In Fulfillment of the Requirements for

CS 123 – Object-Oriented Programming

Final Project

Menu-oriented Program: Learning Set Operations

Presented to:

Dr. Unife O. Cagas

Professor V

Prepared by:

Pagalan, Stifhany D.

BS Computer Science – 2A1

May 2024

I. Project Description

This program is a menu-oriented program, designed to perform four set operations, such as subset checking, union, intersection, and symmetric difference, on two sets provided by the user. It utilizes object-oriented programming concepts, such as abstraction, inheritance, encapsulation, and polymorphism, to organize and execute these operations efficiently. It provides a console-based interface where users can input values for two sets, choose from various set operations: Subset, Union, Intersection, and Symmetric Difference. Then, displays the results with explanatory descriptions. The goal is to help learners understand basic concepts about union, intersection, subset, and symmetric difference, and to provide an interactive way for users to learn and practice set operations in python.

II. Objectives

The program aims to achieve the following:

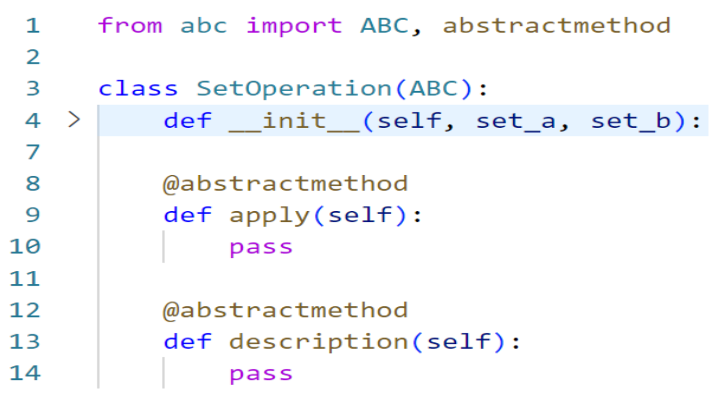
1. Applies the four principles of Object-Oriented Programming (OOP): Abstraction, Inheritance, Encapsulation, and Inheritance.
2. Creates a menu-oriented program that functions as a simple instructional tool, providing a way to simply teach and demonstrate the basic set operations to users.
3. Can perform the following set operations: Subset, Union, Intersection, and Symmetric difference.
4. Allow users to input values, displays the description of the set operation, and display the result.

III. Importance and Contribution of the Project

The importance of the program lies on its ability to provide an interactive way for users to learn and practice set operations like union, intersection, subset, and symmetric difference. By allowing users to input their sets and choose operations, the program promotes hands-on learning, which can lead to better understanding about sets operations. Users also can learn how to use or implement this set operations, and learn its equivalent function or method name in python. This program contributes to the users not just as a way of practicing and learning set operations, but also learn how the four principles of Object-Oriented Programming (OOP) can be used in a program, like this menu-oriented program.

IV. Four Principles of Object-Oriented Programming (OOP)

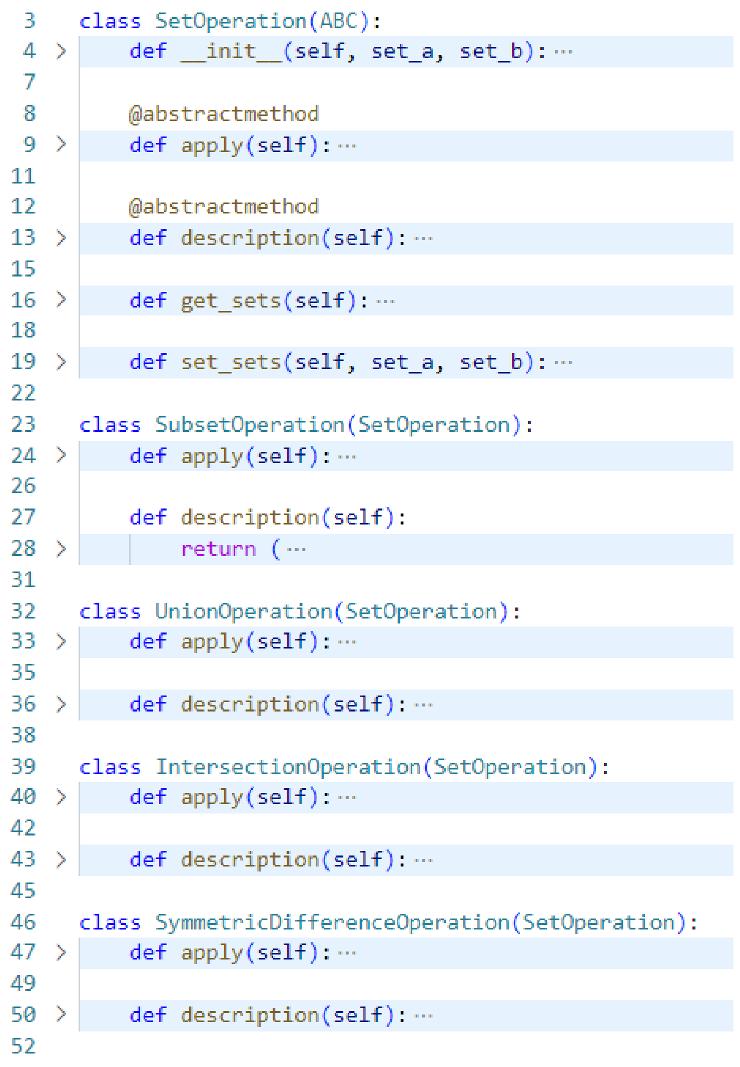
Abstraction:



```python
from abc import ABC, abstractmethod

class SetOperation(ABC):
    def __init__(self, set_a, set_b):

    @abstractmethod
    def apply(self):
        pass

    @abstractmethod
    def description(self):
        pass
```

Applying abstraction, I import the Abstract Base Class (ABC) module so that I can define abstract base classes and abstract methods. Next, I define an abstract base class named ‘SetOperation’ that inherits from the ABC module. This serves as the primary building block for implementing and organizing the set operations (subset, union, intersection, and symmetric difference) in the program. Then, I define two abstract method decorators (@abstractmethod) before defining two methods: ‘apply’ and ‘description’, the decorator marks these methods as abstract methods in the program, indicating that the subclasses must implement each method.

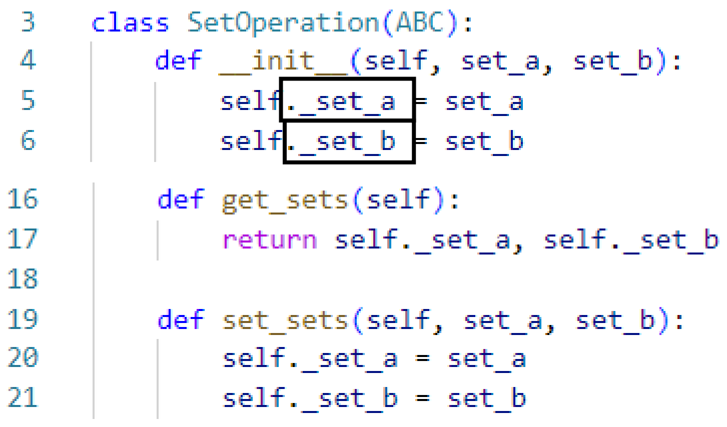
Inheritance:



```
class SetOperation(ABC):
    def __init__(self, set_a, set_b):···

    @abstractmethod
    def apply(self):···

    @abstractmethod
    def description(self):···

    def get_sets(self):···

    def set_sets(self, set_a, set_b):···

class SubsetOperation(SetOperation):
    def apply(self):···

    def description(self):
        return (···

class UnionOperation(SetOperation):
    def apply(self):···

    def description(self):···

class IntersectionOperation(SetOperation):
    def apply(self):···

    def description(self):···

class SymmetricDifferenceOperation(SetOperation):
    def apply(self):···

    def description(self):···
```

Applying Inheritance, in this program, the abstract base class ‘SetOperations’ will also be the parent class. This will be the class whose methods and attributes are inherited by its child classes. Its child classes (sub classes) are: SubsetOperation, UnionOperation, IntersectionOperation, and SymmetricDifferenceOperation. These child classes inherits the methods and attributes of their parent class ‘SetOperations’, but have different implementations of the methods from their parent class.

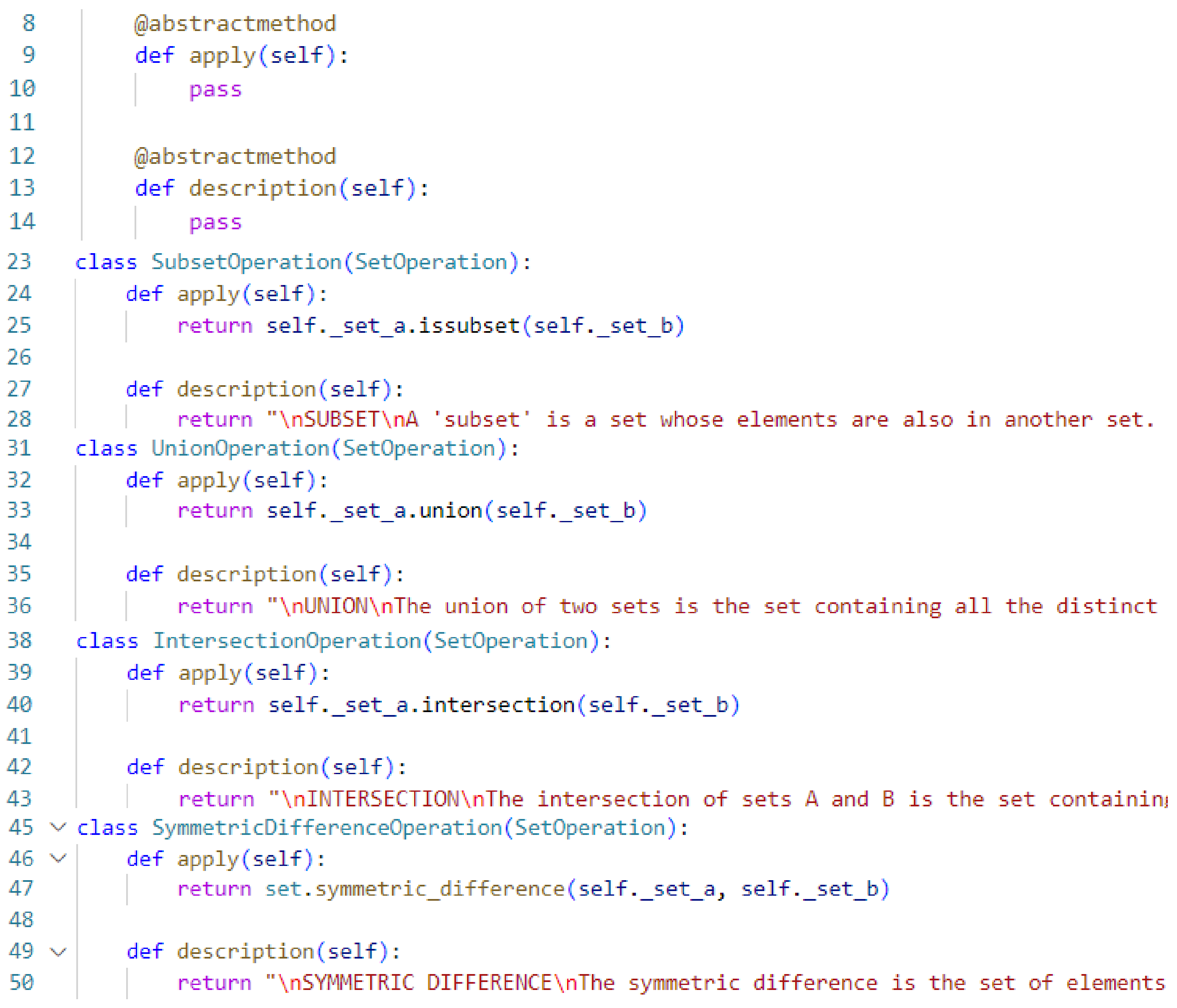
Encapsulation:



```
class SetOperation(ABC):
    def __init__(self, set_a, set_b):
        self._set_a = set_a
        self._set_b = set_b

    def get_sets(self):
        return self._set_a, self._set_b

    def set_sets(self, set_a, set_b):
        self._set_a = set_a
        self._set_b = set_b
```

Encapsulation describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. In this program, I apply encapsulation by creating a protected member, by prefixing the name of the member by a single underscore “_”. In this program, ‘_set_a’ and ‘_set_b’ is a protected member. Encapsulating _set_a and _set_b helps promote data hiding, controlled access, abstraction of implementation details, and flexibility in the design of the SetOperation class and its subclasses, aligning with the principles of encapsulation in object-oriented programming.

Polymorphism:



```
    @abstractmethod
    def apply(self):
        pass

    @abstractmethod
    def description(self):
        pass
```

```
class SubsetOperation(SetOperation):
    def apply(self):
        return self._set_a.issubset(self._set_b)

    def description(self):
        return "\nSUBSET\nA 'subset' is a set whose elements are also in another set.
class UnionOperation(SetOperation):
    def apply(self):
        return self._set_a.union(self._set_b)

    def description(self):
        return "\nUNION\nThe union of two sets is the set containing all the distinct
class IntersectionOperation(SetOperation):
    def apply(self):
        return self._set_a.intersection(self._set_b)

    def description(self):
        return "\nINTERSECTION\nThe intersection of sets A and B is the set containin
class SymmetricDifferenceOperation(SetOperation):
    def apply(self):
        return set.symmetric_difference(self._set_a, self._set_b)

    def description(self):
        return "\nSYMMETRIC DIFFERENCE\nThe symmetric difference is the set of elements
```

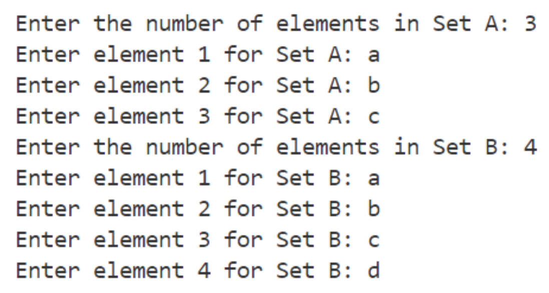
Polymorphism refers to methods/functions/operators with the same name that can be executed on many objects or classes. In this program, polymorphism is implemented through method overriding in the subclasses of the abstract base class ‘SetOperation’. The ‘SetOperation’ class defines two abstract methods ‘apply’ and ‘description’. These methods are present in each subclass, and they have different implementation of this methods or we can say it overrides the subclasses implementing different operation and descriptions. For example, the subclass ‘SubsetOperation’ use issubset() method in its ‘apply’ method, while the subclass ‘UnionOperation’ use the union() method in its ‘apply’ method. As for the ‘description’ method, the subclass ‘SubsetOperation’ will return a description of a Subset operation, its meaning, how it works, and its symbol which is different from the other subclass who have different description and usage for its respective set operations.

V. Hardware and Software Used

In creating this program, the hardware used is a laptop and a cellphone. The software used is Visual Studio Code on the laptop and the Pydroid mobile application on the cellphone.

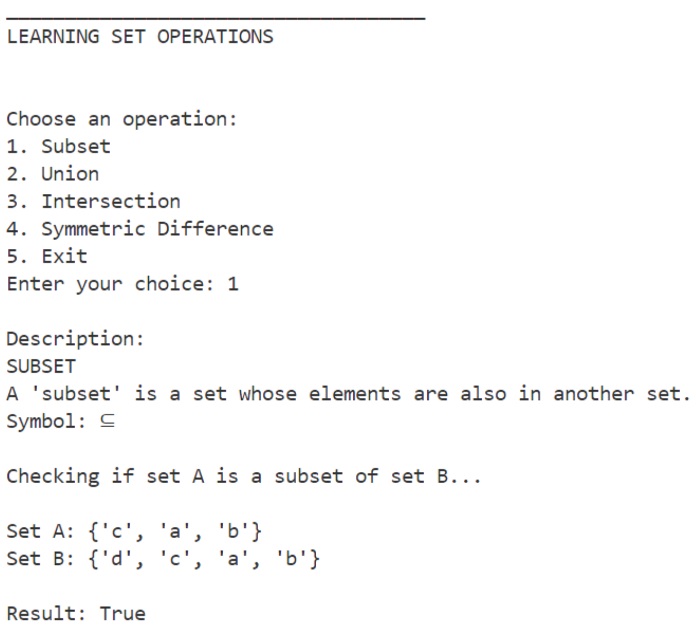
VI. Output

This is the possible output of the program, providing the following input values:

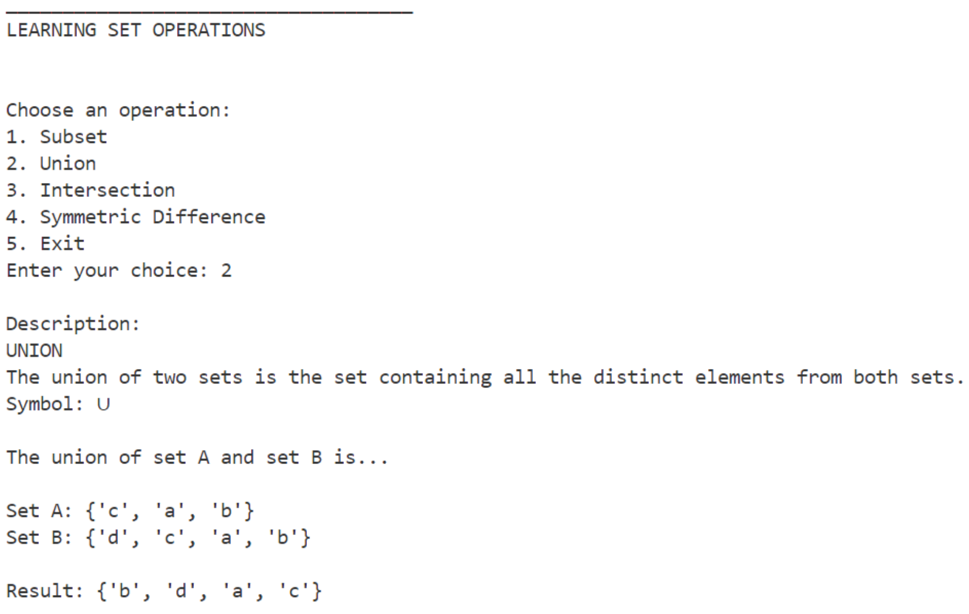


```
Enter the number of elements in Set A: 3
Enter element 1 for Set A: a
Enter element 2 for Set A: b
Enter element 3 for Set A: c
Enter the number of elements in Set B: 4
Enter element 1 for Set B: a
Enter element 2 for Set B: b
Enter element 3 for Set B: c
Enter element 4 for Set B: d
```

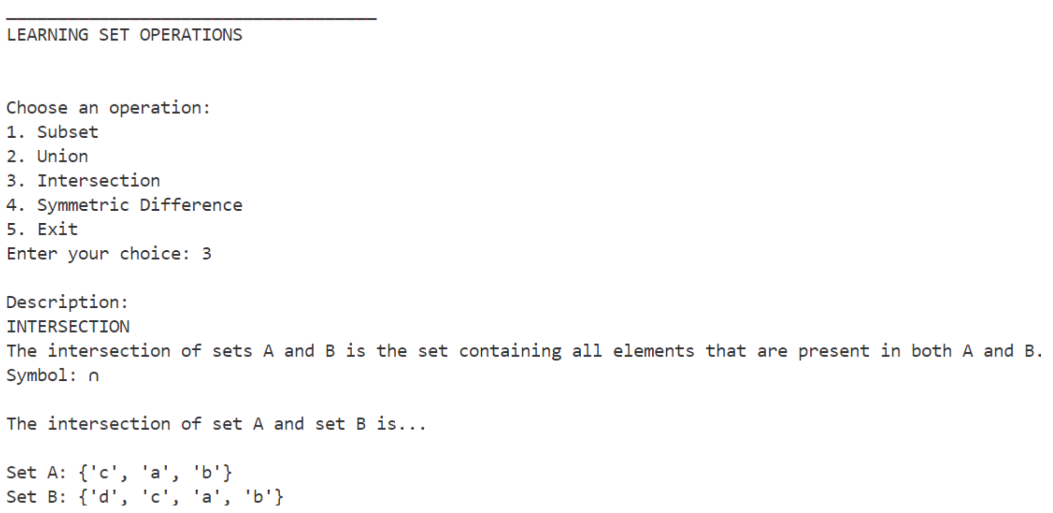
After the user inputs the values, they will be stored in their respective sets. These values in each set are then used for the application of the set operations: Subset, Union, Intersection, and Symmetric Difference. These values will be used throughout the execution process, depending on the user's chosen operation, until the user decides to exit the program, by choosing the option 5. If the user chooses option 5, the user can execute again the program to input another set of values.



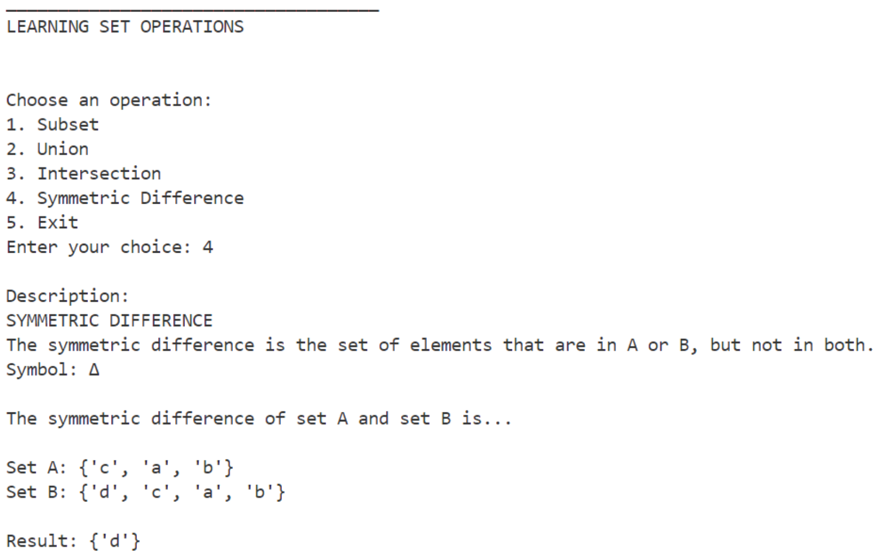
```
________________________________________
LEARNING SET OPERATIONS


Choose an operation:
1. Subset
2. Union
3. Intersection
4. Symmetric Difference
5. Exit
Enter your choice: 1

Description:
SUBSET
A 'subset' is a set whose elements are also in another set.
Symbol: ⊆

Checking if set A is a subset of set B...

Set A: {'c', 'a', 'b'}
Set B: {'d', 'c', 'a', 'b'}

Result: True
```



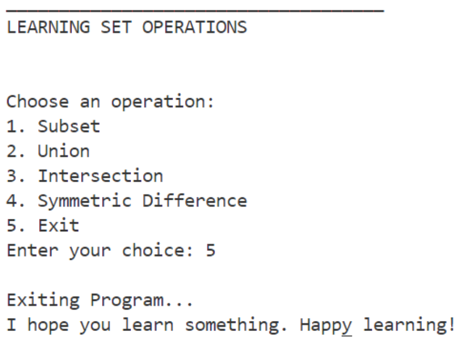
```
________________________________________
LEARNING SET OPERATIONS


Choose an operation:
1. Subset
2. Union
3. Intersection
4. Symmetric Difference
5. Exit
Enter your choice: 2

Description:
UNION
The union of two sets is the set containing all the distinct elements from both sets.
Symbol: ∪

The union of set A and set B is...

Set A: {'c', 'a', 'b'}
Set B: {'d', 'c', 'a', 'b'}

Result: {'b', 'd', 'a', 'c'}
```



```
LEARNING SET OPERATIONS


Choose an operation:
1. Subset
2. Union
3. Intersection
4. Symmetric Difference
5. Exit
Enter your choice: 3

Description:
INTERSECTION
The intersection of sets A and B is the set containing all elements that are present in both A and B.
Symbol: ∩

The intersection of set A and set B is...

Set A: {'c', 'a', 'b'}
Set B: {'d', 'c', 'a', 'b'}
```



```
LEARNING SET OPERATIONS


Choose an operation:
1. Subset
2. Union
3. Intersection
4. Symmetric Difference
5. Exit
Enter your choice: 4

Description:
SYMMETRIC DIFFERENCE
The symmetric difference is the set of elements that are in A or B, but not in both.
Symbol: Δ

The symmetric difference of set A and set B is...

Set A: {'c', 'a', 'b'}
Set B: {'d', 'c', 'a', 'b'}

Result: {'d'}
```



```
LEARNING SET OPERATIONS


Choose an operation:
1. Subset
2. Union
3. Intersection
4. Symmetric Difference
5. Exit
Enter your choice: 5

Exiting Program...
I hope you learn something. Happy learning!
```

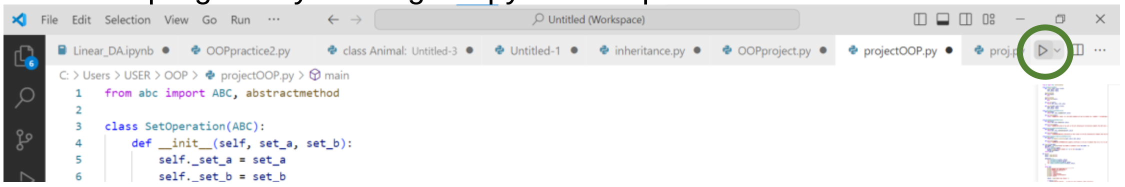
VII. Codes

```
from abc import ABC, abstractmethod  # This application is adapted from Jenny's Lectures CS IT (https://www.youtube.com/watch?v=64tJdFVh1Vs)


class SetOperation(ABC): # Define an abstract base (parent) class for set operations.
    def __init__(self, set_a, set_b):
        self._set_a = set_a # Applying Encapsulation. Protected Member
        self._set_b = set_b # Applying Encapsulation. Protected Member

# Define two abstract methods for applying the set operation and its description.
    @abstractmethod
    def apply(self):
        pass # Define an abstract method for applying the set operation
    @abstractmethod
    def description(self):
        pass # No implementation here, subclasses will implement this.

    def get_sets(self): # A method to retrieve the current sets stored.
        return self._set_a, self._set_b

    def set_sets(self, set_a, set_b): # A method to set new sets.
        self._set_a = set_a
        self._set_b = set_b

class SubsetOperation(SetOperation): # Child class or subclass 'SubsetOperation',
inheriting from SetOperation.
    def apply(self):
        return self._set_a.issubset(self._set_b)

    def description(self):
        return "\nSUBSET\nA 'subset' is a set whose elements are also in another set.
\nSymbol: ⊆ \n\nChecking if set A is a subset of set B..."

class UnionOperation(SetOperation): # Child class or subclass 'UnionOperation', inheriting
from SetOperation.
    def apply(self):
        return self._set_a.union(self._set_b)
    def description(self):
        return "\nUNION\nThe union of two sets is the set containing all the distinct
elements from both sets. \nSymbol: ∪ \n\nThe union of set A and set B is..."

class IntersectionOperation(SetOperation): # Child class 'IntersectionOperation',
inheriting from SetOperation.
    def apply(self):
        return self._set_a.intersection(self._set_b)

    def description(self):
        return "\nINTERSECTION\nThe intersection of sets A and B is the set containing all
elements that are present in both A and B. \nSymbol: ∩ \n\nThe intersection of set A and
set B is..."

class SymmetricDifferenceOperation(SetOperation): # Child class
'SymmetricDifferenceOperation', inheriting from SetOperation.
    def apply(self):
        return set.symmetric_difference(self._set_a, self._set_b)

    def description(self):
        return "\nSYMMETRIC DIFFERENCE\nThe symmetric difference is the set of elements
that are in A or B, but not in both. \nSymbol: Δ \n\nThe symmetric difference of set A and
set B is..."

# Functions to read a set from user input
def read_set(set_name):
    set_length = int(input(f"Enter the number of elements in Set {set_name}: "))
    new_set = set()
    for i in range(set_length):
```

```python
        element = input(f"Enter element {i + 1} for Set {set_name}: ")
        new_set.add(element)
    return new_set

def main(): # The main function to run the set operation
    set_a = read_set("A")
    set_b = read_set("B")

    # Creating a dictionary of set operations, mapped to their choices.
    operations = {
    "1": SubsetOperation(set_a, set_b),
    "2": UnionOperation(set_a, set_b),
    "3": IntersectionOperation(set_a, set_b),
    "4": SymmetricDifferenceOperation(set_a, set_b),
    }

    while True:
        # Display the menu for choosing a set operation.
        print("____________________________________")
        print("LEARNING SET OPERATIONS\n")
        print("\nChoose an operation:")
        print("1. Subset")
        print("2. Union")
        print("3. Intersection")
        print("4. Symmetric Difference")
        print("5. Exit")

        choice = input("Enter your choice: ") # Get user input from the menu

        # If the user chooses 5, it will exit the loop and end the program.
        if choice == "5":
            print("\nExiting Program... \nI hope you learn something. Happy learning!")
            break

        operation = operations.get(choice)

        # If a valid operation was chosen, perform it.
        if operation:
            print(f"\nDescription: {operation.description()}")
            set_a, set_b = operation.get_sets()
            print(f"\nSet A: {set_a}")
            print(f"Set B: {set_b}")
            result = operation.apply()
            print(f"\nResult: {result}")
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main() # If the program script is executed, call the main function and start the
program.
```

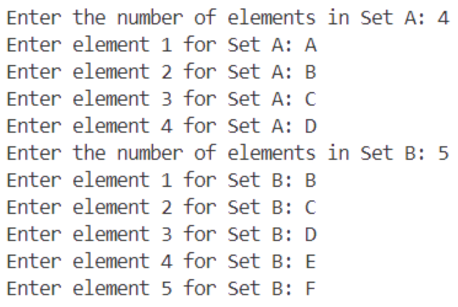
VIII. User Guide

Step 1: Start the program by running the python script.



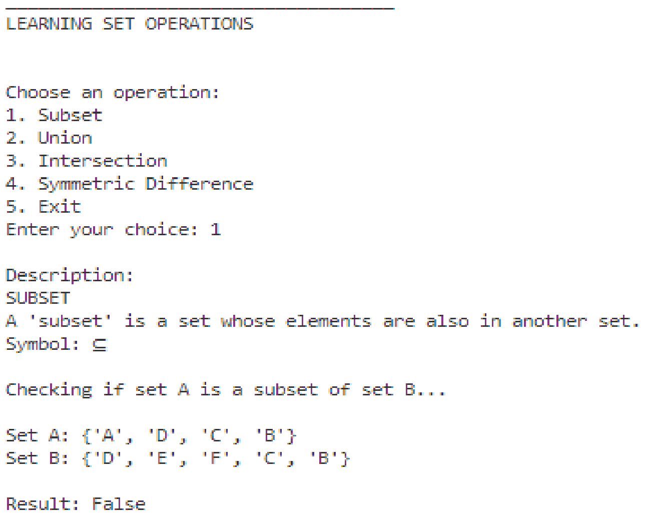
```python
from abc import ABC, abstractmethod

class SetOperation(ABC):
    def __init__(self, set_a, set_b):
        self._set_a = set_a
        self._set_b = set_b
```

Step 2: Input the values for the number of elements to be contained by each set and the values for each set.



```
Enter the number of elements in Set A: 4
Enter element 1 for Set A: A
Enter element 2 for Set A: B
Enter element 3 for Set A: C
Enter element 4 for Set A: D
Enter the number of elements in Set B: 5
Enter element 1 for Set B: B
Enter element 2 for Set B: C
Enter element 3 for Set B: D
Enter element 4 for Set B: E
Enter element 5 for Set B: F
```

Step 3: Enter your chosen operation to be performed in the two sets, and the program will print the result of your chosen operation along with a description of your chosen set operation.



```
LEARNING SET OPERATIONS


Choose an operation:
1. Subset
2. Union
3. Intersection
4. Symmetric Difference
5. Exit
Enter your choice: 1

Description:
SUBSET
A 'subset' is a set whose elements are also in another set.
Symbol: ⊆

Checking if set A is a subset of set B...

Set A: {'A', 'D', 'C', 'B'}
Set B: {'D', 'E', 'F', 'C', 'B'}

Result: False
```

IX. References

1. Python sets operations: https://www.w3schools.com/python/python_ref_set.asp

2. Python abstract class & abstract method: https://www.youtube.com/watch?v=64tJdFVh1Vs

3. Python Polymorphism: https://www.w3schools.com/PYTHON/python_polymorphism.asp

4. Python Inheritance: https://www.youtube.com/watch?v=e2qLUPDDpvk&t=457s

5. Python Encapsulation: https://www.geeksforgeeks.org/encapsulation-in-python/

6. Python Abstraction: https://www.geeksforgeeks.org/data-abstraction-in-python/