University of Engineering and Technology Taxila
Computer Engineering Department

Data Structure and Algorithm

Lab Manual # 02

Lab Instructor: Engr. Shahryar Khan

University of Engineering and Technology Taxila
Computer Engineering Department

Lab Title: **Advanced Python Programming**

Lab Overview

This lab manual aims to guide students in learning advanced Python programming concepts. Topics covered include arrays, object-oriented programming concepts, modules, file handling, error handling, and various Python utilities. Students will strengthen their programming skills and develop proficiency in applying Python for practical problem-solving by completing the guided tasks.

Lab Objectives

1. Develop a deeper understanding of Python arrays and object-oriented programming (OOP) concepts.
2. Implement advanced Python functionalities such as modules, JSON, and regular expressions.
3. Learn to handle files, exceptions, and input/output operations in Python.
4. Enhance programming skills through guided tasks and real-life problem-solving scenarios.

Lab Requirements

- **Python Environment:** Ensure Python 3.10+ is installed. Use the official Python website for downloads (https://python.org).
- **VSCode Installation:** Download and install Visual Studio Code (https://code.visualstudio.com).
- **Documentation**: Use the official Python documentation for reference (https://docs.python.org/3).Reference Official Documentation: Use the Python official documentation (https://docs.python.org/3/) and sample programs to enhance learning.

Guided Tasks

Task 1: Python Arrays

Objective: Understand how arrays work in Python and perform basic operations.

Steps:

- Open a new Python file in VSCode, e.g., arrays_task.py.
- Write the following code:

```
import array

# Create an array of integers

numbers = array.array('i', [1, 2, 3, 4, 5])

# Access and modify array elements

print("Original array:", numbers)

numbers[1] = 10

print("Modified array:", numbers)

# Add and remove elements

numbers.append(6)

numbers.pop(0)

print("Updated array:", numbers)
```

What You Learn: Basics of working with arrays in Python, including accessing, modifying, and performing operations.

Expected Output

```
Original array: array('i', [1, 2, 3, 4, 5])

Modified array: array('i', [1, 10, 3, 4, 5])

Updated array: array('i', [10, 3, 4, 5, 6])
```

Task 2: Advanced Python Arrays

Objective: Explore advanced operations with Python arrays and solve real-world problems like manually sorting, reversing, finding maximum and minimum values, and counting occurrences in arrays. Learn how arrays are useful in memory-efficient storage and computation, such as processing large datasets or performing mathematical operations like calculating stock price variations over time.

Real-World Application:

Consider a stock market application where you need to store daily stock prices. Arrays are used to efficiently manage and process this data, including calculating trends or identifying the highest and lowest prices.

Steps:

- Create a Python file advanced_arrays.py
- Write the following program to implement advanced operations:

```
import array

# Create an array of integers

numbers = array.array('i', [5, 3, 8, 6, 2])

# Reverse the array

numbers.reverse()

print("Reversed array:", numbers)

# Sort the array (manual sorting)

for i in range(len(numbers)):

    for j in range(i + 1, len(numbers)):

        if numbers[i] > numbers[j]:

            numbers[i], numbers[j] = numbers[j], numbers[i]

print("Manually sorted array:", numbers)

# Find the maximum and minimum values
```

```
print("Maximum value:", max(numbers))

print("Minimum value:", min(numbers))

# Count occurrences of an element

numbers.append(5)

print("Occurrences of 5:", numbers.count(5))
```

Expected Output:

```
Reversed array: array('i', [2, 6, 8, 3, 5])

Manually sorted array: array('i', [2, 3, 5, 6, 8])

Maximum value: 8

Minimum value: 2

Occurrences of 5: 2
```

Task 3: Solving Real-Life Problems with Arrays

Objective: Solve real-world problems using arrays, such as analyzing rainfall data to calculate average monthly rainfall and identify months with above-average rainfall. This task demonstrates the practicality of arrays in managing numerical data for decision-making purposes in fields like meteorology or agriculture.

Real-World-Application:
In agriculture, rainfall data over months is crucial for planning crop cycles. Arrays store this data for quick access, processing, and generating insights.

Steps:

- Create a file real_life_arrays.py.
- Write the following program:

```
import array

# Problem: Calculate the average rainfall over a year

rainfall = array.array('f', [2.3, 1.5, 0.0, 1.2, 3.4, 2.0, 4.5, 3.2, 0.8, 1.9, 2.4, 3.1])

total_rainfall = sum(rainfall)

average_rainfall = total_rainfall / len(rainfall)
```

```
print("Total rainfall:", total_rainfall)

print("Average monthly rainfall:", round(average_rainfall, 2))

# Identify months with above-average rainfall

above_average = [i + 1 for i, value in enumerate(rainfall) if value > average_rainfall]

print("Months with above-average rainfall:", above_average)
```

Expected Output:

```
Total rainfall: 26.5

Average monthly rainfall: 2.21

Months with above-average rainfall: [5, 7, 8, 12]
```

Task 4: Python Classes and Objects

Objective:
Understand the foundational concept of object-oriented programming by defining and using classes and objects. Learn how to model entities like students in a class, with attributes like name and age and behaviors like displaying information.

Real-World-Application:
In education management systems, students' data is often modeled as objects to efficiently manage and display their details.

Steps

- Open a Python file classes_objects.py.
- Write the following code:

```
class Student:

    def __init__(self, name, age):

        self.name = name

        self.age = age


    def display_info(self):

        print(f"Student Name: {self.name}, Age: {self.age}")
```

```
# Create an object of the Student class
student1 = Student("Alice", 20)
student1.display_info()
```

Expected Output:

```
Student Name: Alice, Age: 20
```

Task 5: Encapsulation Using Private Members

Objective:
Learn how to implement encapsulation by defining private members in a class. Understand how private attributes enhance data security by restricting direct access, with controlled access through methods.

Real-World Application:
In banking systems, sensitive data like account numbers and balances must be kept secure. Encapsulation ensures such details are only accessed or modified through authorized methods.

Steps:

1. Open a Python file *encapsulation.py*
2. Write the following code:

```
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number  # Private attribute
        self.__balance = balance
    def deposit(self, amount):
        self.__balance += amount
    def withdraw(self, amount):
```

```
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient funds")
    def display_balance(self):
        print(f"Account Number: {self.__account_number}, Balance: {self.__balance}")


# Create an object of BankAccount
account = BankAccount("123456789", 1000)
account.deposit(500)
account.withdraw(200)
account.display_balance()
```

Expected Output:

```
Account Number: 123456789, Balance: 1300
```

Task 6: Python Inheritance

Objective:
Understand inheritance by modeling a base class and extending its functionality through a derived class. Learn how to reuse and enhance existing code.

Real-World Application:
Inheritance is widely used in software systems like inventory management, where base functionality (e.g., item details) is extended to handle specific types like perishable and non-perishable items.

Steps:

1. Open a Python file inheritance.py.
2. Write the following code:

```
class Vehicle:
```

```
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}")

class Car(Vehicle):
    def __init__(self, brand, model, doors):
        super().__init__(brand, model)
        self.doors = doors
    def display_info(self):
        super().display_info()
        print(f"Doors: {self.doors}")
# Create an object of Car
car = Car("Toyota", "Corolla", 4)
car.display_info()
```

Expected Output:

```
Brand: Toyota, Model: Corolla

Doors: 4
```

Task 7: Polymorphism (Overloading and Overriding)

Objective:
Explore polymorphism by implementing method overloading and overriding. Learn how polymorphism provides flexibility in handling multiple data types and behaviors.

Real-World Application:
In e-commerce applications, the same operation (e.g., calculating discounts) can work differently for various user types (e.g., regular customers, premium members).

Steps

Write the following code to learn about the **overloading**

```
class Calculator:
    def add(self, a, b, c=0):
        return a + b + c

calc = Calculator()
print("Addition of 2 numbers:", calc.add(10, 20))
print("Addition of 3 numbers:", calc.add(10, 20, 30))
```

Overriding example:

```
class Animal:
    def sound(self):
        print("Some generic sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

animal = Animal()
dog = Dog()
```

```
animal.sound()

dog.sound()
```

Note: Describe the output and declare more objects to test the code and create more functions.

Task 8: Abstraction Using Abstract Classes

Objective:
Learn how abstraction allows you to define a blueprint for classes, ensuring all derived classes implement specific methods.

Real-World Application:
In transportation systems, an abstract class for vehicles can ensure all types of vehicles implement basic methods like start and stop.

Steps:

- Open a Python file abstraction.py.
- Write the following code:

```
from abc import ABC, abstractmethod

class Shape(ABC):

    @abstractmethod

    def area(self):

        pass

    @abstractmethod

    def perimeter(self):

        pass


class Rectangle(Shape):

    def __init__(self, length, width):

        self.length = length
```

```
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

rect = Rectangle(10, 5)
print("Area:", rect.area())
print("Perimeter:", rect.perimeter())
```

Note: Create more classes for using this abstract class and create more functions.

Task 9: Python Modules

Objective: Understand how to use built-in and custom Python modules.
Steps:

- Create a Python file, **math_operations.py**, with the following code

```
def add(a, b):
    return a + b
def multiply(a, b):
    return a * b
```

Create another file, **use_modules.py,** with the following code:

```
import math_operations
# Use a custom module
print("Addition:", math_operations.add(5, 3))
print("Multiplication:", math_operations.multiply(5, 3))
```

```
# Use built-in module

import math

print("Square root of 16:", math.sqrt(16))
```

Expected Output

```
Addition: 8

Multiplication: 15

Square root of 16: 4.0
```

Task 10: Python File handling

Objective: Learn to read, write, create, and delete files in Python.
Steps:

1. Open a file, file_handling.py, and write the following code:

```
# Write to a file

with open("example.txt", "w") as file:

    file.write("Hello, Python file handling!")

# Read the file

with open("example.txt", "r") as file:

    content = file.read()

    print("File content:", content)

# Append to the file

with open("example.txt", "a") as file:

    file.write("\nAdding a new line.")

# Delete the file

import os

os.remove("example.txt")
```

Expected Output

```
File content: Hello, Python file handling!
```

Task 11: Python JSON

Objective: Understand how to parse and create JSON data in Python.
Steps:

1. Open a Python file, json_task.py, and write the following code:

```
import json

# Convert Python dictionary to JSON

data = {"name": "Alice", "age": 25, "city": "New York"}

json_data = json.dumps(data)

print("JSON Data:", json_data)

# Convert JSON to Python dictionary

python_data = json.loads(json_data)

print("Python Dictionary:", python_data)
```

Expected Output

```
JSON Data: {"name": "Alice", "age": 25, "city": "New York"}

Python Dictionary: {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

Task 12: Python Regular Expressions

Objective: Learn to use regular expressions for pattern matching.
Steps:

1. Open a file, **regex_task.py**, and write the following code:

```
import re

text = "The rain in Spain falls mainly in the plain."
```

```
# Find all words starting with 'S'
pattern = r"\bS\w+"
result = re.findall(pattern, text)
print("Words starting with S:", result)
# Replace 'rain' with 'snow'
modified_text = re.sub(r"rain", "snow", text)
print("Modified text:", modified_text)
```

Expected Output

```
Words starting with S: ['Spain']

Modified text: The snow in Spain falls mainly in the plain.
```

---------- xxxxx ------- xxxx ----------

Exercise Questions:

Basic Problems (5 Questions)

1. **Array Rotation** Write a Python program to rotate an array by a given number of steps.
   - Input: `[1, 2, 3, 4, 5]`, Steps: `2`
   - Output: `[4, 5, 1, 2, 3]`
2. **Encapsulation with Getters and Setters** Create a class `BankAccount` with private attributes `account_number` and `balance`. Use getters and setters to update and retrieve these values.
   - Input: `account_number=12345`, `balance=10000`
   - Output: `Account Number: 12345, Balance: 10000`
3. **Date Difference** Write a program to calculate the number of days between two given dates.
   - Input: `Date 1: 2025-01-01`, `Date 2: 2025-01-15`
   - Output: `Difference: 14 days`
4. **Basic JSON Handling** Write a program to create a JSON object with keys `name`, `age`, and `grade`. Then read and print the values.
   - Input: `{ "name": "Alice", "age": 20, "grade": "A" }`
   - Output:

```
makefile
CopyEdit
Name: Alice
Age: 20
Grade: A
```

Using Python Math Library Create a program to find the greatest common divisor (GCD) and least common multiple (LCM) of two numbers using the `math` module.

- Input: `12, 15`
- Output:

```
makefile
CopyEdit
GCD: 3
LCM: 60
```

Intermediate Problems (5 Questions)

6. **File Handling: Find Longest Word** Write a program to read a text file and find the longest word in the file.
   - Input File Content: `"The quick brown fox jumps over the lazy dog"`
   - Output: `Longest Word: jumps`

7. **Inheritance: Employee and Manager Classes** Create a base class `Employee` with attributes `name` and `salary`. Create a derived class `Manager` with an additional attribute `department`. Write methods to display their details.
   - Input: `name=John, salary=50000, department=HR`
   - Output:

```
makefile
CopyEdit
Name: John
Salary: 50000
Department: HR
```

8. **Regex Validation for Email** Write a program to validate whether an input string is a valid email address using `re` (Regex).
   - Input: `test@example.com`
   - Output: `Valid Email`

9. **Python String Formatting** Write a program to generate a formatted invoice using string formatting.
    - Input: `Item=Pen, Price=5, Quantity=10`
    - Output:

```
makefile
CopyEdit
Item: Pen
Price: Rs. 5
Quantity: 10
Total: Rs. 50
```

10. **User Input and PIP Library Usage** Write a program to install a Python package using `pip` and ask the user for the package name at runtime.
    - Input: `Package name: numpy`
    - Output:

```
CopyEdit
Installing numpy...
Installation successful.
```

Advanced Problems (5 Questions)

11. **Polymorphism: Shapes Area Calculation** Create a base class `Shape` with a method `calculate_area()`.

Implement two child classes `Rectangle` and `Circle` with overridden methods to calculate the area of their respective shapes.

    - Input: `Rectangle: length=5, width=3; Circle: radius=4`
    - Output:

```
mathematica
CopyEdit
Rectangle Area: 15
Circle Area: 50.24
```

12. **Abstract Class for Payment Processing** Create an abstract class `Payment` with an abstract method `process_payment()`. Implement subclasses `CreditCardPayment` and `PaypalPayment` with their respective implementations.
    - Input: `Payment Type: Credit Card, Amount: 1000`
    - Output:

```
yaml
CopyEdit
Processing Credit Card Payment of Rs. 1000
```

13. **JSON Data Analysis** Write a program to load a JSON file containing student data (name, marks) and calculate the average marks of all students.
    - Input JSON:

```
json
CopyEdit
[
  {"name": "Alice", "marks": 85},
  {"name": "Bob", "marks": 90},
  {"name": "Charlie", "marks": 78}
]
```

    - Output: `Average Marks: 84.33`

14. **Regex Search in Log File** Write a program to extract all IP addresses from a given server log file using regex.
    - Input File Content:

```
csharp
CopyEdit
192.168.1.1 - Accessed on 2025-01-19
10.0.0.2 - Accessed on 2025-01-20
```

    - Output:

```
yaml
CopyEdit
IP Addresses: 192.168.1.1, 10.0.0.2
```

15. **File Handling: Merge and Sort Files** Write a program to merge two text files and sort the combined content alphabetically.
    - Input File 1: `"apple, banana, orange"`
    - Input File 2: `"cherry, fig, grape"`
    - Output File: `"apple, banana, cherry, fig, grape, orange"`

All the best