DART ASSIGNMENT- THEORY

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

Ans- The fundamental data types in Dart are:

- **Int**: Represents integer values (e.g., 42, -7). Used for whole numbers without decimals.
- **Double**: Represents floating-point numbers (e.g., 3.14, -0.5). Used for numbers with decimals.
- **String**: Represents a sequence of characters (e.g., "Raj", 'Flutter'). Used for text data.
- **List**: Represents an ordered collection of elements (e.g., [1, 2, 3]). Used for storing multiple values of the same or different types.
- **Map**: Represents a collection of key-value pairs (e.g., {'name': 'Raj', 'age': 21}). Used for associating keys with values.

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

Ans- Control structures in Dart are used to manage the flow of a program based on conditions and loops.

- If and else: used when there is use of conditions in the code.

```
int age = 20;
if (age >= 18) {
  print('You are an adult.');
} else {
  print('You are a minor.');
}
```

- For loop: used to print values according to code.

```
for (int i = 0; i < 5; i++)
{
  print('Number: $i');
}
```

- While loop: used for repeating the loop while condition is true.

```
int count = 0;
while (count < 3) {
  print('Count: $count');
  count++;
}
```

- Do-while loop: it executes the code at least once, and repeat it again if the condition is true.

```
int num = 0;
do
{
 print('Number: $num');
 num++;
}
while (num < 3);
```

- Switch statement: used when there is multiple conditions in the code.

```
String grade = 'A';
switch (grade)
{
 case 'A':
  print('Excellent!');
  break;
 case 'B':
  print('Good job!');
  break;
 default:
  print('Invalid grade');
}
```

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

Ans- Object-Oriented Programming (OOP) in Dart revolves around four core concepts: **classes**, **inheritance**, **polymorphism**, and **interfaces**.

- Classes: it defines properties and methods. Also, it is a blueprint for creating objects.

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

 Person(this.name, this.age);

 void displayInfo()
{
  print('Name: $name, Age: $age');
```

```
  }
}

void main()
{
  var person = Person('Alice', 25);
  person.displayInfo();}
```

- Inheritance: it allows a class to inherit properties and methods of parent class to child class. Inheritance are of different types- Single, Multilevel, Multiple(not supported directly in dart), hierarchical.

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

class Dog extends Animal
{
  void bark() {
    print('Dog barks');
  }
}

void main() {
  var dog = Dog();
  dog.sound();
  dog.bark();
}
```

- Polymorphism: it allows a method to behave differently based on the object calling it. It can be achieved using method overriding.

```
class Shape {
  void draw() {
    print('Drawing a shape');
  }
}

class Circle extends Shape {
  @override
```

```
  void draw() {
    print('Drawing a circle');
  }
}

void main() {
  Shape shape = Circle();  // Polymorphism in action
  shape.draw();
}
```

- Interfaces:

```
abstract class Animal {
  void eat();
}

class Cow implements Animal {
  @override
  void eat() {
    print('Cow eats grass');
  }
}

void main() {
  var cow = Cow();
  cow.eat();
}
```

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

Ans- Asynchronous programming in Dart allows non-blocking execution of tasks, making it ideal for handling operations like network requests or file reading without freezing the application.

- Future: A Future represents a value or error that will be available at some point in the future. It is commonly used for handling asynchronous operations.

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

void main() {
  fetchData().then((data) => print(data));
```

```
}
```

- Async and await: The async keyword is used to define an asynchronous function, while await pauses the execution until the Future is complete.

```
Future<void> fetchData() async {
  print('Fetching data...');
  await Future.delayed(Duration(seconds: 2));
  print('Data fetched');
}

void main() async {
  await fetchData();
  print('Process complete');
}
```

- Stream: A Stream provides a sequence of asynchronous data events over time, making it suitable for handling continuous data flows like user inputs or real-time updates.

```
Stream<int> countStream(int max) async* {
  for (int i = 1; i <= max; i++) {
    await Future.delayed(Duration(seconds: 1));
    yield i;
  }
}

void main() {
  countStream(3).listen((data) => print('Received: $data'));
}
```