Sardar Vallabhbhai National Institute of Technology

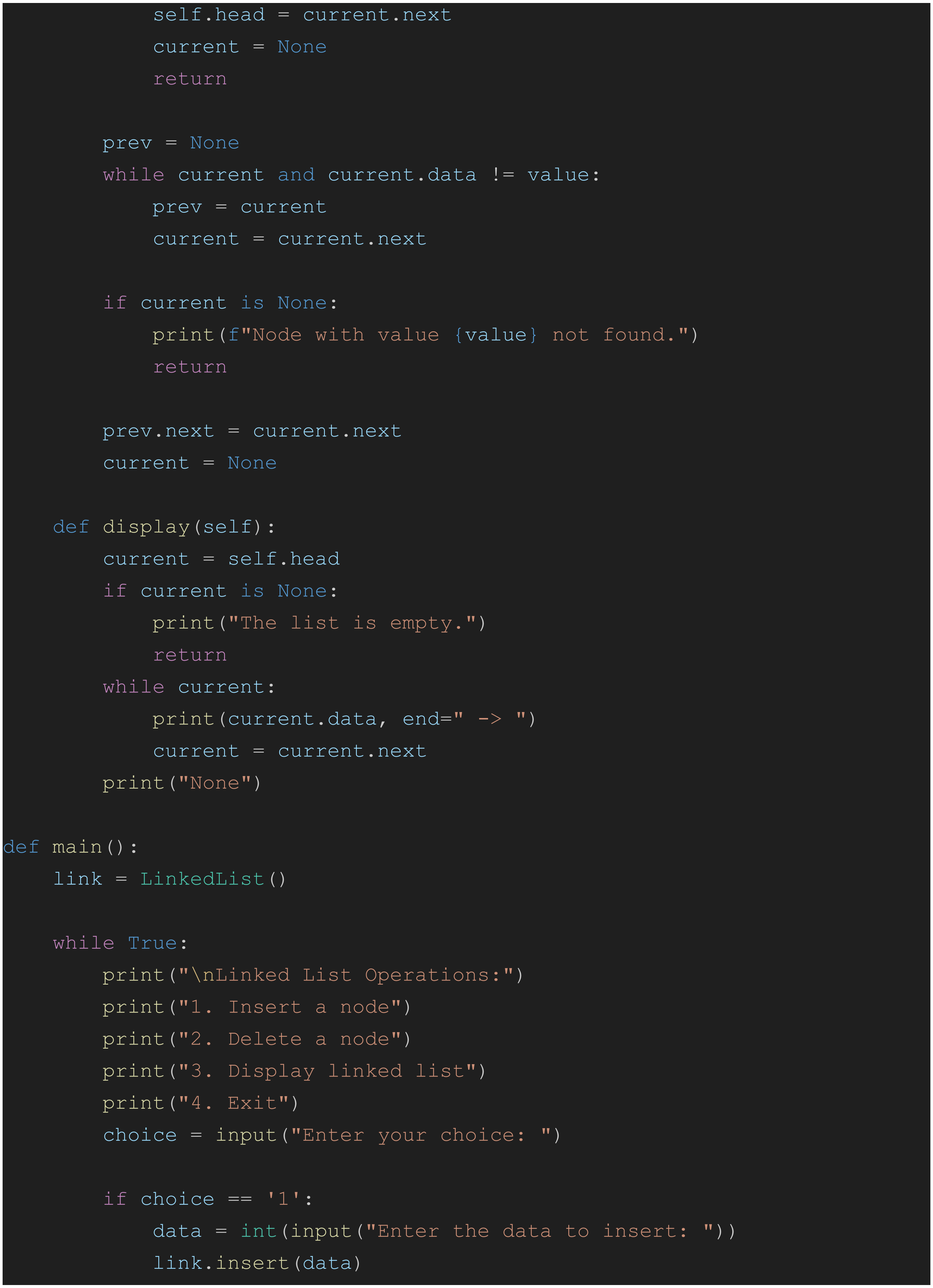
Surat-395007

Web Programming and Python (AI104)

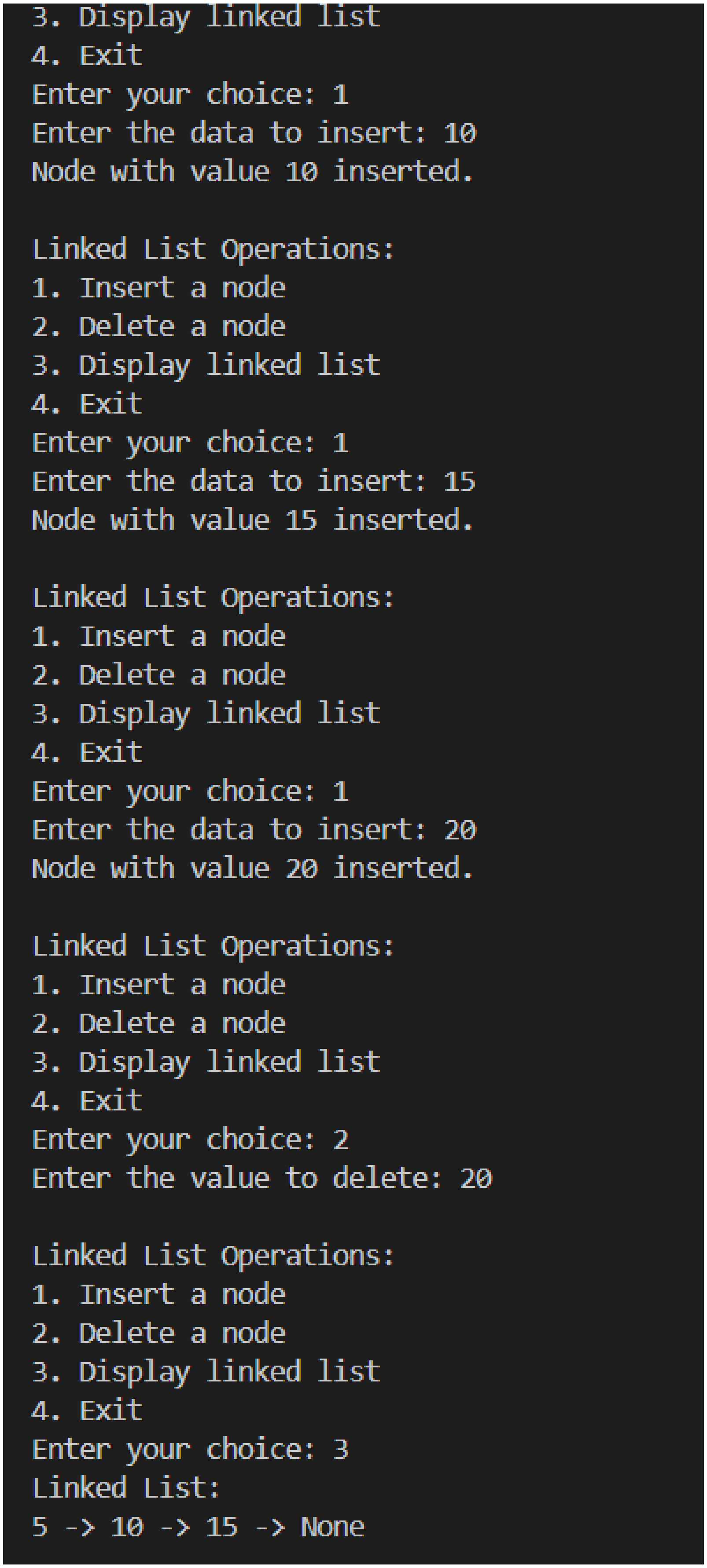
Assignment – 7 and 8
Object-Oriented Programming
ROLL NO: I24AI001

1. Write a Python program to create a class representing a linked list data structure. Include methods for displaying linked list data, inserting and deleting nodes.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        '''insert at end'''
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def delete(self, value):
        """Delete the first node"""
        current = self.head
        if current is None:
            print("The list is empty.")
            return

        if current.data == value:
```



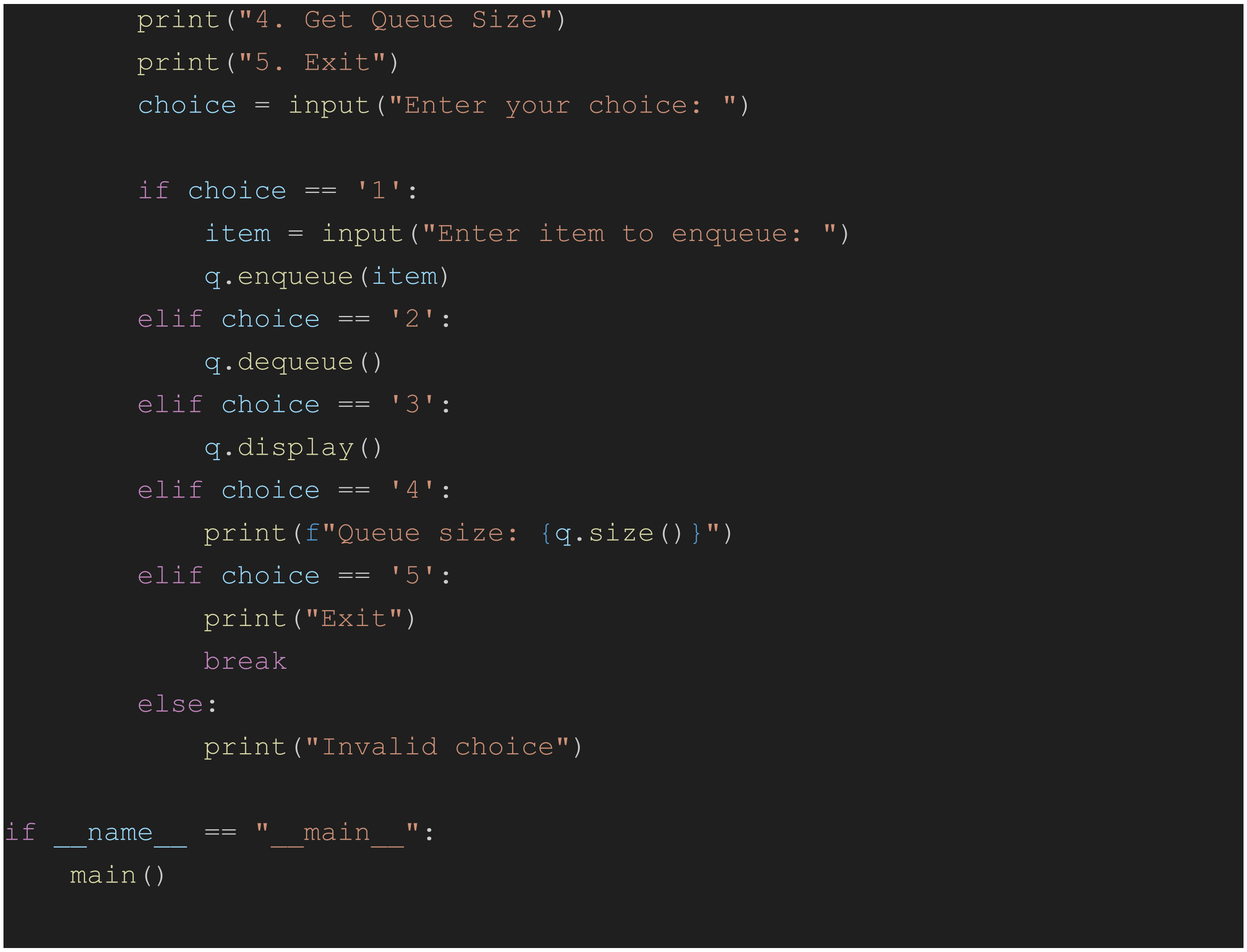
```
            self.head = current.next
            current = None
            return

        prev = None
        while current and current.data != value:
            prev = current
            current = current.next

        if current is None:
            print(f"Node with value {value} not found.")
            return

        prev.next = current.next
        current = None

    def display(self):
        current = self.head
        if current is None:
            print("The list is empty.")
            return
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

def main():
    link = LinkedList()

    while True:
        print("\nLinked List Operations:")
        print("1. Insert a node")
        print("2. Delete a node")
        print("3. Display linked list")
        print("4. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            data = int(input("Enter the data to insert: "))
            link.insert(data)
            print(f"Node with value {data} inserted.")
        elif choice == '2':
            value = int(input("Enter the value to delete: "))
            link.delete(value)
        elif choice == '3':
            print("Linked List:")
            link.display()
        elif choice == '4':
            print("Exit")
            break
        else:
            print("Invalid choice")

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



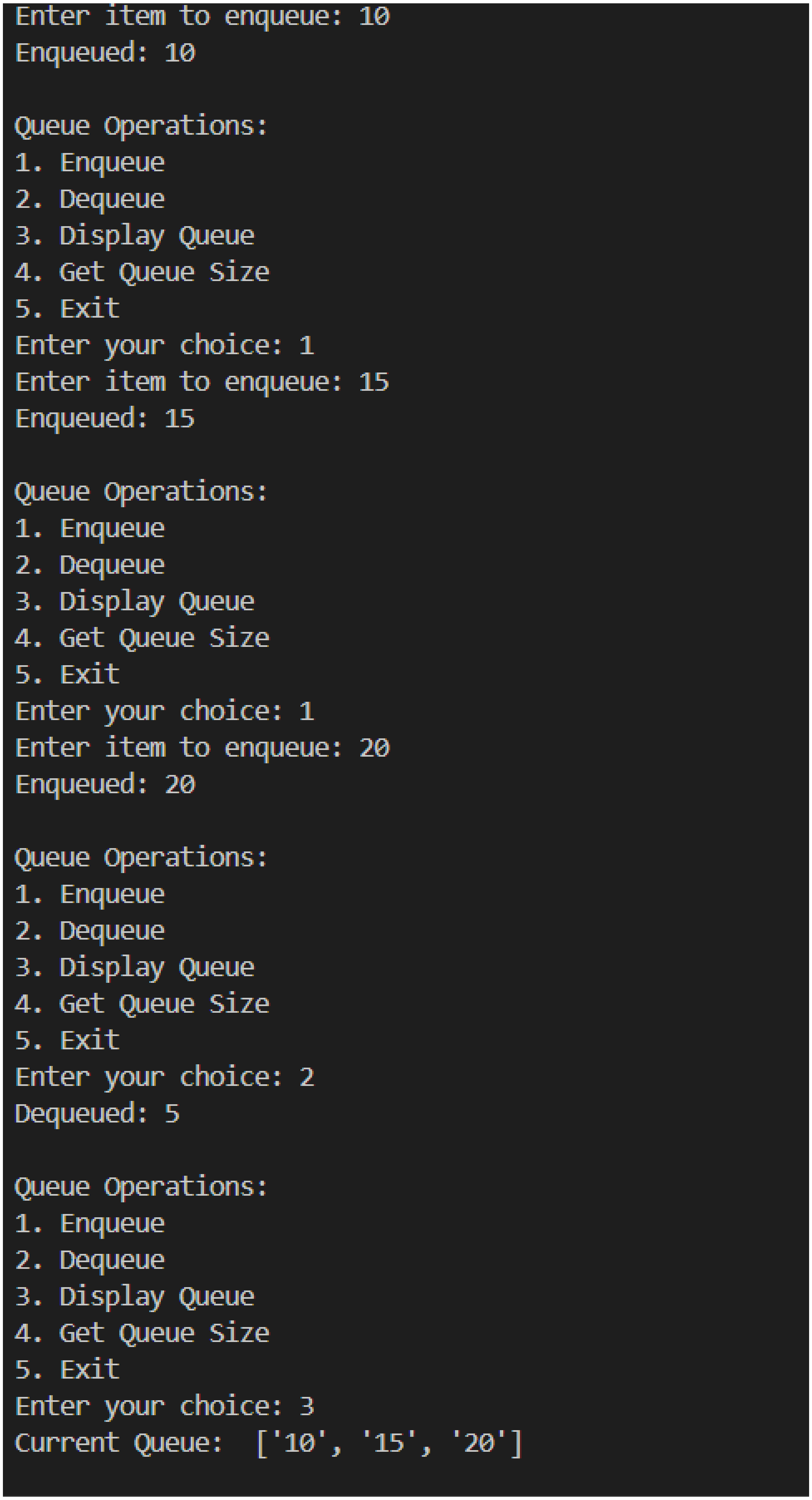
```
3. Display linked list
4. Exit
Enter your choice: 1
Enter the data to insert: 10
Node with value 10 inserted.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Display linked list
4. Exit
Enter your choice: 1
Enter the data to insert: 15
Node with value 15 inserted.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Display linked list
4. Exit
Enter your choice: 1
Enter the data to insert: 20
Node with value 20 inserted.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Display linked list
4. Exit
Enter your choice: 2
Enter the value to delete: 20

Linked List Operations:
1. Insert a node
2. Delete a node
3. Display linked list
4. Exit
Enter your choice: 3
Linked List:
5 -> 10 -> 15 -> None
```

2. Write a Python program to create a class representing a queue data structure. Include methods
for enqueueing and dequeuing elements.

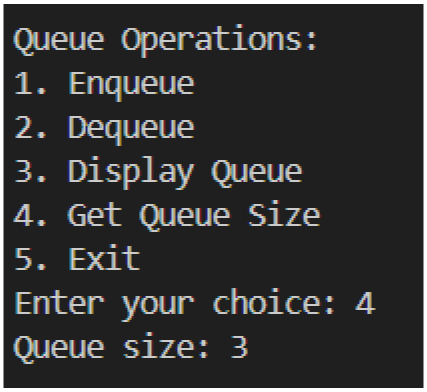
```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        """insert at end"""
        self.queue.append(item)
        print(f"Enqueued: {item}")

    def dequeue(self):
        """delete"""
        if len(self.queue) == 0:
            print("Queue is empty. Cannot dequeue.")
        else:
            removed_item = self.queue.pop(0)
            print(f"Dequeued: {removed_item}")
            return removed_item

    def display(self):
        """Display"""
        if len(self.queue) == 0:
            print("Queue is empty.")
        else:
            print("Current Queue: ", self.queue)

    def size(self):
        return len(self.queue)

def main():
    q = Queue()

    while True:
        print("\nQueue Operations:")
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. Display Queue")
```



```
        print("4. Get Queue Size")
        print("5. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            item = input("Enter item to enqueue: ")
            q.enqueue(item)
        elif choice == '2':
            q.dequeue()
        elif choice == '3':
            q.display()
        elif choice == '4':
            print(f"Queue size: {q.size()}")
        elif choice == '5':
            print("Exit")
            break
        else:
            print("Invalid choice")

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

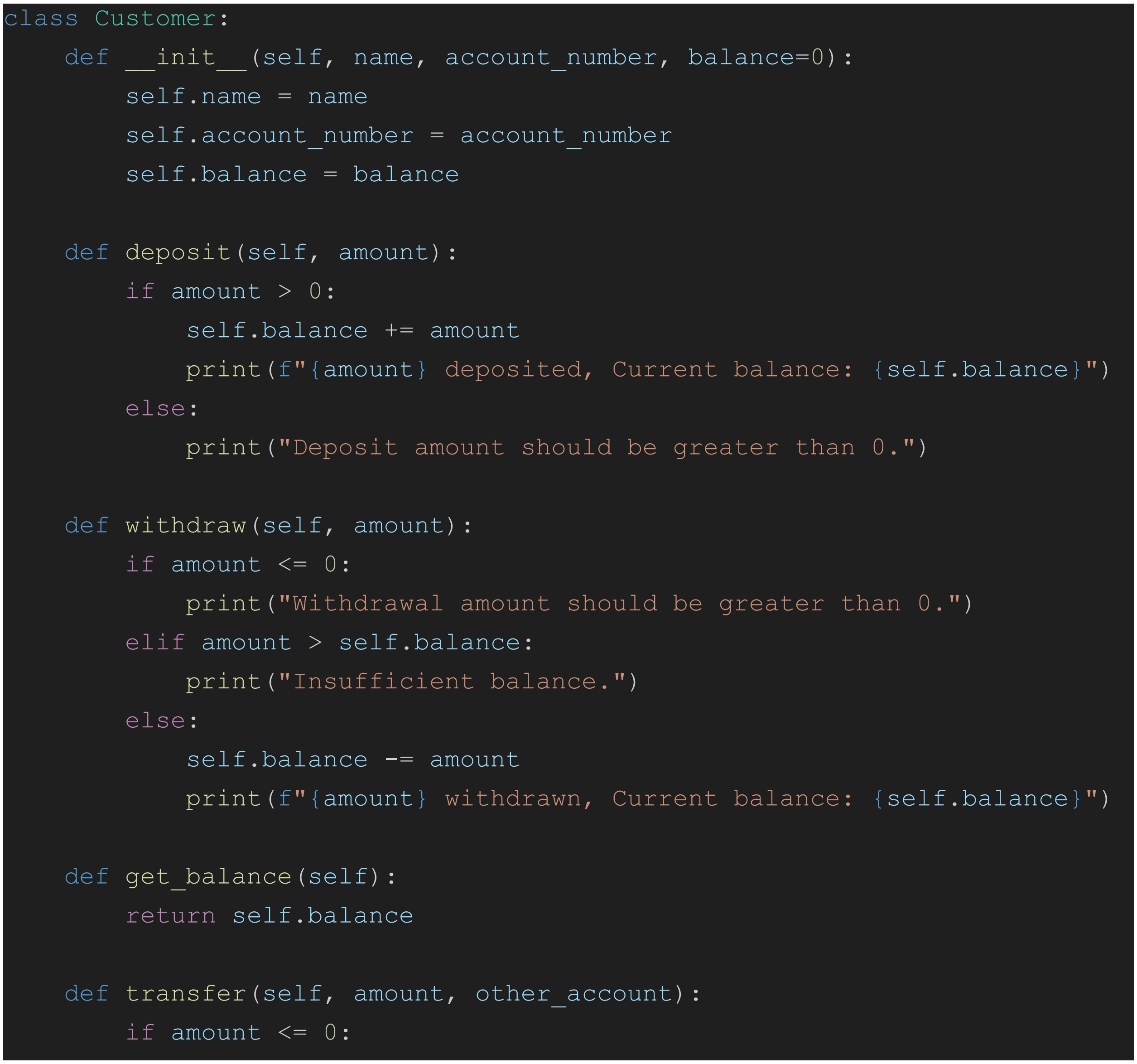


```
Enter item to enqueue: 10
Enqueued: 10

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Get Queue Size
5. Exit
Enter your choice: 1
Enter item to enqueue: 15
Enqueued: 15

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Get Queue Size
5. Exit
Enter your choice: 1
Enter item to enqueue: 20
Enqueued: 20

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Get Queue Size
5. Exit
Enter your choice: 2
Dequeued: 5

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Get Queue Size
5. Exit
Enter your choice: 3
Current Queue:  ['10', '15', '20']
```

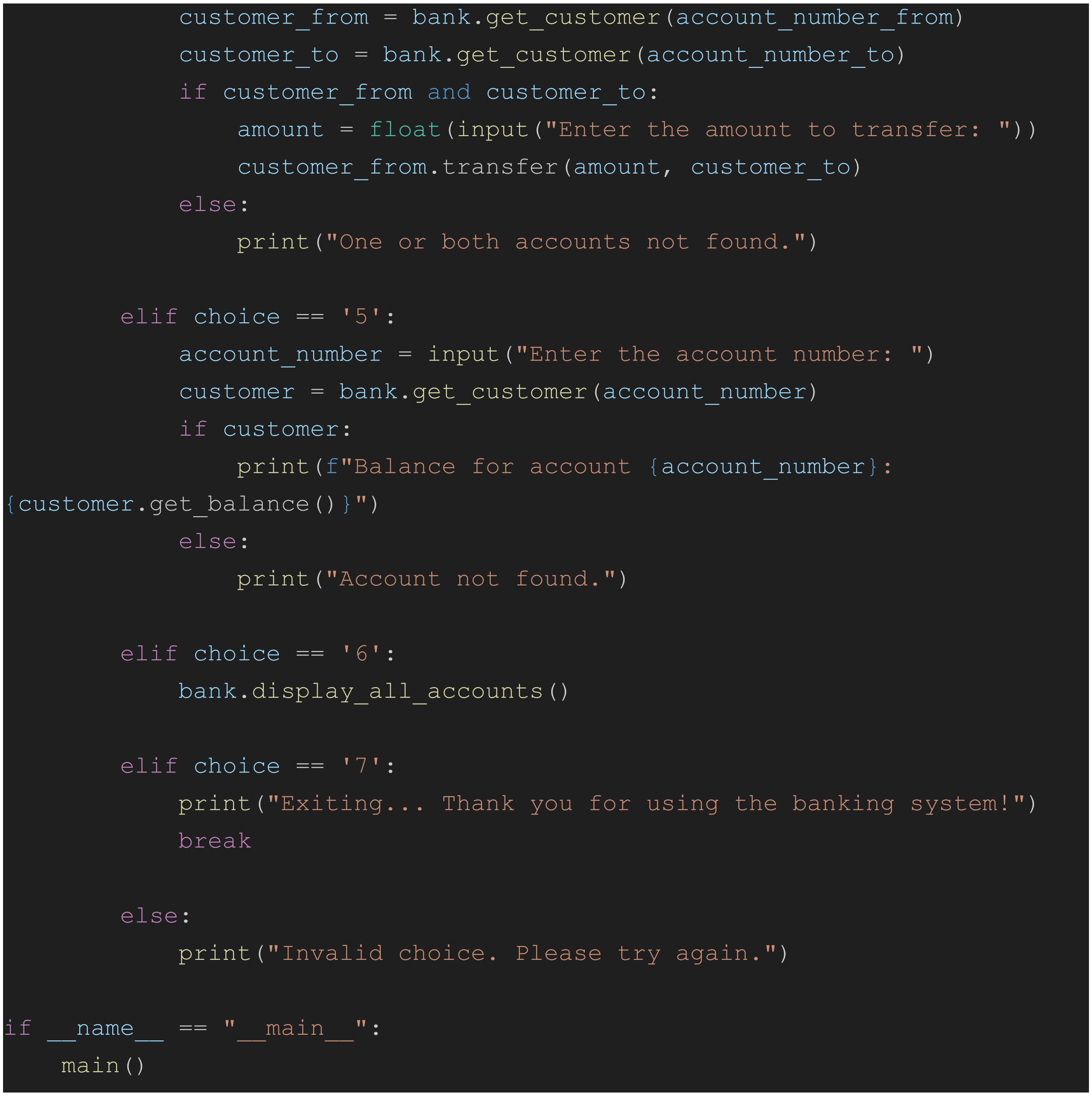


```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Get Queue Size
5. Exit
Enter your choice: 4
Queue size: 3
```

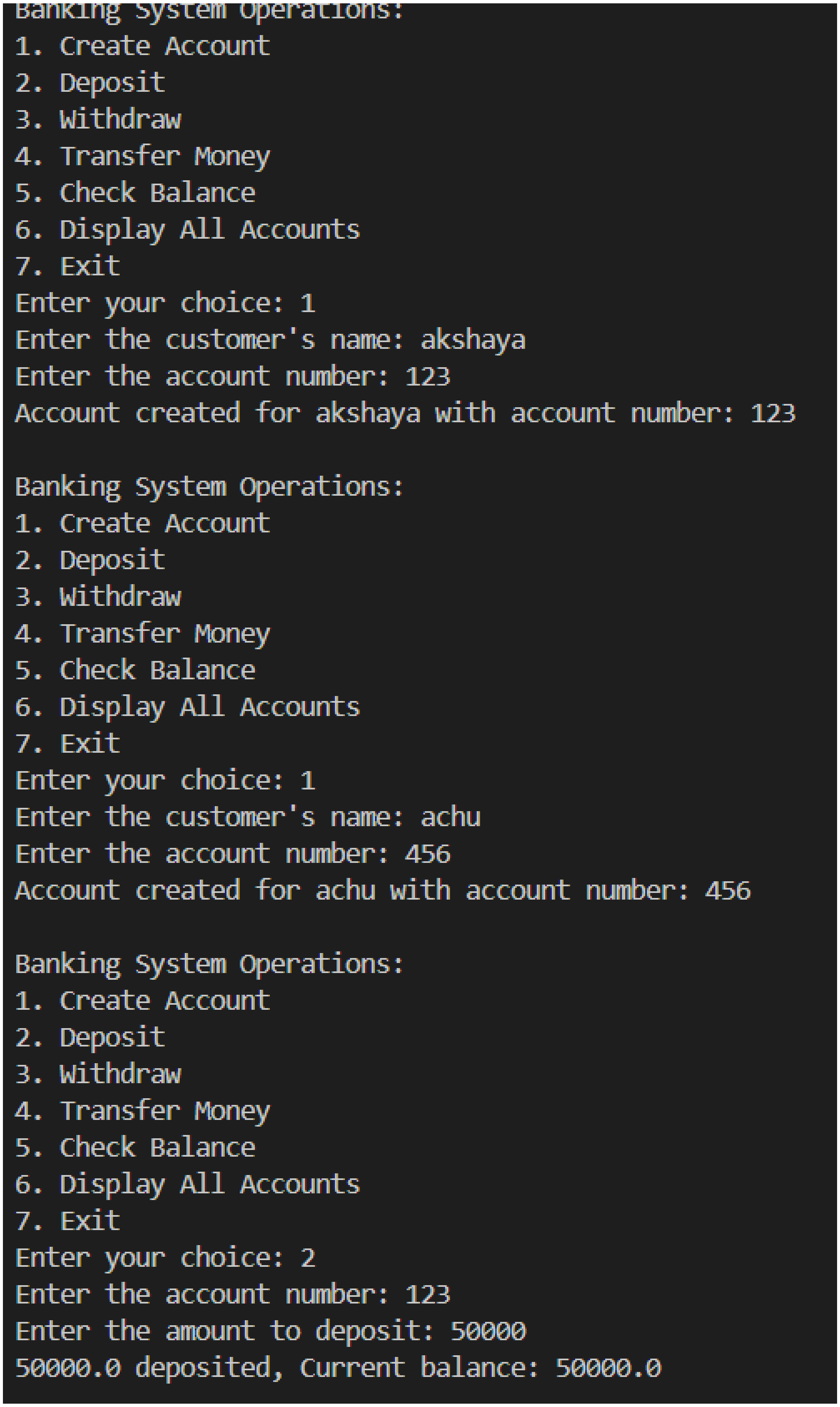
3. Write a Python program to create a class representing a bank. Include methods for managing customer accounts and transactions.



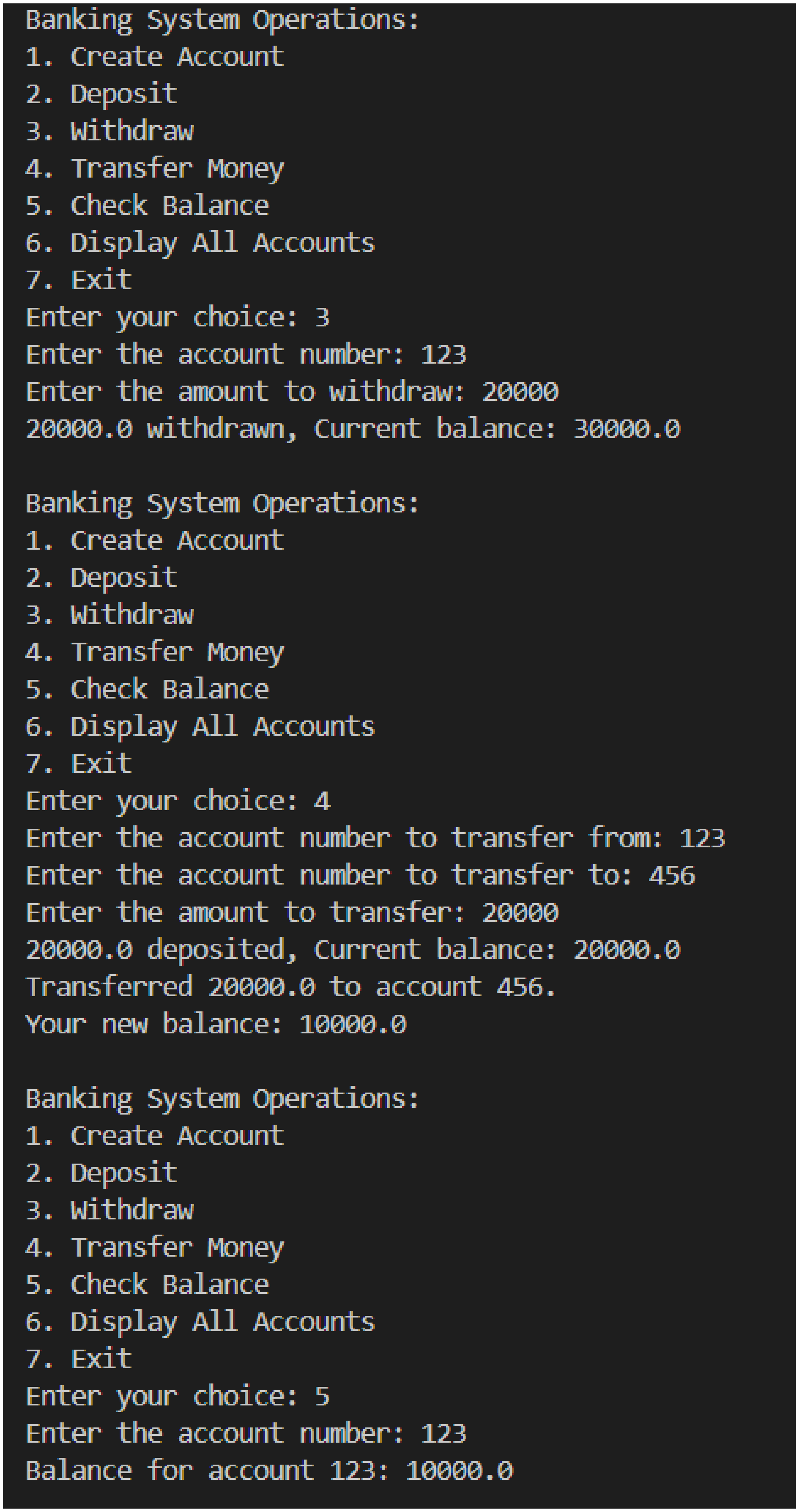
```python
class Customer:
    def __init__(self, name, account_number, balance=0):
        self.name = name
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"{amount} deposited, Current balance: {self.balance}")
        else:
            print("Deposit amount should be greater than 0.")

    def withdraw(self, amount):
        if amount <= 0:
            print("Withdrawal amount should be greater than 0.")
        elif amount > self.balance:
            print("Insufficient balance.")
        else:
            self.balance -= amount
            print(f"{amount} withdrawn, Current balance: {self.balance}")

    def get_balance(self):
        return self.balance

    def transfer(self, amount, other_account):
        if amount <= 0:
            print("Transfer amount should be greater than 0.")
        elif amount > self.balance:
            print("Insufficient balance to transfer.")
        else:
            self.balance -= amount
            other_account.deposit(amount)
            print(f"Transferred {amount} to account
{other_account.account_number}.")
            print(f"Your new balance: {self.balance}")


class Bank:
    def __init__(self):
        self.customers = {}

    def create_account(self, name, account_number):

        if account_number in self.customers:
            print("Account with this account number already exists.")
        else:
            self.customers[account_number] = Customer(name,
account_number)
            print(f"Account created for {name} with account number:
{account_number}")

    def get_customer(self, account_number):

        return self.customers.get(account_number, None)

    def display_all_accounts(self):

        if not self.customers:
            print("No accounts found.")
        else:
            for account_number, customer in self.customers.items():
                print(f"Account Number: {account_number}, Name:
{customer.name}, Balance: {customer.get_balance()}")

def main():
    bank = Bank()
    while True:
        print("\nBanking System Operations:")
        print("1. Create Account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Transfer Money")
        print("5. Check Balance")
        print("6. Display All Accounts")
        print("7. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            name = input("Enter the customer's name: ")
            account_number = input("Enter the account number: ")
            bank.create_account(name, account_number)

        elif choice == '2':
            account_number = input("Enter the account number: ")
            customer = bank.get_customer(account_number)
            if customer:
                amount = float(input("Enter the amount to deposit: "))
                customer.deposit(amount)
            else:
                print("Account not found.")

        elif choice == '3':
            account_number = input("Enter the account number: ")
            customer = bank.get_customer(account_number)
            if customer:
                amount = float(input("Enter the amount to withdraw: "))
                customer.withdraw(amount)
            else:
                print("Account not found.")

        elif choice == '4':
            account_number_from = input("Enter the account number to
transfer from: ")
            account_number_to = input("Enter the account number to
transfer to: ")
```



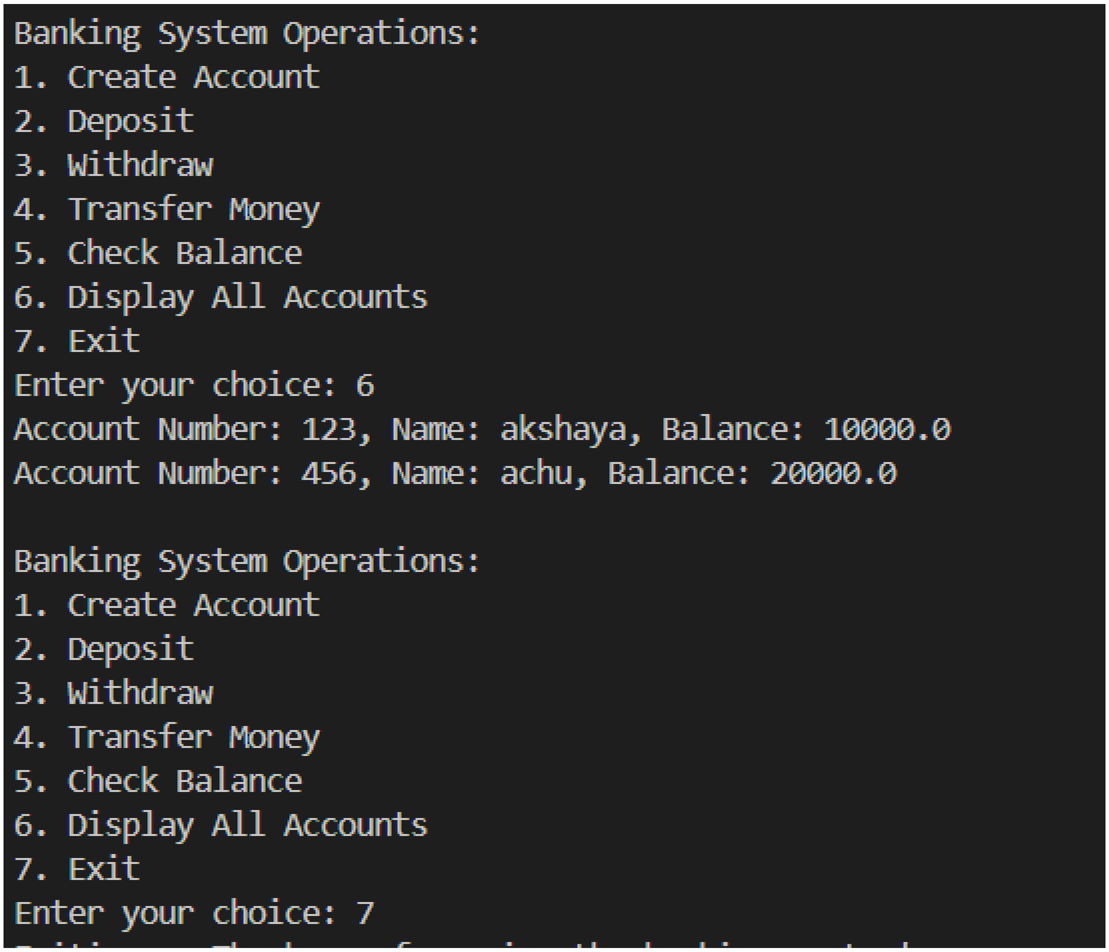
```
            customer_from = bank.get_customer(account_number_from)
            customer_to = bank.get_customer(account_number_to)
            if customer_from and customer_to:
                amount = float(input("Enter the amount to transfer: "))
                customer_from.transfer(amount, customer_to)
            else:
                print("One or both accounts not found.")

        elif choice == '5':
            account_number = input("Enter the account number: ")
            customer = bank.get_customer(account_number)
            if customer:
                print(f"Balance for account {account_number}:
{customer.get_balance()}")
            else:
                print("Account not found.")

        elif choice == '6':
            bank.display_all_accounts()

        elif choice == '7':
            print("Exiting... Thank you for using the banking system!")
            break

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

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



```
Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 1
Enter the customer's name: akshaya
Enter the account number: 123
Account created for akshaya with account number: 123

Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 1
Enter the customer's name: achu
Enter the account number: 456
Account created for achu with account number: 456

Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 2
Enter the account number: 123
Enter the amount to deposit: 50000
50000.0 deposited, Current balance: 50000.0
```



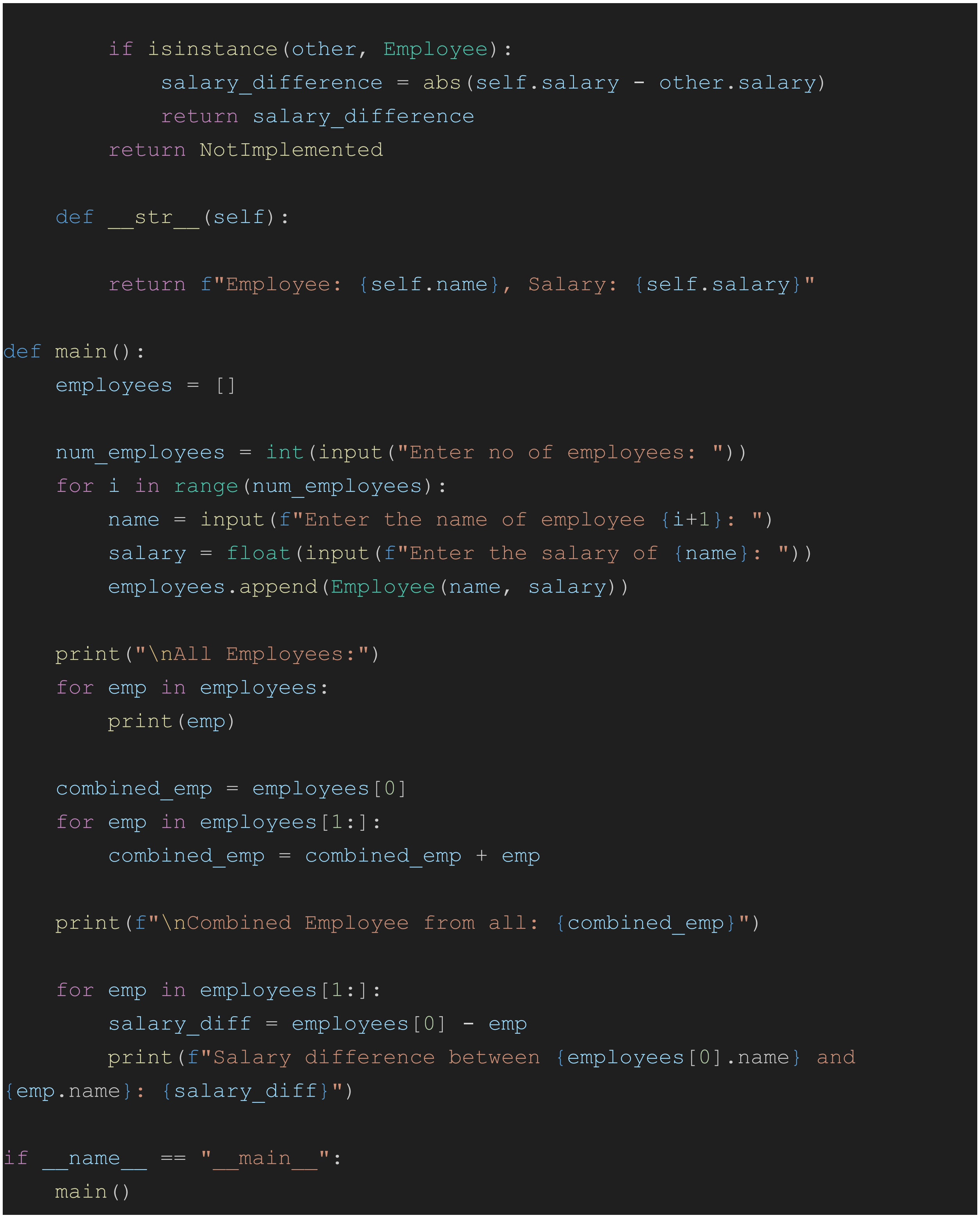
```
Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 3
Enter the account number: 123
Enter the amount to withdraw: 20000
20000.0 withdrawn, Current balance: 30000.0

Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 4
Enter the account number to transfer from: 123
Enter the account number to transfer to: 456
Enter the amount to transfer: 20000
20000.0 deposited, Current balance: 20000.0
Transferred 20000.0 to account 456.
Your new balance: 10000.0

Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 5
Enter the account number: 123
Balance for account 123: 10000.0
```



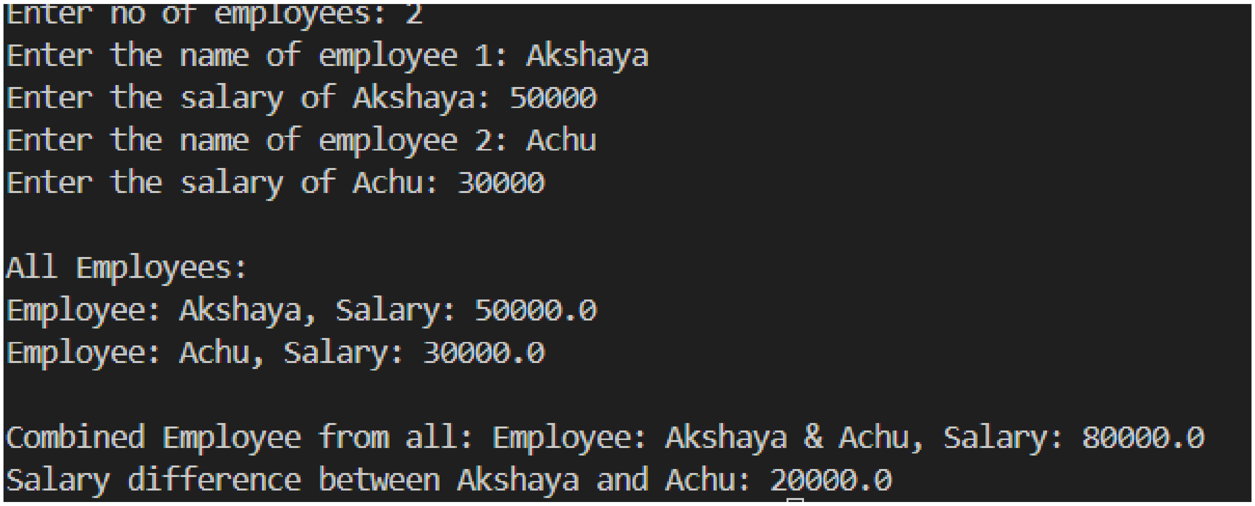
```
Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 6
Account Number: 123, Name: akshaya, Balance: 10000.0
Account Number: 456, Name: achu, Balance: 20000.0

Banking System Operations:
1. Create Account
2. Deposit
3. Withdraw
4. Transfer Money
5. Check Balance
6. Display All Accounts
7. Exit
Enter your choice: 7
```

4. Create a class "Employee" with attributes name and salary. Implement overloaded operators +
and - to combine and compare employees based on their salaries.

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def __add__(self, other):

        if isinstance(other, Employee):
            combined_salary = self.salary + other.salary
            return Employee(f"{self.name} & {other.name}",
combined_salary)
        return NotImplemented

    def __sub__(self, other):
```



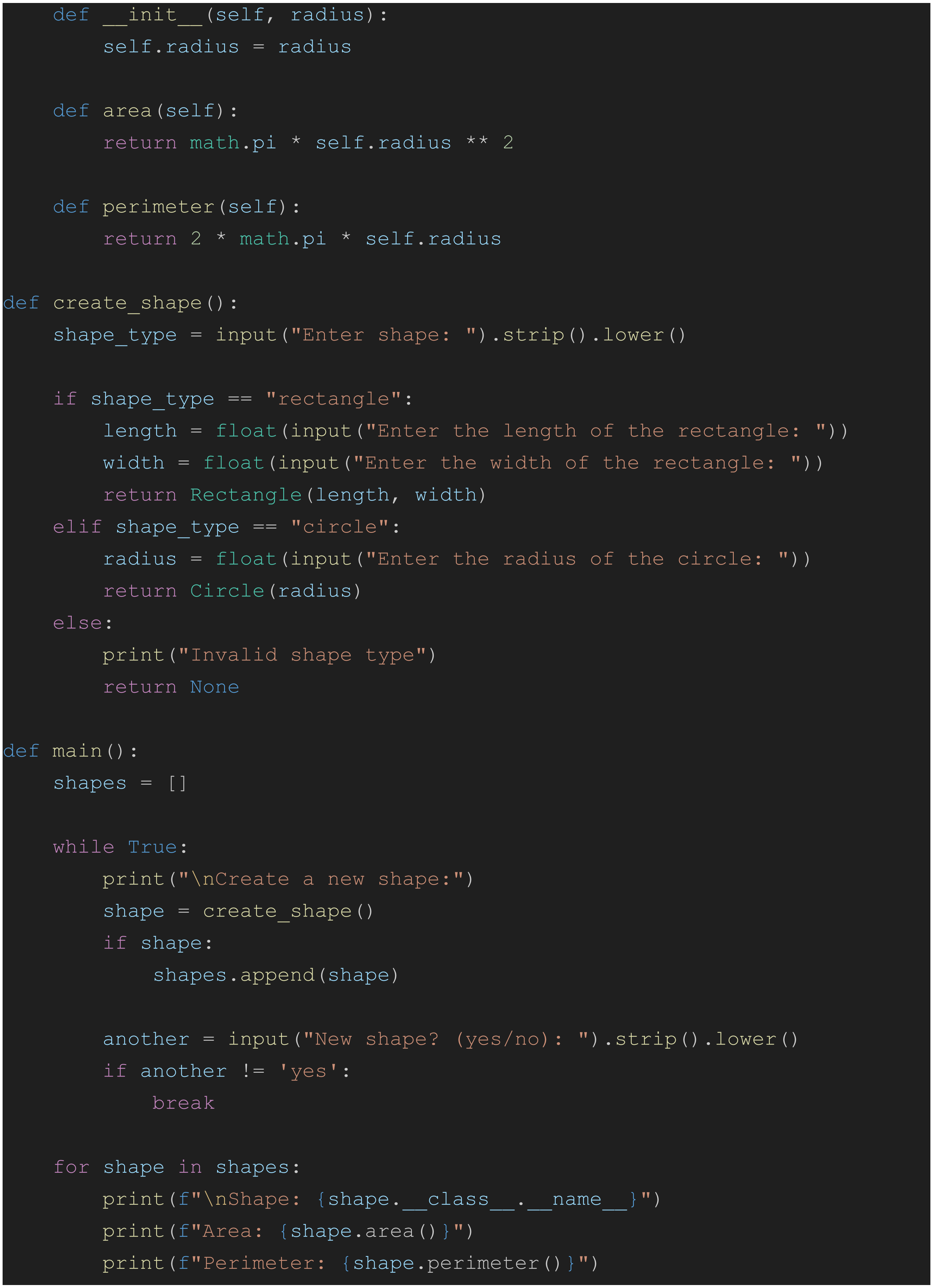
```
        if isinstance(other, Employee):
            salary_difference = abs(self.salary - other.salary)
            return salary_difference
        return NotImplemented

    def __str__(self):

        return f"Employee: {self.name}, Salary: {self.salary}"

def main():
    employees = []

    num_employees = int(input("Enter no of employees: "))
    for i in range(num_employees):
        name = input(f"Enter the name of employee {i+1}: ")
        salary = float(input(f"Enter the salary of {name}: "))
        employees.append(Employee(name, salary))

    print("\nAll Employees:")
    for emp in employees:
        print(emp)

    combined_emp = employees[0]
    for emp in employees[1:]:
        combined_emp = combined_emp + emp

    print(f"\nCombined Employee from all: {combined_emp}")

    for emp in employees[1:]:
        salary_diff = employees[0] - emp
        print(f"Salary difference between {employees[0].name} and
{emp.name}: {salary_diff}")

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



```
Enter no of employees: 2
Enter the name of employee 1: Akshaya
Enter the salary of Akshaya: 50000
Enter the name of employee 2: Achu
Enter the salary of Achu: 30000

All Employees:
Employee: Akshaya, Salary: 50000.0
Employee: Achu, Salary: 30000.0

Combined Employee from all: Employee: Akshaya & Achu, Salary: 80000.0
Salary difference between Akshaya and Achu: 20000.0
```

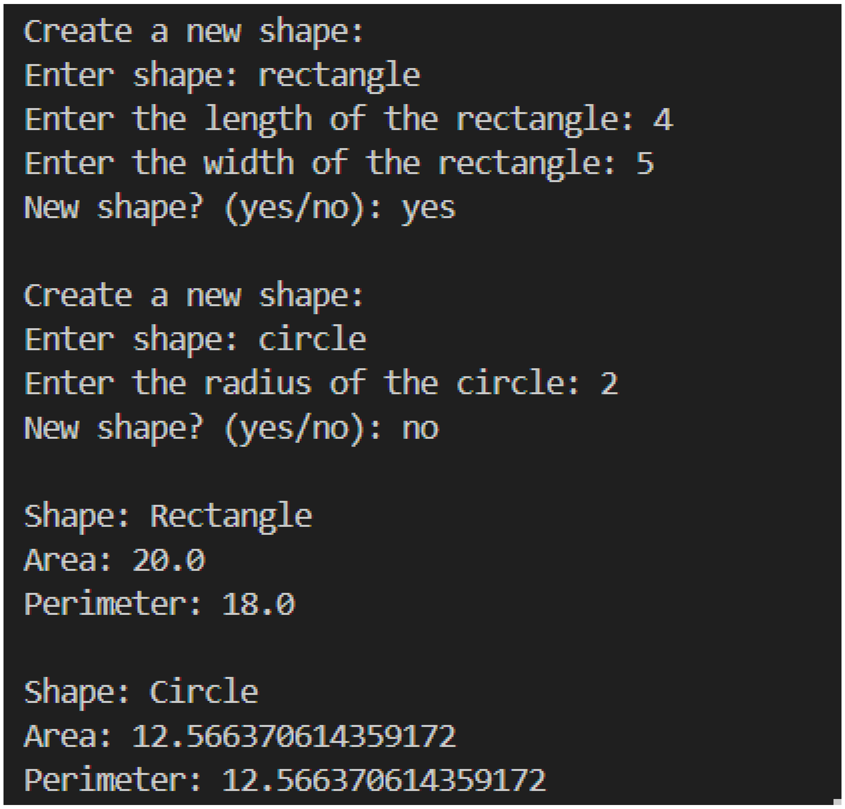
5. Create a base class "Shape" with methods to calculate the area and perimeter. Implement derived classes "Rectangle" and "Circle" that inherit from "Shape" and provide their own area and perimeter calculations.

```
import math

class Shape:
    def area(self):
        raise NotImplementedError("Subclasses should implement this
method")

    def perimeter(self):
        raise NotImplementedError("Subclasses should implement this
method.")

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

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

class Circle(Shape):
```



```python
    def __init__(self, radius):
        self.radius = radius

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

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

def create_shape():
    shape_type = input("Enter shape: ").strip().lower()

    if shape_type == "rectangle":
        length = float(input("Enter the length of the rectangle: "))
        width = float(input("Enter the width of the rectangle: "))
        return Rectangle(length, width)
    elif shape_type == "circle":
        radius = float(input("Enter the radius of the circle: "))
        return Circle(radius)
    else:
        print("Invalid shape type")
        return None

def main():
    shapes = []

    while True:
        print("\nCreate a new shape:")
        shape = create_shape()
        if shape:
            shapes.append(shape)

        another = input("New shape? (yes/no): ").strip().lower()
        if another != 'yes':
            break

    for shape in shapes:
        print(f"\nShape: {shape.__class__.__name__}")
        print(f"Area: {shape.area()}")
        print(f"Perimeter: {shape.perimeter()}")
```

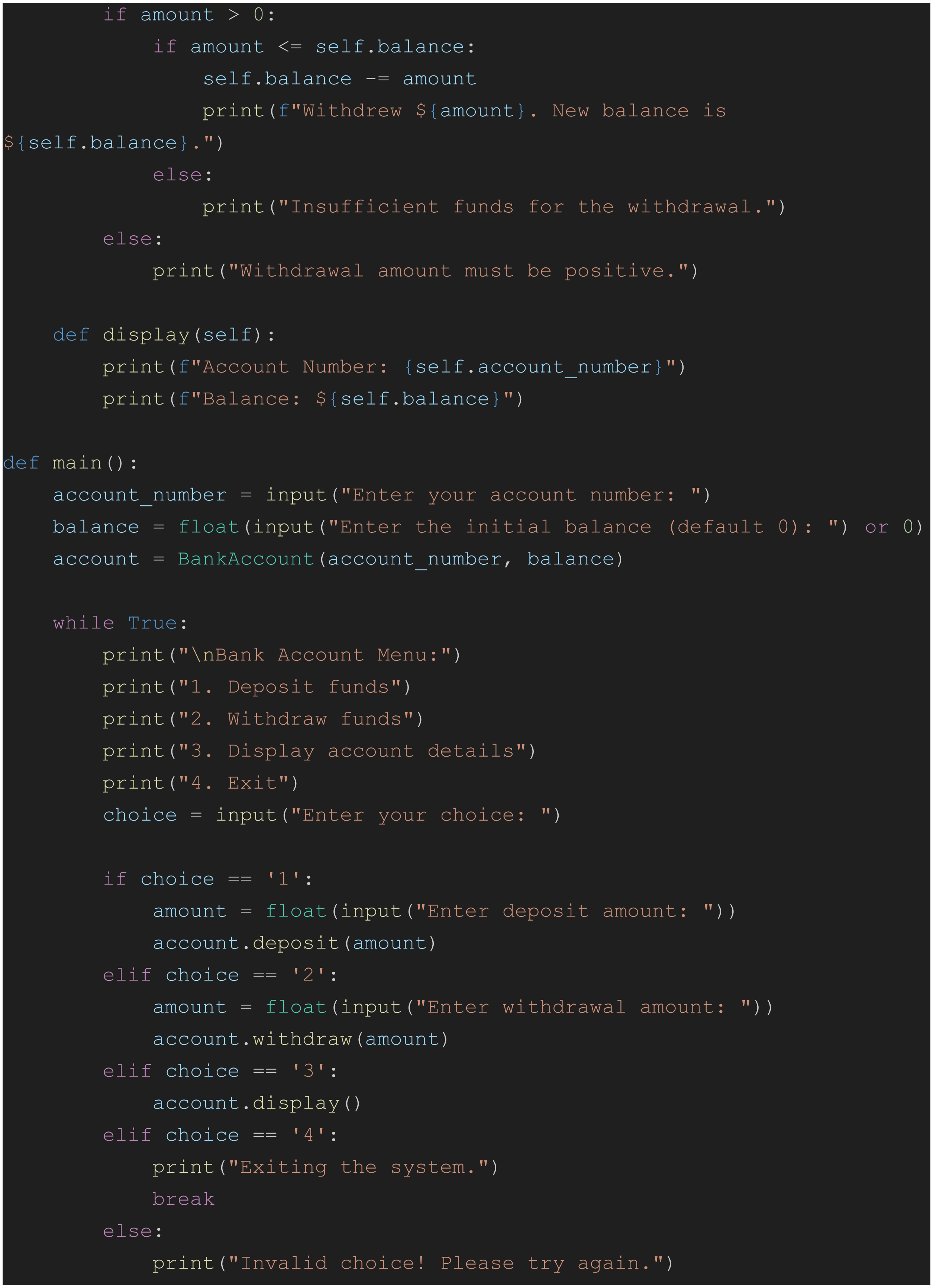
```
if __name__ == "__main__":
    main()
```



```
Create a new shape:
Enter shape: rectangle
Enter the length of the rectangle: 4
Enter the width of the rectangle: 5
New shape? (yes/no): yes

Create a new shape:
Enter shape: circle
Enter the radius of the circle: 2
New shape? (yes/no): no

Shape: Rectangle
Area: 20.0
Perimeter: 18.0

Shape: Circle
Area: 12.566370614359172
Perimeter: 12.566370614359172
```

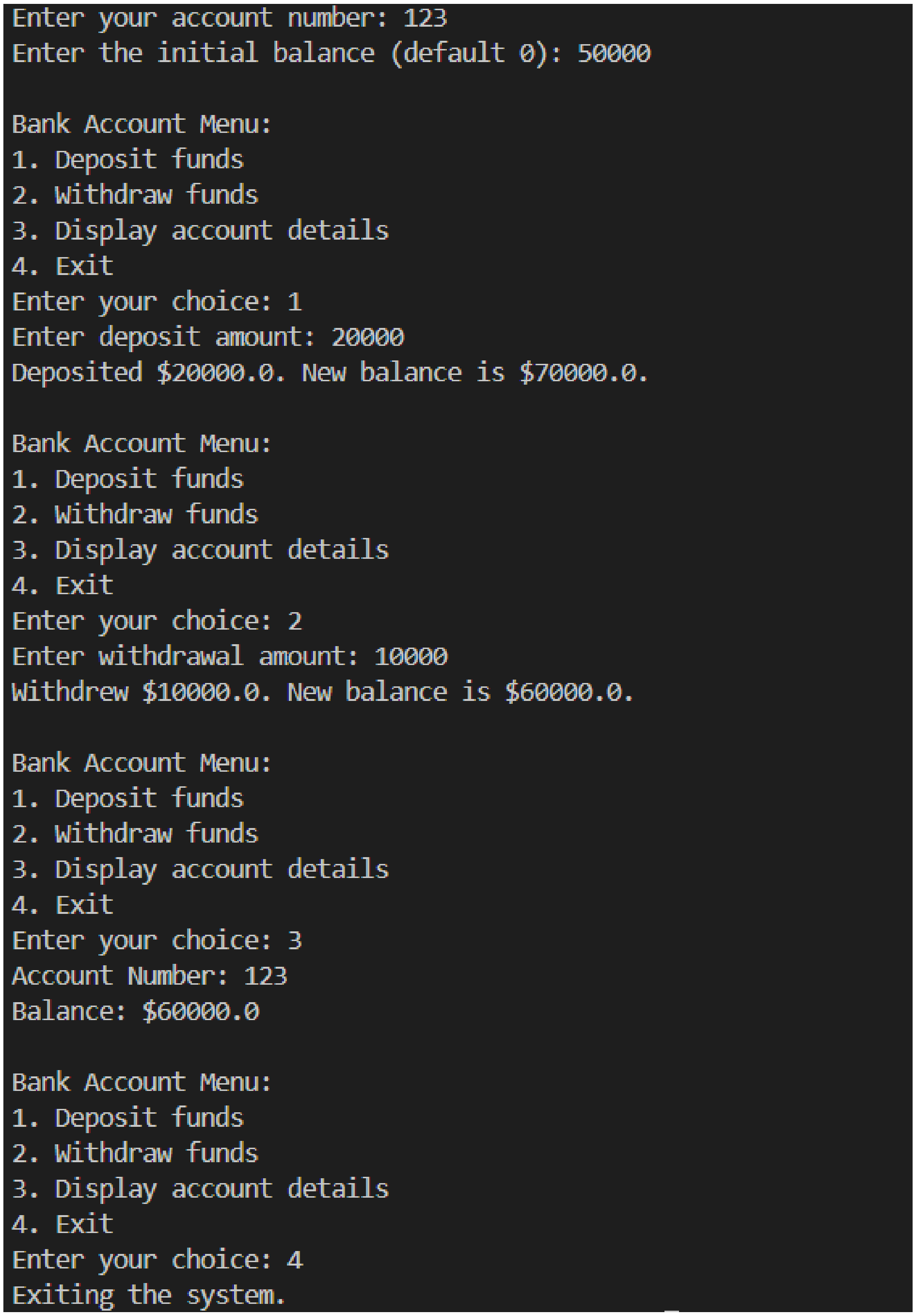
6. Create a class "BankAccount" with attributes account number and balance. Implement methods to deposit and withdraw funds, and a display method to show the account details.

```
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount}. New balance is ${self.balance}.")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
```



```
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew ${amount}. New balance is
${self.balance}.")
            else:
                print("Insufficient funds for the withdrawal.")
        else:
            print("Withdrawal amount must be positive.")

    def display(self):
        print(f"Account Number: {self.account_number}")
        print(f"Balance: ${self.balance}")

def main():
    account_number = input("Enter your account number: ")
    balance = float(input("Enter the initial balance (default 0): ") or 0)
    account = BankAccount(account_number, balance)

    while True:
        print("\nBank Account Menu:")
        print("1. Deposit funds")
        print("2. Withdraw funds")
        print("3. Display account details")
        print("4. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            amount = float(input("Enter deposit amount: "))
            account.deposit(amount)
        elif choice == '2':
            amount = float(input("Enter withdrawal amount: "))
            account.withdraw(amount)
        elif choice == '3':
            account.display()
        elif choice == '4':
            print("Exiting the system.")
            break
        else:
            print("Invalid choice! Please try again.")
```

```python
if __name__ == "__main__":
    main()
```

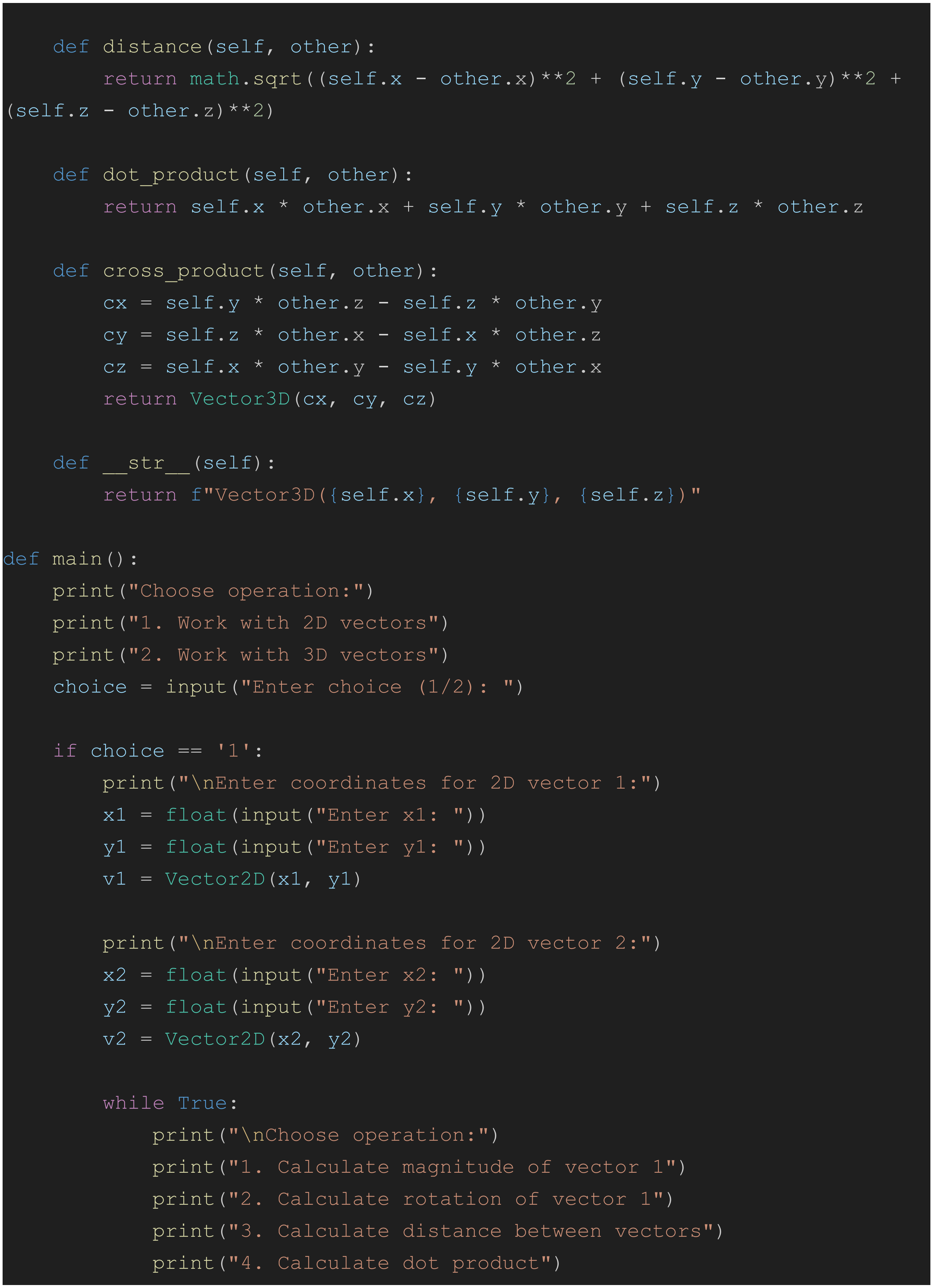


```
Enter your account number: 123
Enter the initial balance (default 0): 50000

Bank Account Menu:
1. Deposit funds
2. Withdraw funds
3. Display account details
4. Exit
Enter your choice: 1
Enter deposit amount: 20000
Deposited $20000.0. New balance is $70000.0.

Bank Account Menu:
1. Deposit funds
2. Withdraw funds
3. Display account details
4. Exit
Enter your choice: 2
Enter withdrawal amount: 10000
Withdrew $10000.0. New balance is $60000.0.

Bank Account Menu:
1. Deposit funds
2. Withdraw funds
3. Display account details
4. Exit
Enter your choice: 3
Account Number: 123
Balance: $60000.0

Bank Account Menu:
1. Deposit funds
2. Withdraw funds
3. Display account details
4. Exit
Enter your choice: 4
Exiting the system.
```

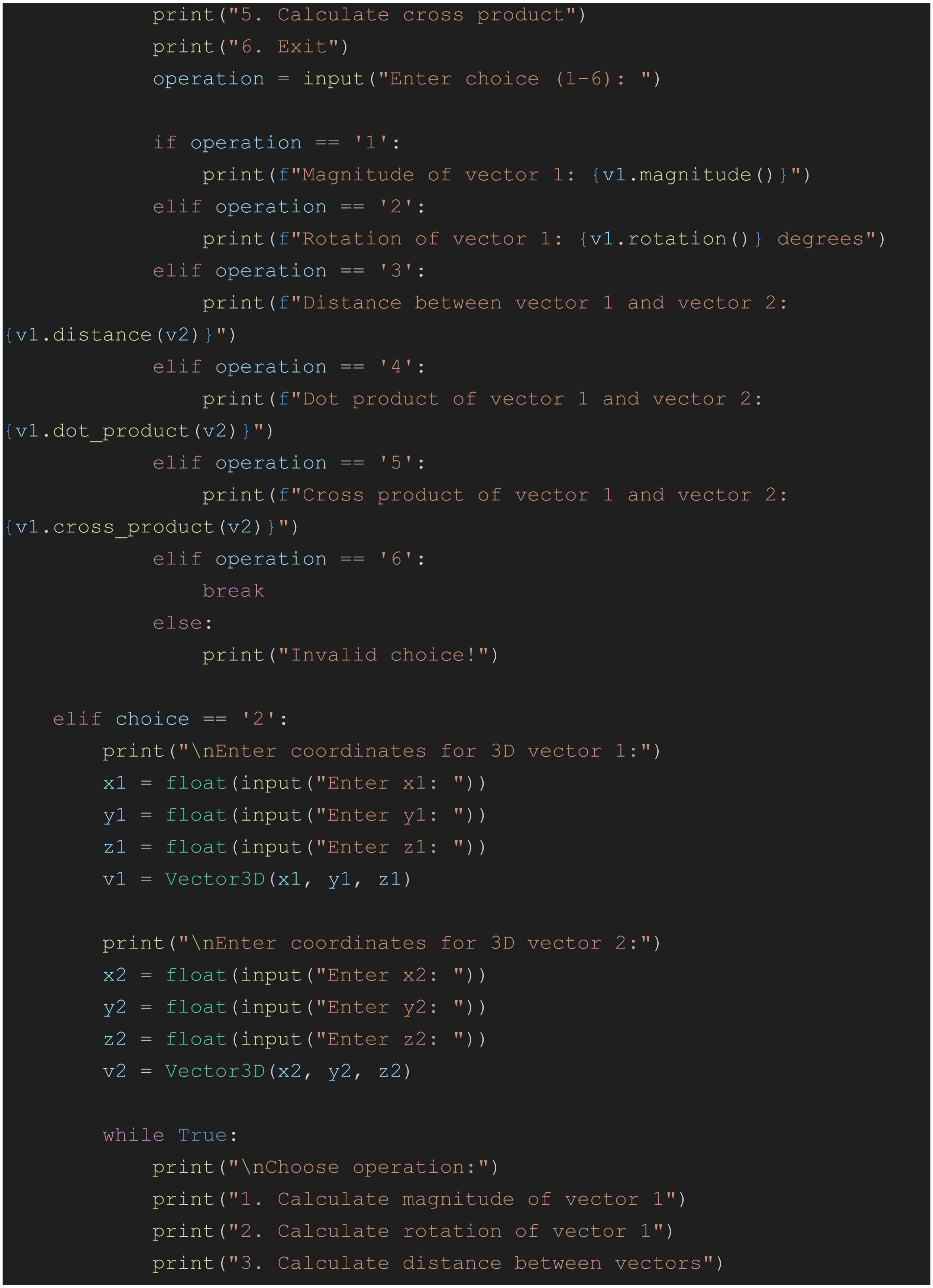
7. Create a class for representing any 2-D point or vector. The methods inside this class include its magnitude and its rotation with respect to the X-axis. Using the objects define functions for calculating the distance between two vectors, dot product, cross product of two vectors. Extend the 2-D vectors into 3-D using the concept of inheritance. Update the methods according to 3-

D.

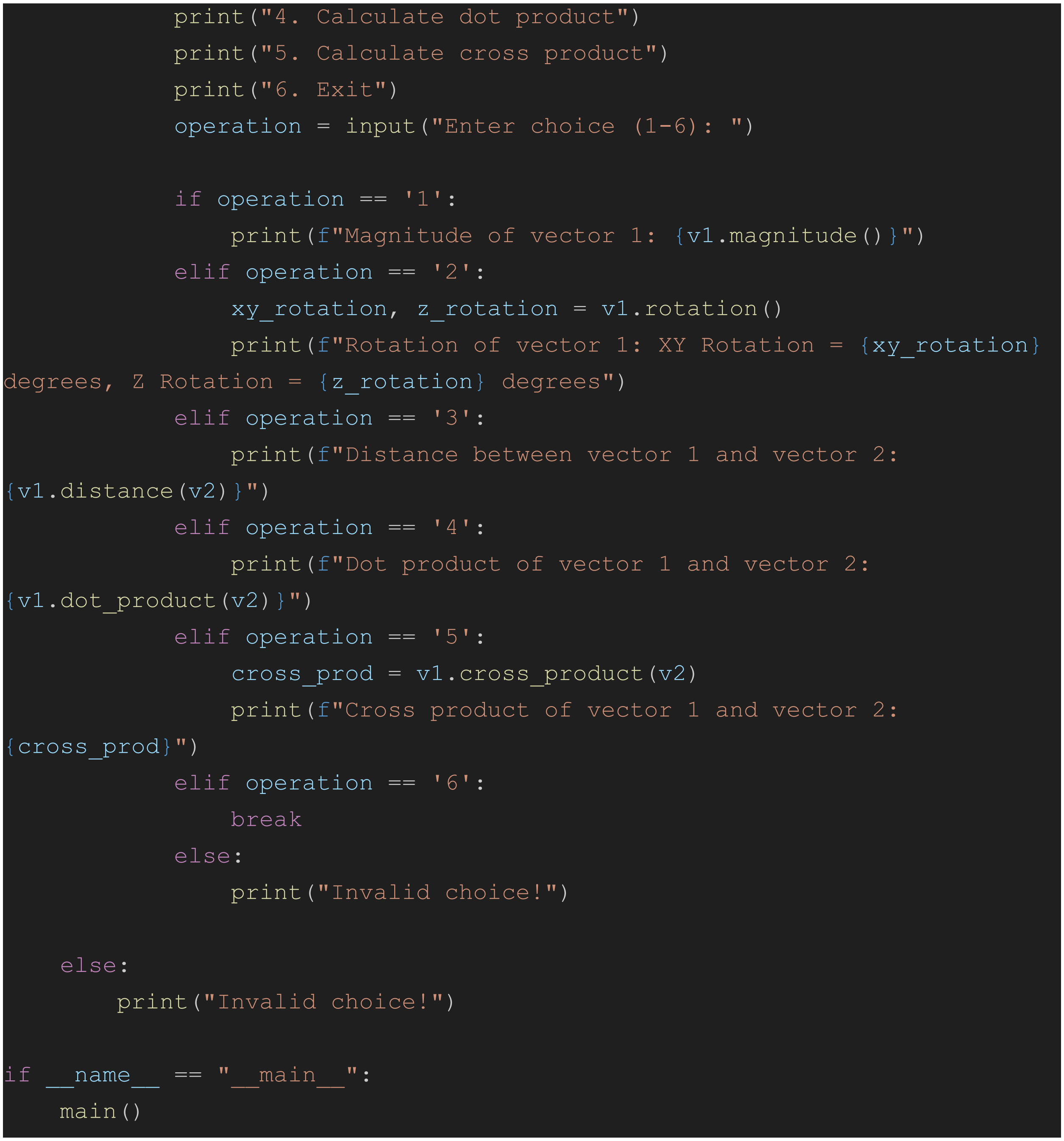
```
import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)

    def rotation(self):
        return math.degrees(math.atan2(self.y, self.x))

    def distance(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

    def dot_product(self, other):
        return self.x * other.x + self.y * other.y

    def cross_product(self, other):
        return self.x * other.y - self.y * other.x

    def __str__(self):
        return f"Vector2D({self.x}, {self.y})"

class Vector3D(Vector2D):
    def __init__(self, x, y, z):
        super().__init__(x, y)  # Initialize the 2D part (x, y)
        self.z = z

    def magnitude(self):
        return math.sqrt(self.x**2 + self.y**2 + self.z**2)

    def rotation(self):
        xy_rotation = math.degrees(math.atan2(self.y, self.x))
        z_rotation = math.degrees(math.atan2(self.z, math.sqrt(self.x**2 +
self.y**2)))
        return xy_rotation, z_rotation
```



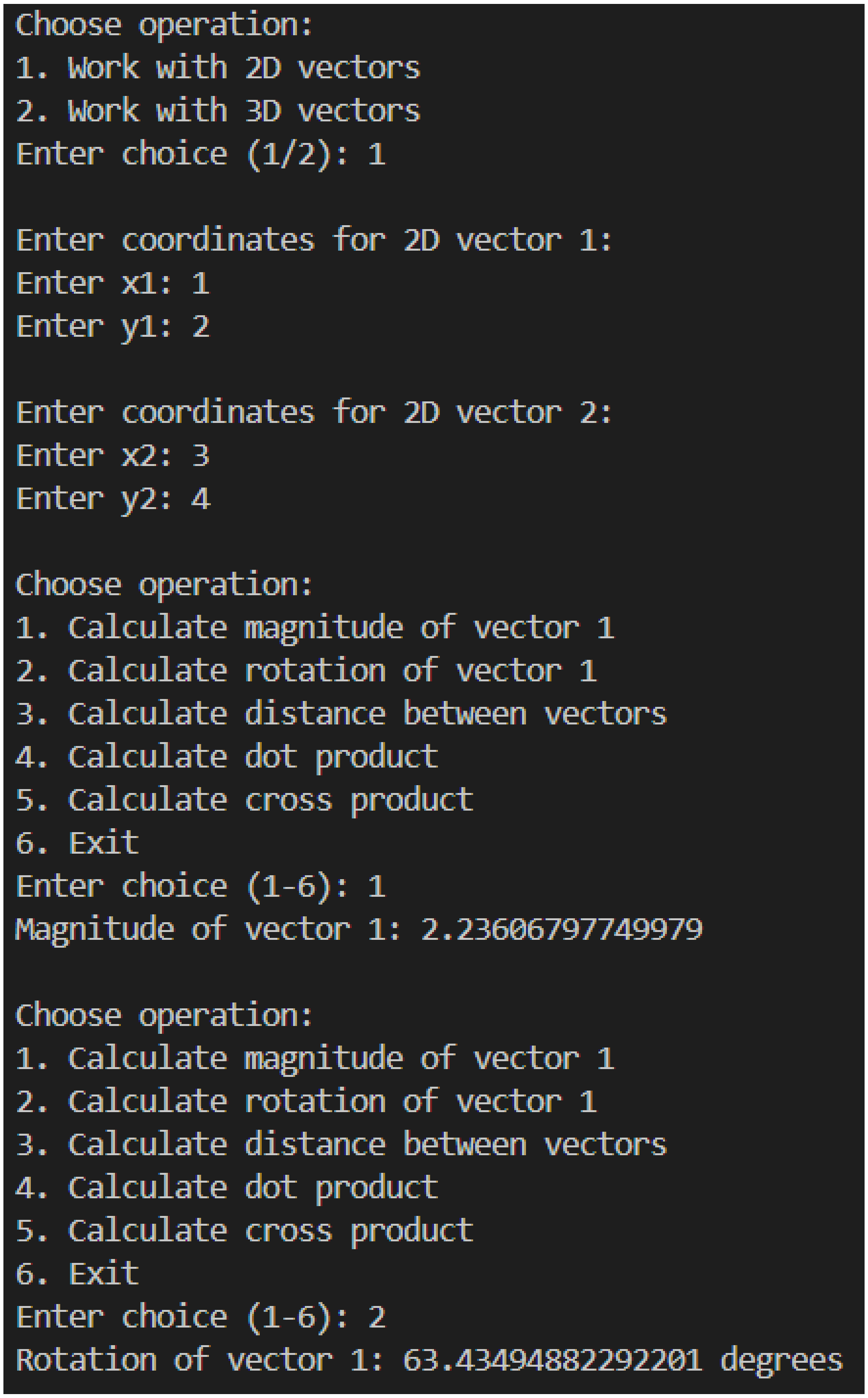
```
    def distance(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2 +
(self.z - other.z)**2)

    def dot_product(self, other):
        return self.x * other.x + self.y * other.y + self.z * other.z

    def cross_product(self, other):
        cx = self.y * other.z - self.z * other.y
        cy = self.z * other.x - self.x * other.z
        cz = self.x * other.y - self.y * other.x
        return Vector3D(cx, cy, cz)

    def __str__(self):
        return f"Vector3D({self.x}, {self.y}, {self.z})"

def main():
    print("Choose operation:")
    print("1. Work with 2D vectors")
    print("2. Work with 3D vectors")
    choice = input("Enter choice (1/2): ")

    if choice == '1':
        print("\nEnter coordinates for 2D vector 1:")
        x1 = float(input("Enter x1: "))
        y1 = float(input("Enter y1: "))
        v1 = Vector2D(x1, y1)

        print("\nEnter coordinates for 2D vector 2:")
        x2 = float(input("Enter x2: "))
        y2 = float(input("Enter y2: "))
        v2 = Vector2D(x2, y2)

        while True:
            print("\nChoose operation:")
            print("1. Calculate magnitude of vector 1")
            print("2. Calculate rotation of vector 1")
            print("3. Calculate distance between vectors")
            print("4. Calculate dot product")
```



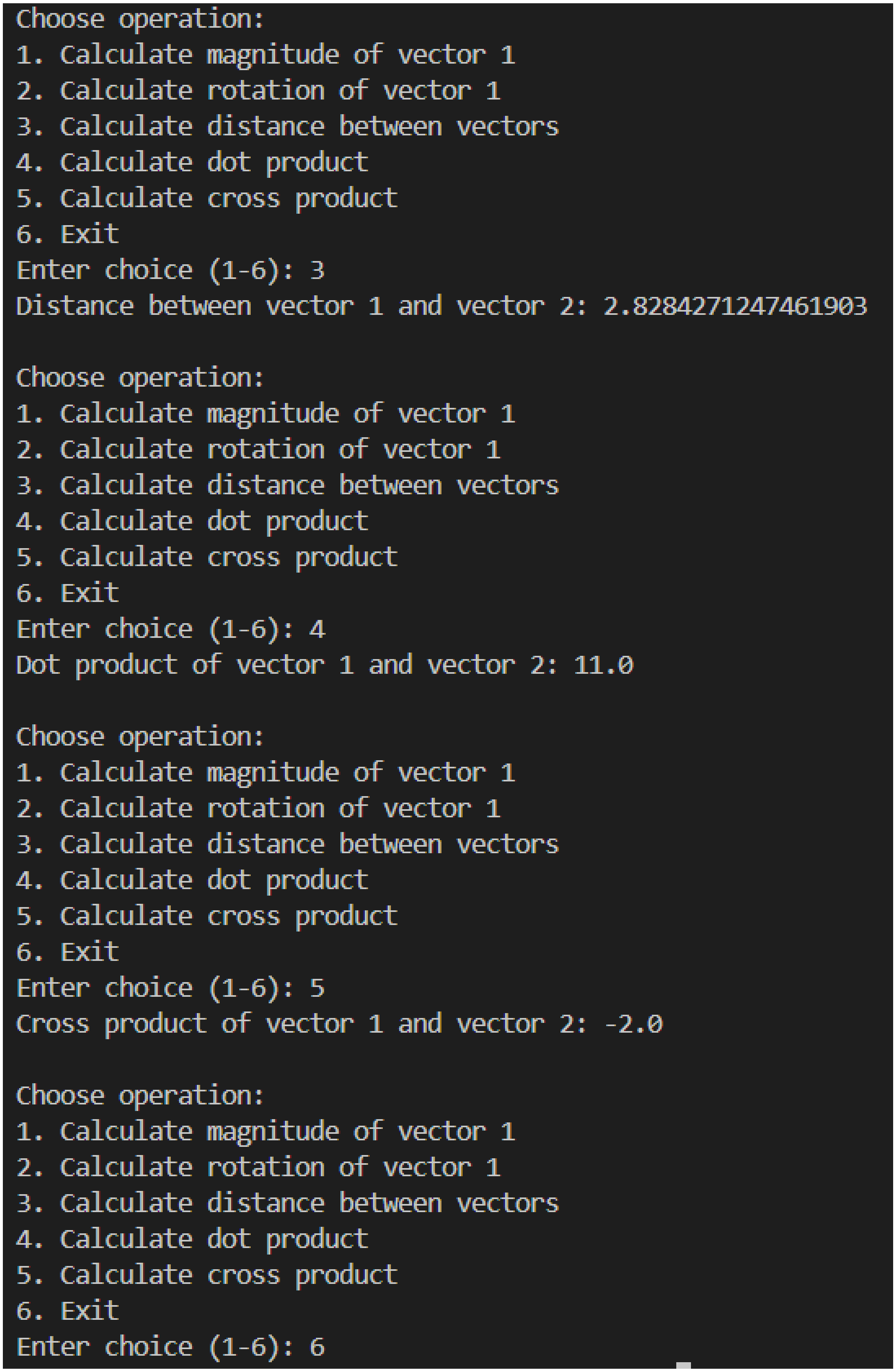
```
            print("5. Calculate cross product")
            print("6. Exit")
            operation = input("Enter choice (1-6): ")

            if operation == '1':
                print(f"Magnitude of vector 1: {v1.magnitude()}")
            elif operation == '2':
                print(f"Rotation of vector 1: {v1.rotation()} degrees")
            elif operation == '3':
                print(f"Distance between vector 1 and vector 2:
{v1.distance(v2)}")
            elif operation == '4':
                print(f"Dot product of vector 1 and vector 2:
{v1.dot_product(v2)}")
            elif operation == '5':
                print(f"Cross product of vector 1 and vector 2:
{v1.cross_product(v2)}")
            elif operation == '6':
                break
            else:
                print("Invalid choice!")

    elif choice == '2':
        print("\nEnter coordinates for 3D vector 1:")
        x1 = float(input("Enter x1: "))
        y1 = float(input("Enter y1: "))
        z1 = float(input("Enter z1: "))
        v1 = Vector3D(x1, y1, z1)

        print("\nEnter coordinates for 3D vector 2:")
        x2 = float(input("Enter x2: "))
        y2 = float(input("Enter y2: "))
        z2 = float(input("Enter z2: "))
        v2 = Vector3D(x2, y2, z2)

        while True:
            print("\nChoose operation:")
            print("1. Calculate magnitude of vector 1")
            print("2. Calculate rotation of vector 1")
            print("3. Calculate distance between vectors")
```



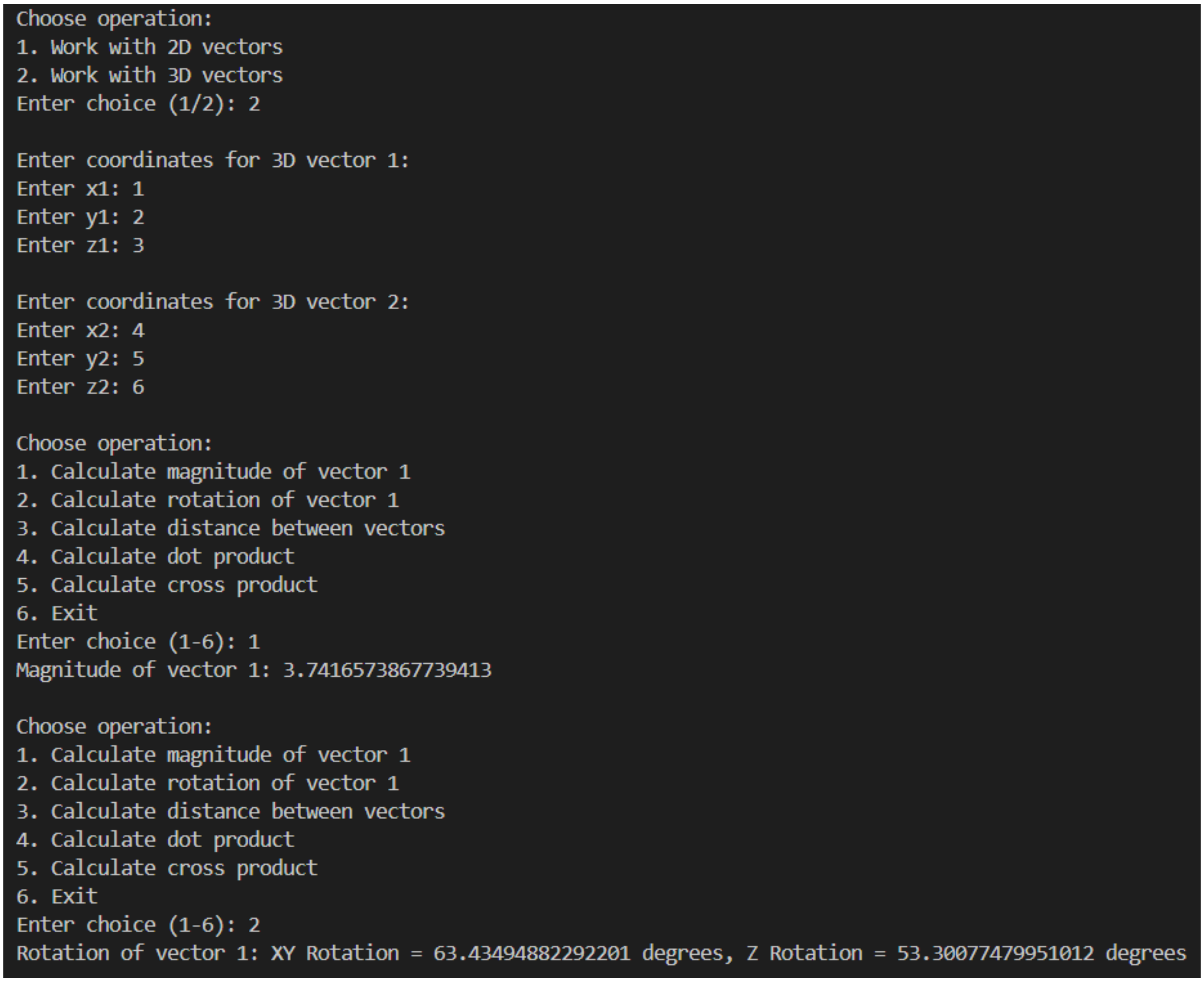
```
            print("4. Calculate dot product")
            print("5. Calculate cross product")
            print("6. Exit")
            operation = input("Enter choice (1-6): ")

            if operation == '1':
                print(f"Magnitude of vector 1: {v1.magnitude()}")
            elif operation == '2':
                xy_rotation, z_rotation = v1.rotation()
                print(f"Rotation of vector 1: XY Rotation = {xy_rotation}
degrees, Z Rotation = {z_rotation} degrees")
            elif operation == '3':
                print(f"Distance between vector 1 and vector 2:
{v1.distance(v2)}")
            elif operation == '4':
                print(f"Dot product of vector 1 and vector 2:
{v1.dot_product(v2)}")
            elif operation == '5':
                cross_prod = v1.cross_product(v2)
                print(f"Cross product of vector 1 and vector 2:
{cross_prod}")
            elif operation == '6':
                break
            else:
                print("Invalid choice!")

    else:
        print("Invalid choice!")

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



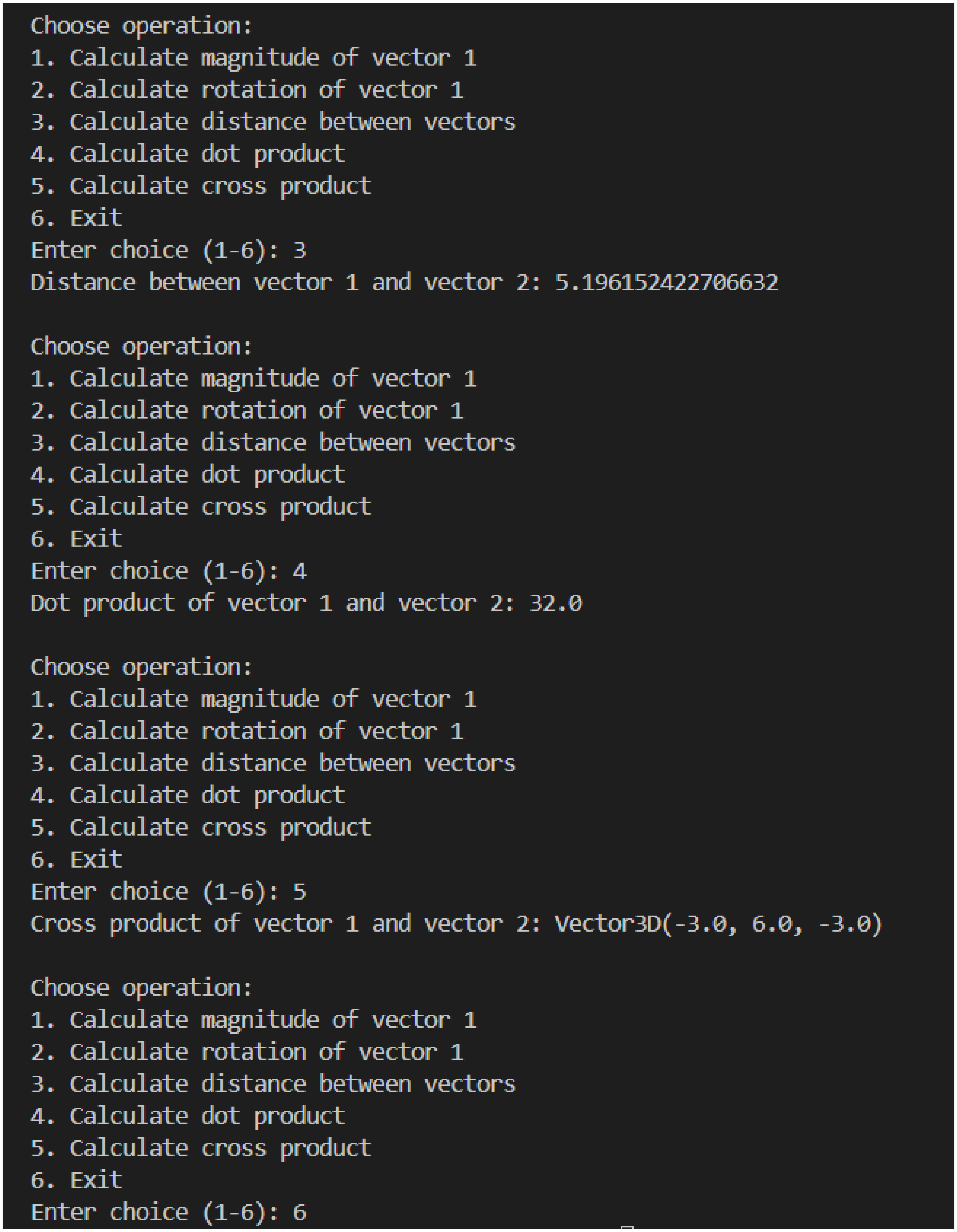
```
Choose operation:
1. Work with 2D vectors
2. Work with 3D vectors
Enter choice (1/2): 1

Enter coordinates for 2D vector 1:
Enter x1: 1
Enter y1: 2

Enter coordinates for 2D vector 2:
Enter x2: 3
Enter y2: 4

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 1
Magnitude of vector 1: 2.23606797749979

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 2
Rotation of vector 1: 63.43494882292201 degrees
```



```
Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 3
Distance between vector 1 and vector 2: 2.8284271247461903

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 4
Dot product of vector 1 and vector 2: 11.0

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 5
Cross product of vector 1 and vector 2: -2.0

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 6
```



```
Choose operation:
1. Work with 2D vectors
2. Work with 3D vectors
Enter choice (1/2): 2

Enter coordinates for 3D vector 1:
Enter x1: 1
Enter y1: 2
Enter z1: 3

Enter coordinates for 3D vector 2:
Enter x2: 4
Enter y2: 5
Enter z2: 6

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 1
Magnitude of vector 1: 3.7416573867739413

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 2
Rotation of vector 1: XY Rotation = 63.43494882292201 degrees, Z Rotation = 53.30077479951012 degrees
```



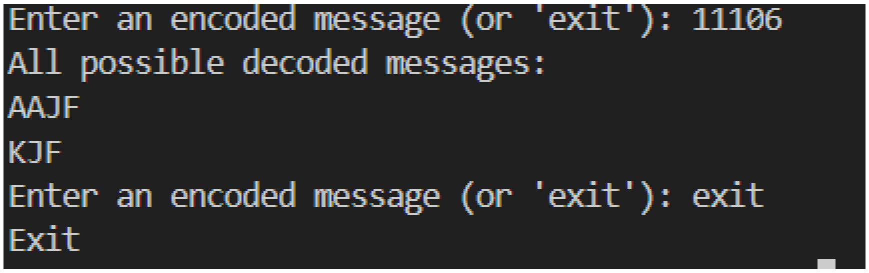
```
Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 3
Distance between vector 1 and vector 2: 5.196152422706632

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 4
Dot product of vector 1 and vector 2: 32.0

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 5
Cross product of vector 1 and vector 2: Vector3D(-3.0, 6.0, -3.0)

Choose operation:
1. Calculate magnitude of vector 1
2. Calculate rotation of vector 1
3. Calculate distance between vectors
4. Calculate dot product
5. Calculate cross product
6. Exit
Enter choice (1-6): 6
```

8. Decode the message:

A message containing the letters from A-Z can be encoded into the numbers using the mapping A-> 1, B-> 2, C-> 3, ..., Z-> 26. To decode an encoded message, you need to group the digits and do the reverse mapping. You are required to display all the possible decoded messages.
For example: "11106" can be decoded into:
a. "AAJF" with the grouping (1 1 10 6)
b. "KJF" with the grouping (11 10 6)

```
def decode_message(encoded_message):
    def decode_helper(index, current_decoded_message):
        if index == len(encoded_message):
            decoded_messages.append(current_decoded_message)
            return

        if index < len(encoded_message):
            one_digit = int(encoded_message[index])
            if 1 <= one_digit <= 9:
                decode_helper(index + 1, current_decoded_message +
chr(one_digit + ord('A') - 1))

        if index + 1 < len(encoded_message):
            two_digits = int(encoded_message[index:index + 2])
            if 10 <= two_digits <= 26:
                decode_helper(index + 2, current_decoded_message +
chr(two_digits + ord('A') - 1))

    decoded_messages = []
    decode_helper(0, "")
    return decoded_messages

def main():
    while True:
        encoded_message = input("Enter an encoded message (or 'exit'):
").strip()

        if encoded_message.lower() == 'exit':
            print("Exit")
            break

        if not encoded_message.isdigit():
            print("Invalid input. Please enter a valid encoded message
containing only numbers.")
```

```
            continue

        decoded_messages = decode_message(encoded_message)

        if decoded_messages:
            print("All possible decoded messages:")
            for message in decoded_messages:
                print(message)
        else:
            print("No valid decoding possible for the given message.")

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

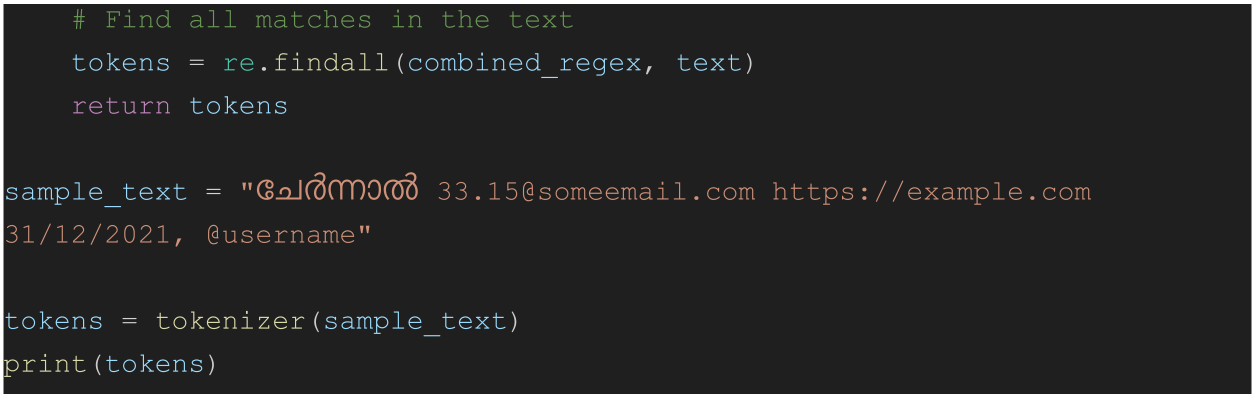


```
Enter an encoded message (or 'exit'): 11106
All possible decoded messages:
AAJF
KJF
Enter an encoded message (or 'exit'): exit
Exit
```

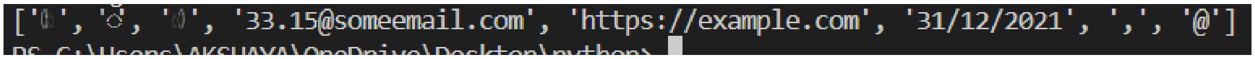
9. Create a tokenizer for your own language (mother tongue you speak). The tokenizer should tokenize punctuations, dates, urls, emails, numbers (in all different forms such as “33.15”,

“3,22,243”, “313/77”), social media usernames/user handles. Use regular expressions to design this. [Hint: Use unicode blocks for your language, check wikipedia pages]

```
import re

punctuation = r'[^\w\s]'
dates = r'\b(?:[0-2]?\d|3[01])[-/.](?:0?\d|1[0-2])[-/.](?:\d{2}|\d{4})\b'
urls = r'https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+'
emails = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b'
numbers = r'\b\d{1,3}(?:,\d{3})*(?:\.\d+)?|\d+/\d+\b'
usernames = r'@\w+'

combined_regex =
f'({punctuation}|{dates}|{urls}|{emails}|{numbers}|{usernames})'

def tokenizer(text):
```



```python
    # Find all matches in the text
    tokens = re.findall(combined_regex, text)
    return tokens

sample_text = "ചേർന്നാൽ 33.15@someemail.com https://example.com
31/12/2021, @username"

tokens = tokenizer(sample_text)
print(tokens)
```



```
['[illegible]', '[illegible]', '[illegible]', '33.15@someemail.com', 'https://example.com', '31/12/2021', ',', '@']
PS C:\Users\AKSHAYA\OneDrive\Desktop\python>
```