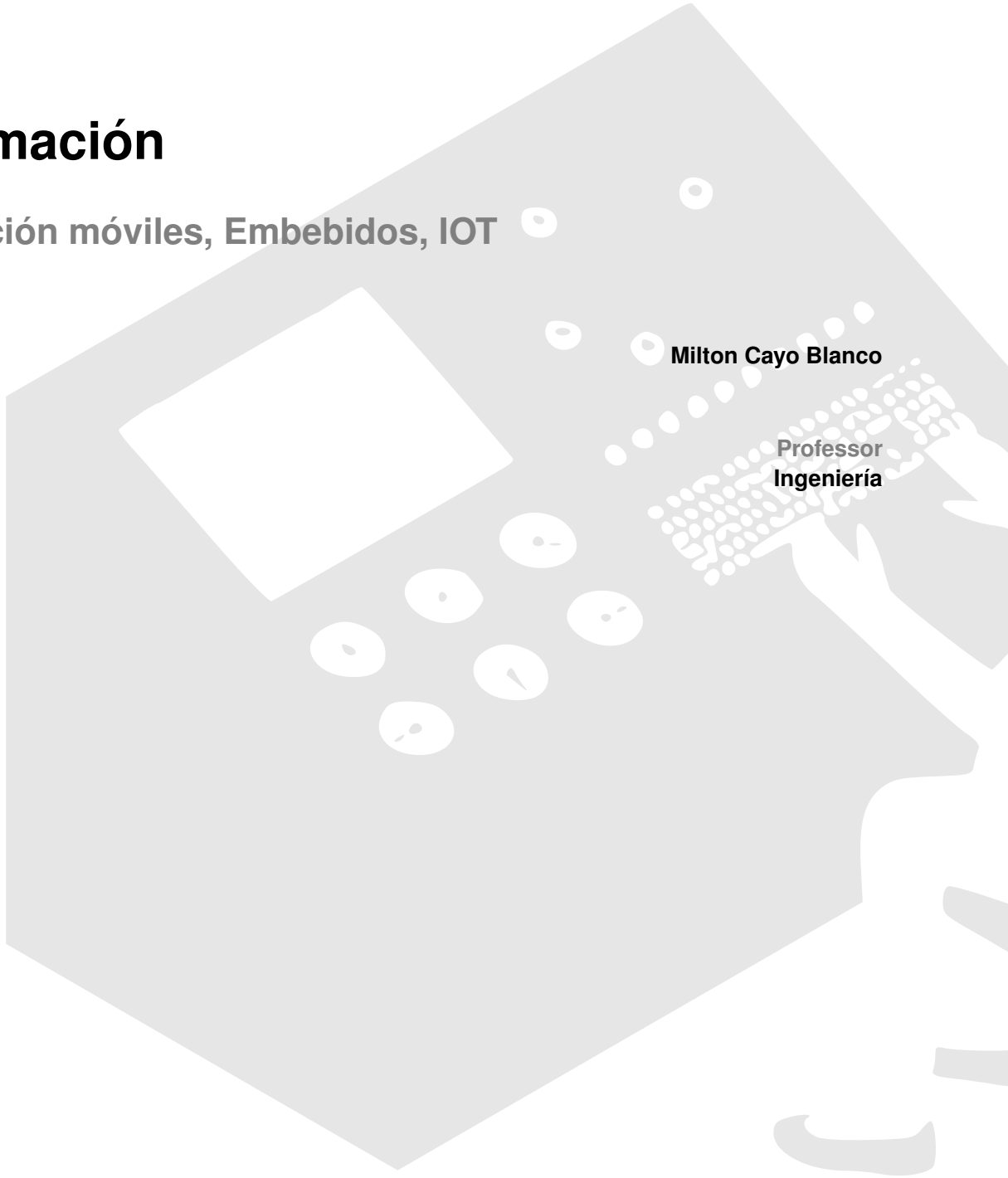
# Programación

## Programación móviles, Embebidos, IOT

**Milton Cayo Blanco**

Professor
**Ingeniería**



February 17, 2025

CHAPTER 1

# Fundamentos básicos de DART

**Note:** Dart es un lenguaje de programación gratuito, de código abierto y orientado a objetos. Fue desarrollado por Google en 2011.

A diferencia de muchos lenguajes, Dart se diseñó con el objetivo de hacer el proceso de desarrollo lo más cómodo y rápido posible para los desarrolladores. Por eso, viene con un conjunto bastante extenso de herramientas integrado, como su propio gestor de paquetes, varios compiladores/transpiladores, un analizador y formateador. Además, la máquina virtual de Dart y la compilación Just-in-Time hacen que los cambios realizados en el código se puedan ejecutar inmediatamente. Una vez en producción, el código se puede compilar en lenguaje nativo, por lo que no es necesario un entorno especial para ejecutar. En caso de que se haga desarrollo web, Dart se transpila a JavaScript.

## 1.1 INSTALACIÓN

### 1.1.1 Windows

1. Descarga el instalador oficial desde: `https://dart.dev/get-dart`.
2. Ejecuta el instalador y sigue las instrucciones.
3. Durante la instalación, marca la opción **"Add Dart to PATH"**.
4. Verifica la instalación abriendo la terminal y ejecutando:

```
dart --version
```

### 1.1.2 macOS

1. Abre la terminal y ejecuta:

```
brew install dart
```

2. Verifica la instalación con:

```
dart --version
```

### 1.1.3 Linux (Ubuntu/Debian)

1. Agrega el repositorio de Dart:

```
sudo apt update
sudo apt install apt-transport-https
sudo sh -c 'wget -qO- https://dl-ssl.google.com/linux/linux_signing_key.pub
| apt-key add -'
sudo sh -c 'wget -qO- https://storage.googleapis.com/download.dartlang.org
/linux/debian/dart_stable.list > /etc/apt/sources.list.d/dart_stable.list'
sudo apt update
sudo apt install dart
```

2. Agrega Dart al `PATH` editando `~/.bashrc` o `~/.zshrc`:

```
export PATH="$PATH:/usr/lib/dart/bin"
```

3. Verifica la instalación:

```
dart --version
```

## 1.2 INSTALACIÓN DE VISUAL STUDIO CODE Y CONFIGURACIÓN DE DART

1. Descarga e instala Visual Studio Code desde: `https://code.visualstudio.com/`.
2. Abre VS Code y ve al panel de extensiones (`Ctrl + Shift + X`).
3. Busca **Dart** e instala la extensión oficial.
4. Abre una terminal en VS Code y verifica que Dart está instalado con:

```
dart --version
```

5. Ahora puedes comenzar a programar en Dart dentro de Visual Studio Code.

## 1.3 TIPOS DE DATOS

Dart es un lenguaje de programación fuertemente tipado que ofrece diversos tipos de datos. A continuación, se presentan los principales tipos de datos en Dart junto con ejemplos de código.

### 1.3.1 Enteros (`int`)

El tipo `int` representa números enteros.

```
void main() {
  int edad = 25;
  print(edad); // 25
}
```

### 1.3.2 Números de punto flotante (`double`)

El tipo `double` representa números con decimales.

```
void main() {
  double temperatura = 36.5;
  print(temperatura); // 36.5
}
```

### 1.3.3 Booleanos (`bool`)

El tipo `bool` almacena valores `true` o `false`.

```
void main() {
  bool esMayorDeEdad = true;
  print(esMayorDeEdad); // true
}
```

### 1.3.4 Cadenas de texto (`String`)

El tipo `String` se usa para representar texto.

```
void main() {
  String nombre = "Carlos";
  print(nombre); // Carlos
}
```

### 1.3.5 Listas (`List`)

Las listas en Dart son estructuras ordenadas de datos.

```
void main() {
  List<int> numeros = [1, 2, 3, 4, 5];
  print(numeros); // [1, 2, 3, 4, 5]
}
```

### 1.3.6 Conjuntos (`Set`)

Los conjuntos (`Set`) son colecciones sin elementos duplicados.

```
void main() {
  Set<String> frutas = {"manzana", "pera", "uva"};
  print(frutas); // {manzana, pera, uva}
}
```

### 1.3.7 Mapas (`Map`)

Los mapas son colecciones de pares clave-valor.

```
void main() {
  Map<String, int> edades = {"Juan": 30, "Ana": 25};
  print(edades["Juan"]); // 30
}
```

### 1.3.8 Valores Nulos (`null`)

Dart permite valores nulos con el operador `?`.

```
void main() {
  int? edad;
  print(edad); // null
}
```

### 1.3.9 Dinámicos (`dynamic`)

El tipo `dynamic` permite cambiar el tipo de valor en tiempo de ejecución.

```
void main() {
  dynamic variable = 10;
  variable = "Ahora es una cadena";
  print(variable); // Ahora es una cadena
}
```

Estos son los tipos de datos fundamentales en Dart.

## 1.4 VARIABLES Y CONSTANTES

Dart permite declarar variables y constantes de diferentes maneras. Aquí se muestran los principales tipos de declaración.

### 1.4.1 Declaración de Variables

Las variables se pueden declarar usando `var`, con inferencia de tipo, o especificando el tipo explícitamente.

```
void main() {
  var nombre = "Carlos"; // Inferencia de tipo: String
  int edad = 25;         // Tipo expl cito: int
  print("$nombre tiene $edad a os");
}
```

### 1.4.2 Constantes

Las constantes pueden ser declaradas utilizando las palabras clave `final` o `const`. La diferencia es que `final` se asigna en tiempo de ejecución y `const` en tiempo de compilación.

```
void main() {
  final pi = 3.1416; // Puede ser asignada en tiempo de ejecuci n
  const e = 2.718;   // Asignada en tiempo de compilaci n
  print("Valor de pi: $pi, e: $e");
}
```

## 1.5 OPERADORES EN DART

Los operadores permiten realizar operaciones matemáticas, lógicas, de asignación, etc. A continuación se detallan algunos de los operadores más utilizados.

### 1.5.1 Operadores Aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas básicas.

```
void main() {
  int a = 10, b = 5;
  print(a + b);   // Suma: 15
  print(a - b);   // Resta: 5
  print(a * b);   // Multiplicaci n: 50
  print(a / b);   // Divisi n: 2.0
  print(a % b);   // M dulo: 0
}
```

### 1.5.2 Operadores de Comparación

Estos operadores permiten comparar dos valores.

```
void main() {
  int a = 10, b = 5;
  print(a > b);   // Mayor que: true
  print(a < b);   // Menor que: false
  print(a == b); // Igual a: false
  print(a != b); // Diferente a: true
}
```

### 1.5.3 Operadores Lógicos

Los operadores lógicos se utilizan para combinar expresiones booleanas.

```
void main() {
  bool x = true, y = false;
  print(x && y); // AND l gico: false
  print(x || y); // OR l gico: true
  print(!x);     // NOT l gico: false
}
```

### 1.5.4 Operadores de Asignación

Los operadores de asignación permiten modificar el valor de una variable.

```
void main() {
  int a = 10;
  a += 5;  // a = a + 5
  print(a); // 15
  a -= 3;  // a = a - 3
  print(a); // 12
}
```

### 1.5.5 Operadores de Incremento y Decremento

Los operadores ++ y −− aumentan o disminuyen el valor de una variable en 1.

```
void main() {
  int a = 10;
  print(a++);  // Post-incremento: 10
  print(a);    // 11
  print(--a);  // Pre-decremento: 10
}
```

## 1.6 ESTRUCTURAS DE CONTROL

Las estructuras de control permiten controlar el flujo de ejecución de un programa.

### 1.6.1 `if` y `else`

El condicional `if` permite ejecutar un bloque de código si una condición es verdadera, mientras que `else` se ejecuta si la condición es falsa.

```
void main() {
  int edad = 18;
  if (edad >= 18) {
    print("Eres mayor de edad");
  } else {
    print("Eres menor de edad");
  }
}
```

### 1.6.2 `switch`

El `switch` permite elegir entre múltiples opciones basadas en el valor de una variable.

```
void main() {
  String color = "rojo";
  switch (color) {
    case "rojo":
      print("Es un color rojo");
      break;
    case "azul":
      print("Es un color azul");
      break;
    default:
      print("Color no reconocido");
  }
}
```

### 1.6.3 Bucles: `for`, `while`, `do-while`

Los bucles permiten ejecutar un bloque de código repetidamente mientras se cumpla una condición.

```
void main() {
  // Bucle for
  for (int i = 0; i < 5; i++) {
    print(i);
  }
```

```
  // Bucle while
  int j = 0;
  while (j < 5) {
    print(j);
    j++;
  }

  // Bucle do-while
  int k = 0;
  do {
    print(k);
    k++;
  } while (k < 5);
}
```

## 1.7 FUNCIONES EN DART

Las funciones permiten organizar el código en bloques reutilizables. Dart soporta tanto funciones nombradas como funciones anónimas.

### 1.7.1 Funciones Nombradas

Las funciones nombradas se definen utilizando la palabra clave `void` para funciones que no retornan un valor, o el tipo de dato correspondiente para funciones que retornan valores.

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

void main() {
  print(suma(3, 4)); // 7
}
```

### 1.7.2 Funciones Anónimas

Las funciones anónimas no tienen nombre y se suelen asignar a variables.

```
void main() {
  var saludo = (String nombre) {
    print("Hola, $nombre");
  };

  saludo("Carlos");
}
```

### 1.7.3 Funciones Flecha

Las funciones flecha (=>) son una forma más corta de declarar funciones con una sola expresión.

```
int suma(int a, int b) => a + b;

void main() {
  print(suma(3, 4)); // 7
}
```

## 1.8 PROGRAMACIÓN ORIENTADA A OBJETOS

Dart es un lenguaje orientado a objetos, lo que significa que se pueden crear clases y objetos. A continuación se muestra cómo trabajar con clases.

### 1.8.1 Clases y Objetos

Una clase se define con la palabra clave `class`. Los objetos son instancias de una clase.

```
class Persona {
  String nombre;
  int edad;

  // Constructor
  Persona(this.nombre, this.edad);

  void mostrar() {
    print("Nombre: $nombre, Edad: $edad");
  }
}

void main() {
  var p = Persona("Carlos", 30);
  p.mostrar();
}
```

### 1.8.2 Herencia

La herencia permite que una clase herede propiedades y métodos de otra clase.

```
class Animal {
  void comer() {
    print("Comiendo...");
  }
}

class Perro extends Animal {
  void ladrar() {
    print("Ladrando...");
  }
}

void main() {
  var perro = Perro();
  perro.comer();
```

```
  perro.ladrar();
}
```

## 1.9 MANEJO DE EXCEPCIONES

Las excepciones permiten manejar errores durante la ejecución de un programa.

### 1.9.1 Bloques `try-catch`

```
void main() {
  try {
    int resultado = 10 ~/ 0; // Error de divisi n por cero
  } catch (e) {
    print("Error: $e");
  }
}
```

### 1.9.2 Bloque `finally`

El bloque `finally` se ejecuta independientemente de si ocurrió una excepción o no.

```
void main() {
  try {
    int resultado = 10 ~/ 0; // Error de divisi n por cero
  } catch (e) {
    print("Error: $e");
  } finally {
    print("El bloque finally siempre se ejecuta");
  }
}
```

## 1.10 EJERCICIOS RESUELTOS

### 1.10.1 Declaración de Variables

**Pregunta 1:** ¿Cuál es la diferencia entre ‘var‘ y un tipo de dato explícito en Dart?

```
void main() {
  var nombre = "Carlos"; // Inferencia de tipo: String
  int edad = 25;         // Tipo explicito: int
  print("$nombre tiene $edad a os");
}
```

```
void main() {
  var ciudad = "La Paz"; // Inferencia de tipo: String
  double precio = 150.75; // Tipo explicito: double
  print("La ciudad es $ciudad y el precio es \$${precio}");
}
```

**Pregunta 2:** ¿Qué sucede si intentamos cambiar el valor de una variable declarada como ‘final‘ o ‘const‘?

```
void main() {
  final pi = 3.1416; // Puede ser asignada en tiempo de ejecucion
  const e = 2.718;   // Asignada en tiempo de compilacion
  print("Valor de pi: $pi, e: $e");
}

void main() {
  final maxUsuarios = 100; // Determinado en tiempo de ejecuci n
  const maxIntentos = 3;   // Determinado en tiempo de compilaci n
  print("N mero m ximo de usuarios: $maxUsuarios");
  print("N mero m ximo de intentos: $maxIntentos");
}
```

### 1.10.2 Operadores Aritméticos

**Pregunta 3:** ¿Qué resultado se obtiene al realizar las operaciones aritméticas en el siguiente código?

```
void main() {
  int a = 10, b = 5;
  print(a + b);  // Suma: 15
  print(a - b);  // Resta: 5
}

void main() {
  int x = 20, y = 4;
  print(x * y);  // Multiplicaci n: 80
  print(x / y);  // Divisi n: 5.0
}
```

### 1.10.3 Operadores de Comparación

**Pregunta 4:** ¿Cómo funcionan los operadores de comparación en Dart, y qué valor se obtiene en los siguientes ejemplos?

```
void main() {
  int a = 10, b = 5;
  print(a > b);  // Mayor que: true
  print(a == b); // Igual a: false
}

void main() {
  int x = 15, y = 20;
  print(x <= y); // Menor o igual que: true
  print(x != y); // Diferente a: true
}
```

### 1.10.4 `if` y `else`

**Pregunta 5:** ¿Cómo funciona una estructura condicional `if` y `else` en Dart? ¿Cuál es el resultado de ejecutar el siguiente código?

```
void main() {
  int edad = 18;
  if (edad >= 18) {
    print("Eres mayor de edad");
  } else {
    print("Eres menor de edad");
  }
}

void main() {
  int horas = 14;
  if (horas < 12) {
    print("Buenos d as");
  } else {
    print("Buenas tardes");
  }
}
```

### 1.10.5 `switch`

**Pregunta 6:** ¿Cómo se utiliza un `switch` en Dart, y qué valor imprime el siguiente código?

```
void main() {
  String color = "rojo";
  switch (color) {
    case "rojo":
      print("Es un color rojo");
      break;
    case "azul":
      print("Es un color azul");
      break;
    default:
      print("Color no reconocido");
  }
}

void main() {
  String dia = "lunes";
  switch (dia) {
    case "lunes":
      print("Inicio de semana");
      break;
    case "viernes":
      print("Casi fin de semana");
      break;
    default:
      print("Un d a cualquiera");
  }
}
```

### 1.10.6 Funciones Nombradas

**Pregunta 7:** ¿Qué es una función nombrada en Dart, y cómo se utiliza?

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

void main() {
  print(suma(3, 4)); // 7
}

int multiplicacion(int x, int y) {
  return x * y;
}

void main() {
  print(multiplicacion(5, 6)); // 30
}
```

### 1.10.7 Funciones Anónimas

**Pregunta 8:** ¿Cómo se define y utiliza una función anónima en Dart?

```
void main() {
  var saludo = (String nombre) {
    print("Hola, $nombre");
  };

  saludo("Carlos");
}

void main() {
  var calcularArea = (double radio) => 3.14 * radio * radio;

  print(calcularArea(5.0)); // 78.5
}
```

### 1.10.8 Clases y Objetos

**Pregunta 9:** ¿Qué es una clase y cómo se crean objetos en Dart? ¿Qué imprime el siguiente código?

```
class Persona {
  String nombre;
  int edad;

  // Constructor
  Persona(this.nombre, this.edad);

  void mostrar() {
    print("Nombre: $nombre, Edad: $edad");
  }
}
```

```
void main() {
  var p = Persona("Carlos", 30);
  p.mostrar();
}

class Vehiculo {
  String marca;
  int velocidad;

  // Constructor
  Vehiculo(this.marca, this.velocidad);

  void mostrar() {
    print("Marca: $marca, Velocidad: $velocidad km/h");
  }
}

void main() {
  var coche = Vehiculo("Toyota", 120);
  coche.mostrar();
}
```

### 1.10.9 Herencia

**Pregunta 10:** ¿Cómo se implementa la herencia en Dart? ¿Qué resultados se obtienen en el siguiente código?

```
class Animal {
  void comer() {
    print("Comiendo...");
  }
}

class Perro extends Animal {
  void ladrar() {
    print("Ladrando...");
  }
}

void main() {
  var perro = Perro();
  perro.comer();
  perro.ladrar();
}

class Persona {
  String nombre;

  Persona(this.nombre);
}
```

```
class Estudiante extends Persona {
  String materia;

  Estudiante(String nombre, this.materia) : super(nombre);

  void estudiar() {
    print("$nombre est  estudiando $materia");
  }
}

void main() {
  var estudiante = Estudiante("Ana", "Matem ticas");
  estudiante.estudiar();
}
```

## 1.11 PROBLEMAS PROPUESTOS

1. **Cálculo de Factorial con Funciones Recursivas:** Escribe una función recursiva en Dart que calcule el factorial de un número. Luego, escribe un programa principal que pida al usuario un número y muestre su factorial.

2. **Conversión de Temperatura:** Crea una función que convierta una temperatura de grados Celsius a Fahrenheit y otra que convierta de Fahrenheit a Celsius. Usa una clase con un constructor para almacenar los valores y realiza las conversiones.

3. **Simulación de un Banco (Clases y Herencia):** Define una clase `CuentaBancaria` con métodos para depositar, retirar y mostrar el balance. Luego, crea una subclase `CuentaDeAhorros` que tenga un interés anual. Implementa un programa que simule la interacción con estas cuentas.

4. **Calculadora Simple (Funciones y Bucles):** Implementa una calculadora en Dart que acepte dos números y una operación (suma, resta, multiplicación, división) de forma continua en un bucle hasta que el usuario decida salir.

5. **Contador de Vocales en una Cadena (Bucles y Funciones):** Escribe una función que cuente cuántas veces aparecen las vocales en una cadena de texto. La función debe ser capaz de contar las vocales en mayúsculas y minúsculas.

6. **Juego de Adivinanza de Números:** Crea un juego en el que la computadora elija un número aleatorio entre 1 y 100, y el usuario tenga que adivinarlo. Después de cada intento, el programa debe decir al usuario si el número es mayor o menor que el que adivinó, hasta que adivine correctamente.

7. **Clases y Herencia en un Sistema de Vehículos:** Crea una clase base `Vehiculo` con propiedades como `marca` y `modelo`. Luego, crea subclases como `Coche`, `Camion` y `Motocicleta`, cada una con propiedades y métodos específicos. Usa un método `mostrarDatos` para imprimir los datos de cada vehículo.

8. **Fibonacci con Bucles:** Escribe una función que calcule los primeros `n` números de la secuencia Fibonacci usando un bucle `for`. Luego, imprime estos números en la consola.

9. **Sistema de Inventario (Clases y Herencia):** Crea una clase `Producto` con propiedades como `nombre`, `precio`, y `cantidad`. Luego, crea una subclase `ProductoPerecedero` con una propiedad adicional `fechaDeCaducidad`. Implementa un sistema que gestione un inventario de productos.

10. **Calculadora de Descuento con Funciones y Condicionales:** Escribe una función que calcule el precio final de un producto después de aplicar un descuento. El descuento debe variar dependiendo de la cantidad comprada. Por ejemplo, si compras más de 10 unidades, el descuento es del 10%.

11. **Clases con Métodos Estáticos:** Crea una clase `Matematica` con métodos estáticos para realizar operaciones como suma, resta, multiplicación y división. Implementa un programa principal que permita al usuario hacer operaciones matemáticas sin tener que crear una instancia de la clase.

12. **Simulación de un Sistema de Estudiantes (Herencia y Polimorfismo):** Crea una clase base `Persona` con propiedades como `nombre` y `edad`. Luego, crea una subclase `Estudiante` con propiedades adicionales como `materia` y `notaFinal`. Usa el polimorfismo para imprimir la información de cada estudiante.

13. **Generador de Números Aleatorios (Funciones y Bucles):** Escribe una función que genere 10 números aleatorios entre 1 y 100, y los imprima en la consola. Luego, escribe una función que determine cuál es el número más alto y el más bajo de esos números.

14. **Ordenamiento de un Arreglo (Algoritmos y Bucles):** Escribe un programa que reciba un arreglo de números enteros y los ordene en orden ascendente utilizando el algoritmo de burbuja (Bubble Sort).

15. **Simulación de un Sistema de Registro de Usuarios (Clases, Herencia y Funciones):** Crea una clase base `Usuario` con propiedades como `nombre` y `correo`. Luego, crea una subclase `Administrador` que agregue propiedades adicionales como `roles` y `permisos`. Implementa un sistema que permita registrar usuarios, agregar administradores y asignarles permisos.

16. **Calculadora de Potencias (Funciones y Recursión):** Crea una función recursiva que calcule el valor de un número elevado a una potencia, sin utilizar el operador `^`.

17. **Averiguar si un Número es Primo (Bucles y Condicionales):** Escribe una función que determine si un número dado es primo. La función debe devolver `true` si el número es primo y `false` en caso contrario.

18. **Tablas de Multiplicar (Bucles):** Crea una función que genere las tablas de multiplicar de un número dado desde el 1 hasta el 10, utilizando un bucle `for`.

19. **Conversión de Texto a Mayúsculas y Minúsculas (Funciones):** Crea una función que reciba una cadena de texto y la convierta a mayúsculas, y otra que la convierta a minúsculas.

20. **Simulación de una Agenda de Contactos (Clases y Herencia):** Define una clase `Contacto` con propiedades como `nombre`, `número de teléfono` y `email`. Luego, crea una subclase `ContactoUrgente` con propiedades adicionales como `prioridad`. Implementa un programa que permita almacenar, modificar y listar los contactos.

21. **Verificación de un Palíndromo (Funciones y Bucles):** Crea una función que verifique si una cadena de texto es un palíndromo (se lee igual hacia adelante y hacia atrás).

22. **Generar una Matriz de Números Aleatorios (Matrices y Funciones):** Escribe una función que genere una matriz de tamaño `n x m` (de tamaño aleatorio entre 2 y 5) con números aleatorios entre 1 y 100, y luego imprima la matriz.

23. **Suma de los Elementos de un Arreglo (Bucles y Funciones):** Crea una función que reciba un arreglo de números y devuelva la suma de todos sus elementos.

24. **Buscar un Elemento en un Arreglo (Bucles y Funciones):** Escribe una función que busque un número en un arreglo y devuelva su índice si lo encuentra, o `-1` si no lo encuentra.

25. **Desplegar un Menú de Opciones (Bucles y Funciones):** Crea un programa con un menú de opciones que permita al usuario elegir entre varias opciones (sumar, restar, multiplicar, dividir). El menú debe continuar apareciendo hasta que el usuario seleccione la opción de salir.

26. **Verificación de una Contraseña Segura (Funciones y Condicionales):** Crea una función que verifique si una contraseña es segura, siguiendo ciertas reglas como: debe tener al menos 8 caracteres, contener al menos una letra mayúscula, una letra minúscula, un número y un carácter especial.

27. **Generar Números Fibonacci Recursivamente (Recursión):** Escribe una función recursiva que imprima los primeros `n` números de la secuencia Fibonacci.

28. **Comprobación de Triángulo Válido (Funciones y Condicionales):** Crea una función que reciba tres números y determine si pueden formar un triángulo. Un triángulo es válido si la suma de dos lados siempre es mayor que el tercer lado.

29. **Cálculo de la Raíz Cuadrada (Funciones):** Crea una función que calcule la raíz cuadrada de un número sin usar la función `sqrt()`. Usa el método de aproximación de Newton.

30. **Sistema de Notas (Clases y Métodos):** Crea una clase `Estudiante` con métodos para agregar notas y calcular la media de las notas. Luego, crea una instancia de la clase y agrega varias notas, mostrando la media final.

## 1.12 PROBLEMAS PROPUESTOS - NIVEL AVANZADO

**1. Serie Aritmética:** Dada una serie aritmética donde el primer término es 3 y la diferencia común es 5, ¿cuál es la suma de los primeros 20 términos de la serie?

$$S_n = \frac{n}{2} \times (a_1 + a_n)$$

**Solución en Dart:**

```
double sumaPA(int n, double a1, double r) {
  return n * (2 * a1 + (n - 1) * r) / 2;
}

void main() {
  print('Suma PA: ${sumaPA(10, 2, 3)}');
}
```

**2. Serie Geométrica:** Dada una serie geométrica con primer término 2 y razón 3, ¿cuál es la suma de los primeros 10 términos?

$$S_n = a_1 \times \frac{1 - r^n}{1 - r}$$

**3. Serie de Fibonacci:** Calcular el $n$-ésimo término de la serie Fibonacci, donde los dos primeros términos son 0 y 1.

$$F_0 = 0, \quad F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} \quad \text{para} \quad n \geq 2$$

**4. Suma de los primeros *n* números naturales:** Calcular la suma de los primeros *n* números naturales.

$$S = \frac{n(n+1)}{2}$$

**5. Suma de los primeros *n* números cuadrados:** Calcular la suma de los primeros *n* números cuadrados.

$$S = \frac{n(n+1)(2n+1)}{6}$$

**6. Suma de los primeros *n* números impares:** Calcular la suma de los primeros *n* números impares.

$$S = n^2$$

**7. Suma de los primeros *n* términos de una progresión aritmética:** Dada una progresión aritmética de la forma $3, 6, 9, 12, \ldots$, ¿cuál es la suma de los primeros 50 términos?

$$S_n = \frac{n}{2} \times (a_1 + a_n)$$

**8. Serie de los números factoriales:** Calcular la suma de los factoriales de los primeros *n* números:

$$S = 1! + 2! + 3! + \ldots + n!$$

**9. Serie de potencias de 2:** Calcular la suma de las primeras *n* potencias de 2:

$$S = 2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}$$

**10. Serie de números primos:** Calcular la suma de los primeros *n* números primos.


Figure 1.1: Tipos de datos

# CHAPTER 2

# FireBase