MODULE 2

Q1. Explain the fundamental data types in Dart.

Answer:
Dart provides several built-in data types to store different kinds of data.

- **int**: Stores whole numbers.
  Example: int age = 20;
- **double**: Stores decimal numbers.
  Example: double price = 99.5;
- **String**: Stores text or characters.
  Example: String name = "Dart";
- **bool**: Stores true or false values.
  Example: bool isActive = true;
- **List**: Stores ordered collection of values.
  Example: List<int> nums = [1, 2, 3];
- **Map**: Stores key–value pairs.
  Example: Map<String, int> marks = {"Math": 90};
- **Set**: Stores unique values only.
  Example: Set<int> ids = {1, 2, 3};

Q2. Describe control structures in Dart with examples.

Answer:
Control structures are used to control the flow of execution in a program.

if–else

```
if (age >= 18) {

  print("Adult");

} else {

  print("Minor");

}
```

for loop

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

  print(i);

}
```

while loop

```
int i = 1;

while (i <= 5) {

  print(i);

  i++;

}
```

switch

```
int day = 1;

switch (day) {

  case 1:

    print("Monday");

    break;

  default:

    print("Invalid");

}
```

Q3. Explain object-oriented programming concepts in Dart.

Answer:
Dart follows Object-Oriented Programming (OOP) concepts.

Class and Object

A class is a blueprint; an object is an instance of a class.

```
class Student {

  String name;

  Student(this.name);

}
```

Inheritance

One class acquires properties of another.

```
class Child extends Parent {}
```

Polymorphism

Same method behaves differently in different classes.

```
class Animal {
  void sound() {}
}
class Dog extends Animal {
  void sound() {
    print("Bark");
  }
}
```

Interface

Dart uses classes as interfaces.

```
class Printable {
  void printData() {}
}
class Report implements Printable {
  void printData() {
    print("Report");
  }
}
```

Q4. Explain asynchronous programming in Dart.

Answer:
Asynchronous programming allows tasks to run without blocking the main program.

Future

Represents a value that will be available later.

```
Future<String> fetchData() async {
  return "Data Loaded";
}
```

async & await

Used to write asynchronous code in a readable way.

```
void main() async {
  String data = await fetchData();
  print(data);
}
```

Stream

Used to handle multiple asynchronous values over time.

```
Stream<int> numbers() async* {
  yield 1;
  yield 2;
}
```