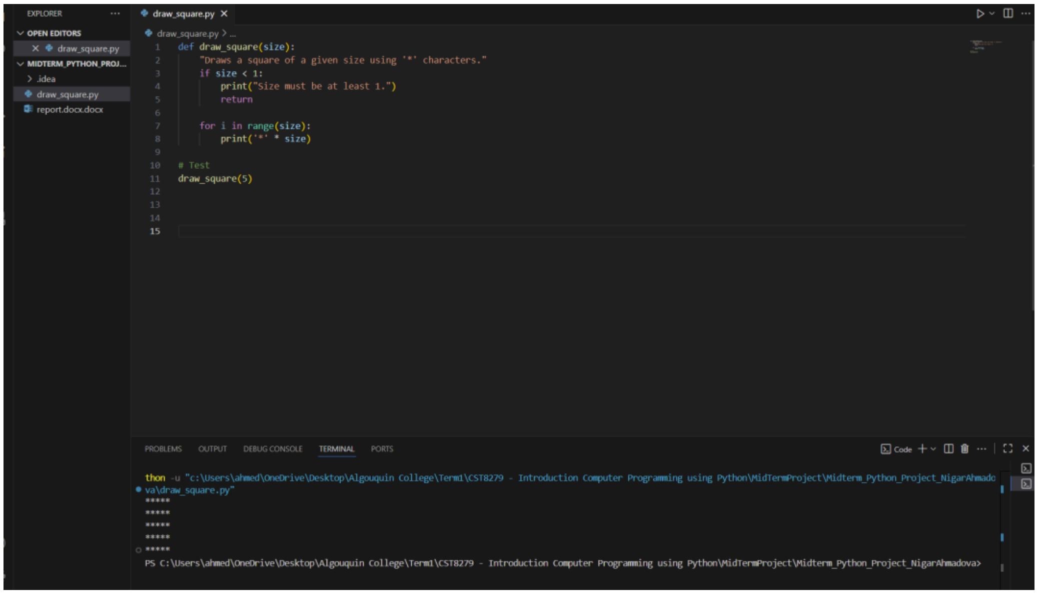
Pattern Drawing in Python using Functions and OOP

Project Description

This project creates a customizable square pattern using both a standalone function and an object-oriented class structure. It includes input validation, loop-based pattern generation, and a modular design that allows easy extension to other shapes. The project demonstrates proficiency in procedural programming, OOP concepts, and clean code practices.

Function-Based Version

The first part of the project defines a function draw_square(size) that prints a square of * characters based on the given size. The function includes a docstring describing its purpose and checks if the provided size is less than 1, displaying an error message if so. If the input is valid, a for loop runs for the specified number of rows, printing a line of * characters equal to the given size.

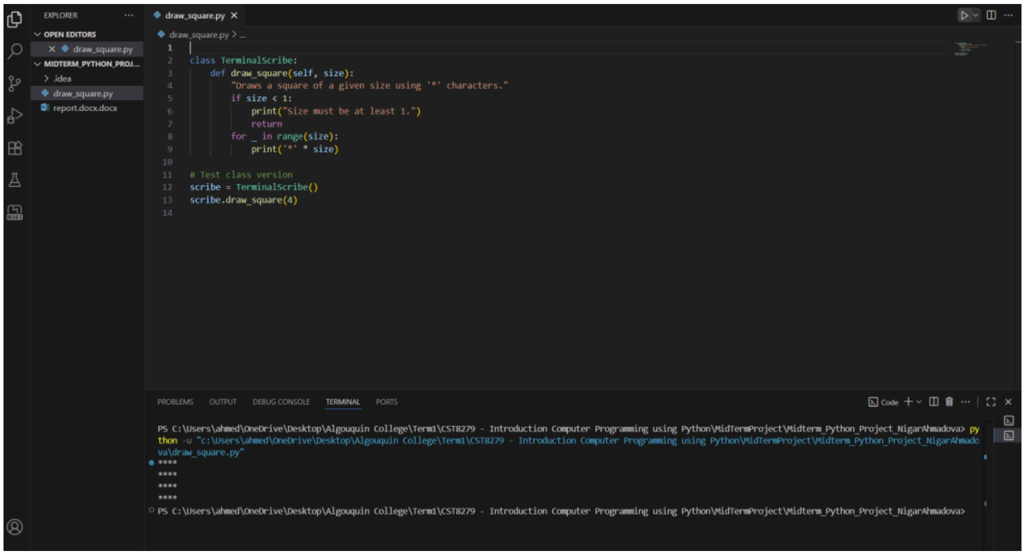


```python
def draw_square(size):
    "Draws a square of a given size using '*' characters."
    if size < 1:
        print("Size must be at least 1.")
        return

    for i in range(size):
        print('*' * size)

# Test
draw_square(5)
```

```
thon -u "c:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmado
va\draw_square.py"
*****
*****
*****
*****
*****
PS C:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmadova>
```

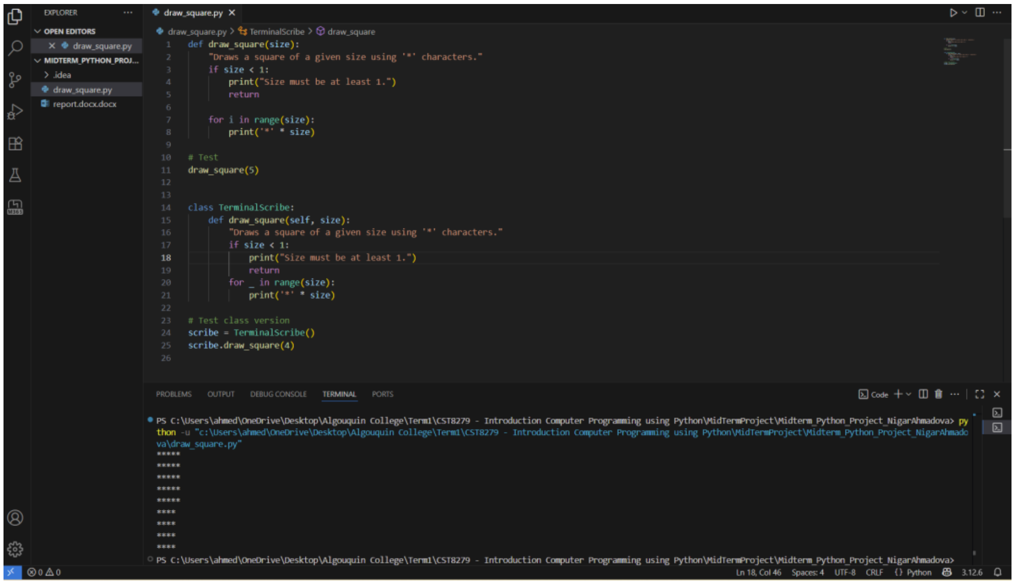
Class-Based Version

The second implementation uses a class called TerminalScribe, which contains the draw_square(self, size) method. Like the function version, it includes a docstring, input validation, and the same loop logic for drawing the square. The method can be reused and extended to draw other shapes by adding more methods to the class.



```python

class TerminalScribe:
    def draw_square(self, size):
        "Draws a square of a given size using '*' characters."
        if size < 1:
            print("Size must be at least 1.")
            return
        for _ in range(size):
            print('*' * size)

# Test class version
scribe = TerminalScribe()
scribe.draw_square(4)
```

```
PS C:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmadova> py
thon -u "c:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmado
va\draw_square.py"
****
****
****
****
PS C:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmadova>
```



```python
def draw_square(size):
    "Draws a square of a given size using '*' characters."
    if size < 1:
        print("Size must be at least 1.")
        return

    for i in range(size):
        print('*' * size)

# Test
draw_square(5)


class TerminalScribe:
    def draw_square(self, size):
        "Draws a square of a given size using '*' characters."
        if size < 1:
            print("Size must be at least 1.")
            return
        for _ in range(size):
            print('*' * size)

# Test class version
scribe = TerminalScribe()
scribe.draw_square(4)
```

```
PS C:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmadova> py
thon -u "c:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmado
va\draw_square.py"
*****
*****
*****
*****
*****
****
****
****
****
PS C:\Users\ahmed\OneDrive\Desktop\Algouquin College\Term1\CST8279 - Introduction Computer Programming using Python\MidTermProject\Midterm_Python_Project_NigarAhmadova>
```

Shape Extensions

The TerminalScribe class could be expanded with additional methods to draw other shapes, such as rectangles, triangles, or hollow squares. These shapes would require adjusting loop logic and conditional statements, making the design scalable and easy to extend.