C# Tutorials

You need to learn C#

1. Flexible (console apps, web services, games)
2. Great for aspiring game developers (Unity)
3. Average salary $63,439/year for C# developers (Glassdoor.com)

1) Get started

```
using System;

namespace MyFirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("I like pizza!");
            Console.WriteLine("It's really good!");
            Console.Beep();
        }
    }
}
```

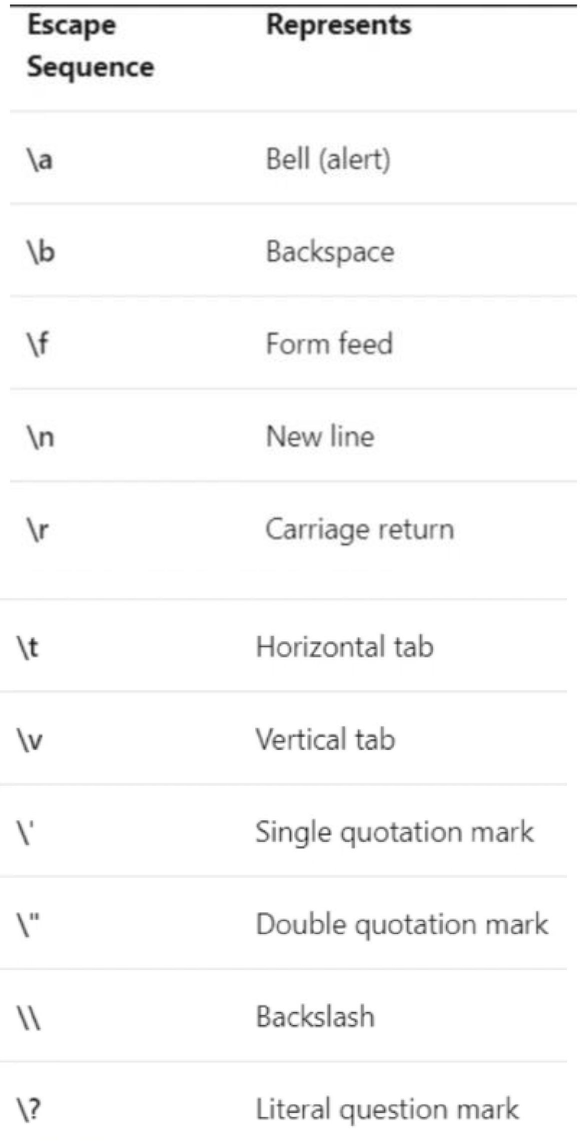
2) Input and Output Formatting

```
Console.Write("Hey!");
Console.WriteLine("Hello!");
```

```
Console.WriteLine("What's your age?");
String name = Console.ReadLine();
Console.WriteLine("What's your age?");
int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Hello " + name);
Console.WriteLine("You are " + age + " years old");
```

```
/*
* multiline
* comment
*/
```

Escape Sequences:



Escape Sequence	Represents
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\?	Literal question mark

3) Comments

```
// this is a comment
/*
* multiline
* comment
*/
```

4) Variables

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int x; // declaration
        x = 123; // initialization
        int y = 321; // declaration + initialization
        int z = x + y;

        int age = 21; // whole integer
        double height = 300.5; // decimal number
        bool alive = false; // true or false
        char symbol = '@'; // single character
        String name = "Bro"; // a series of characters

        Console.WriteLine("Hello " + name);
        Console.WriteLine("Your age is " + age);
        Console.WriteLine("Your height is " + height + "cm");
        Console.WriteLine("Are you alive? " + alive);
        Console.WriteLine("Your symbol is: " + symbol);
        String userName = symbol + name;
        Console.WriteLine("Your username is: " + userName);
        Console.ReadKey();
    }
}
```

5) Constants

```
class Program
{
    static void Main(string[] args)
    {
        // constants  = immutable values which are known at compile time
        //              and do not change for the life of the program

        const double pi = 3.14;

        //pi = 420; //can't change this constant

        Console.WriteLine(pi);

        Console.ReadKey();
    }
}
```

6) Type Casting

```
class Program
{
    static void Main(string[] args)
    {
        // type casting = Converting a value to a different data type
        //                Useful when we accept user input (string)
        //                Different data types can do different things

        double a = 3.14;
        int b = Convert.ToInt32(a);

        int c = 123;
        double d = Convert.ToDouble(c);

        int e = 321;
        String f = Convert.ToString(e);

        String g = "$";
        char h = Convert.ToChar(g);

        String i = "true";
        bool j = Convert.ToBoolean(i);

        Console.WriteLine(b.GetType());
        Console.WriteLine(d.GetType());
        Console.WriteLine(f.GetType());
        Console.WriteLine(h.GetType());
        Console.WriteLine(j.GetType());

        Console.ReadKey();
    }
}
```

7) Arithmetic Operators

```
class Program
{
    static void Main(string[] args)
    {
        int friends = 5;

        friends = friends + 1;
        //friends += 1;
        //friends++;

        //friends = friends - 1;
        //friends -= 1;
        //friends--;

        //friends = friends * 2;
        //friends *= 2;

        //friends = friends / 2;
        //friends /= 2;

        //int remainder = friends % 2;
        //Console.WriteLine(remainder);

        Console.WriteLine(friends);

        Console.ReadKey();
    }
}
```

8) Math Class

```
class Program
{
    static void Main(string[] args)
    {

        double x = 3;
        double y = 5;

        double a = Math.Pow(x, 2);
        double b = Math.Sqrt(x);
        double c = Math.Abs(x);
        double d = Math.Round(x);
        double e = Math.Ceiling(x);
        double f = Math.Floor(x);
        double g = Math.Max(x, y);
        double h = Math.Min(x, y);

        Console.WriteLine(a);

        Console.ReadKey();
    }
}
```

9) Random Numbers

```
class Program
{
    static void Main(string[] args)
    {
        Random random = new Random(); // initialise

        int num = random.Next(1, 7); // last number will be excluded
                                     //double num = random.NextDouble(); // value 
between 0 and 1
        Console.WriteLine(num);
        Console.ReadKey();
    }
}
```

10) String Methods

```
class Program
{
    static void Main(string[] args)
    {
        String fullName = "Bro Code";
        String phoneNumber = "123-456-7890";

        //fullName = fullName.ToUpper();
        //fullName = fullName.ToLower();
        //Console.WriteLine(fullName);

        //phoneNumber = phoneNumber.Replace("-","");
        //Console.WriteLine(phoneNumber);

        //String userName = fullName.Insert(0,"Mr.");
        //Console.WriteLine(userName);

        //Console.WriteLine(fullName.Length);

        String firstName = fullName.Substring(0, 3);
        String lastName = fullName.Substring(4, 4);

        Console.WriteLine(firstName);
        Console.WriteLine(lastName);

        Console.ReadKey();
    }
}
```

11) If Statements

```
static void Main(string[] args)
{
    //if statement = a basic form of decision making

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

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

    Console.ReadKey();
}
```

12) Switch Statement

```
static void Main(string[] args)
{
    // switch = an efficient alternative to many else if statements

    Console.WriteLine("What day is it today?");
    String day = Console.ReadLine();

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

13) Logical Operators

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

    // logical operators = Can be used to check if more than 1 condition is 
true/false

    // && (AND)
    // || (OR)

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

    if (temp >= 10 && temp <= 25)
    {
        Console.WriteLine("It's warm outside!");
    }
    else if (temp <= -50 || temp >= 50)
    {
        Console.WriteLine("DO NOT GO OUTSIDE!");
    }


    Console.ReadKey();
}
```

14) While Loop

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

    // while loop = repeats some code while some condition remains true

    String name = "";

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

    Console.WriteLine("Hello " + name);

    Console.ReadKey();
}
```

15) For Loops

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

    // for loop = repeats some code a FINITE amount of times

    // Count up to 10
    for (int i = 1; i <= 10; i++)
    {
        Console.WriteLine(i);
    }

    // Count down from 10
    for (int i = 10; i > 0; i--)
    {
        Console.WriteLine(i);
    }
    Console.WriteLine("HAPPY NEW YEAR!");

    Console.ReadKey();
}
```

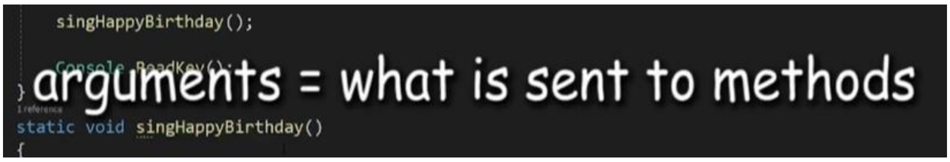
16) Arrays

```
static void Main(string[] args)
{
    // array = a variable that can store multiple values. fixed size

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

    String[] cars = new string[3];

    cars[0] = "Tesla";
    cars[1] = "Mustang";
    cars[2] = "Corvette";

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

    Console.ReadKey();
}
```

17) Foreach loop

```
static void Main(string[] args)
{
    // foreach loop = a simpler way to iterate over an array, but it's less
flexible

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

    foreach (String car in cars)
    {
        Console.WriteLine(car);
    }

    Console.ReadKey();
}
```

18) Methods



```
    singHappyBirthday();

    Console.ReadKey();
}
1 reference
static void singHappyBirthday()
{
```

arguments = what is sent to methods

```
static void Main(string[] args)
{
    // method  = performs a section of code, whenever it's called "invoked".
    //           benefit = Let's us reuse code w/o writing it multiple times
    //           Good practice is to capitalize method names (I forgot in this
video)

    String name = "Bro";
    int age = 21;

    SingHappyBirthday(name, age);
    Console.ReadKey();
}
static void SingHappyBirthday(String birthdayBoy, int yearsOld)
{
    Console.WriteLine("Happy birthday dear " + birthdayBoy);
    Console.WriteLine("You are " + yearsOld + " years old!");
    Console.WriteLine();
}
```

19) Return Keyword

```
static void Main(string[] args)
{
    // return  = returns data back to the place where a method is invoked

    double x;
    double y;
    double result;

    Console.WriteLine("Enter in number 1: ");
    x = Convert.ToDouble(Console.ReadLine());

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

    result = Multiply(x, y);

    Console.WriteLine(result);

    Console.ReadKey();
}
static double Multiply(double x, double y)
{
    return x * y;
}
```

20) Method Overloading

```
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;

    total = Multiply(2, 3, 4);

    Console.WriteLine(total);
    Console.ReadKey();
}

static double Multiply(double a, double b)
{
    return a * b;
}
static double Multiply(double a, double b, double c)
{
    return a * b * c;
}
```

21) Pramas Keyword

```
static void Main(string[] args)
{
    //params keyword = a method parameter that takes a variable number of
arguments.
    //                 The parameter type must be a single - dimensional array

    double total = CheckOut(3.99, 5.75, 15, 1.00, 10.25);

    Console.WriteLine(total);
    Console.ReadKey();
}

static double CheckOut(params double[] prices)
{
    double total = 0;

    foreach (double price in prices)
    {
        total += price;
    }
    return total;
}
```

22) Exception Handling

```
static void Main(string[] args)
{
    // exception = errors that occur during execution

    //        try     = try some code that is considered "dangerous"
    //        catch   = catches and handles exceptions when they occur
    //        finally = always executes regardless if exception is caught or not

    int x;
    int y;
    double result;

    try
    {
        Console.Write("Enter number 1: ");
        x = Convert.ToInt32(Console.ReadLine());

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

        result = x / y;

        Console.WriteLine("result: " + result);
    }
    catch (FormatException e)
    {
        Console.WriteLine("Enter ONLY numbers PLEASE!");
    }
    catch (DivideByZeroException e)
    {
        Console.WriteLine("You can't divide by zero! IDIOT!");
    }
    catch (Exception e)
    {
        Console.WriteLine("Something went wrong!");
    }
    finally
    {
        Console.WriteLine("Thanks for visiting!");
    }

    Console.ReadKey();
}
```

23) Conditional Operator

```
    // conditional operator = used in conditional assignment if a condition is
true/false

    //(condition) ? x : y

    double temperature = 20;
    String message;

    message = (temperature >= 15) ? "It's warm outside!" : "It's cold outside!";

    Console.WriteLine(message);
```

24) String Interpolation

```
static void Main(string[] args)
{
    // conditional operator = used in conditional assignment if a condition is
true/false

    //(condition) ? x : y

    double temperature = 20;
    String message;

    message = (temperature >= 15) ? "It's warm outside!" : "It's cold outside!";

    Console.WriteLine(message);

    Console.ReadKey();
}
```

25) Multidimensional arrays

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

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

    parkingLot[0, 2] = "Fusion";
    parkingLot[2, 0] = "Tacoma";
    /*
    foreach(String car in parkingLot)
    {
        Console.WriteLine(car);
    }
    */
    for (int i = 0; i < parkingLot.GetLength(0); i++)
    {
        for (int j = 0; j < parkingLot.GetLength(1); j++)
        {
            Console.Write(parkingLot[i, j] + " ");
        }
        Console.WriteLine();
    }

    Console.ReadKey();
}
```

26) Class

```
class Program
{
    static void Main(string[] args)
    {
        // class = A bundle of related code.
        //              Can be used as a blueprint to create objects (OOP)

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

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

27) Objects

```
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 fields & methods (characteristics & actions)

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

    human1.name = "Rick";
    human1.age = 65;

    human2.name = "Morty";
    human2.age = 16;

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

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

    Console.ReadKey();
}
    }
    class Human
{
    public String name;
    public int age;

    public void Eat()
    {
        Console.WriteLine(name + " is eating");
    }
    public void Sleep()
    {
        Console.WriteLine(name + " is sleeping");
    }
}
```

28) Constructors

```
class Program
{
    static void Main(string[] args)
    {
        // constructor = A special method in a class
        //               Same name as the class name
        //               Can be used to assign arguments to fields when creating
an object

        Car car1 = new Car("Ford", "Mustang", 2022, "red");
        Car car2 = new Car("Chevy", "Corvette", 2021, "blue");

        car1.Drive();
        car2.Drive();

        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 void Drive()
    {
        Console.WriteLine("You drive the " + make + " " + model);
    }
}
```

29) Static Keyword

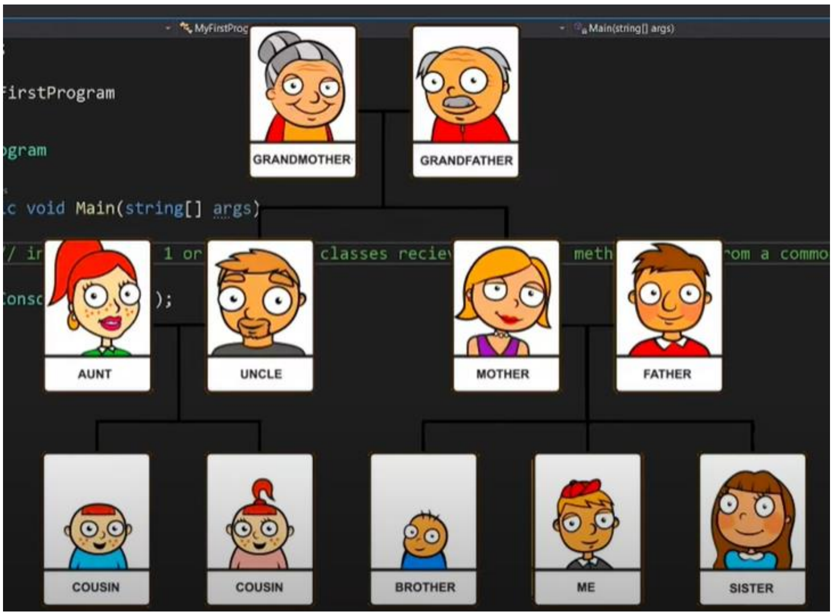
```
class Program
{
    static void Main(string[] args)
    {
        // static = modifier to declare a static member, which belongs to the
class itself
        //          rather than to any specific object

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

        Console.WriteLine(Car.numberOfCars);
        Car.StartRace();

        Console.ReadKey();
    }
}
```

```
class Car
{
    String model;
    public static int numberOfCars;

    public Car(String model)
    {
        this.model = model;
        numberOfCars++;
    }

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

30) Constructor Overloading

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

        Pizza pizza = new Pizza("stuffed crust", "red sauce", "mozzarella");

        Console.ReadKey();
    }
}
class Pizza
{
    String bread;
    String sauce;
    String cheese;
    String topping;

    public Pizza(String bread)
    {
        this.bread = bread;
    }
    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;
    }
}
```

31) Inheritance



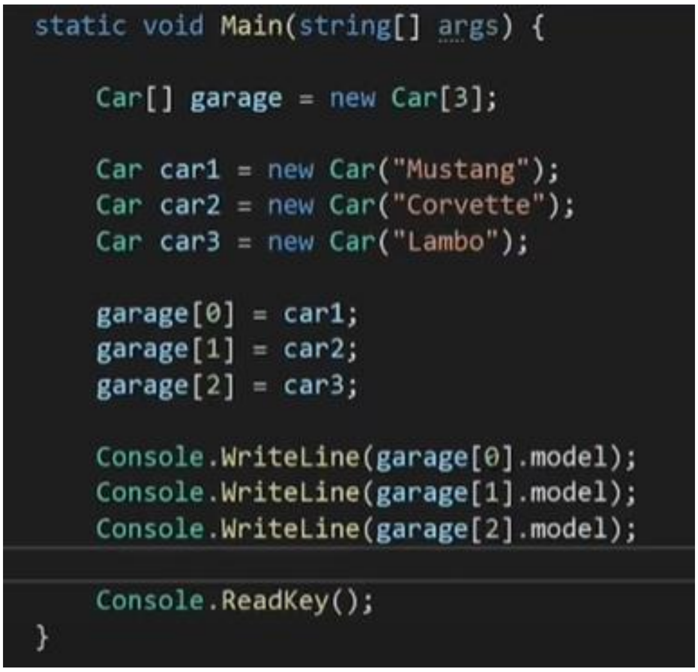
```
class Program
{
    static void Main(string[] args)
    {
        // inheritance = 1 or more child classes recieving fields, methods, etc.
from a common parent

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

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

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

        Console.WriteLine(boat.speed);
        Console.WriteLine(boat.wheels);
        boat.go();

        Console.ReadKey();
    }
}


class Vehicle
{
    public int speed = 0;

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

```
class Car : Vehicle
{
    public int wheels = 4;
}
class Bicycle : Vehicle
{
    public int wheels = 2;
}
class Boat : Vehicle
{
    public int wheels = 0;
}
```

32) Abstract Classes

```
class Program
{
    static void Main(string[] args)
    {
        // abstract classes =  modifier that indicates missing components or 
incomplete implementation

        Car car = new Car();
        Bicycle bicycle = new Bicycle();
        Boat boat = new Boat();
        //Vehicle vehicle = new Vehicle(); //can't create a vehicle object

        Console.ReadKey();
    }
}
abstract class Vehicle
{
    public int speed = 0;

    public void go()
    {
        Console.WriteLine("This vehicle is moving!");
    }
}
class Car : Vehicle
{
    public int wheels = 4;
    int maxSpeed = 500;
}
class Bicycle : Vehicle
{
    public int wheels = 2;
    int maxSpeed = 50;
}
class Boat : Vehicle
{
    public int wheels = 0;
    int maxSpeed = 100;
}
```

33) Arrays Of Objects



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

    Car[] garage = new Car[3];

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

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

    Console.WriteLine(garage[0].model);
    Console.WriteLine(garage[1].model);
    Console.WriteLine(garage[2].model);

    Console.ReadKey();
}
```

```
class Program
{
    static void Main(string[] args)
    {


        Car[] garage = { new Car("Mustang"), new Car("Corvette"), new
Car("Lambo") };

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

        Console.ReadKey();
    }
}
class Car
{
    public String model;

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

34) Object As Arguments

```
using System;

namespace MyFirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {

            Car car1 = new Car("Mustang", "red");

            Car car2 = Copy(car1);
            ChangeColor(car1, "Black");

            Console.WriteLine(car1.color + " " + car1.model);
            Console.WriteLine(car2.color + " " + car2.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;
        }
    }
}
```

35) Method Overriding

```
class Program
{
    static void Main(string[] args)
    {

//method overriding = provides a new version of a method inherited from a parent
//class

//inherited method must be: abstract, virtual, or already overriden
//Used with ToString(), polymorphism

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

        dog.Speak();
        cat.Speak();

        Console.ReadKey();
    }
}
class Animal
{
    public virtual void Speak()
    {
        Console.WriteLine("The animal goes *brrr*");
    }
}
class Dog : Animal
{
    public override void Speak()
    {
        Console.WriteLine("The dog goes *woof*");
    }
}
class Cat : Animal
{

}
```

36) ToString Method

```
class Program
{
    static void Main(string[] args)
    {

        //ToString() = converts an object to its string representation so that it
is suitable for display

        Car car = new Car("Chevy", "Corvette", 2022, "blue");

        Console.WriteLine(car.ToString());

        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;
    }
}
```

37) Polymorphism

```csharp
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();
        }

        Console.ReadKey();
    }
}
class Vehicle
{
    public virtual void Go()
    {

    }
}
class Car : Vehicle
{
    public override void Go()
    {
        Console.WriteLine("The car is moving!");
    }
}
class Bicycle : Vehicle
{
    public override void Go()
    {
        Console.WriteLine("The bicycle is moving!");
    }
}
class Boat : Vehicle
{
    public override void Go()
    {
        Console.WriteLine("The boat is moving!");
    }
}
```

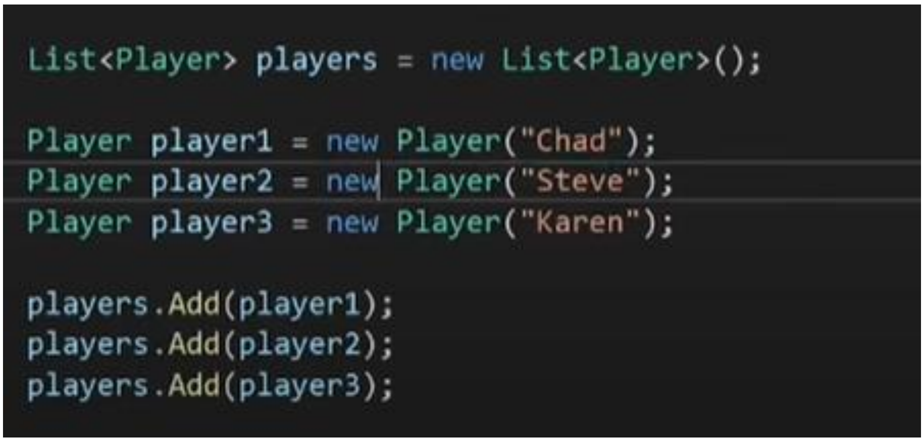
38) Interface

```
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"

        //              Benefit = 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()
        {
            Console.WriteLine("The rabbit runs away!");
        }
    }
    class Hawk : IPredator
    {
        public void Hunt()
        {
            Console.WriteLine("The hawk is searching for food!");
        }
    }
    class Fish : IPrey, IPredator
    {
        public void Flee()
        {
            Console.WriteLine("The fish swims away!");
        }
        public void Hunt()
        {
            Console.WriteLine("The fish is searching for smaller fish!");
        }
    }
}
```

39) Lists

```
using System;
using System.Collections.Generic;

namespace MyFirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {

            // List = data structure that represents a list of objects that can
be accessed by index.
            //        Similar to array, but can dynamically increase/decrease in
size
            //        using System.Collections.Generic;

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

            food.Add("pizza");
            food.Add("hamburger");
            food.Add("hotdog");
            food.Add("fries");

            //Console.WriteLine(food[0]);
            //Console.WriteLine(food[1]);
            //Console.WriteLine(food[2]);
            //Console.WriteLine(food[3]);

            //food.Remove("fries");
            //food.Insert(0, "sushi");
            //Console.WriteLine(food.Count);
            //Console.WriteLine(food.IndexOf("pizza"));
            //Console.WriteLine(food.LastIndexOf("fries"));
            //Console.WriteLine(food.Contains("pizza"));
            //food.Sort();
            //food.Reverse();
            //food.Clear();
            //String[] foodArray = food.ToArray();

            foreach (String item in food)
            {
                Console.WriteLine(item);
            }

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

40) List of objects



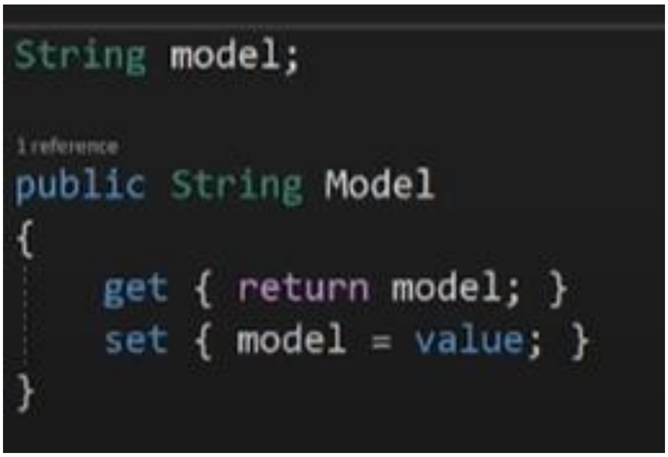
```
List<Player> players = new List<Player>();

Player player1 = new Player("Chad");
Player player2 = new Player("Steve");
Player player3 = new Player("Karen");

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

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

        players.Add(new Player("Chad"));
        players.Add(new Player("Steve"));
        players.Add(new Player("Karen"));

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

        Console.ReadKey();
    }
}

class Player
{
    public String username;

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

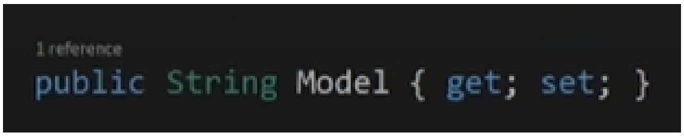
41) Getter and Setter to make things private

```
class Program
{
    static void Main(string[] args)
    {
        //getters & setters = add security to fields by encapsulation
        //                    They're accessors found within properties

        // properties = combine aspects of both fields and methods (share name 
with a field)
        // 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 
(parameter)

        Car car = new Car(400);

        car.Speed = 1000000000;

        Console.WriteLine(car.Speed);

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

}
```

42) Auto Implemented properties



```
String model;

public String Model
{
    get { return model; }
    set { model = value; }
}
```



```
public String Model { get; set; }
```

```
class Program
{
    static void Main(string[] args)
    {
        // auto-Implemented property =  shortcut when no additional logic is
required in the property
        //                                  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("Porsche");

        Console.WriteLine(car.Model);

        Console.ReadKey();
    }
}

class Car
{
    public String Model { get; set; }

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

43) Enums

```
class Program
{

    static void Main(string[] args)
    {
        // enums = special "class" that contains a set of named integer
constants.
        //          Use enums when you have values that you know will not change,
        //          To get the integer value from an item, you must explicitly
convert to an int

        //          name = integer

        //Console.WriteLine(Planets.Mercury + " is planet #" +
(int)Planets.Mercury);
        //Console.WriteLine(Planets.Pluto + " is planet #" + (int)Planets.Pluto);

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

        Console.WriteLine("planet: " + name);
        Console.WriteLine("radius: " + radius + "km");
        Console.WriteLine("volume: " + volume + "km^3");

        Console.ReadKey();
    }
    public static double Volume(PlanetRadius radius)
    {
        double volume = (4.0 / 3.0) * Math.PI * Math.Pow((int)radius, 3);
        return volume;
    }
}
enum Planets
{
    Mercury = 1,
    Venus = 2,
    Earth = 3,
    Mars = 4,
    Jupiter = 5,
    Saturn = 6,
    Uranus = 7,
    Neptune = 8,
    Pluto = 9
}

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

44) Generics

```
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 = { "1", "2", "3" };

        displayElements(intArray);
        displayElements(doubleArray);
        displayElements(stringArray);

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

45) Multithreading

```
using System;
using System.Threading;

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

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

           // Thread thread1 = new Thread(() => CountDown("Timer #1"));
           // Thread thread2 = new Thread(() => CountUp("Timer #2"));

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

            CountDown("Timer #1");
            CountUp("Timer #2");

            Console.WriteLine(mainThread.Name + " is complete!");

            Console.ReadKey();
        }
        public static void CountDown(String name)
        {
            for (int i = 10; i >= 0; i--)
            {
                Console.WriteLine("Timer #1 : " + i + " seconds");
                Thread.Sleep(1000);
            }
            Console.WriteLine("Timer #1 is complete!");
        }
        public static void CountUp(String name)
        {
            for (int i = 0; i <= 10; i++)
            {
                Console.WriteLine("Timer #2 : " + i + " seconds");
                Thread.Sleep(1000);
            }
            Console.WriteLine("Timer #2 is complete!");
        }
    }
}
```