Python – Backend Assignment

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.

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.).

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).

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.

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.

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.

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()`.

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.

---

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.

---

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.

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 $100 and 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.

Module 13) Python Fundamentals

Introduction to Python **Theory:**

- Introduction to Python and its Features (simple, high-level, interpreted language).
- History and evolution of Python.
- Advantages of using Python over other programming languages.
- Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).
- Writing and executing your first Python program.

Lab:

- Write a Python program that prints "Hello, World!".
- Set up Python on your local machine and write a program to display your name.

---

2. Programming Style

Theory:

- Understanding Python's PEP 8 guidelines.
- Indentation, comments, and naming conventions in Python.
- Writing readable and maintainable code.

Lab:

- Write a Python program that demonstrates the correct use of indentation, comments, and variables following PEP 8 guidelines.

---

3. Core Python Concepts

Theory:

- Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.
- Python variables and memory allocation.
- Python operators: arithmetic, comparison, logical, bitwise.

Lab:

- Write a Python program to demonstrate the creation of variables and different data types.
- Practical Example 1: How does the Python code structure work?
- Practical Example 2: How to create variables in Python?

- Practical Example 3: How to take user input using the `input()` function.
- Practical Example 4: How to check the type of a variable dynamically using `type()`.

---

4. Conditional Statements

Theory:

- Introduction to conditional statements: `if`, `else`, `elif`.
- Nested `if-else` conditions.

Lab:

- Practical Example 5: Write a Python program to find greater and less than a number using `if_else`.
- Practical Example 6: Write a Python program to check if a number is prime using `if_else`.
- Practical Example 7: Write a Python program to calculate grades based on percentage using `if-else ladder`.
- Practical Example 8: Write a Python program to check if a person is eligible to donate blood using a nested `if`.

---

5. Looping (For, While)

Theory:

- Introduction to `for` and `while` loops.
- How loops work in Python.
- Using loops with collections (lists, tuples, etc.).

Lab:

- Practical Example 1: Write a Python program to print each fruit in a list using a simple for loop. `List1 = ['apple', 'banana', 'mango']`
- Practical Example 2: Write a Python program to find the length of each string in `List1`.
- Practical Example 3: Write a Python program to find a specific string in the list using a simple for loop and `if` condition.
- Practical Example 4: Print this pattern using nested for loop:

  ```
  markdown
  Copy code
  *
  **
  ***
  ****
  *****
  ```

---

6. Generators and Iterators

Theory:

- Understanding how generators work in Python.
- Difference between `yield` and `return`.
- Understanding iterators and creating custom iterators.

Lab:

- Write a generator function that generates the first 10 even numbers.
- Write a Python program that uses a custom iterator to iterate over a list of integers.

---

7. Functions and Methods

Theory:

- Defining and calling functions in Python.
- Function arguments (positional, keyword, default).
- Scope of variables in Python.
- Built-in methods for strings, lists, etc.

Lab:

- Practical Example: 1) Write a Python program to print "Hello" using a string.
- Practical Example: 2) Write a Python program to allocate a string to a variable and print it.
- Practical Example: 3) Write a Python program to print a string using triple quotes.
- Practical Example: 4) Write a Python program to access the first character of a string using index value.
- Practical Example: 5) Write a Python program to access the string from the second position onwards using slicing.
- Practical Example: 6) Write a Python program to access a string up to the fifth character.
- Practical Example: 7) Write a Python program to print the substring between index values 1 and 4.
- Practical Example: 8) Write a Python program to print a string from the last character.
- Practical Example: 9) Write a Python program to print every alternate character from the string starting from index 1.

---

8. Control Statements (Break, Continue, Pass)

Theory:

- Understanding the role of `break`, `continue`, and `pass` in Python loops.

Lab:

- Practical Example: 1) Write a Python program to skip 'banana' in a list using the `continue` statement. `List1 = ['apple', 'banana', 'mango']`
- Practical Example: 2) Write a Python program to stop the loop once 'banana' is found using the `break` statement.

---

9. String Manipulation

Theory:

- Understanding how to access and manipulate strings.
- Basic operations: concatenation, repetition, string methods (`upper()`, `lower()`, etc.).
- String slicing.

Lab:

- Write a Python program to demonstrate string slicing.
- Write a Python program that manipulates and prints strings using various string methods.

---

10. Advanced Python (map(), reduce(), filter(), Closures and Decorators)

Theory:

- How functional programming works in Python.
- Using `map()`, `reduce()`, and `filter()` functions for processing data.
- Introduction to closures and decorators.

Lab:

- Write a Python program to apply the `map()` function to square a list of numbers.
- Write a Python program that uses `reduce()` to find the product of a list of numbers.
- Write a Python program that filters out even numbers using the `filter()` function.

Assessment:

- Create a mini-project where students combine conditional statements, loops, and functions to create a basic Python application, such as a simple calculator or a grade management system.

Module 14) Python – Collections, functions and Modules

Accessing List

Theory:

- Understanding how to create and access elements in a list.
- Indexing in lists (positive and negative indexing).
- Slicing a list: accessing a range of elements.

Lab:

- Write a Python program to create a list with elements of multiple data types (integers, strings, floats, etc.).
- Write a Python program to access elements at different index positions.

Practical Examples:

1. Write a Python program to create a list of multiple data type elements.
2. Write a Python program to find the length of a list using the `len()` function.

---

2. List Operations

Theory:

- Common list operations: concatenation, repetition, membership.
- Understanding list methods like `append()`, `insert()`, `remove()`, `pop()`.

Lab:

- Write a Python program to add elements to a list using `insert()` and `append()`.
- Write a Python program to remove elements from a list using `pop()` and `remove()`.

Practical Examples: 3) Write a Python program to update a list using `insert()` and `append()`. 4) Write a Python program to remove elements from a list using `pop()` and `remove()`.

---

3. Working with Lists

Theory:

- Iterating over a list using loops.
- Sorting and reversing a list using `sort()`, `sorted()`, and `reverse()`.
- Basic list manipulations: addition, deletion, updating, and slicing.

Lab:

- Write a Python program to iterate over a list using a `for` loop.
- Write a Python program to sort a list using both `sort()` and `sorted()`.

Practical Examples: 5) Write a Python program to iterate through a list and print each element. 6) Write a Python program to insert elements into an empty list using a `for` loop and `append()`.

---

4. Tuple

Theory:

- Introduction to tuples, immutability.
- Creating and accessing elements in a tuple.
- Basic operations with tuples: concatenation, repetition, membership.

Lab:

- Write a Python program to create a tuple with multiple data types.
- Write a Python program to concatenate two tuples.

Practical Examples: 7) Write a Python program to convert a list into a tuple. 8) Write a Python program to create a tuple with multiple data types. 9) Write a Python program to concatenate two tuples into one. 10) Write a Python program to access the value of the first index in a tuple.

---

5. Accessing Tuples

Theory:

- Accessing tuple elements using positive and negative indexing.
- Slicing a tuple to access ranges of elements.

Lab:

- Write a Python program to access values between index 1 and 5 in a tuple.
- Write a Python program to access alternate values between index 1 and 5 in a tuple.

Practical Examples: 11) Write a Python program to access values between index 1 and 5 in a tuple. 12) Write a Python program to access the value from the last index in a tuple.

---

6. Dictionaries

Theory:

- Introduction to dictionaries: key-value pairs.
- Accessing, adding, updating, and deleting dictionary elements.
- Dictionary methods like `keys()`, `values()`, and `items()`.

Lab:

- Write a Python program to create a dictionary with 6 key-value pairs.
- Write a Python program to access values using dictionary keys.

Practical Examples: 13) Write a Python program to create a dictionary of 6 key-value pairs. 14) Write a Python program to access values using keys from a dictionary.

---

7. Working with Dictionaries

Theory:

- Iterating over a dictionary using loops.
- Merging two lists into a dictionary using loops or `zip()`.
- Counting occurrences of characters in a string using dictionaries.

Lab:

- Write a Python program to update a value in a dictionary.
- Write a Python program to merge two lists into one dictionary using a loop.

Practical Examples: 15) Write a Python program to update a value at a particular key in a dictionary. 16) Write a Python program to separate keys and values from a dictionary using `keys()` and `values()` methods. 17) Write a Python program to convert two lists into one dictionary using a `for` loop. 18) Write a Python program to count how many times each character appears in a string.

---

8. Functions

Theory:

- Defining functions in Python.
- Different types of functions: with/without parameters, with/without return values.
- Anonymous functions (lambda functions).

Lab:

- Write a Python program to create a function that takes a string as input and prints it.
- Write a Python program to create a calculator using functions.

Practical Examples: 19) Write a Python program to print a string using a function. 20) Write a Python program to create a parameterized function that takes two arguments and prints their sum. 21) Write a Python program to create a lambda function with one expression. 22) Write a Python program to create a lambda function with two expressions.

---

9. Modules

Theory:

- Introduction to Python modules and importing modules.
- Standard library modules: math, random.
- Creating custom modules.

Lab:

- Write a Python program to import the `math` module and use functions like `sqrt()`, `ceil()`, `floor()`.
- Write a Python program to generate random numbers using the `random` module.

Practical Examples: 23) Write a Python program to demonstrate the use of functions from the `math` module. 24) Write a Python program to generate random numbers between 1 and 100 using the `random` module.

Module 15) Advance Python Programming

1. Printing on Screen

Theory:

- Introduction to the `print()` function in Python.
- Formatting outputs using `f-strings` and `format()`.

Lab:

- Write a Python program to print a formatted string using `print()` and `f-string`.

Practical Example:

1. Write a Python program to print “Hello, World!” on the screen.

---

2. Reading Data from Keyboard

Theory:

- Using the `input()` function to read user input from the keyboard.
- Converting user input into different data types (e.g., int, float, etc.).

Lab:

- Write a Python program to read a name and age from the user and print a formatted output.

Practical Example: 2) Write a Python program to read a string, an integer, and a float from the keyboard and display them.

---

3. Opening and Closing Files

Theory:

- Opening files in different modes (`'r'`, `'w'`, `'a'`, `'r+'`, `'w+'`).
- Using the `open()` function to create and access files.
- Closing files using `close()`.

Lab:

- Write a Python program to open a file in write mode, write some text, and then close it.

Practical Example: 3) Write a Python program to create a file and write a string into it.

4. Reading and Writing Files

Theory:

- Reading from a file using `read()`, `readline()`, `readlines()`.
- Writing to a file using `write()` and `writelines()`.

Lab:

- Write a Python program to read the contents of a file and print them on the console.
- Write a Python program to write multiple strings into a file.

Practical Examples: 4) Write a Python program to create a file and print the string into the file. 5) Write a Python program to read a file and print the data on the console. 6) Write a Python program to check the current position of the file cursor using `tell()`.

5. Exception Handling

Theory:

- Introduction to exceptions and how to handle them using `try`, `except`, and `finally`.
- Understanding multiple exceptions and custom exceptions.

Lab:

- Write a Python program to handle exceptions in a simple calculator (division by zero, invalid input).
- Write a Python program to demonstrate handling multiple exceptions.

Practical Examples: 7) Write a Python program to handle exceptions in a calculator. 8) Write a Python program to handle multiple exceptions (e.g., file not found, division by zero). 9) Write a Python program to handle file exceptions and use the `finally` block for closing the file. 10) Write a Python program to print custom exceptions.

6. Class and Object (OOP Concepts)

Theory:

- Understanding the concepts of classes, objects, attributes, and methods in Python.
- Difference between local and global variables.

Lab:

- Write a Python program to create a class and access its properties using an object.

Practical Examples: 11) Write a Python program to create a class and access the properties of the class using an object. 12) Write a Python program to demonstrate the use of local and global variables in a class.

---

7. Inheritance

Theory:

- Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.
- Using the `super()` function to access properties of the parent class.

Lab:

- Write Python programs to demonstrate different types of inheritance (single, multiple, multilevel, etc.).

Practical Examples: 13) Write a Python program to show single inheritance. 14) Write a Python program to show multilevel inheritance. 15) Write a Python program to show multiple inheritance. 16) Write a Python program to show hierarchical inheritance. 17) Write a Python program to show hybrid inheritance. 18) Write a Python program to demonstrate the use of `super()` in inheritance.

---

8. Method Overloading and Overriding

Theory:

- Method overloading: defining multiple methods with the same name but different parameters.
- Method overriding: redefining a parent class method in the child class.

Lab:

- Write Python programs to demonstrate method overloading and method overriding.

Practical Examples: 19) Write a Python program to show method overloading. 20) Write a Python program to show method overriding.

---

9. SQLite3 and PyMySQL (Database Connectors)

Theory:

- Introduction to SQLite3 and PyMySQL for database connectivity.
- Creating and executing SQL queries from Python using these connectors.

Lab:

- Write a Python program to connect to an SQLite3 database, create a table, insert data, and fetch data.

Practical Examples: 21) Write a Python program to create a database and a table using SQLite3. 22) Write a Python program to insert data into an SQLite3 database and fetch it.

---

10. Search and Match Functions

Theory:

- Using `re.search()` and `re.match()` functions in Python's `re` module for pattern matching.
- Difference between search and match.

Lab:

- Write a Python program to search for a word in a string using `re.search()`.
- Write a Python program to match a word in a string using `re.match()`.

Practical Examples: 23) Write a Python program to search for a word in a string using `re.search()`. 24) Write a Python program to match a word in a string using `re.match()`.

Module 16) Python DB and Framework

1. HTML in Python

Theory:

- Introduction to embedding HTML within Python using web frameworks like Django or Flask.
- Generating dynamic HTML content using Django templates.

Lab:

- Write a Python program to render an HTML file using Django's template system.

Practical Example:

1. Write a Django project that renders an HTML file displaying "Welcome to Doctor Finder" on the home page.

---

2. CSS in Python

Theory:

- Integrating CSS with Django templates.
- How to serve static files (like CSS, JavaScript) in Django.

Lab:

- Create a CSS file to style a basic HTML template in Django.

Practical Example: 2) Write a Django project to display a webpage with custom CSS styling for a doctor profile page.

---

3. JavaScript with Python

Theory:

- Using JavaScript for client-side interactivity in Django templates.
- Linking external or internal JavaScript files in Django.

Lab:

- Create a Django project with JavaScript-enabled form validation.

Practical Example: 3) Write a Django project where JavaScript is used to validate a patient registration form on the client side.

---

4. Django Introduction

Theory:

- Overview of Django: Web development framework.
- Advantages of Django (e.g., scalability, security).
- Django vs. Flask comparison: Which to choose and why.

Lab:

- Write a short project using Django's built-in tools to render a simple webpage.

Practical Example: 4) Write a Python program to create a Django project and understand its directory structure.

---

5. Virtual Environment

Theory:

- Understanding the importance of a virtual environment in Python projects.
- Using `venv` or `virtualenv` to create isolated environments.

Lab:

- Set up a virtual environment for a Django project.

Practical Example: 5) Write a Python program to create and activate a virtual environment, then install Django in it.

---

6. Project and App Creation

Theory:

- Steps to create a Django project and individual apps within the project.
- Understanding the role of `manage.py`, `urls.py`, and `views.py`.

Lab:

- Create a Django project with an app to manage doctor profiles.

Practical Example: 6) Write a Python program to create a Django project and a new app within the project called `doctor`.

---

7. MVT Pattern Architecture

Theory:

- Django's MVT (Model-View-Template) architecture and how it handles request-response cycles.

Lab:

- Build a simple Django app showcasing how the MVT architecture works.

Practical Example: 7) Write a Django project with models, views, and templates to display doctor information.

---

8. Django Admin Panel

Theory:

- Introduction to Django's built-in admin panel.
- Customizing the Django admin interface to manage database records.

Lab:

- Set up and customize the Django admin panel to manage a "Doctor Finder" project.

Practical Example: 8) Write a Django project to create an admin panel and add custom fields for managing doctor information.

---

9. URL Patterns and Template Integration

Theory:

- Setting up URL patterns in `urls.py` for routing requests to views.
- Integrating templates with views to render dynamic HTML content.

Lab:

- Create a Django project with URL patterns and corresponding views and templates.

Practical Example: 9) Write a Django project where URL routing is used to navigate between different pages of a "Doctor Finder" site (home, profile, contact).

---

10. Form Validation using JavaScript

Theory:

- Using JavaScript for front-end form validation.

Lab:

- Write a Django project to implement JavaScript form validation for a user registration form.

Practical Example: 10) Write a Django project that uses JavaScript to validate fields like email and phone number in a registration form.

---

11. Django Database Connectivity (MySQL or SQLite)

Theory:

- Connecting Django to a database (SQLite or MySQL).
- Using the Django ORM for database queries.

Lab:

- Set up database connectivity for a Django project.

Practical Example: 11) Write a Django project to connect to an SQLite/MySQL database and manage doctor records.

---

12. ORM and QuerySets

Theory:

- Understanding Django's ORM and how QuerySets are used to interact with the database.

Lab:

- Perform CRUD operations using Django ORM.

Practical Example: 12) Write a Django project that demonstrates CRUD operations (Create, Read, Update, Delete) on doctor profiles using Django ORM.

13. Django Forms and Authentication

Theory:

- Using Django’s built-in form handling.
- Implementing Django’s authentication system (sign up, login, logout, password management).

Lab:

- Create a Django project for user registration and login functionality.

Practical Example: 13) Write a Django project to handle user sign up, login, password reset, and profile updates.

14. CRUD Operations using AJAX

Theory:

- Using AJAX for making asynchronous requests to the server without reloading the page.

Lab:

- Implement AJAX in a Django project for performing CRUD operations.

Practical Example: 14) Write a Django project that uses AJAX to add, edit, or delete doctor profiles without refreshing the page.

15. Customizing the Django Admin Panel

Theory:

- Techniques for customizing the Django admin panel.

Lab:

- Customize the Django admin panel for better management of records.

Practical Example: 15) Write a Django project that customizes the admin panel to display more detailed doctor information (e.g., specialties, availability).

16. Payment Integration Using Paytm

Theory:

- Introduction to integrating payment gateways (like Paytm) in Django projects.

Lab:

- Implement Paytm payment gateway in a Django project.

Practical Example: 16) Write a Django project that integrates Paytm for handling payments in the "Doctor Finder" project.

---

17. GitHub Project Deployment

Theory:

- Steps to push a Django project to GitHub.

Lab:

- Deploy a Django project to GitHub for version control.

Practical Example: 17) Write a step-by-step guide to deploying the “Doctor Finder” project to GitHub.

---

18. Live Project Deployment (PythonAnywhere)

Theory:

- Introduction to deploying Django projects to live servers like PythonAnywhere.

Lab:

- Deploy a Django project to PythonAnywhere.

Practical Example: 18) Write a Django project and deploy it on PythonAnywhere, making it accessible online.

---

19. Social Authentication

Theory:

- Setting up social login options (Google, Facebook, GitHub) in Django using OAuth2.

Lab:

- Implement Google and Facebook login for the Django project.

Practical Example: 19) Write a Django project to allow users to log in using Google or Facebook.

---

20. Google Maps API

Theory:

- Integrating Google Maps API into Django projects.

Lab:

- Use Google Maps API to display doctor locations in the "Doctor Finder" project.

Practical Example: 20) Write a Django project to display doctor locations using Google Maps API.

Module 17) Rest Framework

Introduction to APIs

Theory:

- What is an API (Application Programming Interface)?
- Types of APIs: REST, SOAP.
- Why are APIs important in web development?

Lab:

- Write a Python program that consumes a simple public API (e.g., a joke API).

Practical Example:

1. Write a Python script to fetch a random joke from an API and display it on the console.

---

2. Requirements for Web Development Projects

Theory:

- Understanding project requirements.
- Setting up the environment and installing necessary packages.

Lab:

- Write a requirements.txt file for a Django project that includes all necessary dependencies.

Practical Example: 2) Write a Python script to set up a Django project and install packages like `django`, `djangorestframework`, `requests`, etc.

---

3. Serialization in Django REST Framework

Theory:

- What is Serialization?
- Converting Django QuerySets to JSON.
- Using `serializers` in Django REST Framework (DRF).

Lab:

- Create a Django REST API with serialization for a `Doctor` model.

Practical Example: 3) Write a Django REST API to serialize a `Doctor` model with fields like name, specialty, and contact details.

---

4. Requests and Responses in Django REST Framework

Theory:

- HTTP request methods (GET, POST, PUT, DELETE).
- Sending and receiving responses in DRF.

Lab:

- Create a Django REST API that accepts POST requests to add new doctor profiles.

Practical Example: 4) Write a Django project where the API accepts a POST request to add a doctor’s details to the database.

---

5. Views in Django REST Framework

Theory:

- Understanding views in DRF: Function-based views vs Class-based views.

Lab:

- Implement a class-based view in DRF for managing doctor profiles.

Practical Example: 5) Write a Django project that implements a class-based view to handle doctor profile creation, reading, updating, and deletion (CRUD operations).

---

6. URL Routing in Django REST Framework

Theory:

- Defining URLs and linking them to views.

Lab:

- Set up URL routing in a Django project to link to CRUD API endpoints for doctors.

Practical Example: 6) Write a Django project that routes URLs to the views handling doctor CRUD operations (`/doctors`, `/doctors/<id>`).

---

7. Pagination in Django REST Framework

Theory:

- Adding pagination to APIs to handle large data sets.

Lab:

- Implement pagination in a Django REST API for fetching doctor profiles.

Practical Example: 7) Write a Django API that returns paginated results for a list of doctors.

---

8. Settings Configuration in Django

Theory:

- Configuring Django settings for database, static files, and API keys.

Lab:

- Modify settings.py to connect Django to a MySQL or SQLite database.

Practical Example: 8) Write a Django project that connects to an SQLite database and stores doctor profiles.

---

9. Project Setup

Theory:

- Setting up a Django REST Framework project.

Lab:

- Create a new Django project and app for managing doctor profiles.

Practical Example: 9) Write a Django project to set up a new app called `doctor_finder` and create models, serializers, and views.

---

10. Social Authentication, Email, and OTP Sending API

Theory:

- Implementing social authentication (e.g., Google, Facebook) in Django.
- Sending emails and OTPs using third-party APIs like Twilio, SendGrid.

Lab:

- Add Google login to a Django project using `django-allauth`.

Practical Example: 10) Write a Django project that integrates Google login and sends OTPs to users using Twilio.

---

11. RESTful API Design

Theory:

- REST principles: statelessness, resource-based URLs, and using HTTP methods for CRUD operations.

Lab:

- Design a REST API for managing doctor profiles using Django REST Framework.

Practical Example: 11) Write a Django REST API with endpoints for creating, reading, updating, and deleting doctors.

---

12. CRUD API (Create, Read, Update, Delete)

Theory:

- What is CRUD, and why is it fundamental to backend development?

Lab:

- Implement a CRUD API using Django REST Framework for doctor profiles.

Practical Example: 12) Write a Django project that allows users to create, read, update, and delete doctor profiles using API endpoints.

---

13. Authentication and Authorization API

Theory:

- Difference between authentication and authorization.
- Implementing authentication using Django REST Framework's token-based system.

Lab:

- Implement user login, logout, and registration APIs in a Django project.

Practical Example: 13) Write a Django project that uses token-based authentication for users and restricts access to certain API endpoints.

---

14. OpenWeatherMap API Integration

Theory:

- Introduction to OpenWeatherMap API and how to retrieve weather data.

Lab:

- Create a Django project that fetches weather data for a given location.

Practical Example: 14) Write a Django project to fetch current weather data for a location using the OpenWeatherMap API.

---

15. Google Maps Geocoding API

Theory:

- Using Google Maps Geocoding API to convert addresses into coordinates.

Lab:

- Create a Django project that takes an address as input and returns the latitude and longitude.

Practical Example: 15) Write a Django project that uses Google Maps API to find the coordinates of a given address.

---

16. GitHub API Integration

Theory:

- Introduction to GitHub API and how to interact with repositories, pull requests, and issues.

Lab:

- Use GitHub API to create a repository and retrieve user data.

Practical Example: 16) Write a Django project that interacts with the GitHub API to create a new repository and list all repositories for a given user.

---

17. Twitter API Integration

Theory:

- Using Twitter API to fetch and post tweets, and retrieve user data.

Lab:

- Create a Django project that fetches recent tweets of a specific user.

Practical Example: 17) Write a Django project to fetch and display the latest 5 tweets from a Twitter user using the Twitter API.

---

18. REST Countries API Integration

Theory:

- Introduction to REST Countries API and how to retrieve country-specific data.

Lab:

- Use REST Countries API to fetch data for a specific country.

Practical Example: 18) Write a Django project that displays details (population, language, currency) of a country entered by the user using the REST Countries API.

---

19. Email Sending APIs (SendGrid, Mailchimp)

Theory:

- Using email sending APIs like SendGrid and Mailchimp to send transactional emails.

Lab:

- Implement email sending functionality in a Django project using SendGrid.

Practical Example: 19) Write a Django project to send a confirmation email to a user using the SendGrid API after successful registration.

---

20. SMS Sending APIs (Twilio)

Theory:

- Introduction to Twilio API for sending SMS and OTPs.

Lab:

- Use Twilio API to send OTP to a user’s phone.

Practical Example: 20) Write a Django project that sends an OTP to the user's mobile number during registration using Twilio API.

---

21. Payment Integration (PayPal, Stripe)

Theory:

- Introduction to integrating payment gateways like PayPal and Stripe.

Lab:

- Add Stripe payment functionality to a Django project.

Practical Example: 21) Write a Django project to allow users to make payments via Stripe for booking doctor appointments.

---

22. Google Maps API Integration

Theory:

- Using Google Maps API to display maps and calculate distances between locations.

Lab:

- Use Google Maps API to display doctor locations on a map.

Practical Example: 23) Write a Django project that integrates Google Maps API to show doctor locations in a specific city.

Module 20) Debugging and Problem Solving with Python

Assignment 1: Syntax Errors

Objective:

Identify and fix basic syntax errors in the Python program.

Problem:

The following Python program should calculate the sum of two numbers, but it contains several syntax errors. Fix the errors and ensure the program runs correctly.

```
python
Copy code
def addNumbers(a, b):
    result = a + b
    return result

number1 = input("Enter the first number:")
number2 = input("Enter the second number:")
sum = addNumbers(number1 number2)
print "The sum is:", sum
```

Task:

- Identify the syntax errors and correct them.
- Explain the corrected issues (e.g., missing commas, incorrect function calls, etc.).

Assignment 2: Logical Errors

Objective:

Understand and resolve logical errors in Python code.

Problem:

The following Python program is intended to check if a number is even or odd, but it always prints that the number is even, even when it is odd. Debug the logical error.

```
python
Copy code
def check even odd(num):
    if num % 2 = 0:
        print("The number is even.")
    else:
        print("The number is odd.")

number = int(input("Enter a number: "))
check_even_odd(number)
```

Task:

- Correct the logical mistake.
- Describe how this type of error impacts the flow of the program.

Assignment 3: Index and Range Errors

Objective:

Fix indexing errors and prevent out-of-range issues in lists.

Problem:

The following code attempts to access and print the first three elements of a list, but it raises an `IndexError`. Identify and correct the problem.

```
python
Copy code
my_list = [10, 20]
print(my_list[0])
print(my_list[1])
print(my_list[2])  # This line causes an error
```

Task:

- Identify the issue and provide a solution to prevent `IndexError`.
- Write an explanation of list indices and how out-of-range access can be avoided.

Assignment 4: Type Errors

Objective:

Debug and resolve type mismatch errors in Python programs.

Problem:

The following program tries to concatenate a string and an integer, but it raises a `TypeError`. Fix the error and make the program functional.

```
python
Copy code
name = input("Enter your name: ")
age = int(input("Enter your age: "))
greeting = "Hello, " + name + ". You are " + age + " years old."
print(greeting)
```

Task:

- Debug the type error in the code and provide a solution.
- Explain why Python throws a `TypeError` when trying to concatenate a string and an integer.

Assignment 5: Infinite Loops

Objective:

Fix an infinite loop issue and understand how control flow works in loops.

Problem:

The following code is supposed to print numbers from 1 to 10, but it never stops. Identify the cause and fix the infinite loop.

```
python
Copy code
counter = 1
while counter <= 10:
    print(counter)
```

Task:

- Debug the infinite loop and correct it.
- Explain the importance of controlling loop conditions to avoid infinite loops.

Assignment 6: Off-by-One Errors

Objective:

Understand and fix off-by-one errors in loops.

Problem:

The following code is designed to print numbers from 1 to 5, but it only prints numbers from 1 to 4. Debug and fix the issue.

```
python
Copy code
for i in range(1, 5):
    print(i)
```

Task:

- Identify the off-by-one error and correct the code.
- Explain how Python's `range()` function works and how to properly specify loop ranges.

Assignment 7: Uninitialized Variables

Objective:

Debug and correct uninitialized variable issues in Python programs.

Problem:

The following code attempts to print a variable `result` without initializing it first. Identify and fix the issue.

```
python
Copy code
def multiply(a, b):
    result = a * b

multiply(5, 10)
print(result)   # This line causes an error
```

Task:

- Debug the code and ensure that the variable `result` is initialized correctly.
- Explain the importance of variable initialization in Python.

Assignment 8: Function Call Errors

Objective:

Debug issues related to incorrect function calls and argument passing.

Problem:

The following program is meant to calculate the area of a rectangle, but it produces an error due to incorrect function arguments. Fix the function call error.

```
python
Copy code
def calculate_area(length, width):
    return length * width

l = 10
area = calculate_area(l)  # Missing second argument
```

```
print("The area is:", area)
```

Task:

- Debug the error and fix the function call by passing the correct number of arguments.
- Explain the concept of function arguments and how Python handles function calls.

Assignment 9: Scope and Variable Shadowing

Objective:

Understand and fix scope-related issues in Python.

Problem:

The following code attempts to modify a global variable inside a function, but the changes are not reflected outside the function. Debug the issue related to variable scope.

```
python
Copy code
counter = 0

def increment_counter():
    counter += 1

increment_counter()
print("Counter:", counter)
```

Task:

- Debug the issue related to variable scope and explain how Python handles variable shadowing.
- Modify the code so that the function can modify the global variable correctly.

Assignment 10: Debugging with `try-except` Blocks

Objective:

Use `try-except` blocks to handle exceptions and debug runtime errors.

Problem:

The following code prompts the user to input two numbers and divides them, but it raises a `ZeroDivisionError` when the second number is zero. Handle the exception using a `try-except` block.

```
python
Copy code
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
result = num1 / num2
print("Result:", result)
```

Task:

- Use `try-except` blocks to handle the `ZeroDivisionError` and provide a meaningful error message to the user.
- Explain how exception handling improves the robustness of Python programs.

Assignment 11: File Handling Errors

Objective:

Debug and handle file-related errors in Python programs.

Problem:

The following code tries to open a file that may not exist, causing a `FileNotFoundError`. Handle this error and ensure the program works correctly.

```
python
Copy code
file = open("data.txt", "r")
content = file.read()
print(content)
file.close()
```

Task:

- Use `try-except` to handle file opening errors and provide a fallback message when the file does not exist.
- Explain the importance of exception handling when working with files.

Assignment 12: Logic Errors in Algorithms

Objective:

Identify and correct logic errors in simple algorithms.

Problem:

The following code is supposed to find the maximum number in a list, but it always prints the first number as the maximum. Debug the issue.

```
python
Copy code
def find_max(numbers):
    max_num = numbers[0]
    for num in numbers:
        if num > max_num:
            max_num = num
    return max_num

nums = [3, 7, 2, 8, 4]
print("Maximum number is:", find_max(nums))
```

Task:

- Debug the logic error and fix the issue.
- Explain how comparison works in Python and why the current logic fails.