# Practical-14

May 19, 2025

## 0.1 Practical 14 :-

**Name:** Sarthak Sanay
**Enrollment No:** 230031101611051

### 0.1.1 Problem Statement 1:-

**Student Grade Management (CSV):**
You are a teacher at a school and maintain student grade records in a CSV file. The file contains student IDs, names, and grades for different subjects (e.g., Math, Science, English). Develop a Python program to read this CSV file, calculate each student's average grade, display the top-performing students, allow the user to search for a student by ID or name, and update their grades. Provide an option to save the updated data back to the CSV file.

[1]:
```
import csv

FILENAME = 'students.csv'

def read_students(filename):
    students = []
    with open(filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:

            for subject in ['Math', 'Science', 'English']:
                row[subject] = int(row[subject])
            students.append(row)
    return students

def calculate_averages(students):
    for student in students:
        grades = [student[subj] for subj in ['Math', 'Science', 'English']]
        student['Average'] = sum(grades) / len(grades)

def display_top_students(students, top_n=3):
    sorted_students = sorted(students, key=lambda x: x['Average'], reverse=True)
    print(f"Top {top_n} students:")
    for s in sorted_students[:top_n]:
        print(f"{s['StudentID']} - {s['Name']}: Average = {s['Average']:.2f}")
```

```
def search_student(students, query):
    query = query.lower()
    for student in students:
        if student['StudentID'] == query or student['Name'].lower() == query:
            return student
    return None

def update_grades(student):
    print(f"Updating grades for {student['Name']} (ID: {student['StudentID']})")
    for subject in ['Math', 'Science', 'English']:
        new_grade = input(f"Enter new grade for {subject} (leave blank to keep␣
 ↪{student[subject]}): ")
        if new_grade.strip():
            student[subject] = int(new_grade)

    grades = [student[subj] for subj in ['Math', 'Science', 'English']]
    student['Average'] = sum(grades) / len(grades)
    print("Grades updated.")

def save_students(filename, students):
    with open(filename, 'w', newline='') as csvfile:
        fieldnames = ['StudentID', 'Name', 'Math', 'Science', 'English']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for s in students:

            writer.writerow({
                'StudentID': s['StudentID'],
                'Name': s['Name'],
                'Math': s['Math'],
                'Science': s['Science'],
                'English': s['English']
            })
    print(f"Data saved to {filename}")

def main():
    students = read_students(FILENAME)
    calculate_averages(students)
    display_top_students(students)

    while True:
        choice = input("\nOptions:\n1. Search student\n2. Update student␣
 ↪grades\n3. Save and Exit\nChoose: ")

        if choice == '1':
            query = input("Enter Student ID or Name to search: ")
```

```
            student = search_student(students, query)
            if student:
                print(f"Found: {student['StudentID']} - {student['Name']}")
                print(f"Grades: Math={student['Math']},␣
↪Science={student['Science']}, English={student['English']}")
                print(f"Average: {student['Average']:.2f}")
            else:
                print("Student not found.")

        elif choice == '2':
            query = input("Enter Student ID or Name to update: ")
            student = search_student(students, query)
            if student:
                update_grades(student)
            else:
                print("Student not found.")

        elif choice == '3':
            save_students(FILENAME, students)
            break
        else:
            print("Invalid choice, try again.")

if __name__ == '__main__':
    main()
```

```
Top 3 students:
104 - Michael Brown: Average = 92.00
102 - Jane Smith: Average = 87.33
101 - John Doe: Average = 85.00


Options:
1. Search student
2. Update student grades
3. Save and Exit
Choose:  1
Enter Student ID or Name to search:  103

Found: 103 - Emily Davis
Grades: Math=76, Science=85, English=80
Average: 80.33


Options:
1. Search student
2. Update student grades
3. Save and Exit
Choose:  2
```

```
Enter Student ID or Name to update:  103

Updating grades for Emily Davis (ID: 103)

Enter new grade for Math (leave blank to keep 76):
Enter new grade for Science (leave blank to keep 85):
Enter new grade for English (leave blank to keep 80):  100

Grades updated.


Options:
1. Search student
2. Update student grades
3. Save and Exit
Choose:  1
Enter Student ID or Name to search:  103

Found: 103 - Emily Davis
Grades: Math=76, Science=85, English=100
Average: 87.00


Options:
1. Search student
2. Update student grades
3. Save and Exit
Choose:  3

Data saved to students.csv
```

### 0.1.2 Problem Statement 2:-

**Product Inventory Tracking (JSON):**
Your company manages a product inventory using a JSON file. Each product entry includes details such as product ID, name, quantity in stock, and unit price. Write a Python program to read this JSON file, display the list of available products, allow the user to search for a product by name or ID, update the quantity of a product after purchase, and generate a report showing low-stock items (products with quantity below a certain threshold).

[2]:
```python
import json

FILENAME = 'inventory.json'

def load_products(filename):
    with open(filename, 'r') as f:
        return json.load(f)

def save_products(filename, products):
    with open(filename, 'w') as f:
        json.dump(products, f, indent=4)
```

```
    print(f"Inventory saved to {filename}")

def display_products(products):
    print("Available Products:")
    for p in products:
        print(f"{p['ProductID']} - {p['Name']} | Quantity: {p['Quantity']} |␣
 ↪Price: ${p['UnitPrice']}")

def search_product(products, query):
    query = query.lower()
    for p in products:
        if p['ProductID'].lower() == query or p['Name'].lower() == query:
            return p
    return None

def update_quantity(product):
    print(f"Current quantity of {product['Name']}: {product['Quantity']}")
    try:
        qty = int(input("Enter quantity purchased (to reduce stock): "))
        if qty > 0 and qty <= product['Quantity']:
            product['Quantity'] -= qty
            print(f"Updated quantity: {product['Quantity']}")
        else:
            print("Invalid quantity. Purchase quantity must be positive and no␣
 ↪more than current stock.")
    except ValueError:
        print("Please enter a valid number.")

def low_stock_report(products, threshold=10):
    print(f"\nProducts with stock below {threshold}:")
    low_stock_items = [p for p in products if p['Quantity'] < threshold]
    if low_stock_items:
        for p in low_stock_items:
            print(f"{p['ProductID']} - {p['Name']} | Quantity: {p['Quantity']}")
    else:
        print("No products with low stock.")


products = load_products(FILENAME)

while True:
    print("\nOptions:")
    print("1. Display all products")
    print("2. Search for a product")
    print("3. Update product quantity after purchase")
    print("4. Generate low-stock report")
    print("5. Save and Exit")
```

```
    choice = input("Choose an option: ")

    if choice == '1':
        display_products(products)

    elif choice == '2':
        query = input("Enter Product ID or Name to search: ")
        product = search_product(products, query)
        if product:
            print(f"Found: {product['ProductID']} - {product['Name']} |␣
↪Quantity: {product['Quantity']} | Price: ${product['UnitPrice']}")
        else:
            print("Product not found.")

    elif choice == '3':
        query = input("Enter Product ID or Name to update quantity: ")
        product = search_product(products, query)
        if product:
            update_quantity(product)
        else:
            print("Product not found.")

    elif choice == '4':
        threshold = input("Enter low stock threshold (default 10): ")
        if threshold.isdigit():
            threshold = int(threshold)
        else:
            threshold = 10
        low_stock_report(products, threshold)

    elif choice == '5':
        save_products(FILENAME, products)
        break

    else:
        print("Invalid choice, please try again.")
```

```
Options:
1. Display all products
2. Search for a product
3. Update product quantity after purchase
4. Generate low-stock report
5. Save and Exit

Choose an option:  1
```

```
Available Products:
P001 - Laptop | Quantity: 15 | Price: $800
P002 - Mouse | Quantity: 50 | Price: $20
P003 - Keyboard | Quantity: 30 | Price: $35
P004 - Monitor | Quantity: 10 | Price: $150
P005 - USB Cable | Quantity: 5 | Price: $10

Options:
1. Display all products
2. Search for a product
3. Update product quantity after purchase
4. Generate low-stock report
5. Save and Exit

Choose an option:  2
Enter Product ID or Name to search:  P003

Found: P003 - Keyboard | Quantity: 30 | Price: $35

Options:
1. Display all products
2. Search for a product
3. Update product quantity after purchase
4. Generate low-stock report
5. Save and Exit

Choose an option:  3
Enter Product ID or Name to update quantity:  P003

Current quantity of Keyboard: 30

Enter quantity purchased (to reduce stock):  10

Updated quantity: 20

Options:
1. Display all products
2. Search for a product
3. Update product quantity after purchase
4. Generate low-stock report
5. Save and Exit

Choose an option:  2
Enter Product ID or Name to search:  P003

Found: P003 - Keyboard | Quantity: 20 | Price: $35

Options:
1. Display all products
2. Search for a product
3. Update product quantity after purchase
```

```
4. Generate low-stock report
5. Save and Exit

Choose an option:  4
Enter low stock threshold (default 10):  8


Products with stock below 8:
P005 - USB Cable | Quantity: 5

Options:
1. Display all products
2. Search for a product
3. Update product quantity after purchase
4. Generate low-stock report
5. Save and Exit

Choose an option:  5

Inventory saved to inventory.json
```

### 0.1.3 Problem Statement 3:-

**Customer Database Management (CSV):**
You are working for a small retail business that maintains a customer database in a CSV file. The file contains customer information such as name, email, phone number, and purchase history. Develop a Python program to read this CSV file, allow the user to search for a customer by name or email, and update their purchase history. Additionally, provide an option to add new customers to the database and save the updated data back to the CSV file.

[2]:
```python
import csv

FILENAME = 'customers.csv'

def read_customers(filename):
    customers = []
    with open(filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            customers.append(row)
    return customers

def save_customers(filename, customers):
    with open(filename, 'w', newline='') as csvfile:
        fieldnames = ['Name', 'Email', 'Phone', 'PurchaseHistory']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for c in customers:
            writer.writerow(c)
    print(f"Data saved to {filename}")
```

```
def search_customer(customers, query):
    query = query.lower()
    for customer in customers:
        if customer['Name'].lower() == query or customer['Email'].lower() ==␣
 ↪query:
            return customer
    return None

def update_purchase_history(customer):
    print(f"Current purchase history: {customer['PurchaseHistory']}")
    new_purchase = input("Enter new purchase item(s) to add (comma separated):␣
 ↪").strip()
    if new_purchase:
        if customer['PurchaseHistory']:
            customer['PurchaseHistory'] += ', ' + new_purchase
        else:
            customer['PurchaseHistory'] = new_purchase
        print("Purchase history updated.")
    else:
        print("No update made.")

def add_customer(customers):
    print("Add New Customer:")
    name = input("Name: ").strip()
    email = input("Email: ").strip()
    phone = input("Phone: ").strip()
    purchase_history = input("Purchase history (comma separated): ").strip()

    for c in customers:
        if c['Email'].lower() == email.lower():
            print("Customer with this email already exists.")
            return

    new_customer = {
        'Name': name,
        'Email': email,
        'Phone': phone,
        'PurchaseHistory': purchase_history
    }
    customers.append(new_customer)
    print("New customer added.")


customers = read_customers(FILENAME)

while True:
```

```
    print("\nOptions:")
    print("1. Search customer")
    print("2. Update purchase history")
    print("3. Add new customer")
    print("4. Save and Exit")

    choice = input("Choose an option: ")

    if choice == '1':
        query = input("Enter customer Name or Email to search: ")
        customer = search_customer(customers, query)
        if customer:
            print(f"Found: {customer['Name']} | Email: {customer['Email']} |␣
↪Phone: {customer['Phone']}")
            print(f"Purchase History: {customer['PurchaseHistory']}")
        else:
            print("Customer not found.")

    elif choice == '2':
        query = input("Enter customer Name or Email to update purchase history:␣
↪")
        customer = search_customer(customers, query)
        if customer:
            update_purchase_history(customer)
        else:
            print("Customer not found.")

    elif choice == '3':
        add_customer(customers)

    elif choice == '4':
        save_customers(FILENAME, customers)
        break

    else:
        print("Invalid choice, please try again.")
```

```
Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  1
Enter customer Name or Email to search:  bobsmith@example.com

Found: Bob Smith | Email: bobsmith@example.com | Phone: 555-5678
```

```
Purchase History: Keyboard

Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  1
Enter customer Name or Email to search:  elonmusk@example.com

Customer not found.

Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  2
Enter customer Name or Email to update purchase history:  bobsmith@example.com

Current purchase history: Keyboard

Enter new purchase item(s) to add (comma separated):  iPhone, Playstation

Purchase history updated.

Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  1
Enter customer Name or Email to search:  bobsmith@example.com

Found: Bob Smith | Email: bobsmith@example.com | Phone: 555-5678
Purchase History: Keyboard, iPhone, Playstation

Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  3

Add New Customer:

Name:  Elon Musk
Email:  elonmusk@example.com
```

```
Phone:  555-1234
Purchase history (comma separated):  Twitter

New customer added.

Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  1
Enter customer Name or Email to search:  elonmusk@example.com

Found: Elon Musk | Email: elonmusk@example.com | Phone: 555-1234
Purchase History: Twitter

Options:
1. Search customer
2. Update purchase history
3. Add new customer
4. Save and Exit

Choose an option:  4

Data saved to customers.csv
```

### 0.1.4 Problem Statement 4:-

**Text File Encryption/Decryption (TXT):**
You are working on a security application that requires encrypting and decrypting text files using a simple substitution cipher. Write a Python program that can encrypt or decrypt the contents of a text file using a provided substitution key. The program should provide options for the user to choose between encryption and decryption, specify the input and output file paths, and enter the substitution key.

[5]:
```python
import string
import os

print("Caesar Cipher File Encryptor/Decryptor :-\n")

mode = input("Enter mode (E for Encrypt / D for Decrypt): ").strip().upper()
file_path = input("Enter path to the text file: ").strip()
key_char = input("Enter a single letter as the Caesar key (A-Z): ").strip().
 ↪upper()

if len(key_char) != 1 or not key_char.isalpha():
    print("Invalid key! Must be a single letter A-Z.")
    exit()
```

```
if not os.path.isfile(file_path):
    print("File not found. Please check the file path.")
    exit()

shift = ord(key_char) - ord('A')

with open(file_path, 'r') as file:
    content = file.read()

result = []

for ch in content:
    if ch.isalpha():
        base = ord('A') if ch.isupper() else ord('a')
        offset = ord(ch) - base

        if mode == 'E':
            shifted = (offset + shift) % 26
        elif mode == 'D':
            shifted = (offset - shift + 26) % 26
        else:
            print("Invalid mode! Use 'E' for Encrypt or 'D' for Decrypt.")
            exit()

        result.append(chr(base + shifted))
    else:
        result.append(ch)

with open(file_path, 'w') as file:
    file.write(''.join(result))

print(f"\n{'Encrypted' if mode == 'E' else 'Decrypted'} content saved back to␣
 ↪{file_path}")
```

```
Caesar Cipher File Encryptor/Decryptor :-


Enter mode (E for Encrypt / D for Decrypt):  D
Enter path to the text file:  test-file.txt
Enter a single letter as the Caesar key (A-Z):  S


Decrypted content saved back to test-file.txt
```

[ ]: