Module 2: Dart Programming Essentials

**Theory Assignments:**

**Q.** 1

**Q.** 1. Explain the fundamental data types in Dart (int, double, String, List, Map, etc.) and their uses.

**A. 1. int (Integer)**

* **Represents whole numbers (no decimal point).**
* **Example:**

**dart**

**CopyEdit**

**int age = 25;**

**int year = 2025;**

* **Use Case: Counting items, age, years, index values in loops.**

**2. double**

* **Represents decimal numbers (floating-point numbers).**
* **Example:**

**dart**

**CopyEdit**

**double price = 99.99;**

**double pi = 3.14159;**

* **Use Case: When precision is needed — money, measurements, scientific data.**

**3. String**

* **Represents a sequence of characters (text).**
* **Example:**

**dart**

**CopyEdit**

**String name = "Prakash";**

**String greeting = 'Hello, Dart!';**

* **Use Case: Names, messages, input/output text, descriptions.**

**4. bool**

* **Represents a Boolean value: true or false.**
* **Example:**

**dart**

**CopyEdit**

**bool isLoggedIn = true;**

**bool hasData = false;**

* **Use Case: Conditional checks, flags, decision-making (like if statements).**

**5. List (Array)**

* **Represents an ordered collection of items.**
* **Can contain any data type.**
* **Example:**

**dart**

**CopyEdit**

**List<int> numbers = [1, 2, 3, 4];**

**List<String> fruits = ['apple', 'banana', 'cherry'];**

* **Use Case: Storing multiple values in a single variable (like products in a cart).**

**6. Map**

* **Represents key-value pairs (like dictionaries).**
* **Example:**

**dart**

**CopyEdit**

**Map<String, String> countryCapital = {**

**'India': 'New Delhi',**

**'USA': 'Washington D.C.'**

**};**

* **Use Case: When data needs to be looked up using a key — e.g., configs, user info.**

**7. var and dynamic (Flexible Types)**

* **var: Automatically infers the type based on the assigned value.**

**dart**

**CopyEdit**

**var city = 'Mumbai'; // inferred as String**

* **dynamic: Can change type at runtime.**

**dart**

**CopyEdit**

**dynamic data = 10;**

**data = 'Now a string';**

* **Use Case:**
  + **var: Best when type doesn’t change.**
  + **dynamic: Use only when absolutely necessary (less type-safe).**

**8. const and final (Constants)**

**Not data types themselves, but modifiers that make variables immutable.**

* **const: Compile-time constant.**

**dart**

**CopyEdit**

**const pi = 3.14;**

* **final: Runtime constant (can be set once).**

**dart**

**CopyEdit**

**final name = 'Dart';**

**In Short**

| **Type** | **Example Value** | **Use Case** |
| --- | --- | --- |
| **int** | **42** | **Whole numbers** |
| **double** | **3.14** | **Decimal/precision numbers** |
| **String** | **"hello"** | **Text data** |
| **bool** | **true/false** | **Conditional logic** |
| **List** | **[1, 2, 3]** | **Ordered collections** |
| **Map** | **{'key': 'value'}** | **Key-value pair storage** |
| **var** | **var x = 10;** | **Type-inferred variables** |
| **dynamic** | **dynamic y = 10;** | **Type-changing values (less safe)** |
|  |  |  |

**Q.** 2

**Q.** 2. Describe control structures in Dart with examples of if, else, for, while, and switch.

**A. 1. if and else**

**Used to make decisions based on conditions.**

**🔹 Syntax:**

**dart**

**CopyEdit**

**if (condition) {**

**// Code if condition is true**

**} else {**

**// Code if condition is false**

**}**

**🔹 Example:**

**dart**

**CopyEdit**

**int age = 20;**

**if (age >= 18) {**

**print("You are an adult.");**

**} else {**

**print("You are a minor.");**

**}**

**2. for Loop**

**Used to repeat a block of code a fixed number of times.**

**🔹 Syntax:**

**dart**

**CopyEdit**

**for (initialization; condition; increment/decrement) {**

**// Loop body**

**}**

**🔹 Example:**

**dart**

**CopyEdit**

**for (int i = 1; i <= 5; i++) {**

**print("Number $i");**

**}**

**3. while Loop**

**Executes a block of code while a condition is true.**

**🔹 Syntax:**

**dart**

**CopyEdit**

**while (condition) {**

**// Loop body**

**}**

**🔹 Example:**

**dart**

**CopyEdit**

**int i = 1;**

**while (i <= 5) {**

**print("Count $i");**

**i++;**

**}**

**4. do-while Loop**

**Like while, but executes at least once before checking the condition.**

**🔹 Syntax:**

**dart**

**CopyEdit**

**do {**

**// Loop body**

**} while (condition);**

**🔹 Example:**

**dart**

**CopyEdit**

**int j = 1;**

**do {**

**print("Running $j");**

**j++;**

**} while (j <= 3);**

**5. switch Statement**

**Used for multiple condition checks (more elegant than multiple if-else).**

**🔹 Syntax:**

**dart**

**CopyEdit**

**switch (expression) {**

**case value1:**

**// Code**

**break;**

**case value2:**

**// Code**

**break;**

**default:**

**// Default code**

**}**

**🔹 Example:**

**dart**

**CopyEdit**

**String day = "Monday";**

**switch (day) {**

**case "Monday":**

**print("Start of the week.");**

**break;**

**case "Friday":**

**print("Weekend is near!");**

**break;**

**default:**

**print("It's just another day.");**

**}**

**In Short**

| **Structure** | **Purpose** | **Example Keyword** |
| --- | --- | --- |
| **if/else** | **Decision making** | **if, else** |
| **for** | **Fixed number of repetitions** | **for** |
| **while** | **Repeats while condition is true** | **while** |
| **do-while** | **Executes once, then checks condition** | **do, while** |
| **switch** | **Multi-condition branching** | **switch, case** |

**Q.** 3

**Q.** 3. Explain object-oriented programming concepts in Dart, such as classes, inheritance,

polymorphism, and interfaces.

## A. 1. Classes and Objects

## Theory:

## A class is a blueprint or template for creating objects.

## It defines properties (variables) and behaviors (methods/functions).

## An object is an instance of a class — it represents a specific real-world entity created from the class blueprint.

## Key Points:

## You define a class once and create many objects from it.

## Objects can have different values for their properties but share the same structure.

## 2. Inheritance

## Theory:

## Inheritance allows a class (called a child or subclass) to acquire the properties and methods of another class (called a parent or superclass).

## Dart uses the keyword extends to indicate inheritance.

## Key Points:

## Promotes code reuse: you don’t have to rewrite common functionality.

## The child class can also have its own additional properties or override methods from the parent.

## 3. Polymorphism

## Theory:

## Polymorphism means "many forms".

## It allows different classes to define methods that have the same name but behave differently.

## In Dart, this is mainly achieved through method overriding, where a child class redefines a method inherited from a parent class.

## Key Points:

## Helps write flexible and reusable code.

## Enables treating objects of different classes in a uniform way if they share the same interface or base class.

## 4. Interfaces

## Theory:

## An interface defines a contract that a class must follow.

## Dart doesn’t have a separate keyword for interfaces; instead, any class can act as an interface.

## A class can implement another class as an interface using the implements keyword.

## When you implement a class, you must override all its methods.

## Key Points:

## Interfaces define what a class must do, not how it does it.

## Supports multiple interfaces, unlike inheritance which only supports single inheritance.

## In Short

| OOP Concept | Description |
| --- | --- |
| Class | Blueprint for creating objects (defines state and behavior) |
| Object | An instance of a class |
| Inheritance | Allows reuse of code from a parent class using extends |
| Polymorphism | Enables different behaviors using the same method name (@override) |
| Interface | A contract that forces a class to implement certain methods (implements) |
|  | Q.4 |

**Q.4** Describe asynchronous programming in Dart, including Future, async, await, and Stream.

**A. Asynchronous Programming in Dart – Theory**

**In Dart, asynchronous programming is used to perform non-blocking operations, such as fetching data from the internet, reading files, or waiting for user input, without freezing the main thread (usually the UI thread in Flutter apps).**

**1. Future**

**Definition:**

**A Future represents a computation or task that completes in the future, either successfully with a value or with an error.**

* **It is used when you expect a single value that will be available later.**

**Key Characteristics:**

* **Asynchronous**
* **Returns a value or error after a delay**
* **Can be in one of three states: uncompleted, completed with data, or completed with error**

**Syntax:**

**dart**

**CopyEdit**

**Future<String> getData() {**

**return Future.delayed(Duration(seconds: 2), () => 'Hello, Future!');**

**}**

**2. async Keyword**

**Definition:**

**The async keyword is used to mark a function as asynchronous. It allows the use of await inside the function and automatically wraps the return value in a Future.**

**Key Characteristics:**

* **Makes a function return a Future**
* **Allows cleaner, readable syntax for asynchronous code**

**Syntax:**

**dart**

**CopyEdit**

**Future<void> fetchData() async {**

**// asynchronous function**

**}**

**3. await Keyword**

**Definition:**

**The await keyword is used to pause the execution of an async function until the awaited Future is complete.**

* **It does not block the entire program, only the function execution.**

**Key Characteristics:**

* **Used only inside async functions**
* **Awaits the result of a Future**
* **Simplifies callback-style code**

**Syntax:**

**dart**

**CopyEdit**

**Future<void> fetchData() async {**

**String result = await getData(); // waits here until getData completes**

**print(result);**

**}**

**4. Stream**

**Definition:**

**A Stream represents a sequence of asynchronous events or data over time.**

* **Unlike Future, which delivers a single result, a Stream can provide multiple values.**
* **Commonly used for real-time data, such as:**
  + **User input (keyboard/mouse events)**
  + **WebSocket connections**
  + **Sensor or location data**
  + **Periodic updates**

**Key Characteristics:**

* **Emits multiple values over time**
* **Can be listened to**
* **Can be transformed, filtered, paused, resumed, and canceled**

**Syntax:**

**dart**

**CopyEdit**

**Stream<int> numberStream() async\* {**

**for (int i = 1; i <= 3; i++) {**

**await Future.delayed(Duration(seconds: 1));**

**yield i;**

**}**

**}**

**dart**

**CopyEdit**

**void main() async {**

**await for (int num in numberStream()) {**

**print("Received: $num");**

**}**

**}**

**Comparison Table**

| **Feature** | **Future** | **Stream** |
| --- | --- | --- |
| **Returns** | **Single value or error** | **Multiple values over time** |
| **Use Case** | **File read, API request** | **Sensor data, live updates, events** |
| **Listens** | **One-time then() or await** | **listen() or await for** |
| **Control** | **Simple to manage** | **Supports pause/resume/cancel** |

**Summary of Keywords**

| **Keyword** | **Meaning** |
| --- | --- |
| **Future** | **Represents a single asynchronous result** |
| **async** | **Marks a function that contains asynchronous code** |
| **await** | **Pauses code execution until the Future completes** |
| **Stream** | **Represents a series of asynchronous results over time** |

**Real-World Example:**

* **When you click a button in a Flutter app to load user profile from a server:**
  + **Use Future to fetch the profile data.**
  + **Use async/await to wait for the data without blocking the UI.**
  + **Use Stream if you want live updates to the profile (e.g., when the user edits it from another device).**