



APRENDE

**C#**

GUÍA PRÁCTICA

A C A D E M I A X

## Contenido

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introducción</b>                                       | <b>4</b>  |
| 1.1      | Bienvenida . . . . .                                      | 4         |
| 1.1.1    | Libro vivo . . . . .                                      | 5         |
| 1.1.2    | Alcance . . . . .   | 5         |
| 1.2      | Prerequisitos . . . . .                                   | 5         |
| 1.3      | ¿Cómo evitar bloqueos? . . . . .                          | 6         |
| <b>2</b> | <b>Primeros pasos</b>                                     | <b>7</b>  |
| 2.1      | Visión general . . . . .                                  | 7         |
| 2.1.1    | ¿Qué es y por qué debes aprenderlo? . . . . .             | 7         |
| 2.1.2    | ¿En dónde se utiliza? . . . . .                           | 7         |
| 2.1.3    | ¿Qué trabajos puedes conseguir? . . . . .                 | 8         |
| 2.1.4    | ¿Cuánto puedes ganar? . . . . .                           | 8         |
| 2.1.5    | ¿Cuales son las preguntas más comunes? . . . . .          | 8         |
| 2.2      | Historia, evolución, y versiones . . . . .                | 9         |
| 2.3      | Características y ventajas . . . . .                      | 10        |
| 2.4      | Diferencias con otros lenguajes de programación . . . . . | 11        |
| 2.5      | Configuración . . . . .                                   | 12        |
| 2.5.1    | IDE . . . . .   | 12        |
| 2.5.2    | Entorno . . . . .   | 12        |
| 2.6      | Hola Mundo . . . . .                                      | 13        |
| <b>3</b> | <b>Gramática</b>  | <b>14</b> |
| 3.1      | Sintaxis . . . . .  | 14        |
| 3.2      | Comentarios . . . . .                                     | 14        |
| <b>4</b> | <b>Tipos y variables</b>                                  | <b>15</b> |
| 4.1      | Variables . . . . .                                       | 15        |
| 4.2      | Listas . . . . .  | 17        |

## Contenido

---

|           |   |           |
|-----------|---|-----------|
| 4.3       | Objetos . . . . .                             | 19        |
| 4.4       | Constantes . . . . .                          | 21        |
| <b>5</b>  | <b>Operadores</b>                             | <b>22</b> |
| 5.1       | Operadores de aritméticos . . . . .           | 22        |
| 5.2       | Operadores comparativos . . . . .             | 23        |
| 5.3       | Operadores lógicos . . . . .                  | 23        |
| <b>6</b>  | <b>Condicionales</b>                          | <b>24</b> |
| 6.1       | Condición única . . . . .                     | 24        |
| 6.2       | Múltiples condiciones . . . . .               | 25        |
| <b>7</b>  | <b>Bucles</b>                                 | <b>27</b> |
| 7.1       | Bucle for . . . . .                           | 27        |
| 7.2       | Bucle while . . . . .                         | 28        |
| <b>8</b>  | <b>Funciones</b>                              | <b>29</b> |
| <b>9</b>  | <b>Programación orientada a objetos (POO)</b> | <b>33</b> |
| 9.1       | Paradigma . . . . .                           | 33        |
| 9.2       | Clases . . . . .                              | 33        |
| <b>10</b> | <b>Módulos</b>                                | <b>34</b> |
| <b>11</b> | <b>Siguientes pasos</b>                       | <b>36</b> |
| 11.1      | Herramientas . . . . .                        | 36        |
| 11.2      | Recursos . . . . .                            | 37        |
| 11.3      | ¿Que viene después? . . . . .                 | 37        |
| 11.4      | Preguntas de entrevista . . . . .             | 38        |

# 1 Introducción

## 1.1 Bienvenida

Bienvenid@ a esta guía de Academia X en donde aprenderás C# práctico.

Hola, mi nombre es Xavier Reyes Ochoa y soy el autor de esta guía. He trabajado como consultor en proyectos para Nintendo, Google, entre otros proyectos top-tier, trabajé como líder de un equipo de desarrollo por 3 años, y soy Ingeniero Ex-Amazon. En mis redes me conocen como Programador X y comparto videos semanalmente en YouTube desde diversas locaciones del mundo con el objetivo de guiar y motivar a mis estudiantes mientras trabajo haciendo lo que más me gusta: la programación.

En esta guía vas a aprender estos temas:

- Gramática
- Tipos y variables
- Operadores
- Condicionales
- Bucles
- Funciones
- Programación orientada a objetos (POO)
- Módulos

La motivación de esta guía es darte todo el conocimiento técnico que necesitas para elevar la calidad de tus proyectos y al mismo tiempo puedas perseguir metas más grandes, ya sea utilizar esta tecnología para tus pasatiempos creativos, aumentar tus oportunidades laborales, o si tienes el espíritu emprendedor, incluso crear tu propio negocio en línea. Confío en que esta guía te dará los recursos que necesitas para que tengas éxito en este campo.

Empecemos!

## 1 INTRODUCCIÓN

---

### 1.1.1 Libro vivo

Esta publicación fue planeada, editada, y revisada manualmente por Xavier Reyes Ochoa. La fundación del contenido fue generada parcialmente por inteligencia artificial usando ChatGPT (Mar 14 Version) de OpenAI. Puedes ver más detalles en <https://openai.com/>

Esto es a lo que llamo un trabajo **VIVO**, esto quiere decir que será actualizado en el tiempo a medida que existan cambios en la tecnología. La versión actual es 1.0.0 editada el 23 de marzo de 2023. Si tienes correcciones importantes, puedes escribirnos a nuestra sección de contacto en <https://www.academia-x.com>

### 1.1.2 Alcance

El objetivo de esta guía es llenar el vacío que existe sobre esta tecnología en Español siguiendo el siguiente enfoque:

- Se revizan los temas con un enfoque práctico incluyendo ejemplos y retos.
- Se evita incluir material de relleno ya que no es eficiente.
- Se evita entrar en detalle en temas simples o avanzados no-prácticos.

Si deseas profundizar en algún tema, te dejo varios recursos populares y avanzados en la lección de recursos como el estándar de esta tecnología (que puede ser difícil de leer si recién estás empezando).

## 1.2 Prerequisitos

Antes de aprender C#, necesitas lo siguiente:

1. Un computador: cualquier computador moderno tiene las capacidades de correr este lenguaje. Te recomiendo un monitor de escritorio o laptop ya que dispositivos móviles o ipads no son cómodos para programar.

## 1 INTRODUCCIÓN

---

2. Sistema operativo: conocimiento básico de cómo utilizar el sistema operativo en el que se ejecutará C# (por ejemplo, Windows, MacOS, Linux). Te recomiendo tener el sistema operativo actualizado.
3. Conocimiento básico de la línea de comandos: se utiliza para ejecutar programas en C#.
4. Un editor de texto: lo necesitas para escribir código de C#. Los editores de texto más populares son Visual Studio Code, Sublime Text, Atom y Notepad ++.
5. Un navegador web y conexión al internet: es útil para investigar más sobre esta tecnología cuando tengas dudas. Los navegadores más populares son Google Chrome, Mozilla Firefox, Safari y Microsoft Edge. Se recomienda tener el navegador actualizado.

Si ya tienes estos requisitos, estarás en una buena posición para comenzar a aprender C# y profundizar en sus características y aplicaciones.

Si todavía no tienes conocimiento sobre algunos de estos temas, te recomiendo buscar tutoriales básicos en blogs a través de Google, ver videos en YouTube, o hacer preguntas a ChatGPT. Alternativamente, puedes empezar ya e investigar en línea a medida que encuentres bloqueos enteniendo los conceptos en esta guía.

### 1.3 ¿Cómo evitar bloqueos?

Para sacarle el mayor provecho a esta guía:

1. No solo leas esta guía. La práctica es esencial en este campo. Practica todos los días y no pases de lección hasta que un concepto esté 100% claro.
2. No tienes que memorizarlo todo, solo tienes que saber donde están los temas para buscarlos rápidamente cuando tengas dudas.
3. Cuando tengas preguntas usa [Google](#), [StackOverflow](#), y [ChatGPT](#)
4. Acepta que en esta carrera, mucho de tu tiempo lo vas utilizar investigando e innovando, no solo escribiendo código.

5. No tienes que aprender inglés ahora pero considera aprenderlo en un futuro porque los recursos más actualizados están en inglés y también te dará mejores oportunidades laborales.
6. Si pierdas la motivación, recuerda tus objetivos. Ninguna carrera es fácil pero ya tienes los recursos para llegar muy lejos. Te deseo lo mejor en este campo!

## 2 Primeros pasos

### 2.1 Visión general

#### 2.1.1 ¿Qué es y por qué debes aprenderlo?

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft. Está diseñado para permitir a los desarrolladores crear aplicaciones para la plataforma .NET. Es un lenguaje de programación moderno, con sintaxis limpia y una gran cantidad de características útiles.

Debes aprender C# porque es un lenguaje de programación versátil y poderoso. Puede ser utilizado para desarrollar aplicaciones de escritorio, aplicaciones web, aplicaciones móviles, juegos, servicios web y mucho más. Además, es un lenguaje de programación fácil de aprender, con una gran cantidad de recursos disponibles para ayudar a los principiantes.

#### 2.1.2 ¿En dónde se utiliza?

C# es un lenguaje de programación multiparadigma desarrollado por Microsoft que se utiliza principalmente para el desarrollo de aplicaciones de Windows, aplicaciones web, servicios web, aplicaciones móviles, videojuegos y aplicaciones de realidad virtual. También se utiliza para el desarrollo de aplicaciones de escritorio, aplicaciones de servidor y aplicaciones de base de datos.

### 2.1.3 ¿Qué trabajos puedes conseguir?

- Desarrollador de aplicaciones de escritorio
- Desarrollador de aplicaciones web
- Desarrollador de aplicaciones móviles
- Desarrollador de juegos
- Desarrollador de software
- Desarrollador de aplicaciones de realidad virtual
- Desarrollador de aplicaciones de inteligencia artificial
- Desarrollador de aplicaciones de Internet de las cosas (IoT)
- Desarrollador de aplicaciones de blockchain
- Desarrollador de aplicaciones de análisis de datos

### 2.1.4 ¿Cuánto puedes ganar?

El salario que puedes ganar usando C# depende de muchos factores, como tu experiencia, el lugar donde trabajes, el tipo de trabajo que estés haciendo, etc. Según Glassdoor, los salarios promedio para desarrolladores de C# en los Estados Unidos son de \$90,000 anuales.

Es importante tener en cuenta que estos son solo promedios y que el salario real que puedes ganar puede ser mayor o menor, dependiendo de los factores mencionados anteriormente. Además, siempre es una buena idea investigar y hacer preguntas sobre los salarios y las condiciones laborales antes de aceptar un trabajo.

### 2.1.5 ¿Cuales son las preguntas más comunes?

1. ¿Cómo se crea una aplicación en C#?
2. ¿Cómo se usa el lenguaje C#?
3. ¿Cómo se pueden crear clases en C#?
4. ¿Cómo se pueden crear métodos en C#?



5. ¿Cómo se pueden crear interfaces en C#?
6. ¿Cómo se pueden crear eventos en C#?
7. ¿Cómo se pueden crear excepciones en C#?
8. ¿Cómo se pueden crear hilos en C#?
9. ¿Cómo se pueden usar expresiones lambda en C#?
10. ¿Cómo se pueden usar LINQ en C#?

Al finalizar este recurso, tendrás las habilidades necesarias para responder o encontrar las respuestas a estas preguntas.

### 2.2 Historia, evolución, y versiones

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft como parte de su .NET Framework. Fue creado por Anders Hejlsberg en 2000 como una evolución de C y C++.

C# es un lenguaje de programación multiparadigma que admite programación orientada a objetos, programación imperativa y programación funcional. Está diseñado para ser simple, moderno y generalizado.

C# se ha desarrollado a lo largo de los años para mejorar su rendimiento y agregar nuevas características. La última versión de C# es C# 8.0, que fue lanzada en 2019. Esta versión incluye características como la programación asíncrona, la programación reactiva y la sintaxis de patrones de coincidencia.

C# se ha utilizado ampliamente para desarrollar aplicaciones de escritorio, aplicaciones web, aplicaciones móviles, juegos y aplicaciones de realidad virtual. Está disponible para Windows, macOS, Linux y otros sistemas operativos.

### 2.3 Características y ventajas

C# es un lenguaje de programación multiparadigma, orientado a objetos, diseñado para la plataforma .NET. Está diseñado para ser fácil de usar, flexible y poderoso.

#### Características:

- Soporta programación orientada a objetos: C# permite a los desarrolladores crear programas orientados a objetos, lo que significa que los programas se pueden construir a partir de objetos y clases. Esto permite a los desarrolladores crear programas más flexibles y modulares.
- Tipado fuerte: C# es un lenguaje de tipado fuerte, lo que significa que los tipos de datos deben ser especificados explícitamente. Esto ayuda a los desarrolladores a evitar errores de programación y asegurar que los programas sean más estables.
- Compilación rápida: C# se compila a código de máquina, lo que significa que los programas se ejecutan más rápido que los lenguajes interpretados. Esto hace que C# sea un lenguaje ideal para aplicaciones de alto rendimiento.
- Soporte para la plataforma .NET: C# es un lenguaje diseñado para la plataforma .NET, lo que significa que los programas escritos en C# se pueden ejecutar en cualquier sistema que tenga .NET instalado. Esto hace que C# sea un lenguaje ideal para la creación de aplicaciones multiplataforma.

#### Ventajas:

- Fácil de aprender: C# es un lenguaje de programación fácil de aprender, especialmente para aquellos que ya tienen experiencia con otros lenguajes de programación orientados a objetos.
- Flexible: C# es un lenguaje de programación flexible, lo que significa que los desarrolladores pueden crear programas que se ajusten a sus necesidades.
- Potente: C# es un lenguaje de programación potente, lo que significa que los desarrolladores pueden crear programas con un alto nivel de complejidad.

- Seguro: C# es un lenguaje de programación seguro, lo que significa que los programas escritos en C# son menos propensos a errores de seguridad. Esto hace que C# sea un lenguaje ideal para la creación de aplicaciones seguras.

### 2.4 Diferencias con otros lenguajes de programación

C# es un lenguaje de programación orientado a objetos, que se basa en el lenguaje de programación C++. Está diseñado para ser fácil de usar y para permitir a los desarrolladores crear aplicaciones robustas y escalables.

Una de las principales diferencias entre C# y otros lenguajes de programación es que C# es un lenguaje compilado, lo que significa que el código fuente se compila en un código binario ejecutable antes de que se ejecute. Esto significa que el código se ejecuta más rápido y es más seguro que los lenguajes interpretados, como JavaScript o Python.

Otra diferencia importante entre C# y otros lenguajes de programación es que C# es un lenguaje de programación orientado a objetos. Esto significa que los programadores pueden crear objetos que contienen datos y lógica, lo que facilita la creación de aplicaciones robustas y escalables.

Además, C# es un lenguaje de programación multiparadigma, lo que significa que los programadores pueden elegir entre diferentes estilos de programación, como programación orientada a objetos, programación funcional y programación imperativa. Esto permite a los desarrolladores elegir el estilo de programación que mejor se adapte a sus necesidades.

### 2.5 Configuración

#### 2.5.1 IDE

Los archivos de C# son archivos de texto. Puedes editarlos con **editores de texto** como Notepad en Windows o Notes en MacOS pero es recomendado utilizar un **IDE** (Integrated Development Environment) que es una aplicación de edición de código más avanzado que le da colores a tu código para que sea más fácil de leer y tengas funciones de autocompletado, entre otras. Algunos IDEs populares son [Brackets](#), [Atom](#), [Sublime Text](#), [Vim](#), y [Visual Studio Code](#).

El editor recomendado para practicar el código que vamos a ver es Visual Studio Code (o VSCode) que puedes bajar desde <https://code.visualstudio.com/>

#### 2.5.2 Entorno

- **Instalar .NET Core:** C# requiere instalar el framework de .NET Core. Para instalar .NET Core, descarga e instala el SDK de .NET Core desde [este enlace](#).
- **Verificar la instalación:** Para verificar que .NET Core se ha instalado correctamente, abre una ventana de línea de comandos y ejecuta el comando:

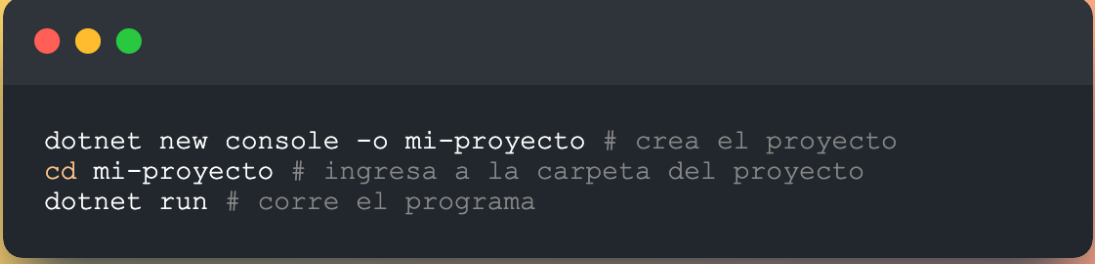


Esto debería devolver la versión de .NET Core que se ha instalado.

- **Correr archivos de C#:** Para correr C# se requiere crear un proyecto de .NET Core. Crea un proyecto accediendo a tu carpeta de preferencia y corriendo este código:

## 2 PRIMEROS PASOS

---



```
dotnet new console -o mi-proyecto # crea el proyecto
cd mi-proyecto # ingresa a la carpeta del proyecto
dotnet run # corre el programa
```

Este código creará una carpeta del proyecto con varios archivos:


- El archivo con extensión **mi-proyecto.csproj** es el archivo de configuración del proyecto.
- El archivo **Program.cs** es donde puedes incluir el código que vamos a ver en las siguientes lecciones.
- La carpeta **bin** se genera con el ejecutable de tu programa al correr el programa.

### 2.6 Hola Mundo

“Hola Mundo” es un ejemplo clásico que se utiliza para mostrar el funcionamiento básico de un lenguaje de programación.

Ejemplo:

En este ejemplo, se imprime el texto “Hola Mundo” en la consola.



```
Console.WriteLine("Hola Mundo");
```

## 3 GRAMÁTICA

---

También podrías modificar este código con tu nombre. Si es “Juan”, debería imprimir en la consola “Hola, Juan” .

Reto:

Modifica el ejemplo anterior para imprimir “Hola Universo” en la consola.

## 3 Gramática

### 3.1 Sintaxis

C# es un lenguaje de programación orientado a objetos de alto nivel. La sintaxis de C# se basa en la sintaxis de C++, con algunas mejoras y extensiones.

Las palabras clave en C# se usan para identificar los elementos del lenguaje, como clases, métodos, variables y otros. La indentación es importante para la legibilidad del código, y se recomienda usar cuatro espacios para cada nivel de indentación. El conjunto de caracteres de C# es Unicode, lo que significa que puede usar caracteres especiales en su código.

C# es sensible a mayúsculas y minúsculas, por lo que debe asegurarse de usar la misma combinación de mayúsculas y minúsculas para los nombres de clases, métodos, variables y otros elementos del lenguaje.

### 3.2 Comentarios

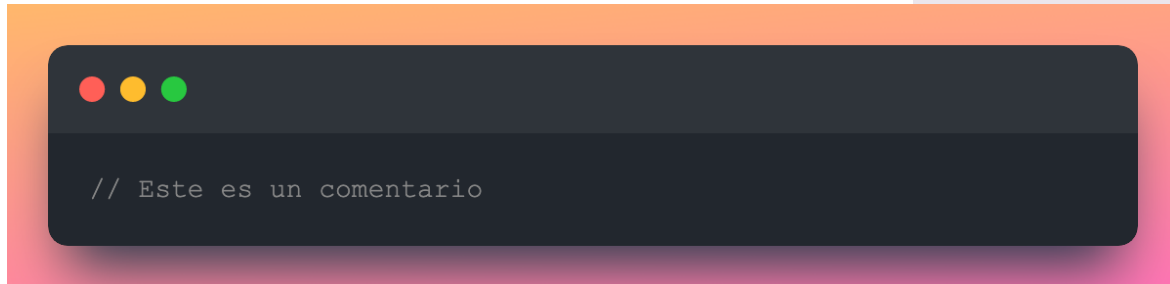
Los comentarios en C# son líneas de texto que no son interpretadas como parte del código. Se usan para proporcionar información adicional sobre el código, como explicaciones, notas, o instrucciones para el desarrollador. Los comentarios también se pueden usar para deshabilitar código temporalmente, sin tener que eliminarlo completamente.

## 4 TIPOS Y VARIABLES

---

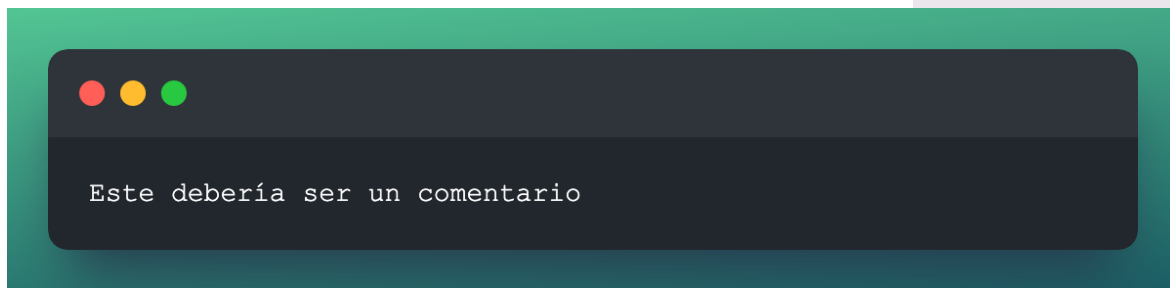
Ejemplo:

Este código es un comentario.



Reto:

Comenta esta línea para que no cause errores y se lea como comentario:



## 4 Tipos y variables

### 4.1 Variables

La manipulación de datos es una tarea fundamental de un lenguaje de programación. Para trabajar con datos necesitamos guardarlos en variables.

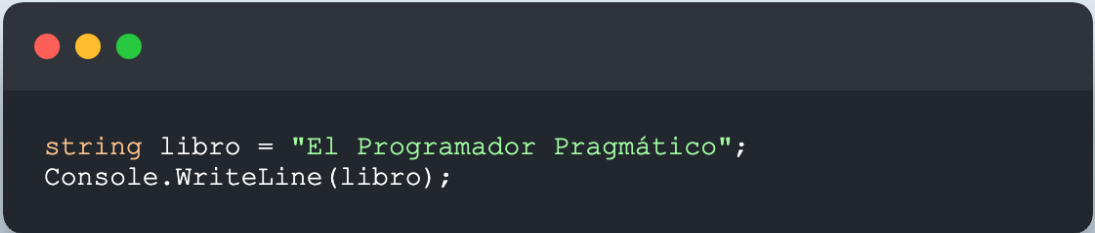
Una variable es un contenedor para almacenar datos que retiene su nombre y puede cambiar su valor a lo largo del tiempo. En los siguientes ejemplos vamos a ver varios tipos de datos que puedes guardar en variables.

## 4 TIPOS Y VARIABLES

---

Ejemplo:


Este código declara una variable llamada “libro” y le asigna el valor de una cadena de texto que contiene el título de un libro. Luego, imprime el valor de la variable “libro” en la consola.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in C# and is as follows:

```
string libro = "El Programador Pragmático";  
Console.WriteLine(libro);
```

```
string libro = "El Programador Pragmático";  
Console.WriteLine(libro);
```

Texto es un tipo de dato útil para guardar información como números telefónicos y colores, entre otros. Este código asigna dos variables, una llamada telf que contiene un número de teléfono como una cadena de caracteres, y otra llamada color que contiene el color amarillo como una cadena de caracteres.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in C# and is as follows:

```
string telf = "406-234 2342";  
string color = "amarillo";
```

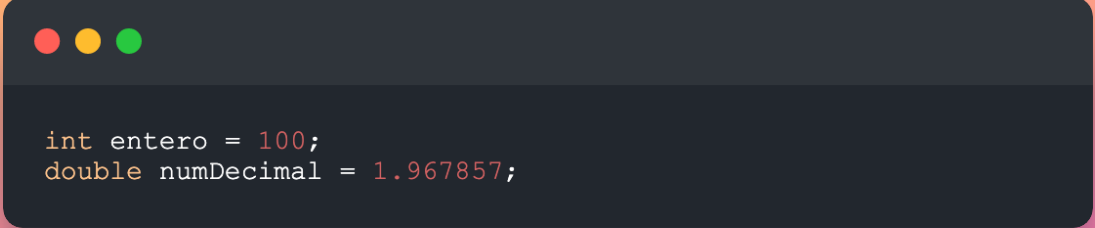
```
string telf = "406-234 2342";  
string color = "amarillo";
```

También podemos guardar datos como números enteros y decimales. Estos datos se usan para realizar operaciones matemáticas y representar valores de peso, dinero, entre otros. Este código asigna dos variables, una con un valor entero (100) y otra con un valor decimal (1.967857).



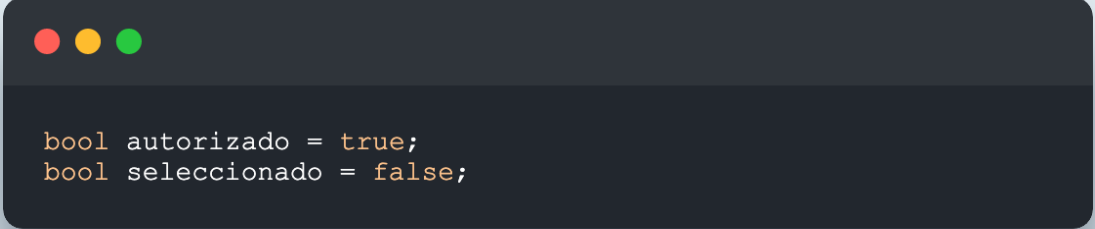
## 4 TIPOS Y VARIABLES

---



```
int entero = 100;  
double numDecimal = 1.967857;
```

El tipo de dato booleano representa los valores de verdadero y falso. Este tipo de datos es útil, por ejemplo, para indicar si un usuario está autorizado a acceder a una app o no, entre varios usos. Este código crea una variable llamada “autorizado” que es verdadera y otra variable llamada “seleccionado” que es falsa.



```
bool autorizado = true;  
bool seleccionado = false;
```

Es importante saber que, en el mundo del código binario, el número 1 representa verdadero y 0 representa falso.

Reto:

Crea una variable “a” con 33 como texto e imprímela a la consola.

### 4.2 Listas

Las listas en C# son una estructura de datos que nos permite almacenar una colección ordenada de elementos. Estos elementos pueden ser de cualquier tipo, desde

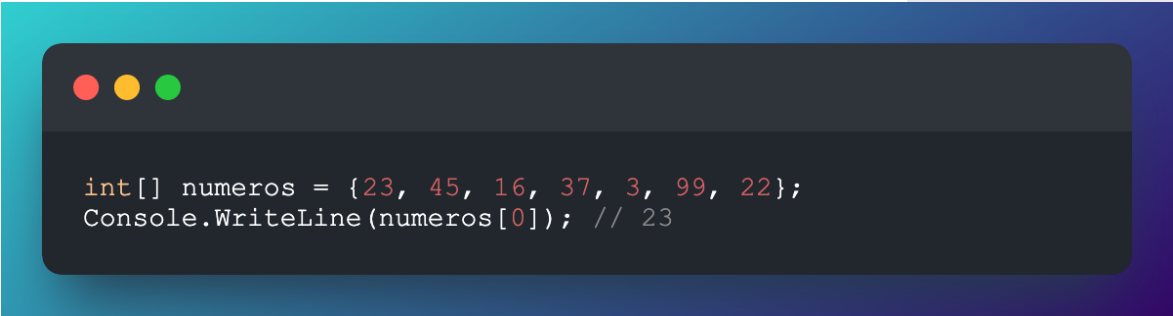
## 4 TIPOS Y VARIABLES

---

números hasta sublistas. También los vas a encontrar con otros nombres como arreglos, matrices, arrays, etc.


Ejemplo:

Este código está imprimiendo el primer elemento de una matriz de números a la consola. En este caso, el primer elemento es 23.

A screenshot of a code editor with a dark background and a blue-to-purple gradient border. It shows two lines of C# code: `int[] numeros = {23, 45, 16, 37, 3, 99, 22};` and `Console.WriteLine(numeros[0]); // 23`. The code is color-coded: `int[]` is blue, `numeros` is black, `=` is black, `{` is black, `23` is red, `45` is red, `16` is red, `37` is red, `3` is red, `99` is red, `22` is red, `};` is black, `Console.WriteLine` is blue, `(numeros[0])` is black, `;` is black, `//` is green, and `23` is red.


```
int[] numeros = {23, 45, 16, 37, 3, 99, 22};
Console.WriteLine(numeros[0]); // 23
```

También existen listas de texto. Este código crea una variable llamada “animales” que contiene una lista de elementos, en este caso, tres animales: perro, gato y tigre.

A screenshot of a code editor with a dark background and a blue-to-purple gradient border. It shows one line of C# code: `string[] animales = { "perro", "gato", "tigre" };`. The code is color-coded: `string[]` is blue, `animales` is black, `=` is black, `{` is black, `"perro"` is green, `"gato"` is green, `"tigre"` is green, `}` is black, and `;` is black.

```
string[] animales = { "perro", "gato", "tigre" };
```

Y esta es una lista de datos mixtos. Este código crea una variable llamada “datosMixtos” que contiene una lista de elementos de diferentes tipos, como texto, números, booleanos y otra lista.



```
dynamic[] datosMixtos = { "texto", "69", "true", new string[]  
    { "lista dentro de otra lista" } };
```

Reto:

Crea una lista en C# que contenga los nombres de los meses del año. Accede al 3er índice e imprímelo en la consola.

### 4.3 Objetos

Los objetos en C# son una forma de almacenar y organizar datos mapeándolos de uno a uno. Estos datos se almacenan en forma de pares clave-valor, donde la clave suele ser una cadena de texto y el valor puede ser cualquier tipo de dato. También los vas a encontrar con otros nombres como mapas, diccionarios, etc.

Ejemplo:

Este código crea un objeto llamado jugadores que contiene dos pares clave-valor. El primer par clave-valor es 10: 'Messi' y el segundo es 7: 'Cristiano Ronaldo' . Luego, se imprime el valor asociado con la clave 10, que es 'Messi' .

## 4 TIPOS Y VARIABLES

```
Dictionary<int, string> jugadores = new Dictionary<int, string>
{
    { 10, "Messi" },
    { 7, "Cristiano Ronaldo" }
};
Console.WriteLine(jugadores[10]); // 'Messi'
```

También podemos mapear de texto a texto. Este código crea un objeto llamado “países” que contiene claves y valores. Las claves son códigos de países (EC, MX, AR) y los valores son los nombres de los países (Ecuador, México, Argentina).

```
Dictionary<string, string> paises = new Dictionary<string, string>
{
    {"EC", "Ecuador"},
    {"MX", "México"},
    {"AR", "Argentina"}
};
```

E incluso podemos mapear de texto a listas de texto, entre muchas otras opciones. Este código establece un objeto llamado “emails” que contiene dos pares clave-valor. Cada clave es un nombre de persona y el valor es un arreglo de direcciones de correo electrónico asociadas con esa persona.

## 4 TIPOS Y VARIABLES

---

```
Dictionary<string, List<string>> emails = new
Dictionary<string, List<string>>()
{
    { "Juan", new List<string> { "juan@gmail.com" } },
    { "Ricardo", new List<string> { "ricardo@gmail.com",
"rick@aol.com" } }
};
```

Reto:

Crea un objeto en C# que represente una computadora, con sus respectivas características (marca, modelo, procesador, memoria RAM, etc).

### 4.4 Constantes

Las constantes son variables que no pueden ser reasignadas. Esto significa que una vez que se asigna un valor a una constante, este no puede ser cambiado.

Ejemplo:

Esta línea de código establece una constante llamada “pi” con un valor de 3.14159265359. Esta constante se puede usar para almacenar un valor numérico que no cambiará a lo largo del programa.

```
const double pi = 3.14159265359;
```

## 5 OPERADORES

---

Reto:

Crea una constante llamada 'SALUDO' y asígnale el valor 'Hola Planeta'. Luego, imprime el valor de la constante en la consola.

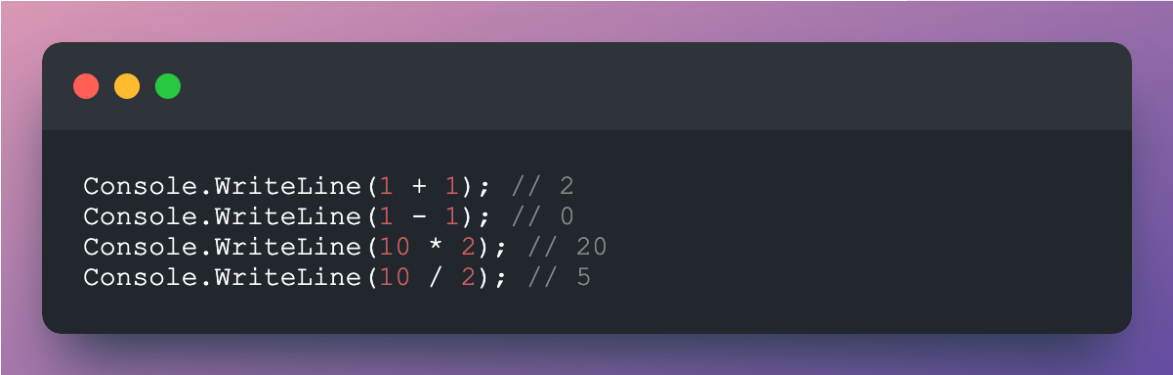
## 5 Operadores

### 5.1 Operadores de aritméticos

Los operadores aritméticos en C# son usados para realizar operaciones matemáticas básicas. Estos operadores incluyen sumar, restar, multiplicar, dividir, entre otros.

Ejemplo:

Este código realiza operaciones matemáticas básicas, imprimiendo los resultados en la consola. Estas operaciones incluyen suma, resta, multiplicación y división.

A screenshot of a C# console application window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The console displays four lines of code, each followed by a comment showing the result of the operation.

```
Console.WriteLine(1 + 1); // 2
Console.WriteLine(1 - 1); // 0
Console.WriteLine(10 * 2); // 20
Console.WriteLine(10 / 2); // 5
```

Reto:

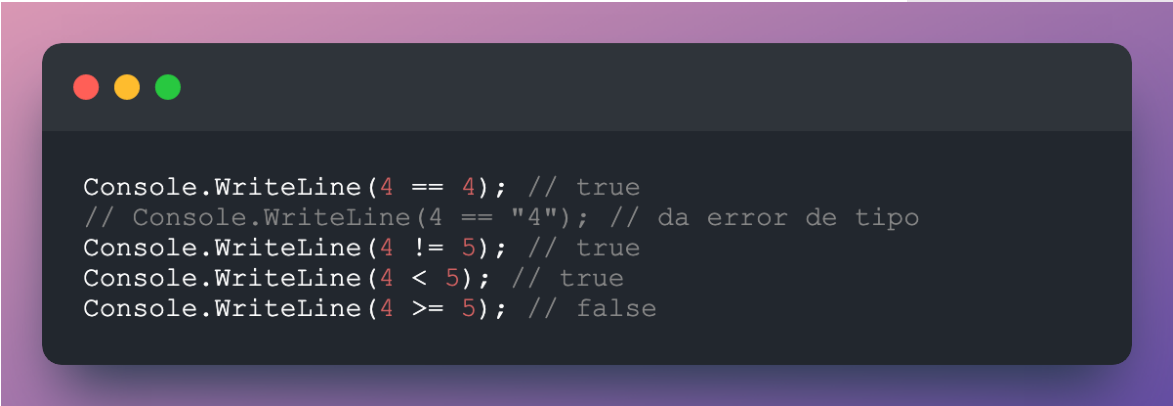
Usa los operadores aritméticos para calcular el área de un círculo con radio 5.

### 5.2 Operadores comparativos

Los operadores comparativos en C# son usados para comparar dos valores y determinar si son iguales o diferentes. Estos operadores son útiles para determinar si dos valores son iguales, si dos valores no son iguales, si un valor es mayor o menor que otro, entre otros.

Ejemplo:

Este código muestra cómo se usan los operadores para comparar valores.

A screenshot of a code editor window with a dark background and light text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code inside the editor is as follows:

```
Console.WriteLine(4 == 4); // true
// Console.WriteLine(4 == "4"); // da error de tipo
Console.WriteLine(4 != 5); // true
Console.WriteLine(4 < 5); // true
Console.WriteLine(4 >= 5); // false
```

Reto:

Escribe un programa que compare dos números 'a' y 'b' y determina si 'a' con el valor de 4 es mayor, menor o igual a 'b' con un valor de 2.

### 5.3 Operadores lógicos

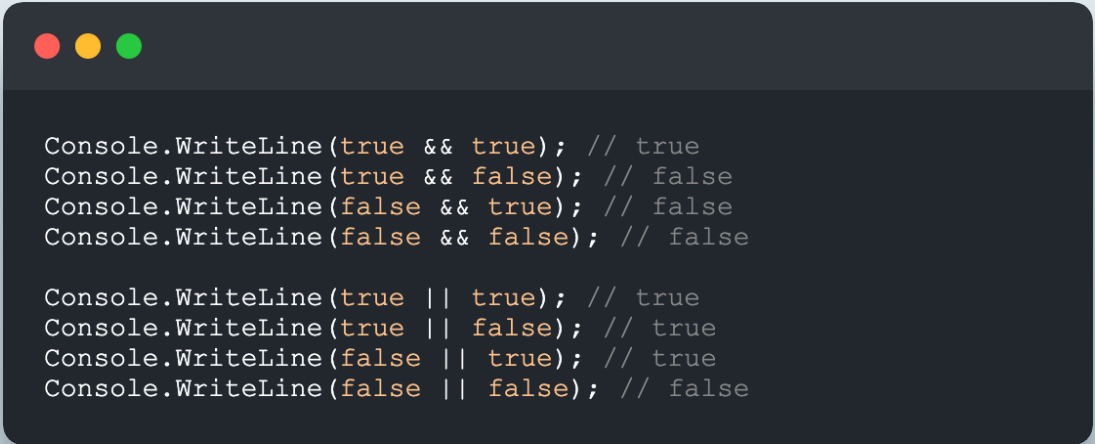
Los operadores lógicos en C# se usan para realizar comparaciones entre valores y devolver un valor booleano (verdadero o falso). Estos operadores pueden ser útiles, por ejemplo, si deseamos saber si un 'animal' es un gato y es un mamífero (al mismo tiempo).

Ejemplo:

## 6 CONDICIONALES

---

Este código muestra el uso de los operadores lógicos 'y' y 'o' para evaluar expresiones booleanas. El operador 'y' devuelve verdadero si ambas expresiones son verdaderas, de lo contrario devuelve falso. El operador 'o' devuelve verdadero si al menos una de las expresiones es verdadera, de lo contrario devuelve falso.

A screenshot of a code editor window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in C# and demonstrates the use of logical operators 'y' (AND) and 'o' (OR) with Console.WriteLine statements. The code is as follows:

```
Console.WriteLine(true && true); // true
Console.WriteLine(true && false); // false
Console.WriteLine(false && true); // false
Console.WriteLine(false && false); // false

Console.WriteLine(true || true); // true
Console.WriteLine(true || false); // true
Console.WriteLine(false || true); // true
Console.WriteLine(false || false); // false
```

Reto:

Escribe una línea de código que devuelva verdadero si 'x' es mayor que 0 y 'y' es menor que 0.

## 6 Condicionales

### 6.1 Condición única

Los condicionales son estructura de control de flujo en C#, es decir, controlan el flujo del código. Permiten ejecutar una sección de código si una condición es verdadera. También permiten ejecutar otra sección de código, si la condición es falsa.

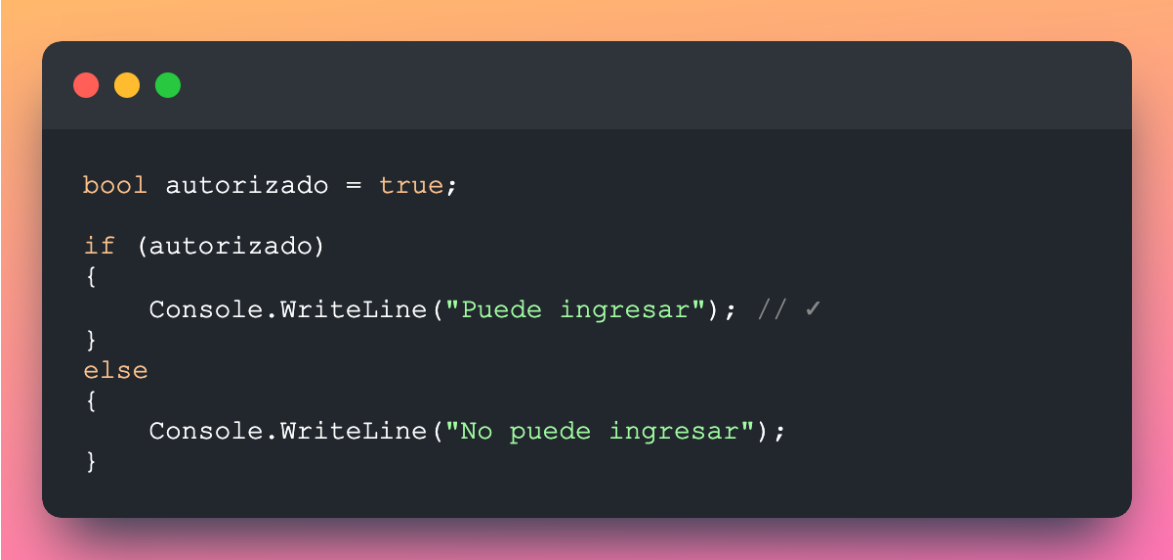
Ejemplo:



## 6 CONDICIONALES

---

Este código comprueba si una variable booleana (autorizado) es verdadera. Si es así, imprime un mensaje en la consola indicando que el usuario puede ingresar. Si no es así, imprime un mensaje indicando que el usuario no puede ingresar.



```
bool autorizado = true;

if (autorizado)
{
    Console.WriteLine("Puede ingresar"); // ✓
}
else
{
    Console.WriteLine("No puede ingresar");
}
```

Reto:

Escribe código condicional que revise si un número 'a' es par o impar. La variable 'a' tiene el valor de 17 y debe imprimir a la consola 'Es par' si es par o 'Es impar' si es impar.

### 6.2 Múltiples condiciones

Adicionalmente, los condicionales permiten ejecutar varias secciones de código de acuerdo a varias condiciones.

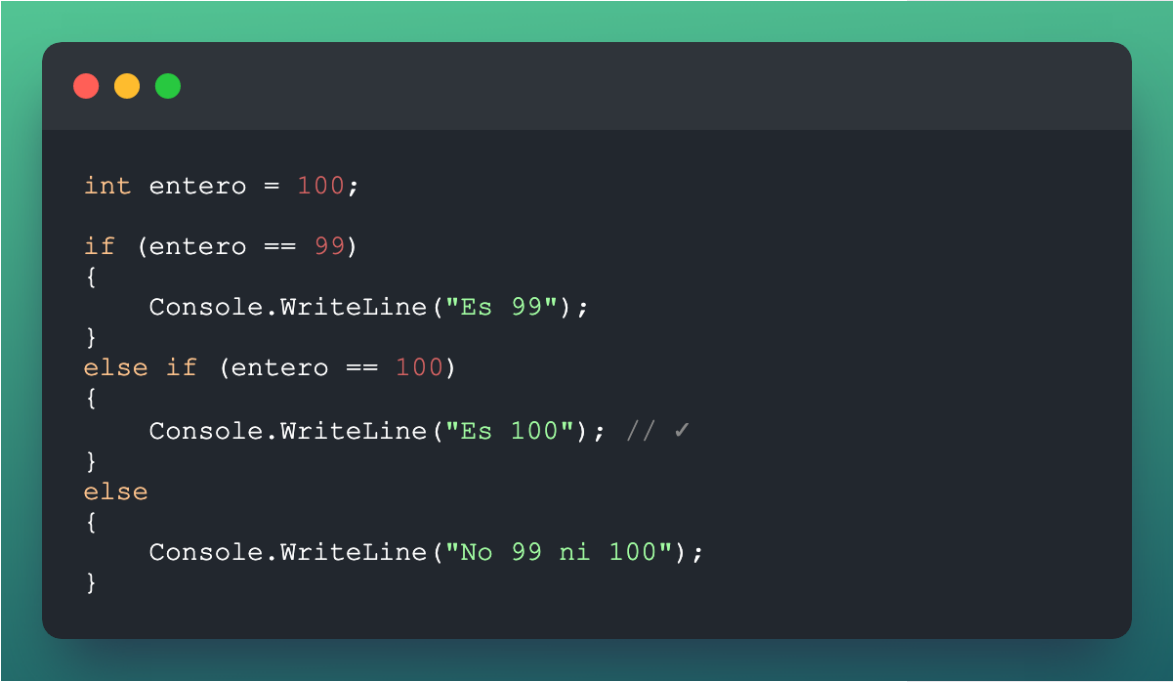
Ejemplo:

En este caso podemos ver que el código que se ejecuta depende de varios valores. Este código comprueba si una variable llamada entero es igual a 99 o 100. Si es

## 6 CONDICIONALES

---

igual a 99, imprime “Es 99” en la consola. Si es igual a 100, imprime “Es 100” en la consola. Si no es ni 99 ni 100, imprime “No 99 ni 100” en la consola.



```
int entero = 100;

if (entero == 99)
{
    Console.WriteLine("Es 99");
}
else if (entero == 100)
{
    Console.WriteLine("Es 100"); // ✓
}
else
{
    Console.WriteLine("No 99 ni 100");
}
```

Este condicional es útil cuando hay una gran cantidad de posibles resultados para una expresión. Este código compara una variable (color) con diferentes valores y ejecuta una acción dependiendo del resultado de la comparación. En este caso, si la variable color es igual a ‘amarillo’, se imprimirá ‘Advertencia’ en la consola.

```
string color = "amarillo";

switch (color)
{
    case "verde":
        Console.WriteLine("Éxito");
        break;
    case "amarillo":
        Console.WriteLine("Advertencia"); // ✓
        break;
    default:
        Console.WriteLine("Error");
        break;
}
```

Reto:

Crea un programa que evalúe una variable 'numero' y dependiendo del resultado, imprima un mensaje diferente. Si el número es mayor a 10, imprime "El número es mayor a 10" . Si el número es menor a 10, imprime "El número es menor a 10" . Si el número es igual a 10, imprime "El número es igual a 10" .

## 7 Bucles

### 7.1 Bucle for

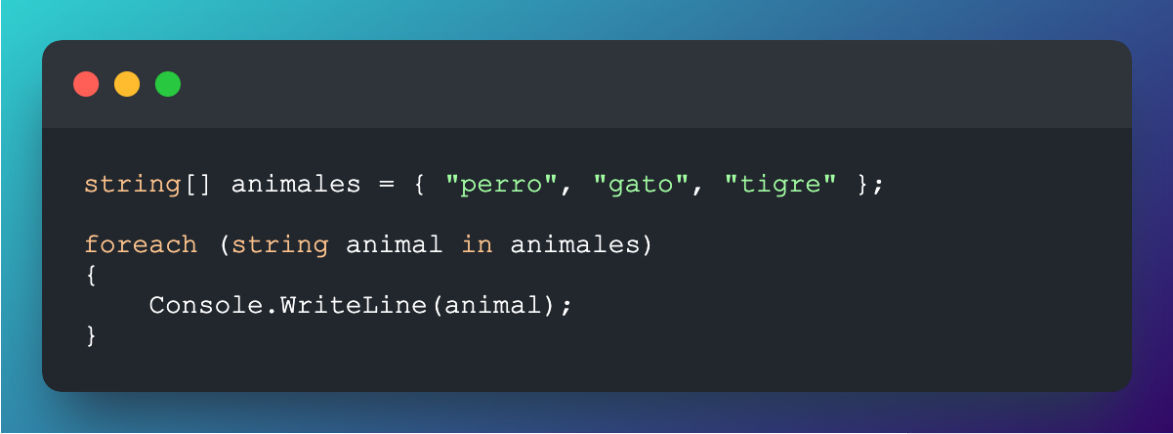
Los bucles son otra estructura de control de flujo que se utilizan para iterar sobre una secuencia de elementos. También se los llama loops o ciclos.

Ejemplo:

## 7 BUCLES

---

Este código itera sobre una lista de animales e imprime cada elemento de la lista en la consola.



```
string[] animales = { "perro", "gato", "tigre" };  
  
foreach (string animal in animales)  
{  
    Console.WriteLine(animal);  
}
```

Reto:

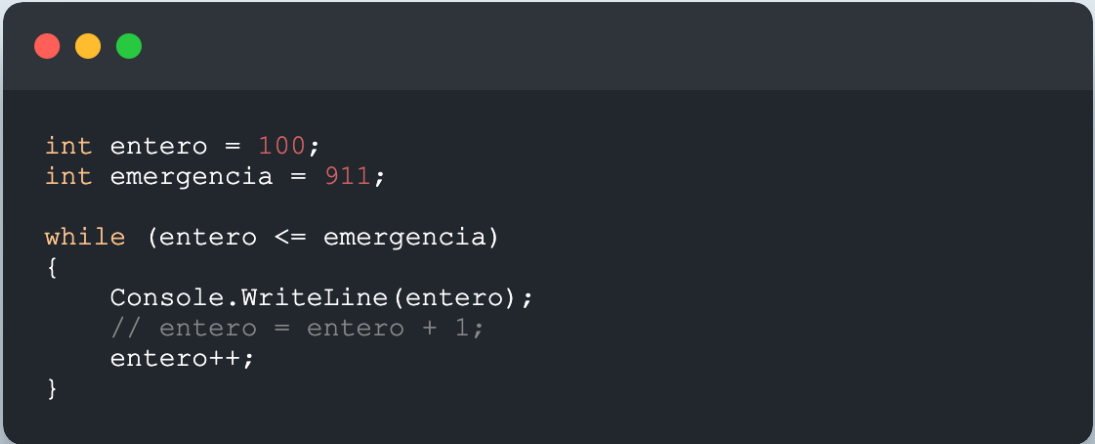
Escribe un bucle que imprima los números del 1 al 10 en orden ascendente.

### 7.2 Bucle while

while es un tipo de bucle en C# que se usa para ejecutar una serie de instrucciones mientras se cumpla una condición. Esta condición se evalúa antes de cada iteración del bucle.

Ejemplo:

Este código establece un bucle while que imprime los números enteros desde 100 hasta 911 en la consola. El bucle while se ejecutará mientras la variable entero sea menor o igual a la variable emergencia. Cada vez que el bucle se ejecuta, la variable entero se incrementa en 1.



```
int entero = 100;
int emergencia = 911;

while (entero <= emergencia)
{
    Console.WriteLine(entero);
    // entero = entero + 1;
    entero++;
}
```

Ten cuidado de no incluir una condición para parar el bucle. Esto podría seguir corriendo tu código indefinidamente hasta congelar tu computador.

Reto:

Crea un bucle while que imprima los números del 1 al 10 en la consola.

## 8 Funciones

En C#, una función es un bloque de código diseñado para realizar una tarea específica y se puede reutilizar en varias partes el código. Las funciones se pueden definir para realizar cualquier tarea, desde realizar cálculos hasta mostrar mensajes en la pantalla.

Ejemplo:

Esta función toma dos argumentos (primero y segundo) de forma dinámica y los suma. Luego, imprime el resultado en la consola. Esta función se llama dos veces con diferentes argumentos para mostrar los resultados de la suma.

## 8 FUNCIONES

```
void Sumar(int primero, int segundo)
{
    Console.WriteLine(primeros + segundos);
}

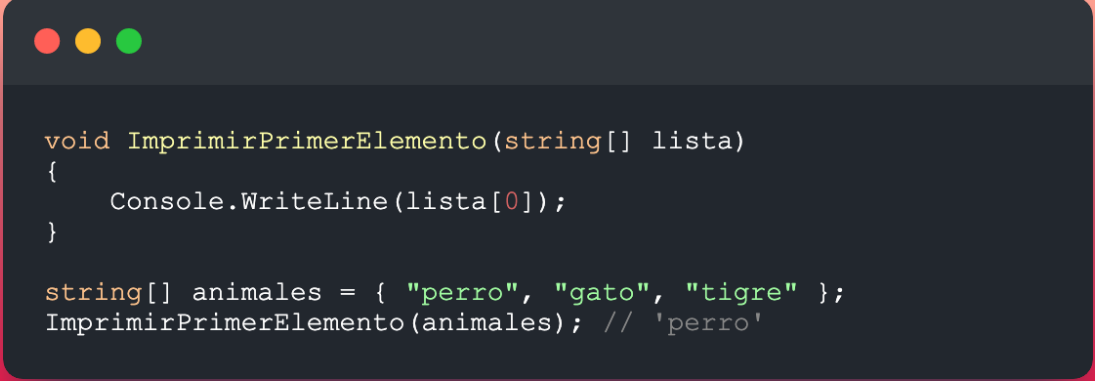
Sumar(2, 2); // 4
Sumar(3, 4); // 7
```

También puedes crear funciones que retornen un valor. Esto significa que una vez que se ejecuta una función, el valor devuelto se puede usar en otra parte del código. Esta función toma dos argumentos (primero y segundo) y los multiplica para devolver el resultado. En este caso, el resultado es 6.

```
int Multiplicar(int primero, int segundo)
{
    return primero * segundo;
}

int resultado = Multiplicar(3, 2);
Console.WriteLine(resultado); // 6
```

Esta función imprime el primer elemento de una lista. En este caso, la lista es una lista de animales, y la función imprime el primer elemento de la lista, que es 'perro'. Funciones como esta son útiles para evitar la repetición de código y para ahorrar tiempo.



```
void ImprimirPrimerElemento(string[] lista)
{
    Console.WriteLine(lista[0]);
}

string[] animales = { "perro", "gato", "tigre" };
ImprimirPrimerElemento(animales); // 'perro'
```

Finalmente, puedes ver otro ejemplo de una función bastante compleja. Esta función implementa el algoritmo de ordenamiento QuickSort para ordenar una lista de elementos. El algoritmo comienza tomando un elemento de la lista como pivote y luego coloca los elementos menores que el pivote a su izquierda y los elementos mayores a su derecha. Luego, se aplica el mismo algoritmo a las sublistas izquierda y derecha hasta que la lista esté ordenada. No tienes que entender todo lo que hace internamente pero te dará una idea de la complejidad que pueden tener programas como apps con miles de funciones.

```
int[] Quicksort(int[] lista)
{
    if (lista.Length <= 1)
    {
        return lista;
    }
    int pivote = lista[0];
    List<int> izquierda = new List<int>();
    List<int> derecha = new List<int>();
    for (int i = 1; i < lista.Length; i++)
    {
        if (lista[i] < pivote)
        {
            izquierda.Add(lista[i]);
        }
        else
        {
            derecha.Add(lista[i]);
        }
    }
    return Quicksort(izquierda.ToArray()).Concat(new int[] {
pivote }).Concat(Quicksort(derecha.ToArray())).ToArray();
}

int[] numeros = { 23, 45, 16, 37, 3, 99, 22 };
int[] listaOrdenada = Quicksort(numeros);
Console.WriteLine(string.Join(", ", listaOrdenada)); // [ 3,
16, 22, 23, 37, 45, 99 ]
```

Reto:

Escribe una función que tome dos números como argumentos y devuelva el mayor de los dos.



## 9 Programación orientada a objetos (POO)

### 9.1 Paradigma

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en la creación de objetos que contienen datos y funcionalidades. Estos objetos se relacionan entre sí para formar una estructura de datos compleja. Los lenguajes de programación orientados a objetos permiten que los programadores puedan crear objetos y usarlos para construir aplicaciones.

### 9.2 Clases

Las clases en C# son una forma de definir objetos creando una plantilla. Se pueden usar para crear objetos con propiedades y métodos similares. Las propiedades son valores que pertenecen al objeto y los métodos son funciones que pertenecen al objeto.

Ejemplo:

En este ejemplo creamos una clase llamada 'Lenguaje' que inicializamos con las propiedades 'nombre' y 'año'. Adicionalmente creamos el método 'descripción' que imprime un texto usando las propiedades previas. Con esa clase podemos crear una instancia para el lenguaje 'HTML' y otra para 'CSS'. Ahora podemos crear, en teoría, un número infinito de lenguajes sin reescribir los objetos manualmente.

```
Lenguaje html = new Lenguaje("HTML", 1993);
Lenguaje css = new Lenguaje("CSS", 1996);
html.Descripcion(); // 'HTML fue creado en 1993'
css.Descripcion(); // 'CSS fue creado en 1996'

// la clase se declara al final del archivo
public class Lenguaje
{
    public string Nombre { get; set; }
    public int Año { get; set; }

    public Lenguaje(string nombre, int año)
    {
        Nombre = nombre;
        Año = año;
    }

    public void Descripcion()
    {
        Console.WriteLine($" {{Nombre}} fue creado en {{Año}}");
    }
}
```

Reto:

Crea una clase llamada Auto que tenga propiedades como marca, modelo y año. Luego, crea un par de instancias de esta clase.

## 10 Módulos

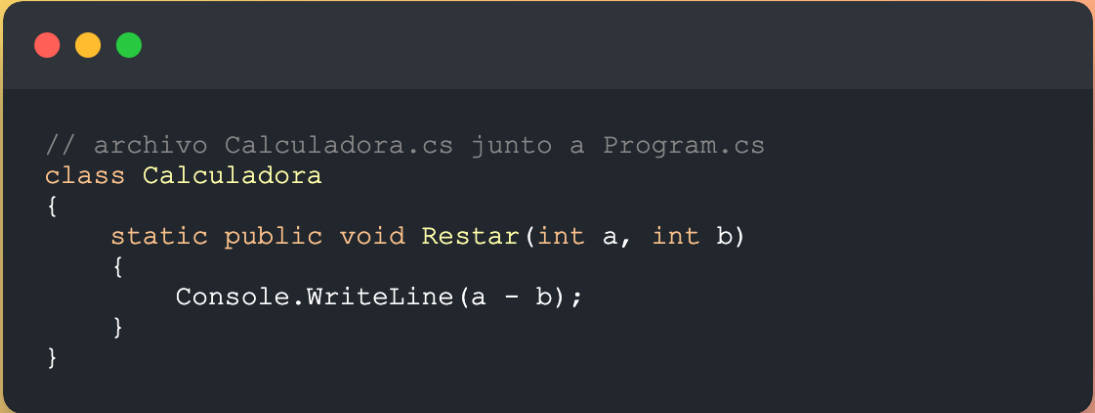
Los módulos son una forma de incluir código externo de otro archivo. Esto permite a los desarrolladores reutilizar código y ahorrar tiempo al escribir código.

## 10 MÓDULOS

---

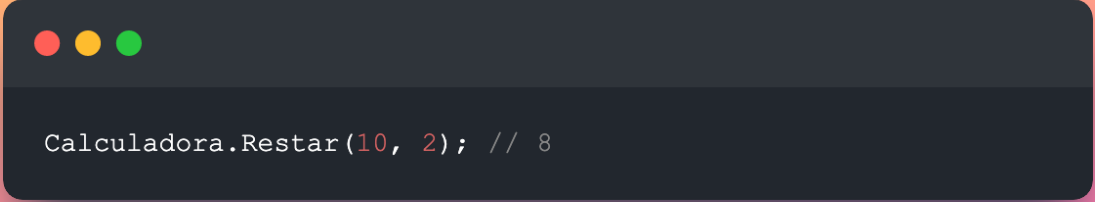
### Ejemplo:

Este archivo define una función que toma dos argumentos (a y b) y luego imprime el resultado de la resta de a y b en la consola. El código exporta la función para que pueda ser usada en otros archivos. Exportar es una forma de compartir código entre archivos. Esto significa que un archivo puede exportar variables, funciones y objetos para que otros archivos los puedan usar.



```
// archivo Calculadora.cs junto a Program.cs
class Calculadora
{
    static public void Restar(int a, int b)
    {
        Console.WriteLine(a - b);
    }
}
```

Y en otro archivo podemos importar y utilizar esta función. Este código importa una función desde el archivo externo y luego la usa para restar 10 menos 2, resultando en 8.



```
Calculadora.Restar(10, 2); // 8
```

### Reto:

Crea un archivo llamado que contenga una función para calcular el área de un rectángulo. Luego, importa la función en otro archivo y usala para calcular el área de

un rectángulo con lados de 3 y 4.

## 11 Siguientes pasos

### 11.1 Herramientas

1. Visual Studio: Es un entorno de desarrollo integrado (IDE) de Microsoft para desarrollar aplicaciones de Windows, web y móviles. Está diseñado para ayudar a los desarrolladores a crear aplicaciones de forma rápida y eficiente.
2. .NET Framework: Es una plataforma de desarrollo de aplicaciones de Microsoft para crear aplicaciones para Windows, web y dispositivos móviles. Está diseñado para ayudar a los desarrolladores a crear aplicaciones de forma rápida y eficiente.
3. Xamarin: Es una plataforma de desarrollo de aplicaciones móviles de Microsoft que permite a los desarrolladores crear aplicaciones nativas para dispositivos iOS, Android y Windows.
4. Mono: Es una plataforma de desarrollo de aplicaciones multiplataforma que permite a los desarrolladores crear aplicaciones para Windows, Mac, Linux y dispositivos móviles.
5. NuGet: Es una herramienta de gestión de paquetes para .NET que permite a los desarrolladores descargar y administrar paquetes de código de terceros para su proyecto.
6. Roslyn: Es un compilador de Microsoft para C# y Visual Basic .NET. Está diseñado para ayudar a los desarrolladores a crear aplicaciones de forma rápida y eficiente.

### 11.2 Recursos

1. [Microsoft Docs](#): Microsoft Docs es una de las mejores fuentes de información para C#. Ofrece documentación detallada sobre la sintaxis, el uso de la biblioteca de clases, la programación orientada a objetos y mucho más.
2. [Stack Overflow](#): Stack Overflow es una comunidad de desarrolladores donde puedes encontrar respuestas a preguntas relacionadas con C#.
3. [Tutoriales de C#](#): Tutoriales de C# es una excelente fuente de información para principiantes y desarrolladores avanzados. Ofrece tutoriales paso a paso sobre la sintaxis, la programación orientada a objetos, la programación web y mucho más.
4. [C# Corner](#): C# Corner es una comunidad de desarrolladores donde puedes encontrar artículos, tutoriales, código de muestra y mucho más.
5. [Channel 9](#): Channel 9 es una excelente fuente de información para desarrolladores de C#. Ofrece videos, tutoriales, charlas y mucho más.

### 11.3 ¿Que viene después?

1. Familiarizarse con el lenguaje de programación C#.
2. Comprender los conceptos básicos de programación, como variables, bucles, condicionales y funciones.
3. Desarrollar habilidades de depuración y pruebas.
4. Aprender a usar herramientas de desarrollo como Visual Studio.
5. Aprender a usar bibliotecas y frameworks de C#.
6. Desarrollar habilidades de diseño de software.
7. Aprender a usar patrones de diseño y principios de programación.
8. Desarrollar habilidades de programación orientada a objetos.
9. Aprender a usar bases de datos relacionales.
10. Desarrollar habilidades de análisis y diseño de algoritmos.

### 11.4 Preguntas de entrevista

1. ¿Qué es C#?

- C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft como parte de su .NET Framework.

2. ¿Qué es .NET Framework?

- .NET Framework es una plataforma de desarrollo de aplicaciones de Microsoft que proporciona un entorno de ejecución para aplicaciones desarrolladas en lenguajes como C#, Visual Basic y F#.

3. ¿Qué es una clase en C#?

- Una clase es una plantilla para crear objetos. Contiene variables y métodos para definir el comportamiento de los objetos creados a partir de la clase.

4. ¿Qué es un objeto en C#?

- Un objeto es una instancia de una clase. Los objetos contienen los datos y el comportamiento definidos por la clase.

5. ¿Qué es una variable en C#?

- Una variable es un contenedor para almacenar un valor. Las variables tienen un nombre, un tipo y un valor.

6. ¿Qué es un método en C#?

- Un método es una función que se puede llamar desde otro lugar en el código. Los métodos contienen lógica para realizar una tarea específica.

7. ¿Qué es una interfaz en C#?

- Una interfaz es una plantilla para definir el comportamiento de una clase. Las interfaces contienen métodos, propiedades y eventos que deben ser implementados por las clases que implementan la interfaz.

## 11 SIGUIENTES PASOS

---

### 8. ¿Qué es una herencia en C#?

- La herencia es un mecanismo que permite a una clase heredar los miembros de otra clase. Esto permite a las clases reutilizar el código existente y extender la funcionalidad de la clase base.

### 9. ¿Qué es una sobrecarga de métodos en C#?

- La sobrecarga de métodos es un mecanismo que permite a una clase tener múltiples versiones de un método con diferentes parámetros. Esto permite a la clase tener múltiples versiones del mismo método para realizar diferentes tareas.

### 10. ¿Qué es una sobreescritura de métodos en C#?

- La sobreescritura de métodos es un mecanismo que permite a una clase reemplazar el comportamiento de un método heredado de una clase base. Esto permite a la clase personalizar el comportamiento del método heredado.

### 11. ¿Qué es una excepción en C#?

- Una excepción es una situación en la que una aplicación no puede realizar una tarea específica. Las excepciones se pueden controlar mediante el uso de bloques try/catch para recuperar la aplicación de la situación excepcional.

### 12. ¿Qué es una colección en C#?

- Una colección es una estructura de datos que se usa para almacenar y recuperar datos. Las colecciones se pueden usar para almacenar y recuperar datos de forma eficiente.

### 13. ¿Qué es una enumeración en C#?

- Una enumeración es un tipo de datos que contiene una lista de valores predefinidos. Las enumeraciones se usan para definir un conjunto de valores constantes que se pueden usar en un programa.

## 11 SIGUIENTES PASOS

---

### 14. ¿Qué es una estructura en C#?

- Una estructura es un tipo de datos que contiene un conjunto de variables. Las estructuras se usan para almacenar datos relacionados que se pueden usar en un programa.

### 15. ¿Qué es una matriz en C#?

- Una matriz es una estructura de datos que se usa para almacenar un conjunto de valores del mismo tipo. Las matrices se pueden usar para almacenar y recuperar datos de forma eficiente.

### 16. ¿Qué es una cadena en C#?

- Una cadena es un tipo de datos que contiene una secuencia de caracteres. Las cadenas se usan para almacenar y manipular texto en un programa.

### 17. ¿Qué es una expresión regular en C#?

- Una expresión regular es un patrón de caracteres que se usa para buscar y reemplazar cadenas de texto. Las expresiones regulares se usan para realizar búsquedas y reemplazos avanzados en cadenas de texto.

### 18. ¿Qué es LINQ en C#?

- LINQ (Language Integrated Query) es una tecnología que permite a los programadores escribir consultas en lenguaje de programación para recuperar datos de una fuente de datos.

### 19. ¿Qué es una cláusula using en C#?

- Una cláusula using es una instrucción que se usa para importar un espacio de nombres en un programa. Esto permite a los programadores usar los tipos definidos en el espacio de nombres sin tener que especificar el espacio de nombres completo.

### 20. ¿Qué es un delegado en C#?



## 11 SIGUIENTES PASOS

---

- Un delegado es un tipo de datos que se usa para almacenar una referencia a un método. Los delegados se usan para pasar métodos como parámetros a otros métodos.