In Partial Fulfillment of the Requirements for the

CS 223 - Object-Oriented Programming

"FOUR PRINCIPLES OF OBJECT ORIENTED PROGRAMMING"

Presented to:

Dr. Unife O. Cagas
Professor

Prepared by:

Tenazas, Danna Rose A.
BSCS 2A2 Student

"Shape Calculator"

Project Title

PROJECT DESCRIPTION

The Shape Calculator is a Python application designed to provide accurate perimeter and area estimations for various geometric shapes. It uses inheritance, polymorphism, abstraction, and encapsulation concepts in object-oriented programming to provide a strong foundation for mathematical computations. The software has several classes, each specifically designed to depict different geometric shapes like Triangle, Square, Rectangle, and Circle. Encapsulating these shapes' distinct behaviors into special classes improves readability, maintainability, and code reusability. The Shape Calculator is an invaluable tool for users learning about geometric calculations, as it simplifies the process of traveling between different shapes. It showcases the effectiveness of object-oriented programming paradigms, allowing users to explore mathematical concepts with accuracy and adaptability.

OBJECTIVES

The Shape Calculator Python program aims to provide a flexible and user-friendly tool for calculating the area and perimeter of various geometric shapes. Built with a focus on simplicity and extensibility, it allows users to interactively input parameters for different shapes and receive immediate calculations of their properties.

Modularity: The code is designed with a modular structure, utilizing object-oriented programming (OOP) principles to define classes for each shape. This modularity facilitates easy maintenance, extension, and reuse of code components.

Ease of Use: The program features an intuitive command-line interface, guiding users through the process of selecting shapes and inputting parameters. This simplicity ensures that users, regardless of their technical background, can easily utilize the tool.

Accuracy: Emphasis is placed on accuracy in the calculations of area and perimeter for each shape. The program leverages well-established mathematical formulas and the math module in Python to ensure precise results.

Interactivity: By providing an interactive interface, the program enhances user engagement and allows for real-time feedback. Users can dynamically explore the properties of different shapes, fostering a deeper understanding of geometric concepts.

IMPORTANCE AND CONTRIBUTION OF THE PROJECT

The "Shape Calculator" project serves as a valuable educational tool and practical utility, offering several key contributions and importance:

1. **Education Enhancement**: This project aids in the understanding of geometric concepts by providing a hands-on experience in calculating areas and perimeters of various shapes. It can be utilized in educational settings, such as classrooms or online learning platforms, to reinforce mathematical concepts in geometry.

2. **Problem-Solving Skills**: By engaging with the codebase, users can develop problem-solving skills, especially in the context of programming. They learn to translate mathematical formulas into code and implement algorithms for shape calculations, fostering computational thinking.

3. **Flexibility and Extensibility**: The project's design allows for easy expansion to include additional shapes beyond circles, rectangles, triangles, and squares. This extensibility promotes exploration and experimentation, encouraging users to contribute new shape implementations or modify existing ones.

4. **Error Handling and Input Validation**: The inclusion of error handling mechanisms ensures robustness and reliability, providing users with informative feedback in case of invalid inputs. This aspect cultivates good programming practices and emphasizes the importance of handling edge cases.

5. **Practical Utility**: Beyond educational purposes, the Shape Calculator has practical utility in real-world scenarios. Professionals in fields such as architecture, engineering, and design can leverage it to quickly compute geometric properties of shapes encountered in their work, facilitating faster decision-making and problem-solving.

6. **Open Source Collaboration:** As an open-source project, the Shape Calculator encourages collaboration and community involvement. Developers can contribute enhancements, bug fixes, or optimizations, thereby enriching the project's functionality and fostering a culture of knowledge sharing and teamwork.

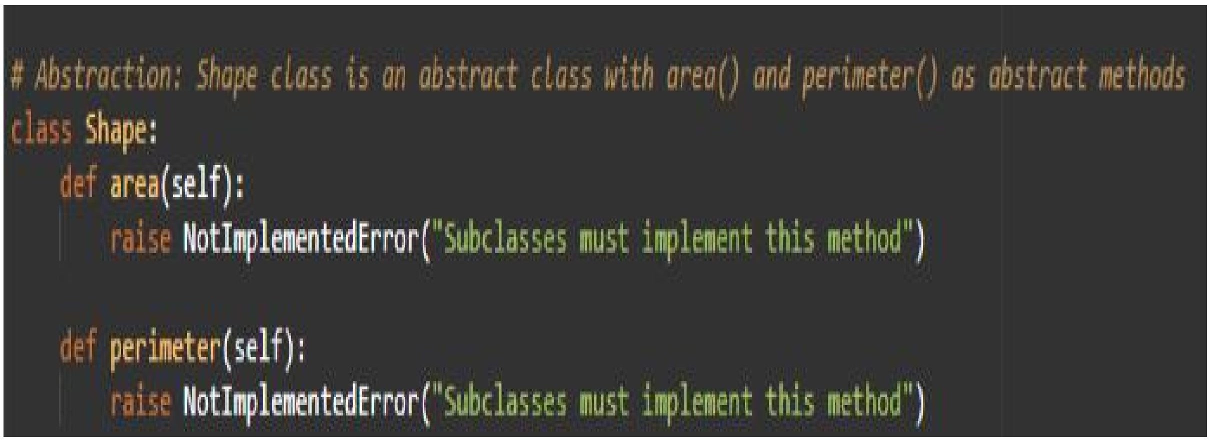
7. **Documentation and Learning Resources**: The project's documentation, including the provided introduction and code comments, serves as a learning resource for individuals seeking to understand Python programming concepts, object-oriented design principles, and mathematical applications in software development.

Overall, the Shape Calculator project not only serves as a practical tool for shape calculations but also makes significant contributions to education, problem-solving skills development, and collaborative learning within the programming community. Its versatility, educational value, and potential for real-world applications underscore its importance in both educational and professional contexts.

FOUR PRINCIPLES OF OOP WITH CODE

ABSTRACT

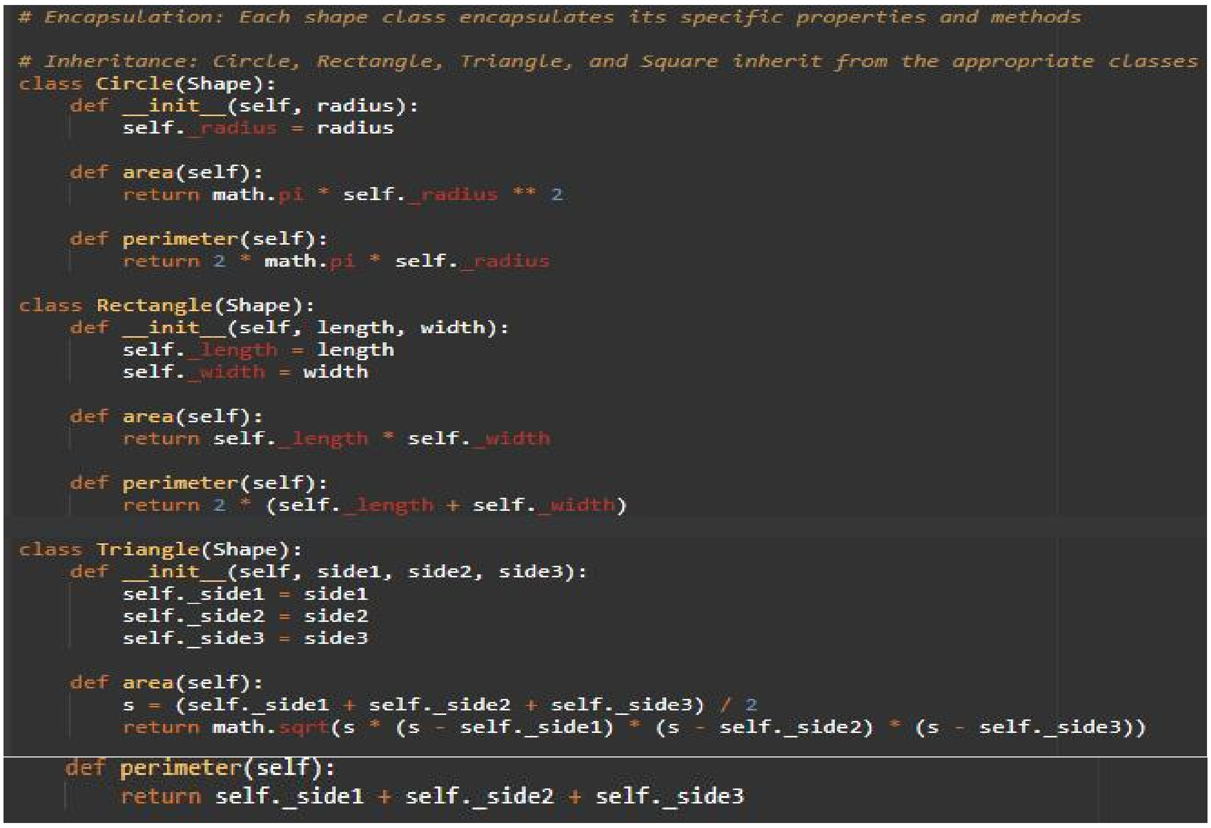
The Shape class serves as an abstract base class. It declares abstract methods area() and perimeter() which all subclasses must implement. This abstraction allows for a unified interface across different shapes, even though their implementations may vary.



```python
# Abstraction: Shape class is an abstract class with area() and perimeter() as abstract methods
class Shape:
    def area(self):
        raise NotImplementedError("Subclasses must implement this method")

    def perimeter(self):
        raise NotImplementedError("Subclasses must implement this method")
```

ENCAPSULATION

Each shape class (Circle, Rectangle, Triangle, Square) encapsulates its specific properties and methods within its own class definition. For example, the Circle class encapsulates the radius property and methods to calculate area and perimeter specific to circles. This helps in organizing and managing code, as well as preventing direct access to internal data from outside the class.

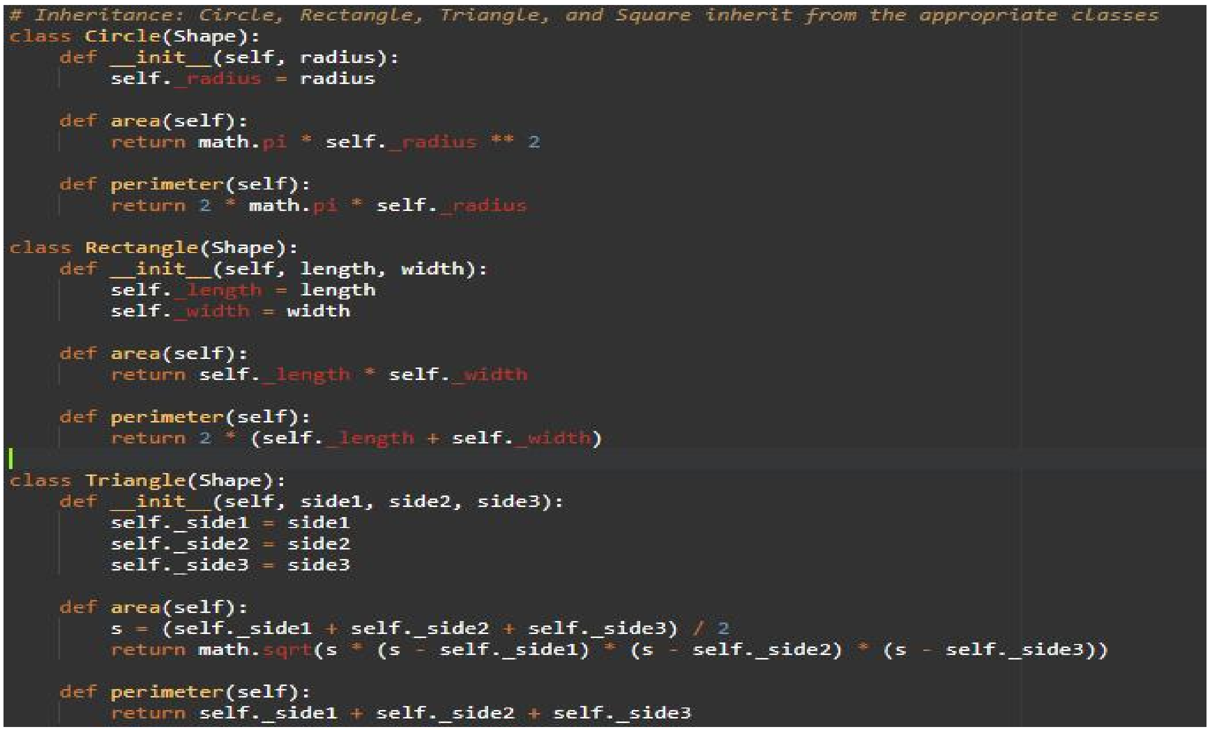


```
# Encapsulation: Each shape class encapsulates its specific properties and methods

# Inheritance: Circle, Rectangle, Triangle, and Square inherit from the appropriate classes
class Circle(Shape):
    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

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 Triangle(Shape):
    def __init__(self, side1, side2, side3):
        self._side1 = side1
        self._side2 = side2
        self._side3 = side3

    def area(self):
        s = (self._side1 + self._side2 + self._side3) / 2
        return math.sqrt(s * (s - self._side1) * (s - self._side2) * (s - self._side3))

    def perimeter(self):
        return self._side1 + self._side2 + self._side3
```

INHERITANCE

Inheritance is demonstrated by how subclasses (Circle, Rectangle, Triangle, Square) inherit from the appropriate parent class (Shape, Rectangle).

Each subclass inherits the abstract methods area() and perimeter() from the Shape class and provides its own implementation.

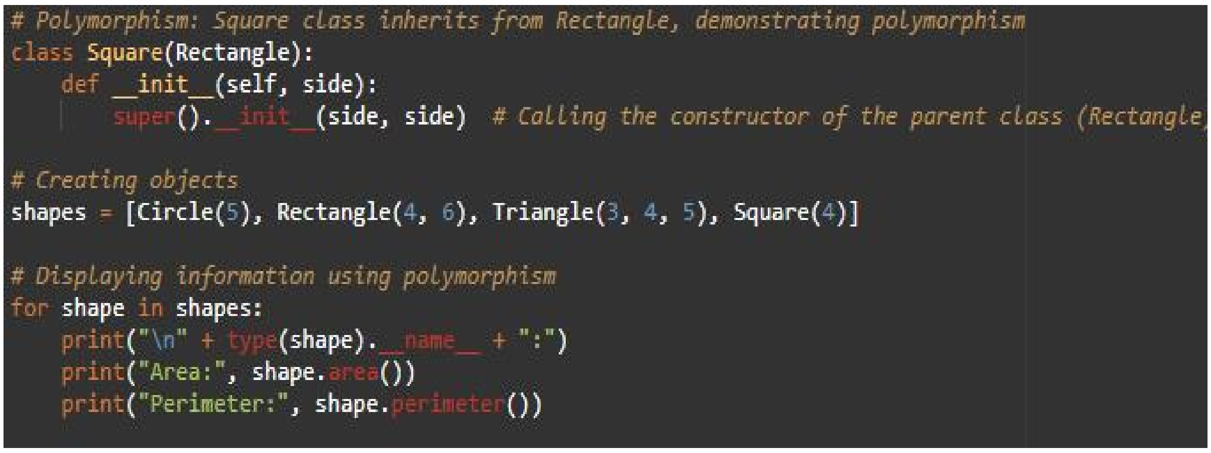


```python
# Inheritance: Circle, Rectangle, Triangle, and Square inherit from the appropriate classes
class Circle(Shape):
    def __init__(self, radius):
        self._radius = radius

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

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

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 Triangle(Shape):
    def __init__(self, side1, side2, side3):
        self._side1 = side1
        self._side2 = side2
        self._side3 = side3

    def area(self):
        s = (self._side1 + self._side2 + self._side3) / 2
        return math.sqrt(s * (s - self._side1) * (s - self._side2) * (s - self._side3))

    def perimeter(self):
        return self._side1 + self._side2 + self._side3
```

POLYMORPHISM

Polymorphism is demonstrated in several ways:

Method overriding: Each subclass provides its own implementation of the area() and perimeter() methods, allowing the same method calls (shape.area() and shape.perimeter()) to produce different behaviors depending on the type of shape. Square class demonstrates polymorphism by inheriting from Rectangle and providing its own implementation. Despite being a subclass of Rectangle, it behaves differently because it sets both length and width to the same value. In the loop where objects are iterated through (for shape in shapes), polymorphism is evident as the same method calls are used (shape.area() and shape.perimeter()) on different types of shapes, and the appropriate implementation is called based on the actual type of the object being referenced.



```
# Polymorphism: Square class inherits from Rectangle, demonstrating polymorphism
class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)  # Calling the constructor of the parent class (Rectangle

# Creating objects
shapes = [Circle(5), Rectangle(4, 6), Triangle(3, 4, 5), Square(4)]

# Displaying information using polymorphism
for shape in shapes:
    print("\n" + type(shape).__name__ + ":")
    print("Area:", shape.area())
    print("Perimeter:", shape.perimeter())
```

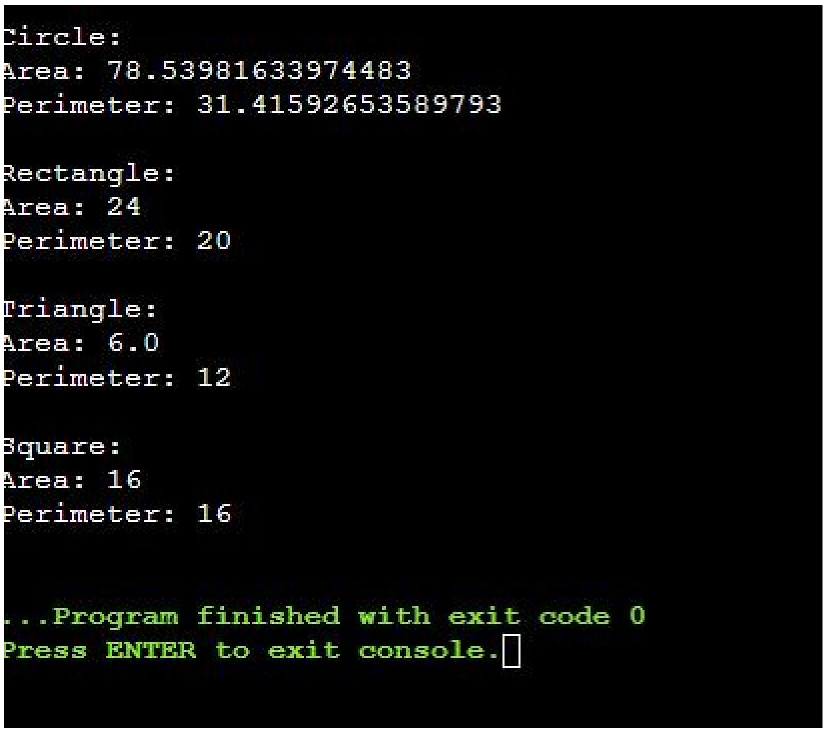
SOFTWARE USED:

- ONLINE GDB
- PYROID APP

HARDWARE USED:

- SCHOOL PC
- MOBILE PHONE
- LAPTOP

OUTPUT



```
Circle:
Area: 78.53981633974483
Perimeter: 31.41592653589793

Rectangle:
Area: 24
Perimeter: 20

Triangle:
Area: 6.0
Perimeter: 12

Square:
Area: 16
Perimeter: 16


...Program finished with exit code 0
Press ENTER to exit console.
```

Base on the output, there are four shapes in the code provided. The first shape in the list is a Circle with a radius of 5 units, which we'll name "Circle A". Its area is approximately 25π square units, and its perimeter is approximately 10π units.

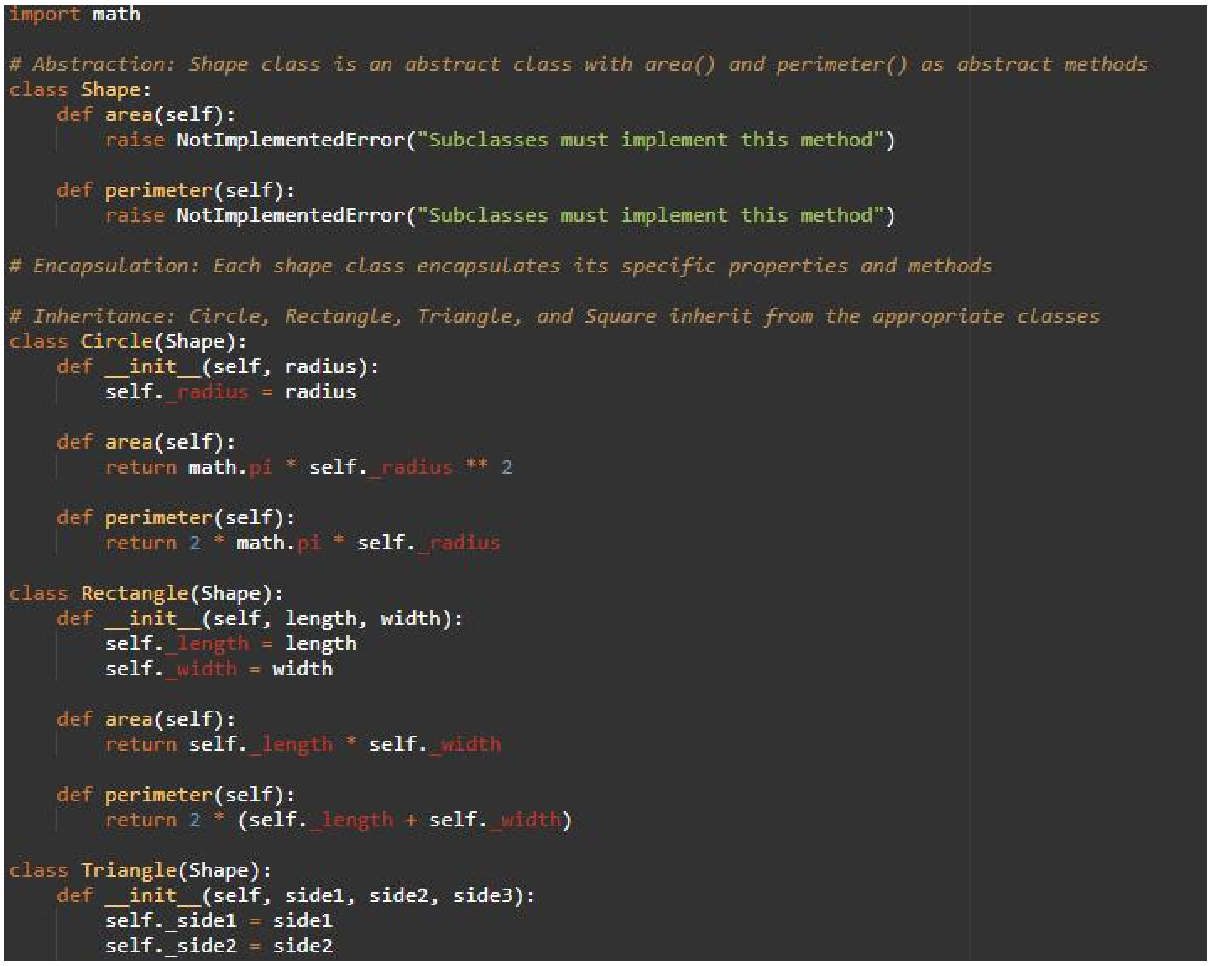
The second shape represents a Rectangle with a length of 4 units and a width of 6 units, named "Rectangle B". It has an area of 24 square units and a perimeter of 20 units.

The third shape is a Triangle with side lengths of 3, 4, and 5 units, named "Triangle C". Its area and perimeter are calculated based on these side lengths.

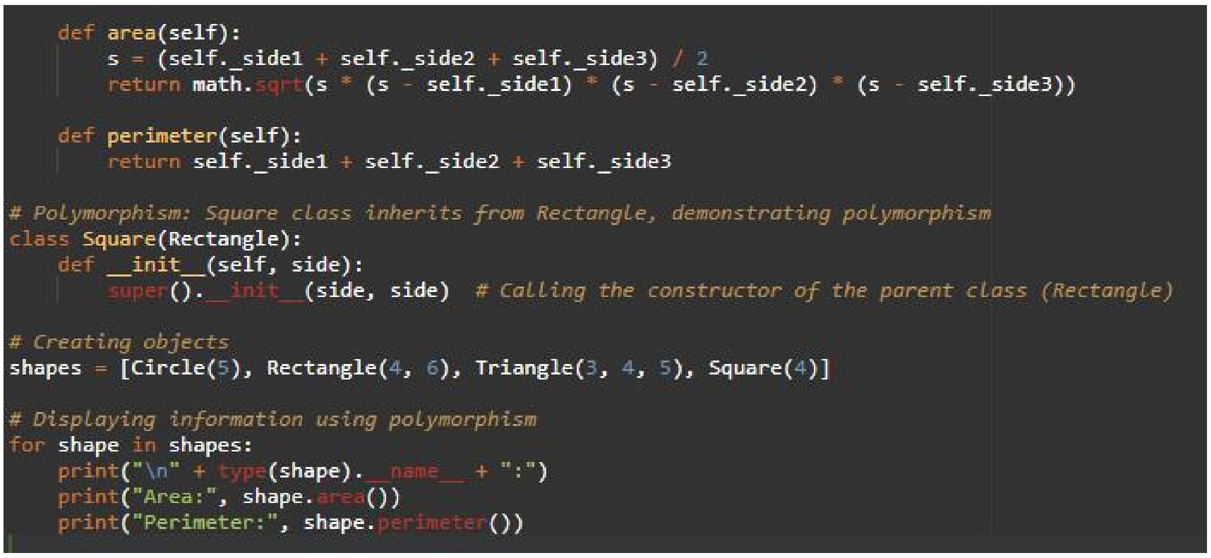
Lastly, the list contains a Square with a side length of 4 units, referred to as "Square D". Its area is 16 square units, and its perimeter is 16 units.

These values represent the characteristics of each shape, including their dimensions, area, and perimeter. Each shape's properties are computed using the appropriate formulas and principles of geometry.

CODE



```python
import math

# Abstraction: Shape class is an abstract class with area() and perimeter() as abstract methods
class Shape:
    def area(self):
        raise NotImplementedError("Subclasses must implement this method")

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

# Encapsulation: Each shape class encapsulates its specific properties and methods

# Inheritance: Circle, Rectangle, Triangle, and Square inherit from the appropriate classes
class Circle(Shape):
    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

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 Triangle(Shape):
    def __init__(self, side1, side2, side3):
        self._side1 = side1
        self._side2 = side2
```



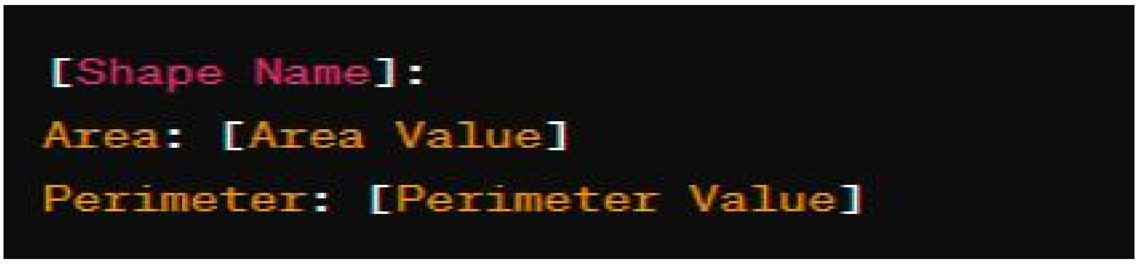
```
    def area(self):
        s = (self._side1 + self._side2 + self._side3) / 2
        return math.sqrt(s * (s - self._side1) * (s - self._side2) * (s - self._side3))

    def perimeter(self):
        return self._side1 + self._side2 + self._side3

# Polymorphism: Square class inherits from Rectangle, demonstrating polymorphism
class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)  # Calling the constructor of the parent class (Rectangle)

# Creating objects
shapes = [Circle(5), Rectangle(4, 6), Triangle(3, 4, 5), Square(4)]

# Displaying information using polymorphism
for shape in shapes:
    print("\n" + type(shape).__name__ + ":")
    print("Area:", shape.area())
    print("Perimeter:", shape.perimeter())
```

USER GUIDE:

Step 1: Running the Program. Save the provided code in a Python file (e.g., shape_calculator.py). Open a terminal or command prompt. Navigate to the directory containing the Python file. Run the program by executing the command: python shape_calculator.py.

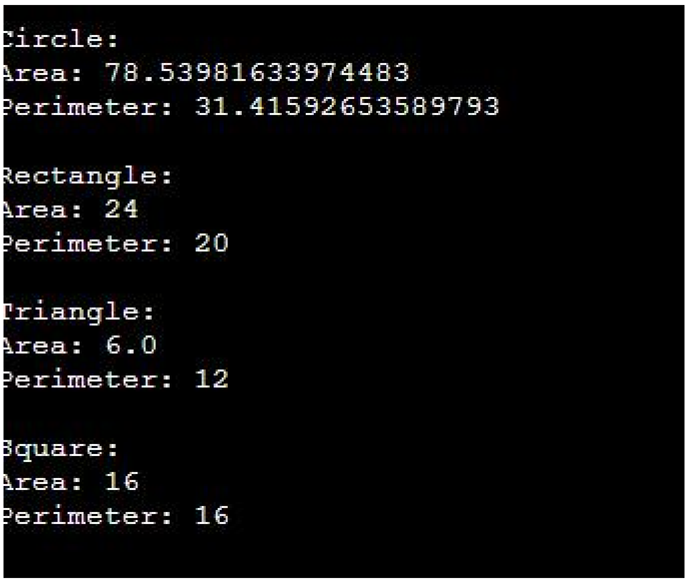
Step 2: Run the code. So that it will display the output of the shape calculator code.

The results are printed in the format:



```
[Shape Name]:
Area: [Area Value]
Perimeter: [Perimeter Value]
```

The output Code:



```
ircle:
rea: 78.53981633974483
erimeter: 31.41592653589793

ectangle:
rea: 24
erimeter: 20

riangle:
rea: 6.0
erimeter: 12

quare:
rea: 16
erimeter: 16
```

REFERENCES

- https://www.onlinegdb.com/
- https://www.w3schools.com/
- https://www.freecodecamp.org/news/four-pillars-of-object-oriented-programming/
- https://chatgpt.com/