes when the none of the above condition is true */
}
```

Example

```
using System;

namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 100;

      /* check the boolean condition */
      if (a == 10) {
        /* if condition is true then print the following */
        Console.WriteLine("Value of a is 10");
      }
      else if (a == 20) {
        /* if else if condition is true */
        Console.WriteLine("Value of a is 20");
      }
      else if (a == 30) {
        /* if else if condition is true  */
        Console.WriteLine("Value of a is 30");
      } else {
        /* if none of the conditions is true */
        Console.WriteLine("None of the values is matching");
      }
      Console.WriteLine("Exact value of a is: {0}", a);
      Console.ReadLine();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
None of the values is matching
Exact value of a is: 100
```

14.1.4. C# - Nested if Statements:

It is always legal in C# to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Syntax

The syntax for a nested if statement is as follows −

```
if( boolean_expression 1) {
  /* Executes when the boolean expression 1 is true */
  if(boolean_expression 2) {
    /* Executes when the boolean expression 2 is true */
  }
}
```

You can nest else if...else in the similar way as you have nested if statement.

Example

```
using System;

namespace DecisionMaking {
   class Program {
      static void Main(string[] args) {
         //* local variable definition */
         int a = 100;
         int b = 200;

         /* check the boolean condition */
         if (a == 100) {

            /* if condition is true then check the following */
            if (b == 200) {
               /* if condition is true then print the following */
               Console.WriteLine("Value of a is 100 and b is 200");
            }
         }
         Console.WriteLine("Exact value of a is : {0}", a);
         Console.WriteLine("Exact value of b is : {0}", b);
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

14.1.5. C# - Switch Statement:

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

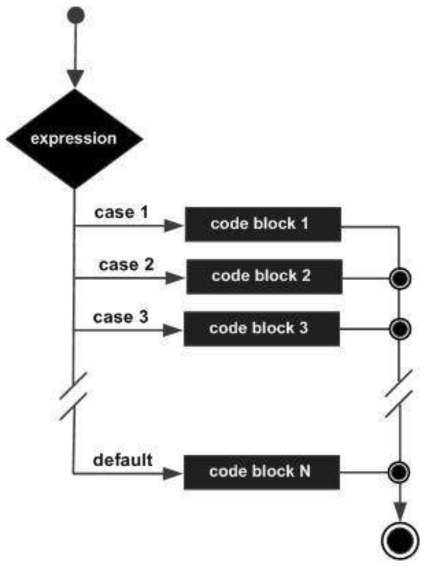
The syntax for a **switch** statement in C# is as follows −

```
switch(expression) {
  case constant-expression1  :
    statement(s);
    break;
  case constant-expression2  :
  case constant-expression3  :
    statement(s);
    break;

  /* you can have any number of case statements */
  default : /* Optional */
  statement(s);
}
```

The following rules apply to a **switch** statement –

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, then it will raise a compile time error.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true.

Flow Diagram



Example

```
using System;

namespace DecisionMaking {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         char grade = 'B';

         switch (grade) {
            case 'A':
               Console.WriteLine("Excellent!");
               break;
            case 'B':
            case 'C':
               Console.WriteLine("Well done");
               break;
            case 'D':
               Console.WriteLine("You passed");
               break;
            case 'F':
               Console.WriteLine("Better try again");
               break;
               default:
```

```
            Console.WriteLine("Invalid grade");
              break;
          }
          Console.WriteLine("Your grade is  {0}", grade);
          Console.ReadLine();
        }
      }
    }
```

When the above code is compiled and executed, it produces the following result −

```
Well done
Your grade is B
```

14.1.6. C# - nested switch Statements:

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

Syntax

The syntax for a **nested switch** statement is as follows −

```
switch(ch1) {
  case 'A':
  Console.WriteLine("This A is part of outer switch" );

  switch(ch2) {
    case 'A':
      Console.WriteLine("This A is part of inner switch" );
      break;
    case 'B': /* inner B case code */
  }
  break;
  case 'B': /* outer B case code */
}
```

Example

```
using System;

namespace DecisionMaking {
  class Program {
      static void Main(string[] args) {
         int a = 100;
         int b = 200;

         switch (a) {
            case 100:
            Console.WriteLine("This is part of outer switch ");

            switch (b) {
               case 200:
               Console.WriteLine("This is part of inner switch ");
               break;
            }
            break;
         }
         Console.WriteLine("Exact value of a is : {0}", a);
         Console.WriteLine("Exact value of b is : {0}", b);
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

14.1.7. The ? : Operator:

We have covered conditional operator ? : in previous chapter which can be used to replace if...else statements. It has the following general form −

```
Exp1 ? Exp2 : Exp3;
```

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

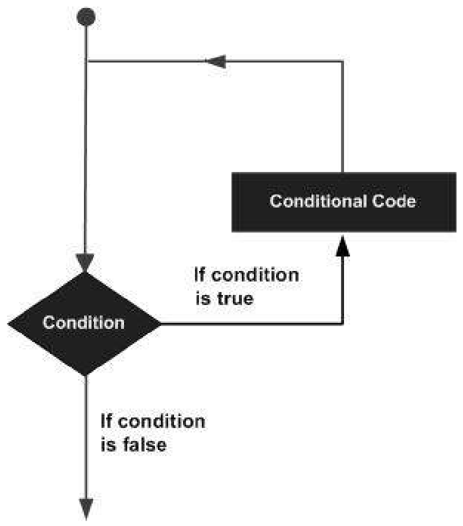
The value of a ? expression is determined as follows: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

14.2. C# - Loops:

There may be a situation, when you need to execute a block of code several number of times. In general, the statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or a group of statements multiple times and following is the general from of a loop statement in most of the programming languages –



C# provides following types of loop to handle looping requirements.

Sr. No.	Loop Type & Description
1	**while loop** It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
2	**for loop** It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	**do...while loop** It is similar to a while statement, except that it tests the condition at the end of the loop body
4	**nested loops** You can use one or more loop inside any another while, for or do..while loop.

14.2.1. C# - While Loop:

A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true.

Syntax

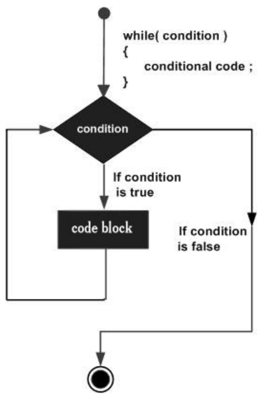
The syntax of a **while** loop in C# is −

```
while(condition) {
   statement(s);
}
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

Example

```
using System;

namespace Loops {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 10;

         /* while loop execution */
         while (a < 20) {
            Console.WriteLine("value of a: {0}", a);
```

```
          a++;
        }
        Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

14.2.2. C# - For Loop:

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for** loop in C# is −

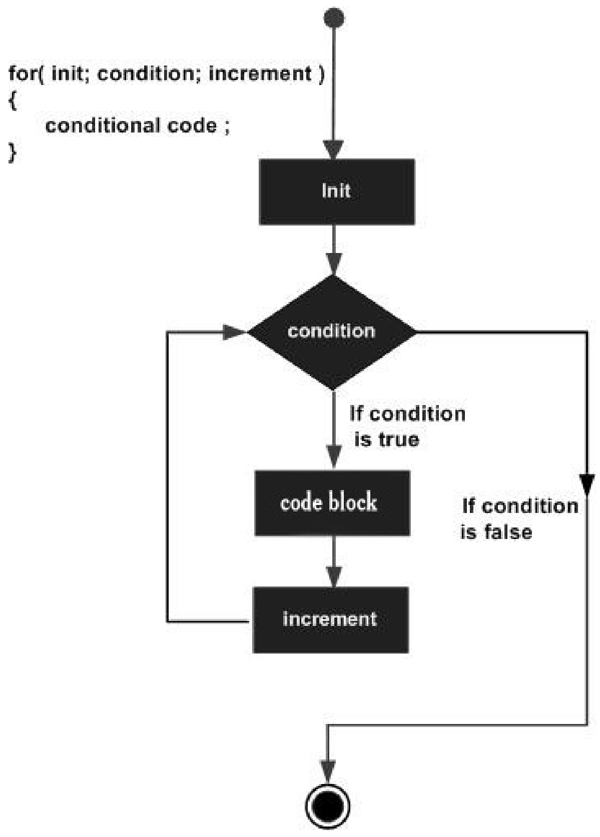
```
for ( init; condition; increment ) {
   statement(s);
}
```

Here is the flow of control in a for loop –

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.

- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again testing for a condition). After the condition becomes false, the for loop terminates.

Flow Diagram



Example

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {

      /* for loop execution */
      for (int a = 10; a < 20; a = a + 1) {
```

```
    Console.WriteLine("value of a: {0}", a);
   }
   Console.ReadLine();
  }
 }
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

14.2.3. C# - Do...While Loop:

Unlike for and while loops, which test the loop condition at the start of the loop, the do...while loop checks its condition at the end of the loop.

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

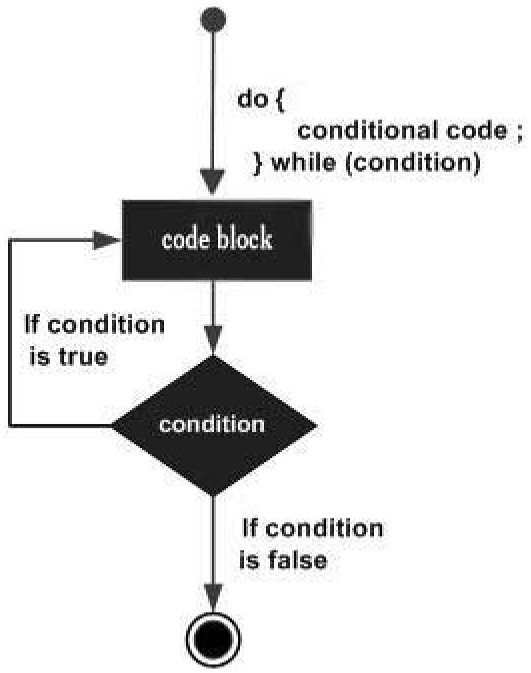
The syntax of a do...while loop in C# is −

```
do {
  statement(s);
} while( condition );
```

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

Flow Diagram



Example

```
using System;

namespace Loops {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 10;

         /* do loop execution */
         do {
            Console.WriteLine("value of a: {0}", a);
            a = a + 1;
         }
         while (a < 20);
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

14.2.4. C# - Nested Loops:

C# allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

The syntax for a nested for loop statement in C# is as follows −

```
for ( init; condition; increment ) {
  for ( init; condition; increment ) {
    statement(s);
  }
  statement(s);
}
```

The syntax for a nested while loop statement in C# is as follows −

```
while(condition) {
  while(condition) {
    statement(s);
  }
  statement(s);
}
```

The syntax for a nested do...while loop statement in C# is as follows −

```
do {
  statement(s);
  do {
    statement(s);
  }
  while( condition );
}
while( condition );
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Example

The following program uses a nested for loop to find the prime numbers from 2 to 100 –

```
using System;

namespace Loops {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int i, j;

         for (i = 2; i < 100; i++) {
            for (j = 2; j <= (i / j); j++)
            if ((i % j) == 0) break; // if factor found, not prime
            if (j > (i / j)) Console.WriteLine("{0} is prime", i);
         }
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
```

```
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

14.2.5. Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C# provides the following control statements.

Sr.No.	Control Statement & Description
1	**break statement** Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

14.2.5.1. C# - Break Statement:

The break statement in C# has following two usage –

- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement.

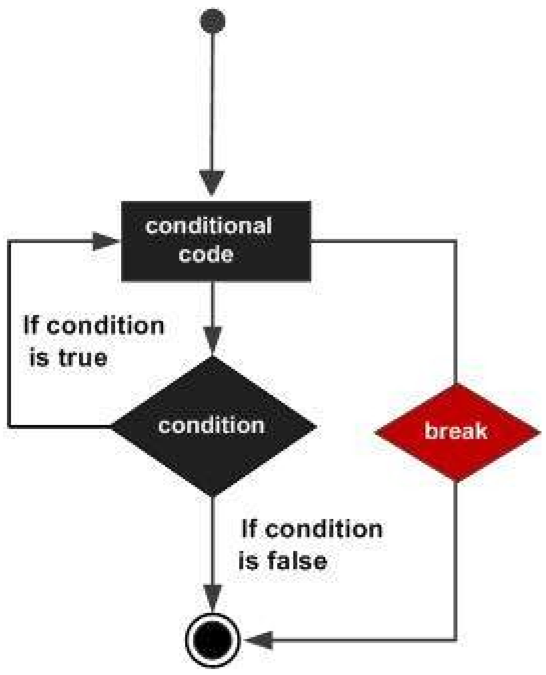
If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

The syntax for a break statement in C# is as follows −

```
break;
```

Flow Diagram



Example

```
using System;

namespace Loops {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 10;

         /* while loop execution */
         while (a < 20) {
            Console.WriteLine("value of a: {0}", a);
            a++;

            if (a > 15) {
               /* terminate the loop using break statement */
               break;
            }
         }
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

14.2.5.2. C# - Continue Statement:

The continue statement in C# works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

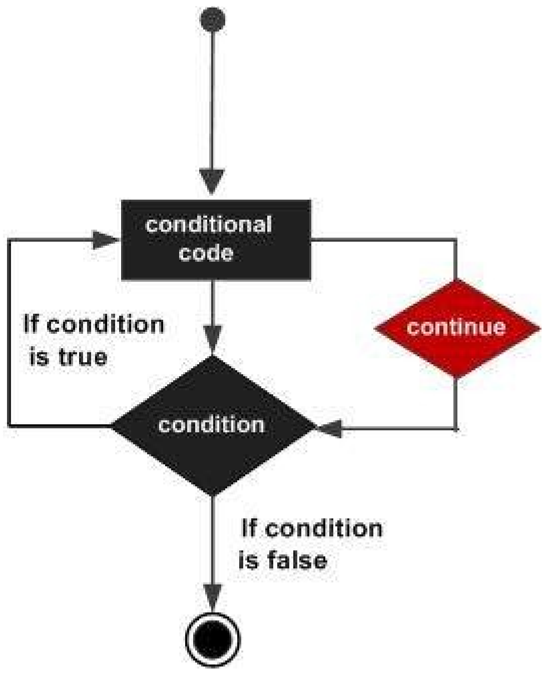
For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control passes to the conditional tests.

Syntax

The syntax for a continue statement in C# is as follows −

```
continue;
```

Flow Diagram



Example

```
using System;

namespace Loops {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 10;

         /* do loop execution */
         do {
            if (a == 15) {
               /* skip the iteration */
               a = a + 1;
               continue;
            }
            Console.WriteLine("value of a: {0}", a);
            a++;
         }
         while (a < 20);
      Console.ReadLine();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

14.2.6. Infinite Loop:

A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

Example

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      for (; ; ) {
        Console.WriteLine("Hey! I am Trapped");
      }
    }
  }
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but programmers more commonly use the for(;;) construct to signify an infinite loop.

14.3. C# - Methods:

A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.

To use a method, you need to −

- Define the method
- Call the method

14.3.1. Defining Methods in C#:

When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows −

```
<Access Specifier><Return Type><Method Name>(Parameter List) {
  Method Body
}
```

Following are the various elements of a method −

- Access Specifier − This determines the visibility of a variable or a method from another class.
- Return type − A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- Method name − Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- Parameter list − Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- Method body − This contains the set of instructions needed to complete the required activity.

Example

Following code snippet shows a function FindMax that takes two integer values and returns the larger of the two. It has public access specifier, so it can be accessed from outside the class using an instance of the class.

```
class NumberManipulator {

  public int FindMax(int num1, int num2) {
    /* local variable declaration */
    int result;
```

```
      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
   ...
}
```

14.3.2. Calling Methods in C#:

You can call a method using the name of the method. The following example illustrates this –

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int FindMax(int num1, int num2) {
         /* local variable declaration */
         int result;

         if (num1 > num2)
            result = num1;
         else
            result = num2;
         return result;
      }

      static void Main(string[] args) {
         /* local variable definition */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //calling the FindMax method
         ret = n.FindMax(a, b);
      Console.WriteLine("Max value is : {0}", ret );
      Console.ReadLine();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
Max value is : 200
```

You can also call public method from other classes by using the instance of the class. For example, the method *FindMax* belongs to the *NumberManipulator*class, you can call it from another class *Test*.

```
using System;

namespace CalculatorApplication {
  class NumberManipulator {
    public int FindMax(int num1, int num2) {
      /* local variable declaration */
      int result;

      if(num1 > num2)
        result = num1;
      else
        result = num2;

      return result;
    }
  }
  class Test {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 100;
      int b = 200;
      int ret;
      NumberManipulator n = new NumberManipulator();

      //calling the FindMax method
      ret = n.FindMax(a, b);
      Console.WriteLine("Max value is : {0}", ret );
      Console.ReadLine();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
Max value is : 200
```

14.3.3. Recursive Method Call:

A method can call itself. This is known as **recursion**. Following is an example that calculates factorial for a given number using a recursive function –

```
using System;

namespace CalculatorApplication {
  class NumberManipulator {
    public int factorial(int num) {
      /* local variable declaration */
      int result;
      if (num == 1) {
        return 1;
      } else {
        result = factorial(num - 1) * num;
        return result;
      }
    }
    static void Main(string[] args) {
      NumberManipulator n = new NumberManipulator();
      //calling the factorial method {0}", n.factorial(6));
      Console.WriteLine("Factorial of 7 is : {0}", n.factorial(7));
      Console.WriteLine("Factorial of 8 is : {0}", n.factorial(8));
      Console.ReadLine();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
Factorial of 6 is: 720
Factorial of 7 is: 5040
Factorial of 8 is: 40320
```

14.3.4. Passing Parameters to a Method:

When method with parameters is called, you need to pass the parameters to the method. There are three ways that parameters can be passed to a method –

Sr. No.	Mechanism & Description
1	**Value parameters** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
2	**Reference parameters** This method copies the reference to the memory location of an argument into the formal parameter. This means that changes made to the parameter affect the argument.
3	**Output parameters** This method helps in returning more than one value.