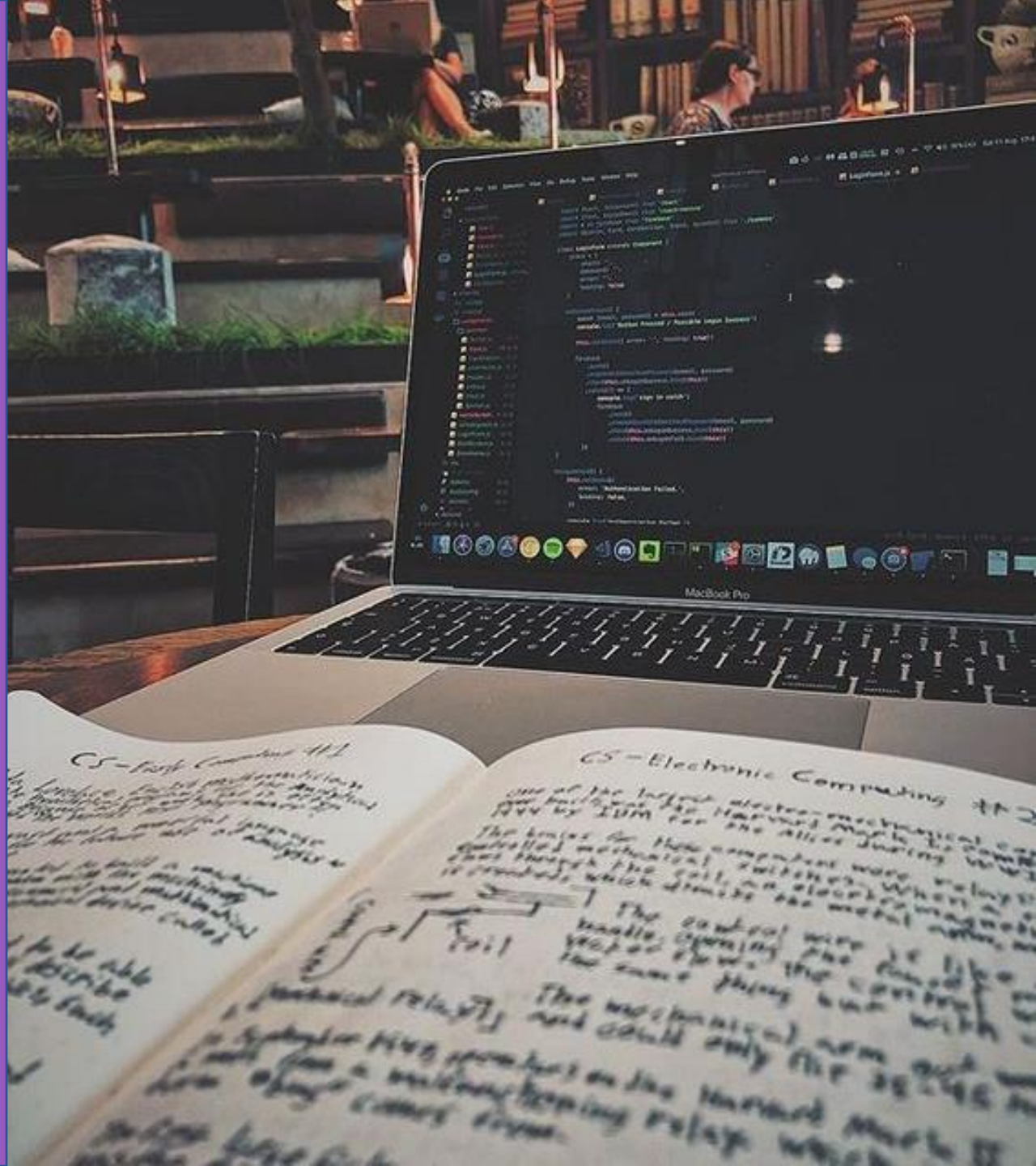


Clase Nro 7

- Trabajo con un grupos de objetos
- Arreglos
- Tipos genéricos
- Colecciones
- Enum



Trabajando con grupos de datos

En muchas aplicaciones, es necesario crear y administrar **grupos** de objetos relacionados.

Existen dos formas de agrupar objetos en C#:

1. Creando Arreglos de objetos
2. Creando colecciones de objetos.

Creando Arreglos

Los Arreglos son útiles para crear y trabajar con un **número fijo** de objetos fuertemente tipados.

Creando colecciones

A diferencia de los arreglos, el grupo de objetos con los que trabaja puede crecer y reducirse dinámicamente en tiempo de ejecución.

ARREGLOS

Un arreglo es un grupo de variables del mismo tipo de datos a las que nos referimos con un nombre común. A cada variable de la matriz llamamos elemento, el Tipo de dato de los elementos pueden ser cualquier tipo válido como ser: char, int, float, etc. Los elementos se almacenan en una ubicación contigua.

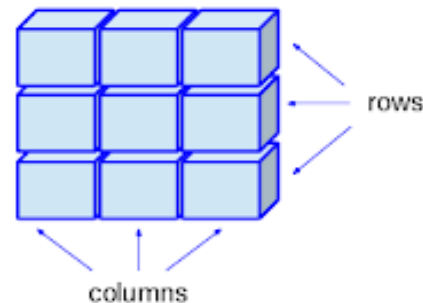
Escalar



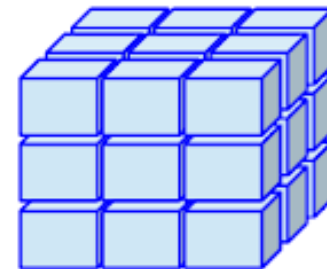
Vector



Matriz



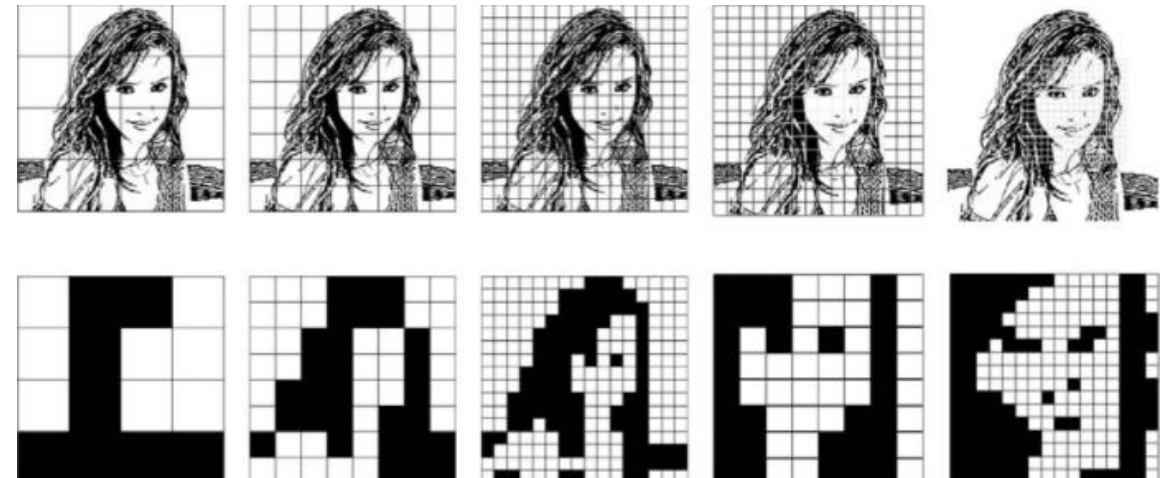
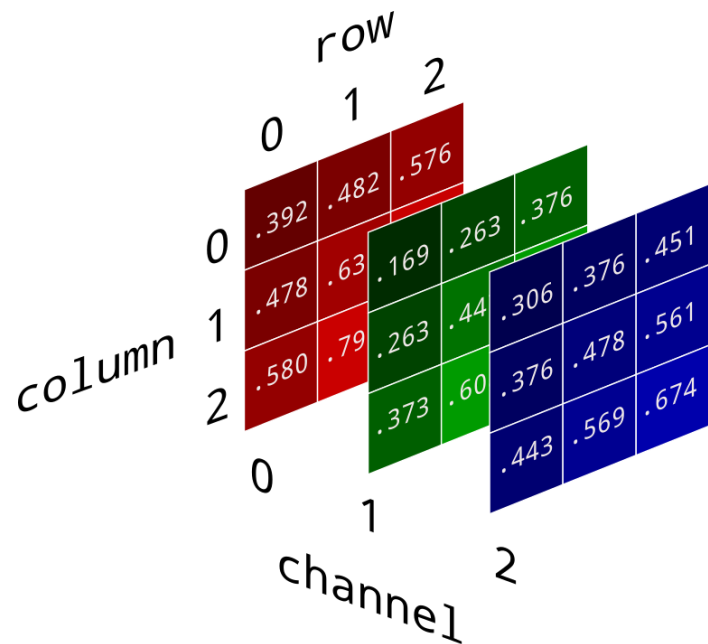
Tensor



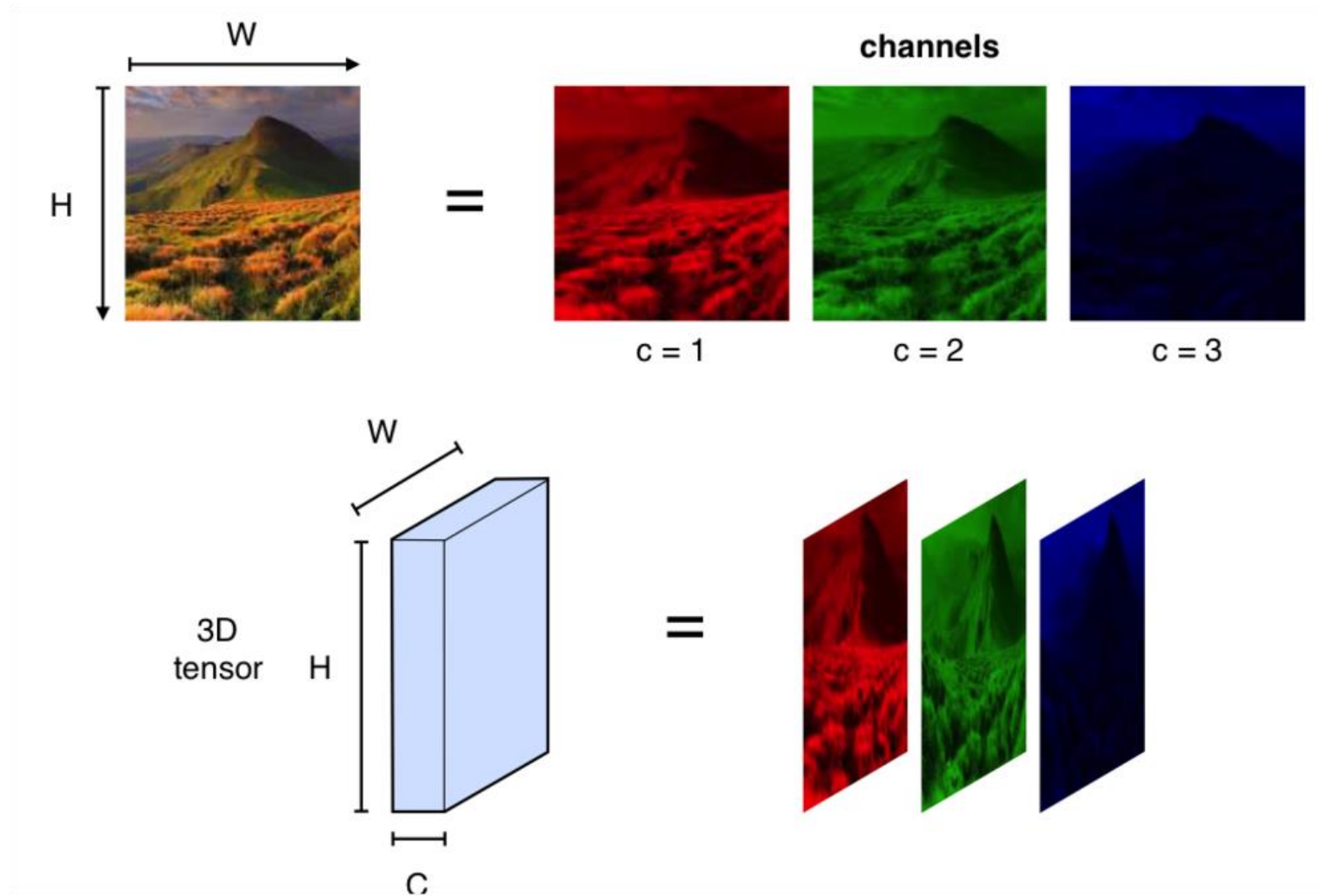
Un escalar es un tensor de orden 0 , un vector es un tensor de orden 1 , una matriz es un tensor de orden 2 , y así hasta el infinito.

ARREGLOS También llamados Array, vector, Matriz o Arreglos

Podemos representar cualquier imagen en color como un tensor de tercer orden, y las tres dimensiones son los datos de altura, anchura y color de la imagen



ARREGLOS También llamados Array, vector, Matriz o Arreglos



ARREGLOS También llamados Array, vector, Matriz o Arreglos

Para los mas curiosos

Css para pintar un bloque HTML de con los componentes RGBA

<https://www.geeksforgeeks.org/css-rgba-function/>

Obtener la matriz de una imagen RGBA con Js.

<https://github.com/bencevans/image-to-rgba-matrix>

ARREGLOS

Sintaxis de un arreglo

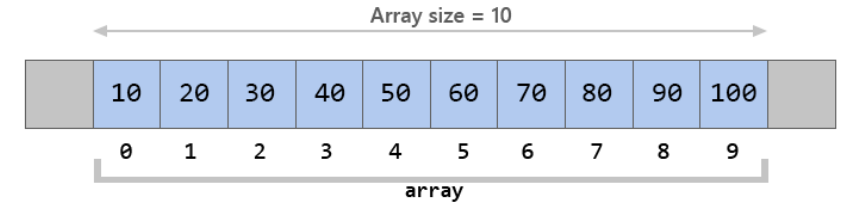
< Tipo de dato > [] Nombre_del_arreglo;

Donde,

< Tipo de dato > : Define el Tipo de cada element de arreglo.

[] : entre los corchetes se define el tamaño y las dimensiones del arreglo.

< Nombre_del_arreglo > : Es el nombre del arreglo.



Ejemplos de declaración de arreglos:

```
int[] x;           // permite guardar elementos de Tipo Entero
string[] s;        // permite guardar elementos de tipo String
double[] d;        // permite guardar elementos de tipo double
Persona[] Estudiantes; // permite guardar elementos de instancia de la clase Persona
```

ARREGLOS

- Un Arreglo es un reference type (tipo por referencia), por lo que se utiliza la palabra clave `new` para crear una instancia de tipo Arreglo.
- Cuando se crea una instancia de un array, el rango y la longitud de cada dimensión se establecen y luego permanecen constantes durante toda la vida útil de la instancia.
- Cada Instancia creada de tipo Array de hereda los miembros declarados por el tipo *System.Array*.

Declaración de un arreglo

```
type [ ] < Name_Array > = new < datatype > [size];
```

Ejemplos de declaración e inicialización de un arreglo:

```
int [,] b = new int[7,8];
```

```
int[] intArray1 = new int[5];
```

```
string[] Autos = {"Volvo", "BMW", "Ford", "Mazda"};
```


ARREGLOS

A tener en cuenta

- Un Arreglo puede ser unidimensional, multidimensional o irregular.
- El número de dimensiones y la longitud de cada dimensión se establecen cuando se crea la instancia de Arreglo. Estos valores no se pueden cambiar durante la vida útil de la instancia.
- Los valores predeterminados de los elementos de un Arreglo numérico se establecen en cero, y los elementos de referencia se establecen en nulo.
- Los arreglos están indexados a cero: un Arreglo con n elementos se indexa de 0 a $n-1$.
- Los elementos de Arreglo pueden ser de cualquier tipo, incluido un tipo de Arreglo.
- Los tipos de Arreglo son tipos de referencia derivados del tipo base abstracto Array.

Genéricos

Es un contenedor para un tipo de dato específico usado al crear una instancia de una variable generica.

Por convención, los parámetros de tipo genérico vienen prefijados con la letra T por convención y deben ser únicos en la declaración para evitar conflictos de nombres en la implementación.

Declaración de un genérico

```
public class MiClase<T>
{
    T value;
}
```

Algunas ventajas del empleo de Generics son:

- Rendimiento.
- Seguridad de tipos.
- Reutilización de código.

Genéricos

Declaración

```
public class MiClase<T>
{
    T value;

    public MiClase(T t)
    {
        value = t;
    }

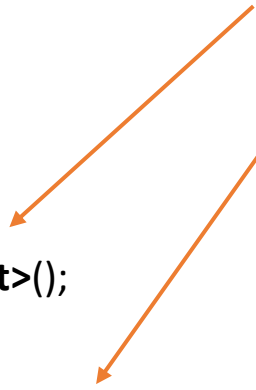
    public void GetValor()
    {
        return value;
    }
}
```

Implementación

```
MiClase<int> Instancia = new MiClase<int>();
```

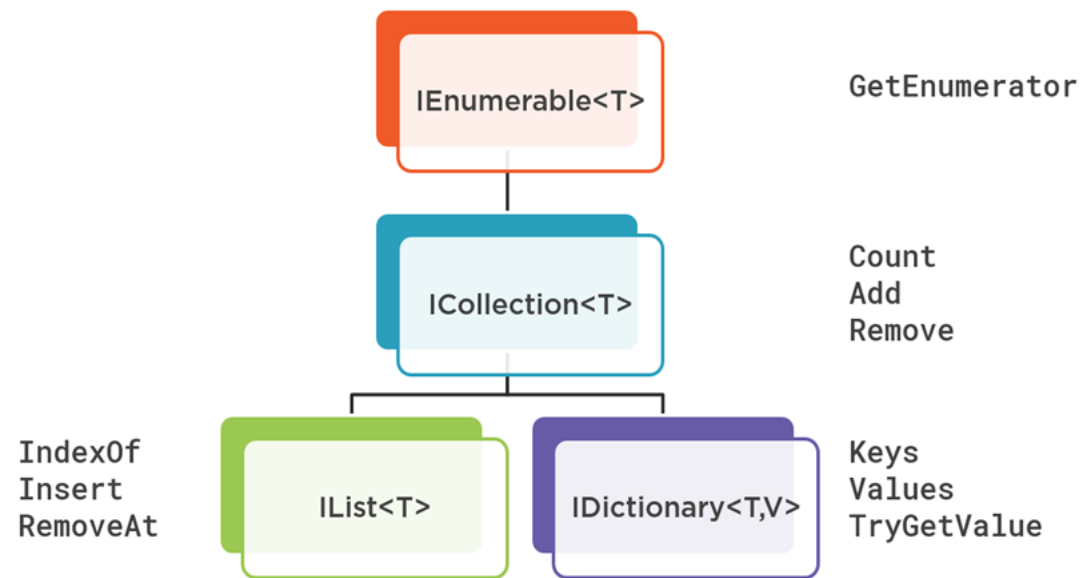
```
MiClase<string> Instancia = new MiClase<string>();
```

Dentro de la implementación, nuestro parámetro de tipo genérico se **sustituye por el tipo específico**.
En el primer caso un **int** y en el segundo caso por un **string**



Colecciones

Las colecciones proporcionan una forma más flexible de trabajar con grupos de objetos. A diferencia de los Arreglos, el grupo de objetos con los que trabaja puede crecer y reducirse dinámicamente a medida que cambian las necesidades de la aplicación



Colecciones

Espacios de nombre de colecciones

Muchas colecciones comunes son proporcionadas por Framework .NET. Cada tipo de colección está diseñada para un propósito específico.

Algunas de las clases de colección comunes pertenecen a los siguientes espacios de nombre:

- [System.Collections](#) (obsoleto)

Las clases en el espacio de nombres System.Collections no almacenan elementos como objetos específicamente escritos, sino como objetos de tipo Object.

- [System.Collections.Concurrent](#)

En .NET Framework 4 o posterior, las colecciones en el espacio de nombres System.Collections.Concurrent proporcionan operaciones eficientes y seguras para el trabajo asíncrono permitiendo acceder a elementos de colección desde múltiples Hilos.

- [System.Collections.Generic](#)

Contiene un conjunto de colecciones genéricas, para ser utilizadas cuando cada elemento de la colección tiene el mismo tipo de datos. Es decir, son colecciones fuertemente tipadas. Lo que provee un mejor rendimiento al ser utilizadas y da una mayor seguridad de tipos.

Colecciones `System.Collections.Generic`

Tipos de colecciones

Algunas de las colecciones más comúnmente utilizadas del espacio de nombre [System.Collections.Generic](#)

Class	Descripción
Dictionary<TKey,TValue>	Representa una colección de Tipo key/value pair que se acceden por su Key
List<T>	Representa una lista de objetos que puede ser accedido por un índice. Provee métodos para búsqueda, ordenar y modificar la lista.
Queue<T>	Representa una colección de objetos organizados de forma first in, first out (FIFO). A esta estructura la conocemos como Fila o Cola.
Stack<T>	Representa una colección de elementos last in, first out (LIFO). A esta estructura la conocemos como Pila.

Colecciones System.Collections.Generic

Uso de colecciones

Una colección es una clase, por lo que debe declarar una instancia de la clase antes de poder agregar elementos a esa colección.

Creando una instancia de tipo Colección

```
List<int> MiLista = new List<int>(); // declaro una lista con objetos de tipo int
```

```
Dictionary<int, string> MiDiccionario = new Dictionary<int,string>(); // declaro una diccionario con objetos de tipo int  
                                como llave y string como valor
```

Agregando elementos a una lista

```
MiLista.add(5); // Agrego el elemento 5 al diccionario
```

```
MiDiccionario.add(1,"Opción 1"); // Agrego el elemento al diccionario indexado con la llave 1 al texto "opción 1"
```

Enum

- Es una estructura de datos de **tipo valor**
- Permite declarar **una lista de constantes** con nombre para cada constantes y referirnos a esta por un nombre que las relaciona y agrupa.

Crear un enum

```
enum DiaDeLaSemana
{
    Domingo = 0,
    Lunes = 1,
    Martes = 2,
    Miercoles = 3,
    Jueves = 4,
    Viernes = 5,
    Sabado = 6
}
```

```
enum ErrorCode : ushort
{
    Ninguno = 0,
    Desconocido = 1,
    PerdidaDeConexion = 100,
    NoSeEncuentraFuenteDeDatos = 200
}
```

Utilizar un enum

```
DiaDeLaSemana MiDía = DiaDeLaSemana.Domingo; // :P
```

```
ErrorCode Erro = ErrorCode.PerdidaDeConexion; // :P
```


Enum - FLAG

- Si se quiere que un tipo enum represente una combinación de elecciones, se utiliza el decorador flag en la declaración y cada miembro del enum se comporta como un campo byte.
- Luego lo debemos utilizar con los operadores lógicos (“or” | o “and” &) para realizar combinaciones e intersecciones

Crear un enum con flag

```
[Flags]
enum DiaDeLaSemana
{
    Ninguno = 0,
    Domingo = 1,
    Lunes = 2,
    Martes = 4,
    Miércoles = 8,
    Jueves = 16,
    Viernes = 32,
    Sabado = 64
}
```

Utilizar un enum con decorador Flag

```
DiaDeLaSemana FindeSemana = DiaDeLaSemana.Domingo | DiaDeLaSemana.Sabado;
```

Sumamente importante!

- Utilizar la documentación oficial del framework/Librería o API con la que estés trabajando.

<https://docs.microsoft.com/es-es/dotnet/api/>

Bibliografía

Genéricos

<https://sparraguerra.wordpress.com/2015/06/21/net-que-son-los-generics-y-su-implementacion-en-c/>

<https://sparraguerra.wordpress.com/2015/07/01/net-que-son-los-generics-y-su-implementacion-en-c-y-ii/>

<https://docs.microsoft.com/es-es/dotnet/standard/generics/>

Arreglos

<https://www.geeksforgeeks.org/c-sharp-arrays/>

Colecciones

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections>

Algunas definiciones y aclaraciones fueron obtenidas de aquí

<https://programmerclick.com/article/8355116650/>

<https://es.quora.com/Qu%C3%A9-es-un-tensor-Es-lo-mismo-que-una-matriz-C%C3%B3mo-se-multiplican-los-n%C3%BAmeros-en-tensores>