

* Console.Writeline() - give output and jumps to new line
* Console.Write() - gives output but doesn’t jump to new line
* || - or statement
* && - and statement
* !<str> - is not statement
* ==,!= - equal to, not equal to
* <arg>++ or <arg>-- - returns the value then adds one to it or subtracts one
* ++<arg> or --<arg> - adds or subtracts one and then returns the value
* Do-while loops - Operates first checks condition later

do{

}

While(<condition>)

* void method – can't return any value
* Comments - // for single line and /\* \*/ for multi line
* switch statement – similar to if-else statement

default is used to give function for values with no defined functionality under case\_\_\_:

switch( )

case \_\_\_\_ :

{ func }

default :

{ func }

* <data\_type> [ ] ; - creates a 1D array
* An array’s dimension depends on the number of spaces in the square brackets

2D array example – int [ , ] = new int[3,4];

* Value inside the square bracket define the size of the array and can be left empty
* Jagged array is an array whose elements are also arrays (nested array)

int[ ][ ] jagged = new int[ ][ ];

* We can omit the size decleration and the new operator when the number of elements are provided in curly brackets

string[] name = {“asdad”,”aned”,”wrgfa”};

* <array>.Length – returns number of elements in the array
* <array>.Rank – returns the number of dimensions in the array
* <array>.Max – returns the largest value
* <array>.Min – returns the smallest value
* <array>.Sum – returns the sum of all elements
* Array.Reverse(<array>); - formats array in reverse order
* Array.Sort(<array>); - formats array in increasing order
* <var>.Length – returns the length of string
* <var>.IndexOf(value) – returns index of given value
* <var>.Insert(index , value) – inserts value at given index
* <var>.Remove(index) – removes the value at given index
* <var>.Substring(index , length) – returns susbstring of specified length starting from given index
* <var>.Replace(old\_value , new\_value) – replaces the value
* <var>.Contains(value) – checks if the value is present in the string
* String.Concat(str1 , str2); - combines the two strings
* String.Equals(str1 , str2); - True of both are same
* Inheritance allows a class to inherit another class’s functionality

Class xyz:abc { } -allows xyz to inherit abc’s functionality

* Class types -
  + Protected – can only be accessed buy a class and it’s sub classes
  + Private – value cant be accessed outside the class
  + Public – makes the value available outside the class
  + Sealed – prevents other classes from inheriting a class or its members
* Abstract method can be called inside abstract classes only
* An abstract class is intended to be a base class of other classes. It acts like a template for its derived classes. Now having a the abstract class we can derive the other classes and define their own methods
* Members marked as abstract oare included in an abstract and must be implemented by classes that derive form abstract classes
* An Interface is a completely abstract class which contains only abstract members.
* All members of the interface are abstract by default, so no need to use the abstract keyword.
* override keyword is not needed when you implement an interface
* A class can inherit from just one base class but it can implement multiple interfaces
* Implementing multiple lists

class A : Ishape, Iab, Iac etc.{ }

* <condition>?<expression\_1>:<expression\_2>;

expression\_1 gets executed if the condition holds true or else expression\_2 is executed.

Is if-else statement in a way

* for loop -

for(<define variable>,<set condition>,<set function>){ }

the spaces can be left empty to create an infinite loop

* for each loop iterates the entire array/list and goes through each value separately
* continue statement skips the remaining statement of that iteration of the loop
* break statement breaks out of the loop
* Optional argument – incase a user/system input for a variable is not given a default parameter is taken

Optional argument should always be at the end of the list

Example - static int alpha(int x , int y =2) { } – is correct but Static int alpha(int x , int y =2) { } is not correct

* Virtual statement allows a method to be overwritten
* Override statement overrides the virtual method
* Polymorphism means that a call to a member method will cause a different implementation to be executed depending on the type of object that invokes the method
* Ways of building strings in C#
* string str;

str = “Username” + username + “,Email:” + email;

* string str;

str = string.format(“username:{0},Email:{1}”,username,email);

* string str;

str = $“Username:{username},Email:{email}”; string interpolation

* When calling the method one can use the parameter anmes to provide the arguments in any order you like

static int Area(int w , int h)

{

return w\*h;

}

static void Main(string[] args)

{

int res = Area(h:3 ,w:3);

Console.WriteLine(res);

}

* Benefits of encapsulation –
* Controls the way data is accessed or modified
* Code is more flexible and easier to change with new requirements
* Change one part without affecting other parts of code
* The ref keyword passes the memory address to the method parameter which allows the method to operate an actual variable

static void Sqr(ref int x)

{

x=x\*x;

}

static void Main(string[] args)

{

int a=3;

Sqr(ref a);

Console.WriteLine(a); //Outputs:9

}

* Unlike ref parameter where value was referred to the method out parameters get their value from the method
* Difference between const and readonly fields
* constant field must be initialized when it is declared whereas a readonly field can be declared without initialization

readonly string a; //OK

const string b; //Error

* readonly field value can be changed in a constructor, but a constant value cannot.
* readonly field can be assigned a value that is a result of a calculation, but constants cannot

readonly double a = Math.Sin(60); // OK

const double b = Math.Sin(60); //Error

* Operator overloading allows operators to have a custom action ex- you can redefine the action of plus(+) operator in a custom class.

class Box

{

public int Height {get; set; }

public int Width {get; set; }

public Box (int h, int w)

{

Height = h;

Width = w;

}

public static Box operator+(Box a , Box b)

{

int h = a.Height + b.Height;

int w = a.Width + b.Width;

Box res = new Box(h,w);

return res;

}

}

static void Main(string[ ] args)

{

Box b1 = new Box(14,3);

Box b2 = new Box(5,7);

Box b3 = b1 + b2; #b3 height=19 and width=10

}

* Class constructor is a special member method executed whenever a new object of a class is created

class Program

{

private int age;

public Person() 🡪 constructor(same name as class)

{

Console.WriteLine(“hi”);

}

}

Person p = new Person() //output = “hi”

* \ - escape character can be used to output symbols such as “ “ .

Console.Write(“”hi””); // hi

Console.Write(“\”hi”\”) //”hi”

* \n – creates a newline
* \t – adds a tab
* Verbatim string allows use of special characters and line breaks in a string
* Struct is a value type used to encapsulate groups of related data
* Unlike classes, structs can be instantiated without using “new” operator

for struct – Book b; for class – Book a = new Book;

* structs don’t support inheritance and cant contain virtual methods
* Enum (enumeration) a type that consists of a set of named constant
* By default, first enumeration has a value of 0 and value of each successive enumerator is increased by 1

enum alpha{a,b,c,d}; //values=0,1,2,3

* The enumerator value is one increment of previous value and enumerator can have custom int values

enum Alpha = {a,b,c=9,d}; //values=0,1,9,10

* Working with files-

using System.IO

* File.WriteAllText(<filename>,<var>); - writes data of the given variable to the given file, if the file doesn’t exist a new one is created
* File.ReadAllText(<filename>);
* File.AppendAllText( ); - appends text to EOF
* File.Create( ) – creates a file in specified location
* File.Delete( ) – deletes the specified file
* File.Exists( ) – checks whether the specified file exists
* File.Copy( ) – copies the file to a new location
* File.Move( ) – moves the file to a new location
* The files are automatically closed after the operation is performed