Lab 1: Applied Data Workflows and Modular Programming

Objective

1. Set up a Python environment for applied data workflows.
2. Write modular programs using functions and object-oriented programming (OOP).
3. Practice applied data handling, analysis, and visualization using Python.

Part 1: Setting Up Python Environment

Step 1: Install Python

- Ensure Python 3.10+ is installed.
- Check installation:

```
python --version
```

Step 2: Create a virtual environment

```
python -m venv myenv
```

Step 3: Activate virtual environment

- Windows:

```
myenv\Scripts\activate
```

- macOS/Linux:

```
source myenv/bin/activate
```

Step 4: Install required packages

```
pip install numpy pandas matplotlib seaborn
```

Step 5: Verify packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

print("Packages loaded successfully")
```

Part 2: Writing Modular Programs with Functions

Task 1: Create a Python module

- Create a file data_utils.py with the following functions:

```
def mean(lst):
    return sum(lst)/len(lst)

def max_value(lst):
    return max(lst)

def min_value(lst):
    return min(lst)
```

Task 2: Import and use module

```
import data_utils as du

data = [10, 20, 30, 40]
print("Mean:", du.mean(data))
print("Max:", du.max_value(data))
print("Min:", du.min_value(data))
```

Task 3: Additional Function Examples

- Calculate range, variance, and standard deviation.

```
def data_range(lst):
    return max(lst) - min(lst)

def variance(lst):
    mean_val = sum(lst)/len(lst)
    return sum((x-mean_val)**2 for x in lst)/len(lst)

def std_dev(lst):
    return variance(lst)**0.5
```

Part 3: Object-Oriented Programming (OOP) for Data Workflows

Task 4: Create a class for a data sample

```
class DataSample:
    def __init__(self, name, values):
        self.name = name
        self.values = values

    def summary(self):
        return {
            "mean": sum(self.values)/len(self.values),
            "max": max(self.values),
            "min": min(self.values),
            "range": max(self.values)-min(self.values)
        }

    def display(self):
        print(f"Data Sample: {self.name}")
        print("Values:", self.values)
```

Task 5: Use the class

```
sample1 = DataSample("Sample1", [10, 20, 30, 40])
sample1.display()
print(sample1.summary())
```

Task 6: Handling multiple samples

```
samples = [
    DataSample("Sample1", [10, 20, 30]),
    DataSample("Sample2", [5, 15, 25, 35]),
    DataSample("Sample3", [8, 12, 18])
]

for s in samples:
    print(s.name, s.summary())
```

Task 7: Extending class functionality

- Add normalization, plot, and filtering methods.

```
class DataSample:
    def __init__(self, name, values):
        self.name = name
        self.values = values

    def normalize(self):
        max_val = max(self.values)
        self.values = [x/max_val for x in self.values]
```

```
def plot(self):
    import matplotlib.pyplot as plt
    plt.plot(self.values, marker='o')
    plt.title(self.name)
    plt.show()
```

Part 4: Modular OOP Workflow

Task 8: Create a module data_sample_module.py

- Move DataSample class here.
- Import it in your main script and manage multiple datasets.

Task 9: Example usage in main script

```
from data_sample_module import DataSample

samples = [
    DataSample("SampleA", [10, 20, 30]),
    DataSample("SampleB", [5, 15, 25, 35])
]

for s in samples:
    s.display()
    print(s.summary())
    s.plot()
```

Questions

1. Write a Python program to check if a given number is prime.
2. Create a function that converts a list of temperatures from Celsius to Fahrenheit.
3. Write a program to count the number of vowels in a given string.
4. Implement a function to find the factorial of a number using recursion.
5. Write a Python script to merge two dictionaries into one.
6. Create a function that takes a list of numbers and returns only the even numbers.
7. Write a program to check if a string is a palindrome.
8. Implement a class Circle with a method to calculate the area and circumference.
9. Write a Python function that reads a text file and counts the number of lines.
10. Create a program to sort a list of tuples based on the second element.
11. Implement a function to remove duplicates from a list without using set().
12. Write a Python program to find the largest and smallest numbers in a list.
13. Create a class Rectangle with methods to calculate area and perimeter.
14. Write a function to reverse a list without using the built-in reverse() method.
15. Implement a Python function that generates the Fibonacci sequence up to n terms
16. Write a program to count the frequency of each word in a given string.
17. Create a function that accepts a variable number of arguments and returns their sum.
18. Implement a class Employee with attributes name and salary and a method to apply a 10% raise.
19. Write a Python program to check if two strings are anagrams of each other.
20. Create a function that calculates the sum of digits of a given number.