###### Introducción a C#.

*Ejemplo de proyecto.*

using System;

namespace MiPrimerProyecto

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Bienvenido a mi primer proyecto en C#!");

Console.WriteLine("Ingresa tu nombre:");

string nombre = Console.ReadLine();

Console.WriteLine("Hola " + nombre + ", ¿cómo estás hoy?");

}

}

}

Este ejemplo crea una aplicación de consola que le pregunta al usuario por su nombre y luego lo saluda. La clase Program es el punto de entrada de la aplicación y el método Main es el método principal que se ejecutará al iniciar la aplicación. La clase Console se utiliza para escribir y leer datos en la consola.

###### Comentar en C#.

Para comentar en bloque se puede usar el atajo ctrl + shift + ç.

// Este es un comentario en línea

/\* Este es un comentario en bloque,

puede tener varias líneas de texto.

\*/

###### Tipos de variables en C#.

Las variables se declaran en C# al escribirlas y se inicializan cuando se le da un valor.

char ➝ representa un único carácter.

string ➝ una secuencia de carácteres, es decir, una cadena de texto.

int ➝ número entero.

decimal ➝ número decimal de la forma más precisa posible

double ➝ número decimal de forma menos precisa pero más que float.

float ➝ número decimal de la forma menos precisa pero que ocupa menos memoria.

bool ➝ variable booleana, puede ser true o false.

const ➝ significa “constante”, números que no van a cambiar nunca, como por ejemplo el número ℼ.

var ➝ se usa para declarar una variable de la cual ya conocemos el tipo y C# la determina automáticamente. por ejemplo: var myVariable = "Hello World" sería reconocida como un string automáticamente.

###### Concatenar variables en C#.

1. Usando el operador ‘+’.

string nombre = "Juan";

string saludo = "Hola, " + nombre + ".";

1. Usando el método String.Concat:

string nombre = "Juan";

string saludo = string.Concat("Hola, ", nombre, ".");

1. Usando el método String.Format:

string nombre = "Juan";

string saludo = string.Format("Hola, {0}.", nombre);

1. Usando interpolación de cadenas:

string nombre = "Juan";

string saludo = $"Hola, {nombre} y tengo {edad} años.";

###### Operaciones básicas y operadores lógicos.

Las *operaciones básicas* en C# incluyen suma (+), resta (-), multiplicación (\*), división (/), módulo (%), incremento (++) y decremento (--).

Si usamos el incremento antes del nombre de la variable, se incrementa antes de que sea utilizada, y si la usamos después, se mantiene su valor original y luego se incrementa. Por ejemplo:

int num = 5;

// Usando ++num

Console.WriteLine(++num); // 6

Console.WriteLine(num); // 6

// Usando num++

num = 5;

Console.WriteLine(num++); // 5

Console.WriteLine(num); // 6

Para decidir cuántos decimales queremos mostrar podemos usar estas dos maneras:

decimal num = 123.45678m;

Console.WriteLine("Número con un decimal: {0:f1}", num);

decimal num = 1.23456m;

Console.WriteLine(num.ToString("0.00"));

También existen *operadores de asignación compuesta* (+=, -=, \*=, /=, %=) que permiten realizar una operación y asignar el resultado a la misma variable en una sola línea de código.

Los *operadores lógicos* son && (and), || (or) y ! (not).

* && (and): Este operador se usa para evaluar dos expresiones booleanas y devuelve true solo si ambas son verdaderas.
* || (or): Este operador también se usa para evaluar dos expresiones booleanas y devuelve true si al menos una de ellas es verdadera.
* ! (not): Este operador de negación invierte el valor de una expresión booleana. Si la expresión es verdadera, el resultado será false, y viceversa.

*Operadores de comparación:* Son aquellos que permiten comparar dos valores y devuelven un resultado booleano. En C#, los operadores de comparación son:

== (igual), != (distinto), > (mayor), < (menor), >= (mayor o igual) y <= (menor o igual).

Estos operadores se usan para comparar dos valores y determinar si son iguales, diferentes, mayor, menor, mayor o igual, o menor o igual.

###### Sentencias de control de flujo.

###### If - Else - Else if ➝

*if* y *else* son sentencias de control de flujo en programación que nos permiten tomar decisiones basadas en una o más condiciones. *if* ejecuta un bloque de código si una condición especificada es verdadera. *else*, por otro lado, proporciona un camino alternativo para ejecutar un bloque de código si la condición especificada en el *if* es falsa.

Por otro lado, *else if* es una extensión del *if* y *else* que permite evaluar múltiples condiciones en un solo bloque de código. Se utiliza para proporcionar una secuencia de alternativas para ejecutar diferentes bloques de código dependiendo de la verificación de diferentes condiciones sucesivas. Por ejemplo:

if (condición 1)

{

// bloque de código a ejecutar si la condición 1 es verdadera

}

else if (condición 2)

{

// bloque de código a ejecutar si la condición 1 es falsa y la condición 2 es verdadera

}

else

{

// bloque de código a ejecutar si ninguna de las condiciones anteriores es verdadera

}

###### Switch ➝

Esta sentencia permite ejecutar un bloque de código dependiendo del valor de una variable. Por ejemplo:

int day = 3;

switch (day)

{

case 1:

Console.WriteLine("Lunes");

break;

case 2:

Console.WriteLine("Martes");

break;

case 3:

Console.WriteLine("Miércoles");

break;

default:

Console.WriteLine("Otro día");

break;

}

###### Bucle While ➝

Esta sentencia permite ejecutar un bloque de código mientras se cumpla una determinada condición. Por ejemplo:

// Declaramos una variable entera

int counter = 0;

// Utilizamos el ciclo while para repetir un bloque de código mientras la condición sea verdadera

while (counter < 5)

{

// Imprimimos en la consola el valor actual de la variable counter

Console.WriteLine(counter);

// Incrementamos el valor de la variable counter en 1

counter++;

}

###### Bucle Do - While ➝

Esta sentencia es similar al bucle while, pero asegura que se ejecute el bloque de código al menos una vez antes de comprobar la condición. Por ejemplo:

// El bucle do-while se ejecuta al menos una vez

do

{

Console.WriteLine("Introduce un número mayor que 0: ");

// Leemos el número del usuario

string entrada = Console.ReadLine();

numero = int.Parse(entrada);

} while (numero <= 0);

###### Bucle For ➝

Esta sentencia permite ejecutar un bloque de código un número determinado de veces. Por ejemplo:

int suma = 0;

// El bucle for recorre un rango de números

for (int i = 1; i <= 10; i++)

{

suma += i;

}

Console.WriteLine("La suma de los primeros 10 números es: " + suma);

###### Strings.

Un *string* en C# es un tipo de dato que representa una secuencia de caracteres Unicode. Es un objeto en tiempo de ejecución que permite almacenar y manipular texto de manera fácil y eficiente.

Podemos acceder a cada uno de los caracteres de un string a través de su índice, que se encuentra en una posición determinada. El primer carácter tiene el índice 0 y el último tiene un índice igual a la longitud del string menos uno.

string nombre = "Juan";

string apellido = "Pérez";

// Concatenación de strings

string nombreCompleto = nombre + " " + apellido;

Console.WriteLine(nombreCompleto); // Imprime "Juan Pérez"

// Longitud de un string

int longitud = nombre.Length;

Console.WriteLine(longitud); // Imprime "4"

###### Funciones de los Strings.

*string.IsNullOrEmpty* es un método en C# que se utiliza para determinar si una cadena es null o está vacía. Este método devuelve true si la cadena es null o si su longitud es igual a 0, y false en caso contrario. La sintaxis es la siguiente:

string input;

do

{

Console.WriteLine("Por favor, ingresa un valor: ");

input = Console.ReadLine();

} while (string.IsNullOrEmpty(input));

String.Join

toma un separador y una matriz de cadenas y devuelve una nueva cadena que consiste en la concatenación de las cadenas de la matriz separadas por el separador especificado.

string[] cadenas = { "Hola", "Mundo", "!" };

string resultado = string.Join(" ", cadenas);

Console.WriteLine(resultado); // Salida: "Hola Mundo !"

###### Arrays.

###### Funciones de los Arrays.

###### Secuencias de escape.

1. ': Sirve para incluir una comilla simple dentro de un string, por ejemplo: "'" representa una comilla simple.
2. ": Sirve para incluir una comilla doble dentro de un string, por ejemplo: """ representa una comilla doble.
3. \r: Sirve para hacer un retorno de carro, que es el movimiento del cursor a la primera columna de la línea siguiente.
4. \n: Sirve para hacer un salto de línea, que es el movimiento del cursor a la primera columna de la siguiente línea.
5. \t: Sirve para insertar una tabulación, que es un espacio de una distancia predeterminada.
6. \0: Sirve para representar el carácter nulo, que es un carácter especial utilizado para indicar el final de una cadena.

###### Funciones de C#.

Console.ReadKey(); ➝ Lee una letra, se suele usar para finalizar un programa desde consola.

Console.WriteLine(“”); ➝ El programa escribe lo que pongamos aquí en consola.

Console.ReadLine()); ➝ El programa leerá lo que pongamos aquí desde la consola.

Convert.To + Tipo de variable ➝ El programa convertirá esta variable a lo que queramos (si es posible.

(Tipo de variable + TryParse(entrada, out salida))

###### Programación orientada a Objetos.

Es un paradigma de programación que se basa en la creación de objetos. Un objeto es una instancia de una clase, que es una plantilla o molde que define unas propiedades (atributos) y unos comportamientos (métodos) que tendrá el propio objeto.

En C#, una clase se define mediante la palabra clave *“class”* seguida del nombre de la clase y el cuerpo de la clase entre llaves. *Ej:*

public class Género

{

public string Nombre { get; set; }

}

En este caso, hemos creado una clase llamada género, que tiene un atributo llamado “Nombre”. La propiedad es pública, lo que significa que se puede acceder y modificar desde cualquier parte del código. También tiene un *“get”* y un *“set”*, lo que permite obtener y establecer el valor de la propiedad. El método *“get”* se utiliza para obtener el valor de una propiedad, mientras que el método *“set” se usa para establecer el valor.*

Para establecer una buena documentación, se usan comentarios de resumen, este comentario se usa para proporcionar una descripción breve y concisa del propósito de la clase, método o atributo.

*Ejemplo:*  
  
/// <summary>

/// Esta clase representa un género de videojuego.

/// </summary>