**C# Course**

* The .Net framework consist of **Common Language Runtime(CLR) + .Net frm. Class Library**
* **CLR:** foundation of .net frmw. –
  + - manage code at execution time,
    - providing core services like as
      * memory management
      * code accuracy
      * many other aspects
* **Class Library:** collection of
  + - Classes
    - Interface
    - Value types , help to accomplished Common programming tasks,
      * Data collection,
      * File access
      * Work with text
* What is variable?
  + Creating variable means reserve memory location or space In memory, for stoing values its called variable.
  + **Name** and **datatype**
  + Its also called identifier.
  + Datatype- type data, memory size,
* Inputs in C#
  + Console.ReadLine() – read only string
    - Converted.ToInt32(Console.ReadLine()) – int type
    - As lile: ToInt32/16/64 , ToDoubke ,ToBloolean.
* **“var”** – automatically based on expression , called **implicitly typed**
* **Constants-** cann’t be changed
  + **Const double PI= 3.1416;**
* **Relational Operation:** >= , <= , == , !=
* **Logical:** && , || (AND , OR), ! Not
* **Method:** A method is a group of statements that perform a particular task.
* There are **three ways** to pass arguments to a method when the method is called: By **value**, By **reference**, and as **Output**.
  + Value – just variable name
  + Refrence – method(ref int x)
    - In main call like: int a=3; method(ref a);
* **Objects and Class**
  + In object-oriented programming, a **class**is a data type that defines a set of variables and methods for a declared **object**. A class is like a **blueprint**. It defines the data and behavior for a type.
  + An object is called an instance of a class.
* **Reference Type** - it contains a reference (the memory address) to the actual object on the heap.
* **Stack** is used for static memory allocation, which includes all your value types, like x.
* **Heap** is used for dynamic memory allocation, which includes custom objects, that might need additional memory during the runtime of your program.
* Stack is for static memory storage. Heap is for dynamic memory storage.
* **Access Modifier:**
  + Public
  + Private
  + Protected
  + C# supports the following access modifiers: public, private, protected, internal, protected internal.
    - The **public** access modifier makes the member accessible from the outside of the class.
    - The **private** access modifier makes members accessible only from within the class and hides them from the outside.
* **Encapsulation**
  + **Encapsulation** is the idea of "surrounding" an entity
  + Encapsulation is also called information hiding.
  + Encapsulation is implemented by using access modifiers. An access modifier defines the scope and visibility of a class member.
  + it is a good practice to encapsulate members of a class and provide access to them only through public methods.
  + Example: encapsulation
* **Constructors**
  + A class constructor is a special member method of a class that is executed whenever a new object of that class is created.
  + A constructor has exactly the same name as its class, is public, and does not have any return type.
* **Properties**
  + A **property** is a member that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they are **public data members**, but they actually include special *methods* called **accessors.**
  + The accessor of a property contains the *executable statements* that help in **getting** (reading or computing) or **setting** (writing) a corresponding field. Accessor declarations can include a get accessor, a set accessor, or both.

Properties > Accessors > get/set

* + **Auto-Implemented Properties**
    - public string Name { get; set; }
    - As you can see, you do not need to declare the private field name separately - it is created by the property automatically. *Name* is called an auto-implemented property. Also called auto-properties, they allow for easy and short declaration of private members.
* **Array**
  + An array is a data structure that is used to store a collection of data. You can think of it as a collection of variables of the same type.
  + int[ ] myArray = new int[5];
  + string[ ] names = new string[3] {"John", "Mary", "Jessica"};
  + double[ ] prices = new double[4] {3.6, 9.8, 6.4, 5.9};
  + **Forreach Loop:**
    - The foreach loop provides a shorter and easier way of accessing array elements.

foreach (int k in a) {

Console.WriteLine(k);

}

* + **Multidimensional Arrays**
    - An array can have multiple dimensions. A multidimensional array is declared as follows:
    - type[, , … ,] arrayName = new type[size1, size2, …, sizeN];
  + **Jagged Arrays**
    - A jagged array is an array whose elements are arrays. So it is basically an array of arrays.
    - The following is a declaration of a single-dimensional array that has three elements, each of which is a single-dimensional array of integers:
    - int[ ][ ] jaggedArr = new int[3][ ];
    - array.rank is basicly array.(how many dimensions does my array hold)
    - There are a number of methods available for arrays.
    - Max returns the largest value.
    - Min returns the smallest value.
    - Sum returns the sum of all elements.
  + ***String objects support a number of useful properties and methods:***
  + **Length** returns the length of the string.
  + **IndexOf(value)** returns the index of the first occurrence of the value within the string.
  + **Insert(index, value**) inserts the value into the string starting from the specified index.
  + **Remove(index)** removes all characters in the string after the specified index.
  + **Replace(oldValue, newValue**) replaces the specified value in the string.
  + **Substring(index, length**) returns a substring of the specified length, starting from the specified index. If length is not specified, the operation continues to the end of the string.
  + **Contains(value)** returns true if the string contains the specified value.
* **An array is a reference type**