Introducción a C#

CONCEPTOS BÁSICOS

Objetivo

- entender y aplicar los conceptos básicos de C#
- Comprender los tipos de datos
- estructuras de control
- interactuar con la consola para leer y mostrar información

conceptos básicos de C#

- Variables
- Tipos de Datos
- Operadores
- Estructuras de Control
- Métodos (Funciones)
- Entrada y Salida de Datos
- Clases y Objetos (POO en C#)

Tipos de Variables

- Números Enteros
  - Byte (8 bits, de 0 a 255)
    - byte edad = 25;
  - Short (16 bits, de -32,768 a 32,767)
    - short temperatura = -10;
  - Int: (32 bits, de -2,147,483,648 a 2,147,483,647)
    - int año = 2024;
  - Long: (64 bits, rango mucho mayor)
    - long poblacion = 7800000000;

Tipos de Variables

- Números Decimales
  - Float: (32 bits, precisión de 7 decimales, usa ‘f’ al final)
    - float precio = 10.99f;
  - Double: (64 bits, precisión de 15-16 decimales)
    - double pi = 3.1415926535;
  - decimal: (128 bits, ideal para valores financieros, usa ‘m’ al final)
    - decimal saldo = 10000.75m;

Tipos de Variables

- Tipo Carácter
  - char: (un solo carácter, se usa comillas simples ‘ ’)
    - char inicial = 'A';

Tipos de Variables

- Tipo Booleano
  - bool: (Valores True o False)
    - bool esMayor = true;

Tipos de Datos por Referencia (Reference Types)

- Cadenas de Texto
  - String: (almacena texto, usa comillas dobles “ ”)
    - `string nombre = "Juan Pérez";`
  - Arreglos (Arrays) - Conjunto de elementos del mismo tipo.
    - `int[] numeros = {1, 2, 3, 4, 5};`
  - Object: (Cualquier Tipo)
    - `object valor = "Hola"; // Puede cambiar de tipo`
  - Dynamic: (Tipo Dinámico) - El tipo se determina en tiempo de ejecución.
    - `dynamic dato = 10;`
    - `dato = "Ahora soy un texto"; // Se puede cambiar`

Operadores en C#

- Operadores Aritméticos

Operador	Descripción	Ejemplo
`+`	Suma	`int suma = 5 + 3; // Resultado: 8`
`-`	Resta	`int resta = 10 - 4; // Resultado: 6`
`*`	Multiplicación	`int multiplicacion = 6 * 2; // Resultado: 12`
`/`	División	`int division = 9 / 3; // Resultado: 3`
`%`	Módulo (Residuo)	`int residuo = 10 % 3; // Resultado: 1`

Ejemplo de uso:

```
int a = 10, b = 3;
Console.WriteLine("Suma: " + (a + b));
Console.WriteLine("Módulo: " + (a % b));
```

Operadores Relacionales (de Comparación)

- Se utilizan para comparar valores y devuelven un resultado booleano

Operador	Descripción	Ejemplo
`==`	Igual a	`5 == 5 // true`
`!=`	Diferente de	`5 != 3 // true`
`>`	Mayor que	`10 > 5 // true`
`<`	Menor que	`3 < 7 // true`
`>=`	Mayor o igual	`5 >= 5 // true`
`<=`	Menor o igual	`4 <= 6 // true`

Ejemplo de uso:

```
int x = 8, y = 3;
bool resultado = x > y; // true
Console.WriteLine("¿X es mayor que Y? " + resultado);
```

Operadores Lógicos

- Se usan para combinar expresiones booleanas.

Operador	Descripción	Ejemplo
`&&`	AND (Y lógico)	`(5 > 3) && (10 > 2) // true`
`		`
`!`	NOT (Negación)	`!(5 > 3) // false`

Ejemplo de uso:

```
bool a = true, b = false;
Console.WriteLine("AND: " + (a && b)); // false
Console.WriteLine("OR: " + (a || b));  // true
Console.WriteLine("NOT: " + !a);       // false
```

Estructuras de Control en C# (if-else)

- Las estructuras de control permiten ejecutar bloques de código según condiciones o repeticiones.

```
int edad = 20;
if (edad >= 18) {
    Console.WriteLine("Eres mayor de edad");
} else {
    Console.WriteLine("Eres menor de edad");
}
```

- Ejecuta código si una condición es TRUE

Condicional SWITCH

- Se usa cuando hay múltiples opciones posibles.

```
int opcion = 2;
switch (opcion) {
    case 1:
        Console.WriteLine("Opción 1 seleccionada");
        break;
    case 2:
        Console.WriteLine("Opción 2 seleccionada");
        break;
    default:
        Console.WriteLine("Opción no válida");
        break;
}
```

Bucles

- For: (se ejecuta un número fijo de veces)

```
for (int i = 1; i <= 5; i++) {
    Console.WriteLine("Iteración " + i);
}
```

- While: (se ejecuta mientras la condición sea TRUE)

```
int n = 1;
while (n <= 3) {
    Console.WriteLine("Número: " + n);
    n++;
}
```

- Do-while: (se ejecuta al menos una vez)

```
int num = 1;
do {
    Console.WriteLine("Número: " + num);
    num++;
} while (num <= 3);
```

Métodos (Funciones) en C#

- Un método es un bloque de código que realiza una tarea específica y puede recibir parámetros.

```
static void Saludar() {
    Console.WriteLine("¡Hola, bienvenido a C#!");
}
```

- *Ejemplo de método sin parámetros*

Métodos (Funciones) en C#

- Un método es un bloque de código que realiza una tarea específica y puede recibir parámetros.

```
static int Sumar(int a, int b) {
    return a + b;
}

static void Main() {
    int resultado = Sumar(5, 3);
    Console.WriteLine("Resultado: " + resultado);
}
```

- *Ejemplo de método con parámetros y retorno de valores*

Entrada y Salida de Datos

- Ejemplo de Console.ReadLine() y Console.WriteLine()

```
Console.Write("Ingresa tu nombre: ");
string nombre = Console.ReadLine();
Console.WriteLine("Hola, " + nombre + "!");
```

```
Console.Write("Ingresa tu edad: ");
int edad = int.Parse(Console.ReadLine());
Console.WriteLine("El próximo año tendrás " + (edad + 1) + " años.");
```

Clases y Objetos (POO en C#)

- C# es un lenguaje orientado a objetos, lo que significa que se basa en el uso de **clases** y **objetos**.

```
class Persona {
    public string Nombre;
    public int Edad;

    public void Presentarse() {
        Console.WriteLine("Hola, soy " + Nombre + " y tengo " + Edad + " años.");
    }
}

class Program {
    static void Main() {
        // Creación de un objeto
        Persona p = new Persona();
        p.Nombre = "Carlos";
        p.Edad = 25;
        p.Presentarse();
    }
}
```

En C#, la POO se basa en cuatro **principios fundamentales**:

- Encapsulamiento
- Herencia
- Polimorfismo
- Abstracción

Encapsulamiento

- Oculta los detalles internos de un objeto y solo expone lo necesario mediante modificadores de acceso.

- Métodos
  - Private
  - Public
  - Protected

```
class Persona {
    // Atributos privados (ocultos)
    private string nombre;
    private int edad;

    // Propiedad para acceder al nombre de manera controlada
    public string Nombre {
        get { return nombre; }
        set { nombre = value; }
    }
```

  - Los atributos nombre y edad son privados (private).

Herencia

- **Permite que una clase herede atributos y métodos de otra** para evitar duplicación de código.

```
// Clase base (padre)
class Animal {
    public string Nombre;

    public void Comer() {
        Console.WriteLine($"{Nombre} está comiendo.");
    }
}

// Clase derivada (hija)
class Perro : Animal {
    public void Ladrar() {
        Console.WriteLine($"{Nombre} está ladrando: ¡Guau guau!");
    }
}
```

Polimorfismo

- **Permite que un mismo método tenga diferentes comportamientos** según el contexto.
- Se logra con el uso de métodos virtual y override.

```
class Animal {
    public virtual void HacerSonido() {
        Console.WriteLine("El animal hace un sonido.");
    }
}

class Perro : Animal {
    public override void HacerSonido() {
        Console.WriteLine("El perro ladra: ¡Guau guau!");
    }
}

class Gato : Animal {
    public override void HacerSonido() {
        Console.WriteLine("El gato maulla: ¡Miau miau!");
    }
}

class Program {
    static void Main() {
        Animal miAnimal1 = new Perro();
        Animal miAnimal2 = new Gato();

        miAnimal1.HacerSonido(); // Ejecuta el método de Perro
        miAnimal2.HacerSonido(); // Ejecuta el método de Gato
    }
}
```

Abstracción

- **Permite definir clases base con métodos que serán implementados en clases derivadas**.
- Se usa la palabra clave **abstract** para crear una clase base con métodos que deben implementarse en las clases hijas.

```
// Clase abstracta (no se puede instanciar directamente)
abstract class Figura {
    public abstract void Dibujar(); // Método abstracto
}

class Circulo : Figura {
    public override void Dibujar() {
        Console.WriteLine("Dibujando un círculo...");
    }
}

class Cuadrado : Figura {
    public override void Dibujar() {
        Console.WriteLine("Dibujando un cuadrado...");
    }
}

class Program {
    static void Main() {
        Figura miFigura1 = new Circulo();
        Figura miFigura2 = new Cuadrado();

        miFigura1.Dibujar();  // Llama a la implementación en Circulo
        miFigura2.Dibujar();  // Llama a la implementación en Cuadrado
    }
}
```

Resumen

Concepto	Descripción
Encapsulamiento	Oculta datos y permite el acceso controlado mediante propiedades.
Herencia	Permite que una clase hija reutilice código de una clase base.
Polimorfismo	Permite que un mismo método tenga diferentes comportamientos según el objeto.
Abstracción	Permite definir clases base con métodos que deben ser implementados en clases hijas.

Ejercicio práctico

Crea un programa en C# que haga lo siguiente:
1.Pida al usuario su nombre y su edad.
2.Verifique si la persona es mayor de edad o menor de edad.
3.Solicite al usuario su calificación en una materia y determine si aprobó (nota mayor o igual a 60) o reprobó.
4.Imprima un mensaje final con toda la información recopilada.

5.Ejemplo de salida:
Ingrese su nombre: Juan
6.Ingrese su edad: 20, Eres mayor de edad.
7.Ingrese su calificación: 75 Felicidades, has aprobado
8..Gracias por usar nuestro programa, Juan.