* Using keyword:

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

* 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>

* 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:

It is a function pointer.

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

* Interfaces:

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.

* Abstracts:

It will contain abstract field as well as abstract methods.

We can use accessors inside abstract.

* 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 debugg