# programming-fundamentals-csharp-dotnet

## **Introduction to C#**

**C#** (pronounced “C Sharp”) is a modern, easy-to-use, object-oriented programming language developed by the Microsoft Corporation. The first version of C# was released in 2002, and the latest version (10.0) was released in November 2021. C# is the most popular language used in Microsoft’s Visual Studio platform.

### Components of C#

C# uses the **.NET Platform**, which contains all components needed to run an application. One component is the **CLR**. The CLR handles various tasks, such as compiling code to 1s and 0s (binary) for the computer to understand.

Also within the .NET Platform is the new **.NET 6 framework** that contains developers’ shared code libraries when developing applications. This means that developers needn’t write new code from scratch.

For example, there’s already a built-in class for applications that read information from a text file, so there’s no need to recreate it.

An additional benefit to using C# is its versatility. C# is a general-purpose language, meaning that it can be used to create many kinds of applications.

Diagrama

El contenido generado por IA puede ser incorrecto.

### Examples of C# applications

Listed below are a few examples of apps created using the C# language.

* **Console Applications:** A console application is a text-based application that contains no graphical components like, buttons or images. This course focuses on text-based applications because they exemplify C# simply and clearly.
* **Websites:** Dynamic websites can be built using C# and ASP.NET in Visual Studio.
* **Mobile Applications:** iOS and Android applications can be created quickly using C# and .NET MAUI in Visual Studio.
* **Games:** Unity is a separate (free) software program used to design 2D, 3D, virtual, and mixed reality games. While the Unity GameEngine is used to design the game, the coding is still developed in Visual Studio with C#.
* **Computer Software:** Software for Windows can be created using WPF, Windows Forms, or .NET MAUI.

## Memory Types

Let's discuss memory types provided in C#.

### Heap and stack

When a variable is declared, a portion of memory is allocated in RAM. This portion of memory contains the variable’s name, data type, and value. Depending on the data type, the portion of memory allocated will be either stack or heap memory.

* The **stack** is used to track local variables and the program’s state.
* The **heap** is used to store data that can be accessed anytime and from anywhere in the program.

Data types are either value or reference types. The most common reference types are strings, arrays, and objects that are on the heap; everything else is a value type on the stack.

Diagrama

El contenido generado por IA puede ser incorrecto.

Reference types can be assigned a **null reference**, meaning that a reference can point to nothing at all. This is accomplished using the null keyword. We can’t assign null to ordinary value types.

### Boxing vs. unboxing

When data moves from value type to a reference type, it is called **boxing**, and the reverse is called **unboxing**.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Let’s look at the following illustration to better understand boxing and unboxing.

Diagrama

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

## Naming Conventions

Using proper naming and capitalization techniques can greatly increase code readability.

**Camel casing** and **pascal casing** are the two most popular capitalization techniques in C# when naming variables, classes, methods, and so on.

### Camel casing

The first letter of the first word is lowercase, and all subsequent words start with a capital letter.

int myFavoriteNumber = 7;

Use camel casing for local variables and arguments.

### Pascal casing

The first letter of every word is capitalized.

public void MyMethodName() { }

Pascal casing is used in most other situations except local variables and arguments.

### Tips and guides

Names shouldn’t include hyphens, spaces, non-alphanumeric characters, and they don’t start with a number.

The name should also be **self-documenting**. For example, instead of calling a variable variable1, which is meaningless, it should be scoreTotal, which describes what the variable is.

## Enumeration

### What is an enumeration?

An enum (enumeration) is a value type that consists of a collection of named constants. Using enums creates a modular design that enhances clarity and reduces the probability of invalid constants.

Enums are generally defined directly in the namespace (outside of any class).

### Example

In the following example, an enumeration is declared and the value is assigned. The output is displayed using the defined Output() method.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

### Example: enum value in an if statement

Enumeration values can be used to compare in an if statement. In the following example, the direction of the enumeration object is compared with the enumeration’s value.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

### Example: enum in a foreach loop

GetValues(type) returns the items of the enumeration that can be iterated with the foreach loop. The example below iterates the values of the enumeration with the foreach loop.

Texto

El contenido generado por IA puede ser incorrecto.

### Example: Assign enum numbers

By default, enum values start at 0. Numbers can be assigned to enum values, though.

Texto

El contenido generado por IA puede ser incorrecto.

## Directives

**Preprocessor directives** improve readability, reduce complexity, and help with program maintenance.

### The #region and #endregion directives

The #region directive is used to indicate a certain block of code. This keeps our code organized as blocks.

### #define, #if, #else, and #endif

* The #define directive must be at the top of the code.
* Texto

  El contenido generado por IA puede ser incorrecto.The #else statement in this section of code doesn’t compile, because DevMode is defined.

Texto

El contenido generado por IA puede ser incorrecto.

### Others

Below is a list of preprocessor directives:

**Conditional compilation:** #if, #else, #elif, #endif, #define, and #undef

**Other:** #warning, #error, #line, #region, #endregion, #pragma, #pragma warning, and #pragma checksum

## Namespaces and Using Directives

A **namespace** helps organize a large group of related code. To use a namespace, call it with the using directive or include it before the class name.

### Syntax

namespace NamespaceExample1Console

{

*// Empty Namespace*

}

*// In C# 10 file scoped namespaces were introduced. The namespace below does the same as the above namespace and will include everything in the file.*

namespace NamespaceExample2Console;

///////////////////////////////////////////////////////////

using System;

// C# 10 comes with ImplicitUsings, meaning the most popular using namespaces such as System are already built in so there is no need to declare them

Console.WriteLine("Hello World!");

// If “using System;” was not used, the fully qualified name must be used

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

/////////////////////////////////////////////////////////////////////

### Global in C#

**Global usings** were introduced in C# version 10. When the word global appears before the using directive, that using statement applies to the entire project. It’s common to create a GlobalUsings.cs file and include all the global usings in one location.

### Using static directive

The static members of a static class can be accessed without specifying the type name, so Console.WriteLine() can be simplified to WriteLine(). This is done with the using static directive.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

## Input and Output

string saveToVariable = Console.ReadLine();

//////////////////////////////////////////////////////////

Console.WriteLine("Adding a backslash before a quote \"Hello\"");

// Output: Adding a backslash before a quote "Hello"

### Escape characters

The @ symbol can be used before the text to make the entire string ignore escape characters. The technical name for this is the **verbatim string literal**. This is often helpful with file names that require many double backslashes, (\\).

Console.WriteLine("C:\\Users\\ComputerUser\\Desktop\\AFile.doc"); // with backslashes

Console.WriteLine(@"C:\Users\ComputerUser\Desktop\AFile.doc"); // with @ symbol

Escape characters can also be useful for starting a new line (\n), tab (\t), and many other options.

Console.WriteLine( "\tHello\nWorld" );

/\* Output

Hello

World

\*/

## Type Casting

### Implicit casting

short num1 = 50;

int num2 = 600;

int sum = num1 + num2; // num1 is implicitly cast to int

Console.WriteLine(sum); // Output: 650

### Explicit casting

decimal firstNumber = 50.1234m;

int secondNumber= 2;

// Explicitly cast firstNumber to int. Data was lost (.1234)

int sum = (int)firstNumber + secondNumber; // Output: 52

Console.WriteLine(sum); // Output: 52

### Parse

int exampleOneInt = int.Parse("6" ); // Parse a string to an int

Console.WriteLine(exampleOneInt + 1);

string exampleTwo = "7" ;

int exampleTwoInt = int.Parse(exampleTwo); // Parse a string to an int

Console.WriteLine(exampleTwoInt + 1);

string exampleThree ="6.5";

double exampleThreeInt = double.Parse(exampleThree);// Parse a string to a double

Console.WriteLine(exampleThreeInt + 1.56);

/////////////////////////////////////////////////////

/\*The program crash -> \*/

string exampleTwo = "Seven" ;

int exampleTwoInt = int.Parse(exampleTwo); // Parse a string to an int

Console.WriteLine(exampleTwoInt + 1);

### TryParse

The TryParse function is a safer approach because it checks if the value can convert to a numeric value. If TryParse can’t convert the value, it sets it to 0.

### TryParse using an if statement

The int.TryParse function returns a boolean value that can be used in an if statement to make a condition on it.

### TryParse using an inline out variable

Using the inline out variable in the approach below, declaring the textExampleInt beforehand is unnecessary.

string textExample = "10";

Console.WriteLine(textExample);

int.TryParse(textExample, out int textExampleInt); //Lo declara y lo inicializa

Console.WriteLine(textExampleInt);

## Ternary Operator

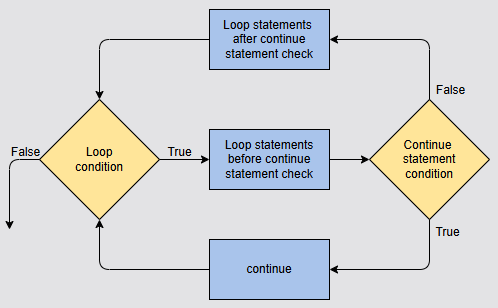
Condition ? First\_Result : Second\_Result

Example

string morningOrNight = (2 < 12) ? "Morning" : "Night"; // Morning is assigned

## What does a continue statement do?

There’s another keyword similar to break, called continue. Instead of exiting the loop, continue goes directly to the start of the loop to recheck the condition.



Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

## Array

string[] animals = { "Cat", "Dog" };

Console.WriteLine(animals[0]); // Output: Cat

Console.WriteLine(animals[1]); // Output: Dog

Console.WriteLine($"The animals are {animals[0]} and {animals[1]}.");

// Output: The animals are Cat and Dog.

/\***Example: Display all items of an array using a foreach loop**

\*/

string[] foods = { "Pizza", "Burger" };

foreach (string food in foods)

{

Console.WriteLine($"The food item is {food}.");

}

// Output: The food item is Pizza.

// Output: The food item is Burger.

### Sort and array length

int[] someNumbers = { 21, 22, 50, 3, 6 };

Array.Sort(someNumbers); // Sort the numbers in ascending order

// someNumbers.Length is the length of the array, in this case 5

for (int i = 0; i < someNumbers.Length; i++)

{

Console.Write($"{someNumbers[i]} "); // Output: 3 6 21 22 50

}

### 2-D arrays

// 2-D Array for a game map

// Place a comma between the brackets[,] to use a 2-D Array

string [,] myGameMap = new string [,]

{

{"Scary Room1", "Safe Room2", "Safe Room3"}, // Row 0

{"Dangerous Room4", "Safe Room5", "Safe Room6"}, // Row 1

{"Safe Room7", "Scary Room8", "Safe Room9"} // Row 2

};

// The first number is the row, the second number is the column

// Keep in mind that array values start at 0

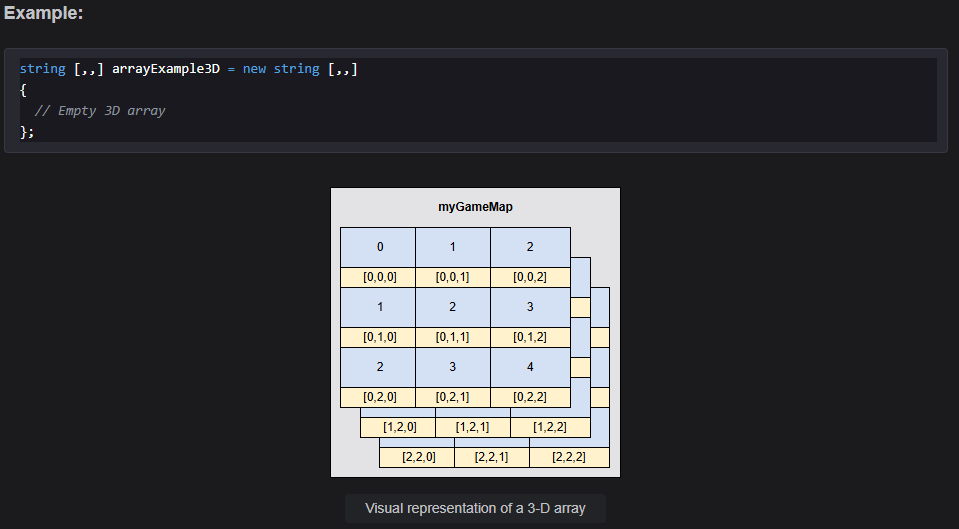
Console.WriteLine(myGameMap[0, 0]); // Output: Scary Room1

Console.WriteLine(myGameMap[1, 0]); // Output: Dangerous Room4

Console.WriteLine(myGameMap[2, 2]); // Output: Safe Room9

### 3-D arrays

While a 2-D array can be thought of as making a square with width and height (row and column), a 3-D array can be thought of as making a cube to include depth (row, column, and depth). When creating a 3-D array, add an additional comma between the square brackets to indicate that the 3-D array is being created.



### Jagged Array

A **jagged array** is when arrays of various sizes are put inside another array.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

## Methods

### Passing by reference

Arguments can be passed by **reference**, instead of by value. Pass references with the ref keyword. Changes made in a method for a reference affects the original value outside the method.

static void PlayerDamaged(ref int playerHealth)

{

playerHealth = playerHealth - 20;

}

int playerHealth = 100;

PlayerDamaged(ref playerHealth); // Pass playerHealth as reference parameter.

Console.WriteLine(playerHealth); // Output: 80

### named argument

Using **named arguments** helps clarify which parameters should be used when calling a method. Without named arguments, a long list of numbers can quickly become confusing.  
methodName(a:10, b:20)

**Named arguments are optional.**

Interfaz de usuario gráfica, Aplicación

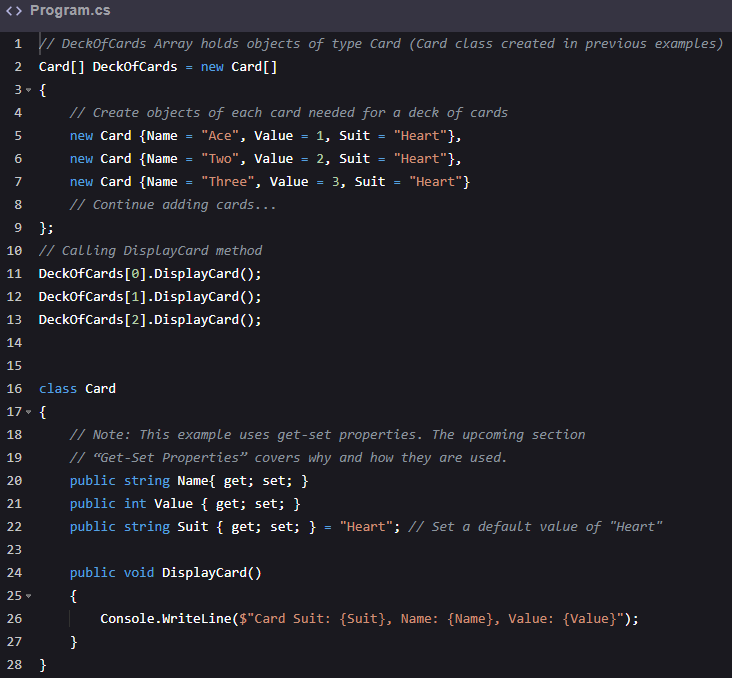
El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

## Classes

### Array of class objects



### Get-Set Properties

#### Texto El contenido generado por IA puede ser incorrecto.Auto-implemented property

### Class Inheritance

Classes can inherit all class features from another class using inheritance. With **class inheritance**, the derived class acquires all of the features from the base class. C# only supports single inheritance, meaning that it can only inherit one class at a time.

### Virtual Methods in Classes

To redefine a method in a *derived class*, we use a **virtual method**. Virtual methods are redefined using the virtual keyword in the *base class* and the override keyword in the *derived class*.

Texto

El contenido generado por IA puede ser incorrecto.

### What is an abstract class?

**Abstract classes** are commonly used for *inheritance* and can’t be *instantiated*. **Abstract methods** within an abstract class must be contained in the *child class* or the program won’t compile. Virtual methods are not required in *child classes*.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

### What is a partial class?

**Partial classes** can be split across two or more source files. Each file source contains a section of the class, and all parts are combined when compiled. The partial keyword is used for classes, structs, and interfaces, and means that other parts may be defined in the same *namespace*.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

### Access Modifier

**Access modifiers** control how information can be accessed. There are four types of access modifiers in C#.

1. Public. The elements defined with a public access modifier can be accessed by any other code in the same assembly or another assembly that references it.
2. Internal. The elements defined with an internal access modifier can be accessed by any other code in the same assembly. The internal modifier is the default if one isn’t specified.
3. Protected. The elements defined with a protected access modifier can be accessed only by the code in the same class or in a class that’s derived from it.
4. Private. The elements defined with a private access modifier can be accessed only by code in the same class. The private modifier is the default method modifier if one isn’t specified.

### Generic Class

Generics define type-safe classes without committing to any specific data types. They’re essentially a placeholder until a specified data type is declared. There are many prebuilt classes that use generics. This lesson covers how to create these generic classes.

Texto

El contenido generado por IA puede ser incorrecto.The syntax below demonstrates the use of the generic type parameter, which uses the angle brackets (<>). Generic type names are usually a single capital letter, T being the most common, or a simple name starting with T, such as TKey or TValue.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

### This

Texto

El contenido generado por IA puede ser incorrecto.

### What’s an extension method?

**Extension methods** are those that appear to belong to a class, but they don’t actually. Extension methods must be defined in static classes and be static methods.

public static Class otra\_clase

{

public static void Metodo\_extension (this metodo\_que\_extiende variable\_x)

}

Donde metodo\_que\_extiende es string en el siguiente ejemplo:

#### Ejemplo

Texto

El contenido generado por IA puede ser incorrecto.

#### Otro ejemplo con el metodo int

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

#### Explanation

**Line 4:** This line uses an extension method defined on line number 8. The extension method is called by numberPicked.LuckyNumber().

**Line 8:** This method is an extension to the int class.

### Interfaces

Texto

El contenido generado por IA puede ser incorrecto.An interface behaves as a contract between itself and any class where it’s used. A class that implements an interface must now implement all its members. These interfaces are usually named with a capital I, though it isn’t required. These interfaces can only contain declarations, and all members are implicitly abstract and public. Beginning with **C# version 8.0**, however, an interface may define default implementations for some or all of its members.

Texto

El contenido generado por IA puede ser incorrecto.

C# doesn’t allow for multiple inheritances in classes but multiple interfaces are allowed. If interfaces are added to a class that’s also inheriting from a base class, the interfaces must follow the base class.

public class DeckOfCards : ABaseClass, IInterface, IAnotherInterface

{

*// Empty Class*

}

#### The IEnumerable and IEnumerator

There are useful built-in interfaces, such as IEnumerable, IList, IDictionary, and IComparable. Adding the interface, IEnumerable, to a class means that it will iterate. The class must now also contain IEnumerator. In the example below, a class for a deck of cards is created and iterated over.

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

#### The yield return and yield break

The yield return and yield break are used when implementing an iterator (IEnumerable). The yield return statement returns the next element in the sequence, whereas yield break ends the iteration.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

## Data Structures

### What is a struct?

A struct is similar to a class and is useful for *lightweight objects*, such as a point, rectangle, or color. Lightweight objects can also be created with classes, but it’s often more memory efficient to use a struct. Unlike classes, structs are value types, not reference types.

Texto

El contenido generado por IA puede ser incorrecto.

A struct can be declared without using the new keyword. This is unique to structs, and wouldn’t work if it were a class. All value types (int, char, bool) are structs, which is why we can declare them without using the new keyword.

### What is a record?

A **record** is a type that makes it easier to create *immutable reference types* and provides *equality comparisons*.

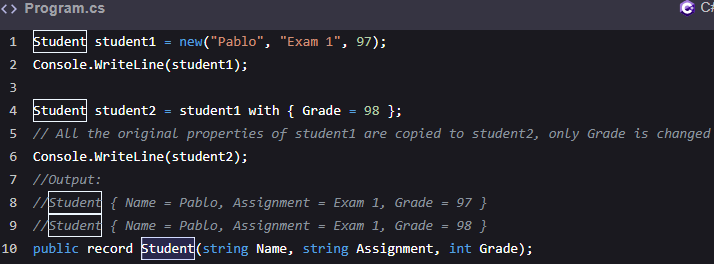
It’s important to point out that, when records are created in the above manner, they’re *immutable*, meaning that they cannot be changed.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

#### The with keyword

The with keyword can be used to create a copy of a record and change its properties as needed.



#### Equality

Records have valued-based equality built-in and includes a check that the types match.

Texto

El contenido generado por IA puede ser incorrecto.

#### Textual representation

The ToString() method is used for the textual representation of record objects. (es el mismo output)

#### Deconstruct

Records can also use the **deconstruct** method to separate the record into component properties.

Texto

El contenido generado por IA puede ser incorrecto.

#### Record with a method

A method can be added to the record class. These methods are similar to class methods and are called with record objects.

#### Inheritance

Records can *inherit* from other records.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

### List

List<int> evenNumbers = new() {12, 4, 26, 8, 10};

#### Insert and remove at a specific index

* The Insert(index, item) method is used to insert a specific index of the list. For example, to insert 12 at index 5, use the statement evenNumbers.Insert(5,12).
* The RemoveAt(index) method is used to remove an item at a specific index. For example, to remove at index 3, use the statement evenNumbers.RemoveAt(3).

evenNumbers.Sort(); // Sorting a list

evenNumbers.Clear(); // Removing all items of a list

// converting a list to an array

string[] foodsArray = foods.ToArray();

#### What’s a LinkedList?

A **LinkedList** is generally slower than a regular list, but it can be beneficial when adding or removing items in the middle of a list. As with lists, a LinkedList uses *generics*.

To use a LinkedList, add the following line of code:

using System.Collections.Generic; *// C# 10 comes with ImplicitUsings, meaning the most popular using namespaces such as this one is already built-in so there is no need to declare it*

Texto

El contenido generado por IA puede ser incorrecto.

### Dictionary

A **dictionary** consists of a *key* and a *value*. In an array, the key is created automatically (0, 1, and so on). With dictionaries, the key name can be set to something meaningful. As with lists, **dictionaries** use *generics*.

To use a dictionary, add the following line of code:

using System.Collections.Generic; *// C# 10 comes with ImplicitUsings, meaning the most popular using namespaces such as this one is already built-in so there is no need to declare it*

#### Example: Instantiate and assign items to a dictionary

Texto

El contenido generado por IA puede ser incorrecto.

#### Example: Update an item

We can use a *key* as the *index* to access or assign the value in the dictionary.

Console.WriteLine(myInventory["Computers"]); // Output: 2

#### Loop through a dictionary

To loop through a dictionary, use a ForEach loop and the keyword, KeyValuePair<TKey, TValue>. This is necessary, because dictionary elements are retrieved as KeyValuePair objects.

foreach (KeyValuePair<string, int> theItem in myInventory)

{

Console.WriteLine($"{theItem.Key}: quantity {theItem.Value} ");

}

#### Get the size of a dictionary

The Count keyword with the object name can be used to get the size of a dictionary (myInventory.Count).

#### Remove items from a dictionary

The Remove(item) method is used to remove from the dictionary (myInventory.Remove("Pens")).

#### Removing all items from a dictionary

The Clear() method is used to remove all elements of the dictionary (myInventory.Clear()).

### Tuples

A Tuple can be thought of much like an array, but a Tuple collection can contain different data types. Classes of the Tuple type are *generic containers* and can hold 1-8 items.

To access the information within a Tuple, use the name of the Tuple and then .Item1, .Item2, and so on.

Texto

El contenido generado por IA puede ser incorrecto.

#### Using a Tuple in a method

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

## Delegates

A delegate is a variable that can store a method and can then be passed around as needed. Previously, the variables discussed held data such as strings or objects. Methods can be treated the same way. In the example below, a delegate named MathExample can store methods that accept two int values and return a string value.

Texto

El contenido generado por IA puede ser incorrecto.

## Events

**Events** allow classes to notify other classes when something occurs. Graphical User Interfaces (GUI) applications often use events to indicate when something occurs. One example is when a button is pressed and another class is listening to handle that event. There are multiple components to an event. Each component is discussed with example syntax below. Following these syntaxes, there’s an example with all the components combined.

An **event** needs to be defined inside a class. Using the event keyword enables others to attach to this event. Specify the delegate type for the type of methods that can be attached to the event. In the example below, a predefined delegate called EventHandler is used. The EventHandler delegate returns void and has two parameters (object, EventArgs). The object is the sender (what sent the event) and the EventArgs object(contains basic information about the event).

### How to define an event

public event EventHandler WelcomeChanged; *// Define the event*

### How to raise an event

The code below verifies that an event handler is attached to an event. Methods that arise from events typically start with the word, On.

public void OnWelcomeChanged()

{

WelcomeChanged?.Invoke(this, EventArgs.Empty);

}

### How to handle the event

The method below specifies what to do when the change is detected. This method needs to be set up, based on the delegate used. In this case, it’s the EventHandler delegate.

public void HandleWelcomeChanged(object sender, EventArgs eventArgs)

{

Console.WriteLine( "The welcome message has changed!!)";

}

### How to attach an event handler to an event

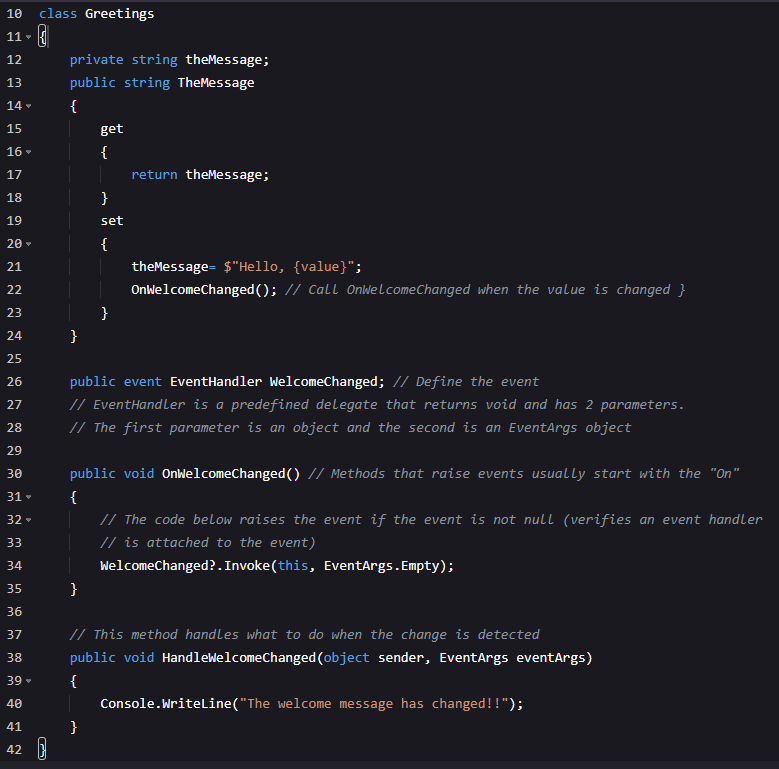
The += operator is used to attach an event handler to an event. We can attach multiple event handlers to an event if needed.

welcomeMessage.WelcomeChanged += welcmoeMessage.HandleWelcomeChanged;

### Ejemplo

Texto

El contenido generado por IA puede ser incorrecto.



## What is a lambda?

A **lambda** is essentially a method without a declaration; it uses clear and short syntax. A lambda can be written as a shorthand method directly in the place it’s intended to be used. This is especially useful on small methods that are used only once. The lambda operator is =>, and can be read as *goes to*. The lambda operator separates the expression into two parts (left side parameter, right side lambda body).

### Syntax

ANewNumber addFive = x => x + 5;

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

Ejemplo delegado vs sin delegado:

…Ejemplo delegado vs sin delegado arreglado

Captura de pantalla con letras

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

## Try catch

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

## Writing/Reading Text Files

Variables are temporary; once the program is turned off, the data is lost. **Writing and Reading** to a text file is a permanent way to handle data that needs to be saved when the program is turned off.

The StreamWriter class writes to text files. This class is contained in the System.IO namespace. To use this class, include the following at the top of the page.

using System.IO; *// C# 10 comes with ImplicitUsings, meaning the most popular using namespaces such as this one are already built-in so there is no need to declare it*

Texto

El contenido generado por IA puede ser incorrecto.

## Unit Testing

**Unit tests** verify that the logic of code is working as expected. Unit testing breaks a program down into small, testable, individual units.

### Why use unit tests?

For example, if we want to test a console-based application that has the following code:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

If we call this method now, it won’t provide us with the correct output.

Console.WriteLine(UTCalculator.CalculatorFeatures.AddTwoNumbers(10, 4)); // Output: 6

The code above runs because there are no syntax errors in it. There’s a logic error, however (10 + 4 should equal 14, not 6). A *unit test* is added to catch this type of error.

### How to write a unit test

Use the following steps to write a unit test.

#### Step 1:

Make a testing directory. In this example, we’re creating a directory named CalculatorTest in the parent directory of UTCalculator.

The UTCalculator directory is in the usercode directory. Use the following commands to navigate to the usercode directory and to create the testing directory.

cd usercode

mkdir CalculatorTest

#### Step 2:

Navigate to the newly-created directory using the *cd CalculatorTest* command and run the following command to create an **xUnit** that is used to write unit tests.

dotnet new xunit

This command takes some time to create an *xUnit test* template. The template has the following structure.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

We’ll add a project reference to CalculatorTest.csproj and modify the UnitTest1.cs file in the next steps.

##### Ejemplo en equipo

Cree el Proyecto en la carpeta ConsoleApp1 y el test en la que termina con \_Test

Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Ejecute en CMD el comando dotnet new xunit

Texto

El contenido generado por IA puede ser incorrecto.

Y se crearon los siguientes archivos dentro

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

#### Step 3:

The template for an *xUnit test* has now been created. Next, a reference needs to be added for the unit test to be connected to the **UTCalculator**. Use the dotnet add reference [address of the project file] command to add the reference.

In our example, we’ll run the following command:

dotnet add reference ../UTCalculator/Program.csproj.csproj

This command adds the **project reference** by adding an <ItemGroup> node to the CalculatorTest.csproj file:

<ItemGroup>

<ProjectReference Include="../UTCalculator/Program.csproj" />

</ItemGroup>

##### Ejemplo en equipo

El archivo estaba en diferente lugar a como lo decía arriba, para encontrarlo tuve que navegar entre las carpetas y lo encontré aquí:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

No se pudo con el archivo Program por lo que agregue el otro, el ConsoleApp1, este es el resultado de los intentos fallidos y el correcto con una flecha amarilla:

Texto

El contenido generado por IA puede ser incorrecto.

Para comprobarlo abrí el \_Test como proyecto en visual

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Y Como menciona en el step 3 esto creo las siguientes líneas en el proyecto de prueba:

Texto

El contenido generado por IA puede ser incorrecto.

#### Step 4:

Originally, the UnitTest1.cs file looks like the following:

using Xunit;

namespace CalulatorTest;

public class UnitTest1

{

[Fact]

public void Test1()

{

}

}

Make the following modifications to the UnitTest1.cs file.

1. First, add the using declaration for UTCalculator:

using UTCalculator; *// Needed to access UTCalculator class*

1. Next, modify the *method* below [Fact] to:

[Fact]

public void TestAddMethodResultShouldBeNine()

{

int result = CalculatorFeatures.AddTwoNumbers(5, 4);

Assert.Equal(9, result);

}

Unit test names should be self-documenting. In our example, the name TestAddMethodResultShouldBeNine describes exactly what’s expected.

The UnitTest1.cs will look like the following after the modification:

Texto

El contenido generado por IA puede ser incorrecto.

##### Ejemplo en equipo

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Quedó igual.

#### Step 5:

Run the test with the following command:

dotnet test

This command displays the test results. In the test, Assert.Equal(9, result) is used to compare the expected result with the answer the method response returns. If 9 matches the result when the test runs, it passes, otherwise it fails.

##### Ejemplo en equipo

Texto

El contenido generado por IA puede ser incorrecto.

En el primer intento se me olvido guardar los cambios en \_Test

Al guardar ya me arroja error:

Texto

El contenido generado por IA puede ser incorrecto.!!!

#### Running the unit test

Pressing the **“Run”** button executes all previous steps on runtime and shows the test results.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

The test will fail. An error message will say, Assert.Equal() Failure, Expected: 9 Actual: 1. This indicates a logical error in the AddTwoNumbers method.

It’s now clear that the numbers are being subtracted: int result = num1 - num2;. Change the minus sign to a plus sign, int result = num1 + num2; and run the test again to verify that all tests are passed.

### The Fact and Theory of unit testing

With **xUnit tests**, they’re marked either as [Fact] or [Theory]. **Fact** indicates a test, while **Theory** gives the ability to pass multiple data tests to the same unit test using *InlineData*. In the previous *unit test* example, the Assert class was utilized to compare results. The Assert class contains various methods that return a bool result. In the following examples, Assert.Equal is used.

#### How to write Theory unit tests

The steps for writing Theory unit testing is the same as Fact unit testing, explained earlier. The only change to make is in the UnitTest1.cs file. Add the [Theory] test block in place of the [Fact] test block. The Theory test block is as follows:

[Theory]

[InlineData(5, 4, 9)]

[InlineData(6, 1, 7)]

[InlineData(2, 3, 5)]

public void AddTwoNumbersAndGetResult(int firstNum, int secondNum, int expectedResult)

{

int result = CalculatorFeatures.AddTwoNumbers(firstNum, secondNum);

Assert.Equal(expectedResult, result);

}

Run the following code to see the results of the Theory unit test. Notice that all tests passed and the message including: Passed! - Failed: 0, Passed: 3, Skipped: 0, Total: 3, Duration: 9 ms.

#### Ejemplo en equipo

Texto

El contenido generado por IA puede ser incorrecto.

### En Visual Studio

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

## Random

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

## DateTime

### Assigning the current date and time with Now

DateTime todaysDate = DateTime.Now;

Console.WriteLine(todaysDate); // Output: 01/19/2022 08:18:24

// Will vary based on your current date/time

### Create a specific date

DateTime dob = new DateTime(1984, 1, 20); // (Year, Month, Day)

Console.WriteLine(dob); // Output: 1/20/1984 00:00:00 AM

### Create a specific date and time

DateTime dob = new DateTime(1984, 1, 20, 2, 30, 5); // (Year,Month,Day,Hour,Minute,Second)

Console.WriteLine(dob); // Output: 1/20/1984 2:30:05

### Display and format date or time

DateTime dob = new(1984, 1, 20, 2, 30, 5); // (Year,Month,Day,Hour,Minute,Second)

Console.WriteLine($"The Month is: {dob.Month}"); // Output: The Month is: 1

Console.WriteLine($"The Month is: {dob:MM}"); // Output: The Month is: 01

Console.WriteLine($"The Month is: {dob:MMM}"); // Output: The Month is: Jan

Console.WriteLine($"The Month is: {dob:MMMM}"); // Output: The Month is: January

Console.WriteLine($"The date is: {dob.Day}"); // Output: The date is: 20

Console.WriteLine($"The date is: {dob:d}"); // Output: The date is: 01/20/1984

Console.WriteLine($"The date is: {dob:D}"); // Output: The date is: Friday, January 20, 1984

Console.WriteLine($"The time is: {dob.TimeOfDay}"); // Output: The time is: 02:30:05

Console.WriteLine($"The time is: {dob:H:mm}"); // Output: The time is: 2:30

Console.WriteLine($"The time is: {dob:H:mm tt}"); // Output: The time is: 2:30 AM

### Add time

DateTime dob = new(1984, 1, 20, 2, 30, 5); // (Year,Month,Day,Hour,Minute,Second)

Console.WriteLine($"The Month: {dob.Month}"); // Output: The Month: 1

DateTime addMonthToDOB = dob.AddMonths(1); // Add a month

Console.WriteLine($"The Month: {addMonthToDOB.Month}"); // Output: The Month: 2

### Compare

DateTime firstDate = new(2017, 1, 23);

DateTime secondDate = new(2018, 1, 23);

int result = DateTime.Compare(firstDate, secondDate);

if (result < 0)

{

Console.WriteLine("The first date is earlier");

}

else if (result > 0)

{

Console.WriteLine("The first date is later");

}

else

{

Console.WriteLine("The dates are the same");

}

// Dates can also be compared without using the Compare method

if (firstDate < secondDate)

{

Console.WriteLine("The first date is earlier");

}

## Introduction to StringBuilder

When a string’s value is changed, a new string object is created in memory. This is because a string is *immutable* (it can’t be changed once created). This uses additional system resources. A **StringBuilder** is a *mutable* string-like object whose value changes based on the sequence of characters (a new string object in memory isn’t created when appended).

To use a StringBuilder, include the using statement:

using System.Text;

### Instantiate a StringBuilder

In the example below, an object of StringBuilder is created.

using System.Text;

StringBuilder theBuilder = new("Hello World!");

Console.WriteLine(theBuilder); // Output: "Hello World!"

### Append a StringBuilder

using System.Text;

StringBuilder theBuilder = new("Hello World!");

Console.WriteLine(theBuilder); // Output: "Hello World!"

theBuilder.Append(" Today is going to be great!");

Console.WriteLine(theBuilder); // Output: "Hello World! Today is going to be great!

### Insert a StringBuilder

using System.Text;

StringBuilder theBuilder = new("Hello how are you?");

Console.WriteLine(theBuilder); // Output: "Hello how are you?

theBuilder.Insert(5, " Jason,");

Console.WriteLine(theBuilder); // Output: "Hello Jason, how are you?

### Remove a StringBuilder at a specific index with a specific length

using System.Text;

StringBuilder theBuilder = new("Hello how are you?");

Console.WriteLine(theBuilder); // Output: "Hello how are you?"

theBuilder.Remove(5, 12);

Console.WriteLine(theBuilder); // Output: "Hello?"

### Replace a part of a StringBuilder

using System.Text;

StringBuilder theBuilder = new("Hello, how are you?");

Console.WriteLine(theBuilder); // Output: "Hello, how are you?"

theBuilder.Replace("Hello", "Jason");

Console.WriteLine(theBuilder); // Output: "Jason, how are you?"

### Replace an entire string

using System.Text;

StringBuilder theBuilder = new("Hello, how are you?");

Console.WriteLine(theBuilder); // Output: "Hello, how are you?"

theBuilder.Replace(theBuilder.ToString(), "Completely new text");

Console.WriteLine(theBuilder); // Output: "Completely new text."

## Threads

**Threads** let us run multiple sections of code simultaneously. Threading allows the code to run on multiple processors, boosting performance.

To use threads, include the following at the top of the page:

using System.Threading; *// C # 10 comes with ImplicitUsings, meaning the most popular using namespaces such as this one are already built-in so there is no need to declare it*

The Thread class constructor used below contains a delegate that accepts methods that return void. Only methods that return void can be used.

### How to create a Thread object

Thread aThread = new(CountTo200);

### How to start a Thread

In the example below, the Start method is used and can accept zero or one parameter.

aThread.Start(); *// 0 parameters*

bThread.Start(45); *// 1 parameter*

### Example: Threads with no parameter

Let’s have a look at the following example.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

This example uses the Start method to create two threads that begin executing the method, CountTo200. At this point, two threads run simultaneously. The results won’t output in sequence 1-200 each time. Both threads run at the same time and each outputs their numbers as their thread is run. For example, the results might be 1-70, and then the other thread could output 1-60. The first thread could then output 71-120, and so on.

### Example: Threads using the Join method

In the example below, the Join method instructs the original thread to wait until aThread is completed. The original thread is frozen until it finishes, essentially joining the two threads.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

### Threads with one parameter

The example below uses a parameter in the Start method. It’s important to note that the method parameter must be an object. The object can later be cast. In the example below, it’s cast from an object to an int.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

### Problem

This exercise uses Threads to display which Thread is being used. Create a console application for a Thread app that meets the requirements below.

1. Create three Threads.
2. Create a method that counts to 100 and displays which Thread is running.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

## Asynchronous

**Asynchronous programming** enhances the responsiveness of applications by using non-blocking operations to prevent bottlenecks that could slow down or freeze an application. The two main components of asynchronous programming are the keyword modifiers, async and await.

A method using the async modifier enables the use of the await operator, which now must be included at least once within the method. The original caller method continues when the await operator is reached, and the async method processes until it’s completed.

The async methods must have a return type of void, Task, Task<T>, or any other type that has a GetAwaiter method. The naming convention for async methods is to append them with an async suffix.

### The async void method

In the example below, the method, FirstMethodAsync(), contains a delay that would generally freeze a program for 3 seconds. Because this is an async method, the original caller of the method continues and the async method keeps processing when await is reached.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

### The async Task method

The async methods can return with void as demonstrated previously. They generally return with Task or Task<T Result>,however, which encapsulates information. A Task is used if there is no return statement, and Task<T Result> is used if there’s a return statement.

To use tasks, include the following at the top of the page:

using System.Threading.Tasks; *// C# 10 comes with ImplicitUsings, meaning the most popular using namespaces such as this one are already built-in so there is no need to declare it*

When the Task return type is used instead of void, the method call can now use await.

Texto

El contenido generado por IA puede ser incorrecto.

### The async Task <T Result> method

Texto

El contenido generado por IA puede ser incorrecto.

It’s very common to see async programming on graphical applications. For example, if a button is pressed and content from the internet needs to be downloaded, the whole program should not be frozen while it downloads. The program should download the content in the background while the user is still able to use the application.

## LINQ - Query Syntax

A **query expression** statement extracts specific information from a collection of data. The query syntax in C# is called **Language Integrated Query**, (**LINQ**). LINQs are used on collections of data, such as lists or arrays, which are container types for the IEnumerable<T> interface. Any container type of IEnumerable can use a LINQ.

To use a LINQ, include the following at the top of the code:

using system.Linq;*// C# 10 comes with ImplicitUsings, meaning the most popular using namespaces such as this one are already built-in so there is no need to declare it*

In the examples below, the various clauses that make up a query are covered. **Clauses** are a smaller syntax than a full expression; LINQs require multiple clauses to be a complete expression.

### What is the from clause?

Queries always start with a from clause. The from clause defines a source of information for the query and a range variable. A range variable is a local variable available throughout the query, like the variable in a foreach loop.

An example at the end of this lesson contains all classes and lists to make the following syntax functional.

**Syntax**

from o in students

### What is the select clause?

The select clause specifies the object or the part of the object to be used for the results of the query expression. The syntax below demonstrates selection of the entire object.

**Syntax**

select o;

The results from a LINQ always output as IEnumerable<T> and can be converted to a list or array if needed using ToList() or ToArray().

**Example**

IEnumerable <StudentInfo> aStudents = from o in students select o;

### What is the where clause?

Queries can only be created using from and select clauses; the where clause specifies which items to use.

**Syntax**

where o.Grade > 90

**Example**

IEnumerable <StudentInfo > aStudents = from o in students

where o.Grade > 90

select o;

### What is the let clause?

The example below uses the let clause that creates derived-range variables. A derived range variable uses the range variable created in the from clause. The let clause defines a variable to use in multiple places in the query or simplifies a complex operation. In the example below, select also selects a string instead of an object, and IEnumerable is modified to reflect this change.

**Example**

IEnumerable<string> studentAverages = from o in students

let averageGrade = (o.Grade1 + o.Grade2) / 2

select$"{o.Name} grade average:{averageGrade}";

### What is the join clause?

A join clause combines two collections based on a condition. In the example below, the collections are joined together when the StudentID values are equal in both collections. The left side of the equal sign must reference an earlier range variable and the right side references a new range variable; this is called an inner join.

**Example**

var inUseBooks = from o in students

join b in books on o.StudentID equals b.StudentID *// Inner join*

select new { o.Name, b.BookName }; *// New collection using these types*

### What is the OrderBy clause?

The OrderBy clause sorts all items in a collection. The default is ascending, but, they can also be specified as descending.

**Syntax**

orderby o.Name *// Ascending by default*

orderby o.Name descending *// Descending example*

orderby o.Name, o.StudentID descending*// Multilevel sort: Sort by name then StudentID*

**Example**

IEnumerable <StudentInfo > aStudents = from o in students

orderby o.Name *// Ascending by default*

select o;

### What is the group clause?

The group clause bundles elements into groups based on the information specified. In the example below, the books collection will have a group for each BookName. Previously, the select clause was needed to finish a full expression.The group clause is the other way to finish a full expression, however.

### Example:

var bookGroups = from o in books

group o by o.BookName;

In the example above, var is representing the syntax, IEnumerable<IGrouping <TKey, TElement>>. The TKey in this case will be a string for the book name and TElement is the class from which the collection was made.

### Example: Query expressions

Let’s execute the above queries in the following example.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

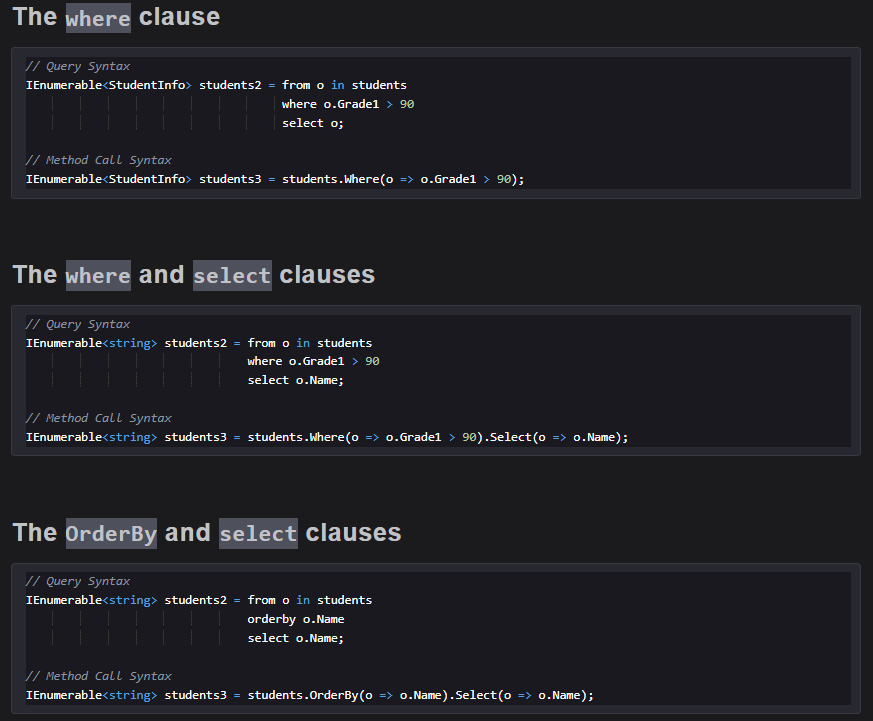
Texto

El contenido generado por IA puede ser incorrecto.

## LINQ Method Call Syntax

**Method call syntax** provides the same results as the query syntax covered in the previous lesson. All clauses previously covered are also available through these simple method invocations that use lambda expressions.

### Difference between query syntax and method call syntax



## Database: Connectivity

**Databases** store collections of information in an organized way that can easily be added, deleted, and updated.

### How to connect a database with C#

In the following example, a local SQL database is created to demonstrate **CRUD (Create, Read, Update, Delete)** in a database.

#### Step 1: Create the project

In the first step, we create a .NET Console Application. In the following, we’re using Program.cs as our project.

#### Step 2: Verify the Entity Frameworks

In the second step, we verify that the Entity Framework is installed. We require the following three Entity Frameworks:

1. Microsoft.EntityFrameworkCore 2 Microsoft.EntityFrameworkCore.Tools
2. Microsoft.EntityFrameworkCore.SQLServer

These frameworks can be installed using the following commands from CLI:

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Microsoft.EntityFrameworkCore.SQLServer

#### Step 3: Set up the Student class

In the next step, We create a folder named People to store and set up the model. In the created folder, we create a Student.cs file and add the following code into it.

namespace People

{

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public int Grade { get; set; }

}

}

#### Step 4: Set up the database context

In this step, we create an additional folder called Data for the project and a class in the Data folder called StudentDBContext. This class manages the connection to the database. The class, StudentDBContext, needs to inherit from the Entity Framework class, DBContext.

* Update the class to class StudentDBContext : DbContext.
* To use DBContext, add using Microsoft.EntityFrameworkCore; to the top of the page.
* To handle the information from the Student class model previously created, the DbSet is used public DbSet <Student> Students { get; set; }.
* An additional using statement, using People;, is also added to access the Student class.
* A connection string is added to connect to the database.

The final code of the StudentDBContext.cs should look like the following:

using Microsoft.EntityFrameworkCore;

using People;

namespace Data

{

internal class StudentDBContext : DbContext

{

public DbSet<Student> Students { get; set; }

*// Connect to database*

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Server=localhost; Database=StudentDatabase; User Id=sa; Password=Educative@123;");

}

}

}

#### Step 5: Use migrations for the database

In this step, we use migrations to create the database. The purpose of migration is to keep the database in sync with the models.

We use the following CLI commands to create a migration:

dotnet ef migrations add InitialMigration

After pressing “Enter”, notice that a new folder called Migrations was added to the project. This folder contains the schema and other information the database needs.

The database isn’t created yet; the following command creates the database.

dotnet ef database update

If these commands execute without any error, the database has been created and we can communicate with the database.

### En Visual Studio

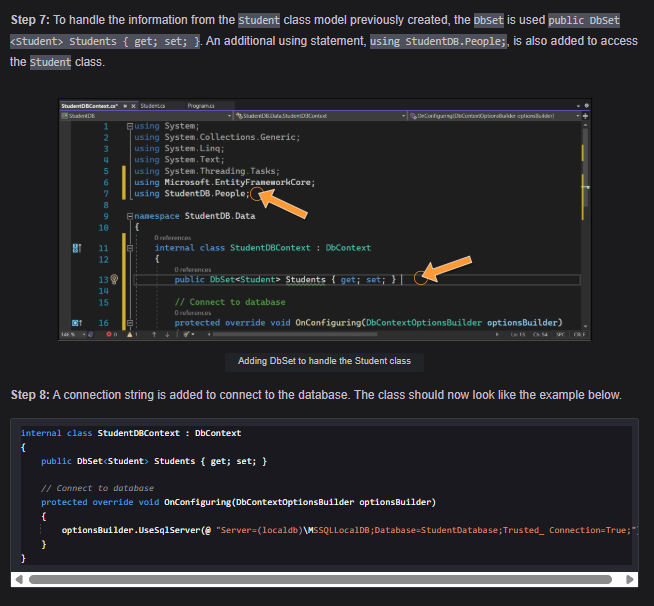
Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.



Texto

El contenido generado por IA puede ser incorrecto.



Texto

El contenido generado por IA puede ser incorrecto.