Data Analytics with Python

Assignment 2: Intermediate Python Programming

Objective:

By the end of this assignment, you will deepen yout understanding of Python functions, modules, packages, advanced data structures, and gain exposure to object-oriented programming concepts.

Task 1: Advanced Function Usage

Instructions:

- Define a function calculate_average(numbers) that takes a list of numbers as an argument and returns the average.
- Create a list scores = [75, 88, 92, 85, 90] and use the calculate_average function to find the average score. Print the result.
- Define a function greet_user(name, greeting="Hello") that takes a user's name and an optional greeting message. If no greeting is provided, it should default to "Hello". Test this function by greeting the user "John" with and without a custom greeting.

Expected Output:

- Average score calculated and displayed.
- Greeting messages displayed.

Task 2: Using Modules and Packages

Instructions:

- Import the math module and use it to calculate and print the square root of 144.
- Import the statistics module and use it to calculate and print the median of the scores list from Task 1.
- Create a custom module named custom_math.py with a function square(number) that returns the square of a number. Import this module in your main script and use the square function.

Expected Output:

- Square root of 144 displayed.
- Median of the scores displayed.
- Square of a number calculated using the custom module and displayed.

Task 3: Working with Built-in Modules and Libraries in Python

Instructions:

- Use the datetime module to print today's date and time.
- Use the random module to generate and print a random number between 1 and 100.
- Create a list items = ['apple', 'banana', 'cherry', 'date'] and use the random.choice() function to select and print a random item from the list.

Expected Output:

- Current date and time displayed.
- Random number displayed.
- Random item selected and displayed.

Task 4: Exploring Object-Oriented Programming (OOP)

Instructions:

- Define a class Employee with attributes name, position, and salary.
- Add a method display_info() to the class that prints the employee's information.
- Create an object employee1 of the Employee class and assign the values "Jane Doe", "Engineer", and 75000 to name, position, and salary respectively.
- Call the display_info() method to print the employee's details.

Expected Output:

- Employee details displayed.

Task 5: Reading and Writing Data Files in Python (Revisited)

Instructions:

- Write a list services = ["Voice", "Data", "SMS", "MMS"] to a file named services.txt, with each service on a new line.
- Read the content of services.txt and store it in a list called read_services. Print the list to verify the content.

Expected Output:

- List of services written to the file.
- List read from the file and displayed.

Task 6: More Control Structures and Loops

Instructions:

- Write a program that iterates over the scores list (from Task 1) and prints whether each score is above or below 80.
- Create a for loop that iterates over the numbers 1 to 10 and prints only the even numbers.
- Write a nested loop that generates a multiplication table (from 1 to 5) and prints the results.

Expected Output:

- Comparison of scores to 80 displayed.
- Even numbers from 1 to 10 displayed.
- Multiplication table displayed.

Task 7: Working with Complex Data Structures

Instructions:

- Create a dictionary employees = {"John": {"position": "Manager", "salary": 80000}, "Jane": {"position": "Engineer", "salary": 75000}}.
- Add a new employee "Doe": {"position": "Analyst", "salary": 65000} to the dictionary.
- Write a loop that prints each employee's name along with their position and salary.

Expected Output:

- Employee details with position and salary displayed.