C# FOR BEGINNERS

**C# vs NET**

**C#** es un lenguaje de programación.

**NET** es un framework para construir aplicaciones en Windows. Consta de 2 componentes:

* CLR (Common Language Runtime): es una aplicación que se encuentra en la memoria, cuyo trabajo es traducir el código dentro del código de la máquina. Esto se le llama **Just-in-Time Compilation (JIT)**.
* Class library.

**OUR FIRST C# APPLICATION**

Creamos un proyecto de aplicación de consola, y con **ctrl F5** activamos la consola.

Un **namespace** es un contendor para las **class**.

Un **assembly** es un archivo, en forma de ejecutable o DLL, que contiene uno o más namespaces y classes.

**VARIABLES and CONSTANTS**

Para declarar variables primero necesitamos pasarle que tipo de datos es, luego el nombre de la variable y por último su valor.

Para declarar variables constantes, necesitamos utilizar la palabra clave **const**, el tipo de dato, el nombre y luego el valor.

**int** number = 1;

**const float** pi = 3.14;

**Identificadores:**

* No pueden empezar con un número: **1route** (mal), **oneRoute** (bien).
* No pueden incluir un espacio en blanco: **first name**(mal), **firstName**(bien).
* No puede ser una palabra clave reservada: **int** (mal), podemos usar una arroba en cambio **@int.**
* Usar nombres descriptivos: **fn** (mal), **firstName** (bien).

**Datos primitivos:**

* Byte.
* Short.
* Int.
* Long.
* Float.
* Double.
* Decimal.
* Char.
* Bool.

**Datos no primitivos:**

* String.
* Array.
* Enum.
* Class.

**OVERFLOWING**

Es cuando excedemos el límite del tipo de datos de bytes en C# por defecto. Por ejemplo:

**byte** number = 255;

number = number + 1; **// 0**

El resultado de esto es 0 porque excedimos el tipo de datos en bytes.

Para evitarlo podemos usar **checked**: esto nos va a dar un error para evitar que todo se rompa.

**Checked** {

**byte** number = 255;

number = number + 1;

}

**DEMO: Variables and Constants**

Para compilar la aplicación vamos a utilizar **Ctrl + Shift + B**.

Para correr la aplicación vamos a utilizar: **Ctrl + F5**.

byte number = 2;

int count = 10;

float totalPrice = 20.95f;

char character = 'a';

string firstName = "Joaquin";

bool isWorking = true;

Podemos cambiar todos los tipos de datos por **var** y los va a reconocer igual:

var number = 2;

var count = 10;

var totalPrice = 20.95f;

var character = 'a';

var firstName = "Joaquin";

var isWorking = true;

**Template:**

Console.WriteLine("{0} {1}", byte.MinValue, byte.MaxValue);

Da como resultado el valor mínimo y máximo de un byte.

**TYPE CONVERSION**

**Implicit Type Conversion:** si esto estaría al revés se perdería data y no se puede copilar.

byte b = 1;

int i = b;

**Explicit Type Conversion:** acá debemos especificarle al copilador que estamos advertidos de que se puede perder data, pero lo hacemos igualmente.

Float f = 1.0f;

Int i = (int)f;

**Non-compatible types:**

String s = “1”;

Int i = **Convert.toInt32(**s**);**

Int j = int.Parse(s);

**Convert:**

* ToByte().
* ToInt16().
* ToInt32().
* ToInt64().

**Try and Catch:** si queremos pasar un string por ejemplo a un byte y este string es más grande de lo que puede soportar el dato bytes (255) hay que hacer un bloque try and catch.

try

{

var number = "1234";

byte b = Convert.ToByte(number);

Console.WriteLine(b);

}

catch (Exception)

{

Console.WriteLine("No se puede convertir en un byte.");

}

**OPERATORS**

Los mismos operadores que en JavaScript.

var a = 10;

var b = 3;

Console.WriteLine((float)a / (float)b);//3,333

**NON-PRIMITIVES TYPES**

**CLASSES**

Para crear una clase utilizamos un modificador de acceso(**public**), la palabra reservada **class** y el nombre en mayúscula que elijamos.

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

}

Si queremos acceder al método **Add** de la **class** Calculator, podemos especificarle un **static**:

Public **static** int Add(){}

Y luego accedemos a ella sin tener que crear un objeto:

Int result = Calculator.Add(1, 2);

Podemos crear una carpeta que almacene las clases dependiendo su característica, como puede ser una carpeta llamada **Math** que contenga una **class** llamada Calculator, luego debemos crear ese archivo dentro de la carpeta que contenga la clase.

**Archivo calculator dentro de Math:**

namespace HelloWorld.Math

{

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

}

}

**Dentro del archivo principal:**

using HelloWorld.Math;

Calculator calculator = new Calculator();

var result = calculator.Add(1, 2);

Console.WriteLine(result);

**STRUCTS**

Para declarar un **struct** es igual que una **class** pero cambiando estas palabras claves.

Utilicémoslo cuando necesitemos crear cosas sencillas.

**ARRAYS**

int[] numbers = new int[3]; //{1, 2, 3};//int[3] significa cuantos elementos le pasamos al array

numbers[0] = 1;

numbers[1] = 2;

numbers[2] = 3;

Podemos pasarle los datos de dos formas con las curly braces después de decirle cuantos elementos tiene, o numbers[0] = 1, por ejemplo.

**STRINGS**

**String Format:**

var firstName = "Joaquín";

var lastName = "Caggiano";

var fullName = string.Format("My name is {0} {1}", firstName, lastName);

Console.WriteLine(fullName);

**Creating Strings:**

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

string list = string.Join(",", numbers);

**Escape characters:**

|  |  |
| --- | --- |
| **Char** | **Description** |
| \n | New line |
| \t | Tab |
| \ \ | Backslash |
| \ ' | Single Quotation mark |
| \” | Double Quotation mark |

**Verbating String:** utilizar el @.

var text = @"Hola Joaquín

Mira las siguientes rutas:

c:\folder1\folder2

c:\folder3\folder4";

Console.WriteLine(text);

**ENUMS**

Es un tipo de dato que representa un conjunto de constantes de valores nominales. Internamente es un **integer**.

Siempre es mejor setearles valores desde el principio, sino el primer valor será cero e ira incrementando.

public enum ShippingMethod

{

RegularAirMail = 1,

RegisterAirMail = 2,

Express = 3,

}

class Program

{

static void Main(string[] args)

{

var method = ShippingMethod.Express;

Console.WriteLine((int)method);//3

var methodId = 3;

Console.WriteLine((ShippingMethod)methodId);//Express

Console.WriteLine(method.ToString());//Express -transformando un int en string

//Convertir un string en un enum

var methodName = "Express";

var shippingMethod = (ShippingMethod)Enum.Parse(typeof(ShippingMethod), methodName);

Console.WriteLine(shippingMethod);

}

}

**REFERENCE TYPES and VALUES TYPES**

Los **structures** son tipos de valores, y las **classes** son tipos de referencia.

Acá a y b se almacenan en distintos lugares de la memoria, y producen una copia de los mismo sin afectar al valor del otro:

class Program

{

static void Main(string[] args)

{

var a = 10;

var b = a;

b++;

Console.WriteLine(String.Format("a: {0}, b: {1}", a, b));//10, 11

}

}

En este segundo caso estamos copiando un array sobre otro, pero en realidad no se copian, porque el array2 va hacia la misma referencia de donde se guardo el array1 en la pila, por lo que al modificar uno, se modifican ambos, NO ES UNA COPIA DE VERDAD:

var array1 = new int[3] { 1, 2, 3 };

var array2 = array1;

array2[0] = 0;

Console.WriteLine(array1[0]);//0

Console.WriteLine(array2[0]);//0

**CONTROL FLOW**

**ITERATION STATEMENTS**

**Foreach:**

string name = "Joaquín";

foreach (var character in name)

{

Console.WriteLine(character);

}

**While Loop:** utilizamos este loop cuando no sabemos cuantas veces se va a producir la iteración.

var i = 0;

while(i <= 10)

{

if (i % 2 == 0)

{

Console.WriteLine(i);

}

i++;

}

while (true)

{

Console.Write("Type your name: ");

var input = Console.ReadLine();

if (String.IsNullOrWhiteSpace(input))

{

break;

}

Console.WriteLine("@Echo: " + input);

}

Otra forma de hacerlo es:

while (true)

{

Console.Write("Type your name: ");

var input = Console.ReadLine();

if (!String.IsNullOrWhiteSpace(input))

{

Console.WriteLine("@Echo: " + input);

continue; //continua desde el inicio del loop

}

break;

}

**RANDOM CLASS**

Genera números aleatorios.

var random = new Random();

for (var i = 0; i < 10; i++)

{

Console.WriteLine(random.Next(1, 10)); //le decimos que queremos n entre 1 y 10

}

**ARRAYS and LISTS**

**ARRAY**

**Single Dimension Array:**

var numbers = new int[5] { 1, 2, 3, 4, 5 };

**Multi Dimension Array:**

Tenemos dos tipos:

* Rectangular: cada fila tiene el número exacto de columnas.

var matrix = new int[3, 5]

{

{1,2,3,4,5},

{6,7,8,9,10},

{12,13,14,15,16},

};

* Jagged: el número de columnas en cada fila puede ser diferente. Un array de arrays.

var matrix = new int[3][];

matrix[0] = new int[4];

matrix[1] = new int[5];

matrix[2] = new int[3];

matrix[0][0] = 1;

**MÉTODOS DE LOS ARRAYS**

var numbers = new int[] { 2,7,5,1,8,4 };

//Length

Console.WriteLine("Length: " + numbers.Length);

//IndexOf

var index = Array.IndexOf(numbers, 2);

Console.WriteLine("Index of 2: " + index);

//Clear

Array.Clear(numbers, 0, 2);

Console.WriteLine("Effect of clear()");

foreach (var number in numbers)

{

Console.WriteLine(number);

}

//Copy

var anotherArray = new int[3];

Array.Copy(numbers, anotherArray, 3);

Console.WriteLine("Effect of copy()");

foreach (var n in anotherArray)

{

Console.WriteLine(n);

}

//Sort - ordena el array

Array.Sort(numbers);

Console.WriteLine("Effect of sort()");

foreach (var num in numbers)

{

Console.WriteLine(num);

}

//Reverse

Array.Reverse(numbers);

Console.WriteLine("Effect of reverse()");

foreach (var numb in numbers)

{

Console.WriteLine(numb);

}

**LISTS**

Utilizamos un **list** cuando no sabemos de antemano cuantos objetos vamos a almacenar en esa lista.

var numbers = new List<int>();

Dentro de <> debemos especificar que tipo de datos va a tener nuestro **list**.

**MÉTODOS de las LISTS**

var numbers = new List<int>() { 1,2,3,4};

//Add

numbers.Add(1);

//AddRange

numbers.AddRange(new int[3] {6,7,8});

foreach (var number in numbers)

{

Console.WriteLine(number);

}

//IndexOf - LastIndexOf

Console.WriteLine("Index of 1: " + numbers.IndexOf(1));

Console.WriteLine("Las Index of 1: " + numbers.LastIndexOf(1));

//Count

Console.WriteLine("Count: " + numbers.Count);

//Remove

numbers.Remove(1);

Console.WriteLine("Remove 1:");

foreach (var n in numbers)

{

Console.WriteLine(n);

}

//Clear - elimina todos los elementos

numbers.Clear();

Console.WriteLine("Count: " + numbers.Count);

**WORKING with DATES**

**DATETIME**

var dateTime = new DateTime(2022, 6, 13);

var now = DateTime.Now;

var today = DateTime.Today;

Console.WriteLine("Hour: " + now.Hour);

Console.WriteLine("Minute: " + now.Minute);

var tomorrow = now.AddDays(1);

var yesterday = now.AddDays(-1);

Console.WriteLine(now.ToLongDateString());

Console.WriteLine(now.ToShortDateString());

Console.WriteLine(now.ToLongTimeString());

Console.WriteLine(now.ToShortTimeString());

Console.WriteLine(now.ToString());

Console.WriteLine(now.ToString("yyyy-MM-dd HH:mm"));

**TIMESPAN**

Representa un período de tiempo.

//Creating

var timeSpan = new TimeSpan(1, 2, 3);//hora-minutos-segundos

var timeSpanHour = TimeSpan.FromHours(1);

var start = DateTime.Now;

var end = DateTime.Now.AddMinutes(2);

var duration = end - start;

Console.WriteLine("Duration: " + duration);

//Properties

Console.WriteLine("Minutes: " + timeSpan.Minutes);//2

Console.WriteLine("Total Minutes: " + timeSpan.TotalMinutes);//62.05

//Add

Console.WriteLine("Add Example: " + timeSpan.Add(TimeSpan.FromMinutes(8)));//01:10:03

Console.WriteLine("Subtract Example: " + timeSpan.Subtract(TimeSpan.FromMinutes(2)));//01:00:03

//ToString

Console.WriteLine("ToString: " + timeSpan.ToString());

//Parse

Console.WriteLine("Parse: " + TimeSpan.Parse("01:02:03"));

**WORKING with TEXT**

**STRING**

**Format String:**

|  |  |  |
| --- | --- | --- |
| **Format Specifier** | **Description** | **Example** |
| **c** or **C** | Currency | 123456 (C) -> $123,456 |
| **d** or **D** | Decimal | 1234(D) -> 001234 |
| **e** or **E** | Exponential | 1052.0329112756 (E) -> 1.052033E+003 |
| **f** or **F** | Fixed Point | 1234.567(F1) -> 1234.5 |
| **x** or **X** | Hexadecimal | 255 (X) -> FF |

**STRING METHODS**

var fullName = "Joaquín Caggiano ";

//Trim

Console.WriteLine("Trim: '{0}'", fullName.Trim());

//ToUpper

Console.WriteLine("Upper: '{0}'", fullName.Trim().ToUpper());

//IndexOf - Substring

var index = fullName.IndexOf(' ');

var firstName = fullName.Substring(0, index);

var lastName = fullName.Substring(index + 1);

Console.WriteLine("Firs Name: " + firstName);

Console.WriteLine("Last Name: " + lastName);

//Split

var names = fullName.Split(" ");

Console.WriteLine("Firs Name Split: " + names[0]);

Console.WriteLine("Last Name Split: " + names[1]);

//Replace

Console.WriteLine("Replace lastname: " + fullName.Replace("Caggiano", "Serra"));

//IsNullOrWhiteSpace

if(String.IsNullOrWhiteSpace(" "))

{

Console.WriteLine("Invalid");

}

//Convert string to number and viceversa

var str = "25";

var age = Convert.ToByte(str);

Console.WriteLine(age);

float price = 29.95f;

Console.WriteLine(price.ToString("C"));//$29,95

Console.WriteLine(price.ToString("C0"));//$30

**SUMMARISING TEXT**

var sentence = "This is going to be a really realy really long text";

var summary = SummerizeText(sentence, 25);

Console.WriteLine(summary);

static string SummerizeText (string text, int maxLength = 20)

{

if (text.Length < maxLength)

{

return text;

}

else

{

var words = text.Split(' ');

var totalCharacters = 0;

var summaryWords = new List<string>();

foreach (var word in words)

{

summaryWords.Add(word);

totalCharacters += word.Length + 1;

if (totalCharacters > maxLength)

{

break;

}

}

var summary = String.Join(" ", summaryWords) + "...";

return summary;

}

}

**STRING BUILDER**

Para usar esta clase debemos importar **using System.Text**.

Tenemos los siguientes métodos:

* Append.
* AppendLine: para bajar una línea.
* Replace: argumento 1 el que queremos cambiar, argumento 2 por el que lo vamos a cambiar.
* Remove: argumento 1 donde esta el elemento a remover, argumento 2 hasta donde vamos a remover.
* Insert: arg1 ubicación donde vamos a insertar, arg2 puede ser un new string().

**PROCEDURAL PROGRAMMING**

static void Main(string[] args)

{

Console.Write("What is your name?");

var name = Console.ReadLine();

var reversed = ReverseName(name);

Console.WriteLine("Reversed name: " + reversed);

}

public static string ReverseName (string name)

{

var array = new char[name.Length];

for (var i = name.Length; i > 0; i--)

{

array[name.Length - i] = name[i - 1];

}

return new string(array);

}

**SEC-9: WORKING with FILES**

**SYSTEM.IO NAMESPACES**

**File, FileInfo:**

Provee métodos para crear, copiar, borrar, mover, y abrir archivos.

La diferencia entre ambas es que:

* FileInfo provee la instancia de métodos.
* File: provee métodos estáticos.

Sus métodos son:

* Create().
* Copy().
* Delete().
* Exists().
* GetAttributes().
* Move().
* ReadAllText().

//FILE

var path = @"c:\somefile.jpg";

//Copy

File.Copy(@"c:\temp\myfile.jpg", @"d:\temp\myfile.jpg", true);

//Delete

File.Delete(path);

//Exists

if (File.Exists(path))

{

//

}

//ReadAllText

var content = File.ReadAllText(path);

//FILEINFO

var fileInfo = new FileInfo(path);

//CopyTo

fileInfo.CopyTo("...");

//Delete

fileInfo.Delete();

//Exists

if (fileInfo.Exists)

{

//

}

**Directory, DirectoryInfo:**

* Directory: provee métodos estáticos.
* DirectoryInfo: provee la instancia de métodos.

Sus métodos son:

* CreateDirectory().
* Delete().
* Exists().
* GetCurrentDirectory(): retorna donde la aplicación esta actualmente en sesión.
* GetFiles().
* Move().
* GetLogicalDrives(): retorna las unidades lógicas de su disco duro.

//DIRECTORY

//CreateDirectory

Directory.CreateDirectory(@"c:\temp\folder1");

//GetFiles

var files = Directory.GetFiles(@"c:\temp\folder1\", "\*.txt", SearchOption.AllDirectories);

foreach (var file in files)

{

Console.WriteLine(file);

}

//GetDirectories

var directories = Directory.GetDirectories(@"c:\temp\folder1", "\*.\*", SearchOption.AllDirectories);

foreach (var directory in directories)

{

Console.WriteLine(directory);

}

//Exists

Directory.Exists("...");

//DIRECTORYINFO

var directoryInfo = new DirectoryInfo("...");

directoryInfo.GetFiles();

directoryInfo.GetDirectories();

**Path:**

Provee métodos para trabajar con strings que contienen archivos o directorios de ruta.

Sus métodos son:

* GetDirectoryName().
* GetFileName().
* GetExtension().
* GetTempPath().

var path = @"c:\temp\folder1\nada.txt";

//GetExtension

Console.WriteLine("Extension: " + Path.GetExtension(path));//.txt

//GetFileName

Console.WriteLine("FileName: " + Path.GetFileName(path));//nada.txt

//GetFileName

Console.WriteLine("FileName without extension: " + Path.GetFileNameWithoutExtension(path));//nada

//GetDirectoryName

Console.WriteLine("Directory name: " + Path.GetDirectoryName(path));//c:\temp\folder1

**SEC-10: DEBUGGING APPLICATIONS**

**DEBUGGING TOOLS in VISUAL STUDIO**

Presionando **F9** colocamos donde queremos un breakpoint.

Con **F5** corremos la aplicación en el modo debug.

Con **F10** la siguiente línea de código se ejecuta.

Con **F11** entro dentro un método para saber que está pasando.

Con **Shift + F11** nos salimos del método.

Con **Shift + F5** paramos el debug mode.