Practical-12

May 19, 2025

0.1 Practical 12 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Given a string and a list of words representing a list, find the longest word in the dictionary that can be formed by deleting some characters of the given string.

Input:
s = “abpcplea”
list = [“ale”, “apple”, “monkey”, “plea”]

Output:
Longest word in list: apple

[2]:
```
s = input("Enter string: ")
size = int(input("Enter size of list: "))
word_list = []
print()
for i in range(size):
    w = input(f"Enter word {i+1}: ")
    word_list.append(w)

longest = ""

for word in word_list:
    word = word.strip()  # remove extra spaces
    i = 0  # pointer for word
    j = 0  # pointer for s
    while i < len(word) and j < len(s):
        if word[i] == s[j]:
            i += 1
        j += 1
    if i == len(word):  # word is subsequence of s
        if len(word) > len(longest) or (len(word) == len(longest) and word <␣
↪longest):
            longest = word
```

```
print("\nLongest word in list:", longest)
```

```
Enter string:  abpcplea
Enter size of list:  4


Enter word 1:  ale
Enter word 2:  apple
Enter word 3:  monkey
Enter word 4:  plea


Longest word in list: apple
```

0.1.2 Problem Statement 2:-

You're tasked with writing a function to determine whether a given sentence is a palindrome, considering only alphanumeric characters and ignoring case sensitivity and other things.

Input:
sentence1 = "A man, a plan, a canal, Panama!"
sentence2 = "race a car"

Output:
Is 'A man, a plan, a canal, Panama!' a palindrome? True
Is 'race a car' a palindrome? False

[1]:
```
def check_palindrome(s):
    sentence = ""
    for char in s:
        if char.isalnum():
            sentence += char.lower()

    return sentence == sentence[::-1]

s = input("Enter sentence: ")
print(f"Is '{s}' a palindrome? {check_palindrome(s)}")
```

```
Enter sentence:  A man, a plan, a canal, Panama!

Is 'A man, a plan, a canal, Panama!' a palindrome? True
```

0.1.3 Problem Statement 3:-

You're tasked with counting the frequency of words in a given paragraph while excluding certain stop words (common words such as "the", "and", "is", etc.).

Input:
paragraph = "Python is a powerful programming language. Python is used for web development, data science, and artificial intelligence."
stop_words = ["is", "a", "for", "and"]

Output:
Word frequency (excluding stop words): {‘python’: 2, ‘powerful’: 1, ‘programming’: 1, ‘language.’: 1, ‘used’: 1, ‘web’: 1, ‘development,’: 1, ‘data’: 1, ‘science,’: 1, ‘artificial’: 1, ‘intelligence.’: 1}

[4]:
```python
paragraph = input("Enter paragraph: ")
size = int(input("\nEnter size for stop words: "))
stop_word = []
for i in range(size):
    w = input(f"Enter stop_word {i+1}: ")
    stop_word.append(w)

words = paragraph.lower().split()

freq = {}
for word in words:
    if word not in stop_word:
        if word in freq:
            freq[word] += 1
        else:
            freq[word] = 1

print("\nWord frequency (excluding stop words): ", freq)
```

```
Enter paragraph:  Python is a powerful programming language. Python is used for
web development, data science, and artificial intelligence.

Enter size for stop words:  4
Enter stop_word 1:  is
Enter stop_word 2:  a
Enter stop_word 3:  for
Enter stop_word 4:  end


Word frequency (excluding stop words):  {'python': 2, 'powerful': 1,
'programming': 1, 'language.': 1, 'used': 1, 'web': 1, 'development,': 1,
'data': 1, 'science,': 1, 'and': 1, 'artificial': 1, 'intelligence.': 1}
```

0.1.4 Problem Statement 4:-

You’re given a list of strings, and you need to find and return a list of characters that appear in every string in the list.

Input:
strings = [“apple”, “banana”, “orange”]

Output:
Common characters: [‘a’]

[7]:
```python
size = int(input("Enter size of list: "))
print()

strings = []
for i in range(size):
    word = input(f"Enter string {i+1}: ")
    strings.append(word)

common = []
for char in strings[0]:
    if all(char in word for word in strings[1:]):
        if char not in common:
            common.append(char)

print("\nCommon characters:", common)
```

```
Enter size of list:  3


Enter string 1:  apple
Enter string 2:  banana
Enter string 3:  orange


Common characters: ['a']
```

[ ]:
```

```

Practical-13

May 19, 2025

0.1 Practical 13 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

Suppose we have a text file named config.txt attached.

0.1.1 Problem Statement 1:-

Write a Python code to convert a dictionary containing configuration data that we want to write to a text file.

Example usage:
new_config = {
“username”: “new_user”,
“password”: “new_password”,
“database”: “new_db”,
“host”: “localhost”,
“port”: “5432”}

`config.txt`
username=old_user
password=old_pass
database=old_db
host=localhost
port=8080

[4]:
```python
# read config.txt and convert it into a dictionary
config_dict = {}

with open("config.txt", "r") as file:
    for line in file:
        if "=" in line:
            key, value = line.strip().split("=", 1)
            config_dict[key] = value

print("Config dictionary from file:", config_dict)

# write a dictionary to a file in key:value format
new_config = {
    "username": "new_user",
    "password": "new_password",
    "database": "new_db",
    "host": "localhost",
    "port": "5432"
}

with open("config.txt", "w") as file:
    for key, value in new_config.items():
        file.write(f"{key}={value}\n")

print("New Config dictionary written to config.txt")
```

```
Config dictionary from file: {'username': 'new_user', 'password':
'new_password', 'database': 'new_db', 'host': 'localhost', 'port': '5432'}
New Config dictionary written to config.txt
```

0.1.2 Problem Statement 2:-

Write a Python code to update a specific configuration parameter in an existing text file. You can achieve this by reading the existing configuration, updating the desired parameter, and then writing the updated configuration back to the file.

Example usage:
update_config_parameter('config.txt', 'password', 'new_password123')

[1]:
```
def update_config_parameter(filename, key_to_update, new_value):
    config = {}

    with open(filename, "r") as file:
        for line in file:
            if "=" in line:
                key, value = line.strip().split("=", 1)
                config[key] = value

    # update the desired parameter
    if key_to_update in config:
        config[key_to_update] = new_value
    else:
        print(f"Key '{key_to_update}' not found. Adding it.")
        config[key_to_update] = new_value

    # write updated parameter
    with open(filename, "w") as file:
        for key, value in config.items():
            file.write(f"{key}={value}\n")

    print(f"Updated '{key_to_update}' to '{new_value}' in {filename}")
```

```
filename = input("Enter filename: ")
parameter = input("Enter parameter: ")
value = input("Enter value to update: ")

update_config_parameter(filename, parameter, value)
```

```
Enter filename:  config.txt
Enter parameter:  password
Enter value to update:  new_password123
```

```
Updated 'password' to 'new_password123' in config.txt
```

0.1.3 Problem Statement 3:-

Suppose you have multiple configuration files. Write a Python code to merge them into a single configuration. You can achieve this by reading all the files, merging their configurations, and then writing the merged configuration to a new file.

[2]:
```
import os

config_dir = "configs"
merged_config = {}

for filename in os.listdir(config_dir):
    if filename.endswith(".txt"):
        filepath = os.path.join(config_dir, filename)
        with open(filepath, "r") as file:
            for line in file:
                if "=" in line:
                    key, value = line.strip().split("=", 1)
                    merged_config[key] = value  # Later files will override␣
 ↪earlier ones

with open("merged_config.txt", "w") as file:
    for key, value in merged_config.items():
        file.write(f"{key}={value}\n")

print("Merged configuration written to merged_config.txt")
```

```
Merged configuration written to merged_config.txt
```

0.1.4 Problem Statement 4:-

Write a Python code to delete a specific configuration parameter from a text file. You can achieve this by reading the existing configuration, removing the desired parameter, and then writing the updated configuration back to the file.

Example usage:
delete_config_parameter('config.txt', 'port')

[3]:
```
def delete_config_parameter(filename, key_to_delete):
    config = {}

    with open(filename, "r") as file:
        for line in file:
            if "=" in line:
                key, value = line.strip().split("=", 1)
                config[key] = value

    if key_to_delete in config:
        del config[key_to_delete]
        print(f"Deleted '{key_to_delete}' from {filename}")
    else:
        print(f"Key '{key_to_delete}' not found in {filename}")

    with open(filename, "w") as file:
        for key, value in config.items():
            file.write(f"{key}={value}\n")


filename = input("Enter filename: ")
parameter = input("Enter parameter to delete: ")

delete_config_parameter(filename, parameter)
```

```
Enter filename:  config.txt
Enter parameter to delete:  port
```

```
Deleted 'port' from config.txt
```

[ ]:

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]:
```python
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}")
```

```python
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:
```

```python
    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()
```

```python
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
```

```
[ ]:
```

Practical-15

May 19, 2025

0.1 Practical 15 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Create a base class Shape with methods to calculate area and perimeter. Then create subclasses Rectangle and Circle that inherit from Shape and implement their specific area and perimeter calculation methods.

[1]:
```python
import math

# Base class
class Shape:
    def area(self):
        return 0

    def perimeter(self):
        return 0

# Rectangle subclass
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

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

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

# Circle subclass
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius * self.radius

    def perimter(self):
        return 2 * math.pi * self.radius

print("Choose shape:-")
print("Enter 1 for Rectangle, or 2 for Circle.")
choice = int(input("Enter choice: "))

if choice == 1:
    w = float(input("Enter width: "))
    h = float(input("Enter height: "))
    rect = Rectangle(w, h)
    print(f"Rectangle Area: {rect.area()}")
    print(f"Rectangle Perimeter: {rect.perimeter()}")

elif choice == '2':
    r = float(input("Enter radius: "))
    circ = Circle(r)
    print(f"Circle Area: {circ.area():.2f}")
    print(f"Circle Perimeter: {circ.perimeter():.2f}")

else:
    print("Invalid choice.")
```

```
Choose shape:-
Enter 1 for Rectangle, or 2 for Circle.

Enter choice:  1
Enter width:  15
Enter height:  14

Rectangle Area: 210.0
Rectangle Perimeter: 58.0
```

0.1.2 Problem Statement 2:-

Create a base class Vehicle with attributes like make, model, and year, and a method to display vehicle information. Then create subclasses of Car and Motorcycle that are inherited from the Vehicle and add their specific attributes like seating capacity or type of engine.

[1]:
```
# Base class
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print(f"Make: {self.make}")
        print(f"Model: {self.model}")
        print(f"Year: {self.year}")

# Car subclass
class Car(Vehicle):
    def __init__(self, make, model, year, seating_capacity):
        super().__init__(make, model, year)
        self.seating_capacity = seating_capacity

    def display_info(self):
        super().display_info()
        print(f"Seating Capacity: {self.seating_capacity}")

# Motorcycle subclass
class Motorcycle(Vehicle):
    def __init__(self, make, model, year, engine_type):
        super().__init__(make, model, year)
        self.engine_type = engine_type

    def display_info(self):
        super().display_info()
        print(f"Engine Type: {self.engine_type}")

print("Choose Vehicle Type:-")
print("Enter 1 for Car, or 2 for Motorcycle.")
choice = int(input("Enter choice: "))

if choice == 1:
    make = input("Enter make: ")
    model = input("Enter model: ")
    year = input("Enter year: ")
    capacity = input("Enter seating capacity: ")
    car = Car(make, model, year, capacity)
    print("\nCar Details:")
    car.display_info()

elif choice == 2:
    make = input("Enter make: ")
    model = input("Enter model: ")
    year = input("Enter year: ")
    engine = input("Enter engine type: ")
    bike = Motorcycle(make, model, year, engine)
    print("\nMotorcycle Details:")
    bike.display_info()

else:
    print("Invalid choice.")
```

```
Choose Vehicle Type:-
Enter 1 for Car, or 2 for Motorcycle.

Enter choice:  1
Enter make:  Tata
Enter model:  Safari XZ Plus
Enter year:  2024
Enter seating capacity:  7


Car Details:
Make: Tata
Model: Safari XZ Plus
Year: 2024
Seating Capacity: 7
```

0.1.3 Problem Statement 3:-

Create a base class Employee with attributes like name, ID, and salary, and methods to calculate and display details. Then create subclasses Manager and Developer that inherit from Employee and add attributes like department or programming language.

[3]:
```
# Base class
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name
        self.emp_id = emp_id
        self.salary = salary

    def display_details(self):
        print(f"Name      : {self.name}")
        print(f"ID        : {self.emp_id}")
        print(f"Salary    :  {self.salary}")

# Subclass: Manager
class Manager(Employee):
    def __init__(self, name, emp_id, salary, department):
        super().__init__(name, emp_id, salary)
        self.department = department

    def display_details(self):
        super().display_details()
        print(f"Department: {self.department}")

# Subclass: Developer
class Developer(Employee):
    def __init__(self, name, emp_id, salary, language):
        super().__init__(name, emp_id, salary)
        self.language = language

    def display_details(self):
        super().display_details()
        print(f"Programming Language: {self.language}")

# Sample usage
print("Select Employee Type:")
print("1. Manager")
print("2. Developer")
choice = input("Enter 1 or 2: ")

name = input("Enter name: ")
emp_id = input("Enter ID: ")
salary = float(input("Enter salary: "))

if choice == '1':
    dept = input("Enter department: ")
    m = Manager(name, emp_id, salary, dept)
    print("\nManager Details:")
    m.display_details()

elif choice == '2':
    lang = input("Enter programming language: ")
    d = Developer(name, emp_id, salary, lang)
    print("\nDeveloper Details:")
    d.display_details()

else:
    print("Invalid choice.")
```

```
Select Employee Type:
1. Manager
2. Developer
```

```
Enter 1 or 2:  2
Enter name:  Sarthak
Enter ID:  DEV51
Enter salary:  2165000
Enter programming language:  Python, Java, C, C++, Kotlin, Dart, Bash


Developer Details:
Name      : Sarthak
ID        : DEV51
Salary    :  2165000.0
Programming Language: Python, Java, C, C++, Kotlin, Dart, Bash
```

0.1.4 Problem Statement 4:-

Create a base class Animal with methods to make sound and display species. Then create subclasses Dog, Cat, and Bird that inherit from Animal and implement their specific sound methods.

[5]:
```python
# Base class
class Animal:
    def __init__(self, species):
        self.species = species

    def make_sound(self):
        print("Some animal sound.")

    def display_species(self):
        print(f"Species: {self.species}")

# Dog subclass
class Dog(Animal):
    def __init__(self):
        super().__init__("Dog")

    def make_sound(self):
        print("Woof! Woof!")

# Cat subclass
class Cat(Animal):
    def __init__(self):
        super().__init__("Cat")

    def make_sound(self):
        print("Meow!")

# Bird subclass
class Bird(Animal):
    def __init__(self):
        super().__init__("Bird")

    def make_sound(self):
        print("Tweet! Tweet!")

# Sample usage
print("Choose Animal:")
print("Enter 1 for Dog, 2 for Cat, or 3 for Bird.")
choice = input("Enter choice: ")

if choice == '1':
    a = Dog()
elif choice == '2':
    a = Cat()
elif choice == '3':
    a = Bird()
else:
    print("Invalid choice.")
    exit()

print("\nAnimal Details:-")
a.display_species()
a.make_sound()
```

```
Choose Animal:
Enter 1 for Dog, 2 for Cat, or 3 for Bird.

Enter choice:  2


Animal Details:-
Species: Cat
Meow!
```

0.1.5 Problem Statement 5:-

Create a base class Account with methods for deposit, withdraw, and display balance. Then create subclasses SavingsAccount and CheckingAccount that inherit from Account and implement their specific interest calculation or overdraft protection methods.

[11]:
```
# Base class
class Account:
    def __init__(self, acc_number, holder_name, balance=0):
        self.acc_number = acc_number
        self.holder_name = holder_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f" {amount} deposited.")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f" {amount} withdrawn.")
        else:
            print("Insufficient balance.")

    def display_balance(self):
        print(f"Account Holder: {self.holder_name}")
        print(f"Account Number: {self.acc_number}")
        print(f"Balance:  {self.balance}")
```

```python
# SavingsAccount subclass
class SavingsAccount(Account):
    def __init__(self, acc_number, holder_name, balance=0, interest_rate=0.05):
        super().__init__(acc_number, holder_name, balance)
        self.interest_rate = interest_rate

    def apply_interest(self):
        interest = self.balance * self.interest_rate
        self.balance += interest
        print(f"Interest of  {interest:.2f} applied at {self.
 ↪interest_rate*100}% rate.")

# CheckingAccount subclass
class CheckingAccount(Account):
    def __init__(self, acc_number, holder_name, balance=0,␣
 ↪overdraft_limit=1000):
        super().__init__(acc_number, holder_name, balance)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount <= self.balance + self.overdraft_limit:
            self.balance -= amount
            print(f" {amount} withdrawn (with overdraft if needed).")
        else:
            print("Withdrawal exceeds overdraft limit.")


print("Choose account type:-")
print("Enter 1. for Savings Account, or 2. for Checking Account.")
choice = int(input("Enter choice: "))

acc_number = input("Enter account number: ")
holder_name = input("Enter account holder name: ")
initial_balance = float(input("Enter initial balance: "))

if choice == 1:
    acc = SavingsAccount(acc_number, holder_name, initial_balance)
    acc.deposit(500)
    acc.apply_interest()
    acc.withdraw(200)
    print("\nAccount Summary:")
    acc.display_balance()

elif choice == 2:
    acc = CheckingAccount(acc_number, holder_name, initial_balance)
    acc.deposit(300)
    acc.withdraw(1500)  # Test overdraft
    print("\nAccount Summary:")
    acc.display_balance()

else:
    print("Invalid account type.")
```

```
Choose account type:-
Enter 1. for Savings Account, or 2. for Checking Account.

Enter choice:  2
Enter account number:  007
Enter account holder name:  James Bond
Enter initial balance:  700

 300 deposited.
 1500 withdrawn (with overdraft if needed).

Account Summary:
Account Holder: James Bond
Account Number: 007
Balance:  -500.0
```

```
[ ]:
```

Practical-16

May 19, 2025

0.1 Practical 16 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Employee Management System:
Create a base class Employee with public, private, and protected attributes representing employee details such as name, employee ID, and salary. Implement methods to display employee details and calculate salary. Then create subclasses Manager and Developer that inherit from Employee and demonstrate access to different types of attributes and methods.

[8]:
```python
# Base class
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name                  # public
        self.__emp_id = emp_id            # private
        self._salary = salary             # protected

    def display_details(self):
        print("Name:", self.name)
        print("Employee ID:", self.__emp_id)
        print("Salary:", self._salary)

    def calculate_salary(self):
        return self._salary

# Manager subclass
class Manager(Employee):
    def __init__(self, name, emp_id, salary, bonus):
        super().__init__(name, emp_id, salary)
        self.bonus = bonus

    def total_salary(self):
        return self._salary + self.bonus  # accessing protected attribute

# Developer subclass
class Developer(Employee):
    def __init__(self, name, emp_id, salary, project):
        super().__init__(name, emp_id, salary)
        self.project = project

    def show_project(self):
        print(f"{self.name} is working on {self.project}")


m1 = Manager("Raj", "M123", 70000, 10000)
m1.display_details()
print("Total Salary:", m1.total_salary())

print()
d1 = Developer("Dhoni", "D456", 60000, "AI Tool")
d1.display_details()
d1.show_project()
```

```
Name: Raj
Employee ID: M123
Salary: 70000
Total Salary: 80000

Name: Dhoni
Employee ID: D456
Salary: 60000
Dhoni is working on AI Tool
```

0.1.2 Problem Statement 2:-

Bank Account System:
Create a base class Account with public, private, and protected attributes representing account details such as account number, balance, and interest rate. Implement methods to deposit, withdraw, and calculate interest. Then create subclasses SavingsAccount and CheckingAccount that inherit from Account and demonstrate access to different types of attributes and methods.

[9]:
```
# Base class
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name                # public
        self.__emp_id = emp_id          # private
        self._salary = salary           # protected

    def display_details(self):
        print("Name:", self.name)
        print("Employee ID:", self.__emp_id)
        print("Salary:", self._salary)

    def calculate_salary(self):
```

```python
        return self._salary

# Manager subclass
class Manager(Employee):
    def __init__(self, name, emp_id, salary, bonus):
        super().__init__(name, emp_id, salary)
        self.bonus = bonus

    def total_salary(self):
        return self._salary + self.bonus  # accessing protected attribute

# Developer subclass
class Developer(Employee):
    def __init__(self, name, emp_id, salary, project):
        super().__init__(name, emp_id, salary)
        self.project = project

    def show_project(self):
        print(f"{self.name} is working on {self.project}")


m1 = Manager("Raj", "M123", 70000, 10000)
m1.display_details()
print("Total Salary:", m1.total_salary())

print()
d1 = Developer("Dhoni", "D456", 60000, "AI Tool")
d1.display_details()
d1.show_project()
```

```
Name: Raj
Employee ID: M123
Salary: 70000
Total Salary: 80000

Name: Dhoni
Employee ID: D456
Salary: 60000
Dhoni is working on AI Tool
```

0.1.3 Problem Statement 3:-

Vehicle Management System:
Create a base class Vehicle with public, private, and protected attributes representing vehicle details such as make, model, and year. Implement methods to display vehicle information. Then create subclasses Car and Motorcycle that inherit from Vehicle and demonstrate access to different types of attributes and methods.

[10]:
```python
# Base class
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make            # public
        self._model = model         # protected
        self.__year = year          # private

    def display_info(self):
        print("Make:", self.make)
        print("Model:", self._model)
        print("Year:", self.__year)

# Subclass Car
class Car(Vehicle):
    def __init__(self, make, model, year, doors):
        super().__init__(make, model, year)
        self.doors = doors

    def show_car(self):
        print(f"Car has {self.doors} doors and model is {self._model}")

# Subclass Motorcycle
class Motorcycle(Vehicle):
    def __init__(self, make, model, year, cc):
        super().__init__(make, model, year)
        self.cc = cc

    def show_motorcycle(self):
        print(f"Motorcycle has {self.cc}cc and make is {self.make}")


print("Vehicle Management :-\n")
car1 = Car("Mahindra", "XUV700", 2024, 7)
car1.display_info()
car1.show_car()

print()
bike1 = Motorcycle("Bajaj Motors", "Avenger", 2015, 220)
bike1.display_info()
bike1.show_motorcycle()
```

```
Vehicle Management :-

Make: Mahindra
Model: XUV700
Year: 2024
Car has 7 doors and model is XUV700
```

```
Make: Bajaj Motors
Model: Avenger
Year: 2015
Motorcycle has 220cc and make is Bajaj Motors
```

0.1.4 Problem Statement 4:-

School Management System:
Create a base class Student with public, private, and protected attributes representing student details such as name, roll number, and marks. Implement methods to display student details and calculate grades. Then create subclasses ClassMonitor and Topper that inherit from Student and demonstrate access to different types of attributes and methods.

[11]:
```python
# Base class
class Student:
    def __init__(self, name, roll, marks):
        self.name = name              # public
        self._marks = marks           # protected
        self.__roll = roll            # private

    def display_details(self):
        print("Name:", self.name)
        print("Roll No:", self.__roll)
        print("Marks:", self._marks)

    def calculate_grade(self):
        if self._marks >= 90:
            return "A"
        elif self._marks >= 75:
            return "B"
        elif self._marks >= 60:
            return "C"
        else:
            return "D"

# Subclass ClassMonitor
class ClassMonitor(Student):
    def show_role(self):
        print(f"{self.name} is the class monitor with grade {self.
 ↪calculate_grade()}.")

# Subclass Topper
class Topper(Student):
    def praise(self):
        if self._marks >= 90:
            print(f"{self.name} is the Topper of the class!")
print("School Management :-\n")
s1 = ClassMonitor("Roshan", 101, 78)
s1.display_details()
s1.show_role()

print()
s2 = Topper("Pankaj", 102, 95)
s2.display_details()
s2.praise()
```

```
School Management :-

Name: Roshan
Roll No: 101
Marks: 78
Roshan is the class monitor with grade B.

Name: Pankaj
Roll No: 102
Marks: 95
Pankaj is the Topper of the class!
```

0.1.5 Problem Statement 5:-

Product Inventory Management:
Create a base class Product with public, private, and protected attributes representing product details such as name, ID, and price. Implement methods to display product information. Then create subclasses ElectronicProduct and ClothingProduct that inherit from Product and demonstrate access to different types of attributes and methods.

[13]:
```
# Base class
class Product:
    def __init__(self, name, pid, price):
        self.name = name            # public
        self._price = price         # protected
        self.__pid = pid            # private

    def show_info(self):
        print("Product Name:", self.name)
        print("Product ID:", self.__pid)
        print("Price:", self._price)

# Subclass ElectronicProduct
class ElectronicProduct(Product):
    def __init__(self, name, pid, price, warranty):
        super().__init__(name, pid, price)
        self.warranty = warranty
```

```python
    def show_warranty(self):
        print(f"Warranty: {self.warranty} years")

# Subclass ClothingProduct
class ClothingProduct(Product):
    def __init__(self, name, pid, price, size):
        super().__init__(name, pid, price)
        self.size = size

    def show_size(self):
        print(f"Clothing Size: {self.size}")

print("Product Inventory :-\n")
e1 = ElectronicProduct("Laptop", "E101", 50000, 2)
e1.show_info()
e1.show_warranty()

print()
c1 = ClothingProduct("T-Shirt", "C202", 599, "L")
c1.show_info()
c1.show_size()
```

```
Product Inventory :-

Product Name: Laptop
Product ID: E101
Price: 50000
Warranty: 2 years

Product Name: T-Shirt
Product ID: C202
Price: 599
Clothing Size: L
```

Practical-17

May 19, 2025

0.1 Practical 17 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Inventory Management System:
A retail company needs a software solution to streamline inventory management. The system should enable employees to add new products to the inventory, specifying details such as the product name, price, and quantity available. Additionally, the system should support an operation to combine two products, calculating the average price and total quantity of the combined product.
Python Feature: This solution will utilize Python classes to represent products and operator overloading to perform calculations when combining products.

[2]:
```
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def display(self):
        print(f"Product: {self.name}")
        print(f"Price:  {self.price}")
        print(f"Quantity: {self.quantity}\n")

    # Overload '+' to combine two products
    def __add__(self, other):
        if self.name != other.name:
            raise ValueError("Cannot combine products with different names.")

        total_quantity = self.quantity + other.quantity
        avg_price = (self.price * self.quantity + other.price * other.quantity)␣
 ↪/ total_quantity
        return Product(self.name, round(avg_price, 2), total_quantity)


print("Add two products to combine them:-\n")
```

```python
# First product
name1 = input("Enter product name: ")
price1 = float(input("Enter product price: "))
qty1 = int(input("Enter product quantity: "))
product1 = Product(name1, price1, qty1)

# Second product
name2 = input("\nEnter another product name (must be same to combine): ")
price2 = float(input("Enter product price: "))
qty2 = int(input("Enter product quantity: "))
product2 = Product(name2, price2, qty2)

# Display original products
print("\nProduct 1 :-")
product1.display()

print("Product 2 :-")
product2.display()

# Combine products
try:
    combined = product1 + product2
    print("Combined Product :-")
    combined.display()
except ValueError as e:
    print(f"Error: {e}")
```

```
Add two products to combine them:-


Enter product name:  Sony PlayStation 5
Enter product price:  56000
Enter product quantity:  12

Enter another product name (must be same to combine):  Sony PlayStation 5
Enter product price:  44000
Enter product quantity:  18


Product 1 :-
Product: Sony PlayStation 5
Price:  56000.0
Quantity: 12

Product 2 :-
Product: Sony PlayStation 5
Price:  44000.0
Quantity: 18
```

```
Combined Product :-
Product: Sony PlayStation 5
Price:  48800.0
Quantity: 30
```

0.1.2 Problem Statement 2:-

Student Record Management:
A school administration requires a digital tool to efficiently manage student records. The program should allow administrators to create individual student profiles, including information such as the student's name, age, and academic grades for different subjects. Moreover, the system should provide functionality to dynamically add new grades for specific subjects and retrieve grades for any subject as needed.
Python Feature: Python classes will be used to represent student profiles, with methods to add and retrieve grades dynamically.

[6]:
```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.grades = {}  # subject:grade pairs

    def add_grade(self, subject, grade):
        self.grades[subject] = grade
        print(f"Grade added: {subject} = {grade}")

    def get_grade(self, subject):
        return self.grades.get(subject, "Grade not found for this subject.")

    def display_profile(self):
        print(f"\nStudent Name: {self.name}")
        print(f"Age: {self.age}")
        print("Grades:")
        if self.grades:
            for subject, grade in self.grades.items():
                print(f"  {subject}: {grade}")
        else:
            print("  No grades available.")


print("Create a Student Profile:")
name = input("Enter student's name: ")
age = int(input("Enter student's age: "))

student = Student(name, age)
```

```python
while True:
    print("\nOptions:")
    print("1. Add Grade")
    print("2. Get Grade for Subject")
    print("3. Display Student Profile")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == '1':
        subject = input("Enter subject name: ")
        grade = input("Enter grade: ")
        student.add_grade(subject, grade)

    elif choice == '2':
        subject = input("Enter subject name to retrieve grade: ")
        print(f"{subject}: {student.get_grade(subject)}")

    elif choice == '3':
        student.display_profile()

    elif choice == '4':
        print("Program terminated successfully!")
        break

    else:
        print("Invalid choice. Please enter a number from 1 to 4.")
```

```
Create a Student Profile:

Enter student's name:  Thomas Edison
Enter student's age:  21


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  1
Enter subject name:  DSA
Enter grade:  A

Grade added: DSA = A


Options:
1. Add Grade
2. Get Grade for Subject
```

```
3. Display Student Profile
4. Exit

Enter your choice (1-4):  1
Enter subject name:  OOPC
Enter grade:  B

Grade added: OOPC = B


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  1
Enter subject name:  OS
Enter grade:  A

Grade added: OS = A


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  2
Enter subject name to retrieve grade:  OOPC

OOPC: B


Options:
1. Add Grade
2. Get Grade for Subject
3. Display Student Profile
4. Exit

Enter your choice (1-4):  3


Student Name: Thomas Edison
Age: 21
Grades:
  DSA: A
  OOPC: B
  OS: A


Options:
1. Add Grade
2. Get Grade for Subject
```

```
3. Display Student Profile
4. Exit

Enter your choice (1-4):  4

Program terminated successfully!
```

0.1.3 Problem Statement 3:-

Bank Account Management:
A banking institution seeks a software solution to handle customer accounts effectively. The program should facilitate the creation and management of bank accounts, storing details such as the account number, current balance, and account holder's name. Furthermore, the system should support deposit operations by overloading the addition operator, enabling customers to deposit funds into their accounts seamlessly.
Python Feature: Python classes will be used to represent bank accounts, and operator overloading will be employed to enable deposit operations.

[1]:
```python
class BankAccount:
    def __init__(self, account_number, holder_name, balance=0.0):
        self.account_number = account_number
        self.holder_name = holder_name
        self.balance = balance

    def display_account(self):
        print(f"\nAccount Holder: {self.holder_name}")
        print(f"Account Number: {self.account_number}")
        print(f"Current Balance:  {self.balance:.2f}")

    # Overload '+' operator to perform deposit
    def __add__(self, amount):
        if isinstance(amount, (int, float)) and amount > 0:
            self.balance += amount
            print(f" {amount:.2f} deposited successfully.")
        else:
            print("Invalid deposit amount.")
        return self


print("Create a Bank Account:-\n")
acc_no = input("Enter account number: ")
name = input("Enter account holder name: ")
initial_balance = float(input("Enter initial balance: "))

account = BankAccount(acc_no, name, initial_balance)

while True:
    print("\nOptions:")
    print("1. Display Account")
    print("2. Deposit Amount")
    print("3. Exit")

    choice = input("Enter your choice (1-3): ")

    if choice == '1':
        account.display_account()

    elif choice == '2':
        amount = float(input("Enter amount to deposit: "))
        account + amount

    elif choice == '3':
        print("Program terminated successfully!")
        break

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

```
Create a Bank Account:-


Enter account number:  007
Enter account holder name:  James Bond
Enter initial balance:  700


Options:
1. Display Account
2. Deposit Amount
3. Exit

Enter your choice (1-3):  1


Account Holder: James Bond
Account Number: 007
Current Balance:  700.00

Options:
1. Display Account
2. Deposit Amount
3. Exit

Enter your choice (1-3):  2
Enter amount to deposit:  1250

 1250.00 deposited successfully.

Options:
```

```
1. Display Account
2. Deposit Amount
3. Exit

Enter your choice (1-3):  1


Account Holder: James Bond
Account Number: 007
Current Balance:  1950.00

Options:
1. Display Account
2. Deposit Amount
3. Exit

Enter your choice (1-3):  3

Program terminated successfully!
```

0.1.4 Problem Statement 4:-

Social Media Account Management:
A social networking platform aims to develop a tool for managing user profiles efficiently. The program should allow users to create and customize their profiles with information like their username and a brief bio. Additionally, the system should enable users to gain followers, with functionality to increment the follower count dynamically as users interact with each other's profiles.
Python Feature: Python classes will represent user profiles, with methods to customize profiles and dynamically manage follower counts.

[4]:
```python
class UserProfile:
    def __init__(self, username, bio):
        self.username = username
        self.bio = bio
        self.followers = 0

    def update_bio(self, new_bio):
        self.bio = new_bio
        print(f"Bio updated to: {self.bio}")

    def add_follower(self):
        self.followers += 1
        print(f"{self.username} gained a follower! Total followers: {self.
 ↪followers}")

    def display_profile(self):
        print("\nUser Profile :-")
        print(f"Username: {self.username}")
        print(f"Bio: {self.bio}")
        print(f"Followers: {self.followers}")
```

```
# Example usage
user1 = UserProfile("james_bond", "Just another tech lover!")
user1.display_profile()
user1.add_follower()
user1.add_follower()
user1.update_bio("Python Developer | Open Source Enthusiast")
user1.display_profile()
```

```
User Profile :-
Username: james_bond
Bio: Just another tech lover!
Followers: 0
james_bond gained a follower! Total followers: 1
james_bond gained a follower! Total followers: 2
Bio updated to: Python Developer | Open Source Enthusiast

User Profile :-
Username: james_bond
Bio: Python Developer | Open Source Enthusiast
Followers: 2
```

0.1.5 Problem Statement 5:-

Shopping Cart Implementation:
An online marketplace requires a robust shopping cart system to enhance the shopping experience for its customers. The system should allow users to add items to their carts while browsing products, specifying details such as the item name, price, and desired quantity. Upon checkout, the system should display a summary of the items in the cart, including their respective details, facilitating a seamless shopping process.
Python Feature: Python classes will be used to represent items and the shopping cart, allowing for easy addition and retrieval of item details.

[6]:
```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price


class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, product, quantity):
        self.items.append((product, quantity))
        print(f"Added {quantity}x {product.name} to the cart.")
    def checkout(self):
        print("\nShopping Cart Summary:-")
        total = 0
        for product, quantity in self.items:
            item_total = product.price * quantity
            print(f"{product.name} - {product.price} x {quantity} =␣
↪{item_total}")
            total += item_total
        print(f"Total Amount: {total}")


p1 = Product("Laptop", 50000)
p2 = Product("Mouse", 500)
p3 = Product("Keyboard", 1500)

cart = ShoppingCart()
cart.add_item(p1, 1)
cart.add_item(p2, 2)
cart.add_item(p3, 1)

cart.checkout()
```

```
Added 1x Laptop to the cart.
Added 2x Mouse to the cart.
Added 1x Keyboard to the cart.

Shopping Cart Summary:-
Laptop - 50000 x 1 = 50000
Mouse - 500 x 2 = 1000
Keyboard - 1500 x 1 = 1500
Total Amount: 52500
```

```
[ ]:
```

Practical-18

May 19, 2025

0.1 Practical 18:-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

RegEx Practicals :- **1.(A)** Write a Python program to check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9).

[1]:
```python
import re

def check_alphanumeric(s):
    return bool(re.fullmatch(r'[a-zA-Z0-9]+', s))

print(check_alphanumeric("Python123"))
print(check_alphanumeric("Hello_World"))
```

```
True
False
```

1.(B) Write a Python program that matches a string that has an a followed by one or more b's.

[2]:
```python
def match_ab1(s):
    return bool(re.fullmatch(r'ab+', s))

print(match_ab1("ab"))
print(match_ab1("abbbbbb"))
print(match_ab1("a"))
```

```
True
True
False
```

1.(C) Write a Python program that matches a string that has an a followed by two to three 'b'.

[4]:
```python
def match_ab2to3(s):
    return bool(re.fullmatch(r'ab{2,3}', s))

print(match_ab2to3("abb"))
print(match_ab2to3("abbb"))
print(match_ab2to3("abbbb"))
```

```
True
True
False
```

1.(D) Write a Python program that matches a string that has an ‘a’ followed by anything, ending in ‘b’.

[5]:
```
def match_a_any_b(s):
    return bool(re.fullmatch(r'a.*b', s))

print(match_a_any_b("acb"))
print(match_a_any_b("a123b"))
print(match_a_any_b("ab"))
```

```
True
True
True
```

1.(E) Write a Python program that matches a word containing ‘z’.

[6]:
```
def contains_z(s):
    return bool(re.search(r'\bz\w*|\w*z\b|\w*z\w*', s))

print(contains_z("amazing"))
print(contains_z("hello"))
```

```
True
False
```

1.(F) Write a Python program where a string will start with a specific number.

[8]:
```
def starts_with_number(s, num):
    pattern = f"^{num}"
    return bool(re.match(pattern, s))

print(starts_with_number("5hello", 5))
print(starts_with_number("9test", 5))
```

```
True
False
```

1.(G) Write a Python program to search the numbers (0-9) of length between 1 to 3 in a given string.

[9]:
```
def search_numbers(s):
    return re.findall(r'\b\d{1,3}\b', s)
```

```
print(search_numbers("He paid 5 for 120 pens and 9999 for laptops."))
```

```
['5', '120']
```

0.1.2 Problem Statement 2:-

Write a Python program to find the substrings within a string.
Sample Text: 'Python exercises, PHP exercises, C# exercises'
Pattern: 'exercises'

[11]:
```
import re

text = input("Enter text: ")
words = re.findall(r'\b\w+\b', text)
counts = {}
for word in words:
    counts[word] = counts.get(word, 0) + 1

repeated = [word for word, count in counts.items() if count > 1]

if repeated:
    print("Pattern(s):-")
    for pat in repeated:
        print(pat)
else:
    print("No pattern found.")
```

```
Enter text:  Python exercises, PHP exercises, C# exercises
```

```
Pattern(s):-
exercises
```

0.1.3 Problem Statement 3:-

Extracting Email Addresses:
Problem Statement: Given a large text file containing various lines of data, extract all the email addresses present in the file.
Example: "Contact us at support@example.com or sales@example.org for more information."
It should yield `["support@example.com", "sales@example.org"]`.

[19]:
```
import re

def extract_emails(filename):
    emails = []
    email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

    with open(filename, 'r') as file:
        for line in file:
            found_emails = re.findall(email_pattern, line)
            emails.extend(found_emails)
    return emails

filename = input("Enter the filename: ")
result = extract_emails(filename)
print(result)
```

```
Enter the filename:  email.txt

['support@example.com', 'sales@example.org', 'marketing@mycompany.net', 'hr-
dept@company.co.uk', 'hr.manager@company.co.uk', 'intern123@university.edu']
```

0.1.4 Problem Statement 4:-

Validating Phone Numbers:
Problem Statement: Create a function to validate whether a given string is a valid phone number in the format (XXX) XXX-XXXX or XXX-XXX-XXXX.
Example: "Phone number (123) 456-7890 is valid, but 123-45-67890 is not." It should yield True for the first phone number and False for the second.

[20]:
```
import re

def validate_phone_number(phone):
    pattern = r'^(\(\d{3}\) |\d{3}-)\d{3}-\d{4}$'
    if re.match(pattern, phone):
        return True
    else:
        return False

print(validate_phone_number("(123) 456-7890"))
print(validate_phone_number("123-456-7890"))
print(validate_phone_number("123-45-67890"))
print(validate_phone_number("1234567890"))
```

```
True
True
False
False
```

0.1.5 Problem Statement 5:-

Finding Dates:
Problem Statement: Write a script to find all the dates in a text document, where dates are in the format DD/MM/YYYY.
Example: "Important dates are 12/05/2021 and 23/08/2022."
It should yield ["12/05/2021", "23/08/2022"].

[26]:
```
import re
```

```
def find_dates(text):
    pattern = r'\b(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[0-2])/(\d{4})\b'
    dates = re.findall(pattern, text)
    extracted_dates = [f"{day}/{month}/{year}" for day, month, year in dates]
    return extracted_dates

text = input("Enter text: ")
print(find_dates(text))
```

```
Enter text:  Important dates are 12/05/2021 and 23/08/2022.
```

```
['12/05/2021', '23/08/2022']
```

0.1.6 Problem Statement 6:-

Replacing URLs:
Problem Statement: Given a string containing multiple URLs, replace all the URLs with a placeholder text [LINK].
Example: "Visit https://example.com or http://example.org for more info."
It should yield "Visit [LINK] or [LINK] for more info.".

[27]:
```
import re

def replace_urls(text):
    url_pattern = r'https?://[^\s]+'
    replaced_text = re.sub(url_pattern, '[LINK]', text)
    return replaced_text

text = input("Enter text: ")
print(replace_urls(text))
```

```
Enter text:  Visit https://example.com or http://example.org for more info.
```

```
Visit [LINK] or [LINK] for more info.
```

0.1.7 Problem Statement 7:-

Extracting Hashtags:
Problem Statement: Extract all the hashtags from a social media post.
Example: "Loving the #sunshine and #beachlife!"
It should yield ["#sunshine", "#beachlife"].

[28]:
```
import re

def extract_hashtags(text):
    pattern = r'#\w+'
    hashtags = re.findall(pattern, text)
    return hashtags
```

```python
post = input("Enter post caption: ")
print(extract_hashtags(post))
```

```
Enter post caption:  Loving the #sunshine and #beachlife!
```

```
['#sunshine', '#beachlife']
```

[ ]:

```

```

Practical-19

May 19, 2025

0.1 Practical 19 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Division with Multiple Exception Types:
Write a function safe_divide(a, b) that takes two arguments and performs division. Handle the following exceptions:
`ZeroDivisionError`: When b is zero, print "Cannot divide by zero" and return None.
`TypeError`: When either a or b is not a number, print "Invalid input type" and return None.

[1]:
```
def safe_divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Cannot divide by zero")
        return None
    except TypeError:
        print("Invalid input type")
        return None

print(safe_divide(10, 2))
print(safe_divide(10, 0))
print(safe_divide(10, "a"))
```

```
5.0
Cannot divide by zero
None
Invalid input type
None
```

0.1.2 Problem Statement 2:-

File Processing with Specific Exceptions:
Create a function process_file(filename) that attempts to open and read a file. Handle exceptions for:
`FileNotFoundError`: Print "File not found"

PermissionError: Print “Permission denied”
IOError: Print “An IOError occurred”.

```
[2]: def process_file(filename):
         try:
             with open(filename, 'r') as file:
                 content = file.read()
                 print(content)
         except FileNotFoundError:
             print("File not found")
         except PermissionError:
             print("Permission denied")
         except IOError:
             print("An IOError occurred")

     filename = input("Enter the filename: ")
     process_file(filename)
```

```
Enter the filename:  samplefile.txt
```

```
File not found
```

0.1.3 Problem Statement 3:-

Custom Exception for Age Validation:
Define a custom exception InvalidAgeError and write a function validate_age(age) that raises this exception if the age is not between 0 and 120 (inclusive). Use this function to check a list of ages and handle the exception by printing an appropriate message.

```
[3]: class InvalidAgeError(Exception):
         pass

     def validate_age(age):
         if age < 0 or age > 120:
             raise InvalidAgeError(f"Invalid age: {age}. Age must be between 0 and␣
       ↪120.")

     ages = [25, -5, 130, 50, 0, 120]

     for age in ages:
         try:
             validate_age(age)
             print(f"Age {age} is valid.")
         except InvalidAgeError as e:
             print(e)
```

```
Age 25 is valid.
Invalid age: -5. Age must be between 0 and 120.
Invalid age: 130. Age must be between 0 and 120.
```

```
Age 50 is valid.
Age 0 is valid.
Age 120 is valid.
```

0.1.4 Problem Statement 4:-

Nested Exception Handling:
Write a function nested_exception_handling() that performs the following:
Tries to open a file and read an integer from it.
Catches exceptions for file-related errors (`FileNotFoundError`, `IOError`).
Within the same try block, convert the read value to an integer and handle potential ValueError.
Print specific error messages for each exception and ensure the file is closed properly using a final block.

[4]:
```python
def nested_exception_handling(filename):
    try:
        file = open(filename, 'r')
        try:
            data = file.read()
            number = int(data)
            print(f"Read integer: {number}")
        except ValueError:
            print("Error: The file does not contain a valid integer.")
        finally:
            file.close()
    except FileNotFoundError:
        print("Error: File not found.")
    except IOError:
        print("Error: An IOError occurred.")

filename = input("Enter the filename: ")
nested_exception_handling(filename)
```

```
Enter the filename:  sample.txt

Error: The file does not contain a valid integer.
```

[ ]: