**C# Basics**

* **Introduction to C#**:
  + C# is a modern, object-oriented programming language developed by Microsoft.
  + It is used to create Windows applications, web applications, and more.
  + Runs on the .NET Framework.
* **C# Syntax**:
  + Every C# application starts with a Main() method.
  + Curly braces {} define code blocks.
  + Each statement ends with a semicolon ;.

**C# Variables**

* **Variables**:
  + Variables are containers for storing data values.
  + Basic types include int, double, char, string, bool.
* **Data Types**:
  + Common data types include int, double, char, string, and bool.
  + int stores whole numbers, double stores decimals, string stores text, and bool stores true/false values.
* **Constants**:
  + const keyword is used to declare constants (unchangeable variables).

**C# Operators**

* **Arithmetic Operators**:
  + +, -, \*, /, %: Used for basic arithmetic operations.
* **Assignment Operators**:
  + =, +=, -=, \*=, /=: Used to assign values to variables.
* **Comparison Operators**:
  + ==, !=, >, <, >=, <=: Compare two values.
* **Logical Operators**:
  + && (AND), || (OR), ! (NOT): Used in conditions.

**C# Control Structures**

* **If/Else Statements**:
  + Used to execute code based on conditions: if, else if, else.
* **Switch Statement**:
  + Selects one of many code blocks to execute based on a condition.
* **Loops**:
  + for, while, do...while: Used to repeat a block of code.

**C# Methods**

* **Methods (Functions)**:
  + A method is a block of code that performs a specific task and can be reused.
  + Declared with a return type, method name, and parameters.
* **Method Parameters**:
  + Parameters can be passed to methods, and methods can return values.

**C# Object-Oriented Programming (OOP)**

* **Classes and Objects**:
  + A class defines the blueprint for objects.
  + An object is an instance of a class.
* **Constructors**:
  + Special methods used to initialize objects.
* **Inheritance**:
  + One class can inherit methods and properties from another class using the : syntax.
* **Polymorphism**:
  + Allows methods to be overridden in derived classes.
* **Encapsulation**:
  + Wrapping data and methods into a single unit (class) and restricting access to some of the object’s components.
* **Abstraction**:
  + Hides complex implementation details and shows only functionality.

**C# Arrays**

* **Arrays**:
  + Used to store multiple values in a single variable.
  + Arrays are zero-indexed, and each element can be accessed using its index.
* **Multidimensional Arrays**:
  + Arrays that contain arrays, allowing for matrices and other complex structures.

**C# Strings**

* **String Manipulation**:
  + Strings in C# are objects, offering various methods like Length, ToUpper(), ToLower(), Substring(), etc.
* **String Concatenation**:
  + Strings can be concatenated using + or String.Concat().

**C# Exception Handling**

* **Try/Catch**:
  + Used to handle errors or exceptions in code.
  + The try block contains the code that may throw an exception, and the catch block handles it.
* **Throwing Exceptions**:
  + Custom exceptions can be thrown using throw.

**C# File Handling**

* **File I/O**:
  + C# provides classes for file handling, such as File, StreamReader, and StreamWriter.
* **Reading from Files**:
  + Use StreamReader or File.ReadAllText().
* **Writing to Files**:
  + Use StreamWriter or File.WriteAllText().

**C# Collections**

* **Lists**:
  + Similar to arrays, but dynamic in size.
  + Defined using List<T>, where T is the type of elements.
* **Dictionaries**:
  + Stores key-value pairs, defined using Dictionary<TKey, TValue>.
* **HashSets**:
  + Collection of unique values.

**C# LINQ (Language Integrated Query)**

* **LINQ Overview**:
  + LINQ is a query syntax in C# used for querying collections like arrays and lists.
* **Basic LINQ Syntax**:
  + Uses query expressions to retrieve and manipulate data: from, where, select.

**C# Delegates and Events**

* **Delegates**:
  + Delegates are type-safe function pointers.
  + They can be used to pass methods as arguments to other methods.
* **Events**:
  + Events allow a class or object to notify other classes or objects when something occurs.

**C# Asynchronous Programming**

* **Async and Await**:
  + Asynchronous methods in C# are written using the async and await keywords.
  + Task is often used to represent asynchronous operations.

This is a simplified outline. For a more detailed explanation or additional examples, you can refer to the actual course on W3Schools. Would you like to dive deeper into any particular topic?

4o

**You said:**

this is good but give me each point code for example C# HOME C# Intro C# Get Started C# Syntax C# Output C# Comments C# Variables C# Data Types C# Type Casting C# User Input C# Operators C# Math C# Strings C# Booleans C# If...Else C# Switch C# While Loop C# For Loop C# Break/Continue C# Arrays C# Methods C# Methods C# Method Parameters C# Method Overloading C# Classes C# OOP C# Classes/Objects C# Class Members C# Constructors C# Access Modifiers C# Properties C# Inheritance C# Polymorphism C# Abstraction C# Interface C# Enums C# Files C# Exceptions

**ChatGPT said:**

ChatGPT

Sure! Here’s a brief summary with sample code for each point you mentioned:

**C# HOME**

csharp

Copy code

// Simple C# Program

using System;

class Program {

static void Main() {

Console.WriteLine("Hello, World!");

}

}

**C# Intro**

* C# is a modern, object-oriented programming language developed by Microsoft.
* It runs on the .NET framework and can be used to develop a variety of applications.

**C# Get Started**

csharp

Copy code

// To get started, install Visual Studio and create a new C# Console Application project.

**C# Syntax**

csharp

Copy code

using System;

class Program {

static void Main() {

Console.WriteLine("Hello, C# Syntax!");

}

}

**C# Output**

csharp

Copy code

Console.WriteLine("This is output in C#");

**C# Comments**

csharp

Copy code

// Single-line comment

/\* Multi-line

comment \*/

**C# Variables**

csharp

Copy code

int myNum = 5;

string myName = "John";

**C# Data Types**

csharp

Copy code

int age = 30; // Integer

double price = 9.99; // Floating-point

char grade = 'A'; // Character

bool isComplete = true; // Boolean

string name = "Alice"; // String

**C# Type Casting**

csharp

Copy code

int myInt = 10;

double myDouble = myInt; // Implicit casting

double anotherDouble = 9.78;

int myIntAgain = (int)anotherDouble; // Explicit casting

**C# User Input**

csharp

Copy code

Console.WriteLine("Enter your name:");

string userName = Console.ReadLine();

Console.WriteLine("Hello " + userName);

**C# Operators**

csharp

Copy code

int x = 5 + 3; // Addition

x += 2; // x is now 10

bool isEqual = (x == 10); // Comparison

**C# Math**

csharp

Copy code

int result = Math.Max(5, 10); // Max of two numbers

Console.WriteLine(result);

**C# Strings**

csharp

Copy code

string greeting = "Hello, World!";

Console.WriteLine(greeting.ToUpper());

Console.WriteLine(greeting.Substring(0, 5));

**C# Booleans**

csharp

Copy code

bool isTrue = true;

bool isFalse = false;

**C# If...Else**

csharp

Copy code

if (x > 5) {

Console.WriteLine("Greater than 5");

} else {

Console.WriteLine("Not greater than 5");

}

**C# Switch**

csharp

Copy code

int day = 3;

switch (day) {

case 1:

Console.WriteLine("Monday");

break;

case 3:

Console.WriteLine("Wednesday");

break;

default:

Console.WriteLine("Other day");

break;

}

**C# While Loop**

csharp

Copy code

int i = 0;

while (i < 5) {

Console.WriteLine(i);

i++;

}

**C# For Loop**

csharp

Copy code

for (int i = 0; i < 5; i++) {

Console.WriteLine(i);

}

**C# Break/Continue**

csharp

Copy code

for (int i = 0; i < 10; i++) {

if (i == 5) break; // Exit the loop

if (i % 2 == 0) continue; // Skip even numbers

Console.WriteLine(i);

}

**C# Arrays**

csharp

Copy code

int[] numbers = { 1, 2, 3, 4 };

Console.WriteLine(numbers[0]);

**C# Methods**

csharp

Copy code

static void Greet() {

Console.WriteLine("Hello from method");

}

**C# Method Parameters**

csharp

Copy code

static void Greet(string name) {

Console.WriteLine("Hello " + name);

}

**C# Method Overloading**

csharp

Copy code

static int Add(int a, int b) {

return a + b;

}

static double Add(double a, double b) {

return a + b;

}

**C# Classes**

csharp

Copy code

class Car {

public string model;

public Car(string modelName) {

model = modelName;

}

}

**C# OOP**

* Object-Oriented Programming (OOP) involves classes and objects, inheritance, encapsulation, polymorphism, and abstraction.

**C# Classes/Objects**

csharp

Copy code

Car myCar = new Car("Toyota");

Console.WriteLine(myCar.model);

**C# Class Members**

csharp

Copy code

class Person {

public string Name;

public void SayHello() {

Console.WriteLine("Hello " + Name);

}

}

**C# Constructors**

csharp

Copy code

class Car {

public string model;

public Car(string modelName) {

model = modelName;

}

}

**C# Access Modifiers**

csharp

Copy code

class Car {

private string model; // private access modifier

public string color; // public access modifier

}

**C# Properties**

csharp

Copy code

class Car {

private string model;

public string Model {

get { return model; }

set { model = value; }

}

}

**C# Inheritance**

csharp

Copy code

class Vehicle {

public string brand = "Ford";

}

class Car : Vehicle {

public string model = "Mustang";

}

**C# Polymorphism**

csharp

Copy code

class Animal {

public virtual void Sound() {

Console.WriteLine("Animal sound");

}

}

class Dog : Animal {

public override void Sound() {

Console.WriteLine("Bark");

}

}

**C# Abstraction**

csharp

Copy code

abstract class Animal {

public abstract void Sound();

}

class Dog : Animal {

public override void Sound() {

Console.WriteLine("Bark");

}

}

**C# Interface**

csharp

Copy code

interface IAnimal {

void Sound();

}

class Dog : IAnimal {

public void Sound() {

Console.WriteLine("Bark");

}

}

**C# Enums**

csharp

Copy code

enum Level {

Low,

Medium,

High

}

**C# Files**

csharp

Copy code

string text = System.IO.File.ReadAllText(@"file.txt");

System.IO.File.WriteAllText(@"file.txt", "Hello, world!");

**C# Exceptions**

csharp

Copy code

try {

int[] numbers = { 1, 2, 3 };

Console.WriteLine(numbers[5]);

} catch (IndexOutOfRangeException e) {

Console.WriteLine("Index out of range!");

}

**Entity Framework:**  
A green square with arrows and a green square with white text

Description automatically generated  
Why EF Core?   
It simplifies database access by allowing developers to work with data as objects, without writing SQL. EF Core lets you focus on your application’s logic, while it handles the database layer.  
  
Understanding ORM (Object-Relational Mapping): ORM bridges the gap between the relational database and object-oriented programming. EF Core maps your database tables to C# classes, making data manipulation intuitive and object-driven.  
  
Today, I also learned to:  
  
Create a simple Model and use Code First development to generate database tables from your C# models. This approach is fast and ensures that your models and database schema are always in sync.  
  
Install packages from NuGet Package Manager to set up EF Core. With a few commands, you’re ready to start connecting your app to a database.  
  
Use add-migration and update-database to create or modify database schema based on your models. This makes database versioning and schema management a breeze!  
  
Perform basic Insert, Update, and Delete operations on the database using EF Core. All done through familiar C# code—no SQL required!