UD 2 ASP.NET Core. Características. El lenguaje C#.

2.1 Instrucciones, operadores y tipos de datos.

El objetivo principal de esta unidad será el de familiarizarnos con el lenguaje de programación C# y sus características principales. En esta primera parte de la unidad comenzaremos por los elementos esenciales de cualquier lenguaje de programación, las instrucciones, los operadores y los tipos de datos.

Se recomienda utilizar como referencia el capítulo 2 del siguiente libro:

https://www.syncfusion.com/ebooks/csharp

Además del apartado correspondiente en la guía de programación de C#, parte de la documentación oficial del lenguaje:

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/statements-expressions-operators

En este punto, conviene recordar que el alumno debería haber superado el módulo programación del primer curso y que por lo tanto conoce los elementos de un lenguaje de programación. Por esta razón, el objetivo de esta unidad es presentar brevemente los elementos del lenguaje C#, pero no nos extenderemos en detalles que deben ser conocidos.

Para aquellos sin conocimientos de programación se recomienda la lectura y seguimiento completos de la guía de programación de C# así como del capítulo 2 del libro de referencia.

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/index

Instrucciones, expresiones y operadores.

El código que conforma una aplicación en C# consta de instrucciones basadas en palabras clave, expresiones y operadores. En el siguiente enlace se pueden consultar todas las palabras clave del lenguaje:

https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/index

Las acciones que realiza un programa se expresan en instrucciones. El orden en el que se ejecutan las instrucciones en un programa se denomina flujo de control o flujo de ejecución.

Una instrucción puede constar de una sola línea de código que finaliza en un punto y coma o de una serie de instrucciones de una sola línea en un bloque. Un bloque de instrucciones se incluye entre llaves {} y puede contener bloques anidados.

A continuación se incluyen algunos ejemplos de instrucciones de asignación:

```
int miVariable = 7;
a = b + c;
valor++;
x += 5;
```

Los operadores en C# son similares a los de otros muchos lenguajes, como puede ser Java. En el siguiente enlace podemos ver todos los operadores disponibles en C#. Es muy importante conocer los tipos de operadores de los que disponemos, además de las reglas de precedencia, que indican el orden en el que estos son evaluados. Por esta razón se recomienda detenerse a estudiar con detalle la siguiente documentación:

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/statements-expressions-operators/operators

Tipos de datos.

C# es un lenguaje fuertemente tipado, lo que quiere decir que todas las variables y constantes tienen un tipo, al igual que todas las expresiones que se evalúan como un valor.

Los nombres de las variables deben comenzar por una letra o por el carácter '_', y pueden contener letras y números.

El compilador usa información de tipo para garantizar que todas las operaciones que se realizan en el código cuentan con seguridad de tipos. Por ejemplo, si declara una variable de tipo int, el compilador le permite usar la variable en operaciones de suma y resta. Si intenta realizar esas mismas operaciones en una variable de tipo bool, el compilador genera un error, como se muestra en el siguiente ejemplo:

```
int a = 5;
int b = a + 2; //OK

bool test = true;

// Error. Operator '+' cannot be applied to operands of type 'int' and 'bool'.
int c = a + test;
```

Esta es una característica muy importante del lenguaje, y permite que el compilador sea capaz de detectar posibles errores antes de que se produzcan en tiempo de ejecución, como puede ocurrir en lenguajes débilmente tipados, como por ejemplo php.

C# incluye la palabra reservada ***var*** para la declaración de variables, de modo que podemos declarar una variable sin especificar el tipo de dato, tal como sigue:

```
var x = 5;
```

Es necesario comprender que en este caso no es que estemos declarando una variable sin especificar un tipo de datos, como ocurre por ejemplo en javascript, sino que lo que hacemos es decir al compilador que decida el tipo de dato de la variable a partir del valor que vamos a almacenar en ella. En nuestro caso, al tratarse del valor 5, nuestra variable se definiría como un entero, y una sentencia como la siguiente resultaría en un error de compilación:

```
x = "hola";
```

En el enlace siguiente se pueden consultar los tipos de datos base del lenguaje C#:

https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/built-in-types-table

En determinadas ocasiones necesitaremos operar con datos de diferentes tipos. Cuando se trata de valores numéricos, el lenguaje dispone de una serie de mecanismos de conversión implícita que convertirán los tipos de modo que no se pierda información. Por ejemplo, si sumamos un número entero con uno real, el compilador convierte el entero en real, que es un tipo de mayor precisión, y procede a sumar los dos números como reales.

```
int a = 5;
double b = 3.5;

Console.WriteLine(a + b);
```

El código anterior convertiría el valor de la primera variable a 5,0 y lo sumaría a 3,5, mostrando como resultado el valor 8,5.

En otras ocasiones, por el contrario, necesitaremos forzar los cambios de tipos, esto se puede hacer de varias formas, siendo la más sencilla una conversión explícita o casting. Los casting en C# son exactamente igual que en Java, especificando el tipo de dato al que queremos convertir entre paréntesis.

```
double x = 3.5;

Console.WriteLine((int)x);
```

En este caso, el valor que se mostrará es 3, debido a que convertimos el valor de un número real a un entero, perdiendo de este modo su parte decimal.

Por último disponemos en C# de una serie de operadores específicos que nos permiten la conversión de unos tipos a otros. Ejemplos de estos operadores son la palabra clave ***as***, los métodos ***Parse*** específicos para cada tipo de datos o la clase genérica ***Convert***, que permite una gran cantidad de conversiones. Se incluye el enlace a la documentación de dicha clase.

https://docs.microsoft.com/es-es/dotnet/api/system.convert?view=netcore-2.1

Ámbito de utilización de variables.

En C#, como en muchos otros lenguajes, podemos utilizar variables en cualquier lugar de un programa. Cuando la variable es declarada, se reserva espacio para ella, y en función de en qué lugar se haya declarado, esta podrá ser utilizada o no en determinadas partes del programa. Esto es llamado visibilidad de la variable, y define su ámbito de utilización.

Si la variable ha sido declarada dentro de un bloque entre llaves, la variable sólo estará disponible dentro de ese bloque, si es declarada en un método será visible para todo el método y si lo hacemos en una clase, lo será para toda la clase. Una vez el programa sale del ámbito de la variable, esta es eliminada y dejará de estar disponible.

En ciertas ocasiones necesitamos que una variable se mantenga a lo largo de la ejecución del programa y que su valor persista independientemente de que estemos ejecutando un fragmento de código u otro, o de la creación de diferentes objetos de una clase. Para conseguir esto, de forma similar a Java, podemos declarar una variable estática, utilizando la palabra clave ***static***.

Cadenas de texto.

Las cadenas de texto son uno de los tipos de datos más importantes en cualquier programa, y en C# se declaran mediante el tipo ***string***. Internamente, el texto se almacena como una colección secuencial de solo lectura de objetos ***char***.

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/strings/index

Una tarea fundamental que tendremos que realizar a menudo es la creación de una cadena de texto a partir de varias partes o subcadenas. Para conseguirlo disponemos de diversos mecanismos en C#, siendo el más simple de ellos la concatenación de cadenas, que consiste en formar una cadena añadiendo varias partes una tras otra. Para ello utilizamos el operador ***+***.

```
string name = "Joe";
string helloViaConcatenation = "Hello, " + name + "!";
Console.WriteLine(helloViaConcatenation);
```

El código anterior mostraría el texto **“Hello, Joe!”** en la pantalla. A continuación haremos lo mismo pero utilizando el método ***string.Format***.

```
string helloViaStringFormat = string.Format("Hello, {0}!", name);
Console.WriteLine(helloViaStringFormat);
```

El método ***string.Format*** permite formatear una cadena empleando para ello una serie de “placeholders”, que son lugares en los que insertar y formatear información dentro de la cadena. Estos placeholders se indican mediante un número con el orden del placeholder, empezando en 0, encerrado entre llaves. A continuación, pasaremos tantos parámetros como placeholders haya en la cadena, de modo que el resultado será el de introducir los valores pasados dentro de cada placeholder. En este ejemplo, el primer y único placeholder es sustituido por el valor que pasamos contenido en la variable name, que en nuestro caso es Joe.

```
string item = "bread";
decimal amount = 2.25m;
string result = string.Format("{0,-10}{1:C}", item, amount);
Console.WriteLine(result);
```

En este nuevo ejemplo, se demuestra el uso de otra característica de los placeholders, y es que nos permite especificar el modo en el que el dato será formateado al ser introducido en la cadena. En el primer placeholder tenemos una variable de tipo cadena, ítem, y después del número de orden del placeholder, separado por una coma, indicamos que la cadena ocupará exactamente 10 caracteres. De este modo si la cadena que pasamos es más grande se truncará y si no se completará con espacios en blanco. El signo menos indica que se alineará a la izquierda.

En el segundo placeholder indicamos el tipo de formato a utilizar, separado del número de orden por dos puntos. En el ejemplo se indica el tipo de dato C o currency, que mostrará el valor pasado como una moneda. De este modo la salida del programa sería la siguiente:

```
bread       2,25 €
```

A continuación, se proporcionan enlaces a las diferentes opciones de formateo de cadenas disponibles, tanto para datos numéricos como para formateo de fechas y horas:

https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings

https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-date-and-time-format-strings

Por último, a partir de C# 6 se ha introducido una nueva forma de formatear cadenas, denominada interpolación de cadenas. Permite realizar las mismas tareas que el método ***string.Format***, pero haciendo más sencilla y más corta la sintaxis. El ejemplo anterior, utilizando el operador de interpolación sería como sigue:

```
Console.WriteLine($"{item,-10}{amount:C}");
```

Para indicar que queremos generar una cadena usando interpolación, utilizamos el prefijo $ antes de las comillas de inicio de la cadena. A continuación, dentro de la cadena indicaremos los placeholders, del mismo modo que hacíamos con format.String, pero en lugar de indicar el número de orden, podemos especificar directamente el nombre de la variable o la expresión que queremos insertar en la cadena, además de las opciones de formateo del texto insertado. Este es el modo recomendado de formatear una cadena en C#. En el siguiente enlace se puede encontrar más información acerca del tema:

https://docs.microsoft.com/es-es/dotnet/csharp/tutorials/string-interpolation

Ejercicio 2.1.1.

En este primer ejercicio vamos a trabajar con la declaración de variables, operadores, conversión de tipos explícita y formateo del resultado.

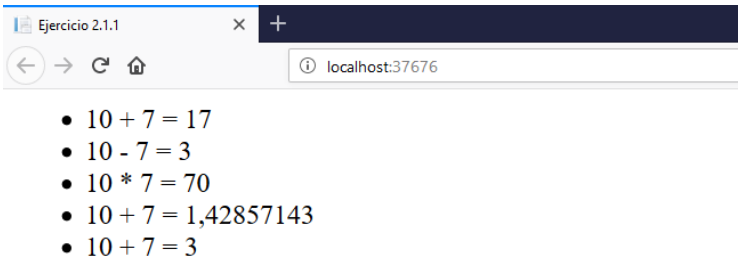
Se pide realizar una aplicación web, partiendo del proyecto base que podemos crear siguiendo las instrucciones proporcionadas en el documento de orientaciones de la unidad 2. Recordamos que el código de nuestro ejercicio se introducirá en el método *Index* del *HomeController*.

```
10  {
11      public class HomeController : Controller
12      {
13          public IActionResult Index()
14          {
15              return View();
16          }
17      }
18  }
19
```

En el controlador deben declararse las siguientes variables, y asignarles los siguientes valores:

a	10
b	7

La ejecución de nuestra aplicación mostrará la siguiente salida:



Los valores del resultado deben ser calculados en el controlador, formateando la salida usando interpolación de cadenas y pasados a la página de la vista, no escritos directamente.

Ejercicio 2.1.2.

Para la realización de este segundo ejercicio utilizaremos una variable estática que cuente el número de veces que se refresque o muestre en el navegador nuestra aplicación. De este modo la salida de la aplicación será un mensaje que indique cuantas veces visualizado la página, de forma similar al siguiente: