Data Structures and Algorithms

(DSA)

Lab Report 2

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 3rd

Department: CPED

Submitted To:

Engineer Sheharyar Khan

Marks Obtained: 8

Total Marks: 8

Marks Distribution:

Total Lab Report Marks: 04 Total Lab Activity Marks: 04

Table of Contents

Examples:

Task 1: Python Arrays

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)
```

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

Code:

```
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))
```

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

Code:

```
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)
```

Output:

```
Total rainfall: 26.300000131130220
Average monthly rainfall: 2.19
Months with above-average rainfall: [1, 5, 7, 8, 11, 12]
```

Task 4: Python Classes and Objects

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()
```

Output:

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

Task 5: Encapsulation Using Private Members

Code:

DSA_L2 > Examples > task5_encapsulation.py > ...

```python
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()
```

Output:

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

Task 6: Python Inheritance

Code:

```python
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()
```

Output:

```
Brand: Toyota, Model: Corolla
Doors: 4
```

Task 7: Polymorphism (Overloading and Overriding)

a- Overloading

Code a:

DSA_L2 > Examples > task7_overloading.py > Calculator > add

```python
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))
```

Output:

```
Addition of 2 numbers: 30
Addition of 3 numbers: 60
```

b- Overriding

Code b:

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

    def eat(self):
        print("This animal is eating")

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

    def fetch(self):
        print("The dog is fetching the ball")

# Creating more instances of Animal and Dog
animal1 = Animal()
animal2 = Animal()
dog1 = Dog()
dog2 = Dog()
```

```
20
21  # Testing sound method
22  animal1.sound()
23  animal2.sound()
24  dog1.sound()
25  dog2.sound()
26
27  # Testing additional methods
28  animal1.eat()
29  animal2.eat()
30  dog1.fetch()
31  dog2.fetch()
```

Output:

```
This animal is eating
This animal is eating
The dog is fetching the ball
The dog is fetching the ball
```

Task 8: Abstraction Using Abstract Classes

Code:

```
1   from abc import ABC, abstractmethod
2   class Shape(ABC):
3    @abstractmethod
4    def area(self):
5     pass
6    @abstractmethod
7    def perimeter(self):
8     pass
9
10  class Rectangle(Shape):
11   def __init__(self, length, width):
12    self.length = length
13    self.width = width
14   def area(self):
15    return self.length * self.width
16   def perimeter(self):
17    return 2 * (self.length + self.width)
18
19  rect = Rectangle(10, 5)
20  print("Area:", rect.area())
21  print("Perimeter:", rect.perimeter())
```

Output:

```
Area: 50
Perimeter: 30
```

Task 9: Python Modules

Code:

```python
import task9_math_operations
# Use a custom module
print("Addition:", task9_math_operations.add(5, 3))
print("Multiplication:", task9_math_operations.multiply(5, 3))
# Use built-in module
import math
print("Square root of 16:", math.sqrt(16))
```

Output:

```
Addition: 8
Multiplication: 15
Square root of 16: 4.0
```

Task 10: Python File handling

Code:

```python
# 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")
```

Output:

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

Task 11: Python JSON

Code:

```python
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)
```

Output:

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

Task 12: Python Regular Expressions

Code:

```python
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)
```

Output:

```
Words starting with S: ['Spain']
Modified text: The snow in Spain falls mainly in the plain.
```

PROBLEMS

Beginner Problems

Question 1: Array Rotation

Write a Python program to rotate an array by a given number of

steps.

o Input: [1, 2, 3, 4, 5], Steps: 2

o Output: [4, 5, 1, 2, 3]

Code:

```
n=int(input("Enter the number of elements in the array: "))
arr=[]
for i in range(n):
    arr.append(int(input("Enter the element: ")))
steps=int(input("Enter the number of steps to rotate the array: "))
print("Original array:",arr)
for i in range(steps):
    arr.insert(0,arr.pop())
print("Rotated array:",arr)
```

Output:

```
Enter the number of elements in the array: 5
Enter the element: 1
Enter the element: 2
Enter the element: 3
Enter the element: 4
Enter the element: 5
Enter the number of steps to rotate the array: 2
Original array: [1, 2, 3, 4, 5]
Rotated array: [4, 5, 1, 2, 3]
```

Question 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.

o Input: account_number=12345, balance=10000

o Output: Account Number: 12345, Balance: 10000

Code:

```
class BankAccount:
    def __init__(self, account_number, balance):
        # Private attributes
        self.__account_number = account_number
        self.__balance = balance
    # Getter for account_number
    def get_account_number(self):
        return self.__account_number
```

```
    # Setter for account_number
    def set_account_number(self, account_number):
        self.__account_number = account_number
    # Getter for balance
    def get_balance(self):
        return self.__balance
    # Setter for balance
    def set_balance(self, balance):
        if balance >= 0:  # Ensure balance is not negative
            self.__balance = balance
        else:
            print("Balance cannot be negative!")
    # Display account details
    def display_details(self):
        print(f"Account Number: {self.__account_number}")
        print(f"Balance: {self.__balance}")
# Main block for testing
if __name__ == "__main__":
    # Take input from the user
    account_number = int(input("Enter account number: "))
    balance = float(input("Enter initial balance: "))
    # Create BankAccount object
    account = BankAccount(account_number, balance)
    # Display initial details
    print("\nInitial Account Details:")
    account.display_details()
    # Update values using user input
    new_account_number = int(input("\nEnter new account number: "))
    account.set_account_number(new_account_number)
    new_balance = float(input("Enter new balance: "))
    account.set_balance(new_balance)
```

```
# Display updated details
print("\nUpdated Account Details:")
account.display_details()
```

Output:

```
Enter account number: 12345
Enter initial balance: 10000

Initial Account Details:
Account Number: 12345
Balance: 10000.0
```

Question 3: Date Difference

Write a program to calculate the number of days between two given dates.

o Input: Date 1: 2025-01-01, Date 2: 2025-01-15

o Output: Difference: 14 days

Code:

```
from datetime import datetime
def date_difference(date1_str, date2_str):
    # Convert the string dates to datetime objects
    date_format = "%Y-%m-%d"
    date1 = datetime.strptime(date1_str, date_format)
    date2 = datetime.strptime(date2_str, date_format)
    # Calculate the difference in days
    difference = (date2 - date1).days
    return difference
# Input dates
date1 = "2025-01-01"
date2 = "2025-01-15"
# Calculate and print the difference
days_difference = date_difference(date1, date2)
print("Difference:", days_difference, "days")
```

Output:

```
Difference: 14 days
```

Question 4: Basic JSON Handling

Write a program to create a JSON object with keys name, age,

and grade. Then read and print the values.

o Input: { "name": "Alice", "age": 20, "grade": "A" }

o Output:

Name: Alice

Age: 20

Grade: A

Code:

```
import json

json_data = '{"name": "Alice", "age": 20, "grade": "A"}'

# Convert JSON data to Python dictionary

data = json.loads(json_data)

# Print the values

print("Name:", data["name"])  # Output: Name: Alice

print("Age:", data["age"])    # Output: Age: 20

print("Grade:", data["grade"]) # Output: Grade: A
```

Output:

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

Question 5: 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.

o Input: 12, 15

o Output:

GCD: 3

LCM: 60

Code:

```
import math

def find_gcd_lcm(num1, num2):

  # Calculate GCD using math.gcd()

  gcd = math.gcd(num1, num2)

  # Calculate LCM using the formula: LCM = (num1 * num2) // GCD

  lcm = (num1 * num2) // gcd
```

```
    return gcd, lcm

# Input numbers

num1 = 12

num2 = 15

# Find GCD and LCM

gcd, lcm = find_gcd_lcm(num1, num2)

# Print the results

print("GCD:", gcd)  # Output: GCD: 3

print("LCM:", lcm)  # Output: LCM: 60
```

Output:

```
GCD: 3
LCM: 60
```

Intermediate Problems

Question 6: File Handling: Find Longest Word

Write a program to read a text file and find the longest word in the file.

o Input File Content: "The quick brown fox jumps over the lazy dog"

o Output: Longest Word: jumps

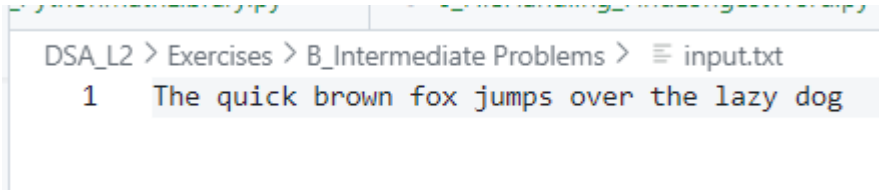
Code:

```
def find_longest_word(file_path):

    with open(file_path, 'r') as file:

        content = file.read()

    words = content.split()

    longest_word = max(words, key=len)

    return longest_word

file_path =
'D:\\4thSemester\\DSA(Python)\\DSA_Lab\\DSA_Lab_Tasks_CodeFiles\\DSA_L2\\Exercises\\B_I
ntermediate Problems\\input.txt'

# Find and print the longest word

longest_word = find_longest_word(file_path)

print("Longest Word:", longest_word)
```

Output:

```
Longest Word: quick
```

Where text file was as follows:



Question 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.

o Input: name=John, salary=50000, department=HR

o Output:

Name: John

Salary: 50000

Department: HR

Code:

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Salary: {self.salary}")
class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department
    def display_details(self):
        super().display_details()
        print(f"Department: {self.department}")
# Input
name = input("Enter name: ")
salary = input("Enter salary: ")
```

```
department = input("Enter department: ")


# Create a Manager object and display details
manager = Manager(name, salary, department)
manager.display_details()
```

Output:

```
Enter name: John
Enter salary: 2000
Enter department: Computer
Name: John
Salary: 2000
Department: Computer
```

Question 8: Regex Validation for Email

Write a program to validate whether an input string is a valid email address using re (Regex).

o Input: test@example.com

o Output: Valid Email

Code:

```
import re
def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return "Valid Email"
    else:
        return "Invalid Email"
# Input
email = input("Enter email: ")
# Validate email
result = validate_email(email)
print(result)
```

Output:

```
Enter email: iqra@gmail.com
Valid Email
```

Question 9: Python String Formatting

Write a program to generate a formatted invoice using string formatting.

o Input: Item=Pen, Price=5, Quantity=10

o Output:

Item: Pen

Price: Rs. 5

Quantity: 10

Total: Rs. 50

Code:

```
def generate_invoice(item, price, quantity):
    total = price * quantity
    invoice = (
        f"Item: {item}\n"
        f"Price: Rs. {price}\n"
        f"Quantity: {quantity}\n"
        f"Total: Rs. {total}"
    )
    return invoice
# Input
item = input("Enter item name: ")
price = float(input("Enter price: "))
quantity = float(input("Enter quantity: "))
# Generate and print invoice
invoice = generate_invoice(item, price, quantity)
print(invoice)
```

Output:

```
Enter item name: pizza
Enter price: 100
Enter quantity: 2
Item: pizza
Price: Rs. 100.0
Quantity: 2.0
Total: Rs. 200.0
```

Question 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.

o Input: Package name: numpy

o Output:

Installing numpy...

Installation successful.

Code:

```
import subprocess

def install_package(package_name):
    try:
        subprocess.check_call(["pip", "install", package_name])
        print(f"Installing {package_name}...")
        print("Installation successful.")
    except subprocess.CalledProcessError:
        print("Installation failed.")
# Input
package_name = input("Package name: ")
# Install the package
install_package(package_name)
```

Output:

```
  Package name: numpy
  Collecting numpy
    Downloading numpy-2.2.2-cp313-cp313-win_amd64.whl.metadata (60 kB)
  Downloading numpy-2.2.2-cp313-cp313-win_amd64.whl (12.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.6/12.6 MB 6.0 MB/s eta 0:00:00
  Installing collected packages: numpy
  Successfully installed numpy-2.2.2
  Installing numpy...
  Installation successful.
  PS D:\4thSemester\DSA(Python)\DSA_Lab\DSA_Lab_Tasks_CodeFiles>
```

Advanced Problems

Question 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.

o Input: Rectangle: length=5, width=3; Circle: radius=4

o Output:

Rectangle Area: 15

Circle Area: 50.24

Code:

```
import math

class Shape:
    def calculate_area(self):
        pass
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def calculate_area(self):
        return self.length * self.width
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def calculate_area(self):
```

```
        return math.pi * self.radius ** 2
# Input
length = float(input("Enter length of rectangle: "))
width = float(input("Enter width of rectangle: "))
radius = float(input("Enter radius of circle: "))
rect = Rectangle(length, width)
circle = Circle(radius)
# Output
print("Rectangle Area:", rect.calculate_area())  # Output: 15
print("Circle Area:", round(circle.calculate_area(), 2))  # Output: 50.24
```

Output:

```
Enter length of rectangle: 10
Enter width of rectangle: 2
Enter radius of circle: 5
Rectangle Area: 20.0
Circle Area: 78.54
```

Question 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.

o Input: Payment Type: Credit Card, Amount: 1000

o Output:

Processing Credit Card Payment of Rs. 1000

Code:

```
from abc import ABC, abstractmethod
class Payment(ABC):
    @abstractmethod
    def process_payment(self, amount):
        pass
class CreditCardPayment(Payment):
    def process_payment(self, amount):
        print(f"Processing Credit Card Payment of Rs. {amount}")
class PaypalPayment(Payment):
```

```
    def process_payment(self, amount):
        print(f"Processing PayPal Payment of Rs. {amount}")
# Input
payment_type = input("Enter payment type('Credit Card' or 'Pay Pal'): ")
amount = float(input("Enter amount: "))
if payment_type == "Credit Card":
    payment = CreditCardPayment()
elif payment_type == "PayPal":
    payment = PaypalPayment()
```

Output:

```
Enter payment type: Credit Card
Enter amount: 1000
Processing Credit Card Payment of Rs. 1000.0
```

Question 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.

o Input JSON:

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

o Output: Average Marks: 84.33

Code:

```
import json
def calculate_average_marks(json_data):
    students = json.loads(json_data)
    total_marks = sum(student['marks'] for student in students)
    average_marks = total_marks / len(students)
    return round(average_marks, 2)
# Input JSON
json_data = '''
[
```

```
    {"name": "Alice", "marks": 85},
    {"name": "Bob", "marks": 90},
    {"name": "Charlie", "marks": 78}
]
'''
average_marks = calculate_average_marks(json_data)
print("Average Marks:", average_marks)
```

Output:

```
Average Marks: 84.33
```

Question 14: Regex Search in Log File

Write a program to extract all IP addresses from a given server log file using regex.

o Input File Content:

192.168.1.1 - Accessed on 2025-01-19

10.0.0.2 - Accessed on 2025-01-20

o Output:

IP Addresses: 192.168.1.1, 10.0.0.2

Code:

```
import re
def extract_ip_addresses(file_content):
    pattern = r'\b(?:\d{1,3}\.){3}\d{1,3}\b'
    ip_addresses = re.findall(pattern, file_content)
    return ip_addresses
# Input File Content
log_content = '''
192.168.1.1 - Accessed on 2025-01-19
10.0.0.2 - Accessed on 2025-01-20
'''
ip_addresses = extract_ip_addresses(log_content)
print("IP Addresses:", ", ".join(ip_addresses))
```

Output:

```
IP Addresses: 192.168.1.1, 10.0.0.2
```

Question 15: File Handling: Merge and Sort Files

Write a program to merge two text files and sort the combined content alphabetically.

o Input File 1: "apple, banana, orange"

o Input File 2: "cherry, fig, grape"

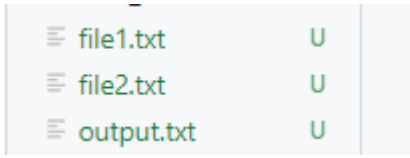
o Output File: "apple, banana, cherry, fig, grape, orange"

Code:

```
def merge_and_sort_files(file1_path, file2_path, output_file_path):
    with open(file1_path, 'r') as file1, open(file2_path, 'r') as file2:
        content1 = file1.read().split(', ')
        content2 = file2.read().split(', ')

    merged_content = sorted(content1 + content2)

    with open(output_file_path, 'w') as output_file:
        output_file.write(', '.join(merged_content))

# Input Files Content
file1_path = 'file1.txt'
file2_path = 'file2.txt'
output_file_path = 'output.txt'

with open(file1_path, 'w') as file1:
    file1.write("apple, banana, orange")

with open(file2_path, 'w') as file2:
    file2.write("cherry, fig, grape")

# Process and Output
merge_and_sort_files(file1_path, file2_path, output_file_path)

with open(output_file_path, 'r') as output_file:
```

```
print(output_file.read())  # Output: apple, banana, cherry, fig, grape, orange
```

Files Created:



Output:

```
apple, banana, cherry, fig, grape, orange
```