Week 1: Dart Programming Fundamentals

Task 1.1: Dart Setup & Basic Syntax (2 hours)

Create a Dart project named "dart_basics" with main.dart file. Write programs demonstrating variables, data types, operators, and control flow (if-else, switch, loops). Create functions for basic operations like sum, factorial, and prime number check. Push to GitHub repository "Flutter-Internship-B01".

Steps:

1. Install Dart SDK
2. Create new directory dart_basics
3. Create main.dart file
4. Demonstrate var, final, const declarations
5. Show int, double, String, bool data types
6. Implement if-else conditions
7. Create for, while loops examples
8. Write function int sum(int a, int b)
9. Create factorial function using recursion
10. Implement isPrime() function
11. Run: dart main.dart
12. Push to GitHub

Task 1.2: Dart Collections & Higher-Order Functions (3 hours)

Create "dart_collections.dart" demonstrating List, Set, Map operations. Implement functions using map(), where(), reduce(), forEach(). Create a student management system using List of Maps to store student data (name, roll, marks). Implement sorting, filtering, and searching operations.

Steps:

1. Create List of integers and demonstrate add, remove, insert
2. Create Set and show unique elements
3. Create Map with key-value pairs
4. Use list.map() to transform data
5. Use list.where() for filtering
6. Implement list.reduce() for sum
7. Create List<Map<String, dynamic>> for students
8. Add 5 student records
9. Sort by marks descending
10. Filter students with marks > 75
11. Search student by name
12. Print formatted output

Task 1.3: Object-Oriented Programming in Dart (4 hours)

Create "dart_oop.dart" implementing OOP concepts. Create classes for User, Product, and ShoppingCart. Demonstrate encapsulation with private variables, inheritance with Admin extending User, polymorphism with method overriding, and abstraction with abstract classes. Implement getters, setters, and constructors.

Steps:

1. Create User class with private _name, _email
2. Add named constructor User.withDetails()
3. Implement getters and setters
4. Create Admin class extending User
5. Override toString() method
6. Create abstract class Animal
7. Implement Dog and Cat classes
8. Add abstract method makeSound()
9. Create Product class with name, price
10. Create ShoppingCart class managing List<Product>
11. Implement addProduct(), removeProduct(), calculateTotal()
12. Test all functionality in main()

Task 1.4: Asynchronous Programming (3 hours)

Create "dart_async.dart" demonstrating Future, async-await, and Stream. Implement functions simulating API calls with delays. Create a function that fetches user data asynchronously, handles errors with try-catch, and uses Future.wait() for parallel requests. Implement a Stream for real-time data updates.

Steps:

1. Create Future<String> fetchUserData() with delayed response
2. Use async-await to call function
3. Implement error handling with try-catch
4. Create multiple Future functions
5. Use Future.wait() for parallel execution
6. Implement Stream<int> counterStream()
7. Listen to stream with await for
8. Handle stream errors
9. Create Stream controller manually
10. Demonstrate stream transformation

Test all async operations.