By-Ramesh Gangasagare

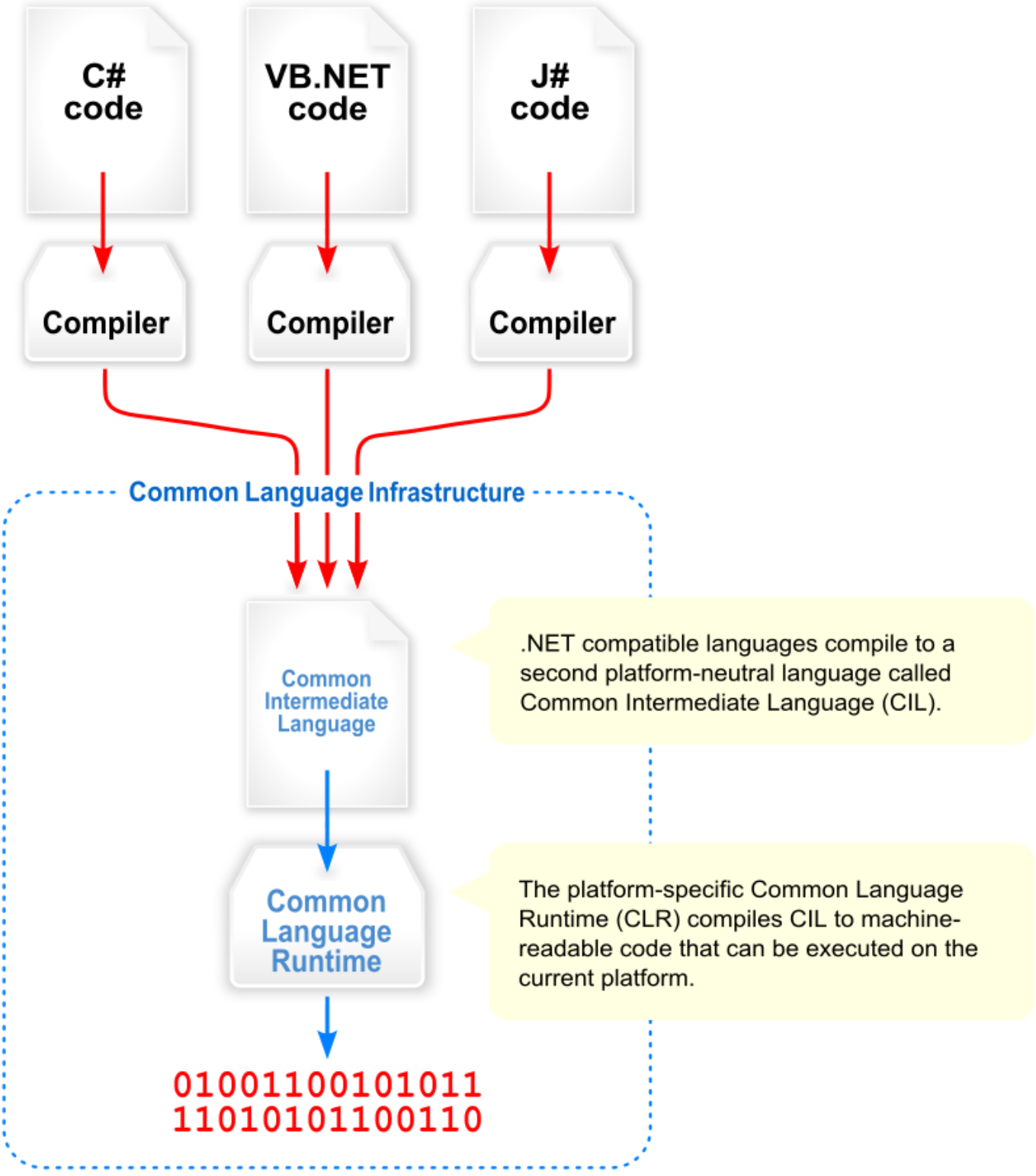
.NET Framework is a managed execution environment for Windows that provides a variety of services to its running apps. It consists of two major components: the common language runtime(CLR), which is the execution engine that handles running apps, and the .NET Framework Class Library, which provides a library of tested, reusable code that developers can call from their own apps. The services that .NET Framework provides to running apps include the following:

Memory management. In many programming languages, programmers are responsible for allocating and releasing memory and for handling object lifetimes. In .NET Framework apps, the CLR provides these services on behalf of the app.

A common type system. In traditional programming languages, basic types are defined by the compiler, which complicates cross-language interoperability. In .NET Framework, basic types are defined by the .NET Framework type system and are common to all languages that target .NET Framework.

An extensive class library. Instead of having to write vast amounts of code to handle common low-level programming operations, programmers use a readily accessible library of types and their members from the .NET Framework Class Library.

Development frameworks and technologies. .NET Framework includes libraries for specific areas of app development, such as ASP.NET for web apps, ADO.NET for data access, Windows Communication Foundation for service-oriented apps, and Windows Presentation Foundation for Windows desktop apps.



Architecture of .NET Framework

The two major components of .NET Framework are the Common Language Runtime and the .NET Framework Class Library.

The Common Language Runtime (CLR) is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.
The Class Library provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.
.NET applications are written in the C#, F#, or Visual Basic programming language. Code is compiled into a language-agnostic Common Intermediate Language (CIL). Compiled code is stored in assemblies—files with a .dll or .exe file extension.

When an app runs, the CLR takes the assembly and uses a just-in-time compiler (JIT) to turn it into machine code that can execute on the specific architecture of the computer it is running on.

MSIL is in .NET
MSIL stands for Microsoft Intermediate Language
During the compile time, the source code is converted into Microsoft Intermediate Language (MSIL) by compiler
MSIL is a CPU-independent set of instructions that can be efficiently converted to the native code

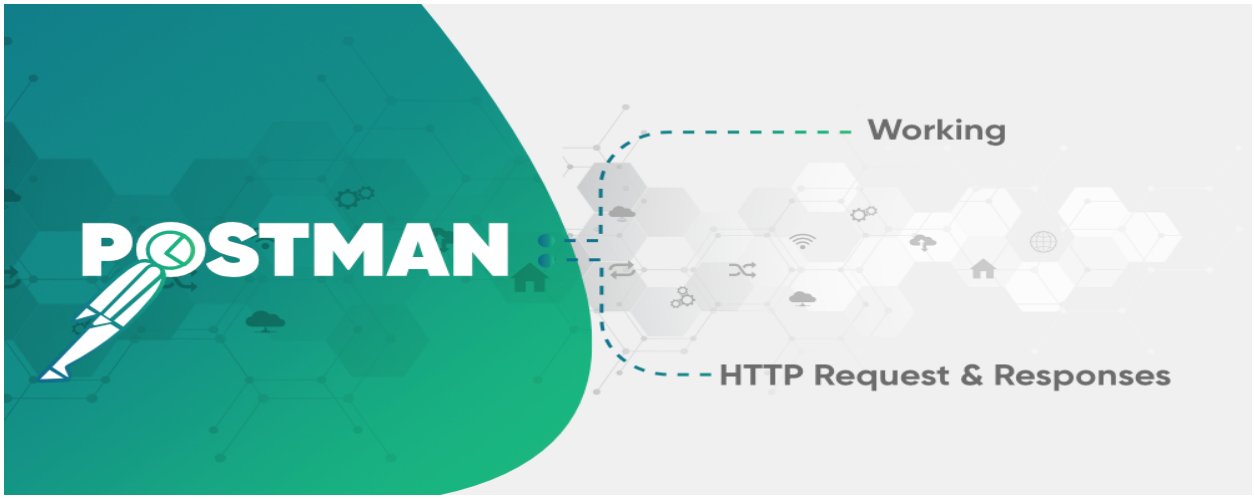
- ☐ **Datatype int in .NET is 32 bits.**
- ☐ **Functions .NET Assembly performs?**
  Assembly is the main unit of deployment in a .NET Framework application executed as .exe or .dll.

Mention the type of code security available in .NET?
The type of code security available in .NET are

Role based security: This authorizes the user.
Code access security: This protects system resources from unauthorized calls.

➢ **Postman:**

- Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.
- Postman is an API Platform for developers to design, build, test and iterate their APIs.



Why Use a Postman?

Postman is an API testing and development tool that is designed to send requests from the client side to the web server and receive a response from the backend. The information received with the response is determined by the data sent along with the request. Thus, Postman is used as an API client to check client-server requests to make sure everything works on the server side as it is expected. Postman supports requests to Restful, SOAP, and GraphQL web services.

A graphical interface makes Postman an easy-to-use tool in the API testing and development process.

To test Restful web services Postman uses HTTP requests to send information to an API. An HTTP request is an HTTP message that a client sends to an HTTP server. Generally, an HTTP request contains a start line, a set of HTTP headers, and a body.

A start line of HTTP request contains HTTP method, URI of the target resource, and the version of the HTTP protocol and has the following structure:

HTTP Method/ Target URI/ HTTP Version

HTTP methods determine the action that must be performed on the resource. The meaning of HTTP methods is described by the protocol specification. The HTTP protocol specification does not limit the number of different methods that can be used. However, only some of the most standard methods are used to support compatibility with a wide range of applications.

Some of the HTTP methods that can be used in API calls are:

GET – to get (read) data (e.g., a user list).

POST – to create new data.

PUT / PATCH – to update data.

DELETE – to delete data.

OPTIONS – to get a full description of API methods available on the service.

The header contains metadata to allow a client to pass clarifying and service information about the HTTP request such as encoding information, authorization parameters, etc.

Information you want to transfer over the network is passed in the body. The body is optional and can be left empty (depending on the HTTP methods and headers).

HTTP-response is the data that comes back from the API server. Besides the data in the body, the response header contains a status HTTP code of the server response. For example, you can receive the following status codes in the response header:

200 – Success;

400 – Bad request;

401 – Unauthorized.

Working With Requests in Postman

Using the Postman graphical interface there is no need for web developers to write their own code to test API features.

Working with requests in Postman includes the following sequence of steps:

Adding a new HTTP request using the Postman interface.

Customizing the request (specifying an HTTP method, header, body, authentication parameters).

Executing the request.

Saving the request (e.g., to a folder or collection).

➢ **Tests in Postman**

To process the response from the server one can create different tests in Postman. The test in Postman is a JavaScript code that is executed automatically after the client receives a response to the request. In other words, Postman tests are applied to the result of the executed request.

Postman offers many ready-to-use code snippets that you can add to your test. Here you can find snippets to validate the codes and content of responses, parse and save values to environment variables or global variables, check their compliance with specified values, etc. For example, you can verify that the status code of the response to the GET request is 200. Tests can be applied not only to a single request but can be moved to a collection level.

Postman Collections

To execute multiple requests one by one automatically a collection of requests is used in Postman. You can run Collections filled in with requests and tests with Collection Runner and use them as automated API tests. To run a collection, you can select the environment, the number of iterations in the run and a delay between requests. Moreover, Postman support logging of requests and storing variables and cookies.

Once a collection of requests has been created, it can be exported to a file to be used in a third- party application. For example, Dotcom-Monitor supports import of Postman Collection to use in monitoring and load testing setup.

Note that in case you need to call an API that requires authentication, the collection of requests to this API must include a POST request to the corresponding authentication service to authorize the client on the server.

➢ **API Application Programming Interface:**

An API stands for Application Programming Interface – an interface that allows applications to communicate. The API makes it possible for software developers to send information directly from one app to another, bypassing a user interface.

For API testing and development, special tools that allow you to send input data in a request and check if the output data is correct are widely used. Postman is one of the tools like this.

- How do APIs work?
  API architecture is usually explained in terms of client and server. The application sending the request is called the client, and the application sending the response is called the server. So in the weather example, the bureau's weather database is the server, and the mobile app is the client.

  In simple terms when we send the request where certain methods like Get, Post, Put, Delete etc.then we get a response from the backend server.

  Note- All the webservices are the API, webservices are the part of the API some web services -REST, RPC's , SOAP

☐ **Web API**
A Web API or Web Service API is an application processing interface between a web server and web browser. All web services are APIs but not all APIs are web services. REST API is a special type of Web API that uses the standard architectural style explained above.using System;

```
namespace MyApplication
{
  // Abstract class
  abstract class Animal
  {
    // Abstract method (does not have a body)
```

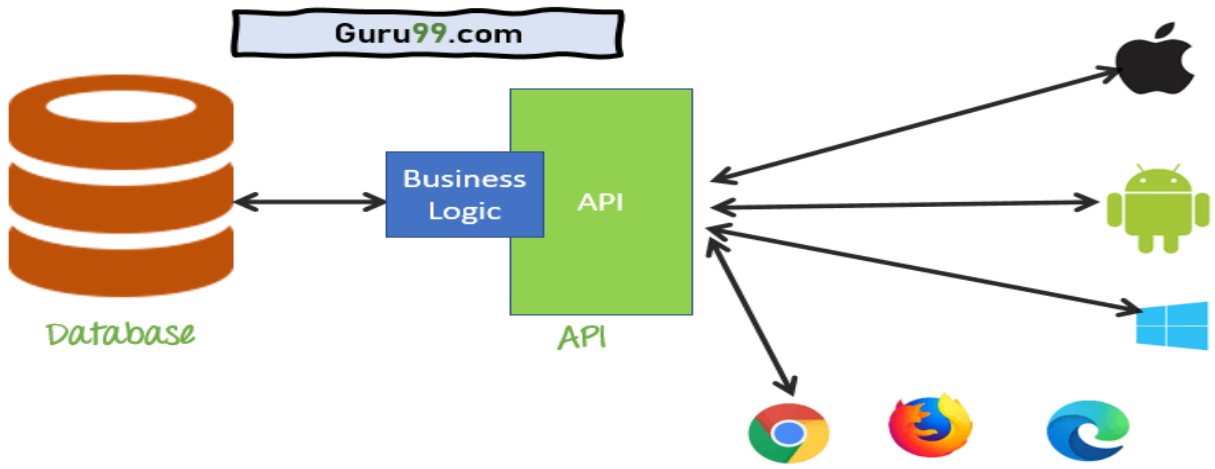
```
  public abstract void animalSound();

  // Regular method
  public void sleep()
  {
    Console.WriteLine("Zzz");
  }
}

// Derived class (inherit from Animal)
class Pig : Animal
{
  public override void animalSound()
  {
    // The body of animalSound() is provided here
    Console.WriteLine("The pig says: wee wee");
  }
}

class Program
{
  static void Main(string[] args)
  {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
 }
}
```

Client-side API

A client-side web API is a programmatic interface to extend functionality within a web browser or other HTTP client. Originally these were most commonly in the form of native plug-in browser extensions however most newer ones target standardized JavaScript bindings.

Server Side API

A server-side web API consists of one or more publicly exposed endpoints to a defined request–response message system, typically expressed in JSON or XML. The web API is exposed most commonly by means of an HTTP-based web server.



Testing APIs with Postman Explained

➢ **ADO.Net**

ADO.NET is a data access technology from the Microsoft .NET Framework that provides communication between relational and non-relational systems through a common set of components.ADO.NET is a set of computer software components that programmers can use to access data and data services from a database. It is a part of the base class library that is included with the Microsoft .NET Framework. It is commonly used by programmers to access and modify data stored in relational database systems, though it can also access data in non-relational data sources. ADO.NET is sometimes considered an evolution of ActiveX Data Objects

(ADO) technology, but was changed so extensively that it can be considered an entirely new product.

ADO.Net is a module of .NET Framework which is used to establish connections between application and data sources.

ADO.NET is conceptually divided into consumers and data providers.

➢ Entity Framework

Entity Framework is an object-relational mapping (ORM) framework for .NET applications. It simplifies database access and management by allowing developers to work with database objects as if they were regular .NET objects. Here are some key points about Entity Framework:

Object-Relational Mapping (ORM): Entity Framework allows developers to work with database entities using familiar object-oriented programming concepts. It maps database tables to .NET classes and database records to objects, making it easier to interact with databases in a more natural way.

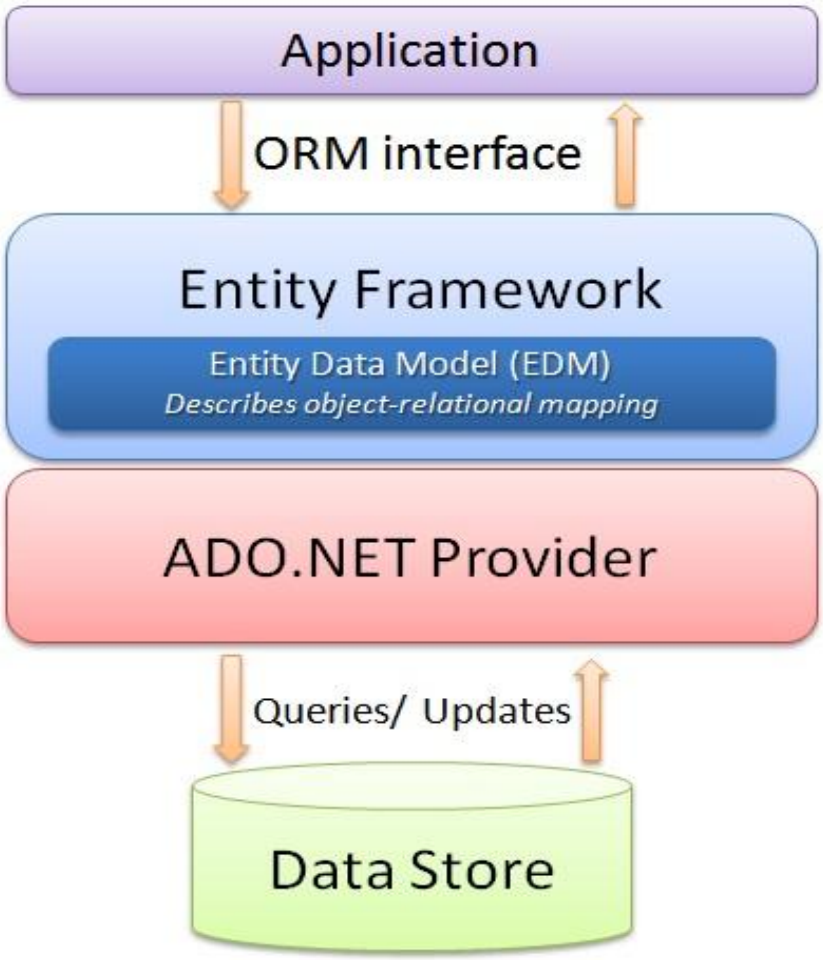
Database Provider Independence: Entity Framework supports multiple database providers, which means you can develop applications using Entity Framework and switch between different database systems (e.g., SQL Server, MySQL, PostgreSQL) without changing your application code significantly.

Entity Data Model (EDM): EDM is a conceptual model that represents the structure of your data. It defines the entities (tables), their relationships, and their attributes. EDM is used by Entity Framework to generate the necessary database schema and SQL queries.

LINQ Integration: Entity Framework seamlessly integrates with Language Integrated Query (LINQ), allowing you to write queries against your data using C# or VB.NET syntax. This makes it easier to query and manipulate data in a strongly typed manner

ADO.NET is an old and powerful technique for accessing databases in .NET applications (desktop and web). In ADO.NET, a lot of code is used.

In simple words, you can say that all the work is manual. You have to create everything you want for your application. For automating the database technique, Microsoft has introduced Entity Framework.



ORM(Object relational mapping):ORM simplifies the process of accessing data from applications.

Entity Framework is a framework that enables developers to manipulate relational data as domain-specific objects. In the Entity framework, developers use queries with LINQ and manipulate the database.

There are two techniques for managing the data with Entity Framework.

1) Model first

In model first, Model class of the project is created first with ORM designer and then, classes are generated. It is useful when your database is already created. When you use the model first approach, you have no need to create a class as it will be created automatically.

2) Code first

In code first, you have to create a Model class in a .cs file. You also have to create a context class for your project that is inherited by the DbContext class. This class will automatically generate your database. This approach is useful when you have not already created a database, so the database is created according to Model class.

ASP.Net

ASP stands for **A**ctive **S**erver **P**ages

ASP is a development framework for building web pages.

ASP.NET Web Pages

ASP.NET Web Pages is an SPA application model (Single Page Application).

The SPA model is quite similar to PHP and Classic ASP.

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.

➢ **MVC**

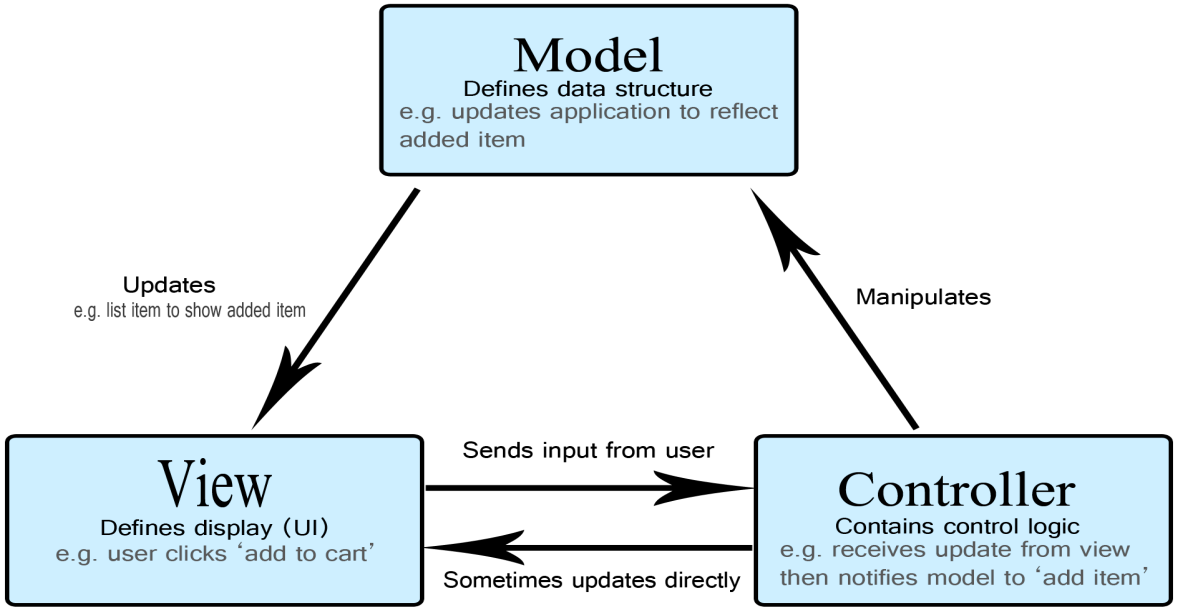
MVC, short for Model, View, and Controller, is a methodology or architectural pattern used for efficiently relating the user interface to underlying data models and organizing to relate the application code. MVC is primarily used to separate an application into three main components: Model, View, and Controller.

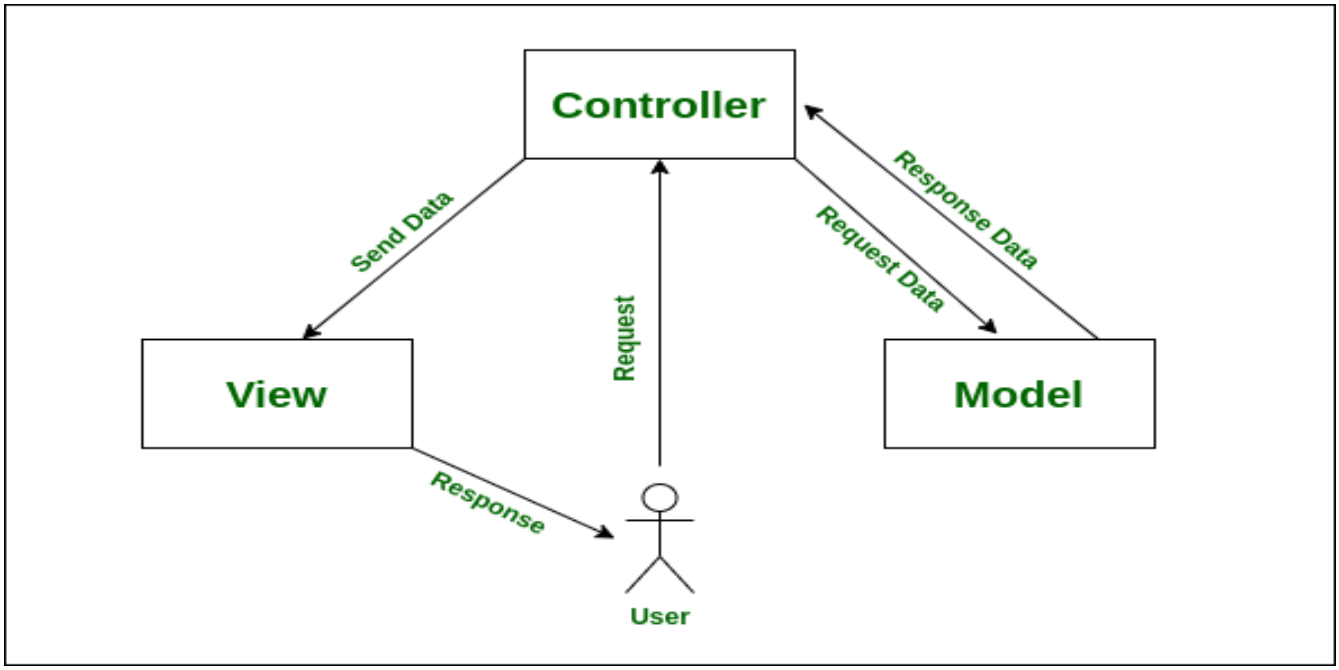
- The three parts of the MVC software-design pattern can be described as follows:

Model: Manages data and business logic.

View: Handles layout and display.

Controller: Routes commands to the model and view parts.

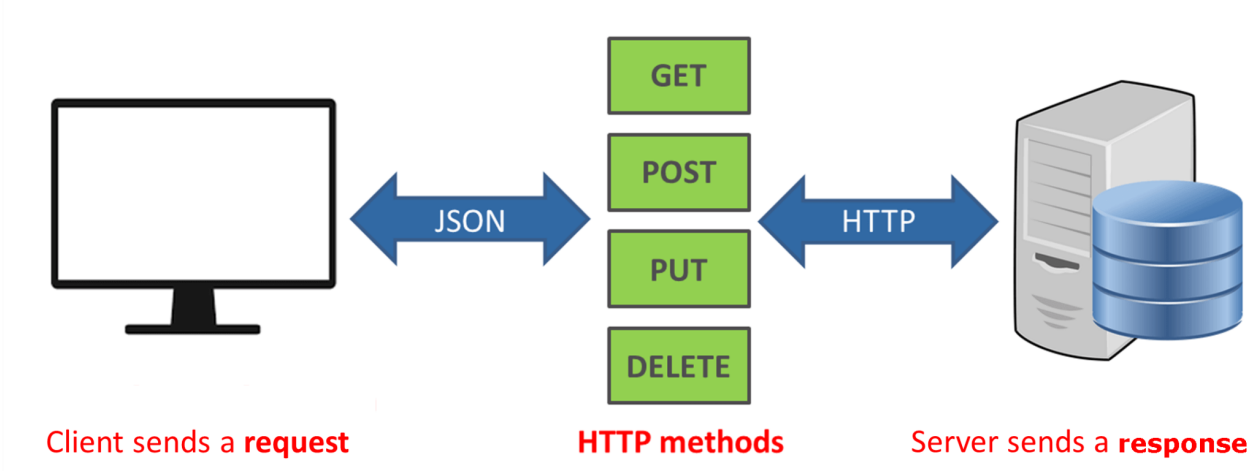




➢ **REST API**

REST API is a way for software applications to communicate over the internet using standard HTTP methods and URLs to perform actions on resources. It's a widely-used architecture for building web services.

A REST API (Representational State Transfer Application Programming Interface) is a set of rules and conventions for building and interacting with web services. It is an architectural style for designing networked applications and is widely used for developing web services and APIs.



Here are some key concepts and principles of a REST API:

Resources: In REST, everything is treated as a resource, which can be an object, data, or service that can be identified by a unique URI (Uniform Resource Identifier). Resources are the fundamental entities that the API deals with, and they can represent data, objects, or services.

HTTP Methods: REST APIs use standard HTTP methods to perform operations on resources. The most commonly used HTTP methods in REST are:

GET: Retrieve data from the server.

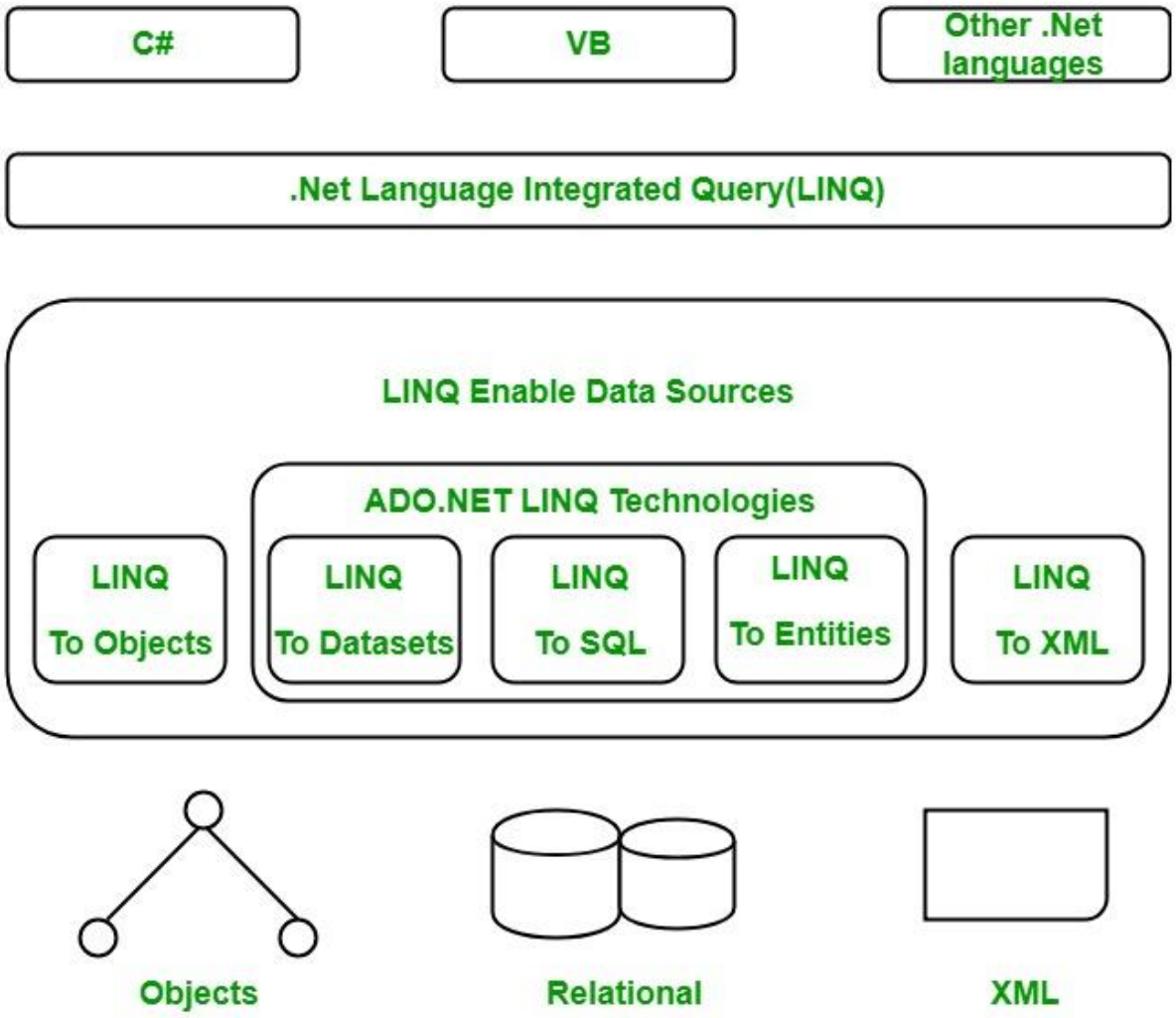
POST: Create a new resource on the server.

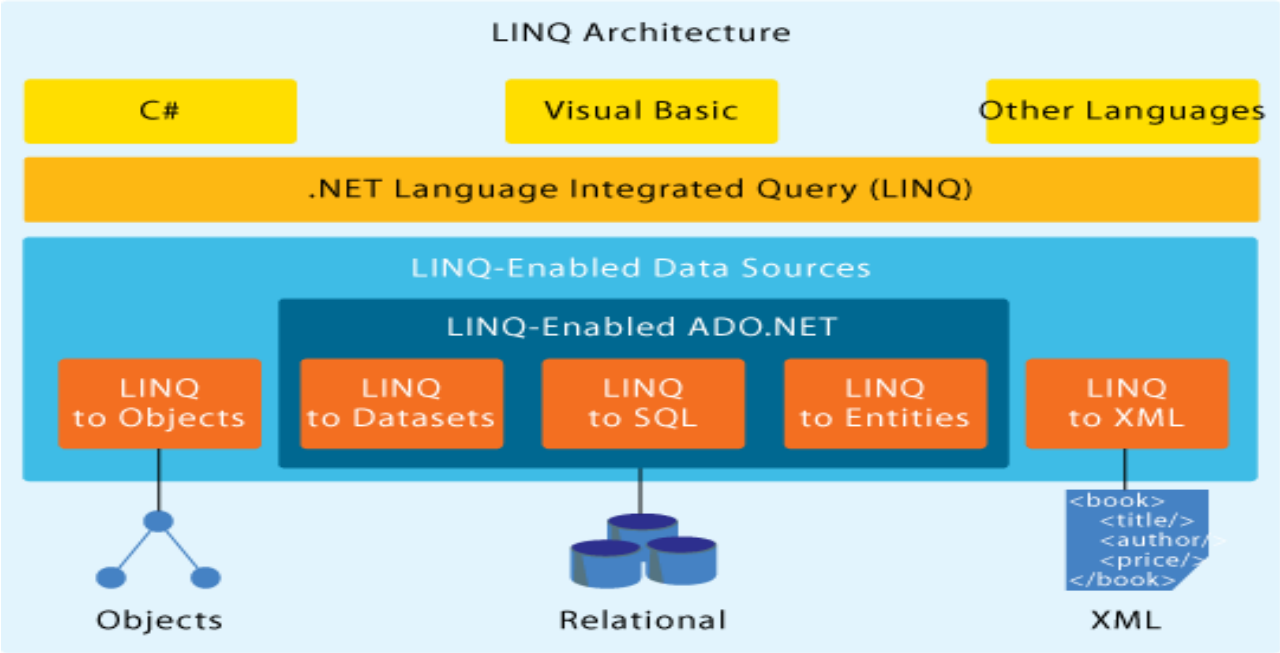
PUT: Update an existing resource on the server.

DELETE: Remove a resource from the server.

PATCH: Partially update a resource on the server.

LINQ (Language Integrated Query)


C#
VB
Other .Net languages
.Net Language Integrated Query(LINQ)
LINQ Enable Data Sources
ADO.NET LINQ Technologies
LINQ To Objects
LINQ To Datasets
LINQ To SQL
LINQ To Entities
LINQ To XML
Objects
Relational
XML



LINQ is known as Language Integrated Query and it is introduced in .NET 3.5 and Visual Studio 2008. **The beauty of LINQ is it provides the ability to .NET languages(like C#, VB.NET, etc.) to generate queries to retrieve data from the data source.** For example, a program may get information from the student records or accessing employee records, etc. In past years, such types of data were stored in a separate database from the application, and you need to learn different types of query language to access such types of data like SQL, XML, etc.

LINQ provides us with a common query syntax that allows us to query the data from various data sources in a uniform manner. That means using a single LINQ query we can get or set the data from various data sources such as SQL Server database, XML documents, ADO.NET Datasets, and any other in-memory objects such as Collections, Generics, etc.

It provides easy transformation, meaning you can easily convert one data type into another data type like transforming SQL data into XML data.

For Example: SQL is a structured query Language that is used to save and recuperate data from the database. Likewise, LINQ is an Organized Query Sentence structure. LINQ is the basic C#. It is utilized to recover information

from various kinds of sources, for example, XML, docs, collections, ADO.Net DataSet, Web Service, MS SQL Server, and different database servers.

Object Oriented Programming Concepts in C#

This article introduces Object Oriented Programming (OOP) in C#. OOPs is a concept of modern programming language that allows programmers to organize entities and objects. Four key concepts of OOPs are abstraction, encapsulation, inheritance, and polymorphism. Here learn how to implement OOP concepts in C# and .NET.

OOP Features

Here are the key features of OOP:

- Object Oriented Programming (OOP) is a programming model where programs are organized around objects and data rather than action and logic.
- OOP allows decomposing a problem into many entities called objects and then building data and functions around these objects.
- A class is the core of any modern object-oriented programming language such as C#.
- In OOP languages, creating a class for representing data is mandatory.
- A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.
- A class will not occupy any memory space; hence, it is only a logical representation of data.

To create a class, you use the keyword "class" followed by the class name:

```
class Employee
{
```

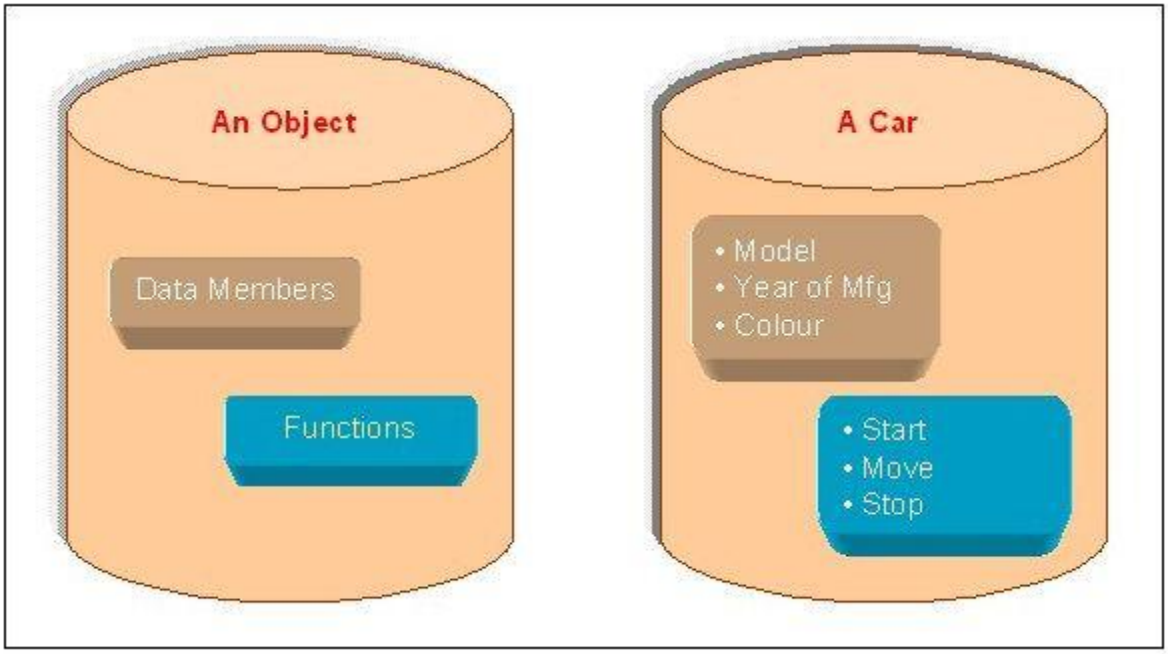
```
}
```

Object

1. The software is divided into several small units called objects. The data and functions are built around these objects.
2. An object's data can be accessed only by the functions associated with that object.
3. The functions of one object can access the functions of another object.

Objects are the basic run-time entities of an object-oriented system. They may represent a person, a place, or any item the program must handle.

- "An object is a software bundle of related variables and methods."
- "An object is an instance of a class."



A class will not occupy any memory space. Hence to work with the data represented by the class, you must create a variable for the class, which is called an object.

- When an object is created using the new operator, memory is allocated for the class in a heap, the object is called an instance, and its starting address will be stored in the object in stack memory.
- When an object is created without the new operator, memory will not be allocated in a heap; in other words, an instance will not be created, and the object in the stack will contain the value **null**.
- When an object contains null, it is impossible to access the class members using that object.

```
class Employee
{


}
```

C#Copy

Syntax to create an object of class Employee:

```
Employee objEmp = new Employee();
```

OOPs Concepts

The key concepts of OOPs are

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Abstraction

Abstraction is "To represent the essential feature without representing the background details."

- Abstraction lets you focus on what the object does instead of how it does it.
- Abstraction provides a generalized view of your classes or objects by providing relevant information.

- Abstraction is the process of hiding the working style of an object and showing the information about an object understandably.

Real-world Example of Abstraction

Suppose you have an object Mobile Phone.

Suppose you have three mobile phones as in the following:

- Nokia 1400 (Features: Calling, SMS)
- Nokia 2700 (Features: Calling, SMS, FM Radio, MP3, Camera)
- BlackBerry (Features: Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading Emails)

Abstract information (necessary and common information) for the "Mobile Phone" object is that it calls any number and can send an SMS.

So, for a mobile phone object, you will have the abstract class as in the following,

```
abstract class MobilePhone {
    public void Calling();
    public void SendSMS();
}
public class Nokia1400: MobilePhone {}
public class Nokia2700: MobilePhone {
    public void FMRadio();
    public void MP3();
    public void Camera();
}
public class BlackBerry: MobilePhone {
    public void FMRadio();
    public void MP3();
    public void Camera();
    public void Recording();
    public void ReadAndSendEmails();
}
```

C#Copy

Abstraction means putting all the necessary variables and methods in a class.

Example

If somebody in your college tells you to fill in an application form, you will provide your details, like name, address, date of birth, which semester, the percentage you have, etcetera.

If some doctor gives you an application, you will provide the details, like name, address, date of birth, blood group, height, and weight.

See in the preceding example what is in common?

Age, name, and address, so you can create a class that consists of the common data. That is called an abstract class.

That class is not complete, and other classes can inherit it.

Here is a detailed article on Abstraction In C#.

Encapsulation

Wrapping up a data member and a method together into a single unit (in other words, class) is called Encapsulation. Encapsulation is like enclosing in a capsule. That is, enclosing the related operations and data related to an object into that object.

Encapsulation is like your bag in which you can keep your pen, book, etcetera. It means this is the property of encapsulating members and functions.

```
class Bag {
```

```
    book;
    pen;
    ReadBook();
}
```

C#Copy

- Encapsulation means hiding the internal details of an object, in other words, how an object does something.
- Encapsulation prevents clients from seeing its inside view, where the behavior of the abstraction is implemented.
- Encapsulation is a technique used to protect the information in an object from another object.
- Hide the data for security, such as making the variables private, and expose the property to access the private data that will be public.

So, when you access the property, you can validate the data and set it.

Example 1

```
class Demo {
    private int _mark;
    public int Mark {
        get {
            return _mark;
        }
        set {
            if (_mark > 0) _mark = value;
            else _mark = 0;
        }
    }
}
```

C#Copy

Real-world Example of Encapsulation

Let's use as an example Mobile Phones and Mobile Phone Manufacturers.

Suppose you are a Mobile Phone Manufacturer and have designed and developed a Mobile Phone design (a class). Now by using machinery, you are manufacturing Mobile Phones (objects) for selling; when you sell your Mobile Phone, the user only learns how to use the Mobile Phone but not how the Mobile Phone works.

This means that you are creating the class with functions and by with objects (capsules), of which you are making available the functionality of your class by that object and without interference in the original class.

Example 2

TV operation

It is encapsulated with a cover, and we can operate it with a remote, and there is no need to open the TV to change the channel. Everything is private except the remote, so anyone can access the remote to operate and change the things on the TV.

Here is a detailed article on Encapsulation In C#.

Inheritance

When a class includes a property of another class, it is known as inheritance. Inheritance is a process of object reusability.

For example, a child includes the properties of its parents.

```
public class ParentClass {
    public ParentClass() {
        Console.WriteLine("Parent Constructor.");
    }
    public void print() {
        Console.WriteLine("I'm a Parent Class.");
```

```
    }
}
public class ChildClass: ParentClass {
    public ChildClass() {
        Console.WriteLine("Child Constructor.");
    }
    public static void Main() {
        ChildClass child = new ChildClass();
        child.print();
    }
}
```

Output

Parent Constructor.

Child Constructor.

I'm a Parent Class.

Polymorphism

Polymorphism means one name, many forms. One function behaves in different forms. In other words, "Many forms of a single object are called Polymorphism."

Real-world Example of Polymorphism

Example 1

A teacher behaves with his students.

A teacher behaves with their seniors.

Here the teacher is an object, but the attitude is different in different situations.

Example 2

A person behaves like a son in a house at the same time that the person behaves like an employee in an office.

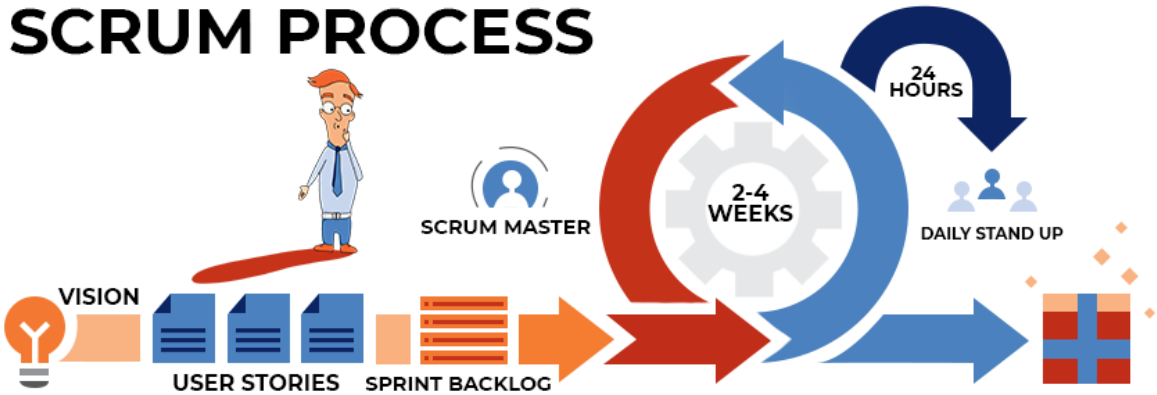
Example 3

Your mobile phone, one name but many forms:

- As phone
- As camera
- As mp3 player
- As radio

➢ **Scrum:**

Scrum is an agile development methodology used in the development of Software based on iterative and incremental processes. Scrum is an adaptable, fast, flexible and effective agile framework that is designed to deliver value to the customer throughout the development of the project.



The primary objective of Scrum is to satisfy the customer's need through an environment of transparency in communication, collective responsibility and continuous progress. The development starts from a general idea of what

needs to be built, elaborating a list of characteristics ordered by priority (product backlog) that the owner of the product wants to obtain.

Installation some extensions that is 1) Auto rename tag, 2) code runner, 3) highlight tag, 4) live server

Web Architecture is made up of three essentials elements:

1.**Website** (frontend) - Browser starts by loading the main HTML file, and then the CSS and JavaScript. User interacts with websites and is not concerned with the backend of web application.

2.**Server** (backend) - Once we've developed our website, we need to host it on server to make it accessible on the internet. Server, along with the database contains all the data of a website and facilitates user interaction

3.**IP Address** -

HTML stands for Hypertext Markup (Structure/layout) Language. It is used to design the web pages. With the help of HTML, you can create a complete website structure. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages and markup language defines the text document within the tag that define the structure of web pages.

Why is HTML used?

HTML is used to create the structure of web pages and website that are displayed on the Internet. HTML basically contains Tags and Attributes that are used to design the web pages. Also, we can link multiple pages using Hyperlinks.

HTML Basic Structure of Web Page:

The basic structure of an HTML page is laid out below. It contains the essential building-block elements (i.e., doctype declaration, HTML, head, title, and body elements) upon which all web pages are created.

HTML Basic Tags

<DOCTYPE! html> - tells browser you are using html file

<html> - root of an html document

<head> –container for metadata (contains meta information that data about data`<head>` primarily holds information for machine processing, not human-readability. For human-visible information, like top-level headings and listed authors, see the `<header>` element.

<body> – The body tag is used to enclose all the visible content of a webpage. In other words, the body content is what the browser will show on the front end.

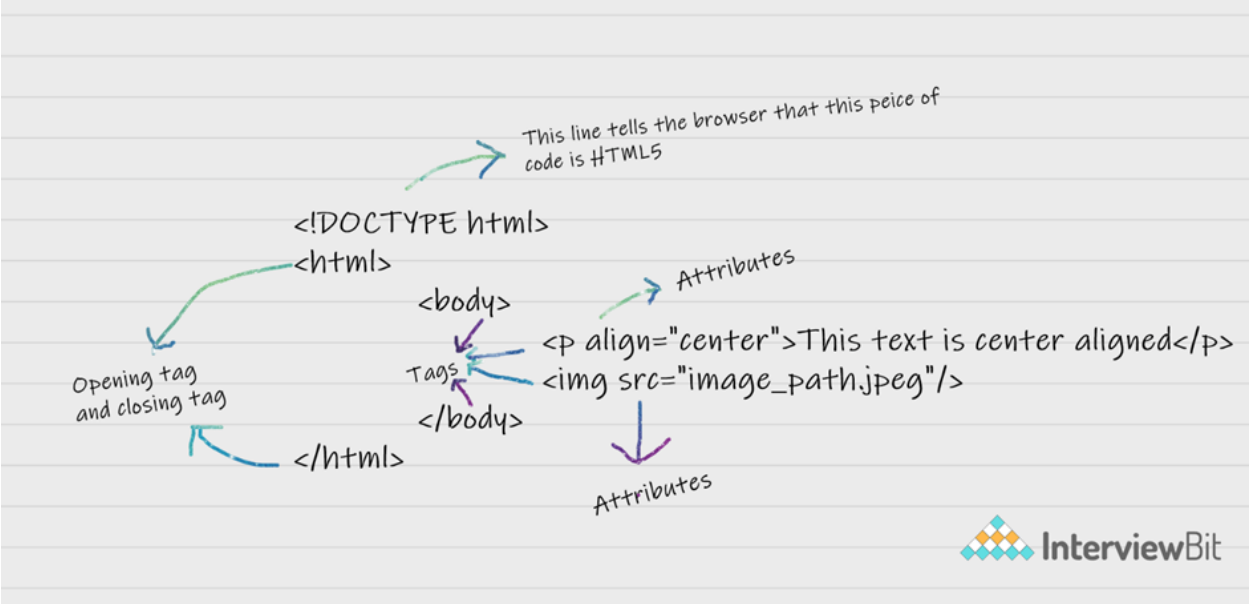
** paragraph of the page **

_{subscript}- we can write different formulas like chemical and maths.

Example: This is the basic example of HTML that display the heading and paragraph content.

```
<!DOCTYPE html>

<html>


<! -- Head Section content -->

<head>


   <! -- Page title -->

   <title>Basic Web Page</title>

</head>


<! -- Body Section content -->

<body>
```

```
    <! -- Used to display heading content -->
    <h1>Welcome to GeeksforGeeks</h1>

    <! -- Used to display paragrapg content -->
    <p>A computer science portal for geeks</p>
</body>

</html>
```



What are tags and attributes in HTML?

Tags are the primary component of the HTML that defines how the content will be structured/ formatted, whereas Attributes are used along with the HTML tags to define the characteristics of the element. For example, Interview questions , in this the 'align' is the attribute using which we will align the paragraph to show in the center of the view

Describe HTML layout structure.

Every web page has different components to display the intended content and a specific UI. But still, there are few things which are templated and are globally accepted way to structure the web page, such as:

- **<header>**: Stores the starting information about the web page.
- **<footer>**: Represents the last section of the page.
- **<nav>**: The navigation menu of the HTML page.
- **<article>**: It is a set of information.
- **<section>**: It is used inside the article block to define the basic structure of a page.
- **<aside>**: Sidebar content of the page.

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

CSS Example

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
```

```
  font-size: 20px;
}
```

It is a language that is used to describe the style of a document.

C#

C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework.

C# is used to develop web apps, desktop apps, mobile apps, games and much more.

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# 11**, was released in November 2022.

C# is used for: Mobile

- applications
- Desktop applications
- Web applications
- Web services
- Web sites
- Games

```
using System;


namespace Hello Capgemini
{
  class Program
  {
        static void Main(string[] args)
        {
```

```
        Console.WriteLine("Hello Capgemini!");
        }
 }
}
```

Main Method Syntax:

static: It means Main Method can be called without an object.

public: It is access modifiers which means the compiler can execute this from anywhere.

void: The Main method doesn't return anything.

Main(): It is the configured name of the Main method.

String []args: For accepting the zero-indexed command line arguments. args is the user-defined name. So you can change it by a valid identifier. [] must come before the args otherwise compiler will give errors.

Why Use 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

C# IDE

The easiest way to get started with C#, is to use an IDE.

An IDE (Integrated Development Environment) is used to edit and compile code.

Applications written in C# use the .NET Framework, so it makes sense to use Visual Studio, as the program, the framework, and the language, are all created by Microsoft.

C# Syntax

Line 1: `using System` means that we can use classes from the `System` namespace.

Line 2: A blank line. C# ignores white space. However, multiple lines makes the code more readable.

Line 3: `namespace` is used to organize your code, and it is a container for classes and other namespaces.

Line 4: The curly braces `{}` marks the beginning and the end of a block of code.

Line 5: `class` is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program.

Line 7: Another thing that always appear in a C# program, is the `Main` method. Any code inside its curly brackets `{}` will be executed. You don't have to understand the keywords before and after Main. You will get to know them bit by bit while reading this tutorial.

Line 9: `Console` is a class of the `System` namespace, which has a `WriteLine()` method that is used to output/print text. In our example it will output "Hello World!".

Note: Every C# statement ends with a semicolon `;`.

Note: C# is case-sensitive: "MyClass" and "myclass" has different meaning.

C# Comments

Single-line Comments

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by C# (will not be executed).

```
// This is a comment

Console.WriteLine("Hello World!");
```

C# Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by C#.

```
/* The code below will print the words Hello World

to the screen, and it is amazing */

Console.WriteLine("Hello World!");
```

C# Variables

Variables are containers for storing data values.

In C#, there are different **types** of variables (defined with different keywords), for example:

- int - stores integers (whole numbers), without decimals, such as 123 or -123
- double - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

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.

To create a variable that should store text, look at the following example:

Example

Create a variable called **name** of type `string` and assign it the value "**John**":

```
string name = "John";

Console.WriteLine(name);
```

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;

Console.WriteLine(myNum);
```

Other Types

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;

double myDoubleNum = 5.99D;

char myLetter = 'D';

bool myBool = true;
```

```
string myText = "Hello"
```

C# Data Types

Example

```
int myNum = 5;               // Integer (whole number)
double myDoubleNum = 5.99D;  // Floating point number
char myLetter = 'D';         // Character
bool myBool = true;          // Boolean
string myText = "Hello";     // String
```

Example

```
int myNum = 100000;
Console.WriteLine(myNum);
```

Booleans

A boolean data type is declared with the `bool` keyword and can only take the values `true` or `false`:

Example

```
bool isCSharpFun = true;
bool isFishTasty = false;
Console.WriteLine(isCSharpFun);   // Outputs True
Console.WriteLine(isFishTasty);   // Outputs False
```

Characters

The `char` data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';

Console.WriteLine(myGrade);
```

Strings

The `string` data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example

```
string greeting = "Hello World";

Console.WriteLine(greeting);
```

C# User Input

Get User Input

You have already learned that `Console.WriteLine()` is used to output (print) values. Now we will use `Console.ReadLine()` to get user input.

In the following example, the user can input his or hers username, which is stored in the variable `userName`. Then we print the value of `userName`:

Example

```
// Type your username and press enter

Console.WriteLine("Enter username:");


// Create a string variable and get user input from the keyboard and store it in the variable
```

```
string userName = Console.ReadLine();


// Print the value of the variable (userName), which will display the input value

Console.WriteLine("Username is: " + userName);
```

User Input and Numbers

The `Console.ReadLine()` method returns a `string`. Therefore, you cannot get information from another data type, such as `int`. The following program will cause an error:

Example

```
Console.WriteLine("Enter your age:");

int age = Console.ReadLine();

Console.WriteLine("Your age is: " + age);
```

The error message will be something like this:

```
Cannot implicitly convert type 'string' to 'int'
```

Like the error message says, you cannot implicitly convert type 'string' to 'int'.Luckily, for you, you just learned from the previous chapter (Type Casting), that you can convert any type explicitly, by using one of the `Convert.To` methods:

Example

```
Console.WriteLine("Enter your age:");

int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Your age is: " + age);
```

C# Operators

Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

Example

```
int x = 100 + 50;
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations:

Example

Operator	Name	Description	Example	Try it
+	Addition	Adds together two values	x + y	Try it »
-	Subtraction	Subtracts one value from another	x - y	Try it »

*	Multiplication	Multiplies two values	x * y	Try it »
/	Division	Divides one value by another	x / y	Try it »
%	Modulus	Returns the division remainder	x % y	Try it »
++	Increment	Increases the value of a variable by 1	x++	Try it »
--	Decrement	Decreases the value of a variable by 1	x--	

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

C# If ... Else

The if Statement

Use the `if` statement to specify a block of C# code to be executed if a condition is `True`.

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is `True`, print some text:

Example

```
if (20 > 18)

{

  Console.WriteLine("20 is greater than 18");

}
```

We can also test variables:

```
int x = 20;

int y = 18;

if (x > y)

{

  Console.WriteLine("x is greater than y");

}
```

The else Statement

Syntax

```
if (condition)

{

  // block of code to be executed if the condition is True

}

else
```

```
{
  // block of code to be executed if the condition is False
}
```

Example

```
int time = 20;
if (time < 18)
{
  Console.WriteLine("Good day.");
}
else
{
  Console.WriteLine("Good evening.");
}
// Outputs "Good evening."
```

C# Switch

This is how it works:

- The switch expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The break and default keywords will be described later in this chapter

Example

```
int day = 4;
switch (day)
```

```
{
  case 1:
        Console.WriteLine("Monday");
        break;
  case 2:
        Console.WriteLine("Tuesday");
        break;
  case 3:
        Console.WriteLine("Wednesday");
        break;
  case 4:
        Console.WriteLine("Thursday");
        break;
  case 5:
        Console.WriteLine("Friday");
        break;
  case 6:
        Console.WriteLine("Saturday");
        break;
  case 7:
        Console.WriteLine("Sunday");
        break;
}
// Outputs "Thursday" (day 4)
```

The break Keyword

When C# reaches a `break` keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

Example

```
int day = 4;

switch (day)

{

  case 6:

        Console.WriteLine("Today is Saturday.");

        break;

  case 7:

        Console.WriteLine("Today is Sunday.");

        break;

  default:

        Console.WriteLine("Looking forward to the Weekend.");

        break;

}

// Outputs "Looking forward to the Weekend."
```

C# While Loop

Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

C# While Loop

The `while` loop loops through a block of code as long as a specified condition is `True`:

Syntax

```
while (condition)
{
  // code block to be executed
}
```

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

Example

```
int i = 0;
while (i < 5)
{
  Console.WriteLine(i);
  i++;
}
```

C# For Loop

When you know exactly how many times you want to loop through a block of code, use the `for` loop instead of a `while` loop:

Syntax

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

Example

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

Example explained

Statement 1 sets a variable before the loop starts (`int i = 0`).

Statement 2 defines the condition for the loop to run (`i` must be less than `5`). If the condition is `true`, the loop will start over again, if it is `false`, the loop will end.

Statement 3 increases a value (`i++`) each time the code block in the loop has been executed.

Another Example

This example will only print even values between 0 and 10:

Example

```
for (int i = 0; i <= 10; i = i + 2)

{

  Console.WriteLine(i);

}
```

C# Break

You have already seen the `break` statement used in an earlier chapter of this tutorial. It was used to "jump out" of a `switch` statement.

The `break` statement can also be used to jump out of a **loop**.

This example jumps out of the loop when `i` is equal to `4`:

Example

```
for (int i = 0; i < 10; i++)

{

  if (i == 4)

  {

        break;

  }
```

```
  Console.WriteLine(i);
}
```

C# Arrays

Create an Array

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

```
string[] cars;
```

We have now declared a variable that holds an array of strings.

To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You access an array element by referring to the index number.

This statement accesses the value of the first element in **cars**:

Example

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

Console.WriteLine(cars[0]);

// Outputs Volvo
```

C# Methods

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A method is defined with the name of the method, followed by parentheses **()**. C# provides some pre-defined methods, which you already are familiar with, such as `Main()`, but you can also create your own methods to perform certain actions

Example

Create a method inside the Program class:

```
class Program

{

 static void MyMethod()

 {

        // code to be executed

 }
```

```
}
```

Example Explained

- `MyMethod()` is the name of the method
- `static` means that the method belongs to the Program class and not an object of the Program class. You will learn more about objects and how to access methods through objects later in this tutorial.
- `void` means that this method does not have a return value. You will learn more about return values later in this chapter.

Note: In C#, it is good practice to start with an uppercase letter when naming methods, as it makes the code easier to read

Call a Method

To call (execute) a method, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, `MyMethod()` is used to print a text (the action), when it is called:

Example

Inside `Main()`, call the `myMethod()` method:

```
static void MyMethod()

{

  Console.WriteLine("I just got executed!");

}



static void Main(string[] args)

{

  MyMethod();
```

```
}


// Outputs "I just got executed!"
```

A method can be called multiple times:

Example

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
  MyMethod();
  MyMethod();
}

// I just got executed!
// I just got executed!
// I just got executed!
```

C# Method Parameters

Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method.

They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a `string` called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
static void MyMethod(string fname)

{

  Console.WriteLine(fname + " Refsnes");

}


static void Main(string[] args)

{

  MyMethod("Liam");

  MyMethod("Jenny");

  MyMethod("Anja");

}


// Liam Refsnes

// Jenny Refsnes

// Anja Refsnes
```

C# Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

If we call the method without an argument, it uses the default value ("Norway"):

Example

```
static void MyMethod(string country = "Norway")
{
  Console.WriteLine(country);
}

static void Main(string[] args)
{
  MyMethod("Sweden");
  MyMethod("India");
  MyMethod();
  MyMethod("USA");
}

// Sweden
// India
// Norway
// USA
```

C# Return Values

In the previous page, we used the `void` keyword in all examples, which indicates that the method should not return a value.

If you want the method to return a value, you can use a primitive data type (such as `int` or `double`) instead of `void`, and use the `return` keyword inside the method:

Example

```
static int MyMethod(int x)

{

  return 5 + x;

}



static void Main(string[] args)

{

  Console.WriteLine(MyMethod(3));

}



// Outputs 8 (5 + 3)
```

This example returns the sum of a method's **two parameters**:

Example

```
static int MyMethod(int x, int y)

{

  return x + y;
```

```
}

static void Main(string[] args)
{
  Console.WriteLine(MyMethod(5, 3));
}

// Outputs 8 (5 + 3)
```

You can also store the result in a variable (recommended, as it is easier to read and maintain):

Example

```
static int MyMethod(int x, int y)
{
  return x + y;
}

static void Main(string[] args)
{
  int z = MyMethod(5, 3);
  Console.WriteLine(z);
}

// Outputs 8 (5 + 3)
```

C# Named Arguments

It is also possible to send arguments with the *key: value* syntax.

That way, the order of the arguments does not matter:

Example

```
static void MyMethod(string child1, string child2, string child3)
{
  Console.WriteLine("The youngest child is: " + child3);
}

static void Main(string[] args)
{
  MyMethod(child3: "John", child1: "Liam", child2: "Liam");
}

// The youngest child is: John
```

Method Overloading

With **method overloading**, multiple methods can have the same name with different parameters:

Example

```
int MyMethod(int x)
float MyMethod(float x)
double MyMethod(double x, double y)
```

Consider the following example, which have two methods that add numbers of different type:

Example

```
static int PlusMethodInt(int x, int y)
{
  return x + y;
}

static double PlusMethodDouble(double x, double y)
{
  return x + y;
}

static void Main(string[] args)
{
  int myNum1 = PlusMethodInt(8, 5);
  double myNum2 = PlusMethodDouble(4.3, 6.26);
  Console.WriteLine("Int: " + myNum1);
  Console.WriteLine("Double: " + myNum2);
}
```