CH: 0 INTRO dotNET

#1 INTRO

* dotNet

- dotNet is a free open source, & cross platform, development platform created by microsoft
- Crossplatform means runs on any OS
- First released: in 2002 by microsoft
- Purpose: to build web, mobile, desktop, console, cloud application
- you can use F# (Fsharp) or C# for coding in .Net

.Net / .Net Core	.Net framework
• Runs on any OS	• Runs only on windows
• Open source & accepts contribution from community	• Open source but does not accept any contribution

* History

- Before .Net we have COM (Component Object Model)
- Com was build by microsoft but was having 2 problems
  1) not support full OOPs concept
  2) platform dependent
- To overcome these issues we have .Net framework which was released in 2002

* .Net

- .Net stands for Network Enabled Technology
- (.) dot refers to object - oriented
- (Net) refers to Internet
- .Net is a framework that supports many programming lang & technologies

  Lang supported = C#, F#, VB, J#

  Technology supported = ASP.Net, LINQ, WCF, ADO.Net

* Types of .Net framWork

1) .Net FrameWork: it only support Windows the version below version 5 are .Net framework
2) .Net / .Net Core: it is cross platform, runs on all OS. Version 5 + are called .Net Core or .Net

* .Net framework

- framework: is a pre-buil set of tools, libraries that developers use to create application on windows or any os
- .Net framework is a software framework created by Microsot which uses C# & F# Lang
- F#: is functional programming language
- C#: is OOP language

#2 COMPILATION PROCESS.

1* .Net SDK

- .Net SDK (software development kit) includes all the nessary tools that are require to develop .Net applications ~~similarly~~ similar to JDK in Java
- .Net SDK Involves

1) Compiler: compiler to compile diff Lang (C#, F#) into intermediate langangue code, like csc (c sharp compiler)

2) CLR: Common language Runtime is a part of SDK which provides nessesary environment like JVM. CLR ~~contains~~ includes JIT (Just in time compiler) to compile the Intermediate code into machine code

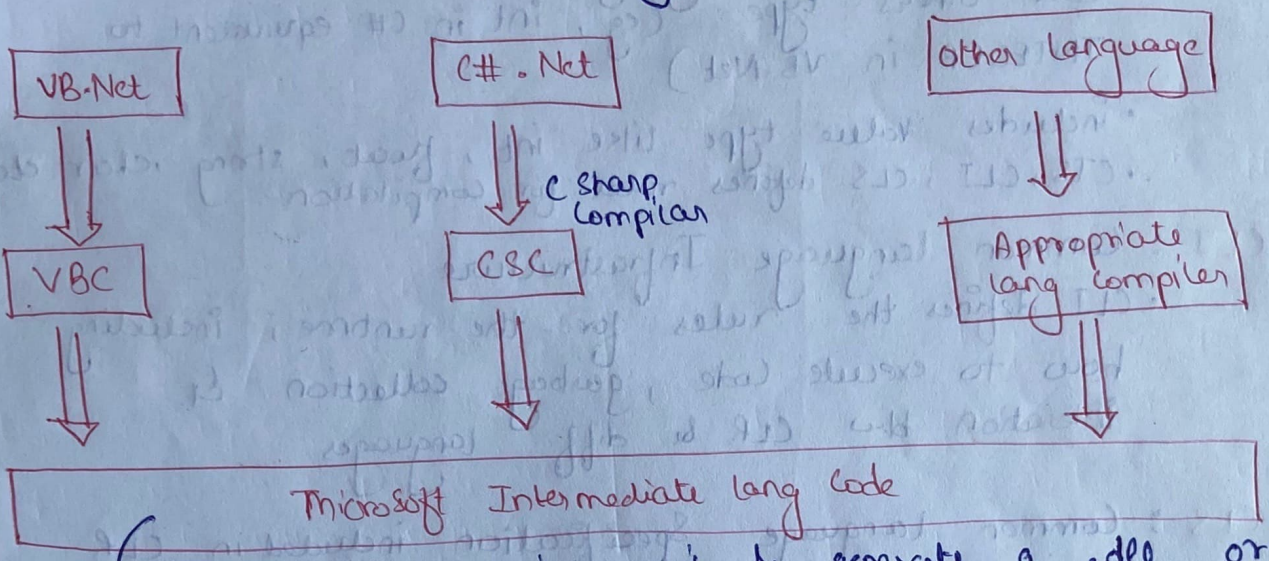
3) .Net CLI: .Net Command line Interface

4) BCL: SDK also includes Base class library that helps to write your codes

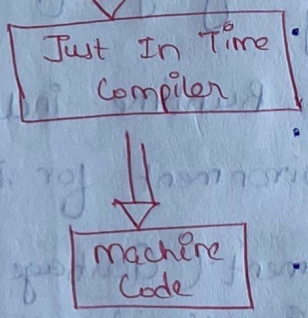
- you need to install .Net SDK to run your .Net application in C# or any Other language
- CLR: provides services like thread management, garbage collection type safety, etc
- ~~Working: dot Net application are written in C#, The~~

(2*) .Net framework Architecture

- Compilation process is done in two steps
  1) 1st phase is done by language compiler CSC
  2) second phase is done by JIT of CLR



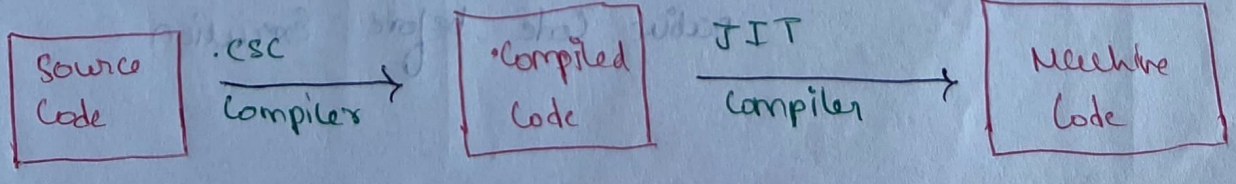
(Language compiler will compile & generate a .dll or .exe file which includes IL Code



- This compiles the Intermediate Code into Machine Code
- Just in time compiler is provided in dot Net runtime Environment (CLR)
- Becoz of IL Code lang independency & OS independency is there

(*3) Compilation Process

- dot net application is written in C#. This source code is complied into Intermidiate language code using C# Compiler
- The compiled code is stored in .DLL or .exe extention file
- When the .exe files runs, the CLR takes the IL code and using JIT compiler converts the compiled code into machine code



#4 TERMS

1) CTS :- Common type System included in CLR
- CTS defines how types are decalcared, used, & managed in .Net framework
- Ensures all .Net language can share & understand each others type (eg, int in C# equivalent to INTEGER in VB.Net)
- includes value type like int, float, string, char etc
- CTS, CLI, CLS defines rules for compilation

2) CLI : Common language Infrastructure
- CLI defines the rules for the runtime, including how to execute code, garbage collection & interation btw CLR & diff languages

3) CLS : Common language Specification included in CLR
- its a subset of CTS that defines Rules for language Interpolability
- Ensures that .Net language follows same rules like case sensitivity

4) CLR : Common language Runtime included in SDK
- The execution Environment for .Net that provides services like
  1) Memory Management (Garbage Collection)
  2) Type safety, Exception handling, Security
  3) JIT compiler, Thread Management

5) JIT : Just In time Compiler, converts IL code to MC

types :
- 1) Normal JIT : Compiles methods as they are called & store compiled code
- 2) Econo JIT : focuses on reducing memory usage by discarding compiled Machine code after execution
- 3) PRE JIT : Compiles the Entire code into native code before Execution

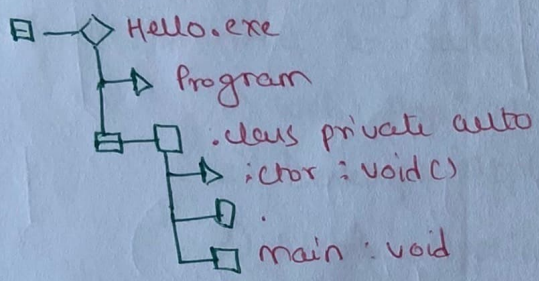
* How IL Code Looks

- open the .cs file in Terminal & do following code

```
csc filename.cs  // compiler of C# will compile & generate IL Code
filename.exe  // Run the IL code file you will get output
ildasm filename.exe  // by this command you can see the visual Representation of IL Code
```

// Stands for Intermediate language Diassembler

* IL Code Structure



* Types Of Application By C#

1) Desktop Application : using WinForm
2) Web Application : using Asp.Net Cor
   Asp.Net MVC
3) Mobile Application : using Xamarin
4) Game Development : using Unity
5) Cloud Based App : using Azure
6) Console Application : using Command Line tools

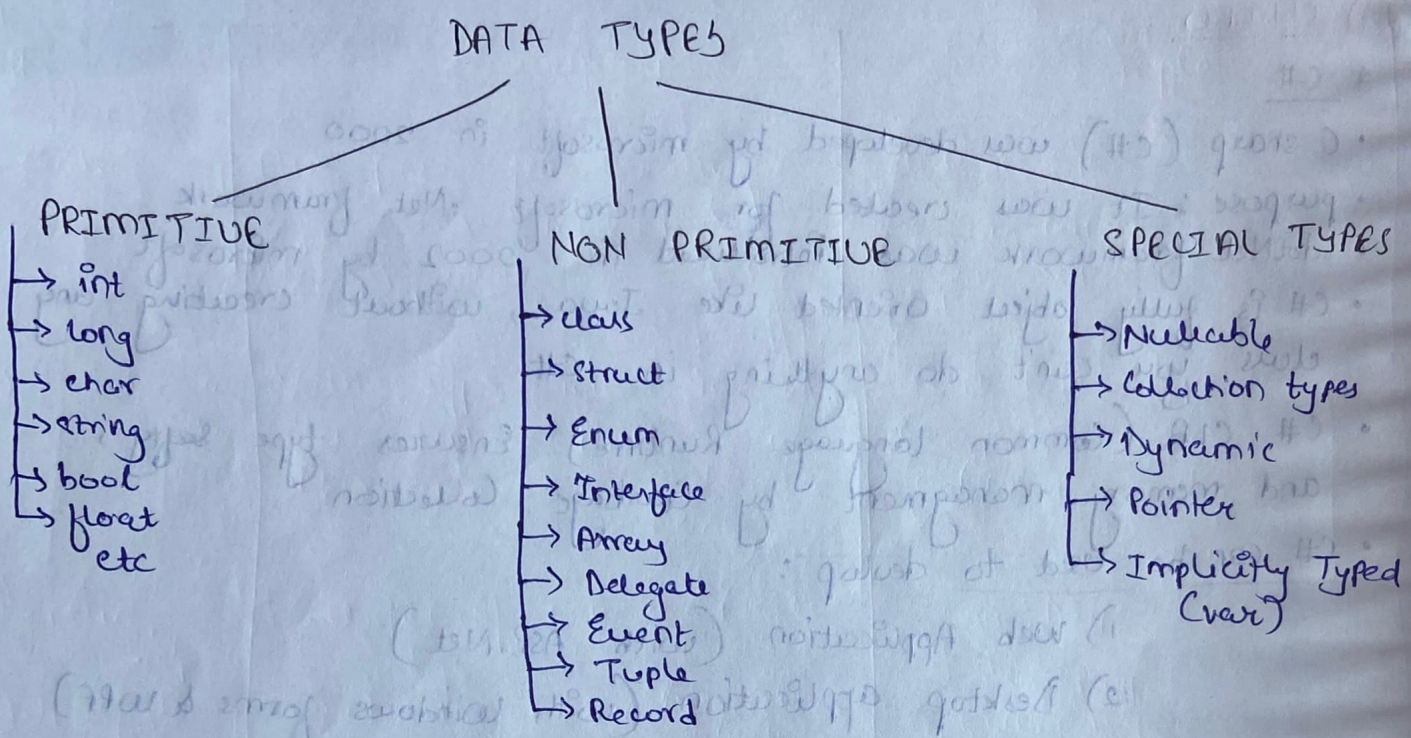
CH: 01 C# INTRO & DATA TYPES

#1 INTRO

* C#

- C sharp (C#) was developed by microsoft in 2000
- purpose: It was created for microsoft .Net framwork
- .dotNet framwork was introduced in 2002 by microsoft
- C# is fully object oriented like Java, without creating any class you can't do anything in C#
- C# CLR (Common language Runtime) Ensures type safty, and memory managment by Garbage Collection
- C# can be used to develop:
  1) Web Application (with ASP.Net)
  2) Desktop application (with Windows forms & WPF)
  3) Mobile Application (with .NET MAUI)
  4) Game Development (with Unity)
  5) Cloud Application (with Azure)
- C# file has extention of .cs
- C# complier compiles code to IL code csc file-name
- Assemblies: Assembly is a complied code library used for deployment, versioning & security
  - it is building block of .Net application
  - Assembly files have Extention as .DLL or .exe
- CLR is responsible for Memory Management, Loading Assemblies, JIT, Type Safety
- CLR has Garbage Collection which automatically releases unused memory in C#
- Assemblies: Is the executed code (IL code) with .dll or .exe extention
  - Assembly also consists of meta data about the IL Code)
- dll: Dynamic Link Library is the base class Libraries or like packages in Java
- .exe: is starting point of our ~~code~~ App which includes Main

#2 DATA TYPES



* SPECIAL TYPES

Data type	Description	Example
1) Nullable	Allows assigning null to any data type	int? age = null;
2) Collection Types	Data structures like List, Dictionary etc	List<int> n = new List<int>
3) Dynamic	Type is decided & checked at runtime instead of compile time	dynamic obj = 10;
4) pointer	Used for memory manipulation or to store address like in C++	int * ptr;

5) Implicitly Typed (var)	Automatically decides the type. Used in foreach loop to iterate over array collection string etc	var num = 10;
		foreach (var i in list)
		{
		//Code
		}

* Primitive types

Data type	Description	Example
1) byte	8 bit unsigned integer Range : 0 to 255	byte b = 255;
2) sbyte	8 bit signed integer Range : -128 to 128	sbyte b = 055; Sbyt c = -120;
3) short	16 bit signed integer -32,768 to 32,767	short s = -3000; short k = 27000;
4) ushort	16 bit unsigned integer 0 to 65,535	ushort us = 60,000; ushort ks = 75000; //error
5) int	4 byte signed integer -2,147,483,648 to 2,147,483,647	int i = 10000;
6) uint	4 byte unsigned integer	unit ui = 2000;
7) long	8 byte signed integer -9 quintillion to 9 quintillion	long l = -200000;
8) ulong	8 byte unsigned integer	ulong lb = 20000;
9) float	4 byte floating point no.	float f = 3.14;
10) double	8 byte floating point no.	double d = 3.14789;
11) decimal	16 byte high precision floating point no.	decimal m = 100.5m;
12) char	2 byte unicode character	char c = 'A';
13) bool	Boolean value true or false 1 ~~byte~~ bit	bool t = true;
14) string	Sequence of unicode character 2 byte for each character	string s = "vishu";
15) object	Base type for all types in C#	object obj = "Hello";

(*3) NON Primitive

type	Description	Example

1) class	Blue print for creating object with methods & properties	class Person
		{
		string name;
		int DOB;
		}

2) Struct	Value type similar to class but without inheritance	struct Point
		{
		int x;
		int y;
		}

3) Enum	Collection of named contacts	enum Days
		{
		Sunday,
		Monday
		}

4) Interface	Contract for class/structs to implement multiple inheritance	
5) Array	Collection of element of same type	int[] arr = new int[5]
6) Delegate	Type that holds refrence to methods	delegate void Print (string msg);
7) Event	Specialized delegate for publish-subscribe senarios	event Print onPrint;
8) Tuple	Lightweight struct grouping multiple values	var tuple = (1, "Hello")
9) Record	Immutable referance type used for representing data	record Person (string Name, int Age);

* Struct

- A struct is a value type that is often used for lightweigh objects such as coordinates or points
- eg:

```
using system;
Struct Point
{
    public int x;
    public int y;
    public Point (int x, int y)
    {
        X = x;
        Y = y;
    }
    public void Display()
    {
        Console.WriteLine ($"Point {x}, {y}");
    }
}
class Program
{
    Static void Main()
    {
        Point p = new point (1, 2);
        P.Display();
    }
}
```

* Enum

- defines a named set of constant values, often used for state, days etc

eg:

```
using system;
enum DaysOfWeek
{
    Sunday,
    Monday,
    .
    .
    Saturday.
}
class Program
{
    Static void Main()
    {
        DaysOfWeek today = DaysOfWeek.Monday;
        Console.WriteLine ($"Today is {today}");
    }
}
```

#3 TYPE CASTING

1. Rules

- Smaller types (byte, short) are promoted to int during arithmetic
- Mixing types promotes to the larger type
  int → float → double
- by default c++ takes floating point no. as double
  ∴ you need to add a "f" suffix in front of float to store in float
  eg: float b = 34.f;

byte + byte = int
int + double = double
double + float = double
float + int = float

2. Type Casting

1) Implicit Casting: Compiler Automatically do this

char → int → long → float → double

eg:
```
int a = 100;
long b = a; // Implicit cast from int to long

float c = 10.5f;
double d = c; // implicit cast from float to double
```

- small to larger types are automatically casted

NOTE
if you convert char to int or double then you will get ascii value of that char

2) Explicit Casting: Requires manual casting using parentheses (type)

- Necessary when converting b/w types that may result in data loss
- larger to smaller you have to manually do type casting

double → float → int

eg:
```
int x = (int) 3.5; // Explicitly converting float to int

double d = (double) 3.5;

int z = (int) d; // Explicitly converting double (d) into a int (z)
```

#3 Using functions Type Casting

- Using Inbuit functions also you can do type casting
- Syntax:

1) float varr = Convert.ToInt32 (3.55); // Convert 3.55 (double) value to float, float stores only integer in C#
2) Convert.ToDouble ( ); → // pass any data type it will get converted to double
3) Convert.ToString ( );
4) datatype.Parse (any other datatype); → // will return the passed data type val to datatype.Parse wala value

- by default the input taken from Console in C# is in string data type
- To Convert it in int you can parse it

- eg:

```
using system
class program
{
    static void Main()
    {
        1) float x = Convert.ToInt64(3.55);   // Convert double to float
           string input = Console.ReadLine(); // taking string as input
           int number = Convert.ToInt32(input);
           int xyz = int.Parse(input);        // parsing or converting into int
```

NOTE
TryParse is method used to safely convert string to int without throwing exceptions if conversion fails

OR

```
        2) int no = int.Parse(Console.ReadLine());
           // directly parse to int from Console only
```

OR

```
        3) string input = Console.ReadLine();
           if (int.TryParse(input, out int no))
           {
               cwl("you Enter" + no);
           }
           else{
               cwl("invalid input");
           }
    }
}
```

(4*) Basic Structure of C# Program

```
using System;          // A built in namespace in C# that contains essential classes such as Console, Math, DateTime etc
// key word to include diff namespaces like we do import in Java or include

namespace Hello
{
    // namespace is used to declare a namespace, which is a way to logically organize code and avoid naming conflicts
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("");
            ReadLine();
            string s = Console.ReadLine();
            Console.WriteLine(s);
        }
    }
}
```

- System namespace allows you to use "System.Console" in the program, so that you can call "Console.WriteLine()"
- Command line arguments are optional in main function
- namespace Hello का मतलब हम एक "Hello" namespace के अंदर डाल रहे हैं अपना लिखा हुआ सारा का सारा code
- मतलब हम एक file से दूसरे file में classes use कर सकते हैं namespace को use (import) करके जैसे packages होते हैं Java में

NOTE

- हम namespace के अंदर classes, struct etc डाल के दूसरे project file में उन्हें "using" keyword से import करके use कर सकते हैं
- मतलब Program class और Hello namespace के अंदर आ गई, हम उसे कहीं भी import कर सकते हैं using keyword से

namespace : "using System" इसमें "System" एक namespace है & इसके अंदर कई सारी classes हैं & "using System" लिखने से हम system namespace की classes use कर सकते हैं

- namespace एक container होता है जिसके अंदर classes & दूसरी namespaces हैं तो होती हैं
- namespace से code better organize होता है, एक तरह की classes को हम एक namespace में रखते हैं
- you can group similar classes in one namespace & use the classes into different files by creating their object, you just have to include that namespace using (using) keyword

#14 Literals

* Literals

- Literals are the fixed values given to variables
- eg int x = 100 // 100 is literal
- We can represent the Integer (numbers) in the form of Decimal (base 10), Binary (Base 2) or Hexadecimal (Base 16). for Binary & Hexadecimal literal we need suffix.
- There is no Octal Number literal in C#
- Hexadecimal : 0 to 9 digits are allowed
  a to f characters are allowed
  Prefix with 0x
  Sufix with b to a anything
- Binary literal : 0 to 1 digit is allowed
  Prefix with 0b

eg:
```
using system;
namespace LiteralDemo
{
    class Program
    {
        static void Main()
        {
            int a = 1029; // output = 1029

            int h = 0x123f; // output = 4671
            int hd = 0x123a; //Output = 4666

            int d = 0b1111; //output = 15
            int dn = 0b1010; //output = 10

            Cwl ($" Decimal Literal : {a}");
            Cwl ($" Hexa-decimal base 16 : {h}, {hd}");
            Cwl ($" Binary literal base 2 : {d}, {dn}");
        }
    }
}
```

CH : 02 String, Array & MATH

#1 Console Class

* Console Class

- To implement UI in Console Application, we have Console class, Console class is available in System namespace
- To work with Console. window (input or Output) we need Console class

* Console class properties

Console.Title : Gets/sets the title of Console Window

Console.BackGroundColor : sets the BG color of Console

Console.ForeGroundColor : sets the ForeGround color of text

Console.CursorSize : change size of cursor

* Console class functions

Console.Clear() : clears the Console Window

Console.Beep() : produces a Beep sound

Console.ResetColor() : Resets the FG & BG color to default

Console.ReadKey() : Reads a key & display information about that key like its Ascii value etc

eg:
```
using system;
namespace Program
{
    class Program
    {
        static void Main ()
        {
            Console.Title = "My Console App" // set title

            Console.BackGroundColor = Console.Color.DarkBlue;
            Console.Clear (); // set BG color

            Console.ForegroundColor = Console.Color.Yellow;

            Console.CursorSize = 50; // set cursor size
            Console.Clear (); // clears the window
            Console.Beep (); // play Beep sound

            Console.ResetColor ; // reset color to default

            Console.KeyInfo keyP = Console.ReadKey ();
            Console.WriteLine ($"{keyP.Key}, char : {keyP.KeyChar}");
                                  (gives ascii value)   (gives character)
        }
    }
}
```

NOTE

```
bool b1 = true;
bool b2 = false;        // Correct

bool B3 = True
bool B4 = 1
bool B5 = TRUE          // Error
```

#12 OPERATORS

Arithmetic = +, -, *, /, %

Relational = ==, !=, >, <, >=, <=

Logical = &&, ||, !

Bitwise = &, |, ^, ~, <<, >>
(AND, OR, XOR, Complement, Left shift, Right shift)

Unary = ++a, a++

Ternary Operator = condition ? trueExp : falseExp

eg: int max = (a>b) ? a : b;

* Goto statement

- Goto statement transfers the control to a labeled statement
- often used to jump to a diff part of code within same function
- Syntax :

```
goto label_anything_no_after_label
```

- you can also use it with switch case
- Syntax :

```
goto case 3;
```

- eg:

```
static void Main()
{
    int n=5;
    if (n==5)
    {
        goto Label101; // it will jump to ~~get~~ Label101 & start execution from label101:
    }
    Console.WriteLine("n is not 5, but it will skip because n=5");

    Label101:
        Console.WriteLine("jumped to label101");

}
```

* Break & Continue

- Break & Continue are same as C++

#13 Functions

* function

- its syntax is same as of c++ & java
- you can also do pass by ref and pass by value in c#
- By default parameters are passed by value
- you can use ref & out keywords to pass parameter as refrence (address passing)
- class, interface, delegate, arrays are passed by refrence by default
  modifying the object, array inside the fun will effect the original object (array, etc)

* ref

- The ref keyword is used to pass argument by refrence, this means change to that parameter inside fun will reflect in original variable
- The variable must be initialized befor passing, ~~and only value type~~

eg:
```
static void Main()
{
    int no = 20;
    cwl("no befor fun call" + no); // output = 20

    IncrementByRef(ref no);   // -> // actual parameter is no
    cwl("after fun call" + no); // output = 30
}
static void IncrementByRef(ref int x)   // -> x is formal parameter
{
    x += 10;
}
```

NOTE

Actual Parameter: Values or variable passed to method while fun. calling is called Actual parameters

formal Parameter : defined in function defination

* out keyword

- out is similar to ref but you have to assign value to that parameter before the fun returns
- out is also used to pass parameter as reference
- in out assigning value to variable during initialization is not necessary
- eg:

```
class Program
{
    static void GetSum&Product (int a, int b, out int sum)
    {
        sum = a+b; // Assigning value to sum which is reference variable
    }
    static void Main()
    {
        int x=5, y=10;
        int sum1;  // sum1 & sum are diff parameters but refer to same memory location
        GetSum&Product (x, y, out sum1);
        Console.WriteLine ($" sum is {sum1}");
                                  // output = 15
    }
}
```

* Command Line Arguments

- you can pass parameter to Main method, they are optional, as a string array you can pass parameter through command line Arguments
- you can pass command line Arguments to main fun
- Main method doesn't accept any parameter from any method, its the starting point of Execution
- Main method can accept n no. of parameter through command line at run time
- to pass command line parameter to Main fun :

properties → debug → Command line argument : value1 val2 val3

eg:

```
static void Main (string[] arg)
{
    CWL ($" first Command Line Argument : {arg[0]}");   // output
    CWL ($" Second argument : {arg[1]}");               // value1
}                                                       // val2
```

#4 String

* String

- string or String : The small string is alias of String.
- actual class name is String, but you can use any string or String
- while creating variables use small string and whenever you want to invoke methods on string use capital letters
- eg: string s = String.Concat(" ");
- String class is part of system namespace
- Strings are immutable in C#, meaning Once string is created it cannot be changed. Many method return new string

* String Methods

1) str.Length : this property returns the length
2) str.ToLower() : Returns a new string which has all lower characters
3) str.Trim() : Returns new string by removing all leading & trailing white spaces from string
4) str.Substring(st_index, end_index) : Returns a new string, by Extracting a substring from original string from starting to ending index

eg:
```
static void Main()
{
    string s = "  Hello World  ";
    string trimstr = s.Trim();
    string substr = trimstr.Substring(7, 5);
    Console.WriteLine(substr); // output : World
}
```

5) str.Replace ("string", "replacestring") : Replaces all occurance of specified string with another string

6) str.Split(',') : splits the string into an array of substrings based on specified delimiter.

7) str.IndexOf ("string") : finds the index of the first occurance of specified string in the string

8) str.Contains ("substring") : returns true if the sub string is present in the string

9) str.Join (",", array) : Joins elements of a collection like string.Join() an array into a single string with separator (,)
- Join is static method & returns new string

10) str.Compare (str1, str2) : Compares two string lexicographically & returns int 0, -1 or 1, 0 if both are Equal

eg:
```
static void Main()
{
    string str = "Hello World";
    string astr = str.Replace("World", "C#");

    string str2 = "apple,banana,orange";
    string[] fruits = str2.Split(',');
    foreach (var fruit in fruits)
    {
        Console.WriteLine(fruit);
    }

    int index = str2.IndexOf("banana");
    bool x = str2.Contains("banana");

    string[] words = {"apple", "GFAM", "banana", "orange"};
    string result = string.Join(",", words);

    string str111 = "apple";
    string str222 = "banana";
    int comparision = string.Compare(str111, str222);
    Console.WriteLine(comparision);
    //output: -ve value bcoz apple < banana lexicographically
}
```

* str.ToString ()

- it is used to convert any value to string
- but ToString should only be used when you're sure that the object is not null
- bcoz it throws NullRefrenceException if obj is Null

eg: static void Main ()

```
{
    int no = 123;
    int k = null;
    CWL (no.ToString ()); //output = "123"
    CWL (k.ToString ()); // throws Exception :: not null safe
}
```

* Convert.ToString (variable)

- it will convert the variable to string even if that variable or object is null
- eg: stalk void main ()

```
{
    string Name = Null;
    Convert.ToString (Name); // no Exception, no Error

    double dc = 45.67;
    CWL (Convert.ToString (dc)); //"45.67"
}
```

∴ always use Convert.ToString (variable) method for converting into string bcoz it is Null safe

#5 Math class

* Math class

- Math class is a part of system namespace that provides methods for performing mathametical operations
- if you pass int, float, long, or any other numeric data type in the Math class function, it is automatically converted to double

* Methods :

1. Math.Pow(a,b) : returns a power b
2. Math.Sqrt(a) : returns square root of a
3. Math.Cbrt(a) : return cube root of a
4. Math.Floor(a) : Rounds down 'a' to near int
5. Math.Round(a) : Returns round 'a' to nearest int
6. Math.Sin(a) : returns sin of a
7. Math.Cos(a) : returns cos of a
8. Math.ToRadians(360°) : Converts & returns degree to Radians
9. Math.ToDegrees(radians) : Converts & returns radians to degree
10. Math.log(a) : log base e of a
11. Math.log10(a) : log base 10 of a
11. Math.Abs(a) : returns absolute (+ve no. of a)
12. Math.Min(a,b) : returns minimum of a and b
13. Math.Max(a,b) : returns maximum of a and b
14. Math.Random() : generate and returns a random no.

eg:
```
static void Main()
{
    int num1 = -8.5;
    double num2 = 3.6;

    Cwl ($"Absolute is {Math.Abs(num1)}");
    Cwl ($"Rounded is {Math.Round(num2)}");
    Cwl ($"Max of both is {Math.Max(num1, num2)}");
    Cwl ($"Min of both is {Math.Min(num1, num2)}");

}
```

#16 Array

* 1D Array

- Syntax:

1) `int [] arr = new int [size];`
2) `int [] arr = new int [] {1,2,3,4,5};`
3) Accessing Elements directly using indices

```
arr[2] = 10;
arr[0] = 100; ] // assigns values
```

* 2D Array

- Syntax:

1) `int [,] arr = new int [3,3]`
   `int [,] matrix = new int [row_size, col_size];`
2) `int [,] matrix = new int [,] {`

```
{1,2,3},
{4,5,6},
{7,8,9}
};
```

3) Accessing Elements in 2D array using indices

```
Console.WriteLine (matrix [1,1]);
matrix [2,0] = 10; // assigns value at 2 row & 0th column
```

* Array fun & Properties

1) arr.length : Returns the length of array
2) arr.GetLength (diamension) : Returns size of array along a specific diamension
   1 = for colums
   0 = for rows

eg:

```
Main()
{
    int rows = matrix.GetLength (0) //returns no. of rows
    int col = matrix.GetLength (1) //returns no. of col.
}
```

3) Array.Sort (array) : Sort the array in ascending order

4) Array.Reverse (array) : Reverses the 1D array passed

5) Array.IndexOf (arr, ele) : Returns the index of passed element

eg: static void Main ()
{
```
int [] arr = new int[5];
CWL ("Enter elements");
for (int i=0; i<5; i++)
{
    int x = int.Parse (ReadLine ());
    arr [i]= x;
}

Array.Sort (arr); // sorts the array
Array.Reverse (arr); // reverses the array.

int ind = Array.IndexOf(arr, 100); // returns index of 100 if present
CWL ("index is" + ind);
```
}

* 2D Array Methods

1) matrix.length : returns total no. of elements (row x col)

2) matrix.GetLength (dia) : returns no. of elements in rows
if dia=0 & if dia=1 then no. of columns in each row

3) Array.Clear (matrix, 0, matrix.length) : clears all elements in 2D array, sets them to 0 for integers

4) Array.Sort (flatarray) : you can flatten the array to a 1D array, sort it & then reassign it to 2D array

eg: Main ()
{
```
int [] flatarray = matrix.Cast<int>().ToArray();
Array.Sort (flatarray)
int index = 0;
for (int i=0; i< matrix.GetLength (0) ; i++)
{
    for(int j=0 ; j<matrix.GetLength (1) ; j++)
    {
        matrix [i,j] = flatArray [ind++];
    }
}
```
}

5) Array.Reverse(flatarray) : similarly to sorting you can create a 1D array & then reverse it, & then reassign it to 2D array

* iterating

• syntax:

```
foreach (var i in matrix)
{
    cout (i); // output all element rows & col both
}
```

(#7) NOTE

* if else : same as Java & C++
* switch case : same as Java & C++
* Break, continue : same as C++
* loops : for
  do while
  while
  foreach
  parallel.forEach loop
* Comments : same as C++
* Collection framework : same as Java
* functions : same as Java, C++

CH : 03 DATA TYPES

#1 Data types

1* Nullable type

- Nullable type allow value type (like int, bool, float) to represent null values
- you have to use "?" in front of data type to store Null
- Syntax :

```
int ? a = null;
```

- Use case : Database interaction unallable is use full
- eg :

```
static void Main ()
{
    int? age = null;
    if (age.HasValue)          // returns true if that var has any value, or else false if that variable is null
    {
        Cwl ("age has value "+ (age.Value));
                                  // returns the value
    }
    else
    {
        Cwl ("Age is not specified");
    }

    int? no = null;
    no = 100; // assign 100 to no

    int? age1 = null;
    int defaultval = age1 ?? 100; // use 100 if age1 is null
    Cwl (defaultval); // output : 100

    int? nullableno = 50;
    int no = 20;

    int res = (nullableno ?? 0) + no;
    Cwl (res); // output = 70

    string? name = null;
    name ??= "Default Name"; // use default Name if name is Null
    Cwl (name); // output = Default Name
}
```

(?? is called null coalescing operator
this means if age1 is null then use 100, if its not null then it won't do anything

nullable is not null ∴ nullable will contain 50 value only

null coalescing assignment operator

(#2) var (Implicitly Typed)

- using var you can decalare variables with implicit typing
- When variable is decalare using var, the compiler desides the data type based on value assign to it at compile time
- you cannot decalare a var variable without assigning a value to it on right side
- Once a type is infered, it cannot be changed for var
- syntax : 1) `var no = 10; // Compiler infers type as int`
  `CWL(no.GetType()); // Output : System.Int32`

  2) `var list = new List<int>{1,2,3}; // Compiler infers type as List`
  `CWL(list.GetType()); //Output: System.Collections.Generic.List`1[System.Int32]`

person is type of object (anonymous object, means object without class)

3) `var person = new{Name="vishu", Age=22};`
`CWL($"Name : {person.Name}, Age : {person.Age}");`
`CWL($"{person.GetType()});`
`//output: System.RuntimeCompilerServices.AnonymousType`
`2[System.String, System.Int32]`
`// this creates an anonymous type with two properties Name & age. The compiler infers the type as anonymous class with these properties`

- **Key Points** : you cannot do following things :

1) `var no; // gives compiletime Error, variable must be initialized`

2) `var no = 10;`
`no = "vishu" // Error: cannot convert int to string beacz it is already infered`

3) `var no = 10;`
`no = 1000; // allowed, no Error`

4) `var no = 10;`
`no = "1000"; // Error: no is infered as int, you cannot change it to string`

> **NOTE**
> - The compiler analyzes the value on right hand side of assignment & determines the type at ~~runtime~~ compiletime for var

- modifying "var" variable value, with same type is allowed

- Once a type is inferred by Compiler at Compile time, it cannot be change when you declare variable with "var"
- The compiler analyzes the value on right side, & then determines the most suitable type at Compile time
- you must assign a value to variables declared using var
- eg:

```
void main()
{
    var no = "1000" //no is inferred as string
    no = int.Parse("2000"); //gives Error Compile time
    cwl(no);
}
```

output: Error
because int.Parse("2000") returns int, but "no" was already inferred as string type by Compiler.
since type of no is string, you cannot assign a diff type (int) to it later after initial assignment.
- you should always assign a string data only to no

eg:

```
void Main()
{
    var no = "1000";
    no = "2000"
    cwl(no) //output :"2000"
}
```

* Creating Anonymous Obj

- you can create a obj without a class also, it is called Anonymous object
- Syntax:

```
var obj_name = new { obj properties };
```

- eg: `var personobj = new {Name = "John", Age = 30, Job = "Developer"};`

```
personobj.Name //accessing Name & Age
personobj.Age
```

#8 Enum

- Enum is used to represent a collection of related values that are logically grouped together like days, weeks, Months etc
- Syntax:

```
enum Days
{
    Sunday, // By default 0
    Monday, // 1
    Tuesday, // 2
    .
    .
    Saturday // 6
}
```

Syntax:

```
enum Days
{
    Sunday = 10;
    Monday = 20;
    .
    .
    Saturday = 1000;
}
```

- you can define Enum inside class or fun also
- you can also assign custom values to Enum
- you should use Enum when you have fixed set of related constants like Months etc
- you can also use to verify that user has choosen a value you provided or diff value which you don't have
- Enum is a kind of user defined data type

* Casting Btw Enum & Underlying Data types

- Enum has an Underlying data type which is usually int by default. you can caste btw enum & the value
- eg:

```
enum Days
{
    Sunday = 10,
    Monday = 20,
    .
    .
    Saturday = 70,
}
```

// these are kind of properties of Days Enum

```
Days today = Days.Monday
int todayVal = (int) today; // casting Enum to int
cwl (todayVal); // 20 will pe output

int value = 30;
Days d = (Days) value; // casting int to enum
cwl (d) // output : Tuesday
```

* Methods of Enum

1) Enum.IsDefined (enumType, objectval) : returns true or false, if that value passed exists as a member in Enum then returns true
   - it helps in validating if a particular value corresponds to a defined Enum member

eg:
```
enum Days
{
    Sunday = 10,
    Monday = 20,
    .
    Saturday = 70
}
class Program
{
    static void Main ()
    {
        int dayVal = 30;
        if (Enum.IsDefined (typeof(Days), dayVal))
        {
            Days today = (Days) dayVal;
            Cwl ($" The day is : {today} ");
        }
        else
        {
            Cwl (" Invalid value for Days enum");
        }
    }
}
```

2) Enum.GetValues (typeof(enumname)) : Retrives all values from an Enum
   - used for traversal, as Returns all ~~int values~~ string names in Enum

3) Enum.GetName (typeof(Enumname), value) : Retrives the name of an enum by its value passed

4) Enum.Parse (typeof(enumname), otherdatatype) : Converts a string to its corresponding enum no.

eg:
```
void Main ()
{
    foreach (Days d in Enum.GetValues (typeof(Days)))
    {
        Cwl (d)
    }
    int value = 40
    string dayName = Enum.GetName (typeof(Day), value);
    Cwl (dayName); // Wednesday
}
```

(#4) OBJECT Type

- OBJECT type is parent of all data types value & reference Data types in C#
- object is a type like we have int, char, bool etc
- you can define variables using Object type
- Object is the base type from which all types (refrence & ~~value type~~ value type) implicitly inherit, it is defined in the system namespace as system.Object. • object stores on heap :. refrence type
- any data type can be assigned to object type variable
- you can use system.Object or Object, its same only
- when a variable is decalare using object keyword, its type is always System.Object
- The ~~runtime~~ compile time keeps track of the actual ~~value~~ type of value stored in object variable.

• object will boxed value type & store them on heap memory & as refrence type

- syntax :

```
1) object myobj = 42;
   Cwl (myobj) //output : 42
   Cwl (myobj.GetType ()) // will print output : system.Int32
                          beoz myObj is storing int value
```

• Although "struct" is a value type, it can still be boxed & treated as object
• Boxing happens automatically when you store a instance of struct as object

```
2) object a = 3.14;
   Cwl (a.GetType ()) //output : system.double
```

- **Local object variable** : you cannot decalare a variable of type object without assigning a value at time of decalaration
- **class-level variables** : you can decalare variables inside class. without assigning them a value, beoz by default they will have null value

eg:
```
void Main ()
{
    void SomeMethod ()
    {
        object obj ; // gives Error
    }
    class MyClass
    {
        object obj ; // automatically null is assigned
    }
}
```

```
object obj = null;
obj = 1;
obj = true;
obj = "vishu"
```
//valid Code

you can change the type of object like dynamic at any time

• for object, no need to assign value at time of initializing like we do for var

* Value Types & Refrence Types

- In c# there are Mainly two types of DataTypes value & Refrence

- **Value Types**: when these data type variables are parsed inside fun as a parameter then their copy of value is parsed to the fun
  - Value types are typically stored in stack if decalared under function or local variables
  - if a value type (int) is a field of a refrence type (class) it becomes part of class instance & class instance are stored in heap

eg:
```
void Main
{
    int x=10; //x is stored in stack
}
class MyClass
{
    int y; // y is stored in heap bcoz class is refrence type & y is part of class
}
```

- **Example**. eg of value types :- int, float, char etc
  - Struct types = DateTime, TimeSpan, Guid, structs
  - enum = enum color{Red, Green}
  - Nullable values = int?, char?
  - all premitive data types

- **Refrence type**: when these data types variables are parsed inside fun as parameter then their address is parsed & if we made change to that refrence then actual object (variable) is changed
  - This means the data is stored on heap
  - Refrence types are stored on heap
  - eg: class, Array, string, interface, Delegate, object, dynamic.

* Boxing & UnBoxing

- **Boxing**: if you want to convert value types variables to Refrence (object) type, then this proces is called Boxing
  - Boxing is done automatically done, implicity value type variable is converted to object
  - Boxing involves allocation memory on heap for new object
  - eg:

```
static void Main()
{
    int a = 10; // value type
    object b = a; // value type converted to refrence type this is called boxing
    Cwl(b);       // it is automatically converted
}
```

- **UnBoxing**: if you want to convert reference (object) type variables (class, string) to value type, then this proces is called UnBoxing
  - UnBoxing involves casting, which requires a type check at runtime
  - UnBoxing is Explicitly done by us using Convert Methods
  - eg:

```
static void Main()
{
    object b = "100";
    double d = (double) b; // UnBoxing
}
```

Boxing = value type → (to) object type

UnBoxing = object type → (to) value type

- Unboxing: converting object type to value type
  - Unboxing applies only on value type.
  - you cannot unbox a object of a class, bcoz object/instance of a class cannot be stored on stack memory

(*5) Const, readonly

- **Const** : stands for constant, it must be initialized at time of decalaration, class objects are not allowed with const
  - can hold only primitive types (value types) & string
  - Cannot be modified after initialization
  - Static by default (implicitly shared across all instance of class)
  - Immutable (cannot be changed once assign)
  - syntax : `const int a = 100;`
- **readonly** : readonly field are assigned at runtime or during constructor execution
  - value can be initialized either at decalaration or in a Constructor of object
  - the value once set, cannot be modified
  - Immutable after initialization
  - not static, instance level, can store both (value & reference) type
  - syntax : `readonly int variable;`
- eg:

```
public class Example
{
    public const int value=100; // by default static beoz of const
    public readonly int x;      // non static
    public Example (int val)
    {
        x = val;
    }
}
Example obj1 = new Example (10);
Example obj2 = new Example (1000);

~~cwl (obj1.value);~~
~~cwl (obj2.value);~~

Output :
cwl (Example.value); // 100
                     // value is static beoz decalare using constant
cwl (obj1.x); // 10
cwl (obj2.x); // 1000
              // x is non static beoz decalare using readonly
```

(*6) Dynamic

- Dynamic is a type which is resolved at runtime
- Dynamic is used for late Binding
- you can also change the type of data storing in dynamic
- Dynamic is similar to var, but the type is checked at run time & you can also assign a diff type value to Dynamic but in var you can't
- The variable which are declare using Dynamic can change a type at runtime
- There is no need to initialize (assign value) to dynamic initially
- eg:

```
dynamic x; // you can decalare dynamic without assigning value
dynamic y = 10; ] // you can change the type of dynamic
y = "Hello string" ]
Console.WriteLine (y); // Hello string

x = 13.5;
Console.WriteLine(x);
```

- Syntax: dynamic variable = "value"

Property	Dynamic	Var	Object
Type Resolution	Runtime	Compile	Compile
Type Safty	No (Error caught at runtime)	Yes	Yes
Implicit decalaration	No	Yes	No/Yes
Casting requirment	No need for type casting	Not applicable	Requires explicit casting
flexiblity	very flexible can hold any type and even type can be changed	inferred from assign value मतलब एक बार compiler ने type infered कर लिया तो उसी type की value store करनी पड़ेगी	Can hold any type but less flexible, but type can be changed at any time
~~Performance~~			

#7 Tuples

- Tuples allow you to store multiple values of different types in a single object
- these are Type of System.Tuple only when you print Tuple.GetType()
- Tuples are Immutable, their values cannot be changed after creation
- you can use tuples for returning multiple values from method
- syntax:

1) `var myTup = (1, "Hello", true); // without mentioning type`
2) `(int, string, bool) myTup = (1, "Hello", true) // you can also specify types explicitly`
3) `(int age, string name) person = (30, "vishu");`
   `↳ // Tuple with named elements`
4) `var myTup = (Id:1, Name: "vishu");`
   `↳ // you can also write named elements here directly`

eg:
```
var myPerson = (name: "vishu", age: 21);
~~cwl (tuple.Item1);~~
~~cwl (tuple.Item2);~~
cwl (myPerson.Item1); // output: vishu
cwl (myPerson.Item2); // output: 21
cwl (myPerson.name); // output: vishu
```

eg 2:
```
public (int, string) GetPersonInfo()
{
    return (25, "Bob");
}
var personInfo = GetPersonInfo();
cwl(personInfo.Item1);
```

(8*) Convert.Int32, Parse(), tryParse()

1) Convert.Int32()

- Converts a value (eg string, object, etc) into int
- handles null gracefully (returns 0 if input is null)
- Throws Exception if the conversion fails
- eg:

```
static void Main()
{
    string input = "123";
    int res = Convert.ToInt32(input);

    string invalid = "abc";
    try
    {
        int res2 = Convert.ToInt32(invalid);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

// throws Exception because can't convert abc to int

2) datatype.Parse(input)

- Convert a string to a numeric type eg int.Parse(string), double.Parse(string)
- Does not handle null (throws Exception) if value passed is null
- Throws Exception if the input is invalid or too large
- eg:

```
static void Main()
{
    string n = "123";
    int res = int.Parse(n);

    string invalid = "abc";
    try
    {
        int res2 = int.Parse(invalid); // throws Exception
    }
    Catch
    {
        //same code as above
    }
}
```

3) TryParse ()

- same as Parse () but don't throw any exception
- Returns a bool indicating wether the conversion was successful or not
- if conversion ~~is~~ fails, the output is set to 0 or default
- syntax:

```
bool a = int.TryParse (string input , out int res)
```

  - give the input string which we need to convert in int
  - res is the output or the converted value

- eg:

```
static void Main ()
{
    string in = "123"
    if (int.TryParse (in , out int res))
    {
        Cwl ("Conversion Succesed" + res );
    }
    else
    {
        Cwl ("Conversion fails");
    }
}
```

*9 Diffrences

1) object Vs dynamic

feature	object	dynamic
Type Checking	At compile time	At runtime
Casting required	Yes	No
Erros	Detected at compile time	Dectected at Runtime
Type Safty	strong	weak
Use Cases	Gneral purpose	Dynamic, unknown structures

*10 Anonymous Obj

- you can create a lightweight obj without creating class
- its type will be Anonymous Type
- you can simply use new keyword & write properties in { } bracks to create anonymous object

- syntax:
  object obj = new { first = "vishu", last = "Gaikwad" };

- eg:
```
static void Main()
{
    object obj = new { f_name = "vishu", l_name = "Gaikwad" };
    Console.WriteLine($" obj is : {obj} "); //will print whole obj
    Console.WriteLine($" obj type : {obj.GetType()}");
                                  //output : Anonymous Object
    Console.WriteLine($" name is : {obj.f_name}");
}
```