educación IT
CODE YOUR FUTURE

# Introducción a C# .NET

Módulo 4

# Funciones y procedimientos

# Introducción

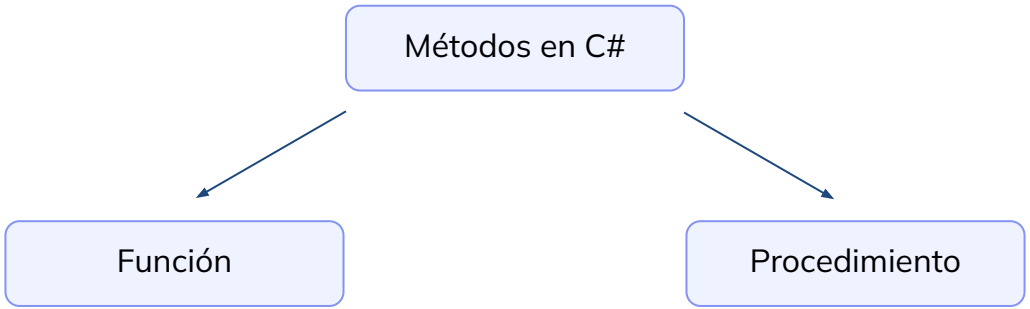
Un problema complejo se puede dividir en pequeños subproblemas más sencillos. Estos subproblemas se conocen como módulos y su implementación en un lenguaje de programación se llama “subprogramas”.

Existe una clasificación de subprograma: **los procedimientos y las funciones**. Veremos usos y diferencias de cada uno.

Un **subprograma** realiza las mismas acciones que un programa, sin embargo, se utilizará el subprograma o módulo para una acción u operación específica.

Un subprograma recibe datos de un programa y éste le devuelve resultados (el programa “llama” o “invoca” al subprograma, éste ejecuta una tarea específica formada por un conjunto de instrucciones y devuelve el “control” al programa que lo llamó).

En C#, y en la programación orientada a objetos (POO), las funciones y procedimientos se conocen con el nombre de “**métodos**” y son subprogramas.


Métodos en C#
Función
Procedimiento

# Funciones (retornan un valor)

En el ámbito de la programación, una **función** es un subprograma que contiene una s**ecuencia de instrucciones** que realizan una tarea específica **dentro de un programa o aplicación más grande.** Las declaraciones de las funciones generalmente son especificadas por:

- **Un nombre único en el ámbito o alcance**: nombre de la función con el que se la identifica y se la distingue de otras. No podrá haber otra función o procedimiento con ese nombre (salvo en sobrecarga o polimorfismo, una de las características de la Programación Orientada a Objetos, o POO).
- **Un tipo de dato de retorno**: tipo de dato del valor que la función devolverá al terminar su ejecución.
- **Una lista de parámetros**: especificación del conjunto de parámetros (pueden ser cero, uno o más) que la función debe recibir para realizar su tarea.

## Sintaxis de una Función

```
ModificadorDeAcceso TipoDeDatoDevuelto Nombre_Funcion(tipo(s) parámetros(s) nombres)
{
      // declaración de datos y cuerpo de la función.
      return (valor);
}
```

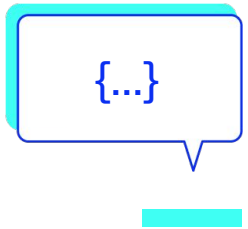
**Ejemplo:**

```
static int Funcion_Sumar(int numero1, int numero2)
        {
            int resultado = numero1 + numero2;
            return resultado;
        }
```

# Procedimientos (no retornan valor)

Un **procedimiento es un subprograma** que realiza una tarea específica y es relativamente **independiente del resto del código de la aplicación.**

Los procedimientos suelen utilizarse para **reducir la duplicación de código en un programa.** Los procedimientos pueden recibir parámetros, pero **no necesitan devolver un valor** como es el caso de las funciones.



## Sintaxis Procedimiento

```
ModificadorDeAcceso void Nombre_Procedimiento(tipo(s) parámetros(s) nombres)
{
      // declaración de datos y cuerpo del procedimiento.
}
```

**Ejemplo:**

```
static void Procedimiento_Sumar(int numero1, int numero2)
        {
            int resultado = numero1 + numero2;
            Console.WriteLine(resultado);
        }
```

# Parámetros

**Un parámetro es un dato que ingresa a una función o procedimiento a través de su llamada.** Todos los tipos de dato de C#, son *tipos de valor* o *tipos de referencia.* De forma predeterminada, los tipos de valor y los tipos de referencia se pasan a un procedimiento o función por valor.

Asociando con lo visto en clase, un **arreglo** es un tipo de referencia dado que es necesario instanciarlo con la palabra reservada **new**.
Es decir, **es un objeto.**

Para más detalles puedes consultar: Tipos y variables y Tipos integrados

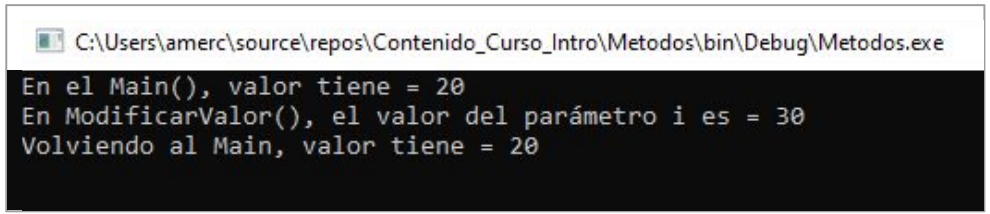
## Pasar parámetros por valor

Cuando un tipo de valor se pasa a un método por valor, se pasa una copia del objeto y no el propio objeto. Por lo tanto, los cambios realizados en el objeto en el método llamado no tienen ningún efecto en el objeto original cuando el control vuelve al autor de la llamada.

En el ejemplo siguiente se pasa un tipo de valor a un método por valor, y el método llamado intenta cambiar el valor del tipo de valor. Define una variable de tipo **`int`**, que es un tipo de valor, inicializa su valor en 20 y lo pasa a un método denominado **`ModificarValor`** que cambia el valor de la variable a 30. Pero cuando el método vuelve, el valor de la variable no cambia.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Metodos
{
       class MetodoFuncion
        {
              static void Main(string[] args)
              {
              int valor = 20;

              Console.WriteLine("En el Main(), valor tiene = {0}", valor);

              ModificarValor(valor);

              Console.WriteLine("Volviendo al Main, valor tiene = {0}", valor);
              Console.ReadKey();
              }
```

```
            static void ModificarValor(int i)
            {
            i = 30;
                  Console.WriteLine($"En ModificarValor(), el valor del parámetro i es = {i}");
                  return;
            }
      }
}
```



Cuando un objeto de un tipo de referencia se pasa a un método por valor, se pasa por valor una referencia al objeto. Es decir, el método no recibe el objeto concreto, sino un argumento que indica la ubicación en memoria del objeto (una dirección de memoria).

Si cambia un miembro del objeto mediante esta referencia, el cambio se reflejará en el objeto cuando el control vuelva al método de llamada. Pero el reemplazo del objeto pasado al método no tendrá ningún efecto en el objeto original cuando el control vuelva al autor de la llamada.

En el ejemplo del siguiente slide se define una clase (que es un tipo de referencia) denominada **`EjemploRefType`**.

Crea una instancia de un objeto **`EjemploRefType`**, asigna 44 a su campo value y pasa el objeto al método **`ModificarObjeto`**. Fundamentalmente, este ejemplo hace lo mismo que el ejemplo anterior: pasa un argumento por valor a un método. Pero, debido a que se usa un tipo de referencia, el resultado es diferente.

La modificación que se lleva a cabo en **`ModificarObjeto`** para el campo **`obj.value`** cambia también el campo value del argumento **`rt`**, en el método **`Main`** a 33, tal y como muestra el resultado del ejemplo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Metodos
{
	class MetodoFuncion
	{
		static void Main(string[] args)
		{
			var rt = new EjemploRefType();
			rt.value = 44;
			Console.WriteLine("En el Main(), el valor del objeto rt es = {0}", rt.value);

			ModificarObjeto(rt);
			Console.WriteLine("Y al volver del procedimiento ModificarObjeto es = {0}", rt.value);

			Console.ReadKey();
		}
```

...

```
            static void ModificarObjeto(EjemploRefType obj)
            {
                  obj.value = 33;
            }
      }

      public class EjemploRefType
      {
            public int value;
      }
}
```

```
En el Main(), el valor del objeto rt es = 44
Y luego de volver del procedimiento ModificarObjeto es = 33
```

## Pasar parámetros por referencia

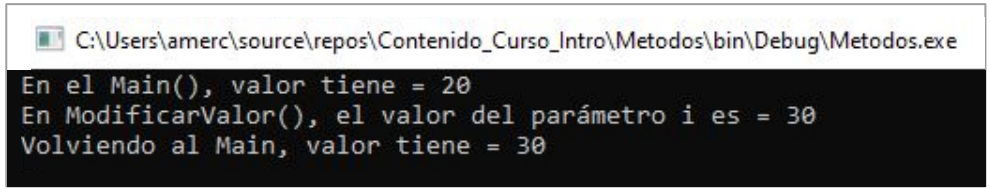
Se pasa un parámetro por referencia cuando se quiere cambiar el valor de un argumento en un método, y reflejar ese cambio cuando el control vuelva al método de llamada. Para pasar un parámetro por referencia, se usa la palabra clave `ref` o `out`. También se puede pasar un valor por referencia para evitar la copia e impedir modificaciones usando la palabra clave `in`.

El ejemplo siguiente es idéntico al anterior, salvo que el valor se pasa por referencia al método **`ModificarValor`**. Al modificar el valor del parámetro en el método **`ModificarValor`**, el cambio del valor se refleja cuando el control vuelve al autor de la llamada.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Metodos
{
	class MetodoFuncion
	{
		static void Main(string[] args)
		{
			int valor = 20;

			Console.WriteLine("En el Main(), valor tiene = {0}", valor);
			ModificarValor(ref valor);
			Console.WriteLine("Volviendo al Main, valor tiene = {0}", valor);

			Console.ReadKey();
		}
```

...

```
            Static void ModificarValor(ref int i)
            {
            i = 30;
                  Console.WriteLine("En ModificarValor(), el valor del parámetro i es = {0}", i);
                  return;
            }
      }
}
```



## Importante: POO

Se recomienda que antes de tomar el siguiente curso de **C#**, realices el curso de **Paradigma de Objetos**.

Fuente consultada:
Microsoft Docs

educación IT
CODE YOUR FUTURE

# ¡Sigamos trabajando!