Submitted To:

Engineer Sheharyar Khan

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 3rd

Department: CPED

Data Structures and Algorithms

(DSA)  
Lab Report 3

# **Lab Report 3**

Marks Obtained: Not evaluated yet

Total Marks: 08

**Marks Distribution:**

Total Lab Activity Marks: 4

Total Lab Report Marks: 4

## **Example Tasks**

### Task : To-Do List Application

class ToDoList:

def \_\_init\_\_(self):

self.tasks = []

def add\_task(self, task):

self.tasks.append(task)

def remove\_task(self, task):

if task in self.tasks:

self.tasks.remove(task)

else:

print(f"Task '{task}' not found.")

def view\_tasks(self):

print("Tasks:")

for i, task in enumerate(self.tasks, 1):

print(f"{i}. {task}")

to\_do = ToDoList()

to\_do.add\_task("Buy groceries")

to\_do.add\_task("Complete homework")

to\_do.view\_tasks()

to\_do.remove\_task("Buy groceries")

to\_do.view\_tasks()

**Output:**

A close-up of black text

Description automatically generated

### Task 2: Expense Tracker

class ExpenseTracker:

def \_\_init\_\_(self):

self.expenses = []

def add\_expense(self, amount):

self.expenses.append(amount)

def total\_expenses(self):

return sum(self.expenses)

def max\_expense(self):

return max(self.expenses) if self.expenses else 0

def min\_expense(self):

return min(self.expenses) if self.expenses else 0

expense\_tracker = ExpenseTracker()

expense\_tracker.add\_expense(20.5)

expense\_tracker.add\_expense(100.75)

print("Total Expenses:", expense\_tracker.total\_expenses())

print("Max Expense:", expense\_tracker.max\_expense())

print("Min Expense:", expense\_tracker.min\_expense())

**Output:**

A close up of words

Description automatically generated

### Task 3: Student Grade Tracker

class GradeTracker:

def \_\_init\_\_(self):

self.grades = []

def add\_grade(self, grade):

self.grades.append(grade)

def average\_grade(self):

return sum(self.grades) / len(self.grades) if self.grades else 0

def highest\_grade(self):

return max(self.grades) if self.grades else 0

def lowest\_grade(self):

return min(self.grades) if self.grades else 0

grades = GradeTracker()

grades.add\_grade(85)

grades.add\_grade(90)

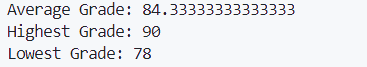
grades.add\_grade(78)

print("Average Grade:", grades.average\_grade())

print("Highest Grade:", grades.highest\_grade())

print("Lowest Grade:", grades.lowest\_grade())

**Output:**



### Task 4: Library Management System

class Library:

def \_\_init\_\_(self):

self.books = []

def add\_book(self, title, author, status):

self.books.append([title, author, status])

def search\_book(self, title):

for book in self.books:

if book[0] == title:

return book

return None

def display\_books(self):

for book in self.books:

print(book)

library = Library()

library.add\_book("Book1", "Author1", "Available")

library.add\_book("Book2", "Author2", "Issued")

library.display\_books()

**Output:**



### Task 5: RGB Image Processing

class RGBImage:

def \_\_init\_\_(self, rows, cols):

self.image = [[[0, 0, 0] for \_ in range(cols)] for \_ in range(rows)]

def update\_pixel(self, row, col, rgb):

self.image[row][col] = rgb

def get\_pixel(self, row, col):

return self.image[row][col]

image = RGBImage(2, 2)

image.update\_pixel(0, 0, [255, 0, 0])

image.update\_pixel(0, 1, [0, 255, 0])

print("Pixel RGB Value:", image.get\_pixel(0, 1))

**Output:**



## **Exercise Problems**

### **Easy Problems**

#### 1. To-Do List Enhancement

Create a to-do list program that allows users to mark tasks as "completed" and filter

only completed tasks to display.

**Hint:** Use a 1-D array to store tasks and a parallel array to store their completion

status (True/False).

**Code:**

class ToDoList:

def \_\_init\_\_(self):

self.tasks = [] # List of task names

self.status = [] # Parallel list for task status (True = Completed, False = Pending)

def add\_task(self, task):

"""Add a new task with default status as False (Pending)."""

self.tasks.append(task)

self.status.append(False)

def mark\_completed(self, task):

"""Mark a task as completed if it exists."""

if task in self.tasks:

index = self.tasks.index(task)

self.status[index] = True

print(f"Task '{task}' marked as completed.")

else:

print(f"Task '{task}' not found.")

def view\_completed\_tasks(self):

"""Display only completed tasks."""

print("Completed Tasks:")

found = False

for i, (task, done) in enumerate(zip(self.tasks, self.status), 1):

if done:

print(f"{i}. {task}")

found = True

if not found:

print("No completed tasks yet.")

def view\_all\_tasks(self):

"""Display all tasks with their status."""

print("All Tasks:")

for i, (task, done) in enumerate(zip(self.tasks, self.status), 1):

status\_text = "✓ Completed" if done else "✗ Pending"

print(f"{i}. {task} - {status\_text}")

# Testing the To-Do List Enhancement

to\_do = ToDoList()

to\_do.add\_task("Buy groceries")

to\_do.add\_task("Complete homework")

to\_do.mark\_completed("Buy groceries")

to\_do.view\_completed\_tasks()

to\_do.view\_all\_tasks()

**Output:**

#### A white background with black text Description automatically generated

#### 2. Daily Expense Calculator

Write a program to store daily expenses in an array and calculate the total expenses

for the first seven days.

**Hint:** Use a for loop to sum up the first seven elements of the array.

**Code:**

class ExpenseCalculator:

def \_\_init\_\_(self):

self.expenses = [] # List to store daily expenses

def add\_expense(self, amount):

"""Add a daily expense amount."""

self.expenses.append(amount)

def total\_first\_week(self):

"""Calculate total expenses for the first 7 days."""

return sum(self.expenses[:7]) # Sum of first 7 elements

def view\_expenses(self):

"""Display all stored expenses."""

print("Daily Expenses:", self.expenses)

# Testing Daily Expense Calculator

tracker = ExpenseCalculator()

tracker.add\_expense(200)

tracker.add\_expense(150)

tracker.add\_expense(300)

tracker.add\_expense(400)

tracker.add\_expense(500)

tracker.add\_expense(250)

tracker.add\_expense(350)

tracker.add\_expense(600) # Extra expense (8th day, should not be included)

tracker.view\_expenses()

print("Total expenses for the first 7 days:", tracker.total\_first\_week())

**Output:**

****

#### 3. Student Grade Summary

Develop a program to store grades of students for a single subject and display grades

greater than or equal to the class average.

**Hint:** Calculate the average first, then use a loop to filter grades that meet the

condition.

**Code:**

class GradeSummary:

def \_\_init\_\_(self):

self.grades = [] # List to store grades

def add\_grade(self, grade):

"""Add a new grade to the list."""

self.grades.append(grade)

def calculate\_average(self):

"""Calculate the average grade."""

return sum(self.grades) / len(self.grades) if self.grades else 0

def filter\_above\_average(self):

"""Return grades that are greater than or equal to the class average."""

avg = self.calculate\_average()

return [grade for grade in self.grades if grade >= avg]

def view\_grades(self):

"""Display all stored grades."""

print("Grades:", self.grades)

# Testing Student Grade Summary

summary = GradeSummary()

summary.add\_grade(85)

summary.add\_grade(90)

summary.add\_grade(78)

summary.add\_grade(88)

summary.add\_grade(92)

summary.view\_grades()

print("Class Average:", summary.calculate\_average())

print("Grades >= Class Average:", summary.filter\_above\_average())

**Output:**

**A number on a white background

Description automatically generated**

#### 4. Find the Maximum Element

Create a program to find the maximum number in a list of positive integers entered by

the user.

**Hint:** Use a max() function or iterate through the array with a for loop.

**Code:**

# Problem 4: Find the Maximum Element

class MaxFinder:

def \_\_init\_\_(self):

self.numbers = []

def add\_number(self, number):

self.numbers.append(number)

def find\_max(self):

return max(self.numbers) if self.numbers else None

def view\_numbers(self):

print("Numbers:", self.numbers)

# Testing MaxFinder

max\_finder = MaxFinder()

numbers = [12, 45, 78, 23, 89, 56]

for num in numbers:

max\_finder.add\_number(num)

max\_finder.view\_numbers()

print("Maximum Number:", max\_finder.find\_max())

**Output:**

****

#### 5. Simple Library Search

Write a program to store book names in a library and allow a user to search for a

specific book by its name.

**Hint:** Use the in keyword to check if the book is in the array.

**Code:**

# Problem 5: Simple Library Search

class Library:

def \_\_init\_\_(self):

self.books = []

def add\_book(self, book\_name):

self.books.append(book\_name)

def search\_book(self, book\_name):

return book\_name in self.books

def view\_books(self):

print("Books in Library:", self.books)

# Testing Library Search

library = Library()

books = ["Python Basics", "Data Science Handbook", "Machine Learning Guide"]

for book in books:

library.add\_book(book)

library.view\_books()

search\_query = "Data Science Handbook"

print(f"Is '{search\_query}' available?:", library.search\_book(search\_query))

**Output:**

****

### **Intermediate Problems**

#### 1. Expense Breakdown by Category

Develop a program to track expenses for different categories (food, travel, utilities,

etc.) using a 2-D array. Calculate the total expenses for each category.

**Hint:** Use a nested list where each row corresponds to a category, and each column is

an expense.

**Code:**

# Problem 1: Expense Breakdown by Category

class ExpenseTracker:

def \_\_init\_\_(self):

self.categories = {}

def add\_expense(self, category, amount):

if category not in self.categories:

self.categories[category] = []

self.categories[category].append(amount)

def total\_expense(self, category):

return sum(self.categories.get(category, []))

def view\_expenses(self):

for category, expenses in self.categories.items():

print(f"{category}: {sum(expenses)}")

# Testing ExpenseTracker

expense\_tracker = ExpenseTracker()

expense\_tracker.add\_expense("Food", 100)

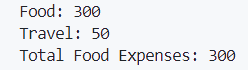
expense\_tracker.add\_expense("Travel", 50)

expense\_tracker.add\_expense("Food", 200)

expense\_tracker.view\_expenses()

print("Total Food Expenses:", expense\_tracker.total\_expense("Food"))

**Output:**

****

#### 2. Attendance Tracker

Create a program to track attendance for 5 employees over 5 days using a 2-D array.

Calculate the attendance percentage for each employee.

**Hint:** Use a loop to count the number of 1s in each row and divide by the total days.

**Code:**

# Problem 2: Attendance Tracker

class AttendanceTracker:

def \_\_init\_\_(self, employees, days):

self.attendance = [[0] \* days for \_ in range(employees)]

def mark\_attendance(self, employee, day):

self.attendance[employee][day] = 1

def attendance\_percentage(self, employee):

return (sum(self.attendance[employee]) / len(self.attendance[employee])) \* 100

def view\_attendance(self):

for emp, record in enumerate(self.attendance):

print(f"Employee {emp + 1}: {record}")

# Testing AttendanceTracker

tracker = AttendanceTracker(5, 5)

tracker.mark\_attendance(0, 0)

tracker.mark\_attendance(1, 1)

tracker.mark\_attendance(2, 2)

tracker.mark\_attendance(3, 3)

tracker.mark\_attendance(4, 4)

tracker.view\_attendance()

print("Attendance Percentage for Employee 1:", tracker.attendance\_percentage(0))

**Output:**

**A number of letters and numbers

Description automatically generated with medium confidence**

#### 3. Matrix Addition

Write a program to add two 2-D matrices (3x3) and display the resulting matrix.

**Hint:** Use nested loops to add corresponding elements from two matrices.

**Code:**

# Problem 3: Matrix Addition

class Matrix:

def \_\_init\_\_(self, matrix):

self.matrix = matrix

def add(self, other):

result = [[self.matrix[i][j] + other.matrix[i][j] for j in range(len(self.matrix[0]))] for i in range(len(self.matrix))]

return Matrix(result)

def display(self):

for row in self.matrix:

print(row)

# Testing Matrix Addition

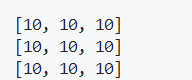
matrix1 = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

matrix2 = Matrix([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

result\_matrix = matrix1.add(matrix2)

result\_matrix.display()

**Output:**

****

#### 4. Sort Grades

Develop a program to store student grades for five students and sort them in

descending order.

**Hint:** Use a sorting algorithm or Python's sorted() function with the reverse

parameter set to True.

**Code:**

# Problem 4: Sort Grades

class GradeSorter:

def \_\_init\_\_(self, grades):

self.grades = grades

def sort\_grades(self):

return sorted(self.grades, reverse=True)

def display(self):

print("Sorted Grades:", self.sort\_grades())

# Testing GradeSorter

grades = GradeSorter([85, 92, 78, 90, 88])

grades.display()

**Output:**



#### 5. 2-D Tic-Tac-Toe Enhancements

Extend the tic-tac-toe game to announce the winner (player "X" or "O") or declare it as a draw after all moves are completed.

**Hint:** Check rows, columns, and diagonals for identical values to determine the winner.

**Code:**

class TicTacToe:

def \_\_init\_\_(self):

self.board = [[' ' for \_ in range(3)] for \_ in range(3)]

def make\_move(self, row, col, player):

if self.board[row][col] == ' ':

self.board[row][col] = player

else:

print("Invalid Move!")

def check\_winner(self):

for row in self.board:

if row[0] == row[1] == row[2] != ' ':

return row[0]

for col in range(3):

if self.board[0][col] == self.board[1][col] == self.board[2][col] != ' ':

return self.board[0][col]

if self.board[0][0] == self.board[1][1] == self.board[2][2] != ' ':

return self.board[0][0]

if self.board[0][2] == self.board[1][1] == self.board[2][0] != ' ':

return self.board[0][2]

return None

def display(self):

for row in self.board:

print(" | ".join(row))

print("-" \* 9)

# Testing TicTacToe

game = TicTacToe()

game.make\_move(0, 0, 'X')

game.make\_move(1, 1, 'O')

game.make\_move(0, 1, 'X')

game.make\_move(2, 2, 'O')

game.make\_move(0, 2, 'X')

game.display()

winner = game.check\_winner()

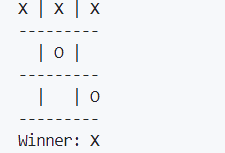
if winner:

print("Winner:", winner)

else:

print("No Winner Yet!")

**Output:**

****

### **Advanced Problems**

#### 1. Facebook Notifications System

Implement a notifications queue to manage and display sequential updates, allowing

users to "clear all" or view the latest five notifications.

**Hint:** Use a 1-D array as a queue and maintain a size limit for the array.

**Code:**

class NotificationSystem:

def \_\_init\_\_(self, limit=5):

self.notifications = []

self.limit = limit

def add\_notification(self, notification):

if len(self.notifications) >= self.limit:

self.notifications.pop(0) # Remove the oldest notification

self.notifications.append(notification)

def clear\_all(self):

self.notifications = []

def view\_latest\_notifications(self):

print("Latest Notifications:")

for i, notification in enumerate(self.notifications[-5:], 1): # Display only the latest 5 notifications

print(f"{i}. {notification}")

# Example usage

notif\_system = NotificationSystem()

notif\_system.add\_notification("New friend request")

notif\_system.add\_notification("Message from John")

notif\_system.add\_notification("Your post was liked")

notif\_system.view\_latest\_notifications()

notif\_system.clear\_all()

notif\_system.view\_latest\_notifications()

**Output:**

A screenshot of a computer screen

Description automatically generated

#### 2. Instagram Image Filter

Write a program to apply a grayscale filter on a 3-D array representing RGB pixel

values of an image. Convert each pixel to grayscale using the formula:

Gray = (R + G + B) / 3.

**Hint:** Iterate through each pixel (row and column) and apply the formula to update the

pixel values.

**Code:**

class ImageProcessor:

def \_\_init\_\_(self, image):

self.image = image # 3D array representing RGB pixels

def apply\_grayscale(self):

for i in range(len(self.image)):

for j in range(len(self.image[i])):

r, g, b = self.image[i][j]

gray = (r + g + b) // 3

self.image[i][j] = [gray, gray, gray] # Apply grayscale

def display\_image(self):

for row in self.image:

print(row)

# Example usage

image = [

[[255, 0, 0], [0, 255, 0]],

[[0, 0, 255], [255, 255, 0]]

]

processor = ImageProcessor(image)

processor.apply\_grayscale()

processor.display\_image()

**Output:**



#### 3. Snapchat Streak Tracker

Develop a program to store a 2-D array of streak counts between users over a week.

Calculate the highest streak for each user and display the user with the longest streak.

**Hint:** Iterate through each row to find the maximum streak and use it to identify the

user.

**Code:**

class StreakTracker:

def \_\_init\_\_(self):

self.streaks = {}

def add\_streak(self, user, streak\_count):

if user in self.streaks:

self.streaks[user].append(streak\_count)

else:

self.streaks[user] = [streak\_count]

def highest\_streak(self):

max\_streak\_user = None

max\_streak = 0

for user, streak\_list in self.streaks.items():

highest = max(streak\_list)

if highest > max\_streak:

max\_streak = highest

max\_streak\_user = user

return max\_streak\_user, max\_streak

# Example usage

streak\_tracker = StreakTracker()

streak\_tracker.add\_streak("User1", 5)

streak\_tracker.add\_streak("User1", 7)

streak\_tracker.add\_streak("User2", 8)

streak\_tracker.add\_streak("User2", 6)

user, streak = streak\_tracker.highest\_streak()

print(f"User with highest streak: {user} with a streak of {streak} days.")

**Output:**



#### 4. Twitter Hashtag Tracker

Create a program to count the occurrences of hashtags in a given list of tweets.

Display the top three most-used hashtags.

**Hint:** Use a dictionary to store hashtag counts and sort the dictionary by values to find

the top three.

**Code:**

class HashtagTracker:

def \_\_init\_\_(self):

self.hashtags = {}

def add\_tweet(self, tweet):

words = tweet.split()

for word in words:

if word.startswith("#"):

self.hashtags[word] = self.hashtags.get(word, 0) + 1

def top\_hashtags(self, top\_n=3):

sorted\_hashtags = sorted(self.hashtags.items(), key=lambda x: x[1], reverse=True)

return sorted\_hashtags[:top\_n]

# Example usage

hashtag\_tracker = HashtagTracker()

hashtag\_tracker.add\_tweet("I love #Python programming")

hashtag\_tracker.add\_tweet("Learning #Python with fun")

hashtag\_tracker.add\_tweet("Hello #Java and #Python fans")

top\_hashtags = hashtag\_tracker.top\_hashtags()

for tag, count in top\_hashtags:

print(f"{tag}: {count} times")

**Output:**

