



# Ejercicios programación funcional

[Descargar estos ejercicios](#)

## Índice

- [Ejercicio 1](#)
- [Ejercicio 2](#)
- [Ejercicio 3](#)
-  [Ejercicio 4](#)
-  [Ejercicio 5](#)


## Ejercicio 1

Crea una aplicación que a partir de una Lista de enteros, te muestre los múltiplos de un número introducido por teclado que existan en la lista, usando **funciones-λ** y el operador **?:**

## Ejercicio 2

Crea una aplicación que sirva para **buscar coincidencias en una lista de cadenas**. Para ello, asignaremos a **un delegado genérico** una **función-λ** que reciba una lista y una cadena y sobre la lista con el método **FindAll** busque la cadena.

Ten en cuenta que el método **FindAll** necesitará un predicado para el cual utilizaremos otra función-λ para formarlo.

 **Nota:** Puede serte de utilidad la operación **Contains** sobre cadenas.

Por último, muestra, con el método `public void ForEach (Action<T> action);` **definido en** `List<T>`, la lista resultante que devuelva la invocación del delegado.

## Ejercicio 3

Vamos a modificar el ejercicio de la biblioteca del bloque anterior para aplicar funciones lambda, dando tipo a las mismas a través de los delegados genéricos predefinidos en .NET como **Action<...>**, **Predicate<...>**, **Func<...>**, etc.

Los métodos a modificar se indicarán a continuación y en ellos deberás completar lo indicado entre corchetes.

```
public Libro BuscaPorISBN(String isbn)
{
    [Tipo] TieneISBN = [funcion-λ];
    return Libros.Find(TieneISBN);
}
```

```
public int CuentaLibrosConNumeroDePaginasMenorA(int num)
{
    [Tipo] TienepaPaginasMenorANum = [funcion-λ];
    return Libros.FindAll(TienepaPaginasMenorANum).Count;
}
```

```
public void EliminaPorAutor(String autor)
{
    [Tipo] EsDeAutor = [funcion-λ];
    Libros.RemoveAll(EsDeAutor);
}
```

```
public bool EstaPrestado(String isbn)
{
    [Tipo] ContieneCandena = [funcion-λ];
    return (LibrosPrestados().ToList().Find(ContieneCandena) != null);
}
```

```
public IEnumerable<string> LibrosPrestados(){
    // Incluye aquí el código donde se lee el StreamReader con los libros
    // prestados que se guardarón con el ToString de los objetos anónimos
    // préstamo. Además, para generar la secuencia utiliza la producción
    // perezosa yield return...
}
```

## Ejercicio 4

Vamos a realizar una serie de operaciones funcionales usando funciones-λ con el patrón **Map – Filter – Fold** (Select – Where – Aggregate en C#)

Partiremos de la siguiente lista de números reales:

```
List<double> reales = new List<double>
{
    0.5, 1.6, 2.8, 3.9, 4.1, 5.2, 6.3, 7.4, 8.1, 9.2
};
```

Vamos a realizar las siguientes operaciones:

1. Mostrar la lista usando el método `ForEach(Action<T> action)` de lista. Pasando a la función-λ action, una **clausura** de la variable string texto, en la que iremos componiendo su contenido separado por un espacio en blanco.
2. Cuenta aquellos elementos cuya **parte decimal es menor que 0.5**
  - **Map:** Paso del valor real a su parte decimal. Ej:  $2.8 \rightarrow 0.8$
  - **Filter:** Filtro aquellas partes decimales que cumplen el predicado:  $d < 0.5$
  - **Fold:** Contar los elementos en la secuencia resultante.
3. Calcular la suma de todos los valores de la secuencia cuya **parte entera sea múltiplo de 3**.
  - **Map:** Mapea el valor real a un objeto anónimo con la parte entera y el propio valor real de la secuencia. Ej:  $2.8 \rightarrow \text{new } \{ e = 2, r = 2.8 \}$
  - **Filter:** Filtro aquellas partes enteras que cumplen el predicado:  $o.e \% 3 == 0$
  - **Fold:** Suma todos los `o.r` de la secuencia resultante.
4. Calcular el máximo valor de la secuencia cuya parte decimal es mayor que **0.5**
  - **Map:** Mapea el valor real a un objeto anónimo con la parte decimal y el propio valor real de la secuencia. Ej:  $2.8 \rightarrow \text{new } \{ d = 0.8, r = 2.8 \}$
  - **Filter:** Filtro aquellas partes decimales que cumplen el predicado:  $o.d > 0.5$
  - **Fold:** Me quedo con el máximo de todos los `o.r` de la secuencia resultante.
5. **Ampliación:** Muestra aquellos elementos de la secuencia cuya **parte entera es un valor primo**.

💡 **Idea:** Seguramente has de hacer más de un **Filter** e incluso otro **Filter** dentro de uno de ellos. Además, para crear un predicado que me diga si un número primo de forma funcional (pero poco eficiente) puedes hacer lo siguiente ...

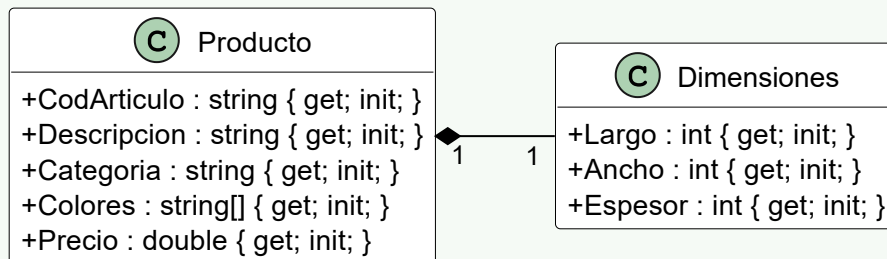
- i. Generar una **secuencia entre 2 y la parte entera menos uno** con `Enumerable.Range()`
- ii. Filtrar aquellos valores divisibles por la parte entera.
- iii. Preguntar si la secuencia resultante tiene 0 elementos.

Una vez completadas todas las operaciones. Al ejecutar el programa la salida por pantalla del programa deberá ser ...

Elementos: 0,5 1,6 2,8 3,9 4,1 5,2 6,3 7,4 8,1 9,2  
Número elementos con parte decimal < 0,5 = 6  
Suma elementos con parte entera múltiplo de 3 = 19,4  
Máximo cuya parte decimal > 0,5 = 3,9  
Elementos parte entera es primo: 2,8 3,9 5,2 7,4

## ✓ Ejercicio 5

En el el **zip de propuestas de solución para autoevaluación** de este bloque de ejercicios, hay un fichero denominado **Ej5\_ConsultasProductos\_a\_completar.cs** donde hay definidas las siguientes clases ...



En la propiedad estática **Productos** de la clase **Datos** te devolverá una secuencia de productos ( **IEnumerable<Producto>** ) sobre la que realizar las consultas.

Además, en el programa principal tienes un 'esqueleto' a completar con descripción de cada consulta. Por ejemplo, para la primera consulta tendríamos ...

```
Console.WriteLine(SeparadorConsulta);
Console.WriteLine(
    "Consulta 1: Usando las funciones Where y Select.\n" +
    "Mostrar CodArticulo, Descripcion y Precio .\n" +
    "de productos con Precio entre 10 y 30 euros\n");
var consulta1 = "";
Console.WriteLine(string.Join("\n", consulta1));
```

Nosotros deberemos rellenar la consulta de acuerdo a las especificaciones de la descripción, cuidando la presentación y sangría para que sean legibles. Por ejemplo ...

```

...
var consulta1 = Productos.Where(p => p.Precio >= 10 && p.Precio <= 30)
    .Select(p => new
    {
        CodArticulo = p.CodArticulo,
        Descripcion = p.Descripcion,
        Precio = p.Precio
    });
...

```

Una vez completadas todas las consultas. Al ejecutar el programa la salida por pantalla del programa deberá ser ...

Consulta 1: Usando las funciones Where y Select.

Mostrar CodArticulo, Descripcion y Precio .  
de productos con Precio entre 10 y 30 euros

```

{ CodArticulo = A01, Descripcion = Uno, Precio = 15,05 }
{ CodArticulo = A02, Descripcion = Dos, Precio = 25,95 }
{ CodArticulo = A04, Descripcion = Cuatro, Precio = 18,45 }

```

Consulta 2: Usando las funciones Select, OrderByDescending y Take.

Muestra CodArticulo, Descripcion y Precio de los 3 productos.  
más caros (ordenando por Precio descendente)

```

{ CodArticulo = A03, Descripcion = Tres, Precio = 30,25 }
{ CodArticulo = A02, Descripcion = Dos, Precio = 25,95 }
{ CodArticulo = A04, Descripcion = Cuatro, Precio = 18,45 }

```

Consulta 3: Usando las funciones GroupBy, OrderBy y First.

Muestra el precio más barato por categoría

```

{ Categoria = C1, PrecioMasBarato = 15,05 }
{ Categoria = C2, PrecioMasBarato = 18,45 }

```

Consulta 4: Usando las funciones GroupBy, Count.

¿Cuántos productos hay de cada categoría?

```
{ Categoria = C1, NumeroProductos = 3 }  
{ Categoria = C2, NumeroProductos = 1 }
```

Consulta 5: Usando las funciones GroupBy, Count, Where y Select

Mostrar las categorías que tengan más de 2 productos

C1

Consulta 6: Usando la función Select

Mostrar CodArticulo, Descripcion, Precio y Descuento redondeado a 2 decimales,  
siendo Descuento el 10% del Precio

```
{ CodArticulo = A01, Descripcion = Uno, Precio = 15,05, Descuento = 1,5 }  
{ CodArticulo = A02, Descripcion = Dos, Precio = 25,95, Descuento = 2,6 }  
{ CodArticulo = A03, Descripcion = Tres, Precio = 30,25, Descuento = 3,03 }  
{ CodArticulo = A04, Descripcion = Cuatro, Precio = 18,45, Descuento = 1,84 }
```

Consulta 7: Usando las funciones Where, Contains y Select.

Mostrar CodArticulo, Descripcion y Colores

de los productos de color verde o rojo

(es decir, que contengan alguno de los dos)

```
{ CodArticulo = A02, Descripcion = Dos, Colores = [blanco, gris, rojo] }  
{ CodArticulo = A03, Descripcion = Tres, Colores = [rojo, gris, verde] }  
{ CodArticulo = A04, Descripcion = Cuatro, Colores = [verde, rojo] }
```

Consulta 8: Usando las funciones Where, Count y Select.

Mostrar CodArticulo, Descripcion y Colores.

de los productos que se fabrican en tres Colores

```
{ CodArticulo = A01, Descripcion = Uno, Colores = [blanco, negro, gris] }  
{ CodArticulo = A02, Descripcion = Dos, Colores = [blanco, gris, rojo] }  
{ CodArticulo = A03, Descripcion = Tres, Colores = [rojo, gris, verde] }
```

Consulta 9: Usando las funciones Where, Select.  
Mostrar CodArticulo, Descripcion y Dimensiones  
de los productos con espesor de 3 cm

```
{ CodArticulo = A01, Descripcion = Uno, Dimensiones = L:4 x A:4 x E:3 }  
{ CodArticulo = A03, Descripcion = Tres, Dimensiones = L:5 x A:5 x E:3 }
```

Consulta 10: Usando las funciones SelectMany, Distinct y OrderBy.  
Mostrar los colores de productos ordenados y sin repeticiones

```
blanco  
gris  
negro  
rojo  
verde
```

Consulta 11: Usando las funciones SelectMany, GroupBy, OrderByDescending.  
Mostrar TotalProductos que hay de cada Color ordenando de mayor a menor cantidad

```
{ Color = gris, TotalProductos = 3 }  
{ Color = rojo, TotalProductos = 3 }  
{ Color = blanco, TotalProductos = 2 }  
{ Color = verde, TotalProductos = 2 }  
{ Color = negro, TotalProductos = 1 }
```