Assignment-Dart programming

Theory:

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

Ans: **1. int – Integer Numbers**

What it is:
Used to store **whole numbers** (no decimal point).

Use when:
You need counting, age, quantity, loop counters, etc.

Example:

```dart

int age = 21;
int score = 100;
```

Can be positive or negative
Cannot store decimal values

2. double – Decimal Numbers

What it is:
Used to store **numbers with decimal points**.

Use when:
You need precision like price, height, weight, percentage.

Example:

```dart

double price = 99.99;
double height = 5.8;
```

✓ Supports decimal values
✓ Used in calculations

3. num – Common Type for Numbers

What it is:
Parent type of both int and double.

Use when:
Value can be **either integer or decimal**.

Example:

```dart
num value = 10;
value = 10.5;
```

4. String – Text Data

What it is:
Used to store **text**, words, sentences, characters.

Use when:
Names, addresses, messages, emails, etc.

Example:

```dart
String name = "Harshil";
String address = "Ahmedabad";
```

String interpolation:

```
print("My name is $name");
```

5. bool – True / False

What it is:
Stores only **true or false**.

Use when:
Conditions, decisions, validations.

Example:

```
bool isLoggedIn = true;
bool isAdult = false;
```

6. List – Collection of Items (Array)

What it is:
Stores **multiple values in one variable**.

Use when:
You need a group of values (students, marks, items).

Example:

```
List<int> numbers = [1, 2, 3, 4];
List<String> names = ["Amit", "Rahul", "Neha"];
```

Access elements:

```
print(names[0]); // Amit
```

7. Map – Key–Value Pair

What it is:
Stores data in **key : value** format.

Use when:
You want structured data (like JSON, user details).

Example:

```
Map<String, String> user = {
  "name": "Harshil",
  "city": "Surat"
};
```

Access value:

```
print(user["name"]);
```

8. Set – Unique Values Only

What it is:
Collection that **does not allow duplicate values**.

Use when:
You need unique data (IDs, unique names).

Example:

```
Set<int> numbers = {1, 2, 3, 3};
print(numbers); // {1, 2, 3}
```

9. var – Type Inference

What it is:
Dart automatically decides the type.

Example:

```
var name = "Harshil"; // String
var age = 22;         // int
```

10. Object – Parent of All Types

What it is:
All data types in Dart come from Object.

Example:

```
Object data = "Hello";
data = 10;
```

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

Ans: **1. if Statement**

What it does:
Runs code **only if a condition is true**.

Syntax:

```
if (condition) {
  // code
}
```

Example:

```
int age = 20;

if (age >= 18) {
  print("You are eligible to vote");
}
```

Executes only when condition is true.

2. if – else Statement

What it does:
Chooses between **two paths**.

Example:

```
int marks = 35;

if (marks >= 40) {
  print("Pass");
} else {
  print("Fail");
}
```

One block always executes.

3. if – else if – else

What it does:
Checks **multiple conditions**.

Example:

```
int score = 85;

if (score >= 90) {
  print("Grade A");
} else if (score >= 75) {
  print("Grade B");
} else if (score >= 60) {
  print("Grade C");
} else {
  print("Fail");
}
```

First true condition runs.

4. for Loop

What it does:
Runs code **a fixed number of times**.

Syntax:

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

5. while Loop

What it does:
Repeats code **while condition is true**.

Example:

```
int i = 1;

while (i <= 5) {
  print(i);
  i++;
}
```

Condition checked **before** loop runs.

6. do – while Loop

What it does:
Runs code **at least once**, then checks condition.

Example:

```
int i = 1;

do {
  print(i);
  i++;
} while (i <= 5);
```

Executes minimum once.

7. switch Statement

What it does:
Selects code based on **exact value matching**.

Example:

```
int day = 3;

switch (day) {
  case 1:
    print("Monday");
    break;
  case 2:
    print("Tuesday");
    break;
  case 3:
    print("Wednesday");
    break;
  default:
    print("Invalid day");
}
```

✓ Cleaner than many else if

✓ break is required to stop execution

8. break and continue

break

Stops the loop immediately.

```
for (int i = 1; i <= 5; i++) {
  if (i == 3) break;
  print(i);
}
```

Output:

```
1
2
```

continue

Skips current iteration

```
for (int i = 1; i <= 5; i++) {
  if (i == 3) continue;
  print(i);
}
```

Output:

```
1
2
4
5
```

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

Ans: **Object-Oriented Programming (OOP) in Dart**

OOP is a way of writing programs using **objects and classes** to make code:

- reusable
- organized
- easy to maintain

1. Class and Object

Class

A **class** is a blueprint or template that defines:

- properties (variables)
- behaviors (methods)

Object

An **object** is a real instance of a class.

Example:

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

  Student(this.name, this.age);

  void display() {
    print("Name: $name, Age: $age");
  }
}

void main() {
  Student s1 = Student("Harshil", 21);
  s1.display();
}
```

✓ Student → class
✓ s1 → object

2. Inheritance

What is Inheritance?

Inheritance allows one class (**child**) to **reuse properties and methods** of another class (**parent**).

Achieved using the extends keyword.

Example:

```
class Person {
  void speak() {
    print("Person is speaking");
  }
}

class Student extends Person {
  void study() {
    print("Student is studying");
  }
}

void main() {
  Student s = Student();
  s.speak();  // inherited
  s.study();
}
```

✓ Code reusability

✓ Parent → Child relationship

3. Polymorphism

What is Polymorphism?

One method, different behavior depending on the object.

Achieved using **method overriding**.

Example:

```
class Animal {
  void sound() {
    print("Animal makes a sound");
  }
}

class Dog extends Animal {
  @override
  void sound() {
    print("Dog barks");
  }
}

class Cat extends Animal {
  @override
  void sound() {
    print("Cat meows");
  }
}
```

```
void main() {
  Animal a1 = Dog();
  Animal a2 = Cat();

  a1.sound();
  a2.sound();
}
```

Output:

```
Dog barks
Cat meows
```

✓ Same method name

✓ Different outputs

4. Encapsulation

What is Encapsulation?

Wrapping data and methods together and **protecting data**.

Use:

- private variables (_variableName)
- getters and setters

Example:

```
class BankAccount {
  double _balance = 0;

  double get balance => _balance;

  void deposit(double amount) {
    _balance += amount;
  }
}

void main() {
  BankAccount acc = BankAccount();
  acc.deposit(5000);
  print(acc.balance);
}
```

✓ Data is hidden
✓ Controlled access

5. Abstraction

What is Abstraction?

Hiding implementation details and showing **only essential features**.

Achieved using:

- abstract class
- interfaces

Example (Abstract Class):

```
abstract class Vehicle {
  void start(); // abstract method
}

class Car extends Vehicle {
  @override
  void start() {
    print("Car starts with key");
  }
}

void main() {
  Vehicle v = Car();
  v.start();
}
```