C#

Intro >>

```
Console.WriteLine("Mina");                //Mina
Console.WriteLine("number= " + num);      //number= -5

Console.WriteLine("{0}{1}"  , num, num2); //-555
Console.WriteLine("{0}\n{1}", num ,num2); //-5  //55
//Console.WriteLine($" {num} ,{num2}");
                          ************************
string y = null;  //
int?   x = null;  //

                          ************************
int z = 15 % 10;
Console.WriteLine(z); // 5
                          ************************
Console.WriteLine("  \'  "); // '
Console.WriteLine("  \"  "); // "
Console.WriteLine("  \\  "); // \
Console.WriteLine("  \0  "); // string قيمة نهاية على تدل
Console.WriteLine("  \a  "); // صوت بتعمل
Console.WriteLine("  \b  "); // string قيمة فى حرف اخر يحذف
Console.WriteLine("  \n  "); // جديد سطر بينزل
Console.WriteLine("  \r  "); // string بعدة ما ويكتب قبلة ما بيحذق
Console.WriteLine("  \t  "); // الكلام بين مسافة بيعمل
Console.WriteLine("  \v  "); //  ♂
Console.WriteLine("  \f  "); //  ♀


Console.WriteLine( DateTime.Now.ToString("hh:mm:ss \n MM dd yyyy") );
   ************************************************************
#region Types Of Errors

    // Compile Error(CLR)      خطا وانتا بتكتب الكود
    // Run Time Error          خطا وبعد متعمل رن
    // Logical Error           خطا فى اللوجيك بتاع الناتج

#endregion
```

Datatype >>

	C# Type	.NET Type	Bytes	Range
Integral Numbers	**byte**	Byte	1	0 to 255
	short	Int16	2	-32,768 to 32,767
	int	Int32	4	-2.1B to 2.1B
	long	Int64	8	…
Real Numbers	**float**	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	…
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}
Character	**char**	Char	2	Unicode Characters
Boolean	**bool**	Boolean	1	True / False

Numeric Types

C# has the predefined numeric types shown in Table 2-1.

Table 2-1. Predefined numeric types in C#

C# type	System type	Suffix	Size	Range
Integral—signed				
sbyte	SByte		8 bits	-2^7 to 2^7-1
short	Int16		16 bits	-2^{15} to $2^{15}-1$
int	Int32		32 bits	-2^{31} to $2^{31}-1$
long	Int64	L	64 bits	-2^{63} to $2^{63}-1$
Integral—unsigned				
byte	Byte		8 bits	0 to 2^8-1
ushort	UInt16		16 bits	0 to $2^{16}-1$
uint	UInt32	U	32 bits	0 to $2^{32}-1$
ulong	UInt64	UL	64 bits	0 to $2^{64}-1$
Real				
float	Single	F	32 bits	$\pm(\sim 10^{-45}$ to $10^{38})$
double	Double	D	64 bits	$\pm(\sim 10^{-324}$ to $10^{308})$
decimal	Decimal	M	128 bits	$\pm(\sim 10^{-28}$ to $10^{28})$

```
int x = -5;  //32 bits
uint y = 5u; //32 bits

long  num = -5l;  //64 bits
ulong num2 = 5UL; //64 bits

float   x = 5.5 f; //32  bits
double  x = 5.5 d; //64  bits
decimal x = 5.5 m; //128 bits
```

****

Var

```
//var x = 20;
```

الفار بيفهم نوع الداتا الى جواة

Object

```
//object y = 30;
```

الاوبجيكت مش بيفهم نوع الداتا الى جواة

Dynamic

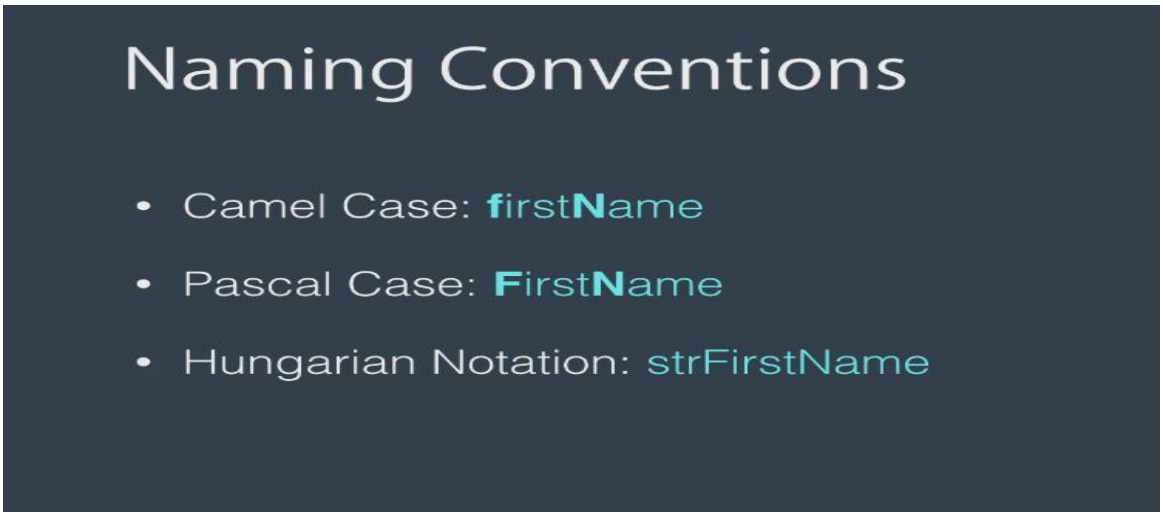
```
//dynamic z = 40;
```

الداينمك بيفهم نوع الداتا بس فى الرن تيم
او بيقبل انة يحجز ماكن فى المومورى فى الرن تيم

```
//var x ;       //Error
//dynamic z ;  // not Error
```

****

Variables >>



****

Constant

```
const double x = 3.14;        قيمة ثابتة لا بتتغير
```

Readonly

```
//readonly int x = 10;
//public Program()   //Constractor  قيمة ثابتة لا تتغير غير فى
//{
   // x = 20;
//}
```

****

Scope

```
//{
//   int x = 10;
//    {
//    }
//}
```

****

#Region

```
      //    code
```

#endRegion

****

Casting (Implicit , Explicit)

```
// Implicit
//byte x = 250;
//int y = x;

//Explicit
//int z = int.Parse(Console.ReadLine());

//int x = 250;
//byte y = (byte)x;
//Console.WriteLine(y);   250


//checked
//{
//    int x = 300;
//    byte y = (byte)x;
//    Console.WriteLine(y);   44
//}
```

```
string s = "1";
int i = Convert.ToInt32(s);
int j = int.Parse(s);

int z = i + 4;
int w = j + 4;

Console.WriteLine(z);       //5
Console.WriteLine(w);       //5
```

```
   ********************************************************************
float x = 5f, y = 2f;                int x = 5, y = 2;
float z = (x / y);                       float z = (x / y);
Console.WriteLine(z); // 2.5         Console.WriteLine(z); // 2
```

```
   ********************************************************************
int x = 1;                              int x = 1;
int y = x++;                                int y = ++x;
Console.WriteLine(y); //1                   Console.WriteLine(y); //2
```

```
   ********************************************************************
string x =                  Console.ReadLine()  ;
int    y =       int.Parse( Console.ReadLine() );
int    z = Convert.ToInt32( Console.ReadLine() );
   ********************************************************************
```

Conversion Operations

```
long x = 1000;
float y = x;
Console.WriteLine(y);  //1000

char x = 'c';
int y = x;             // float  double  decimal
Console.WriteLine(y);  //99
      ---------------------------------------------------------------------
byte x = byte.MaxValue;
byte y = byte.MinValue;
Console.WriteLine(x);      // 256
Console.WriteLine(y);      // 0
      ---------------------------------------------------------------------
int z = 260;
byte x = (byte)z;
Console.WriteLine(x);      // 4
      ---------------------------------------------------------------------
```

Priority of Operation >>>>>>>>>

1-Braces **()**

2-Power **^**

3-Prefix **++x , --x**

4-Arithmatic Operations *** , / , % , + , -**

5-Comparison Operations

"Inquality" **> , >= , < , <=**

"Enquality" **== , !=**

6-Logical Operations **&& , ||,&,|**

7-Assignmant Operations **+= , -= , *= , /= , %=**

8-Postfix **x++ , x--**

```
int x = 1;
Console.WriteLine(x++);    // 1
Console.WriteLine(++x);    // 3
```

****

Multiple Catch Statement

```
public static void Main(String[] args)
{
    try
    {
        string[] s={"aa","bb","cc","dd"};
        for (int i = 0; i < 4; i++)
        { Console.WriteLine(s[i]); } //IndexOutOfRangeException

        Console.WriteLine("Enter your number = ");//FormatException
        double num = Convert.ToDouble( Console.ReadLine() );

        int x=10,y=0;
        int z = x / y; //DivideByZeroException
    }

    catch (IndexOutOfRangeException e)
    { Console.WriteLine(e.ToString()); } //e.ToString() or e.Message

    catch (FormatException e)
    { Console.WriteLine(e.ToString()); } //e.ToString() or e.Message

    catch (DivideByZeroException e)
```

```
            { Console.WriteLine(e.ToString()); } //e.ToString() or e.Message

            catch (Exception e)// of any exception
            { Console.WriteLine(e.ToString()); } //e.TargetSite

            finally
            { Console.WriteLine("End"); }
        }
```

****

Selection Statements

```
// IF  ,  IF..Else  ,  IF..Else..(Neasted IF)  ,  Switch

1>> if() {  }
2>> if() {  } else { }
3>> if() {  }
else
{
      if() {  }
      else
      {
            if() {  }
            else { }
      }
}
```

```
int x = 50;
switch(x)
{
    case 30: Console.WriteLine("30");    break;
    case 10: Console.WriteLine("10");    break;
    case 50: Console.WriteLine("50");    break;  // 50
    default: Console.WriteLine("Error"); break;
}
```

##########

Itration Statements

```
// for  ,  While  ,  do..while  ,  foreach  (Array)

for (int i = 0; i < 5; i++)
{ Console.WriteLine(i); }                    // 0
                                                   // 1
int i = 0;                                         // 2
while ( i < 5 )                                    // 3
{ Console.WriteLine(i); i++; }               // 4
```

```
int x = 10;
do                          //10
{                           //20
Console.WriteLine(x);       //30
x += 10;                    //40
}while(x < 50);
```

Jump Statements

```
// Continue  ,  Break  ,  go..to
```

break;

```
for (int i = 0; i < 5; i++)
{                                   //0
    Console.WriteLine(i);           //1
    if (i == 2) { break; }          //2
}
```

continue;

```
for (int i = 0; i < 5; i++)         //0   //0
{                                   //1   //1
    Console.WriteLine(i);           //2
    if (i >= 2) { continue; }       //3   //3
    Console.WriteLine(i);           //4   //4
}
```

****

goto

```
for (int i = 0; i < 10; i++)
{
     Console.WriteLine(i); //0 //1 //2 //3
     if (i == 3) { goto g; }
}
g:
```

****

Array

Array ===> Multi value In One Variable with the same data type
Array Types ===> Single Dimention , Multi Dimention , Jagged Array

```
const int size = 5;
int[] xxx = new int[size]  { 1, 2, 3, 4, 5 };
int[] xx  = new int[5];    Assert 5 variable but no values
int[] x   =                { 1, 2, 3, 4, 5 };

for (int i = 0; i < 5; i++)   or (i < x.Length)
{ Console.WriteLine( x[i] ); }  //1 //2 //3 //4 //5

string[] name = { "name1","name2","name3" };
foreach (string item in name)
{ Console.WriteLine( item ); }   //name1 //name2 //name3
```

Ex- ArrayFunction

```
//Array.Reverse(x); 10 20 30   //30 //20 //10
//Array.Sort(x);    20 10 30   //10 //20 //30
//Console.WriteLine(Array.IndexOf(x, 20)); //1
//Array.Copy(arr1, arr2, Length);
//Array.Clear(arr1, Index, Length); (array , From , To)
```

****

Multi Dimention Array

```
int[,] x = new int[3, 5];

int[,] x={ {1, 2, 3, 4, 5},

            {1, 2, 3, 4, 5},

            {1, 2, 3, 4, 5} };

for (int i = 0; i < 3; i++)
{
     for (int j = 0; j < 5; j++)        //1 2 3 4 5
     { Console.Write(x[i,j]+" "); }     //1 2 3 4 5
     Console.WriteLine( "\n" );         //1 2 3 4 5
}

foreach (int item in x)
{ Console.Write(item+" "); }
```

****

Jagged Array

```
int[][] x   = new int[2][];
x[0]=new int[2] { 1, 2 };
x[1]=new int[4] { 1, 2, 3, 4 };

for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < x[i].Length; j++)       // 1  2
    {
        Console.Write(x[i][j] + " ");           // 1  2  3  4
    }
    Console.WriteLine("\n");
}
```

****

Multi Dimention and Jagged Array

```
int[,,] x   = new int[3,2,5]

{
    { { 1, 2, 3, 4, 5 },{ 5, 4, 3, 2, 1} },
    { { 1, 2, 3, 4, 5 },{ 5, 4, 3, 2, 1} },
    { { 1, 2, 3, 4, 5 },{ 5, 4, 3, 2, 1} }
};

for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 2; j++)
    {
         for(int k =0;k<5;k++)
         { Console.Write(x[i, j, k] + " "); }
    }
    Console.WriteLine("\n");
```

```
}

foreach (int item in x)
{ Console.Write(item+" "); }
```

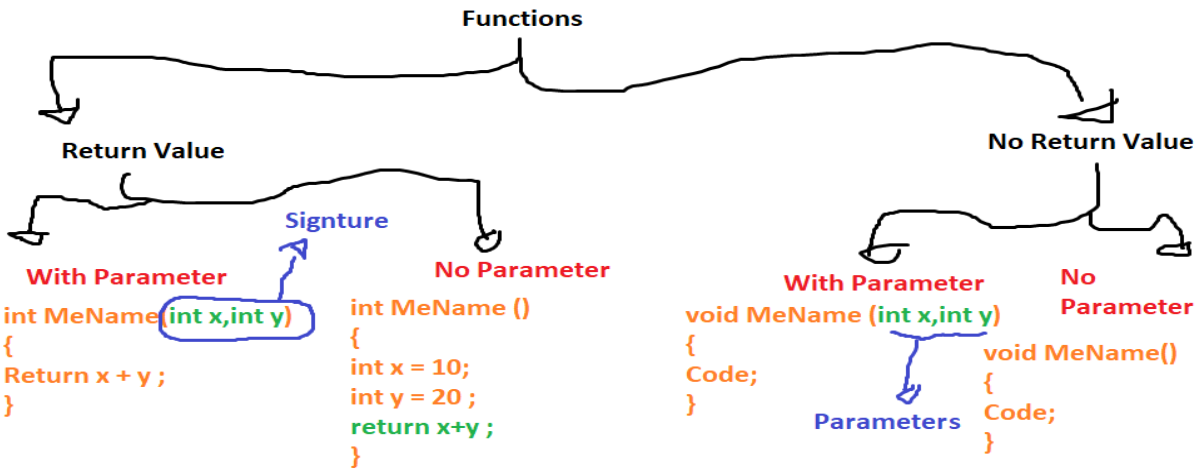
****

List

```
List<int> IntergerList = new List<int>();
IntergerList.Add(10);
IntergerList.Add(20);
IntergerList.Add(30);

foreach (var item in IntergerList)
{
       Console.WriteLine(item); //10 //20 //30
}
       ----------------------------------------------------------------------

object Emp1 = new { Id = 1, Name = "Ahmed", Salary = 2000 };
object Emp2 = new { Id = 2, Name = "Ali", Salary = 3000 };
object Emp3 = new { Id = 3, Name = "Sara", Salary = 4000 };

List<object> EmpList = new List<object>();
EmpList.Add(Emp1);
EmpList.Add(Emp2);
EmpList.Add(Emp3);

foreach (object item in EmpList)
{
        Console.WriteLine(item);
}
```

****

Function



```
    class MyClass
    {
        public static void Display()  // No Parameter, No Return
        {
            Console.WriteLine("Welcome");
        }
        public static void Display(string name) // With Parameter, No Return
        {
            Console.WriteLine("Welcome " + name);
        }
                    ----------------------------------------
        public static int Sum() // No Parameter, Return
        {
            int x = 1;
            int y = 2;
            int z = x + y;
            return z;
        }
        public static int Sum(int a, int b) // With Parameter, Return
        {
            int z = a + b;
            return z;
        }
    }
            ------------------------------------------------------------
```

How To Call Function

```
public static void Main(String[] args)
{
                      Display();       //Welcome
                      Display("aaa")    //Welcomeaaa
    Console.WriteLine( Sum()       );    //3
    Console.WriteLine( Sum(3,2)    );    //5
}
    ****************************************************************
```

How to document your functions

```
/// <summary>
/// This Function Return Welcome Message With Name
/// </summary>
/// <param name="name">Please Enter String Parameter</param>
    ****************************************************************
```

Param

```
static void Display(params string[] name) //array of parameter
{
      foreach (var item in name)
      {
           Console.WriteLine("Welcome " + item);
      }
}
    ****************************************************************
```

call by Value, Reference, Out

```
static void fun( int num )
{
  num += 2;
  Console.WriteLine(num);
}

public static void Main(String[] args)
{
    int x = 3;
    Console.WriteLine(x);   //3
    fun(x);                 //5
    Console.WriteLine(x);   //3
}
```

Reference values (ref)

```
static void fun(ref int num )
{
  num = 5;
  Console.WriteLine(num);
}

public static void Main(String[] args)
{
    int x = 3;
    Console.WriteLine(x);   //3
    fun(ref x);             //5
    Console.WriteLine(x);   //5
}
```

Out parameter (out)

```
static void fun(out int num )
{
  num = 5;
  Console.WriteLine(num);
}

public static void Main(String[] args)
{
    int x ;
    fun(out x);             //5
    Console.WriteLine(x);   //5
    Console.ReadKey();
}
```

Ex:-

```
static int sum(ref int a ,ref int b , out int z)
{
    z = a + b;
    return z;
}
public static void Main(String[] args)
{
    int x = 10;
```

```
    int y = 20;
    int result ;
    Console.WriteLine(sum(ref x,ref y ,out result));
    Console.ReadKey();
}
```

****

Generic

```
static void display<g>(g name) //Generic Function
{
      Console.WriteLine("Welcome "+name);
}
display<string>("Ahmed");
display<int>(1);
display<double>(10.5);
```

```
class MyClass2<t1,t2,t3>  //Generic Class
{
   public  t1 x;
   public  t2 y;
   public  t3 z;
}

MyClass2<int, string, double> m = new MyClass2<int, string, double>();
m.x = 23;
m.y = "aaaaa";
m.z = 12.56;
```

****

Delegate

Pointer to function

****

OOP

Static Methods

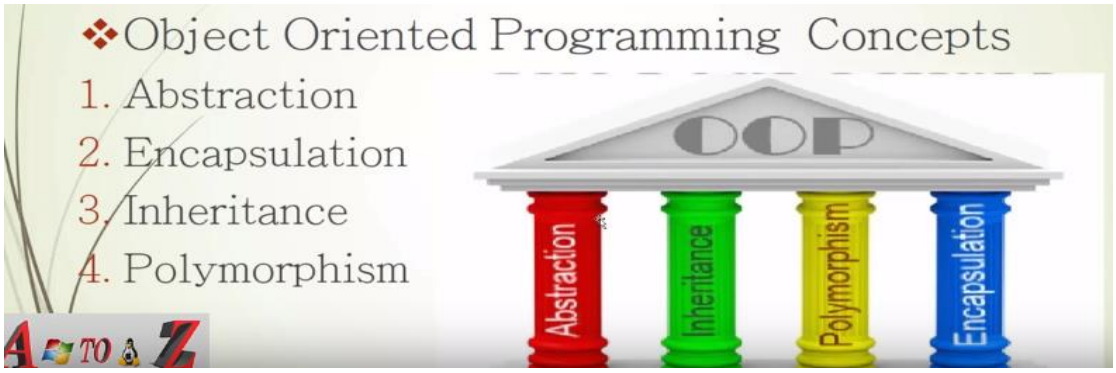
```
class Class1
{
   public int x;
   public void method1(int id)
   { x = id; }

   public static int y;
   public static void method2(int id)
   { y = id; }
}
                  ------------------------------------------
    public static void Main(String[] args)
    {
       Class1 c = new Class1();
       c.x = 1;                                  // x=1
       c.method1(2);                             // x=2

       Class1.y = 3;      // static properties   // y=3
       Class1.method2(4); // static method       // y=4
    }
 ************************************************************
```

Static Classes

```
static class Class1
{
    static public string name = "mina";
    static public void print()
    { Console.WriteLine(name); }
}
                  ------------------------------------------
    public static void Main(String[] args)
    {
        Class1.print();//mina
    }
 ************************************************************
```

this

```
class Class1
{
   public int id;
   public string name;

   public Class1(int id, string name)// Parametrized Constructor(2 Values)
    {
        this.id = id;
        this.name = name;
    }
}
      ******************************************************
```



Abstraction

Abstruct & Sealed Class

Inheritance

Base & Child Class

ابيستريكت هو كلاس ينفع اورث منة ومينفعش اخد منة اوبجيكت
السيلد هو كلاس مينفعش اورث منة وينفع اخد منة اوبجيكت

```
abstract class Human  //Base class
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
}

class Employee : Human   //Child class
{
    public double Salary { get; set; }

    public double GetSalary( double BSal , double BounSal , double Deduction)
    {
        double TotalSalary = (BSal + BounSal) -Deduction;
        return TotalSalary;
    }
    public void Display()
    {
        Console.WriteLine("Welcome to Employee");
    }
}

class NotEmployee : Human
{
    public void Eat()
    {
        Console.WriteLine("Eating");
    }
    public void Drink()
    {
        Console.WriteLine("Drinking");
    }
}
```

```
sealed class Manager : Employee
{
    public double Bouns { get; set; };
    public void Hire()
    {
        Console.WriteLine("Hiring");
    }
    public void Fire()
    {
        Console.WriteLine("Firing");
    }
}
```

Sealed Method

```
class Class1
{
    public void print()
    { Console.WriteLine("mina");}
}
```

```
class Class2 : Class1
{
    public sealed override void print()
    { Console.WriteLine("mina 2");}
}
```

```
class Class3 : Class2
{
    public  override void print() //Error the Method is Sealed in class2
    { Console.WriteLine("mina 3");}
}
```

****

Abstract Methods

- Abstruct class = Abstruct method
- Abstruct method without body
- Abstruct class without object

```
abstract class Class1
{
    public int x;
    public int y;

    public abstract void printX();
    public abstract void printY();
}
```

```
class Class2 : Class1
{
    public override void printX()
    {
```

```
        Console.WriteLine(x);
    }
    public override void printY()
    {
        Console.WriteLine(y = 3);
    }
}
                ---------------------------------------------
class Class3 : Class1
{
    public override void printX()
    {
        Console.WriteLine(x);
    }
    public override void printY()
    {
        Console.WriteLine(y = 6);
    }
}
                ---------------------------------------------

   public static void Main(String[] args)
    {
        Class2 c2 = new Class2();
        c2.x = 2;
        c2.printX();  //2
        c2.printY();  //3

        Class3 c3 = new Class3();
        c3.x = 5;
        c3.printX();  //5
        c3.printY();  //6
    }
```

****

Interface

```
interface Iemployee
{
    public double GetSalary(double BSal , double BounSal , double Deduction);
    public void Desplay();
}
interface INotEmployee
{
    public void Eat();
}
interface INotEmployee2
{
    public void Drink();
}
interface IManager
{
    public void Hire();
    public void Fire();
}
                ---------------------------------------------
class Employee : Human , Iemployee
{
   public double GetSalary( double BSal , double BounSal , double Deduction)
```

```
        {
        }
        public void Display()
        {
        }
    }

    class NotEmployee : Human , INotEmployee , INotEmployee2
    {
        public void Eat()
        {
        }
        public void Drink()
        {
        }
    }
    sealed class Manager : Employee , IManager
    {
        public void Hire()
        {
        }
        public void Fire()
        {
        }
    }
```

****

Encapsulation

Access Modifiers Types

```
Private            >> only by the class OR in the struct OR {}
Protected          >> only by the class OR the sub class
Internal           >> any class in the same Solution(name space)
protected internal >>( protected in other name space + internal in the name space)
Public             >>.......
```

Default Access Modifier for class members = internal
Default Access Modifier for Properitie = private

****

Automatic Properities (prop+Tab)

```
class Class1
    {
        public string name { public get; public set; }
        public int age { get; set; }
    }
                    ---------------------------------------------

        public static void Main(String[] args)
        {
            Class1 c = new Class1();
            c.name = "mina";
            c.age = 19;
```

```
            Console.WriteLine(c.name); //mina
            Console.WriteLine(c.age);  //19
        }
```

****

```
private int x;
public int Percentage { get; private set; }
public int Percentage
{
     set
     {
          if (value >= 50)
          {
              x = 1;
          }
          else
          {
              x = 0;
          }
     }
     get
     {
           return x;
     }
}
```

****

Polymorphism (Override , Overload)

Overload >> 2 Function but different head in the same class

```
    class people
    {
        public void Show()
        {
            Console.WriteLine("Welcome");
        }
        public void Show(string name)
        {
            Console.WriteLine("Welcome" + name);
        }
        public void Show(string name1 , string name2)
        {
            Console.WriteLine("Welcome" + name1);
            Console.WriteLine("Welcome" + name2);
        }
```

---

at the parameter (ref,out) >> address is

Error not overload

```
        public int Sum(ref int a , ref int b)
        {
            return 0;
        }
        public int Sum(out int a , out int b )
        {
            return 0;
        }
    }
```

Override >> 2 Function in the same head but different class

```
class Class1
{
    public int x = 2;

    public virtual void print()
    { Console.WriteLine(x); }
}
                    ---------------------------------------------
class Class2 : Class1
{
    public override void print()
    { Console.WriteLine("Ford"); }
}
                    ---------------------------------------------
class Class3 : Class1
{
    public override void print()
    { Console.WriteLine("Nissan"); }
}
                    ---------------------------------------------
class Class4 : Class1
{
    public override void print()
    { Console.WriteLine("Toyta"); }
}
                    ---------------------------------------------
public static void Main(String[] args)
{
     Class1 car = new Class1();
     car.print();    // 2

     Class2 ford   = new Class2();
     ford.print();   // Ford

     Class3 nessan = new Class3();
     nessan.print(); // Nissan

     Class4 toyta  = new Class4();
     toyta.print();  // Toyta

    Console.ReadKey();
}
```

****

Partial Class

```
partial class Class // Class1
{
    public string name = "mina";
    public int age = 19;
}
                    ---------------------------------------------
partial class Class // Class2
```

```
{
    public void printName()
    { Console.WriteLine(name); }

    public void printAge()
    { Console.WriteLine(age); }
}
                    ---------------------------------------------
    public static void Main(String[] args)
    {
        Class c = new Class();
        c.printName(); //mina
        c.printAge();  //19
    }
  ------------------------------------------------------------------------------
```

Partial Method

```
partial class Class // Class1
{
    private string name = "mina";
    public int age = 19;

    partial void print(); // method1
}
                    ---------------------------------------------
partial class Class// Class2
{
    partial void print() // method1
    { Console.WriteLine(age); }

    public void printAge()
    { print(); }
}
                    ---------------------------------------------
    public static void Main(String[] args)
    {
        Class c = new Class();
        c.printAge(); // 19
    }
   ************************************************************
```

Constructor

```
class people
{
    public people() //Default Constructor
    {
        Console.WriteLine("Welcome Constructor");
    }
    public people(string name) //Parametrized Constructor
    {
        Console.WriteLine("Welcome Constructor" + name);
    }

    public static int Id { get; set; }
    public static string Name { get; set; }
```

```
    static people() //static Constructor(Access static member in the class)
    {
        Id = 1;
        Name = "Ahmed";
        Console.WriteLine("Welcome " + Name + " Your Id Is " + Id);
    }
                ------------------------------------------
    public static void Main(String[] args)
    {
        people c1 = new people ();//Welcome Constructor

        people c2 = new people (" Mina");//Welcome Constructor Mina
    }
        ----------------------------------------------------------------
```

Destructor

```
    ~people()//Destructor(run in End the object)
    {
        Console.WriteLine("Welcome Destructor");
        GC.Collect();//clear the variable memory is null
    }

    public void Dispose()
    {
        Console.WriteLine("Welcome Dispose");
    }
}
    static void Main(string[] args)
    {
        using (people p1 = new people())
        {

        }
        using (people p2 = new people())
        {

        }
    }
```

****

Nested Classes

```
class Class1
{
     public int x=1;
     public int y=2;

     public void print1()
     { Console.WriteLine("x="+x+" - "+"y="+y); }

     public class class2
     {
        public int z=3;
        public void print2()
        { Console.WriteLine("z="+z); }
     }
```

```
    }
                    ------------------------------------------
    class Program
    {
        public static void Main(String[] args)
        {
           Class1 c = new Class1();
           c.print1(); //x=1  -  y=2

           Class1.class2 c2 = new Class1.class2();
           c2.print2();//z=3
        }
    }
```

****

enum

```
enum number     // index = integer
{
     num1,           // index = 0
     num2,           //         1
     num3            //         2
}
                                --------OR-----------
enum data : byte // index = byte
{
    num1=4,          // index = 4
    num2=5,          //         5
    num3=6           //         6
}
data d = data.num1;
Console.WriteLine(d);   //num1
int x = (int) d;
Console.WriteLine(x);   //4
```

****

struct

```
struct fullName
{
    public string first;
    public string last;
}

public static void Main(String[] args)
{
    fullName fn;
    fn.first="Modern";
    fn.last ="Academy";
    Console.WriteLine(fn.first+" "+fn.last); //Modern Academy
}
```

****

Throw

```
    class Program
    {
        public static void fun()
        {
```

```
        try
        {   int x, y, z ;

            Console.WriteLine("Enter your number 1= ");
            x = Convert.ToInt16( Console.ReadLine() );

            Console.WriteLine("Enter your number 2= ");
            y = Convert.ToInt16(Console.ReadLine());

            z = x * y;
            Console.WriteLine("z= "+z);
        }
        catch (FormatException e)
        {
              throw new Exception();             // print Exception
            //throw new Exception("Try again"); // print "Try again"
            //throw new Exception("Try again", e.InnerException);
                     //print "Try again" + save the data
        }
    }
```

****

Attribute

****

Outline >>

1- Intro
2- Datatypes (Integers , Strings , Date and time , Fraction , Booleans)
4- ReadOnly , Constant and Variables
4- Scope , #Region
4- Casting (Implicit , Explicit)
5- Conversion Operations
6- Priority Of Operations
6- Multiple Catch (Try , Catch and Finally)
7- Statements (Selection , Iteration , Jump)
8- array and array types and (foreach)
10- List
10- Function (Param)
12- Call By Value ,Call By Reference , Out
13- Generic
13- Delegate
14- Object Oriented Programming
15- Abstraction (Abstract Class & Sealed Class)

15- Inheritance (Base Class & Child Class)
16- Sealed Method
16- Abstract Method
17- Interface Or Header File
18- Encapsulation
18- Access Modifiers
18- Default Access Modifier for class members
19- Polymorphism (Overload , Override)
20- Partial (Class & Method)
21- Constructor
- **default Constructor**
- **parametarized Constructor**
- **static Constructor**
- **Function Destructor**
- **Function Dispose**

22- Nested Classes
23- Enum
23- struct
23- Throw
24- Attribute