Day 1 Assignment

Type 1 : Easy Questions:

Task 1: Variables and Data Types

Write a Python script that does the following:

- Create variables of different data types: integer, float, string, and boolean.
- Print each variable with an appropriate message indicating its data type.
- Use the type() function to verify and print the data type of each variable.

Task 2: String Manipulation

Given the string "Python programming is fun!", write a Python script that:

- Converts the string to uppercase.
- Replaces the word "fun" with "powerful".
- Slices the string to print only the word "programming".
- Prints the length of the original string.

Task 3: Lists and Loops

Write a Python program that:

- Creates a list of at least 5 integers.
- Uses a for loop to iterate through the list and print each number.
- Uses a while loop to find the sum of all numbers in the list.

Task 4: Conditionals

Write a Python program that:

- Asks the user for their age using input().

- Checks if the age is greater than or equal to 18 and prints a message based on whether the user is an adult or not.

- Add error handling to ensure the user enters a valid number.

Task 5: Functions

Write a Python function calculate_area() that:

- Takes two parameters: length and width (both integers or floats).

- Returns the area of a rectangle (length × width).

- Call the function with different sets of parameters and print the results.

Task 6: Dictionaries

Write a Python program that:

- Creates a dictionary containing at least 3 key-value pairs where the keys are student names, and the values are their scores.

- Adds a new key-value pair to the dictionary for another student.

- Updates the score of one student.

- Prints out all the student names and their scores.

Task 7: File Handling

Write a Python program that:

- Creates a text file named students.txt.

- Writes the names of five students into the file, each on a new line.

- Reads the file and prints its contents to the console.

Task 8: Error Handling

Write a Python program that:

- Attempts to divide two numbers.
- Handles any ZeroDivisionError by printing an error message.
- Handles any other exceptions using a generic except block and prints an appropriate message.

Type 2 : Moderate Questions:

Task 1: Lambda Functions and Built-in Functions

Write a Python program that:

- Creates a list of tuples representing the names and ages of 5 people.
- Sorts the list of tuples based on age using a lambda function.
- Uses the map() function to create a list of only the names from the sorted list.
- Uses the filter() function to return a list of people whose age is greater than or equal to 30.
- Print the final lists.

Task 2: List Comprehensions and Nested Loops

Write a Python program that:

- Uses a list comprehension to create a list of all even numbers from 1 to 100.
- Uses a list comprehension with a nested loop to create a multiplication table (i.e., a list of lists) for the numbers 1 to 5.
- Print both the even numbers list and the multiplication table.

Task 3: OOP - Classes and Inheritance

Write a Python program that:

- Defines a class Person with attributes for name and age, and a method greet() that prints a greeting with the person's name.
- Defines a subclass Employee that inherits from Person, adds an additional attribute employee_id,

and overrides the greet() method to include the employee ID in the greeting.

- Create objects of both Person and Employee and call their respective greet() methods.

Task 4: Generators

Write a Python program that:

- Implements a generator fibonacci_generator(n) that generates the first n Fibonacci numbers.
- Use the generator to print the first 10 Fibonacci numbers.
- Explain the benefits of using generators over lists for large data.

Task 5: Exception Handling with Custom Exceptions

Write a Python program that:

- Defines a custom exception class InvalidAgeError that is raised when an invalid age (less than 0 or greater than 150) is entered.
- Write a function validate_age() that takes an age as input, raises InvalidAgeError for invalid ages, and prints a message if the age is valid.
- Handle the exception with a try-except block and test with both valid and invalid age inputs.

Task 6: Regular Expressions

Write a Python program that:

- Prompts the user to input an email address.
- Uses a regular expression to validate if the entered email address follows the format: [username]@[domain].[extension] (e.g., user@example.com).
- Print a message indicating whether the email is valid or not.
- Use the re module to implement the regular expression.

Task 7: Working with JSON

Write a Python program that:

- Creates a dictionary with details about a book, including title, author, and year published.

- Converts the dictionary to a JSON string using the json module and writes it to a file named book.json.

- Reads the JSON string back from the file and converts it back to a Python dictionary.

- Print both the JSON string and the final dictionary.

Task 8: Threading

Write a Python program that:

- Defines a function print_numbers() that prints numbers from 1 to 10, with a 1-second delay between each number.

- Creates two threads that run the print_numbers() function concurrently.

- Print a message indicating when both threads have finished executing.

- Use the threading module to implement this task.

Task 9: Command-line Arguments

Write a Python program that:

- Accepts two numbers as command-line arguments.

- Multiplies the two numbers and prints the result.

- Use the sys.argv list to get the command-line arguments.

Type 3 : Challenging Questions:

Problem Statement: Student Management System

Objective:

Create a Python program that manages a list of students and their grades. The program should allow the user to add new students, update student grades, calculate statistics, and store the data in a file. The system should be interactive and offer the user a menu of options to choose from.

Requirements:

1. Menu System:

- The program should display a menu of options that the user can choose from:
  * Add a new student
  * Update student grade
  * View all students
  * Calculate class average
  * Find the highest and lowest grades
  * Save student data to a file
  * Load student data from a file
  * Exit the program

2. Student Data:

Each student will have the following information:

- Name (string)
- Age (integer)
- Grade (float)

Store the student data in a dictionary where the key is the student's name and the value is a dictionary containing their age and grade.

Features to Implement:

A. Adding a New Student

- Prompt the user to input the student's name, age, and grade.

- Add the student to the dictionary with their corresponding data.

- Handle errors if the user enters invalid data types (e.g., non-numeric age or grade).

B. Updating a Student's Grade

- Ask the user for the student's name.

- If the student exists, prompt the user to enter the new grade and update the student's record.

- If the student does not exist, display an appropriate message.

C. Viewing All Students

- Print a formatted list of all students with their name, age, and grade.

- If there are no students, display a message indicating the list is empty.

D. Calculating Class Average

- Calculate and display the average grade of all students.

- Handle the case where there are no students by displaying an appropriate message.

E. Finding the Highest and Lowest Grades

- Identify the students with the highest and lowest grades.

- Display their names and grades.

F. Saving Data to a File

- Save the student data to a text file (students.txt).

- Each line in the file should contain a student's name, age, and grade, separated by commas.

G. Loading Data from a File

- Load student data from the students.txt file.
- Ensure that the data is properly parsed and stored in the dictionary.

H. Error Handling

- Use try-except blocks to handle potential errors, such as file not found errors, invalid inputs, or division by zero when calculating averages.

I. Exit the Program

- Allow the user to exit the program by selecting the exit option from the menu.