Módulo 2: Fundamentos para el Desarrollo de Aplicaciones Móviles

Lección 3: Fundamentos de Dart

Objetivos de la Lección

- Comprender los conceptos fundamentales de **Dart** como lenguaje de programación.
- Explicar la sintaxis básica de Dart y cómo se utilizan las variables, tipos de datos, funciones y estructuras de control.
- Conocer las características clave que hacen de Dart un lenguaje eficiente para el desarrollo de aplicaciones, especialmente en Flutter.
- Escribir programas simples en Dart aplicando los fundamentos aprendidos.

Introducción de la Lección

Dart es un lenguaje de programación moderno, rápido y optimizado desarrollado por **Google**. Es especialmente popular por ser el lenguaje base de **Flutter**, un framework utilizado para construir aplicaciones móviles, web y de escritorio. Dart está diseñado para crear aplicaciones de alto rendimiento, con compilación tanto a código nativo como a JavaScript para su ejecución en diversas plataformas.

Uno de los principales beneficios de Dart es su facilidad de uso, ya que combina la simplicidad de lenguajes como JavaScript con características avanzadas que permiten escribir código eficiente y escalable. En esta lección, exploraremos los fundamentos de Dart, su sintaxis y sus características más importantes.

Características Clave de Dart

- **Simplicidad**: Dart tiene una sintaxis simple, inspirada en lenguajes como C, lo que lo hace fácil de aprender para programadores con experiencia en otros lenguajes.
- **Orientado a Objetos**: Dart es un lenguaje completamente orientado a objetos. Todo en Dart es un objeto, desde los tipos básicos hasta las estructuras más complejas.
- **Soporte para Concurrencia**: Dart cuenta con una característica llamada **Isolates**, que permite ejecutar tareas en paralelo sin interferir entre sí.
- **Tipado Estático y Dinámico**: Dart soporta tipado estático y dinámico, lo que significa que puedes definir el tipo de datos o dejar que Dart lo infiera automáticamente.
- **Compilación AOT y JIT**: Dart se compila tanto en tiempo de ejecución (JIT) como antes de la ejecución (AOT), lo que lo hace adecuado para desarrollar aplicaciones móviles de alto rendimiento.

Sintaxis Básica de Dart

A continuación, explicaremos los elementos básicos de la sintaxis de Dart.

1. Programa Básico en Dart

Un programa básico en Dart comienza con la función `main()`, que actúa como punto de entrada.

```
void main() {
  print('Hello, Dart!');
}
```

Explicación:

- **`void main()`**: La función `main` es el punto de inicio de todo programa Dart. `void` indica que la función no devuelve ningún valor.
- **`print()`**: Es una función integrada que imprime un valor en la consola.

Variables y Tipos de Datos

En Dart, las variables pueden ser declaradas con tipos explícitos o inferidos por el lenguaje.

1. Declaración de Variables

Dart es un lenguaje con tipado estático, pero permite la inferencia de tipos cuando no se especifican.

```
void main() {
  int edad = 25;        // Tipo explícito
  var nombre = 'Alice'; // Tipo inferido como String
  double altura = 5.9;   // Tipo explícito
  bool esProgramador = true; // Booleano
}
```

Tipos de Datos Principales en Dart:

- **int**: Números enteros.
- **double**: Números decimales.
- **String**: Cadenas de texto.
- **bool**: Valores booleanos (`true` o `false`).
- **var**: Permite inferir el tipo de la variable automáticamente.

Estructuras de Control

Las estructuras de control en Dart permiten manejar el flujo del programa, como condicionales y bucles.

1. Condicionales

Dart utiliza `if`, `else if` y `else` para manejar decisiones en el programa.

```
void main() {
  int edad = 18;

  if (edad >= 18) {
    print('Eres mayor de edad');
  } else {
    print('Eres menor de edad');
  }
}
```

2. Ciclos (Loops o Bucles)

Dart admite varios tipos de ciclos, incluyendo `for`, `while`, y `do-while`.

Ciclo for:

```
void main() {
  for (int i = 0; i < 5; i++) {
    print('Número: $i');
  }
}
```

Ciclo while:

```
void main() {
  int contador = 0;

  while (contador < 3) {
    print('Contador: $contador');
    contador++;
  }
}
```

Funciones en Dart

Las funciones son bloques de código reutilizables que pueden recibir parámetros y devolver valores. En Dart, las funciones se definen utilizando la palabra clave `void` si no devuelven un valor, o con un tipo de retorno si devuelven uno.

1. Función Básica sin Retorno:

```
void saludar() {
  print('Hola, ¿cómo estás?');
}

void main() {
  saludar();
}
```

2. Función con Retorno:

```
int sumar(int a, int b) {
  return a + b;
}
void main() {
  int resultado = sumar(5, 3);
  print('La suma es: $resultado');
}
```

En el ejemplo anterior, la función `sumar` toma dos enteros como parámetros, los suma y devuelve el resultado.

Listas y Mapas en Dart

Dart tiene estructuras de datos útiles como **Listas** (arrays) y **Mapas** (dictionaries).

1. Listas

Una lista es una colección ordenada de elementos.

```
void main() {
  List<String> frutas = ['Manzana', 'Banana', 'Naranja'];
  print(frutas[0]);  // Imprime 'Manzana'

  frutas.add('Mango');  // Añade un nuevo elemento
  print(frutas);
}
```

2. Mapas

Los mapas son colecciones de pares clave-valor.

```
void main() {
  Map<String, int> edades = {
    'Alice': 25,
    'Bob': 30,
    'Charlie': 35
  };

  print(edades['Alice']);  // Imprime 25
}
```

Orientación a Objetos en Dart

Dart es un lenguaje completamente orientado a objetos, lo que significa que todo en Dart es un objeto, incluso los tipos primitivos.

1. Clases y Objetos

Las **clases** son plantillas para crear objetos. Un **objeto** es una instancia de una clase.

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

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

```
  // Método
  void saludar() {
    print('Hola, me llamo $nombre y tengo $edad años.');
  }
}

void main() {
  Persona persona = Persona('Carlos', 28);
  persona.saludar();  // Imprime: Hola, me llamo Carlos y tengo
28 años.
}
```

En el ejemplo anterior, hemos creado una clase `Persona` con un constructor y un método `saludar`.

Asincronía en Dart

Dart soporta programación asincrónica utilizando **futures** y **async/await**. Esto es útil para manejar operaciones que toman tiempo, como solicitudes a servidores o la lectura de archivos.

Ejemplo de uso de async y await:

```
Future<String> obtenerDatos() async {
  // Simula una solicitud de datos
  await Future.delayed(Duration(seconds: 2));
  return 'Datos recibidos';
}
```

```
void main() async {
  print('Solicitando datos...');
  String datos = await obtenerDatos();
  print(datos);  // Imprime: Datos recibidos
}
```

En este ejemplo, la función **`obtenerDatos`** es asincrónica y usa **`await`** para esperar la finalización de una tarea antes de continuar.

Resumen de la Lección

En esta lección, hemos explorado los fundamentos del lenguaje de programación **Dart**, aprendiendo sobre su sintaxis básica, tipos de datos, estructuras de control, funciones y características orientadas a objetos. Dart es un lenguaje poderoso y flexible, ideal para desarrollar aplicaciones móviles, web y de escritorio utilizando **Flutter**. También hemos aprendido a manejar asincronía, lo cual es fundamental en el desarrollo de aplicaciones modernas que requieren tiempos de respuesta rápidos.

Actividad de la Lección

Esta actividad te permitirá practicar los conceptos básicos de Dart, desde la definición de clases hasta la implementación de funciones y estructuras de control.

Instrucciones:

1. Crea un programa en Dart que defina una clase `Producto` con las propiedades `nombre`, `precio` y un método `mostrarDetalles` que imprima la información del producto.
2. Escribe una función que reciba dos enteros como parámetros y devuelva la multiplicación de esos valores.
3. Usa una estructura de control `for` para iterar a través de una lista de números y mostrar solo los números pares.