Math Class

```
// Math Class methods
double x = 3;

double a = Math.Pow(x, 4);
Console.WriteLine($"{x}^3 = {a}");

double b = Math.Sqrt(x);
Console.WriteLine($"{x} squqrt is = {b}");

x = -3.14;
double c = Math.Abs(x);
Console.WriteLine($"Absolute of {x} number is {c}");

x = 3.45;
double d = Math.Round(x);
Console.WriteLine($"Round of {x} number is {d}");

x = 3.99;
double e = Math.Ceiling(x);
Console.WriteLine($"Ceiling of {x} is {e}");

x = 4.9999;
double f = Math.Floor(x);
Console.WriteLine($"Floor of {x} is {f}");

double y = 99;
double g = Math.Max(x, y);
System.Console.WriteLine($"Greater among {x} and {y} is {g}");

double h = Math.Min(x, y);
System.Console.WriteLine($"Smaller amoung {x} and {y} is {h}");


Console.ReadKey();
```

Random Class

```
// Pseudo random numbers
Random random = new Random();

int num = random.Next(1, 7);
double number = random.NextDouble();

System.Console.WriteLine(num);
System.Console.WriteLine(number);

// Three random numbers

int num1, num2, num3;

num1 = random.Next(1, 7);
num2 = random.Next(1, 7);
num3 = random.Next(1, 7);

System.Console.WriteLine($"Three Random numbers are {num1}, {num2}, {num3}");
```

Sample Hypotenuse Program

```
// Find the hypotenuse of a triangle

System.Console.WriteLine("Enter side A");
double a = Convert.ToDouble(Console.ReadLine());

System.Console.WriteLine("Enter side B");
double b = Convert.ToDouble(Console.ReadLine());

double c = Math.Sqrt((a*a) + (b*b));
System.Console.WriteLine($"Hypotenuse of {a} and {b} is {c}");
```

String Methods

```
// String methods
using System.Runtime.InteropServices.Swift;

System.Console.WriteLine("Enter your full name: ");
string fullName = "Kishor Gowda O K";
fullName = fullName.ToUpper();
System.Console.WriteLine($"fullName in UpperCase {fullName}");

fullName = fullName.ToLower();
System.Console.WriteLine($"fullName in LowerCase {fullName}");

string phoneNo = "123-456-7890";
phoneNo = phoneNo.Replace("-", "");
System.Console.WriteLine(phoneNo);

string userName = fullName.Insert(0, "Mr.");
System.Console.WriteLine("Username is : " + userName);

// Length attribute
System.Console.WriteLine("Length of the username: " + userName.Length);

string firstname = fullName.Substring(0, 6);
System.Console.WriteLine(firstname);

string lastName = fullName.Substring(7, 9);
System.Console.WriteLine(lastName);

Console.ReadKey();
```

if, else, else if conditions

```
// if statement - basic form of decision making

System.Console.WriteLine("Please enter your age");
int age = Convert.ToInt32(Console.ReadLine());


if (age > 100)
{
    System.Console.WriteLine("You are too old to have a VoterId");
}
else if (age >= 18)
{
    System.Console.WriteLine("You are eligible to have a VoterId!");
}
else if (age < 0)
{
    System.Console.WriteLine("You are not born yet");
}
else
{
    System.Console.WriteLine("You must be 18+ to have a VoterId");
}


// Another kind

System.Console.WriteLine("Please enter your name: ");
string name = Console.ReadLine();

if (name == "")
{
    System.Console.WriteLine("You did not enter your name!");
}
else
{
    System.Console.WriteLine("Hello " + name);
}

Console.ReadKey();
```

Switch Case statements

```
// Switch statements

using System.Runtime.InteropServices.Swift;

System.Console.WriteLine("What day is today?");
string day = Console.ReadLine();

switch (day)
{
    case "Monday":
        System.Console.WriteLine("It's Monday");
        break;
    case "Tuesday":
        System.Console.WriteLine("It's Tuesday");
        break;
    case "Wednesday":
        System.Console.WriteLine("It's Wednesday");
        break;
    case "Thursday":
        System.Console.WriteLine("It's Thursday");
        break;
    case "Friday":
        System.Console.WriteLine("It's Friday");
        break;
    case "Saturday":
        System.Console.WriteLine("It's Saturday");
        break;
    case "Sunday":
        System.Console.WriteLine("It's Sunday");
        break;
    default:
        System.Console.WriteLine(day + " is not a day!");
        break;
}

Console.ReadKey();
```

Logical Operators

```
// logical operators && (AND), || (OR)

using System.Formats.Asn1;

Console.WriteLine("What's the temperature outside today? ");
double temp = Convert.ToDouble(Console.ReadLine());

if (temp < 10 || temp > 50)
{
    System.Console.WriteLine("Do not go outside");
}
else if (temp >= 10 && temp <= 25)
{
    System.Console.WriteLine("It's warm outside");
}
else
{
    System.Console.WriteLine("It's hot outside");
}
```

While Loop

```
// While Loops

String name = "";

while (name == "")
{
    Console.Write("Enter your name: ");
    name = Console.ReadLine();
}

System.Console.WriteLine("Hello " + name);

while (1 == 1)
{
    System.Console.WriteLine("Infinite loop ");
}
```

For Loop

```
// for loops

for (int i = 1; i <= 10; i += 3)
{
    System.Console.WriteLine(i);
}

// counting down
for (int i = 10; i > 0; i--)
{
    System.Console.WriteLine(i);
}
System.Console.WriteLine("Happy new year!");
```

Nested Loop

```
// Nested Loops
System.Console.Write("How many rows: ");
int rows = Convert.ToInt16(Console.ReadLine());

System.Console.Write("How many columns: ");
int columns = Convert.ToInt16(Console.ReadLine());

System.Console.Write("Which symbol: ");
String symbol = Console.ReadLine();
int i = 0;
while (i < rows)
{
    for (int j = 0; j < columns; j++)
    {
        System.Console.Write(symbol);
    }
    System.Console.WriteLine();
    i++;
}
```

Number guessing Game

```
// Number guessing game in CSharp

Random random = new Random();
bool playAgain = true;
int min = 1;
int max = 100;
int guess;
int number;
int guesses;


while (playAgain)
{
    number = random.Next(min, max + 1);
    guess = -1;
    guesses = 0;
    while (number != guess)
    {
        System.Console.Write("Enter your guess: ");
        guess = Convert.ToInt32(Console.ReadLine());
        if (guess > number)
            System.Console.WriteLine("Your guess is too big");
        else
            System.Console.WriteLine("Your guess is too small");
        guesses++;
    }
    System.Console.WriteLine("Congradulations you guessed correct");
    System.Console.WriteLine("You guessed in " + guesses + " gusses.");
    System.Console.Write("\nDo you wanna keep playing ? (press enter to stop)");
    if (Console.ReadLine() == "")
    {
        playAgain = false;
    }
}

Console.ReadKey();
```

Rock paper scissors

```
// Rock, Paper, Scissors
Random random = new Random();
bool playAgain = true;
String player;
String computer;
String answer;
while (playAgain)
{
    player = "";
    computer = "";
    while (player != "ROCK" && player != "SCISSORS" && player != "PAPER")
    {
        System.Console.WriteLine("\nEnter ROCk, PAPER or SCISSORS: ");
        player = Console.ReadLine();
        player = player.ToUpper();
    }
    System.Console.WriteLine(player);

    int randomNum = random.Next(1, 4);
    switch (randomNum)
    {
        case 1:
            computer = "ROCK";
            break;
        case 2:
            computer = "PAPER";
            break;
        case 3:
            computer = "SCISSORS";
            break;
    }
    System.Console.WriteLine("Player: " + player);
    System.Console.WriteLine("Computer: " + computer);

    switch (player)
    {
        case "ROCK":
            if (computer == "ROCK")
                System.Console.WriteLine("It's a Draw");
            else if (computer == "PAPER")
```

```
                System.Console.WriteLine("You Loose");
            else
                System.Console.WriteLine("You Win");
            break;
        case "SCISSORS":
            if (computer == "SCISSORS")
                System.Console.WriteLine("It's a Draw");
            else if (computer == "PAPER")
                System.Console.WriteLine("You Win");
            else
                System.Console.WriteLine("You Loose");
            break;
        case "PAPER":
            if (computer == "PAPER")
                System.Console.WriteLine("It's a Draw");
            else if (computer == "ROCK")
                System.Console.WriteLine("You Win");
            else
                System.Console.WriteLine("You Loose");
            break;
    }
    System.Console.Write("Do you want to play again (Y/N): ");
    answer = Console.ReadLine().ToUpper();
    if (answer != "Y")
    {
        playAgain = false;
    }
}

Console.ReadKey();
```

Calculator Program

```
do
{
    double num1 = 0;
    double num2 = 0;
    double result = 0;


    System.Console.WriteLine("-------------------");
    System.Console.WriteLine("Calculator Program");
    System.Console.WriteLine("-------------------");

    System.Console.Write("Enter number 1: ");
    num1 = Convert.ToDouble(Console.ReadLine());

    System.Console.Write("Enter number 2: ");
    num2 = Convert.ToDouble(Console.ReadLine());

    System.Console.WriteLine("Enter an option: ");
    System.Console.WriteLine("\t+ : Add");
    System.Console.WriteLine("\t- : Subtract");
    System.Console.WriteLine("\t* : Multiply");
    System.Console.WriteLine("\t/ : Divide");
    System.Console.Write("Enter an option: ");

    switch (Console.ReadLine())
    {
        case "+":
            result = num1 + num2;
            System.Console.WriteLine($"Your result: {num1} + {num2} = {result}");
            break;
        case "-":
            result = num1 - num2;
            System.Console.WriteLine($"Your result: {num1} - {num2} = {result}");
            break;
        case "*":
            result = num1 * num2;
            System.Console.WriteLine($"Your result: {num1} * {num2} = {result}");
            break;
        case "/":
            result = num1 / num2;
```

```
            System.Console.WriteLine($"Your result: {num1} / {num2} = {result}");
            break;
        default:
            System.Console.WriteLine("That was not a valid option!");
            break;
    }
    System.Console.WriteLine("Would you like to continue (Y/N)");
} while (Console.ReadLine().ToUpper() == "Y");

System.Console.WriteLine("\nBye");
```

foreach Loop and Array's

```
// Arrays

// How to declare an array

String[] students = new string[4]; // an array of size 4w

students[0] = "Karan";
students[1] = "Karma";
students[2] = "Kushi";
students[3] = "Krish";

// foreach loop
foreach (string student in students)
{
    System.Console.WriteLine(student);
}

System.Console.WriteLine();

String[] cars = { "BMW", "Mustang", "Audi" };

cars[0] = "Tesla";

System.Console.WriteLine(cars[0]);
System.Console.WriteLine(cars[1]);
System.Console.WriteLine(cars[2]);
// System.Console.WriteLine(cars[3]); IndexOutOfRange Exception

for (int i = 0; i < cars.Length; i++)
{
    System.Console.WriteLine(cars[i]);
}
```

Methods

```
class Program
{
    public static void Main(String[] args)
    {
        // methods = performs a section of code, whenever it is called "invoked".
        //          benifit = let's us reuse code w/o writing it multiple times
        String name = "Bro";
        int age = 22;

        singHappyBirthday(name, age);
    }

    static void singHappyBirthday(String birthdayBoy, int yearsOld)
    {
        System.Console.WriteLine($"Happy birthday to you!");
        System.Console.WriteLine($"Happy birthday to you!");
        System.Console.WriteLine($"Happy birthday dear {birthdayBoy}!");
        System.Console.WriteLine($"You are {yearsOld} years old!");
        System.Console.WriteLine($"Happy birthday to you!");
        System.Console.WriteLine();
    }
}
```

Return Keyword

```
class Program
{
    static void Main(String[] args)
    {
        double result;
        double x;
        double y;
        System.Console.Write("Enter in number 1: ");
        x = Convert.ToDouble(Console.ReadLine());

        System.Console.Write("Enter in number 2: ");
        y = Convert.ToDouble(Console.ReadLine());

        result = Multiply(x, y);
        System.Console.WriteLine($"Multiple of {x} * {y} = {result}");
    }
    static double Multiply(double num1, double num2)
    {
        double num3 = num1 * num2;
        return num3;
    }
}
```

Method Overloading

```
class Program
{
    static void Main(String[] args)
    {
        // method overloading = methods share same name, but different parameters
        //                      name + parameters = signature
        //                      methods must have a unique signature

        double total = Multiply(9, 8);

        System.Console.WriteLine($"First total = {total}");

        total = Multiply(9, 7, 5);

        System.Console.WriteLine($"Second total = {total}");

    }

    static double Multiply(double x, double y)
    {
        return x * y;
    }

    static double Multiply(double x, double y, double z)
    {
        return x * y * z;
    }
}
```

Params keyword

```
using System.ComponentModel.Design;
using System.Numerics;

class Program
{
    static void Main(String[] args)
    {
        // params keyword = a method parameter that takes a variable number of arguments
        //                  The parameter type must be a single - dimentional array

        double total = CheckOut(1, 2, 3, 4, 5, 6);

        System.Console.WriteLine($"Your final total is: {total}");

    }

    static double CheckOut(params double[] prices)
    {
        double total = 0;
        foreach (double price in prices)
            total += price;

        return total;
    }
}
```

Exception handeling

```
using System.ComponentModel.Design;
using System.Numerics;

class Program
{
    static void Main(String[] args)
    {
        // exception -> errors that occur during execution

        //          try -> try some code that is considered "dangerous"
        //          catch -> catched and handles exceptions when they occur
        //          finally -> always executes regardless if exception is caught or not
        int x, y;
        double result;
        try
        {
            System.Console.Write("Enter number 1: ");
            x = Convert.ToInt32(Console.ReadLine());

            System.Console.Write("Enter number 2: ");
            y = Convert.ToInt32(Console.ReadLine());

            result = x / y;

            System.Console.WriteLine($"Division of {x} / {y} = {result}");

        }
        catch (FormatException e)
        {
            System.Console.WriteLine("Enter only numbers please");
        }
        catch (DivideByZeroException e)
        {
            System.Console.WriteLine("You can't devide by zero!");
        }
        catch (Exception e)
        {
            System.Console.WriteLine("Something went wrong!");
        }
        finally
```

```
        {
            System.Console.WriteLine("Thanks for visiting!");
        }
    }
}
```

Conditional operator

```
using System.ComponentModel.Design;
using System.Numerics;

class Program
{
    static void Main(String[] args)
    {
        // conditional operator -> used in conditional assignment if condition is true / false
        // (condition) ? x : y;

        double temperatur = 20;
        String message;

        message = (temperatur >= 15) ? "It's warm outside" : "It's cold outside";
        System.Console.WriteLine(message);
    }
}
```

String Interpolation

```
String name = "Bro";
int age = 21;
System.Console.WriteLine($"Hello {name,10}; how are you ?");
System.Console.WriteLine($"You are {age,-10} years age");
```

Multi-dimentional Arrays

```
String[] ford = { "Mustang", "F-150", "Explorer" };
String[] chevy = { "Corvette", "Camaro", "Silverado" };
String[] toyota = { "Corolla", "Camry", "Rav4" };

String[,] parkinglot = { { "Mustang", "F-150", "Explorer" },
                         { "Corvette", "Camaro", "Silverado" },
                         { "Corolla", "Camry", "Rav4" }
                       };

parkinglot[0, 2] = "Fusion";
parkinglot[2, 0] = "Takoma";
/*
foreach (String car in parkinglot)
{
    System.Console.WriteLine(car);
}
*/

for (int i = 0; i < parkinglot.GetLength(0); i++)
{
    for (int j = 0; j < parkinglot.GetLength(1); j++)
    {
        System.Console.Write(parkinglot[i, j] + "   ");
    }
    System.Console.WriteLine();
}
```

Class

Create a new file and write the below code to create a class "blueprint"

```
static class Messages
{
    public static void Hello()
    {
        System.Console.WriteLine("Hello! Welcome to the program");
    }
    public static void Waiting()
    {
        System.Console.WriteLine("I am waiting for something!");
    }
    public static void Bye()
    {
        System.Console.WriteLine("Bye! Thanks for visiting!");
    }
}
```

To access this call it using the below code

```
Messages.Hello();
Messages.Waiting();
Messages.Bye();
```

Object creation

```
using System.ComponentModel.Design;
using System.Numerics;

class Program
{
    static void Main(String[] args)
    {
        // object -> An instance of a class
        //           A class can be used as a blueprint to create objects (OOP)
        //           objects can have fileds & methods (characteristics and actions)

        Human human1 = new Human();
        Human human2 = new Human();

        human1.name = "Rick";
        human1.age = 77;
        human1.Eat();
        human1.Sleep();

        human2.name = "Morty";
        human2.age = 17;
        human2.Eat();
        human2.Sleep();
    }
}

class Human
{
    public String name;
    public int age;

    public void Eat()
    {
        System.Console.WriteLine($"{name} is eating");
    }

    public void Sleep()
    {
        System.Console.WriteLine($"{name} is sleeping");
```

```
        }
    }
```

Constructor

```
using System.ComponentModel.Design;
using System.Numerics;
using System.Runtime.CompilerServices;

class Program
{
    static void Main(String[] args)
    {
        // cunstructor -> A special method in class
        //                Same name as class name
        //                Can be used to assign arguments to fields when creating an object

        Human human1 = new Human("Rick", 77);
        Human human2 = new Human("Morty", 17);

        human1.Eat();
        human1.Sleep();

        human2.Eat();
        human2.Sleep();

        System.Console.WriteLine();

        Car car1 = new Car("Ford", "Mustang", 2022, "Blue");
        car1.Drive();

        Car car2 = new Car("Chevy", "Chorvette", 2021, "Green");
        car2.Drive();
    }
}

class Human
{
    public String name;
    public int age;

    public Human(String name, int age)
    {
        this.name = name;
        this.age = age;
```

```
    }
    public void Eat()
    {
        System.Console.WriteLine($"{name} is eating");
    }

    public void Sleep()
    {
        System.Console.WriteLine($"{name} is sleeping");
    }
}

class Car
{
    String make;
    String model;
    int year;
    String color;

    public Car(String make, String model, int year, String color)
    {
        this.make = make;
        this.model = model;
        this.year = year;
        this.color = color;
    }

    public void Drive()
    {
        System.Console.WriteLine($"You drive this {make} {model}");
    }
}
```

Static Modifier

```
class Program
{
    static void Main(String[] args)
    {
        // static -> modifier to declare a static member, which belongs to the class itself
        //           rather than to any specific object

        Car car1 = new Car("Mustang");
        Car car2 = new Car("Corvette");
        Car car3 = new Car("Lambo");

        System.Console.WriteLine(Car.numberOfCars);

        Car.StartRace();
    }
}
// we cannot create an object from a static class
// and static methods and attributes are accessed using a class name instead of the usual object r
class Car
{
    String model;
    public static int numberOfCars;
    public Car(String model)
    {
        this.model = model;
        numberOfCars++;
    }

    public static void StartRace()
    {
        System.Console.WriteLine("The race has begun");
    }
}
```

Overloaded Constructor

```
class Program
{
    static void Main(String[] args)
    {
        // overloaded constructors -> technique to create multiple constructors
        //                            with a different set of parameters.
        //                            name + parameters = signatur

        Pizza pizza1 = new Pizza("stuffed crust", "red sauce", "Motserrella", "pepperoni");
        Pizza pizza2 = new Pizza("stuffed crust", "red sauce", "Motserrella");
        Pizza pizza3 = new Pizza("stuffed crust", "red sauce");
        Console.ReadKey();
    }

}
class Pizza
{
    String bread;
    String sauce;
    String cheese;
    String topping;

    public Pizza(String bread, String sauce)
    {
        this.bread = bread;
        this.sauce = sauce;
    }
    public Pizza(String bread, String sauce, String cheese)
    {
        this.bread = bread;
        this.sauce = sauce;
        this.cheese = cheese;
    }
    public Pizza(String bread, String sauce, String cheese, String topping)
    {
        this.bread = bread;
        this.sauce = sauce;
        this.cheese = cheese;
        this.topping = topping;
```

```
        }
    }
```

Inheritance

```
using System.Security.Cryptography.X509Certificates;
using System.Threading.Tasks.Dataflow;

class Program
{
    static void Main(String[] args)
    {
        // inheritance -> 1 or more child classes recieving fields, methods etc. from a common pa
        Car car = new Car();
        Bicycle bicycle = new Bicycle();
        Boat boat = new Boat();

        System.Console.WriteLine(car.wheels);
        System.Console.WriteLine(car.speed);
        car.go();

        System.Console.WriteLine(bicycle.wheels);
        System.Console.WriteLine(bicycle.speed);
        bicycle.go();

        System.Console.WriteLine(boat.wheels);
        System.Console.WriteLine(boat.speed);
        boat.go();
        Console.ReadKey();
    }

}
class Vehicle
{
    public int speed = 0;

    public void go()
    {
        System.Console.WriteLine("This vehicle is moving!");
    }
}

class Car : Vehicle
{
    public int wheels = 4;
```

```
}
class Bicycle : Vehicle
{
    public int wheels = 2;

}
class Auto : Vehicle
{
    public int wheels = 3;

}
class Boat : Vehicle
{
    public int wheels = 0;

}
```

Abstract class

- Basically a class that cannot be used to create an object, it's a blue-print for other classes and can be created using abstract modifier

```
class Program
{
    static void Main(String[] args)
    {
        // abstract classes -> modifier that indicates missing components or incomplete implementa
        Car car = new Car();
        Bicycle bicycle = new Bicycle();

        // Vehicle vehicle = new Vehicle();

        Console.ReadKey();
    }
}

abstract class Vehicle
{
    public int speed = 0;

    public void go()
    {
        System.Console.WriteLine("This vehicle is moving!");
    }
}

class Car : Vehicle
{
    public int wheels = 4;
    int maxSpeed = 150;

}
class Bicycle : Vehicle
{
    public int wheels = 2;
    int maxSpeed = 30;
}
class Auto : Vehicle
{
    public int wheels = 3;
    int maxSpeed = 80;
}
class Boat : Vehicle
{
    public int wheels = 0;
```

```
    int maxSpeed = 200;
}
```

Array of objects

```
using System.Security.Cryptography.X509Certificates;
using System.Threading.Tasks.Dataflow;

class Program
{
    static void Main(String[] args)
    {
        // array of objects

        Car[] garage = new Car[3];
        Car car1 = new Car("Mustang");
        Car car2 = new Car("Tesla SX");
        Car car3 = new Car("BMW");

        garage[0] = car1;
        garage[1] = car2;
        garage[2] = car3;

        foreach (Car car in garage)
        {
            System.Console.WriteLine(car.model);
        }

        // second way

        Car[] garage2 = { car1, car2, car3 };

        Console.ReadKey();
    }

}

class Car
{
    public String model;

    public Car(String model)
    {
        this.model = model;
```

```
        }
    }
```

Objects as Argumrnts

```
using System.Net.WebSockets;
using System.Security.Cryptography.X509Certificates;
using System.Threading.Tasks.Dataflow;

class Program
{
    static void Main(String[] args)
    {
        // objects as arguments

        Car[] garage = new Car[3];
        Car car1 = new Car("Mustang", "Blue");
        Car car2 = new Car("Tesla SX", "Grey");
        Car car3 = new Car("BMW", "Purple");

        Car car4 = Copy(car2);

        System.Console.WriteLine($"{car2.color} x {car2.model}");
        System.Console.WriteLine($"{car4.color} x {car4.model}");

        System.Console.WriteLine($"{car1.color}   {car1.model}");
        changeColor(car1, "Pink");
        System.Console.WriteLine($"{car1.color}   {car1.model}");

        Console.ReadKey();
    }

    public static void changeColor(Car car, String color)
    {
        car.color = color;
    }

    public static Car Copy(Car car)
    {
        return new Car(car.model, car.color);
    }
}

class Car
{
```

```
    public String model;
    public String color;

    public Car(String model, String color)
    {
        this.model = model;
        this.color = color;
    }
}
```

Method overriding

```
class Program
{
    static void Main(String[] args)
    {
        // method overriding -> provides a new version of a method inherited from parent class
        //                      inherited methods must be: abstract, virtual, or already overrider
        //                      Used with ToString(), polymorphism

        Dog dog = new Dog();
        Cat cat = new Cat();

        dog.Speek();
        cat.Speek();

        Console.ReadKey();
    }
}

class Animal
{
    public virtual void Speek()
    {
        System.Console.WriteLine("The animal goes burrr");
    }
}

class Dog : Animal
{
    public override void Speek()
    {
        System.Console.WriteLine("The dog goes *woof*");
    }
}

class Cat : Animal
{
    public override void Speek()
    {
        System.Console.WriteLine("The cat goes *meow*");
```

```
    }
}
```

Overriding ToString() method

```
class Program
{
    static void Main(String[] args)
    {
        // ToString() -> converts an object to it's string representation so that it is suitable f

        Car car = new Car("Ford", "Mustang", 2024, "Blue");
        System.Console.WriteLine(car.ToString());
        System.Console.WriteLine(car);
        Console.ReadKey();
    }

}

class Car
{
    String make;
    String model;
    int year;
    String color;

    public Car(String make, String model, int year, String color)
    {
        this.make = make;
        this.model = model;
        this.year = year;
        this.color = color;
    }

    public override string ToString()
    {
        return $"This is a {make} {model}";
    }
}
```

Polymorphism

```
class Program
{
    static void Main(String[] args)
    {
        // polymorphism = Greek word that means to "have many forms"
        //                Objects can be identified by more than one type
        //                Ex. A Dog is also: Canine, Animal, Organism

        Car car = new Car();
        Bicycle bicycle = new Bicycle();
        Boat boat = new Boat();

        Vehicle[] vehicles = { car, bicycle, boat };

        foreach (Vehicle vehicle in vehicles)
        {
            vehicle.Go();
        }
    }
}

class Vehicle
{
    public virtual void Go()
    {

    }
}

class Car : Vehicle
{
    public override void Go()
    {
        System.Console.WriteLine("The car is moving");
    }
}

class Bicycle : Vehicle
{
    public override void Go()
```

```
    {
        System.Console.WriteLine("The bicycle is moving");
    }
}

class Boat : Vehicle
{
    public override void Go()
    {
        System.Console.WriteLine("The boat is moving");
    }
}
```

Intefaces

```
class Program
{
    static void Main(String[] args)
    {
        // interface -> defines a "contract" that all the classes inheriting from should follow

        //              An interface declares "What a class should have"
        //              An inheriting class defines "how it should do it"

        //              Benifit = security + multiple inheritance + "plug-and-play"

        Rabbit rabbit = new Rabbit();
        Hawk hawk = new Hawk();
        Fish fish = new Fish();

        rabbit.Flee();
        hawk.Hunt();
        fish.Flee();
        fish.Hunt();

        Console.ReadKey();
    }
}

interface IPrey
{
    void Flee();
}

interface IPredator
{
    void Hunt();
}

class Rabbit : IPrey
{
    public void Flee()
    {
        System.Console.WriteLine("The Rabbit runs away");
    }
```

```
}

class Hawk : IPredator
{
    public void Hunt()
    {
        System.Console.WriteLine("The hawk is searching for food!");
    }
}

class Fish : IPrey, IPredator
{
    public void Hunt()
    {
        System.Console.WriteLine("The fish is searching for smaller fish");
    }
    public void Flee()
    {
        System.Console.WriteLine("The fish swims away");
    }
}
```

List

```
class Program
{
    static void Main(String[] args)
    {
        // List = data structures that represent a list of objects that can be accessed by index
        //        Similar to array, but can dynamically increase / decrease it's size
        //        using System.Collections.Generic;

        List<String> foods = new List<String>();
        foods.Add("Coke");
        foods.Add("Pizza");
        foods.Add("Veg Burger");
        foods.Add("French Frise");
        foods.Add("Coke");

        foods.Remove("Coke");
        foods.Insert(0, "Sushi");
        System.Console.WriteLine($"Length of the foods List: {foods.Count}");
        System.Console.WriteLine($"Index of an element: {foods.IndexOf("Pizza")}");
        System.Console.WriteLine($"Last index of an element: {foods.LastIndexOf("Coke")}");
        System.Console.WriteLine($"Weather it contains it or not: {foods.Contains("Coke")}");
        foods.Sort();

        System.Console.WriteLine("Sorted in alphabetical order");
        foreach (String food in foods)
        {
            System.Console.WriteLine(food);
        }

        foods.Reverse();
        System.Console.WriteLine("Sorted in reverse order");
        foreach (String food in foods)
        {
            System.Console.WriteLine(food);
        }

        foods.Clear(); // clears all the elements of an list
        String[] foods1 = foods.ToArray(); // converts the List to an array !

        System.Console.WriteLine(foods[2]);
```

```
            Console.ReadKey();
        }
    }
```

List of objects

```
class Program
{
    static void Main(String[] args)
    {
        // List of objects
        List<Player> players = new List<Player>();

        Player player1 = new Player("Kendric");
        Player player2 = new Player("Michal");
        Player player3 = new Player("Virat Kohli");

        players.Add(player1);
        players.Add(player2);
        players.Add(player3);

        foreach (Player player in players)
        {
            System.Console.WriteLine(player);
        }

        Console.ReadKey();
    }
}

class Player
{
    String username;

    public Player(String username)
    {
        this.username = username;
    }

    public override string ToString()
    {
        return this.username;
    }
}
```

Getters and Setters

```
class Program
{
    static void Main(String[] args)
    {
        // gettters and setters -> add security to fields by encapsulation
        //                         they are accessors found within properties

        // Porperties = combine aspects of both fiels and methods (share name with fields )
        // get accessor = used to return the property value
        // set accessor = used to assign a new value
        // value keyword = defines the value  being assigned by the set (parameters)

        Car car = new Car(300);
        System.Console.WriteLine(car.Speed);
        car.Speed = 10000;
        System.Console.WriteLine(car.Speed);


    }
}

class Car
{
    private int speed;

    public Car(int speed)
    {
        Speed = speed;
    }

    public int Speed
    {
        get { return speed; }
        set
        {
            if (value > 500)
            {
                speed = 500;
            }
            else
```

```
            {
                speed = value;
            }
        }
    }
}
```

Auto Implemented Properties

```
using System.Dynamic;

class Program
{
    static void Main(String[] args)
    {
        // Auto implimented properties = shortcuts when no additional logic is required in the pr
        //              you do not have to define a field for a property
        //              you only have to write get; and/or set; inside the property

        Car car = new Car("Ford");
        System.Console.WriteLine(car.Model);
    }
}

class Car
{
    // String model;
    // public String Model
    // {
    //     get { return model; }
    //     set { model = value; }
    // }

    // Instead of the above code all we need to write is the below autoimplementation

    public String Model { get; set; }

    public Car(String model)
    {
        Model = model;
    }
}
```

Enums

```
class Program
{
    static void Main(String[] args)
    {
        // enums = special "class" that contains a set of named integers constraints.
        //         Use enums when you have values that you know will not change
        //         To get the interger value from an item, you must explicitely convert it to an i

        System.Console.WriteLine(Planets.Pluto + " is a planet, it's number: " + (int)Planets.Plut

        String name = PlanetRadius.Earth.ToString();
        int radius = (int)PlanetRadius.Earth;
        double volume = Volume(PlanetRadius.Earth);

        System.Console.WriteLine($"planet: {name}");
        System.Console.WriteLine($"Radius: {radius} km");
        System.Console.WriteLine($"Volume: {volume} km^3");
    }

    public static double Volume(PlanetRadius radius)
    {
        double volume = (4.0 / 3.0) * Math.PI * Math.Pow((int)radius, 3);
        return volume;
    }
}

enum Planets
{
    Mercury,
    Venus,
    Earth,
    Mars,
    Jupiter,
    Saturn,
    Uranus,
    Neptune,
    Pluto
}

enum PlanetRadius
```

```
{
    Mercury = 2439,
    Venus = 6051,
    Earth = 6371,
    Mars = 3389,
    Jupiter = 69911,
    Saturn = 58232,
    Uranus = 25362,
    Neptune = 24622,
    Pluto = 1188
}
```

Generics

```
using System.Dynamic;

class Program
{
    static void Main(String[] args)
    {
        // generic = "not specific to a particular data type"
        //           add <T> to: classes, methods, fields, etc
        //           allows for code reusability for different data types

        int[] intArray = { 1, 2, 3 };
        double[] doubleArray = { 1.0, 2.0, 3.0 };
        String[] stringArray = { "One", "Two", "Three" };

        displayElements(intArray);
        displayElements(doubleArray);
        displayElements(stringArray);
        Console.ReadKey();
    }

    public static void displayElements<Thing>(Thing[] array)
    {
        foreach (Thing item in array)
        {
            System.Console.Write(item + " ");
        }
        System.Console.WriteLine();
    }
}
```

MultiThredding

```
using System.Threading;

class Program
{
    static void Main(String[] args)
    {
        // threds = an execution path of a program
        //          We can use multiple threads to perform,
        //          different tasks of our program at same time.
        //          Current thread runnding is "main" thread
        //          using System.Threading;

        Thread mainThread = Thread.CurrentThread;
        mainThread.Name = "Main Thread";
        System.Console.WriteLine(mainThread.Name);

        Thread thread1 = new Thread(() => countDown("Timer #1"));  // if we have to pass a paramet
        Thread thread2 = new Thread(countUp);

        thread1.Start();
        thread2.Start();

        // countDown();
        // countUp();

        System.Console.WriteLine(mainThread.Name + " is completed!");
        Console.ReadKey();
    }

    public static void countDown(String name)
    {
        for (int i = 10; i >= 0; i--)
        {
            System.Console.WriteLine("Timer #1: " + i + " seconds");
            Thread.Sleep(1000);
        }
        System.Console.WriteLine("Timer #1 is complete ");
    }
    public static void countUp()
    {
```

```
        for (int i = 0; i <= 10; i++)
        {
            System.Console.WriteLine("Timer #2: " + i + " seconds");
            Thread.Sleep(1000);
        }
        System.Console.WriteLine("Timer #2 is complete ");
    }
}
```