- By Aakash

Assignment

MODULE: 3

Fundamental - Dart Programming

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

Ans. In Dart, a statically-typed language developed by Google (often used with Flutter), fundamental data types are the building blocks for handling data in your applications. Here's an overview of the most commonly used core data types:

---

1. `int`

Description: Represents integer values (whole numbers without decimals).

Example: `int age = 30;`

Use: Ideal when you need to work with counters, indexes, or any whole-number values (e.g., age, score, quantity).

-

---

2. `double`

Description: Represents floating-point numbers (numbers with decimals).

Example: `double price = 99.99;`

Use: Use this for representing precise numerical values such as measurements, percentages, or currency.

---

3. `String`

Description: A sequence of characters enclosed in single (`'`) or double (`"`) quotes.

Example: `String name = "Alice";`

Use: Used to store and manipulate textual data like names, messages, or file paths.

-

---

4. `bool`

Description: Represents boolean values — either `true` or `false`.

Example: `bool isLoggedIn = false;`

Use: Useful in control flow, logic checks, and conditionals (if/else, while loops, etc.).

5. `List<E>`

Description: An ordered collection of items of type `E`. Equivalent to an array in many other languages.

Example:

```
List<String> fruits = ['apple', 'banana', 'mango'];
List<int> numbers = [1, 2, 3];
```

Use: Use when you want to manage a sequence of objects — for example, a list of names, products, or tasks.

---

6. `Map<K, V>`

Description: A collection of key-value pairs where each key is unique.

Example:

```
Map<String, int> scores = {'Alice': 90, 'Bob': 85}
```

Use: Use maps when you need to look up values by keys, such as configuration settings or JSON-like structures.

---

7. `Set<E>`

Description: An unordered collection of unique items of type

```
Set<int> uniqueNumbers = {1, 2, 3};
```

Use: Useful for ensuring there are no duplicate items, such as tracking selected items or distinct tags.

---

8. `dynamic`

Description: A special type that disables static type checking for the variable. The variable can hold any type of data.

Example

```
dynamic x = 10;
x = "now a string";
```

Use: When flexibility is needed, but it sacrifices type safety. Use sparingly and cautiously.

---

9. `var`

Description: Tells Dart to infer the type based on the value assigned. The type is fixed once assigned.

Example:

```
var name = 'Charlie';  // inferred as String
```

Use: Convenient when the type is obvious and reduces verbosity.

- By Aakash

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

Ans. Control structures in Dart are used to control the flow of execution in a program. These include conditional statements and loops. Below are the main types:

---

1. `if` and `else` Statements

Used to execute code based on conditions.

Syntax

```
if (condition) {
  // code if condition is true
} else {
  // code if condition is false
}
```

Example:

```
int age = 18;
if (age >= 18) {
  print('Eligible to vote');
} else {
  print('Not eligible');
}
```

---

2. `for` Loop

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

Syntax:

```
for (initialization; condition; increment) {
  // repeated code
}
```

Example:

```
for (int i = 1; i <= 5; i++) {
  print('Number $i');
}
```

---

3. `while` Loop

Repeats code **as long as** the condition is true.

Syntax:

```
while (condition) {
  // repeated code
}
```

Example:

```
int i = 1;
while (i <= 3) {
  print('Count $i');
  i++;
}
```

---

4. `do...while` Loop

Similar to `while`, but it **runs at least once** before checking the condition.

Syntax:

```
do {
  // code
} while (condition);
```

Example:

```
int i = 1;
do {
  print('Hello $i');
  i++;
} while (i <= 2);
```

5. `switch` Statement

Used to execute one block of code among many options based on the value of a variable.

Syntax:

```
switch (variable) {
  case value1:
    // code
    break;
  case value2:
    // code
    break;
  default:
    // code
}
```

Example:

```
String day = 'Monday';
switch (day) {
  case 'Monday':
    print('Start of week');
    break;
  case 'Friday':
    print('End of week');
    break;
  default:
    print('Midweek day');
}
```

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

➢ **Ans.**

Object-Oriented Programming (OOP) Concepts in Dart

1. Classes

Blueprint for creating objects (instances).

Defines properties (variables) and methods (functions).

Example:

```
class Person {
  String name;
  int age;

  Person(this.name, this.age);

  void greet() {
    print('Hello, my name is $name.');
  }
}
```

---

2. Inheritance

Allows a class to inherit properties and methods from another class (base/superclass).

Promotes code reuse.

Example:

```
class Student extends Person {
  String course;

  Student(String name, int age, this.course) : super(name, age);

  void study() {
    print('$name is studying $course.');
  }
}
```

---

3. Polymorphism

Ability to use a single interface to represent different underlying forms (methods can behave differently).

Achieved by method overriding.

Example:

```
dart
```

```
CopyEdit
class Animal {
  void sound() {
    print('Some sound');
  }
}

class Dog extends Animal {
  @override
  void sound() {
    print('Bark');
  }
}

void makeSound(Animal animal) {
  animal.sound();  // Calls overridden method depending on the object
type
}
```

---

4. Interfaces

In Dart, any class can act as an interface.

Classes implement interfaces to define specific behavior.

Provides a contract without implementation.

Example:

```
dart
CopyEdit
class Flyer {
  void fly() {
    print('Flying');
  }
}

class Bird implements Flyer {
  @override
  void fly() {
    print('Bird is flying');
  }
}
```

Summary:

Class: Defines objects.

Inheritance: Derives new classes from existing ones.

Polymorphism: Same method behaves differently in subclasses.

Interface: Defines methods a class must implement.

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

- **Ans.**

Asynchronous programming allows your program to perform tasks (like fetching data or reading files) without blocking the main execution flow.

---

1. Future

Represents a value that will be available **later**, like a promise.

Used for single asynchronous operations.

```
Future<String> fetchData() {
  return Future.delayed(Duration(seconds: 2), () => 'Data loaded');
}
```

---

2. async

Marks a function as asynchronous, allowing you to use `await` inside it.

```
Future<void> loadData() async {
  String data = await fetchData();
  print(data);
}
```

---

3. await

Pauses execution inside an async function until the Future completes and returns the result.

---

4. Stream

Represents a sequence of asynchronous events or data over time.

Useful for handling multiple values, like user inputs or real-time data.

```
Stream<int> countStream = Stream.periodic(Duration(seconds: 1), (x)
=> x).take(5);
```

Summary

Future: Single async result.

async/await: Syntax to write async code clearly.

Stream: Multiple async events over time