ICT4153

Mobile Application Development

Practical 05

Flutter – Introduction to Dart Programming

Objective

Students will be able to develop a simple cross-platform mobile application using Dart programming and Flutter framework, demonstrating the use of Dart's syntax, asynchronous programming, and UI-building capabilities.

What is Dart Programming?

Dart is a modern, open-source, **general-purpose programming language** developed by **Google**. It is designed for building **high-performance, cross-platform applications**. Dart is primarily known for its use in **Flutter**, a popular framework for building natively compiled mobile, web, and desktop applications from a single codebase.

Key Features of Dart

1. **Object-Oriented:**
    - Dart is an object-oriented language, meaning it uses classes and objects to organize code into reusable components.
2. **Strongly Typed:**
    - Dart uses static and dynamic typing, providing flexibility in how data types are handled while also catching errors early during development.
3. **Cross-Platform:**
    - Dart supports development for multiple platforms, including **mobile (iOS, Android)**, **web**, **desktop**, and **backend** systems.
4. **Just-in-Time (JIT) and Ahead-of-Time (AOT) Compilation:**
    - JIT Compilation: Speeds up development with hot reload during testing.
    - AOT Compilation: Optimizes performance for production-ready apps by compiling code into efficient machine code.

5. **Asynchronous Programming:**
    - Dart has robust support for **asynchronous operations** using async, await, and Future, making it ideal for handling tasks like network requests and file I/O.
6. **UI-Friendly:**
    - Dart was designed with a focus on **user interfaces**, making it a great choice for frameworks like **Flutter**, where UI-building is declarative and efficient.

Why Use Dart?

- **Optimized for UI:** Dart allows developers to write code that is easy to read and maintain while focusing on creating smooth, responsive user interfaces.
- **Cross-Platform Development:** One codebase can be used to build applications for multiple platforms, saving time and resources.
- **Fast Development:** Dart's hot reload feature in Flutter accelerates development by enabling instant updates to the user interface without restarting the application.
- **High Performance:** Dart's AOT compilation ensures fast startup times and optimized runtime performance.
- **Versatility:** It supports building mobile, web, desktop, and backend applications.

Applications of Dart

1. **Mobile App Development:**
    - Primarily used with **Flutter** for creating cross-platform mobile applications.
2. **Web Development:**
    - Dart can compile to JavaScript, enabling it to run in browsers.
3. **Desktop App Development:**
    - Applications for Windows, macOS, and Linux can also be built using Dart and Flutter.
4. **Backend Development:**
    - Dart can be used to build server-side applications using frameworks like **Aqueduct** or **Shelf**.

Simple Hello World from Dart

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

How to run a Dart Program

1. **Using DartPad (Online)**

DartPad is an online tool for writing, running, and experimenting with Dart code.

Steps:

1. Go to DartPad.
2. Write or paste your Dart code into the editor.
3. Click the **Run** button at the top to execute the code.
4. The output will appear in the console below the code editor.

2. **Using Command Line (Locally)**

Prerequisites:

1. Install the Dart SDK from Dart's official site.
2. Ensure Dart is added to your system's PATH.

Steps:

1. Open a terminal or command prompt.
2. Create a Dart file, e.g., example.dart:

```
touch example.dart
```

3. Write your Dart code in the file using any text editor:

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

4. Run the Dart file with the following command:

```
dart example.dart
```

5. The output will appear in the terminal.

3. **Using an IDE (e.g., Visual Studio Code or IntelliJ IDEA)**

Visual Studio Code:

1. **Install Dart SDK**: Follow the instructions on Dart's website.
2. **Install Dart Extension**: In VS Code, go to Extensions Marketplace and install the **Dart** plugin.
3. **Write Your Dart Code**:
    - Create a new file with the .dart extension, e.g., example.dart.

- Write your Dart code:

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

4. **Run the Program**:
   - Open the terminal in VS Code (Ctrl + `).
   - Run the program using:

```
dart example.dart
```

IntelliJ IDEA or Android Studio:

1. Install the Dart plugin in IntelliJ IDEA or Android Studio.
2. Create a Dart project or open a .dart file.
3. Write your code and click the **Run** button to execute.

4. **Using Flutter for Dart Apps**

If you're working with Flutter:

1. Install Flutter from Flutter's website.
2. Write your Dart code in the main.dart file.
3. Use the following command to run the Flutter app:

```
flutter run
```

Example Code:

Here's a simple Dart program:

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

Expected Output:

```
Hello, Dart!
```

Variables and Data Types

Code Examples:

1. **Declaring a Variable:**

```
void main() {
   var name = 'Dart';
   print(name);
}
```

2. **Using Constants:**

```
void main() {
   final a = 12;
   const pi = 3.14;
   print(a);
   print(pi);
}
```

3. **Numbers, Strings, Booleans, Lists, and Maps:**

```
void main() {
   // Numbers
   int number = 10;
   double decimal = 10.5;
   print('Integer: $number, Double: $decimal');

   // String
   String greeting = "Hello, Dart!";
   print(greeting);

   // Boolean
   bool isDartFun = true;
   print('Is Dart fun? $isDartFun');

   // List
   var list = [1, 2, 3, 4, 5];
   print('List: $list');

   // Map
   var mapping = {'id': 1, 'name': 'Dart'};
   print('Map: $mapping');
}
```

4. **Dynamic Variables:**

```
void main() {

   dynamic name = "Dart";
```

```
    print(name);
}
```

Decision Making and Loops

Code Examples:

1. **Decision Making:**

```
void main() {
   int number = 10;
   if (number > 5) {
      print('Number is greater than 5');
   } else {
      print('Number is less than or equal to 5');
   }
}
```

2. **Loops:**

```
void main() {
   for (var i = 1; i <= 10; i++) {
      if (i % 2 == 0) {
         print(i); // Prints even numbers from 1 to 10
      }
   }
}
```

Functions

Code Example:

```
void main() {
   add(3, 4);
}

void add(int a, int b) {
   int c;
   c = a + b;
   print(c); // Output: 7
}
```

Object-Oriented Programming (OOP)

Code Example:

```
class Employee {
   String name;

   // Getter method
   String get emp_name {
      return name;
   }

   // Setter method
   void set emp_name(String name) {
      this.name = name;
   }

   // Function definition
   void result() {
      print(name);
   }
}

void main() {
   // Object creation
   Employee emp = new Employee();
   emp.name = "employee1";
   emp.result(); // Output: employee1
}
```

For more practice, please refer to the https://dart.dev/

1. Write a program to declare variables of different data types (numbers, strings, booleans, lists, and maps) and print their values.

2. Write a program to demonstrate the use of `final` and `const` in Dart. Try assigning a new value to a `final` variable and observe the result.
   Hint: Declare a `final` variable for today's date and a `const` value for the value of pi.

3. Write a Dart program that:

   - Prints all numbers from 1 to 50.
   - Only displays numbers divisible by 5.

4. Create a Dart program that has a function to calculate the area of a rectangle. Pass the length and width as parameters and return the area.

5. Create a `Student` class with the following fields:

   - Name
   - Age
   - Marks

Add methods to:

   - Set the student's details.
   - Print the student's details.

6. Write a program that prints whether a number from 1 to 20 is even or odd using a loop.

7. Create a list of 10 numbers. Write a program to:

   - Print the entire list.
   - Print only the numbers greater than 5.

8. Create a map that stores a product's details with the following keys:

   - id
   - name
   - price

9. Write a program to display the map's content and add a new key `category` to it.

10. Create a variable using `dynamic` and:

    - Assign a string to it.
    - Change the value to an integer and Change the value to a list.

11. Write a program (function) that takes a list and returns a new list that contains all the elements of the first list minus all the duplicates.

12. Write a program that asks the user how many Fibonnaci numbers to generate and then generates them. Take this opportunity to think about how you can use functions.

13. Write a program (using functions!) that asks the user for a long string containing multiple words. Print back to the user the same string, except with the words in backwards order.
    For example, say I type the string:

    My name is Michele.

    Then I would see the string:

    Michele is name My

14. Write a password generator in Dart. Be creative with how you generate passwords - strong passwords have a mix of lowercase letters, uppercase letters, numbers, and symbols. The passwords should be random, generating a new password every time the user asks for a new password. Include your run-time code in a main method.

15. Suppose, your distance to office from home is 25 km and you travel 40 km per hour. Write a program to calculate time taken to reach office in minutes.

    Formula= (distance) / (speed)