**Printing**

*Console.WriteLine("Hello World"); Console.WriteLine($"{num}");*

**Variables**

*string characterName = “John”;*

*int characterAge = 35;*

*Console.WriteLine("There was a man named " +characterName);*

*char grade = ‘A’;*

*float, double, decimal*

*bool isMale = true;*

**Strings**

*string phrase = “Academy”*

*phrase.Length;*

*phrase.ToUpper();*

*phrase.Contains(“Academy”); ->True*

*phrase[0] -> A*

*phrase.Indexof(“Academy”); -> 0*

*phrase.Substring(0, 3); -> Aca*

**Userinput**

*Console.Write(“Enter your name: ”); //Doesn’t go to newline*

*string name = Console.ReadLine();*

*Console.WriteLine(“Hello, ” + name);*

**String to Integer/Double**

*int num = Convert.ToInt32("45");*

*double num = Convert.ToDouble("45.12")*

**Arrays**

*int[] LuckyNumbers = {4, 8, 15, 16, 23, 42};*

*LuckyNumbers[0]; 🡪 4*

*LuckyNumbers[1] = 7;*

*LuckyNumbers.length();*

*string[] friends = new string[5]; friends[0] = “Jim”;*

**Methods**

*static void SayHi(string name){*

*Console.Write(“Hello, ” + name);*

*}*

**If Statements**

*bool isMale = true;*

*bool isTall = true;*

*if(isMale && isTall){*

*Console.WriteLine(“You are a tall male”);*

*}*

*else if(isMale || isTall){*

*Console.WriteLine(“You are either not male or not tall”);*

*}*

*else{*

*}*

**Switch Statements**

*string dayName;*

*int dayNum;*

*switch(dayName){*

*case 0:*

*dayName = “Sunday”;*

*break;*

*case 1:*

*dayName = “Monday”;*

*break;*

*default:*

*dayName = “Invalid Day”;*

*break;*

*}*

**While Loops C# Quick Review**

*int index = 1;*

*while(index <= 5){*

*Console.WriteLine(index);*

*index++;*

*}*

*do{*

*Console.WriteLine(index);*

*index++;*

*}while(index <= 5);*

**For Loops**

*int[] luckyNumbers = {4, 8, 15, 16, 23, 42};*

*for(int i = 0; i < luckyNumbers.length(); i++){*

*Console.WriteLine(luckyNumbers[i]);*

*}*

**2d Arrays**

*int[,] numberGrid = {*

*{1, 2},*

*{3, 4},*

*{5, 6}*

*};*

*Console.WriteLine(numberGrid[1][1]); 🡪 4*

*int[,] myArray = new int[2,3]; //2 rows, 3 columns*

*int[,,] array1 = new int[4, 2, 3];//D1 4, D2 2, D3 3*

**Exception Handling**

*try{*

*Console.WriteLine(“Enter a number: ”);*

*int num1 = Convert.ToInt32(Console.ReadLine());*

*Console.WriteLine(“Enter another number: ”);*

*int num2 = Convert.ToInt32(Console.ReadLine());*

*Console.WriteLine(num1/num2);*

*}*

*catch(DivideByZeroException e){*

*Console.WriteLine(e.Message);*

*}*

*catch(FormatException){*

*Console.WriteLine(e.Message);*

*}*

*catch(Exception e){*

*Console.WriteLine(e.Message);*

*}*

*//Executed no matter what*

*finally{*

*}*

**Classes and Objects**

*class file, Books.cs*

*class Book{*

*public string title;*

*public string author;*

*public int pages;*

*}*

*Book book1 = new Book();*

*book1.title = “Harry Potter”;*

*book1.author = “JK Rowling”;*

*book1.pages = 400;*

*Book book2 = new Book();*

*book2.title = “Lord of the Rings”;*

*book2.author = “Tolkein”;*

*book2.pages = 700;*

**Constructors**

*class Book{*

*public string title;*

*public string author;*

*public int pages;*

*public Book(){*

*}*

*public Book(string aTitle, string aAuthor, int aPage){*

*title = aTitle;*

*author = aAuthor;*

*pages = aPage;*

*}*

*}*

*Book book1 = new Book(“Harry Potter”, “JK Rowling”, 400);*

*Book book2 = new Book(“Lord of the Rings”, “Tolkein”, 700);*

*Book book3 = new Book();*

**Object Methods**

*class student{*

*public string name;*

*public string major;*

*public double gpa;*

*public Student(string aName, string aMajor, double aGpa){*

*name = aName;*

*major = aMajor;*

*gpa = aGpa;*

*}*

*public bool HasHonors(){*

*if(gpa > 3.5){*

*return true;*

*}*

*return false;*

*}*

*}*

*student student1 = new Student(“Jim”, “Business”, 2.8);*

*student student2 = new Student(“Pam”, “Art”, 3.6);*

*student1.HasHonors();*

**Getters and Setters**

*class Movie{*

*public string title {get;set;}*

*public string director {get;set;}*

*private string rating {get;set;}*

*public Movie(string aTitle, string aDirector, string aRating){*

*title =aTitle;*

*director=aDirector;*

*Rating=aRating;*

*}*

*public string Rating(){*

*get{return rating;}*

*set{*

*if(value == “G” || value == “PG” || value == “PG-13” || value == “R” || value == “NR”){*

*rating = value;*

*}*

*else{*

*rating = “NR”;*

*}*

*}*

*}*

*}*

*Movie avengers = new Movie("The Avengers", "Joss Whedon", “PG-13”);*

*Movie avengers = new Movie("Shrek", "Adam Adamson", “PG”);*

**Static Class Attribute**

*class Song{*

*public string title {get;set;}*

*public string artist {get;set;}*

*public int duration {get;set;}*

*public static int songCount = 0;*

*public Song(string aTitle, string aArtist, int aDuration){*

*title = aTitle;*

*artist = aArtist;*

*duration = aDuration;*

*songCount++;*

*}*

*public int getSongCount(){*

*return songCount;*

*}*

*}*

*Song.songCount;*

**Static Methods & Classes**

*static class UsefulTools{*

*public static void SayHi(string name){*

*Console.WriteLine(“Hello ” + name);*

*}*

*}*

*UsefulTools.SayHi(“K”);*

*UsefulTools tools = new UsefulTools();*

*//static makes it so you can’t declare a Usefultool*

**Inheritance**

*class chef{*

*public void MakeChicken(){*

*Console.WriteLine(“Chef makes chicken”);*

*}*

*public void MakeSalad(){*

*Console.WriteLine(“Chef makes salad”);*

*}*

*//Subclasses can overwrite this method*

*public virtual void MakeSpecialDish(){*

*Console.WriteLine(“Chef makes bbq ribs”);*

*}*

*}*

*class Program{*

*static void Main(string[] args){*

*Chef chef = new chef();*

*chef. MakeSpecialDish();*

*ItalianChef italianChef = new ItalianChef();*

*italianChef. MakeSpecialDish();*

*Console.ReadLine();*

*}*

*}*

*//Italian chef inherits all the functionality of the chef*

*class ItalianChef : Chef{*

*//Overwrites*

*public override void MakeSpecialDish (){*

*Console.Write(“Chef makes pasta”);*

*}*

*}*

**List**

*using Systems.Collections.Generic;*

*string[] food = new string[3]; 🡨 array*

*food[0] = “pizza”;*

*food[1] = “ham”;*

*food[2] = “hotdog”;*

*foreach(String item in food){*

*Console.WriteLine(item);*

*}*

*List<String> food = new List<String>();*

*food.Add(“pizza”);*

*food.Add(“ham”);*

*food.Add(“hotdog”);*

*food.Add(“fries”);*

*food.Remove(“fries”);*

*food.Insert(0, “sushi”);*

*food.Count;*

*food.indexOf(“pizza”);*

*food.LastIndexOf(“fries”);*

*food.Contains(“pizza”); 🡪 true*

*food.Sort(); //sorts alphabetically*

*food.Reverse();*

*food.Clear();*

*public static List<string> optionList = new List<string>*

*{"hello", "world"};*

*string[] foodArray = food.ToArray();*

**Function to Return a List**

*static List<int> function(){*

*List<int> listRange = new List<int>();*

*return listRange;*

*}*

**Initialize List with Elements**

*List<string> mylist = new List<string>(new string[] { "element1", "element2", "element3" });*

**List of List**

*List<List<string>> myList = new List<List<string>>();*

*myList.Add(new List<string> { "a", "b" });*

*myList.Add(new List<string> { "c", "d", "e" });*

*myList.Add(new List<string> { "qwerty", "asdf", "zxcv" });*

*myList.Add(new List<string> { "a", "b" });*

*myList.AddRange(list2); //append another list*

*// To iterate over it.*

*foreach (List<string> subList in myList)*

*{*

*foreach (string item in subList)*

*{*

*Console.WriteLine(item);*

*}*

*}*

*List<List<String>> strList = new List<List<String>>{*

*new List<String> {String.Empty, String.Empty},*

*new List<String> {String.Empty, String.Empty},*

*new List<String> {String.Empty, String.Empty},*

*new List<String> {String.Empty, String.Empty},*

*};*

**Input Parsing**

*string line = Console.ReadLine();  
int a = int.Parse(line);  
line = Console.ReadLine();  
int b = int.Parse(line);*

*Console.Write("Enter the Number: ");     
num = int.Parse(Console.ReadLine());*

**Class Inheritance, Overloading & Overriding**

**Overloading** – method with same names and different signatures

**Overriding** – using a virtual keyword and overriding in child class

encapsulation, abstraction, polymorphism, inheritance

public class Employee{

public string name {get;set;}

public string address {get;set;}

public virtual void Validate(){

CheckName();

CheckAddress();

}

private void CheckName()

private void CheckAdress()

}

public class Manager : Employee{

public void Management()

public override void Validate()

public void Validate(bool strict)

{

}

public void Validate(bool strict, int num1)

{

}

}

**String Sorting**

*using System;*

*using System.Collections.Generic;*

*using System.Linq;*

*static string SortString(string input)*

*{*

*char[] characters = input.ToArray();*

*Array.Sort(characters);*

*return new string(characters);*

*}*

*.Trim() //To remove spaces*

**Public**

The type or member can be accessed by any other code in the same assembly or another assembly that references it.

**Private**

The type or member can only be accessed by code in the same class or struct.

**Protected**

The type or member can only be accessed by code in the same class or struct, or in a derived class.

**Private Protected** (added in C# 7.2)

The type or member can only be accessed by code in the same class or struct, or in a derived class from the same assembly, but not from another assembly.

**Internal**

The type or member can be accessed by any code in the same assembly, but not from another assembly.

**Protected internal**

The type or member can be accessed by any code in the same assembly, or by any derived class in another assembly.

Stack and heap are memory types in an application. Stack memory stores data types like int, double, boolean etc. While heap stores data types like string and objects.

Both the **Thread class** and the **Task class** are used for parallel programming in C#. **A Thread is a lower-level implementation while a Task is a higher-level implementation**. It takes resources while a Task does not. Thread tries to stick to a processor.

Tasks utilizes your hardware properly

Threads have a cpu affinity

In case of tasks, I can return a result, can cancel, can change it, can wait for it to finish

**Boxing**- is a process of converting a value type to an object type where value type is placed on the stack memory, and the object type is placed in the heap memory. This conversion is an implicit conversion and you can directly assign any value to an object, and C# will handle the rest of the conversion on its own.

Int a=111;  
Object b=a;

**Unboxing**- it is the reverse process of the boxing process. It is a conversion of the object type to the value type and the value of the boxed object type placed on the heap memory which will be transferred to the value type which is placed on the stack. This conversion of the unboxing process has to be done explicitly.

Object b=111;  
Int a=(int)b;

**Structure**

A structure is a data type of a value type. A struct keyword is used when you are going to define a structure. A structure represents a record and this record can have many attributes that define the structure.

Although both class and structure are user-defined data types, they are different in several fundamental ways. A class is a reference type and stores on the heap. Struct, on the other hand, is a value type and is, therefore, stored on the stack. While the structure doesn’t support inheritance and polymorphism, the class provides support for both.

**Async and Await**

Processes belonging to asynchronous programming run independently of the main or other processes. In C#, using Async and Await keywords for creating asynchronous methods.

Asynchronous code does not use threads

Asynchronous – making application usable, non-blocking main thread

Not about improving performance, nor creating new threads

Async does not use or create threads in the background

Async can spawn threads to execute the remaining part of code