**Boxing/Unboxing –**

Boxing is the process of converting a [value type](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/value-types) to the type object or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit.

**Struct/Class –**

|  |  |
| --- | --- |
| * The struct is value type in C# and it inherits from System.Value Type. * Struct is usually used for smaller amounts of data. * Struct can’t be inherited to other type. * A structure can't be abstract. * No need to create object by new keyword. * Do not have permission to create any default constructor. | * The class is reference type in C# and it inherits from the System.Object Type. * Classes are usually used for large amounts of data. * Classes can be inherited to other class. * A class can be abstract type. * We can’t use an object of a class with using new keyword. * We can create a default constructor. |

**Interface/Abstract Class –**

* A class can implement any number of interfaces but a subclass can at most use only one abstract class.
* An abstract class can have non-abstract methods (concrete methods) while in case of interface all the methods has to be abstract.
* An abstract class can declare or use any variables while an interface is not allowed to do so.
* In an abstract class all data member or functions are private by default while in interface all are public, we can’t change them manually.
* In an abstract class we need to use abstract keyword to declare abstract methods while in an interface we don’t need to use that.
* An abstract class can’t be used for multiple inheritance while interface can be used as multiple inheritance.
* An abstract class use constructor while in an interface we don’t have any type of constructor.

**Const/readonly –**

**Constant**is known as “const” keyword in C# which is also known immutable values which are known at compile time and do not change their values at run time like in any function or constructor for the life of application till the application is running.  
  
**Readonly** is known as “readonly” keyword in C# which is also known immutable values and are known at compile and run time and do not change their values at run time like in any function for the life of application till the application is running. You can assay their value by constructor when we call constructor with “new” keyword.

**Ref/out -**

|  |  |
| --- | --- |
| **Ref** | **Out** |
| The parameter or argument must be initialized first before it is passed to ref. | It is not compulsory to initialize a parameter or argument before it is passed to an out. |
| It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method. | A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method. |
| Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter. | Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method. |
| It is not compulsory to initialize a parameter value before using it in a calling method. | A parameter value must be initialized within the calling method before its use. |
| When we use REF, data can be passed bi-directionally. | When we use OUT data is passed only in a unidirectional way (from the called method to the caller method). |
| Both ref and out are treated differently at run time and they are treated the same at compile time. | |
| Properties are not variables, therefore it cannot be passed as an out or ref parameter. | |

**Property(C#) –**

Properties are members that provide a flexible mechanism to read, write or compute the values of private fields, in other words by the property we can access private fields. In other words we can say that a property is a return type function/method with one parameter or without a parameter. These are always public data members. It uses methods to access and assign values to private fields called accessors.  
  
Now question is what are accessors?  
  
The get and set portions or blocks of a property are called accessors. These are useful to restrict the accessibility of a property, the set accessor specifies that we can assign a value to a private field in a property and without the set accessor property it is like a read-only field. By the get accessor we can access the value of the private field, in other words it returns a single value. A Get accessor specifies that we can access the value of a field publically.  
  
We have the three types of properties

* Read/Write.
* ReadOnly.
* WriteOnly

**Dispose/finalize –**

finalizer and dispose both are used for same task like to free unmanaged resources