Unit # 1: Introduction to ASP.NET

Introduction to .NET Platform and Web:

- Introduction to ASP (Server-side Technology)
- .NET Framework (FCL and CLR)
- Overview of IIS
- Processing of ASP.NET page (Execution model)
- Features of .NET IDE
- Features of ASP.NET
- Working with ASP.NET
- Coding Model (Inline and Code-behind)
- Introduction to Web-Forms and its Events
- ASP.NET Built-in directory structure: App_data, App_code, Bin

Application Configuration: Global.asax file, Web.config

Common properties: AccessKey, BackColor, BorderWidth, BorderStyle, CSSClass, Enabled, Font, ForeColor, Height, TabIndex, Tooltip, Width, ID, Runat, Text

Using Visual C# in ASP. NET:

- Introduction
- Variables
- Data Types
- Value Types
- Scope of Variables
- Operators
- OOPs Concepts: (Encapsulations, Inheritance, Polymorphism and Abstraction)

Introduction to ASP

ASP stands for Active Server Pages. ASP is a development framework for building web pages.

ASP supports many different development models:

- Classic ASP
- ASP.NET Web Forms
- ASP.NET MVC
- ASP.NET Web Pages
- ASP.NET API
- ASP.NET Core

Classic ASP

ASP (Classic ASP) was introduced in 1998 as Microsoft's first server side scripting language.

Classic ASP pages have the file extension .asp and are normally written in VBScript.

Features of ASP

- Edit, change, add content, or customize any web page
- Respond to user queries or data submitted from HTML forms
- Access databases or other server data and return results to a browser
- Provide web security since ASP code cannot be viewed in a browser
- Offer simplicity and speed

ASP. NET Web Forms

ASP.NET Web Forms is an event driven application model.

ASP.NET Web Forms is **not** a part of the new ASP.NET Core.

ASP. NET MVC

ASP.NET MVC is an MVC application model (Model-View-Controller).

ASP.NET MVC is being merged into the new ASP.NET Core.

ASP. NET Web Pages

Web Pages are one of many programming models for creating ASP.NET web sites and web applications. ASP.NET Web Pages is an SPA application model (Single Page Application).

The SPA model is quite similar to PHP and Classic ASP. Web Pages provide an easy way to combine HTML, CSS, and server code:

- Easy to learn, understand, and use
- Uses an SPA application model (Single Page Application)
- Similar to PHP and Classic ASP
- VB (Visual Basic) or C# (C sharp) scripting languages

In addition, Web Pages applications are easily extendable with programmable helpers for databases, videos, graphics, social networking and more.

ASP. NET API

ASP.NET API is an API application model (Application Programming Interface).

ASP.NET API is being merged into the new ASP.NET Core.

ASP. NET Core

ASP.NET Core was released in 2016.

ASP.NET Core merges ASP.NET MVC, ASP.NET Web API, and ASP.NET Web Pages into one application framework.

What is ASP.Net?

ASP.NET is the advanced version of ASP. It is a server side technology. ASP.Net is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web application for PC, as well as mobile devices.ASP.NET is run inside the IIS(Internet Information Services).

ASP.Net works on top of the HTTP protocol and uses the HTTP commands and policies to set a browser-to-server two-way communication and cooperation.

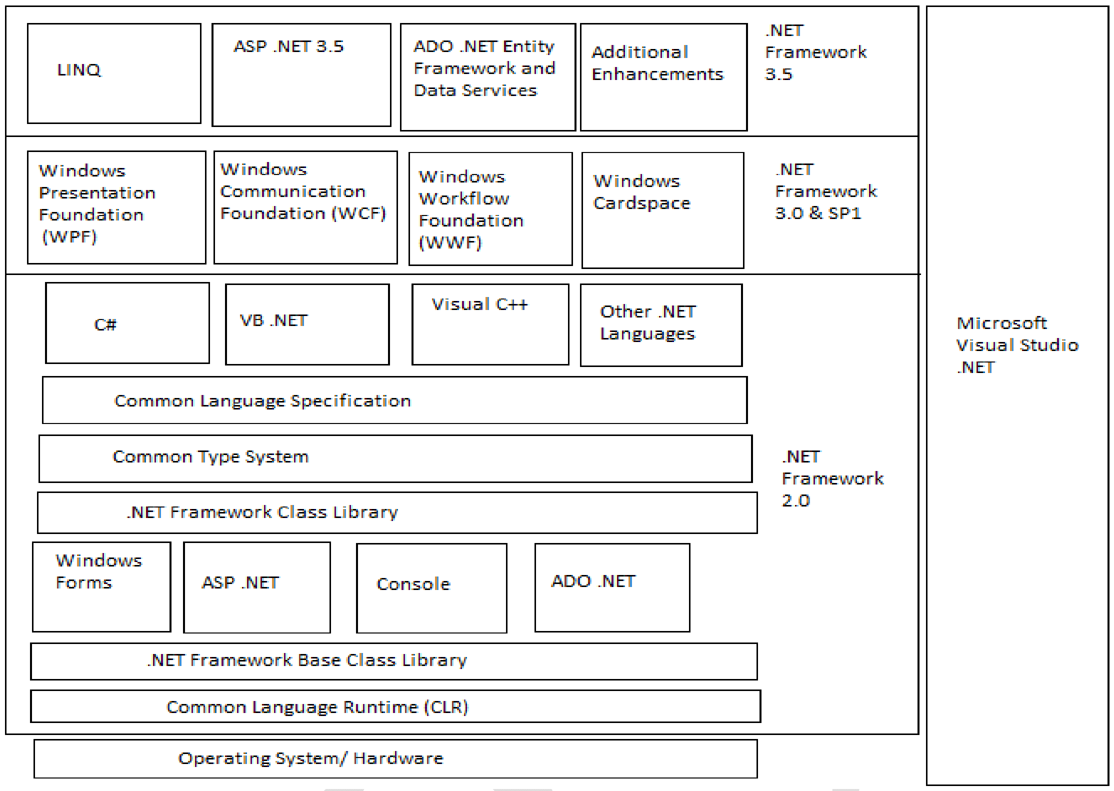
ASP.Net is a part of Microsoft .Net platform. ASP.Net applications are complied codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.Net application codes could be written in either of the following languages:

- C#
- Visual Basic .Net
- Jscript
- J#

ASP.Net is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls like text boxes, buttons and labels for assembling, configuring and manipulating code to create HTML pages.

.Net Framework


LINQ
ASP .NET 3.5
ADO .NET Entity Framework and Data Services
Additional Enhancements
.NET Framework 3.5
Windows Presentation Foundation (WPF)
Windows Communication Foundation (WCF)
Windows Workflow Foundation (WWF)
Windows Cardspace
.NET Framework 3.0 & SP1
C#
VB .NET
Visual C++
Other .NET Languages
Common Language Specification
Common Type System
.NET Framework Class Library
.NET Framework 2.0
Windows Forms
ASP .NET
Console
ADO .NET
.NET Framework Base Class Library
Common Language Runtime (CLR)
Operating System/ Hardware
Microsoft Visual Studio .NET

CLR (Common Language Runtime)

The .NET Framework provides a runtime environment called the Common Language Runtime or CLR (similar to the Java Virtual Machine or JVM in Java), which handles the execution of code and provides useful services for the implementation of the program.

The CLR is the execution engine for .NET applications and serves as the interface between .NET applications and the operating system. The CLR provides many services such as:

- Loads and executes code
- Converts intermediate language to native machine code
- Manages memory and objects
- Enforces code and access security
- Handles exceptions
- Interfaces between managed code, COM objects, and DLLs

- Provides type-checking
- Provides code meta data (Reflection)
- Provides profiling, debugging, etc.
- Separates processes and memory

FCL (Framework Class Library)

It is also known as a base class library. .NET applications, components and controls are built on the foundation of .NET Framework types .These entities perform the following functions

- Representing base data types and exceptions
- Encapsulating data structures
- Performing input/output operations
- Accessing information about loaded types
- Calling .NET Framework security tasks
- Facilitating data access , rich client-side Graphical User Interface(GUI) and server control client side GUI

The .Net Framework class library (FCL) organized in a hierarchical tree structure and it is divided into Namespaces. Namespaces is a logical grouping of types for the purpose of identification. Framework class library (FCL) provides the consistent base types that are used across all .NET enabled languages. The Classes are accessed by namespaces, which reside within Assemblies.

CTS (Common Type System)

- The CLS is a common platform that integrates code and components from multiple .NET programming languages.
- CTS allow programs written in different programming languages to easily share information.
- CLS forms a subset of CTS. This implies that all the rules that apply to CTS also apply to CLS also.
- It defines rules that a programming language must follow to ensure that objects written in different programming languages can interact with each other.

- CTS provide cross language integration.
- The common type system supports two general categories of **types**:

1. Value Type
2. Reference Type

Value Type: Stores directly data on stack. In built data type. For example: int a;

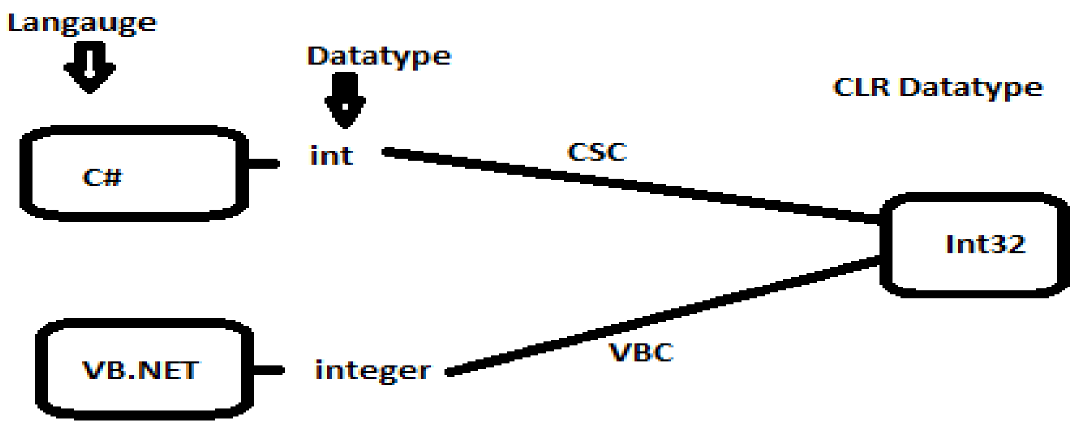
Reference Type: Store a reference to the value's memory address, and are allocated on the heap. For example: Dim obj as New OledbConnection.

The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS).

Common Type System (CTS) describes the data types that can be used by managed code. CTS define how these types are declared, used and managed in the runtime. It facilitates cross-language integration, type safety, and high-performance code execution. The rules defined in CTS can be used to define your own classes and values.

OR we can also understand like, CTS deals with the data type. So here we have several languages and each and every language has its own data type and one language data type cannot be understandable by other languages but .NET Framework language can understand all the data types.

C# has an **int** data type and VB.NET has **Integer** data type. Hence a variable **declared as an int in C# and Integer in VB.NET**, finally after compilation, uses the same structure **Int32 from CTS.**



Common Language Specification (CLS)

- CLS includes basic language features needed by almost all applications.
- CLS id a subset of the Common Type System.
- It serves as a guide for library writes and compiler writers.
- The CLS is also important to application developers who are writing code that will be used by other developers.

Microsoft Intermediate Language

- When you compile your Visual Basic .NET source code, it is changed to an intermediate language (IL) that the CLR and all other .NET development environments understand.
- All .NET languages compile code to this IL, which is known as Microsoft Intermediate Language, MSIL, or IL.
- MSIL is a common language in the sense that the same programming tasks written with different .NET languages produce the same IL code.
- At the IL level, all .NET code is the same regardless of whether it came from C++ or Visual Basic.
- When a compiler produces Microsoft Intermediate Language (MSIL), it also produces Metadata.
- The Microsoft Intermediate Language (MSIL) and Metadata are contained in a portable executable (PE) file.

- Microsoft Intermediate Language (MSIL) includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations

Meta Data and Assembly

A metadata is binary information, which describes your program, stored in a CLR portable Executable file or in memory. Metadata contains the following:

- Assembly information, such as its identity which can be name , version , culture , public key ,the types of assembly , other referenced assemblies and security permission.
- Information about types such as name , visibility , base class , interfaces used and members
- Attributes information that modifies the types and members of a class.

Contents of an Assembly

- Assembly Manifest
- Assembly Name
- Version Information
- Types
- Locale
- Cryptographic Hash
- Security Permissions

There are two kinds of assemblies in .NET:

- private and
- shared

Private assemblies Is the assembly which is used by application only, normally it resides in your application folder directory.

Shared assemblies - It resides in GAC, so that anyone can use this assembly. Public assemblies are always sharing the common functionalities with other applications.

Global Access cache (GAC)

It is a central place for registering assemblies, so the different applications on the computer can use it later on. It is the machine wide code cache in any computer in which CLR is installed.

Windows Forms

This contains the graphical representation of any window displayed in the application.

ASP.Net and ASP.Net AJAX

ASP.Net is the web development model and AJAX is an extension of ASP.Net for developing and implementing AJAX functionality. ASP.Net AJAX contains the components that allow the developer to update data on a website without a complete reload of the page.

ADO.Net

It is the technology used for working with data and databases. It provides accesses to data sources like SQL server, OLE DB, XML etc. The ADO .Net allows connection to data sources for retrieving, manipulating and updating data.

Windows Workflow Foundation (WF)

It helps in building workflow based applications in Windows. It contains activities, workflow runtime, workflow designer and a rules engine.

Windows Presentation Foundation

It provides a separation between the user interface and the business logic. It helps in developing visually stunning interfaces using documents, media, two and three dimensional graphics, animations and more.

Windows Communication Foundation (WCF)

It is the technology used for building and running connected systems.

Windows CardSpace

It provides safety of accessing resources and sharing personal information on the internet.

LINQ

It is LANQUAGE INTEGRATED QUERY. It imparts data querying capabilities to .Net languages. It defines a common syntax and a programming model to query different types of data using a common language.

Understanding ASP.NET Controls

ASP.NET actually provides *two* sets of server-side controls that you can incorporate into your web forms. These two different types of controls play subtly different roles:

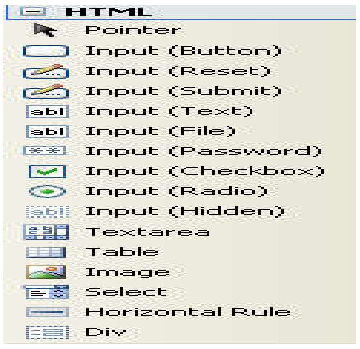
HTML server controls

These are server-based equivalents for standard HTML elements. These controls are ideal for web programmer who prefers to work with familiar HTML tags (at least at first).

HTML server controls provide an object interface for standard HTML elements. They provide three key features:

1. **They generate their own interface:** You can set properties in code, and the underlying HTML tag is created automatically when the page is rendered and sent to the client.
2. **They retain their state:** Because the Web is stateless, ordinary web pages need to do a lot of work to store information between requests. HTML server controls handle this task automatically.
3. **They fire server-side events:** For example, buttons fire an event when clicked; text boxes fire an event when the text they contain is modified, and so on. Your code can respond to these events, just like ordinary controls in a Windows application.

HTML server controls are ideal when you're performing a quick translation to add server side code to an existing HTML page.



Web controls: These are similar to the HTML server controls, but they provide a richer object model with a variety of properties for style and formatting details. They also provide more events and more closely resemble the controls used for Windows development. Web controls also feature some user interface elements that have no direct HTML equivalent, such as the Grid View, Calendar, and validation controls.

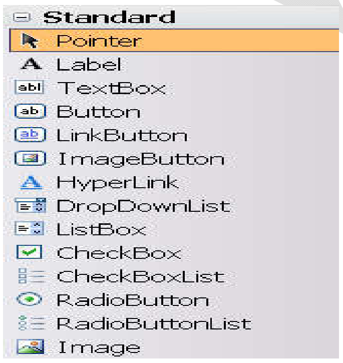
The ASP.NET Framework contains more than 90 controls. These controls can be divided into seven groups:

- Standard Controls— enable you to render standard form elements such as buttons, input fields, and labels.

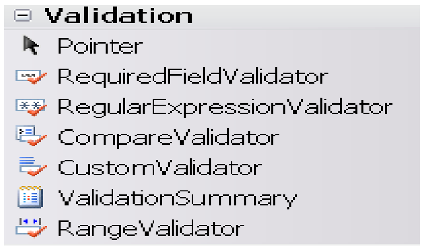
They are server side objects. They are programmable objects that act as user interfaces (UI) elements on a web page. The class of these controls is System.Web.UI.WebControls.

Syntax:

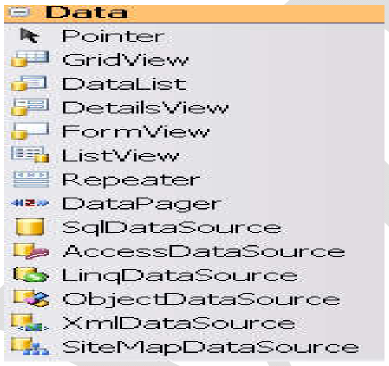
```
<asp:Button ID="Button1" runat="server" Text="Button" />
```



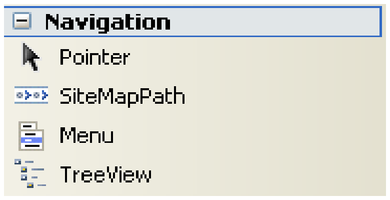
- Validation Controls— enable you to validate data before you submit the data to the server. For example, you can use a `RequiredFieldValidator` control to check whether a user entered a value for a required input field. Validation Controls are attached to input controls to test the data the user enters into it. They are used to test the input.



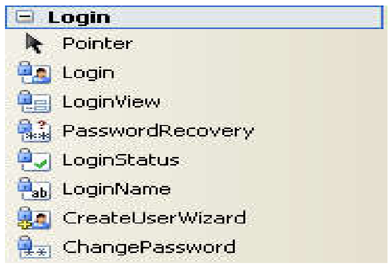
- Rich Controls— enable you to render things such as calendars, file upload buttons, rotating banner advertisements, and multistep wizards.
- Data Controls— enable you to work with data such as database data. For example, you can use these controls to submit new records to a database table or display a list of database records.



- Navigation Controls— enable you to display standard navigation elements such as menus, tree views, and bread crumb trails.



- Login Controls— Enables you to display login, change password, and registration forms.



Adding Web Control

To add an ASP.NET web control, drag the control you want from the toolbox on the left of IDE. In web form controls are positioned line by line, like in a word processor document. To organize several controls in design view, spaces are added or enter key can be pressed to position elements at proper location.

As you add web controls to the design surface, Visual Studio automatically adds the corresponding control tags to your .aspx file. We can see the markup code in source button to switch to source view.

Understanding ASP.NET Pages

Dynamic Compilation

When you create an ASP.NET page, you are actually creating the source code for a .NET class. You are creating a new instance of the `System.Web.UI.Page` class. The entire contents of an ASP.NET page, including all script and HTML content, are compiled into a .NET class.

When you request an ASP.NET page, ASP.NET Framework checks for a .NET class that corresponds to the page. If a corresponding class does not exist, the Framework automatically compiles the page into a new class and stores the compiled class (the assembly) in the Temporary ASP.NET Files folder located at the following path:

```
\WINDOWS\Microsoft.NET\Framework\v4.0.30128\Temporary ASP.NET Files
```

The next time anyone requests the same page in the future, the page is not compiled again. The previously compiled class is executed, and the results are returned to the browser.

The compiled class is preserved in the Temporary ASP.NET Files folder until the source code for your application is modified.

When the class is added to the Temporary ASP.NET Files folder, a file dependency is created between the class and the original ASP.NET page. If the ASP.NET page is modified in any way, the corresponding .NET class is automatically deleted. The next time someone requests the page, the Framework automatically compiles the modified page source into a new .NET class.

This process is called dynamic compilation, which enables ASP.NET applications to support thousands of simultaneous users. Unlike an ASP Classic page, for example, an ASP.NET page does not need to be parsed and compiled every time it is requested. An ASP.NET page is compiled only when an application is modified.

Using Code-Behind Pages

The ASP.NET Framework (and Visual Web Developer) enables you to create two different types of ASP.NET pages. You can create both single-file and two-file ASP.NET pages.

In a single-file ASP.NET page, a single file contains both the page code and page controls. The page code is contained in a `<script runat="server">` tag.

As an alternative to a single-file ASP.NET page, you can create a two-file ASP.NET page. A two-file ASP.NET page is normally referred to as a code-behind page. In a code-behind page, the page code is contained in a separate file.

There are a few differences in the processing of code-behind and single-file pages.

Code Behind	Single File
The HTML and controls are in the *.aspx* file, and the code is in a separate *.aspx.vb* or *.aspx.cs* file.	The code is in <script> blocks in the same *.aspx* file that contains the HTML and controls.
The code for the page is compiled into a separate class from which the *.aspx* file derives.	The *.aspx* file derives from the Page class.
All project class files (without the *.aspx* file itself) are compiled into a *.dll* file, which is deployed to the server without any source code.	When the page is deployed, the source code is deployed along with the Web Forms page, because it is physically in the *.aspx* file.

When a request for the page is received, then an instance of the project *.dll* file is created and executed.	However, you do not see the code, only the results are rendered when the page runs.

ASP.NET - Life Cycle

ASP.Net life cycle specifies, how:

- ASP.Net processes pages to produce dynamic output
- The application and its pages are instantiated and processed
- ASP.Net compiles the pages dynamically

ASP.Net Page Life Cycle:

Following are the different stages of an ASP.Net page:

- **Page request :** when ASP.Net gets a page request, it decides whether to parse and compile the page or there would be a cached version of the page; accordingly the response is sent.
- **Starting of page life cycle :** at this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
- **Page initialization :** At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and themes are applied. For a new request postback data is loaded and the control properties are restored to the view-state values.
- **Page load :** A this stage, control properties are set using the view state and control state values.
- **Validation :** Validate method of the validation control is called and if it runs successfully, the IsValid property of the page is set to true.
- **Postback event handling :** If the request is a postback (old request), the related event handler is called.
- **Page rendering :** At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Page's Response property.

- **Unload :** The rendered page is sent to the client and page properties, such as Response and Request are unloaded and all cleanup done.

ASP.Net Page Life Cycle Events:

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes like Onclick or handle.

Following are the page life cycle events:

- **PreInit :** PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.
- **Init :** Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.
- **InitComplete :** InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
- **LoadViewState :** LoadViewState event allows loading view state information into the controls.
- **LoadPostData :** During this phase, the contents of all the input fields defined with the <form> tag are processed.
- **PreLoad :** PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.
- **Load :** The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.
- **LoadComplete :** The loading process is completed, control event handlers are run and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler.

- **PreRender :** The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
- **PreRenderComplete :** As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
- **SaveStateComplete :** State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.
- **UnLoad :** The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

Using the `Page.IsPostBack` Property

The `Page` class includes a property called the `IsPostBack` property, which you can use to detect whether the page has already been posted back to the server.

Because of View State, when you initialize a control property, you do not want to initialize the property every time a page loads. Because View State saves the state of control properties across page posts, you typically initialize a control property only once, when the page first loads.

Many controls don't work correctly if you reinitialize the properties of the control with each page load. In these cases, you must use the `IsPostBack` property to detect whether the page has been posted.

Advantages of ASP.NET

1. **Separation of Code from HTML**

   To make a clean sweep, with ASP.NET you have the ability to completely separate layout and business logic. This makes it much easier for teams of programmers and designers to collaborate efficiently. This makes it much easier for teams of programmers and designers to collaborate efficiently.

2. **Support for compiled languages**

   Developer can use VB.NET and access features such as strong typing and object-oriented programming. Using compiled languages also means that ASP.NET pages do not suffer the performance penalties associated with interpreted code. ASP.NET pages are precompiled to byte-code and Just In Time (JIT) compiled when first requested. Subsequent requests are directed to the fully compiled code, which is cached until the source changes.

3. **Use services provided by the .NET Framework**

   The .NET Framework provides class libraries that can be used by your application. Some of the key classes help you with input/output, access to operating system services, data access, or even debugging. We will go into more detail on some of them in this module.

4. **Graphical Development Environment**

   Visual Studio .NET provides a very rich development environment for Web developers. You can drag and drop controls and set properties the way you do in Visual Basic 6. And you have full IntelliSense support, not only for your code, but also for HTML and XML.

5. **State management**

   To refer to the problems mentioned before, ASP.NET provides solutions for session and application state management. State information can, for example, be kept in memory or stored in a database. It can be shared across Web farms, and state information can be recovered, even if the server fails or the connection breaks down.

6. **Update files while the server is running**

   Components of your application can be updated while the server is online and clients are connected. The Framework will use the new files as soon as they are copied to the application. Removed or old files that are still in use are kept in memory until the clients have finished.

7. **XML-Based Configuration Files**

   Configuration settings in ASP.NET are stored in XML files that you can easily read and edit. You can also easily copy these to another server, along with the other files that comprise your application.

IDE in ASP. Net

An Integrated Development Environment (IDE) is a software environment used to write other programs using tools like an editor and compiler. This can prove to be an extremely useful tool when coding uses various languages for many reasons. Examples of different IDE's include Eclipse, Visual Studios, and Net Beans.

Features of IDE

1. Code insight

This is one of the most helpful tools that an IDE can provide, which is the ability for the program to interpret what is typed out. The program can change text colour to represent different classes, functions, and variables. Microsoft Visual Studios offers something called IntelliSense that can also predict what you are typing out and finishes your words for you. Other IDE's offer a similar tool called something proprietary to their program.

2. Ability to debug your program

This is arguably one of the most important tools to deploying a successful program. Testing is crucial to make sure that your user does not experience an incident where the code does not handle a specific error and crash the program. Debugging provides you with the ability to run through the program, stopping the code at specified points to check values of variables or other interests as needed, to verify that the code and functions are running as intended.

3. Visual representation of the location of these files

Finally a few more tools that IDE's offer can be resource management and the ability to compile your code. When writing a new program there are usually many different files that have been referenced in specific path locations so it is very important for the running of the program that these files are in the correct locations. Using an IDE makes it easy to see a visual representation of the location of these files and makes it more understandable for the user.

Advantages to IDEs:

- Increased Efficiency – faster coding with less effort

- Collaboration – A group of programmers can easily work together within an IDE
- Project Management – Program resources are easily

Disadvantages to IDEs:

- May be too complex for beginning programmers
- Each IDE will have a unique learning curve requiring time to learn
- Cannot automatically fix errors, still need knowledge to code efficiently

Introduction to Web Server

- Web Server is a Computer or Combination of computers, which is connected through internet or intranet to serve the clients quests, coming from their web browser.
- It is a large repository of web pages which transfer to the client in response to their request.
- Every web server has a unique IP address and domain name which identifies that machine on the network.
- The client request to the server through protocol such as FTP, HTTP, SMTP etc for their own specific use.

When you run a web application, Visual Studio starts its integrated web server. ASP.NET compiles the code for your web application, runs your web page, and then returns the final HTML to the browser. The test server only runs while Visual Studio is running, and it only accepts requests from your computer. When you run a web page, the URL in the browser includes a port number.
For example, if you run a web application in a folder named OnlineBank, you might see a URL like http://localhost:4235/OnlineBank/Default.aspx. This URL indicates that the web server is running on your computer (localhost), so its requests aren't being sent over the Internet. It also indicates that all requests are being transmitted to port number 4235. That way, the requests won't conflict with any other applications that might be running on your computer and listening for requests. Every time Visual Studio starts the integrated web server, it randomly chooses an available port.

There are many types of web server, Enterprise uses according to their need. Some of the popular categories of web servers are -

- **HTTP Server** - It handles HTTP request coming from clients' browser and transfer the static pages to client in response to their request. This pages runs of the client browser. It generally contains the static pages.
- **FTP Server** - This type of server used for file transfer from one machine (Computer) to another using the internet or intranet. It uses File Transfer Protocols to transfer file from one computer to another. Such type of server uses some file transfer policies, authentication, login validation etc
- **Mail Server** - A Mail Server store and retrieve mail messages from client mail box.
- **Application Server** - It is installed database and web servers

Apache Tomcat is popular web server being used today for the implementation of some java technologies. It is open source software used for implementing web applications.

IIS (Internet Information Server)

IIS (Internet Information Server) is one of the most powerful web servers from Microsoft that is used to host your ASP.NET Web application. IIS has its own ASP.NET Process Engine to handle the ASP.NET request. So, when a request comes from client to server, IIS takes that request and process it and send response back to clients.

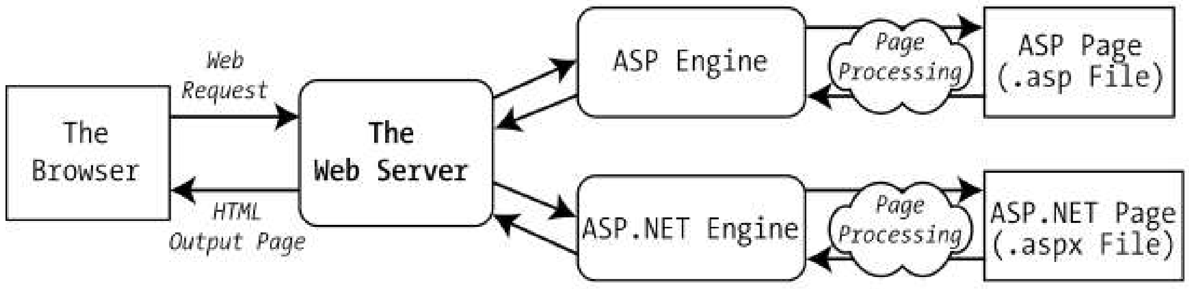


Figure . *How IIS handles an ASP file request*

Introduction of an ASP.NET Application

An ASP.NET application is a combination of files, pages, handlers, modules, and executable code that can be invoked from a virtual directory on a web server. Every ASP.NET application is executed inside a separate *application domain*.

ASP.NET File Types

ASP.NET applications can include many types of files.

File Name	**Description**
Ends with .aspx	: These are ASP.NET web pages. They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start your web application.
Ends with .ascx	: These are ASP.NET user controls. User controls are similar to web pages, except that the user can't access these files directly. Instead, they must be hosted inside an ASP.NET web page. User controls allow you to develop a small piece of user interface and reuse it in as many web forms as you want without repetitive code.
Ends with .asmx	: These are ASP.NET web services—collections of methods that can be called over the Internet. Web services work differently than web pages, but they still share the same application resources, configuration settings, and memory.
web.config	: This is the XML-based configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more.
Global.asax	: This is the global application file. You can use this file to define global variables
Ends with .vb	: These are code-behind files that contain VB code. They allow you to separate the application logic from the user interface of a web page.

ASP.NET Application Directories

Every web application should have a well-planned directory structure. Along with the directories you create, ASP.NET also uses a few specialized subdirectories, which it recognizes by name.

Directory	**Description**
Bin	Contains all the compiled .NET components (DLLs) that the ASP.NET web application uses. For example, if you develop a custom component for accessing a database you'll place the component here. ASP.NET will automatically detect the assembly, and any page in the web application will be able to use it.
App_Code	Contains source code files that are dynamically compiled for use in your application. You can use this directory in a similar way as the Bin directory; the only difference is that you place source code files here instead of compiled assemblies.
App_GlobalResources	Stores global resources that are accessible to every page in the web application. This directory is used in localization scenarios, when you need to have a website in more than one language.
App_LocalResources	Serves the same purpose as App_GlobalResources, except these resources are accessible to a specific page only.
App_WebReferences	Stores references to web services that the web application uses.
App_Data	Stores data, including SQL Server 2005 Express Edition database files and XML files.
App_Themes	Stores the themes that are used by your web application.

The Page Class

Every web page is a custom class that inherits from System.Web.UI.Page. By inheriting from this class, your web page class acquires a number of properties and methods that your code can use.

Property	**Description**
IsPostBack	This Boolean property indicates whether this is the first time the page is being run (False) or whether the page is being resubmitted in response to a

	control event, typically with stored view state information(True). You'll usually check this property in the Page.Load event handler to ensure that your initial web page initialization is only performed once.
EnableViewState	When set to False, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
Application	This collection holds information that's shared between all users in your website. For example, you can use the Application collection to count the number of times a page has been visited.
Session	This collection holds information for a single user, so it can be used in different pages. For example, you can use the Session collection to store the items in the current user's shopping basket on an e-commerce website.
Cache	This collection allows you to store objects that are time-consuming to create so they can be reused in other pages or for other clients. This technique, when implemented properly, can improve performance of your web pages.
Request	This refers to an HttpRequest object that contains information about the current web request.
Response	This refers to an HttpResponse object that represents the response ASP.NET will send to the user's browser.
Server	This refers to an HttpServerUtility object that allows you to perform a few miscellaneous tasks. For example, it allows you to encode text so that it's safe to place it in a URL or in the HTML markup of your page.
User	If the user has been authenticated, this property will be initialized with user information.

The web.config File

- **Web.config** is the main settings and configuration file for an ASP.NET web application.
- The file is an XML document that defines configuration information regarding the web application.
- This file stores the information about how the web application will act.

- The web.config file contains information that controls module loading, security configuration, session state configuration, and application language and compilation settings.
- Web.config files can also contain application specific items such as database connection strings.
- The entire content of the file is nested in a root <configuration> element. Inside this element are several more subsections, some of which you'll never change, and others which are more important.
- The web.config file is case-sensitive.
- You can have any number of *Web.config* files for an application. Each *Web.config* applies settings to its own directory and all the child directories below it.
- All the *Web.config* files inherit the root *Web.config* file available at the following location *systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\Web.config* location.
- The changes in *Web.config* don't require the reboot of the web server.

Here's the basic skeletal structure of the web.config file, with the three most important sections highlighted in bold:

```
<?xml version="1.0" ?>
<configuration>
<configSections>...</configSections>
<appSettings>...</appSettings>
<connectionStrings>...</connectionStrings>
<system.web>...</system.web>
<system.codedom>...</system.codedom>
<system.webServer>...</system.webServer>
</configuration>
```

The three sections in the web.config file are:

1. The **<appSettings>** section allows you to add your own miscellaneous pieces of information.
2. The **<connectionStrings**> section allows you to define the connection information for accessing a database. And

3. The **<system.web>** section holds every ASP.NET setting you'll need to configure. Inside the <system.web> element are separate elements for each aspect of website configuration.

Storing Custom Settings in the web.config File

The custom settings that you add are written as simple string variables. You might want to use a special web.config setting for several reasons:

***To centralize an important setting that needs to be used in many different pages*:** For example, you could create a variable that stores a database query. Any page that needs to use this query can then retrieve this value and use it.

***To make it easy to quickly switch between different modes of operation*:** For example, you might create a special debugging variable. Your web pages could check for this variable and, if it's set to a specified value, output additional information to help you test the application.

***To set some initial values*:** Depending on the operation, the user might be able to modify these values, but the web.config file could supply the defaults.

Global.asax File

Global.asax is an optional file which is used to handling higher level application events such as Application_Start, Application_End, Session_Start, Session_End etc. It is also popularly known as ASP.NET Application File. This file resides in the root directory of an ASP.NET-based application.

Global.asax contains a Class representing your application as a whole. At run time, this file is parsed and compiled into a dynamically generated .NET Framework class derived from the HttpApplication base class. You can deploy this file as an assembly in the \bin directory of an ASP.NET application. The Global.asax file itself is configured so that if a user requests the file, the request is rejected. External users cannot download or view the code written within it.

The Global.asax file allows you to write code that responds to global application events. These events fire at various points during the lifetime of a web application, including when the application domain is first created (when the first request is received for a page in your website folder).

To add a Global.asax file to an application in Visual Studio, choose Website ➢ Add New Item, and select the Global Application Class file type. Then, click OK.

The Global.asax file looks like:

```
<%@ Application Language="C#" %>
<script runat="server">
  void Application_Start(object sender, EventArgs e)
  {
    // Code that runs on application startup
  }
  void Application_End(object sender, EventArgs e)
  {
    //  Code that runs on application shutdown
  }
  void Application_Error(object sender, EventArgs e)
  {
    // Code that runs when an unhandled error occurs
  }
  void Session_Start(object sender, EventArgs e)
  {
    // Code that runs when a new session is started
  }
  void Session_End(object sender, EventArgs e)
  {
    // Code that runs when a session ends.
    // Note: The Session_End event is raised only when the sessionstate mode
    // is set to InProc in the Web.config file. If session mode is set to StateServer
    // or SQLServer, the event is not raised.
  }
</script>
```

After that you need to add a class in your project.

```
Right clicking App_Code
          Add New Item
                    Class
                              name it Global.cs
                                        Add
```

Inherit the newly generated by System.Web.HttpApplication and copy all the method created Global.asax to Global.cs and also add an inherit attribute to the Global.asax file.

Now your Global.asax will look like following:

```
public class Global : System.Web.HttpApplication
{
    public Global()
    {
        //
        // TODO: Add constructor logic here
        //
    }

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup

    }
    /// Many other events like begin request...e.t.c, e.t.c
}
```

Each ASP.NET application can have one Global.asax file. Once you place it in the appropriate website directory, ASP.NET recognizes it and uses it automatically. For example, if you add the Global.asax file shown previously to a web application, every web page in that application will include footer.

Using Visual C# in ASP. Net

What is C#?

- C# is pronounced "C-Sharp".
- It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.
- C# has roots from the C family, and the language is close to other popular languages like C++ and Java.

- The first version was released in year 2002. The latest version, **C# 8**, was released in September 2019.
- C# is used for:

  Mobile applications, Desktop applications, Web applications, Web services, Web sites, Games, VR, Database applications etc.

Features of C#

- It is one of the most popular programming language in the world
- It is easy to learn and simple to use
- It has a huge community support
- C# is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- As C# is close to C, C++ and Java, it makes it easy for programmers to switch to C# or vice versa

Variables

- The general rules for constructing names for variables (unique identifiers) are:
- Names can contain letters, digits and the underscore character (_)
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like C# keywords, such as int or double) cannot be used as names

Declaring (Creating) Variables

- Syntax
  - type variableName = value;
    - Where *type* is a C# type (such as int or string), and *variableName* is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.
- Example
  - string msg = “Hello";
  - int myNum = 15;

Display Variables

- The WriteLine() method is often used to display variable values to the console window.
- To combine both text and a variable, use the + character:
- Example 1:

```
string name = "John";
Console.WriteLine("Hello " + name);
```

- Example 2:

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName; Console.WriteLine(fullName);
```

- Example 3:

```
int x = 5; int y = 6;
Console.WriteLine(x + y); // Print the value of x + y
```

- Example 4:

```
int x = 5, y = 6, z = 50;
Console.WriteLine(x + y + z);
```

Constants

To declare constant variable **const** keyword is used.

Example: const int myNum = 15;

- You cannot declare a constant variable without assigning the value. If you do, an error will occur: A const field requires a value to be provided. As well as you cannot change value of constant variable that is assigned at declaration time.

  Example:

```
const int myNum = 15;
myNum = 20; // error
```

Constants are immutable values which are known at compile time and do not change for the life of the program. Constants are declared with the const modifier. Only the C# built-in types (excluding System.Object) may be declared as const. User-defined types, including classes, structs, and arrays, cannot be const. Use the read-only modifier to create a class, struct, or array that is initialized one time at runtime (for example in a constructor) and thereafter cannot be changed.

C# does not support const methods, properties, or events.

The enum type enables you to define named constants for integral built-in types (for example int, uint, long, and so on). Constants must be initialized as they are declared.

For example:

```
class Calendar1
{
   public const int Months = 12;
}
```

In this example, the constant months are always 12, and it cannot be changed even by the class itself. In fact, when the compiler encounters a constant identifier in C# source code (for example, months), it substitutes the literal value directly into the intermediate language (IL) code that it produces. Because there is no variable address associated with a constant at run time, const fields cannot be passed by reference and cannot appear as an l-value in an expression.

Multiple constants of the same type can be declared at the same time, for example:

```
class Calendar2
{
   public const int Months = 12, Weeks = 52, Days = 365;
}
```

The expression that is used to initialize a constant can refer to another constant if it does not create a circular reference. For example:

```
class Calendar3
{
   public const int Months = 12;
   public const int Weeks = 52;
   public const int Days = 365;

     public const double DaysPerWeek = (double) Days / (double) Weeks;
     public const double DaysPerMonth = (double) Days / (double) Months;
}
```

Constants can be marked as public, private, protected, internal, protected internal or private protected. These access modifiers define how users of the class can access the constant.
Constants are accessed as if they were static fields because the value of the constant is the same for all instances of the type. You do not use the static keyword to declare them. Expressions that are not in the class that defines the constant must use the class name, a period, and the name of the constant to access the constant.
For example: int mnthVal = Calendar.Months;

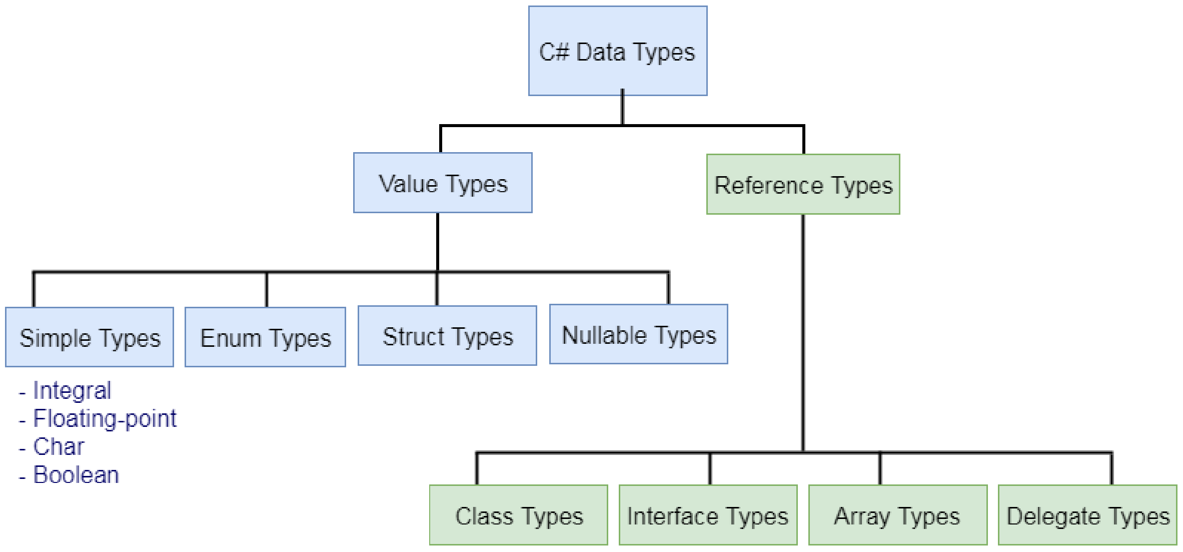
Access Modifiers

All types and type members have an accessibility level. The accessibility level controls whether they can be used from other code in your assembly or other assemblies. Use the following access modifiers to specify the accessibility of a type or member when you declare it:

- **Public:** The type or member can be accessed by any other code in the same assembly or another assembly that references it.
- **Private:** The type or member can be accessed only by code in the same class or struct.
- **Protected:** The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
- **Internal:** The type or member can be accessed by any code in the same assembly, but not from another assembly.
- **Protected internal:** The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
- **Private protected:** The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

Data Types

C# mainly categorized data types in two types: Value types and Reference types. Value types include simple types (such as int, float, bool, and char), enum types, struct types, and Nullable value types. Reference types include class types, interface types, delegate types, and array types.



Predefined Data Types in C#

C# includes some predefined value types and reference types. The following table lists predefined data types:

Type	Description	Range
byte	8-bit unsigned integer	0 to 255
sbyte	8-bit signed integer	-128 to 127
short	16-bit signed integer	-32,768 to 32,767
ushort	16-bit unsigned integer	0 to 65,535
int	32-bit signed integer	-2,147,483,648 to 2,147,483,647
uint	32-bit unsigned integer	0 to 4,294,967,295
long	64-bit signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	64-bit unsigned integer	0 to 18,446,744,073,709,551,615
float	32-bit Single-precision floating point type	-3.402823e38 to 3.402823e38

Type	Description	Range
double	64-bit double-precision floating point type	-1.79769313486232e308 to 1.79769313486232e308
decimal	128-bit decimal type for financial and monetary calculations	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
char	16-bit single Unicode character	Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode)
bool	8-bit logical true/false value	True or False
object	Base type of all other types.	
string	A sequence of Unicode characters	
DateTime	Represents date and time	0:00:00am 1/1/01 to 11:59:59pm 12/31/9999

The predefined data types are alias to their .NET type (CLR class) name. The following table lists alias for predefined data types and related .NET class name.

Alias	.NET Type	Type
byte	System.Byte	struct
sbyte	System.SByte	struct
int	System.Int32	struct
uint	System.UInt32	struct
short	System.Int16	struct
ushort	System.UInt16	struct
long	System.Int64	struct
ulong	System.UInt64	struct
float	System.Single	struct
double	System.Double	struct

Alias	.NET Type	Type
char	System.Char	struct
bool	System.Boolean	struct
object	System.Object	Class
string	System.String	Class
decimal	System.Decimal	struct
DateTime	System.DateTime	struct

It means that whether you define a variable of int or Int32, both are the same.

Example:

int i = 345;

Int32 i = 345; // same as above

Operators

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.

For Example:

```
int x = 5;

int y = 10;

int z = x + y;// z = 15
```

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x – y
*	Multiplication	Multiplies two values	x * y

/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

Assignment Operator

Basic assignment operator (=) is used to assign values to variables. For example,

```
double x;

x = 50.05;
```

Here 50.05 is assigned to variable x.

Relational Operators

Relational operators are used to check the relationship between two operands. If the relationship is true the result will be true, otherwise it will result in false.

Relational operators are used in decision making and loops.

C# Relational Operators

Operator	Operator Name	Example
= =	Equal to	6 = = 4 evaluates to false
>	Greater than	3 > -1 evaluates to true
<	Less than	5 < 3 evaluates to false
>=	Greater than or equal to	4 >= 4 evaluates to true

C# Relational Operators

Operator	Operator Name	Example
<=	Less than or equal to	5 <= 3 evaluates to false
!=	Not equal to	10 != 2 evaluates to true

Logical Operators

Logical operators are used to perform logical operation such as and, or. Logical operators operate on Boolean expressions (true and false) and returns Boolean values. Logical operators are used in decision making and loops.

Operator	**Name**	**Description**	**Example**
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Unary Operators

Unlike other operators, the unary operators operate on a single operand.

C# unary operators

Operator	Operator Name	Description
+	Unary Plus	Leaves the sign of operand as it is

-	Unary Minus	Inverts the sign of operand
++	Increment	Increment value by 1
--	Decrement	Decrement value by 1
!	Logical Negation (Not)	Inverts the value of a boolean

Ternary Operator

The ternary operator ? : operates on three operands. It is shorthand for if-then-else statement. Ternary operator can be used as follows:

variable = Condition? Expression1 : Expression2;

The ternary operator works as follows:

- If the expression stated by Condition is true, the result of Expression1 is assigned to variable.
- If it is false, the result of Expression2 is assigned to variable.

Bitwise and Bit Shift operators

Bitwise and bit shift operators are used to perform bit manipulation operations.

C# Bitwise and Bit Shift operators

Operator	Operator Name
~	Bitwise Complement
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR

C# Bitwise and Bit Shift operators	
Operator	Operator Name
<<	Bitwise Left Shift
>>	Bitwise Right Shift

Compound Assignment Operators

C# Compound Assignment Operators

Operator	Operator Name	Example	Equivalent To
+=	Addition Assignment	`x += 5`	`x = x + 5`
-=	Subtraction Assignment	`x -= 5`	`x = x - 5`
*=	Multiplication Assignment	`x *= 5`	`x = x * 5`
/=	Division Assignment	`x /= 5`	`x = x / 5`
%=	Modulo Assignment	`x %= 5`	`x = x % 5`
&=	Bitwise AND Assignment	`x &= 5`	`x = x & 5`
|=	Bitwise OR Assignment	`x |= 5`	`x = x | 5`
^=	Bitwise XOR Assignment	`x ^= 5`	`x = x ^ 5`
<<=	Left Shift Assignment	`x <<= 5`	`x = x << 5`
>>=	Right Shift Assignment	`x >>= 5`	`x = x >> 5`
=>	Lambda Operator	`x => x*x`	`Returns x*x`