



Reporte de Curso #1.

Nombre de Curso: C# Fundamentals For Absolute Beginners

Por: Julián Mejía Sánchez

Para: José de Jesús Narváez Pérez

Fecha de realización: martes 27 de agosto de 2019.

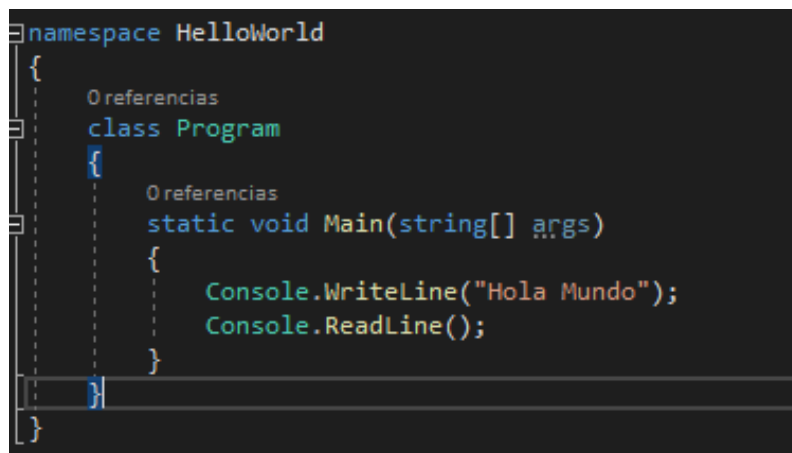
Fecha de entrega: jueves 29 de agosto de 2019.

Introducción.

Este curso consta de varios bloques en los que se exponen diversos temas de interés acerca de C#, los cuales a continuación trataré de explicar de manera breve y precisa.

Desarrollo.

1. El primer tema tiene que ver con hacer mi primera aplicación de consola en C#, el cual es un “Hola Mundo” utilizando la instrucción `Console.WriteLine` a continuación se muestra una imagen para ilustrar la instrucción. (Ver ilustración 1)

A screenshot of a code editor showing C# code for a 'Hello World' application. The code is as follows:

```
namespace HelloWorld
{
    //Referencias
    class Program
    {
        //Referencias
        static void Main(string[] args)
        {
            Console.WriteLine("Hola Mundo");
            Console.ReadLine();
        }
    }
}
```

The code is color-coded: namespaces in blue, classes in green, and methods in blue. Comments are in grey. The code is enclosed in curly braces to indicate scope.

Ilustración 1. Bloque de instrucciones para un Hola Mundo

Dentro de este tema se explican los requerimientos que debe de cumplir dicha instrucción para que funcione de manera correcta, por ejemplo, el texto que se mostrara debe estar entre comillas, la instrucción tiene que estar dentro de la clase main y que debe estar acompañada de la instrucción `Console.ReadLine` la cual detiene la ventana de visualización de resultados para que podamos darnos cuenta de lo que esta pasando. De lo contrario se cerraría la ventana de inmediato impidiéndonos ver el resultado.

2. El segundo tema nos invita a entender nuestra primera aplicación, en el cual se explica porque no deben de estar las instrucciones fuera de la clase main, además que estas siempre deben llevar paréntesis, así como punto y como al final de las declaraciones.

3. El tercer tema nos explica como se declara una sentencia de selección *if* de manera correcta, que hace cada parte del código y como podemos interactuar con la sentencia para obtener algún resultado. A continuación, se muestra una ilustración para entender mejor el código. (ver ilustración 2)

```
class Program
{
    //Referencias
    static void Main(string[] args)
    {
        Console.WriteLine("The Game");
        Console.Write("Selecciona una puerta bro: 1, 2 o 3: ");
        String puerta = Console.ReadLine();
        String mensaje = "";
        if (puerta == "1")
        {
            mensaje = "Te ganaste un coche";
        }

        else if (puerta == "2")
        {
            mensaje = "Te ganaste una moto";
        }
        Console.WriteLine(mensaje);
        Console.ReadLine();
    }
}
```

Ilustración 2. Código para una sentencia If.

Este tema se explica también que la instrucción `Console.ReadLine()` funciona para guardar lo que ingresa el usuario a una variable, en este caso de tipo `String`, después esa variable nos sirve para hacer comparaciones a través de la sentencia `if`, así mismo del operador `"=="` nos ayuda a elegir entre las posibles entradas del usuario, con lo cual cada una nos lleva a un resultado específico.

4. El siguiente tema nos ayuda a entender un poco mejor acerca de los tipos de datos, y variables. Algo que puedo recalcar de lo aquí explicado es que se nos muestra como podemos concatenar en una sola instrucción varias variables así sean diferentes tipos de datos.

5. Este tema nos ayuda a entender cómo se declara y funciona la instrucción *for* la cual se muestra a continuación. (ver ilustración 3),

```
namespace ForIteration
{
    Oreferencias
    class Program
    {
        Oreferencias
        static void Main(string[] args)
        {
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine(i);
                if (i == 7)
                {
                    Console.WriteLine("7 Encontrado");
                    break;
                }
            }
            Console.ReadLine();
        }
    }
}
```

Ilustración 3. Instrucción for

De manera breve, dentro del *for* debe declararse primero la variable la cual se encargará de hacer las iteraciones necesarias, después separado por punto y coma, llevara el límite de operación del *for*, es decir hasta donde queremos que se repita el ciclo y por ultimo el incremento o decremento de la variable de iteración.

Además, se agregó la instrucción *break*, la cual sencillamente nos saca de algún determinado proceso.

6. Entendiendo Arrays, dentro de este tema se explican varias cosas respecto a los arreglos por ejemplo como acceder a ellos a través de los índices (empezando desde 0), como declararlos sin inicializar, y también como declararlos a la vez que se inicializan (a través de llaves). También al igual que otros miembros, los arreglos pueden ser de diferentes tipos de datos a la cual podemos acceder completamente a través de un ciclo *for* por ejemplo.

Podemos hacer conversiones en los tipos de datos, por ejemplo, de una cadena podemos guardarlo en un arreglo de tipo *char*, como se nos mostró en este tema.

7. Definiendo e invocando métodos, esta sección se nos muestra como declarar métodos de manera sencilla, ya sea métodos estáticos o no, y además si son métodos vacíos o que regresan algún tipo de valor en específico. Se aclara que independiente el tipo de método que vayamos a declarar, este debe de ir afuera del método *main*, pero dentro de las llaves del programa en sí. Tal vez una de las principales funciones de un método es reutilizar código, de tal manera que no tengamos que escribir una y otra vez la misma instrucción, al igual que al momento de encontrar un error, no hay que revisar en muchas partes, si no que yendo al método nos podemos dar cuenta de que esta mal.

```
private static string ReverseString(string message)
{
    char[] messageArray = message.ToCharArray();
    Array.Reverse(messageArray);
    return String.Concat(messageArray);
}
```

Ilustración 4. Ejemplo sencillo de un método.

Esta ilustración nos sirve para darnos cuenta de las partes que contiene un método, en este caso es estático y devuelve un valor de tipo *String*, además tiene un parámetro del mismo tipo. Y con este simple ejemplo podemos reutilizar código.

8. Sentencia *While*, esta sentencia de iteración es muy útil sobre todo en acciones que requieren que ocurra un evento para que termine el ciclo. Dicho “evento” podría ser que el usuario ingrese un número o carácter en específico, de esta manera el ciclo *Do-While* ejecuta el código dentro de sí hasta llegar a este lapso.

9. En los siguientes tres temas se nos explica acerca de los datos *DateTime*, los cuales como su nombre lo dice son formatos de fecha y hora, en el video se nos dan varias formas de representar estos datos, por ejemplo, en su forma larga, corta, solo día, solo hora, aumentar días o horas, etc. Además, se explican diversas formas de trabajar con el tipo de dato *String*, por ejemplo, agregar caracteres a la cadena, como comillas, diagonales, como mostrar una cadena de números con ciertos formatos específicos, como cortar parte de una cadena, convertir el texto en mayúsculas, remplazar caracteres por otros, etc. Y por ultimo se explica el como entender mejor las clases dentro de C#, es decir, como declararlas, como instanciar objetos de las mismas para después acceder a ellas, además de agregar métodos dentro de la misma clase. Algo que no había utilizado eran los *set* y *get*.

10. Este tema trato acerca del alcance de las variables y de los modificadores de accesibilidad. De forma breve, el alcance de las variables se puede clasificar en alcance de clase, método y sentencia (*for*, *if*, *etc.*), es decir dependiendo donde este declarada la variable es el espacio en el que podemos utilizarla. Respecto a los modificadores de accesibilidad podríamos definir los públicos, privados, protegido, internos y combinaciones entre algunos. Dependiendo de la accesibilidad que tenga un método, una clase o una variable será la forma que podemos acceder a sus datos, muy eficaz a la hora de encapsular información.

11. En esta sección se nos explica como crear y agregar referencias (*.dll*) a nuestros proyectos, al hacer esto estamos agregando todos los atributos, miembros, métodos y clases de nuestra librería. Es decir, podemos hacer todo el código que necesitemos dentro de una biblioteca de clases, a la cual después podemos agregar simplemente la referencia y pasarle los parámetros que necesitemos. De igual manera se mostró como instalar paquetes *NuGet*, los cuales dependiendo el tipo de trabajo que vayamos a realizar nos serán de utilidad.

12. Esta sección abarco un par de temas, entre ellos el crear desde un arreglo, listas (colecciones), diccionarios. A continuación, se muestran un par de ilustraciones para ejemplificar algunos de estos.

```
// List <T>
List<Car> myList = new List<Car>();
myList.Add(car1);
myList.Add(car2);
```

Ilustración 5. Ejemplo para crear una lista de tipo de dato Car (previamente creado)

```
//Diccionario <TKey, TValue>
Dictionary<String, Car> myDictionary = new Dictionary<string, Car>();
myDictionary.Add(car1.VIN, car1);
myDictionary.Add(car2.VIN, car2);
```

Ilustración 6. Ejemplo de creación para un diccionario.

Como se puede observar en estos ejemplos, es relativamente sencillo entender como trabajan las colecciones de datos. Cabe mencionar que además de estas formas, podríamos optar por inicializar nuestras colecciones al mismo tiempo de declararlas, sin duda una buena práctica para algún fin en específico.

13. Este tema esta relacionado con LINQ que en términos sencillos es un lenguaje integrado de consultas que utiliza C# para distintas fuentes, como objetos y bases de datos, para ello, usa un tipo de funciones propias, que unifica las operaciones más comunes en todos los entornos, para conseguir un mismo lenguaje para todo tipo de tareas. Además, estuvimos trabajando con colecciones de *Listas* para dejar un poco mas claro los temas anteriores.

14. El siguiente tema se extendió en base a el tipo de dato *enum*, y a la sentencia *Switch*, la cual nos ayuda a elegir entre diversas opciones, para este caso se utilizo una lista *enum* para trabajar con de la mano con Switch, en la siguiente imagen se muestra primero una lista enum, y después como se manda llamar en la sentencia, acompañada de una clase.

```
15 referencias
class Todo
{
    13 referencias
    public String Descripcion { get; set; }
    12 referencias
    public int HorasEstimadas { get; set; }
    13 referencias
    public Status Status { get; set; }
}

18 referencias
enum Status
{
    NoComienza,
    EnProgreso,
    EnEspera,
    Completado,
    Eliminado
}
```

Ilustración 7. Clase para acceder a la lista enum.

Después de declarar esto, podemos utilizarlo dentro de nuestra sentencia, como se muestra a continuación. (ver ilustración 8)

```

private static void imprimir(List<Todo> todos)
{
    foreach (var todo in todos)
    {
        switch (todo.Status)
        {
            case Status.NoComienza:
                Console.ForegroundColor = ConsoleColor.DarkGray;
                break;
            case Status.EnProgreso:
                Console.ForegroundColor = ConsoleColor.Blue;
                break;
            case Status.EnEspera:
                Console.ForegroundColor = ConsoleColor.Yellow;
                break;
            case Status.Completado:
                Console.ForegroundColor = ConsoleColor.Green;
                break;
            case Status.Eliminado:
                Console.ForegroundColor = ConsoleColor.Red;
                break;
            default:
                break;
        }
        Console.WriteLine(todo.Descripcion);
    }
}

```

Ilustración 8. Sentencia Switch utilizando la lista enum.

Para llegar a esta parte se utilizaron varios de los temas que se explicaron anteriormente, tales como *listas*, *foreach*, etc. Y en palabras sencillas, para cada caso se utilizó un atributo de la lista enum, a la cual se le asignó un color. Es mucho más complejo que esto, pero está muy ligado con la creación de objetos, métodos, tipos de datos, miembros, etc.

15. Los siguientes dos temas trataron respecto a Excepciones y Eventos, respectivamente hablando, las excepciones se trata de “atrapar” posibles errores dentro de nuestro código, esto se logra a través de la sentencia *Try* y *Catch*, dentro del *try* se encuentra el código que debe ejecutarse cuando no ocurra ninguna incidencia, mientras que el *catch* se hará presente una vez que ocurra algo “inesperado” en nuestro código, aunque realmente se esperan estas incidencias, por lo cual es importante tener noción de cuáles pueden ser posibles errores. Mientras que los eventos, como su nombre lo dicen, se encargan de ejecutar alguna instrucción cuando algo ocurra, por ejemplo, al pasar un tiempo, al hacer clic en un botón, etc. Para ello, en el ejemplo que se nos dio se utilizó un tipo de *Windows Forms*, este tipo de proyectos son más que útiles para tener una mejor noción en el control de eventos.

16. En este par de temas se explicó más a fondo el uso de operadores dentro de C#, por ejemplo, de asignación, de iteración, de adición, división, comparación, decisión, etc. Sin duda muy útiles a la hora de programar, además de esto se explico acerca de como se acomodan nuestros proyectos, soluciones y códigos dentro de los cuales trabajamos. Se despejan dudas de que función tiene cada uno y como se organizan internamente, por ejemplo, que la dentro de la solución puede haber mas de un proyecto a la vez, que esta almacena las configuraciones, versiones etc. Y dentro del proyecto se encuentran nuestras clases y demás archivos necesarios para la ejecución de este.

Conclusiones.

Este curso es de mucha utilidad, ya que, aunque hay cosas que ya tenia conocimiento, hay muchas otras que no y se especifica mas a fondo algunas funciones o expresiones, que sin duda hacen más fácil la comprensión al momento de escribir y leer código. En resumen, se reforzo el conocimiento ya adquirido, y el nuevo conocimiento es de mejor comprensión para futuras referencias.