Assignments for Diploma in Mobile Application Development (Native)

Module 1 – Overview of IT Industry

What is a Program?

***LAB EXERCISE**: Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.*

***THEORY EXERCISE**: Explain in your own words what a program is and how it functions.*

What is Programming?

***THEORY EXERCISE**: What are the key steps involved in the programming process?*

Types of Programming Languages

***THEORY EXERCISE**: What are the main differences between high-level and low-level programming languages?*

World Wide Web & How Internet Works

***LAB EXERCISE**: Research and create a diagram of how data is transmitted from a client to a server over the internet.*

***THEORY EXERCISE**: Describe the roles of the client and server in web communication.*

Network Layers on Client and Server

***LAB EXERCISE**: Design a simple HTTP client-server communication in any language.*

***THEORY EXERCISE**: Explain the function of the TCP/IP model and its layers.*

Client and Servers

***THEORY EXERCISE**: Explain Client Server Communication*

Types of Internet Connections

***LAB EXERCISE**: Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.*

***THEORY EXERCISE**: How does broadband differ from fiber-optic internet?*

Protocols

***LAB EXERCISE**: Simulate HTTP and FTP requests using command line tools (e.g., curl).*

***THEORY EXERCISE**: What are the differences between HTTP and HTTPS protocols?*

Application Security

***LAB EXERCISE**: Identify and explain three common application security vulnerabilities. Suggest possible solutions.*

***THEORY EXERCISE**: What is the role of encryption in securing applications?*

Software Applications and Its Types

***LAB EXERCISE**: Identify and classify 5 applications you use daily as either system software or application software.*

***THEORY EXERCISE**: What is the difference between system software and application software?*

Software Architecture

***LAB EXERCISE**: Design a basic three-tier software architecture diagram for a web application.*

***THEORY EXERCISE**: What is the significance of modularity in software architecture?*

Layers in Software Architecture

***LAB EXERCISE**: Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.*

***THEORY EXERCISE**: Why are layers important in software architecture?*

Software Environments

***LAB EXERCISE**: Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.*

***THEORY EXERCISE**: Explain the importance of a development environment in software production.*

Source Code

***LAB EXERCISE**: Write and upload your first source code file to Github.*

***THEORY EXERCISE**: What is the difference between source code and machine code?*

Github and Introductions

***LAB EXERCISE**: Create a Github repository and document how to commit and push code changes.*

***THEORY EXERCISE**: Why is version control important in software development?*

Student Account in Github

***LAB EXERCISE**: Create a student account on Github and collaborate on a small project with a classmate.*

***THEORY EXERCISE**: What are the benefits of using Github for students?*

Types of Software

***LAB EXERCISE**: Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.*

***THEORY EXERCISE**: What are the differences between open-source and proprietary software?*

GIT and GITHUB Training

***LAB EXERCISE**: Follow a GIT tutorial to practice cloning, branching, and merging repositories.*

***THEORY EXERCISE**: How does GIT improve collaboration in a software development team?*

Application Software

***LAB EXERCISE**: Write a report on the various types of application software and how they improve productivity.*

***THEORY EXERCISE**: What is the role of application software in businesses?*

Software Development Process

***LAB EXERCISE**: Create a flowchart representing the Software Development Life Cycle (SDLC).*

***THEORY EXERCISE**: What are the main stages of the software development process?*

Software Requirement

***LAB EXERCISE**: Write a requirement specification for a simple library management system.*

***THEORY EXERCISE**: Why is the requirement analysis phase critical in software development?*

Software Analysis

***LAB EXERCISE**: Perform a functional analysis for an online shopping system.*

***THEORY EXERCISE**: What is the role of software analysis in the development process?*

System Design

***LAB EXERCISE**: Design a basic system architecture for a food delivery app.*

***THEORY EXERCISE**: What are the key elements of system design?*

Software Testing

***LAB EXERCISE**: Develop test cases for a simple calculator program.*

***THEORY EXERCISE**: Why is software testing important?*

Maintenance

***LAB EXERCISE**: Document a real-world case where a software application required critical maintenance.*

***THEORY EXERCISE**: What types of software maintenance are there?*

Development

***THEORY EXERCISE**: What are the key differences between web and desktop applications?*

27. Web Application

***THEORY EXERCISE**: What are the advantages of using web applications over desktop applications?*

28. Designing

***THEORY EXERCISE**: What role does UI/UX design play in application development?*

29. Mobile Application

***THEORY EXERCISE**: What are the differences between native and hybrid mobile apps?*

30. DFD (Data Flow Diagram)

***LAB EXERCISE**: Create a DFD for a hospital management system.*

***THEORY EXERCISE**: What is the significance of DFDs in system analysis?*

31. Desktop Application

***LAB EXERCISE**: Build a simple desktop calculator application using a GUI library.*

***THEORY EXERCISE**: What are the pros and cons of desktop applications compared to web applications?*

32. Flow Chart

***LAB EXERCISE**: Draw a flowchart representing the logic of a basic online registration system.*

***THEORY EXERCISE**: How do flowcharts help in programming and system design?*

Module 2 – Introduction to Programming

Overview of C Programming

- **THEORY EXERCISE**:
  - Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.
- **LAB EXERCISE**:
  - Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.

2. Setting Up Environment

- **THEORY EXERCISE**:
  - Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.
- **LAB EXERCISE**:
  - Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.

3. Basic Structure of a C Program

- **THEORY EXERCISE**:
  - Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.
- **LAB EXERCISE**:
  - Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.

4. Operators in C

- **THEORY EXERCISE**:
  - Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.
- **LAB EXERCISE**:
  - Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.
  - WAP to Find Area And Circumference of Circle
  - Find Area of Square formula : a = a2 5. Find Area of Cube formula : a = 6a2 6. Find area of Triangle Formula : A = 1/2 × b × h
  - Accept number of students from user. I need to give 5 apples to each student. How many apples are required?
  - Find character value from ascii
  - Find ascii value of given number

5. Control Flow Statements in C

- **THEORY EXERCISE**:
  - Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

- **LAB EXERCISE**:
  - Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user's input (1 for January, 2 for February, etc.).

  - Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user's input (1 for January, 2 for February, etc.).

  - WAP to find maximum number among 3 numbers using ternary operator

  - Write a C program to calculate profit and loss on a transaction

6. Looping in C

- **THEORY EXERCISE**:
  - Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.
- **LAB EXERCISE**:
  - Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).
  - WAP to take 10 no. Input from user find out below values a. How many Even numbers are there b. How many odd numbers are there c. Sum of even numbers d. Sum of odd numbers?
  - WAP to print number in reverse order e.g.: number = 64728 ---> reverse = 82746

7. Loop Control Statements

- **THEORY EXERCISE**:
  - Explain the use of `break`, `continue`, and `goto` statements in C. Provide examples of each.
- **LAB EXERCISE**:
  - Write a C program that uses the `break` statement to stop printing numbers when it reaches 5. Modify the program to skip printing the number 3 using the `continue` statement.
  - Calculate the Sum of Natural Numbers Using the While Loop
  - Program of Armstrong Number in C Using For Loop & While Loop
  - WAP to accept 5 numbers from user and display in reverse order using for loop and array

8. Functions in C

- **THEORY EXERCISE**:
  - What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.
- **LAB EXERCISE**:
  - Write a C program that calculates the factorial of a number using a function Include function declaration, definition, and call.
  - WAP to find factorial using recursion
  - WAP to reverse a string and check that the string is palindrome or no

9. Arrays in C

- **THEORY EXERCISE**:
  - Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.
- **LAB EXERCISE**:
  - Write a C program that stores 5 integers in a one-dimensional array and prints

them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

- Accept number from user store in array
- Convert array into asce and dec order
- Find max element from the array

10. Pointers in C

- **THEORY EXERCISE**:
  - Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?
- **LAB EXERCISE**:
  - Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.

11. Strings in C

- **THEORY EXERCISE**:
  - Explain string handling functions like `strlen()`, `strcpy()`, `strcat()`, `strcmp()`, and `strchr()`. Provide examples of when these functions are useful.
- **LAB EXERCISE**:
  - Write a C program that takes two strings from the user and concatenates them using `strcat()`. Display the concatenated string and its length using `strlen()`.
  - Find length of string which is entered by user without using inbuilt function.
  - Join 2 strings using of user defined function without using inbuilt function
  - Accept string from user and check it is palindrome or not

12. Structures in C

- **THEORY EXERCISE**:
  - Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.
- **LAB EXERCISE**:
  - Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.

13. File Handling in C

- **THEORY EXERCISE**:
  - Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.
- **LAB EXERCISE**:
  - Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.

EXTRA LAB EXERCISES FOR IMPROVING PROGRAMMING LOGIC

1. Operators

LAB EXERCISE 1: Simple Calculator

- Write a C program that acts as a simple calculator. The program should take two numbers and an operator as input from the user and perform the respective operation (addition, subtraction, multiplication, division, or modulus) using operators.
- **Challenge**: Extend the program to handle invalid operator inputs.

LAB EXERCISE 2: Check Number Properties

- Write a C program that takes an integer from the user and checks the following using different operators:
  - Whether the number is even or odd.
  - Whether the number is positive, negative, or zero.

- Whether the number is a multiple of both 3 and 5.

---

2. Control Statements

LAB EXERCISE 1: Grade Calculator

- Write a C program that takes the marks of a student as input and displays the corresponding grade based on the following conditions:
  - Marks > 90: Grade A
  - Marks > 75 and <= 90: Grade B
  - Marks > 50 and <= 75: Grade C
  - Marks <= 50: Grade D
- Use *if-else* or *switch* statements for the decision-making process.

LAB EXERCISE 2: Number Comparison

- Write a C program that takes three numbers from the user and determines:
  - The largest number.
  - The smallest number.
- **Challenge**: Solve the problem using both *if-else* and *switch-case* statements.

LAB EXERCISE 3: Temperature calculation

- Write a C program to read temperature in centigrade and display a suitable message according to the temperature state below: Temp < 0 then Freezing weather Temp 0-10 then Very Cold weather Temp 10-20 then Cold weather Temp 20-30 then Normal in Temp Temp 30-40 then Its Hot Temp >=40 then Its Very Hot

---

3. Loops

LAB EXERCISE 1: Prime Number Check

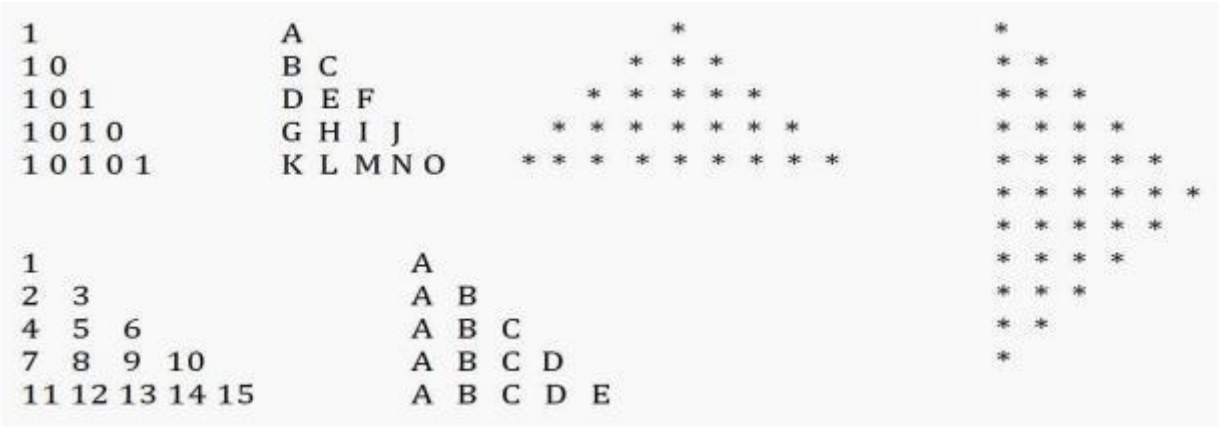
- Write a C program that checks whether a given number is a prime number or not using a *for* loop.
- **Challenge**: Modify the program to print all prime numbers between 1 and a given number.

LAB EXERCISE 2: Multiplication Table

- Write a C program that takes an integer input from the user and prints its multiplication table using a *for* loop.
- **Challenge**: Allow the user to input the range of the multiplication table (e.g., from 1 to N).

LAB EXERCISE 3: Sum of Digits

- Write a C program that takes an integer from the user and calculates the sum of its digits using a *while* loop.
- **Challenge**: Extend the program to reverse the digits of the number.
- **Patterns** :



- Write a program to find out the max from given number (E.g., No: -1562 Max number is 6)

4. Arrays

LAB EXERCISE 1: Maximum and Minimum in Array

- Write a C program that accepts 10 integers from the user and stores them in an array. The program should then find and print the maximum and minimum values in the array.

- **Challenge**: Extend the program to sort the array in ascending order.

LAB EXERCISE 2: Matrix Addition

- Write a C program that accepts two 2x2 matrices from the user and adds them. Display the resultant matrix.
- **Challenge**: Extend the program to work with 3x3 matrices and matrix multiplication.

LAB EXERCISE 3: Sum of Array Elements

- Write a C program that takes N numbers from the user and stores them in an array. The program should then calculate and display the sum of all array elements.
- **Challenge**: Modify the program to also find the average of the numbers.

---

5. Functions

LAB EXERCISE 1: Fibonacci Sequence

- Write a C program that generates the Fibonacci sequence up to N terms using a recursive function.
- **Challenge**: Modify the program to calculate the Nth Fibonacci number using both iterative and recursive methods. Compare their efficiency.

LAB EXERCISE 2: Factorial Calculation

- Write a C program that calculates the factorial of a given number using a function.
- **Challenge**: Implement both an iterative and a recursive version of the factorial function and compare their performance for large numbers.

LAB EXERCISE 3: Palindrome Check

- Write a C program that takes a number as input and checks whether it is a palindrome using a function.
- **Challenge**: Modify the program to check if a given string is a palindrome.

---

6. Strings

LAB EXERCISE 1: String Reversal

- Write a C program that takes a string as input and reverses it using a function.
- **Challenge**: Write the program without using built-in string handling functions.

LAB EXERCISE 2: Count Vowels and Consonants

- Write a C program that takes a string from the user and counts the number of vowels and consonants in the string.
- **Challenge**: Extend the program to also count digits and special characters.

LAB EXERCISE 3: Word Count

- Write a C program that counts the number of words in a sentence entered by the user.
- **Challenge**: Modify the program to find the longest word in the sentence.

Extra Logic Building Challenges

Lab Challenge 1: Armstrong Number

- Write a C program that checks whether a given number is an Armstrong number or not (e.g., 153 = 1^3 + 5^3 + 3^3).
- **Challenge**: Write a program to find all Armstrong numbers between 1 and 1000.

Lab Challenge 2: Pascal's Triangle

- Write a C program that generates Pascal's Triangle up to N rows using loops.
- **Challenge**: Implement the same program using a recursive function.

Lab Challenge 3: Number Guessing Game

- Write a C program that implements a simple number guessing game. The program should generate a random number between 1 and 100, and the user should guess the number within a limited number of attempts.
- **Challenge**: Provide hints to the user if the guessed number is too high or too low.

Lab Challenge 4: Sum of Prime Numbers

- **Description**: Write a C program that calculates the sum of all prime numbers up to a given number N.
- **Challenge**: Extend the program to find and print all the prime numbers found.

Module #3 Introduction to OOPS Programming

1. Introduction to C++

LAB EXERCISES:

1. *First C++ Program: Hello World*
   - Write a simple C++ program to display "Hello, World!".
   - *Objective*: Understand the basic structure of a C++ program, including **#include**, **main()**, and **cout**.
2. *Basic Input/Output*
   - Write a C++ program that accepts user input for their name and age and then displays a personalized greeting.
   - *Objective*: Practice input/output operations using **cin** and **cout**.
3. *POP vs. OOP Comparison Program*
   - Write two small programs: one using Procedural Programming (POP) to calculate the area of a rectangle, and another using Object-Oriented Programming (OOP) with a class and object for the same task.
   - *Objective*: Highlight the difference between POP and OOP approaches.
4. *Setting Up Development Environment*
   - Write a program that asks for two numbers and displays their sum. Ensure this is done after setting up the IDE (like Dev C++ or CodeBlocks).
   - *Objective*: Help students understand how to install, configure, and run programs in an IDE.

THEORY EXERCISE:

1. *What are the key differences between Procedural Programming and Object-Oriented Programming (OOP)?*
2. *List and explain the main advantages of OOP over POP.*
3. *Explain the steps involved in setting up a C++ development environment.*
4. *What are the main input/output operations in C++? Provide examples.*

---

2. Variables, Data Types, and Operators

LAB EXERCISES:

1. *Variables and Constants*
   - Write a C++ program that demonstrates the use of variables and constants. Create variables of different data types and perform operations on them.
   - *Objective*: Understand the difference between variables and constants.
2. *Type Conversion*
   - Write a C++ program that performs both implicit and explicit type conversions and prints the results.

- *Objective*: Practice type casting in C++.

3. *Operator Demonstration*
   - Write a C++ program that demonstrates arithmetic, relational, logical, and bitwise operators. Perform operations using each type of operator and display the results.
   - *Objective*: Reinforce understanding of different types of operators in C++.

THEORY EXERCISE:

1. *What are the different data types available in C++? Explain with examples.*
2. *Explain the difference between implicit and explicit type conversion in C++.*
3. *What are the different types of operators in C++? Provide examples of each.*
4. *Explain the purpose and use of constants and literals in C++.*

---

3. Control Flow Statements

LAB EXERCISES:

1. *Grade Calculator*
   - Write a C++ program that takes a student's marks as input and calculates the grade based on if-else conditions.
   - *Objective*: Practice conditional statements (**if-else**).
2. *Number Guessing Game*
   - Write a C++ program that asks the user to guess a number between 1 and 100. The program should provide hints if the guess is too high or too low. Use loops to allow the user multiple attempts.
   - *Objective*: Understand **while** loops and conditional logic.
3. *Multiplication Table*
   - Write a C++ program to display the multiplication table of a given number using a **for** loop.
   - *Objective*: Practice using loops.
4. *Nested Control Structures*
   - Write a program that prints a right-angled triangle using stars (*) with a nested loop.
   - *Objective*: Learn nested control structures.

THEORY EXERCISE:

1. *What are conditional statements in C++? Explain the* if-else *and* switch *statements.*
2. *What is the difference between* for, while*, and* do-while *loops in C++?*
3. *How are* break *and* continue *statements used in loops? Provide examples.*
4. *Explain nested control structures with an example.*

---

4. Functions and Scope

LAB EXERCISES:

1. *Simple Calculator Using Functions*
    - Write a C++ program that defines functions for basic arithmetic operations (add, subtract, multiply, divide). The main function should call these based on user input.
    - *Objective*: Practice defining and using functions in C++.
2. *Factorial Calculation Using Recursion*
    - Write a C++ program that calculates the factorial of a number using recursion.
    - *Objective*: Understand recursion in functions.
3. *Variable Scope*
    - Write a program that demonstrates the difference between local and global variables in C++. Use functions to show scope.
    - *Objective*: Reinforce the concept of variable scope.

THEORY EXERCISE:

1. *What is a function in C++? Explain the concept of function declaration, definition, and calling.*
2. *What is the scope of variables in C++? Differentiate between local and global scope.*
3. *Explain recursion in C++ with an example.*
4. *What are function prototypes in C++? Why are they used?*

---

5. Arrays and Strings

LAB EXERCISES:

1. *Array Sum and Average*
    - Write a C++ program that accepts an array of integers, calculates the sum and average, and displays the results.
    - *Objective*: Understand basic array manipulation.
2. *Matrix Addition*
    - Write a C++ program to perform matrix addition on two 2x2 matrices.
    - *Objective*: Practice multi-dimensional arrays.
3. *String Palindrome Check*
    - Write a C++ program to check if a given string is a palindrome (reads the same forwards and backwards).
    - *Objective*: Practice string operations.

THEORY EXERCISE:

1. *What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.*
2. *Explain string handling in C++ with examples.*
3. *How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.*
4. *Explain string operations and functions in C++.*

6. Introduction to Object-Oriented Programming

LAB EXERCISES:

1. *Class for a Simple Calculator*
   - Write a C++ program that defines a class **Calculator** with functions for addition, subtraction, multiplication, and division. Create objects to use these functions.
   - *Objective*: Introduce basic class structure.
2. *Class for Bank Account*
   - Create a class **BankAccount** with data members like **balance** and member functions like **deposit** and **withdraw**. Implement encapsulation by keeping the data members private.
   - *Objective*: Understand encapsulation in classes.
3. *Inheritance Example*
   - Write a program that implements inheritance using a base class **Person** and derived classes **Student** and **Teacher**. Demonstrate reusability through inheritance.
   - *Objective*: Learn the concept of inheritance.

THEORY EXERCISE:

1. ***Explain the key concepts of Object-Oriented Programming (OOP).***
2. ***What are classes and objects in C++? Provide an example.***
3. ***What is inheritance in C++? Explain with an example.***
4. ***What is encapsulation in C++? How is it achieved in classes?***

Module 4 – Introduction to DBMS

Introduction to SQL

Theory Questions:

1. What is SQL, and why is it essential in database management?
2. Explain the difference between DBMS and RDBMS.
3. Describe the role of SQL in managing relational databases.
4. What are the key features of SQL?

LAB EXERCISES:

- **Lab 1**: Create a new database named `school_db` and a table called `students` with the following columns: `student_id`, `student_name`, `age`, `class`, and `address`.
- **Lab 2**: Insert five records into the `students` table and retrieve all records using the `SELECT` statement.

---

2. SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?
2. Write the general structure of an SQL `SELECT` statement.
3. Explain the role of clauses in SQL statements.

LAB EXERCISES:

- **Lab 1**: Write SQL queries to retrieve specific columns (`student_name` and `age`) from the `students` table.
- **Lab 2**: Write SQL queries to retrieve all students whose age is greater than 10.

---

3. SQL Constraints

Theory Questions:

1. What are constraints in SQL? List and explain the different types of constraints.
2. How do `PRIMARY KEY` and `FOREIGN KEY` constraints differ?
3. What is the role of `NOT NULL` and `UNIQUE` constraints?

LAB EXERCISES:

- **Lab 1**: Create a table `teachers` with the following columns: `teacher_id` (Primary Key), `teacher_name` (NOT NULL), `subject` (NOT NULL), and `email` (UNIQUE).
- **Lab 2**: Implement a `FOREIGN KEY` constraint to relate the `teacher_id` from the `teachers` table with the `students` table.

---

4. Main SQL Commands and Sub-commands (DDL)

Theory Questions:

1. Define the SQL Data Definition Language (DDL).
2. Explain the `CREATE` command and its syntax.
3. What is the purpose of specifying data types and constraints during table creation?

LAB EXERCISES:

- **Lab 1**: Create a table `courses` with columns: `course_id`, `course_name`, and `course_credits`. Set the `course_id` as the primary key.
- **Lab 2**: Use the `CREATE` command to create a database `university_db`.

---

5. ALTER Command

Theory Questions:

1. What is the use of the `ALTER` command in SQL?
2. How can you add, modify, and drop columns from a table using `ALTER`?

LAB EXERCISES:

- **Lab 1**: Modify the `courses` table by adding a column `course_duration` using the `ALTER` command.
- **Lab 2**: Drop the `course_credits` column from the `courses` table.

---

6. DROP Command

Theory Questions:

1. What is the function of the `DROP` command in SQL?
2. What are the implications of dropping a table from a database?

LAB EXERCISES:

- **Lab 1**: Drop the `teachers` table from the `school_db` database.
- **Lab 2**: Drop the `students` table from the `school_db` database and verify that the table has been removed.

---

7. Data Manipulation Language (DML)

Theory Questions:

1. Define the `INSERT`, `UPDATE`, and `DELETE` commands in SQL.
2. What is the importance of the `WHERE` clause in `UPDATE` and `DELETE` operations?

LAB EXERCISES:

- **Lab 1**: Insert three records into the `courses` table using the `INSERT` command.
- **Lab 2**: Update the course duration of a specific course using the `UPDATE` command.
- **Lab 3**: Delete a course with a specific `course_id` from the `courses` table using the `DELETE` command.

---

8. Data Query Language (DQL)

Theory Questions:

1. What is the `SELECT` statement, and how is it used to query data?
2. Explain the use of the `ORDER BY` and `WHERE` clauses in SQL queries.

LAB EXERCISES:

- **Lab 1**: Retrieve all courses from the `courses` table using the `SELECT` statement.
- **Lab 2**: Sort the courses based on `course_duration` in descending order using `ORDER BY`.
- **Lab 3**: Limit the results of the `SELECT` query to show only the top two courses using `LIMIT`.

---

9. Data Control Language (DCL)

Theory Questions:

1. What is the purpose of `GRANT` and `REVOKE` in SQL?
2. How do you manage privileges using these commands?

LAB EXERCISES:

- **Lab 1**: Create two new users `user1` and `user2` and grant `user1` permission to `SELECT` from the `courses` table.
- **Lab 2**: Revoke the `INSERT` permission from `user1` and give it to `user2`.

---

10. Transaction Control Language (TCL)

Theory Questions:

1. What is the purpose of the `COMMIT` and `ROLLBACK` commands in SQL?
2. Explain how transactions are managed in SQL databases.

LAB EXERCISES:

- **Lab 1**: Insert a few rows into the `courses` table and use `COMMIT` to save the changes.
- **Lab 2**: Insert additional rows, then use `ROLLBACK` to undo the last insert operation.
- **Lab 3**: Create a `SAVEPOINT` before updating the `courses` table, and use it to roll back specific changes.

---

11. SQL Joins

Theory Questions:

1. Explain the concept of `JOIN` in SQL. What is the difference between `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and `FULL OUTER JOIN`?
2. How are joins used to combine data from multiple tables?

LAB EXERCISES:

- **Lab 1**: Create two tables: `departments` and `employees`. Perform an `INNER JOIN` to display employees along with their respective departments.
- **Lab 2**: Use a `LEFT JOIN` to show all departments, even those without employees.

---

12. SQL Group By

Theory Questions:

1. What is the `GROUP BY` clause in SQL? How is it used with aggregate functions?
2. Explain the difference between `GROUP BY` and `ORDER BY`.

LAB EXERCISES:

- **Lab 1**: Group employees by department and count the number of employees in each department using `GROUP BY`.
- **Lab 2**: Use the `AVG` aggregate function to find the average salary of employees in each department.

---

13. SQL Stored Procedure

Theory Questions:

1. What is a stored procedure in SQL, and how does it differ from a standard SQL query?
2. Explain the advantages of using stored procedures.

LAB EXERCISES:

- **Lab 1**: Write a stored procedure to retrieve all employees from the `employees` table based on department.
- **Lab 2**: Write a stored procedure that accepts `course_id` as input and returns the course details.

---

14. SQL View

Theory Questions:

1. What is a view in SQL, and how is it different from a table?
2. Explain the advantages of using views in SQL databases.

LAB EXERCISES:

- **Lab 1**: Create a view to show all employees along with their department names.
- **Lab 2**: Modify the view to exclude employees whose salaries are below $50,000.

---

15. SQL Triggers

Theory Questions:

1. What is a trigger in SQL? Describe its types and when they are used.
2. Explain the difference between `INSERT`, `UPDATE`, and `DELETE` triggers.

LAB EXERCISES:

- **Lab 1**: Create a trigger to automatically log changes to the `employees` table when a new employee is added.
- **Lab 2**: Create a trigger to update the `last_modified` timestamp whenever an employee record is updated.

---

16. Introduction to PL/SQL

Theory Questions:

1. What is PL/SQL, and how does it extend SQL's capabilities?
2. List and explain the benefits of using PL/SQL.

LAB EXERCISES:

- **Lab 1**: Write a PL/SQL block to print the total number of employees from the `employees` table.
- **Lab 2**: Create a PL/SQL block that calculates the total sales from an `orders` table.

---

17. PL/SQL Control Structures

Theory Questions:

1. What are control structures in PL/SQL? Explain the `IF-THEN` and `LOOP` control structures.
2. How do control structures in PL/SQL help in writing complex queries?

LAB EXERCISES:

- *Lab 1*: Write a PL/SQL block using an `IF-THEN` condition to check the department of an employee.
- **Lab 2**: Use a `FOR LOOP` to iterate through employee records and display their names.

---

18. SQL Cursors

Theory Questions:

1. What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.
2. When would you use an explicit cursor over an implicit one?

LAB EXERCISES:

- **Lab 1**: Write a PL/SQL block using an explicit cursor to retrieve and display employee details.
- **Lab 2**: Create a cursor to retrieve all courses and display them one by one.

---

19. Rollback and Commit Savepoint

Theory Questions:

1. Explain the concept of `SAVEPOINT` in transaction management. How do `ROLLBACK` and `COMMIT` interact with savepoints?
2. When is it useful to use savepoints in a database transaction?

LAB EXERCISES:

- **Lab 1**: Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.
- **Lab 2**: Commit part of a transaction after using a savepoint and then rollback the remaining changes.

EXTRA LAB PRACTISE FOR DATABASE CONCEPTS

1. Introduction to SQL

LAB EXERCISES:

- **Lab 3**: Create a database called `library_db` and a table `books` with columns: `book_id`, `title`, `author`, `publisher`, `year_of_publication`, and `price`. Insert five records into the table.
- **Lab 4**: Create a table `members` in `library_db` with columns: `member_id`, `member_name`, `date_of_membership`, and `email`. Insert five records into this table.

---

2. SQL Syntax

LAB EXERCISES:

- **Lab 3**: Retrieve all `members` who joined the library before 2022. Use appropriate SQL syntax with `WHERE` and `ORDER BY`.
- **Lab 4**: Write SQL queries to display the titles of books published by a specific author. Sort the results by `year_of_publication` in descending order.

---

3. SQL Constraints

LAB EXERCISES:

- **Lab 3**: Add a `CHECK` constraint to ensure that the `price` of books in the `books` table is greater than 0.
- **Lab 4**: Modify the `members` table to add a `UNIQUE` constraint on the `email` column, ensuring that each member has a unique email address.

---

4. Main SQL Commands and Sub-commands (DDL)

LAB EXERCISES:

- **Lab 3**: Create a table `authors` with the following columns: `author_id`, `first_name`, `last_name`, and `country`. Set `author_id` as the primary key.
- **Lab 4**: Create a table `publishers` with columns: `publisher_id`, `publisher_name`, `contact_number`, and `address`. Set `publisher_id` as the primary key and `contact_number` as unique.

---

5. ALTER Command

LAB EXERCISES:

- **Lab 3**: Add a new column `genre` to the `books` table. Update the `genre` for all existing records.
- **Lab 4**: Modify the `members` table to increase the length of the `email` column to 100 characters.

---

6. DROP Command

LAB EXERCISES:

- **Lab 3**: Drop the `publishers` table from the database after verifying its structure.
- **Lab 4**: Create a backup of the `members` table and then drop the original `members` table.

---

7. Data Manipulation Language (DML)

LAB EXERCISES:

- **Lab 4**: Insert three new authors into the `authors` table, then update the last name of one of the authors.
- **Lab 5**: Delete a book from the `books` table where the `price` is higher than $100.

---

8. UPDATE Command

LAB EXERCISES:

- **Lab 3**: Update the `year_of_publication` of a book with a specific `book_id`.
- **Lab 4**: Increase the `price` of all books published before 2015 by 10%.

---

9. DELETE Command

LAB EXERCISES:

- **Lab 3**: Remove all members who joined before 2020 from the `members` table.
- **Lab 4**: Delete all books that have a `NULL` value in the `author` column.

---

10. Data Query Language (DQL)

LAB EXERCISES:

- **Lab 4**: Write a query to retrieve all `books` with `price` between $50 and $100.
- **Lab 5**: Retrieve the list of `books` sorted by `author` in ascending order and limit the results to the top 3 entries.

---

11. Data Control Language (DCL)

LAB EXERCISES:

- **Lab 3**: Grant `SELECT` permission to a user named `librarian` on the `books` table.
- **Lab 4**: Grant `INSERT` and `UPDATE` permissions to the user `admin` on the `members` table.

---

12. REVOKE Command

LAB EXERCISES:

- **Lab 3**: Revoke the `INSERT` privilege from the user `librarian` on the `books` table.
- **Lab 4**: Revoke all permissions from user `admin` on the `members` table.

---

13. Transaction Control Language (TCL)

LAB EXERCISES:

- **Lab 3**: Use `COMMIT` after inserting multiple records into the `books` table, then make another insertion and perform a `ROLLBACK`.
- **Lab 4**: Set a `SAVEPOINT` before making updates to the `members` table, perform some updates, and then roll back to the `SAVEPOINT`.

---

14. SQL Joins

LAB EXERCISES:

- **Lab 3**: Perform an `INNER JOIN` between `books` and `authors` tables to display the `title` of books and their respective authors' names.
- **Lab 4**: Use a `FULL OUTER JOIN` to retrieve all records from the `books` and `authors` tables, including those with no matching entries in the other table.

---

15. SQL Group By

LAB EXERCISES:

- **Lab 3**: Group `books` by `genre` and display the total number of books in each genre.
- **Lab 4**: Group `members` by the year they joined and find the number of members who joined each year.

---

16. SQL Stored Procedure

LAB EXERCISES:

- **Lab 3**: Write a stored procedure to retrieve all `books` by a particular `author`.
- **Lab 4**: Write a stored procedure that takes `book_id` as an argument and returns the `price` of the book.

---

17. SQL View

LAB EXERCISES:

- **Lab 3**: Create a view to show only the `title`, `author`, and `price` of books from the `books` table.
- **Lab 4**: Create a view to display `members` who joined before 2020.

---

18. SQL Trigger

LAB EXERCISES:

- **Lab 3**: Create a trigger to automatically update the `last_modified` timestamp of the `books` table whenever a record is updated.
- **Lab 4**: Create a trigger that inserts a log entry into a `log_changes` table whenever a `DELETE` operation is performed on the `books` table.

---

19. Introduction to PL/SQL

LAB EXERCISES:

- **Lab 3**: Write a PL/SQL block to insert a new `book` into the `books` table and display a confirmation message.
- **Lab 4**: Write a PL/SQL block to display the total number of books in the `books` table.

---

20. PL/SQL Syntax

LAB EXERCISES:

- **Lab 3**: Write a PL/SQL block to declare variables for `book_id` and `price`, assign values, and display the results.
- **Lab 4**: Write a PL/SQL block using `constants` and perform arithmetic operations on book prices.

---

21. PL/SQL Control Structures

LAB EXERCISES:

- **Lab 3**: Write a PL/SQL block using `IF-THEN-ELSE` to check if a book's price is above $100and print a message accordingly.
- **Lab 4**: Use a `FOR LOOP` in PL/SQL to display the details of all books one by one.

---

22. SQL Cursors

LAB EXERCISES:

- **Lab 3**: Write a PL/SQL block using an explicit cursor to fetch and display all records from the `members` table.
- **Lab 4**: Create a cursor to retrieve books by a particular author and display their titles.

---

23. Rollback and Commit Savepoint

LAB EXERCISES:

- **Lab 3**: Perform a transaction that includes inserting a new `member`, setting a `SAVEPOINT`,and rolling back to the savepoint after making updates.
- **Lab 4**: Use `COMMIT` after successfully inserting multiple books into the `books` table, then use `ROLLBACK` to undo a set of changes made after a savepoint.

Flutter Assignments

Module 1: Introduction to Mobile Development and Flutter

Theory Assignments:

1. Explain the benefits of using Flutter over other cross-platform frameworks.
2. Describe the role of Dart in Flutter. What are its advantages for mobile development?
3. Outline the steps to set up a Flutter development environment.
4. Describe the basic Flutter app structure, explaining main.dart, the main function, and the widget tree.

Practical Assignments:

1. Set up the Flutter development environment on your system and verify installation with

flutter doctor.

2. Create a new Flutter project and customize it to display a personalized “Hello, World!” message with your name.

3. Customize the app by adding a background color and using different text styles (font, size, color) for the welcome message.

Module 2: Dart Programming Essentials

Theory Assignments:

1. Explain the fundamental data types in Dart (int, double, String, List, Map, etc.) and their uses.
2. Describe control structures in Dart with examples of if, else, for, while, and switch.
3. Explain object-oriented programming concepts in Dart, such as classes, inheritance, polymorphism, and interfaces.
4. Describe asynchronous programming in Dart, including Future, async, await, and Stream.

Practical Assignments:

Basic Syntax and Data Types

Task 1:
Write a program that takes a user's name and age as input and prints a welcome message that includes their name and how many years they have left until they turn 100.

Task 2:
Create a program that converts temperature from Celsius to Fahrenheit and vice versa. Implement functions to handle both conversions and let the user choose the conversion type.

Task 3:
Write a program to take a number from the user and determine if it’s even or odd.

Task 4:
Create a program that calculates the area and circumference of a circle. Use constants to store the value of pi.

---

2. Control Flow Statements

Task 5:
Write a program that prints all numbers from 1 to 100. For multiples of 3, print "Fizz" instead of the number, and for multiples of 5, print "Buzz." For numbers that are multiples of both 3 and 5, print "FizzBuzz."

Task 6:
Create a simple grading system that takes a student's score as input and prints their grade:

- A: 90–100
- B: 80–89
- C: 70–79
- D: 60–69
- F: Below 60

Task 7:
Write a program that accepts a number and checks if it is a prime number or not.

Task 8:
Implement a basic calculator that performs addition, subtraction, multiplication, or division based on the user's choice.

---

3. Functions

Task 9:
Write a function to calculate the factorial of a number entered by the user.

Task 10:
Create a function that checks if a string is a palindrome (reads the same backward as forward).

Task 11:
Write a recursive function to generate the Fibonacci series up to a specified number.

Task 12:
Create a function that accepts a list of numbers and returns the largest and smallest numbers in the list.

---

4. Collections (Lists, Sets, and Maps)

Task 13:
Write a program to input a list of integers and sort them in ascending and descending order without using built-in sort methods.

Task 14:
Create a program that takes a list of words and removes any duplicates. Use a set to eliminate duplicates, then display the unique words in alphabetical order.

Task 15:
Write a program that counts the frequency of each character in a given string and stores the result in a map.

Task 16:
Create a simple address book using a map, where the keys are names and the values are phone numbers. Add, update, and remove entries based on user input.

---

5. Object-Oriented Programming

Task 17:
Create a class called Book with properties like title, author, and publication year. Add methods to display the book's details and a method to check if it's over 10 years old.

Task 18:
Define a BankAccount class with properties like account number, account holder, and balance. Add methods to deposit, withdraw, and check the balance. Ensure the withdraw method doesn't allow overdrafts.

Task 19:
Create a class hierarchy with a Vehicle superclass and Car and Bike subclasses. Implement methods in each subclass that print specific details, like the type of vehicle, fuel type, and max speed.

Task 20:
Write a program that simulates a shopping cart. Define classes for Product, Cart, and Order. Allow users to add products to the cart and calculate the total price of the items.

---

6. Exception Handling

Task 21:
Create a program that accepts a number from the user and performs division by another number. Use exception handling to manage division by zero errors.

Task 22:
Write a program that reads a file and displays its contents. Handle potential file not found exceptions and display an error message if the file doesn't exist.

Task 23:
Implement a calculator that catches invalid input errors (like entering a string instead of a number). Display appropriate error messages and ask for re-entry.

Task 24:
Create a program that accepts a list of integers from the user. Use exception handling to handle cases where the user inputs non-integer values.

---

7. Asynchronous Programming

Task 25:
Write a program that simulates a delayed operation using Future.delayed. Display a loading message, wait for 3 seconds, and then show a completion message.

Task 26:
Create a program that fetches data from a fake API endpoint (using the http package). Display the data after it's loaded and catch any errors if the request fails.

Task 27:
Write a function that simulates fetching multiple data points (e.g., list of users) asynchronously. Use await and async keywords to wait for each "data point" to load, then display all the data once loaded.

Task 28:
Implement a mock weather fetching program. Display different messages as it waits for a response and then shows a message like "Weather data loaded successfully."

---

8. Advanced Dart Features

Task 29:
Write a program that uses the spread operator to combine multiple lists into one list. Remove duplicates and sort the list in ascending order.

Task 30:
Create a function that uses higher-order functions. Define a List of numbers and pass it to a function that returns a list of squares, cubes, or halves based on the function passed as an argument.

Task 31:
Write a program that uses async* to create a stream of integers. Display each integer as it's emitted and stop the stream after a certain count.

Task 32:
Implement a simple number guessing game where the computer generates a random number, and the user has to guess it. Use a lambda function to provide hints, such as “too high” or “too low.”

---

9. Dart Packages and Libraries

Task 33:
Explore and integrate the intl package to format dates and numbers in different locales.

Task 34:
Create a program that generates a QR code from a given string using a QR code package, and display it in the console.

Task 35:
Write a program that uses the path package to manipulate file paths. Extract the directory, filename, and extension from a file path string.

Task 36:
Implement a basic file reading and writing program using the dart:io library. Write data to a file and read it back, handling any file errors that may occur.

Module 3: Introduction to Flutter Widgets and UI Components

Theory Assignments:

1. Explain the difference between Stateless and Stateful widgets with examples.
2. Describe the widget lifecycle and how state is managed in Stateful widgets.
3. List and describe five common Flutter layout widgets (e.g., Container, Column, Row).

Practical Assignments:

Introduction to Basic Widgets

Task 1:
Create a simple "Hello World" Flutter app with a Text widget displaying "Hello, Flutter!" centered on the screen.

Task 2:
Build a Flutter app with an AppBar that has a title and a FloatingActionButton. Use the button to increment a counter displayed in the center of the screen.

Task 3:
Create an app with two Text widgets. Apply different TextStyle properties (like font size, color, and weight) to each widget to explore styling options.

Task 4:
Make a "Login Screen" with two TextField widgets for email and password inputs, and a RaisedButton or ElevatedButton widget for the login button.

---

2. Working with Layout Widgets

Task 5:
Create a responsive layout using a Row widget with three Container widgets, each with different background colors and widths.

Task 6:
Design a profile card with Column and Row widgets. Include an avatar (using CircleAvatar), a name, and a short bio text.

Task 7:
Build a grid of images using the GridView widget. Load images from the network, and display four images per row.

Task 8:
Create a screen with ListView.builder to display a list of names. When tapped, each name should print "Name tapped" in the console.

---

3. Buttons and User Interaction

Task 9:
Build a simple calculator UI with RaisedButton or ElevatedButton widgets for numbers and operations. Display the result at the top of the screen.

Task 10:
Create a shopping cart app with a list of items and an "Add to Cart" button for each. When the button is pressed, update a counter in the AppBar showing the total items in the cart.

Task 11:
Design a counter app with increment and decrement buttons. Use setState to update the displayed value as the buttons are pressed.

Task 12:
Make a toggle switch that changes the app’s background color when turned on or off. Use the
Switch widget to handle the toggle state.

---

4. Images and Media

Task 13:
Create an app with an Image widget displaying a picture from the network. Add a button below the image to change the image source when pressed.

Task 14:
Develop a photo gallery app using GridView to display multiple images from URLs. Include a loading indicator while images are loading.

Task 15:
Add an Image.asset widget to display an image from your local assets folder. Experiment with different BoxFit properties like cover, contain, and fill.

Task 16:
Build an image carousel that displays a different image every 3 seconds using PageView and an auto-slide feature.

---

5. Forms and Input Widgets

Task 17:
Create a registration form with fields for name, email, password, and phone number. Use validation to ensure email and password meet specific criteria.

Task 18:
Build a feedback form with TextField widgets for entering name and comments, and a DropdownButton for selecting a feedback category.

Task 19:
Design a search bar using TextField and display suggestions below as the user types. Filter suggestions based on input.

Task 20:
Make a simple task manager app with a form to add tasks. Use CheckboxListTile widgets to mark tasks as completed.

---

6. Navigation and Routing

Task 21:
Create a two-screen app. The first screen should have a button that navigates to the second screen. Display a welcome message on the second screen.

Task 22:
Develop a three-screen app with a home, details, and settings screen. Use Navigator.pushNamed for navigation, and pass data between screens.

Task 23:
Create a navigation drawer that allows switching between three different screens: Home, Profile, and Settings.

Task 24:
Design a bottom navigation bar with three tabs: News, Messages, and Profile. Change the displayed content based on the selected tab.

---

7. Lists and Scrolling

Task 25:
Build a to-do list app where users can add tasks. Use ListView.builder to display the list of tasks and allow each task to be removed with a swipe gesture.

Task 26:
Create an infinite scrolling list using ListView.builder that loads more data as the user scrolls to the bottom.

Task 27:
Implement a custom-styled list using ListTile widgets with leading icons, titles, and trailing icons (like a delete button).

Task 28:
Design a product listing page using ListView with horizontal scrolling, showing product images, names, and prices.

---

8. Advanced UI with Stacks and Positioned Widgets

Task 29:
Use a Stack widget to create an overlay effect, where a centered image has a partially transparent overlay with some text.

Task 30:
Build a basic profile page where the profile image is centered on the screen using Positioned inside a Stack, and other details (like name and bio) are displayed below.

Task 31:
Create a "card" UI with a floating action button positioned at the bottom right using Stack and Positioned.

Task 32:
Design a custom button with an icon positioned above the text. Use Stack to overlay the icon slightly on top of the text.

---

9. Animations and Transitions

Task 33:
Add a fade-in animation to an image using FadeInImage when it's loaded from a network source.

Task 34:
Create a button that, when pressed, expands or collapses a section of content using the AnimatedContainer widget.

Task 35:
Make a button with a pulsing effect using TweenAnimationBuilder to change the button size.

Task 36:
Implement a slide transition when navigating between two screens using the PageRouteBuilder with custom transition animations.

---

10. State Management with Provider

Task 37:
Build a counter app using the Provider package to manage and display the counter value across multiple widgets.

Task 38:
Create a shopping cart where the total price is updated in real-time using Provider as items are added or removed.

Task 39:
Develop a theme switcher using Provider that lets the user toggle between light and dark modes.

Task 40:
Implement a simple authentication flow with login and logout, where user status (logged in or out) is managed globally using Provider.

---

11. Custom Widgets

Task 41:
Create a custom RatingWidget that displays a series of stars and allows the user to select a rating from 1 to 5.

Task 42:
Build a custom ProgressBar widget that takes a percentage as input and displays a progress bar accordingly.

Task 43:
Create a custom ProfileCard widget that takes name, image, and bio as properties and displays them in a nicely styled card.

Task 44:
Develop a custom AvatarBadge widget that shows a user's avatar with an optional online/offline status indicator.

Create a profile screen layout using widgets such as Container, Column, Row, and Image. Build a list of cards displaying items in a product catalog (e.g., product name, price, and image). Design a custom button using a combination of Container, Padding, and Text widgets.

Module 4: Navigation and Routing

Theory Assignments:

1. Explain how the Navigator widget works in Flutter.
2. Describe the concept of named routes and their advantages over direct route navigation.
3. Explain how data can be passed between screens using route arguments.

Practical Assignments:

1. Build a multi-screen app that navigates between a home screen, a product list screen, and a details screen.
2. Implement a navigation drawer in an app with at least three screens.
3. Create a bottom navigation bar to switch between multiple pages (e.g., Home, Profile, Settings).

Module 5: State Management in Flutter

Theory Assignments:

1. Explain what state management is and why it's important in Flutter applications.
2. Compare the different types of state management solutions in Flutter, like Provider, Riverpod, and Bloc.
3. Describe the Provider package and how it differs from basic setState usage.

Practical Assignments:

1. Implement a counter app with two buttons for incrementing and decrementing a counter, using setState.
2. Use the Provider package to manage a shopping cart, with a screen to add items and view cart contents.
3. Create a to-do list app using the Riverpod package to manage and persist tasks.

Module 6: Working with Forms and User Input

Theory Assignments:

1. Explain the structure and purpose of forms in Flutter.
2. Describe how controllers and listeners are used to manage form input.
3. List some common form validation techniques and provide examples.

Practical Assignments:

1. Create a login form with fields for email and password, with basic validation.
2. Design a registration form with validation for fields like name, email, phone number, and password.
3. Implement a feedback form that includes dropdowns, checkboxes, and text input fields, with submission handling.

Module 7: Networking and API Integration

Theory Assignments:

1. Explain what a RESTful API is and its importance in mobile applications.
2. Describe how JSON data is parsed and used in Flutter.
3. Explain the purpose of HTTP methods (GET, POST, PUT, DELETE) and when to use each.

Practical Assignments:

1. Create a basic weather app that fetches weather data from a public API and displays it.
2. Implement a news feed app that retrieves and displays articles from an API.
3. Create a search app that fetches and displays movie information based on user input from an external API.

Module 8: Local Storage and Persistence

Theory Assignments:

1. Explain the difference between local storage options (shared_preferences, SQLite, Hive).
2. Describe CRUD operations and how they are implemented in SQLite or Hive.
3. Explain the advantages and use cases for shared_preferences.

Practical Assignments:

1. Create an app that uses shared_preferences to store and display user preferences.
2. Build a to-do list app using Hive or SQLite with functionality to add, update, delete, and view tasks.
3. Implement a simple notes app where notes are saved locally on the device.

Module 9: Animations and Transitions

Theory Assignments:

1. Explain the difference between implicit and explicit animations in Flutter.
2. Describe the purpose of AnimationController and its usage.
3. Explain the concept of Hero animations in Flutter.

Practical Assignments:

1. Create a button that animates size and color on press using implicit animations.
2. Implement a Hero animation that transitions an image smoothly between two screens.
3. Create a loading animation for data fetching in an app.

Module 10: Firebase Integration

Theory Assignments:

1. Explain the purpose of Firebase and list its core services.
2. Describe Firebase Authentication and its use cases in Flutter applications.
3. Explain how Firestore differs from traditional SQL databases.

Practical Assignments:

1. Implement a sign-up and login feature in a Flutter app using Firebase Authentication.
2. Create a chat application using Firebase Firestore.
3. Build an image gallery app that uploads images to Firebase Storage and displays them.

Module 11: App Deployment and Publishing

Theory Assignments:

1. Explain the app release process for both iOS and Android platforms.
2. Describe the steps involved in generating app bundles and APKs for deployment.
3. Outline the best practices for submitting apps to the App Store and Google Play.

Practical Assignments:

1. Create app icons and a splash screen for a sample Flutter app.
2. Prepare a Flutter app for release by creating an APK and testing it on a physical device.
3. Write the necessary steps to publish a Flutter app on Google Play Store.

Capstone Project

Theory Assignment:

1. Prepare a project plan and requirements document for your capstone project.

Practical Assignments:

1. Build a full-featured mobile application (e.g., task management app, e-commerce app, or social media feed) that utilizes all learned modules (Firebase, REST API, state management, local storage).
2. Write test cases and perform user testing to ensure smooth app functionality.
3. Optimize the app for performance and prepare it for publishing.