Python – Short Term Assignment

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.