Comparison of proc & OOP

Certainly! Let's compare procedural programming and object-oriented programming with simple Python examples.

Procedural Programming Example:

```python
# Procedural programming
def calculate_area(length, width):
    return length * width

def calculate_perimeter(length, width):
    return 2 * (length + width)

length = 5
width = 3

area = calculate_area(length, width)
perimeter = calculate_perimeter(length, width)

print(f"Area: {area}")
print(f"Perimeter: {perimeter}")
```

In this procedural example, we have two functions `calculate_area` and `calculate_perimeter` that operate on data (length and width). These functions are separate from the data they work on.

Object-Oriented Programming Example:

```python
# Object-Oriented Programming
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

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

# Create an instance of the Rectangle class
rectangle = Rectangle(5, 3)

area = rectangle.calculate_area()
perimeter = rectangle.calculate_perimeter()

print(f"Area: {area}")
print(f"Perimeter: {perimeter}")
```

In this object-oriented example, we define a `Rectangle` class that encapsulates both data (length and width) and the methods (`calculate_area` and `calculate_perimeter`) to operate on that data. We create an instance of the class and then use its methods to calculate the area and perimeter.

In the procedural example, functions are separate from the data they work on, while in the object-oriented example, data and behavior are encapsulated within the class, promoting a more organized and self-contained structure. The OOP approach is beneficial when dealing with more complex systems with multiple related attributes and behaviors.