* C# is a general-purpose, modern and object-oriented programming language pronounced as “C sharp”.
* A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages.
* **Why C#?**
  + C# has many other reasons for being popular and in demand. Few of the reasons are mentioned below:
  + Easy to start: C# is a high-level language so it is closer to other popular programming languages like C, C++, and Java and thus becomes easy to learn for anyone.
  + **Widely used for developing Desktop and Web Application**: C# is widely used for developing web applications and Desktop applications. It is one of the most popular languages that is used in professional desktop. If anyone wants to create Microsoft apps, C# is their first choice.
  + **Community:** The larger the community the better it is as new tools and software will be developing to make it better. C# has a large community so the developments are done to make it exist in the system and not become extinct.
  + **Game Development**: C# is widely used in game development and will continue to dominate. C# integrates with Microsoft and thus has a large target audience. The C# features such as Automatic Garbage Collection, interfaces, object-oriented, etc. make C# a popular game developing language.
* **Class:** A class is like a blueprint of a specific object that contains properties for storing data and functions to perform operations on the data. A class will not occupy any memory space and hence it is only a logical representation of data.

For example, the luxury car Ferrari. Ferrari is an object of the luxury car type. The luxury car is a class that indicates some characteristics like speed, color, shape, interior, etc. So any company that makes a car that meets those requirements is an object of the luxury car type. For example, every single car of BMW, Lamborghini, Cadillac are an object of the class called 'Luxury Car'. Here, 'Luxury Car' is a class, and every single physical car is an object of the luxury car class.

Likewise, in object-oriented programming, a class defines some properties, fields, events, methods, etc. A class defines the kinds of data and the functionality their objects will have.

* static void Main(): static keyword tells us that this method is accessible without instantiating the class. 5. void keywords tells that this method will not return anything. Main() method is the entry-point of our application. In our program, Main() method specifies its behavior with the statement Console.WriteLine(“Hello Geeks”); .
* OOPs concept:

Object Oriented Programming (OOP) is a programming model where programs are organized around objects and data rather than action and logic. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

Object: Objects are the basic run-time entities of an object-oriented system. They may represent a person, a place or any item that the program must handle. It has variables and methods in it.

"An object is an instance of a class"

When an object is created using the new operator, memory is allocated for the class in the heap, the object is called an instance and its starting address will be stored in the object in stack memory.

(Main points are already included in this documentation only go through examples from below link)  
<https://www.c-sharpcorner.com/UploadFile/mkagrahari/introduction-to-object-oriented-programming-concepts-in-C-Sharp/>

**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 means putting all the variables and methods in a class that are necessary.  
    For example: Abstract class and abstract method.
  + <https://www.c-sharpcorner.com/uploadfile/4624e9/abstraction-in-C-Sharp/>

**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 prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented.
* The need of encapsulation is to protect or prevent the code (data) from accidental corruption due to the silly little errors that we are all prone to make. In Object oriented programming data is treated as a critical element in the program development and data is packed closely to the functions that operate on it and protects it from accidental modification from outside functions. <https://www.c-sharpcorner.com/article/encapsulation-in-C-Sharp/>
* <https://www.geeksforgeeks.org/c-sharp-encapsulation/?ref=lbp>

**Polymorphism**

* <https://www.c-sharpcorner.com/UploadFile/ff2f08/understanding-polymorphism-in-C-Sharp/>
* See this topic in above link: Preventing Derived class from overriding virtual members

**Inheritance:**

Inheritance enables you to create new classes that reuse, extend, and modify the behavior defined in other classes. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class. A derived class can have only one direct base class. However, inheritance is transitive. If ClassC is derived from ClassB, and ClassB is derived from ClassA, ClassC inherits the members declared in ClassB and ClassA.

Conceptually, a derived class is a specialization of the base class. For example, if you have a base class Animal, you might have one derived class that is named Mammal and another derived class that is named Reptile. A Mammal is an Animal, and a Reptile is an Animal, but each derived class represents different specializations of the base class.

Ex. Base class can be Animal, Derived class can be Birds, Mammals, Reptiles.

<https://www.geeksforgeeks.org/c-sharp-inheritance/>

Another usage of new class: Your class has a base class, and this base class also has a property (which is not virtual or abstract) which is being overridden by your class. If you intend to override it put the "new" keyword after the public modifier.

* **C# Namespaces**

Namespaces in C# are used to organize too many classes so that it can be easy to handle the application.

In a simple C# program, we use System.Console where System is the namespace and Console is the class. To access the class of a namespace, we need to use namespacename.classname. We can use using keyword so that we don't have to use complete name all the time.

* Using keyword:

[https://www.c-sharpcorner.com/UploadFile/manas1/usage-and-importance-of-using-in-C-Sharp472](https://www.c-sharpcorner.com/UploadFile/manas1/usage-and-importance-of-using-in-C-Sharp472/)

(<http://burnignorance.com/c-coding-tips/two-ways-to-use-using-keyword-in-c/>)

In the context of C#, **dispose** is an object method invoked to execute code required for memory cleanup and release and reset unmanaged resources, such as file handles and database connections.

* Difference between array and list:

(<https://www.quora.com/What-is-the-difference-between-an-ARRAY-and-a-LIST>)

A list is a different kind of data structure from an array. The biggest difference is in the idea of direct access Vs sequential access. Arrays allow both; direct and sequential access, while lists allow only sequential access. And this is because the way that these data structures are stored in memory.

* What is Serialization?

<https://www.javatpoint.com/c-sharp-serialization>

* **String vs string:**

String (capital S) is a class in the .NET framework in the System namespace. The fully qualified name is System.String. Whereas, the lower case string is an alias of System.String.

* **enum:**

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain. for example, WeekDays.Monday is more readable then number 0 when referring to the day in a week.

* **StringBuilder:**

<https://www.tutorialsteacher.com/csharp/csharp-stringbuilder>

* **anonymous-type**

<https://www.tutorialsteacher.com/csharp/csharp-anonymous-type>

* **value-type-and-reference-type**

<https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type>

* **stack and heap/ boxing and unboxing**  
  <https://www.codeproject.com/Articles/76153/Six-important-NET-concepts-Stack-heap-value-types>
* **Interface:**

<https://www.tutorialsteacher.com/csharp/csharp-interface>

An interface is a declaration of methods, properties, indexers, and events. The entities that implement the interface must provide the implementation of declared functionalities.

* + It will tell about the structure.
  + It contains only methods(with declaration only).
  + Methods declaration inside interface doesn’t contain accessors.
  + Explicit interface implementation(by using typecasting).
  + Default and Explicit interface implementation.
* **Difference between field and property:**[**https://stackoverflow.com/questions/295104/what-is-the-difference-between-a-field-and-a-property**](https://stackoverflow.com/questions/295104/what-is-the-difference-between-a-field-and-a-property)
* **Partial**:  
  You can split the implementation of a class, a struct, a method, or an interface in multiple .cs files using the partial keyword. The compiler will combine all the implementation from multiple .cs files when the program is compiled.
* **Static**:  
  <https://www.tutorialsteacher.com/csharp/csharp-static>
* **Note:**   
  It is not recommended to use the ArrayList class due to performance issue. Instead, use List<object> to store heterogeneous objects. To store data of same data type, use Generic List<T>.
* **LINQ:**

LINQ (Language Integrated Query) is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats.  
LINQ is a structured query syntax built in C# and VB.NET to retrieve data from different types of data sources such as collections, ADO.Net DataSet, XML Docs, web service and MS SQL Server and other databases.

* **Exception handling:**

https://www.tutorialsteacher.com/csharp/csharp-exception-handling

* Conditional Attribute:

This attribute help in executing conditional debugging and tracing.

[Conditional("DEBUG")]

[Conditional("YO\_YO")]

Any constant can be inside the conditional attribute and it will check if that constant is defined or not. We will define the constant as a predefined processor.

#define DEBUG

#define YO\_YO

* Obsolete operator:

(<http://csharp-video-tutorials.blogspot.com/2012/07/part-52-c-tutorial-attributes-in-c.html?_sm_au_=iVV4jGL664674JDH>)

* Getters and setters

(<https://www.youtube.com/watch?v=EbW-1fPhXQE>)

* Reflection:

(<https://www.youtube.com/watch?v=y8-uq6Ur7Dc&list=PLAC325451207E3105&index=53>)

* Generics:

(<https://www.youtube.com/watch?v=iGR425gMKeA>)

Generics allow you to write a class or method that can work with any data type.

It can also be achieved by using object data type. But it will cause the performance degradation by causing problem of boxing and un-boxing.

* Ref keyword:

(<https://www.geeksforgeeks.org/difference-between-ref-and-out-keywords-in-c-sharp/>)

The ref is a keyword in C# which is used for the passing the arguments by a reference. Or we can say that if any changes made in this argument in the method will reflect in that variable when the control return to the calling method. The ref parameter does not pass the property.

Ex. <https://www.tutorialspoint.com/compile_csharp_online.php>

* Equals() performs value comparison whereas == performs reference comparison.
* **Delegates:**  
  What if we want to pass a function as a parameter? How does C# handles the callback functions or event handler? The answer is - delegate.

It is a function pointer.

It is mainly used to make your methods reusable. We can pass methods as a parameter in other functions.

**Multicast delegate:** If a delegate returns a value, then the last assigned target method's value will be return when a multicast delegate called.

Func is a generic delegate included in the System namespace. It has zero or more input parameters and one out parameter. The last parameter is considered as an out parameter.

**A Func delegate** type can include 0 to 16 input parameters of different types. However, it must include an out parameter for the result. For example, the following Func delegate doesn't have any input parameter, and it includes only an out parameter.

**Action is a delegate type** defined in the System namespace. An Action type delegate is the same as Func delegate except that the Action delegate doesn't return a value. In other words, an Action delegate can be used with a method that has a void return type.

**Predicate is the delegate** like Func and Action delegates. A predicate delegate methods must take one input parameter and return a boolean - true or false.

* **Events:**<https://www.tutorialsteacher.com/csharp/csharp-event>
* Abstracts:

It will contain abstract field as well as abstract methods.

We can use accessors inside abstract.

* **Virtual**: The virtual keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class.
* Employer e = new Employer();

Here, e is variable pointing to the object storage.

* Class has class members which include field and methods.
* List grow in size where array doesn’t
* this keyword:   
  <https://stackoverflow.com/questions/23250/when-do-you-use-the-this-keyword>  
  The “this” keyword in C# is used to refer to the current instance of the class. It is also used to differentiate between the method parameters and class fields if they both have the same name.

**Important Topics:**

* ORM:  
  <https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a>
* Web api response for requests:  
  <https://www.oreilly.com/ideas/how-a-restful-api-server-reacts-to-requests>
* POCO: Plain Old CLR Objects

**MVC Architecture:**

* Routing url:   
  <https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-2.2>
* MVC tutorial details:  
  <https://www.tutorialsteacher.com/mvc/layout-view-in-asp.net-mvc>
* \_ViewImports.cshtml:  
  <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-2.2>  
  Views and pages can use Razor directives to importing namespaces and use dependency injection. Directives shared by many views may be specified in a common \_ViewImports.cshtml file.  
    
  taghelpers examples: **asp-route-id**=""
* ASP.NET MVC - Razor. ... Razor is a markup syntax that lets you embed server-based code into web pages using C# and VB.Net. It is not a programming language. It is a server side markup language. Razor has no ties to ASP.NET MVC because Razor is a general-purpose templating engine.
* Razor View Engine

Microsoft introduced the Razor view engine and packaged with MVC 3. You can write a mix of html tags and server side code in razor view. Razor uses @ character for server side code instead of traditional <% %>.

* By default, it **returns** a **View**with same name as your action name or **returns** a custom **view** say myView.cshtml if you explicitly provide the **view** name like **return View**("myView")
* Validation:  
  <https://www.tutorialsteacher.com/mvc/implement-validation-in-asp.net-mvc>
* **Route:**<https://www.tutorialsteacher.com/mvc/routing-in-mvc>

Route defines the URL pattern and handler information. All the configured routes of an application stored in **RouteTable** and will be used by **Routing engine** to determine appropriate handler class or file for an incoming request.

* asp-route-AttributeName: here AttributeName should be same param passed inside the action method.

**ASP.NET Core :**

* **IoC Container (Inversion of control):** IoC Container (a.k.a. DI Container) is a framework for implementing automatic dependency injection. It manages object creation and it's life-time, and also injects dependencies to the class.
* ASP.NET Core refers dependent class as a Service. So, whenever you read "Service" then understand it as a class which is going to be used in some other class.
* string.Format(new System.Globalization.CultureInfo("en-US"),"{0:C2}", \_teacher.Salary)  
  In C2 C is currency and 2 is the number of decimal digits

**Entity framework core :**  
<https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext?view=efcore-2.1>

For help in package manager: Get-Help about\_entityframeworkcore  
  
Why EF core DbContext needed?

When you develop a new application, your data model changes frequently, and each time the model changes, it gets out of sync with the database. You started these tutorials by configuring the Entity Framework to create the database if it doesn't exist. Then each time you change the data model -- add, remove, or change entity classes or change your DbContext class -- you can delete the database and EF creates a new one that matches the model, and seeds it with test data.

add-migration SchoolDB  : this command creates the migration folder and file according to the context file created according to the created Model.

dotnet ef database update : this command takes creates the db (db name taken from the same configuration file that is passed in configure services) and then it creates the table according to the migration file.

EF commands:

* Add-Migration
* Remove-Migration
* Update-Database or Update-Database <name of the migration>

**Onion layer:**<https://www.c-sharpcorner.com/article/onion-architecture-in-asp-net-core-mvc/>

Doubts:

viewImports

@model

Scss not working (404)

Css files will load with every view

How to debug

* <https://docs.microsoft.com/en-us/aspnet/core/client-side/spa/angular?view=aspnetcore-3.1&tabs=visual-studio>

Important interview questions:

* Method overloading
* Method overriding
* Interface
* Loose coupling
* Constant
* Using
* Keyword
* Extension method
* Asynchronous
* Class
* String manipulation
* Dependency inversion
* Solid principles
* Readonly, lock keywod
* Garbage collection
* Dispose and finally
* Lampda expressions
* Linq
* Try catch finally
* Up casting down casting