Homework Assignment: Object-Oriented Programming in Python

Due date: April 23rd, 2024

Introduction

This homework assignment is designed to deepen your understanding of object-oriented programming concepts in Python. You will explore function call monitoring, abstract base classes, and the implementation of concrete classes derived from these abstract foundations. Additionally, you will apply your skills to model and analyze a strategic card game.

Submission Instructions

- For **Exercise 1** and **Exercise 2**, submit your solutions in separate Jupyter notebooks. Ensure each notebook is well-documented, explaining your approach and analyzing the results where applicable.
- For the **Problem** on the card game:
  - Submit a **Python module** (.py file) containing the implementation of the required classes. This module should be well-commented and organized.
  - Submit a **Jupyter notebook** that demonstrates how your module is used. This notebook should illustrate the strategies you've implemented, including any experiments or analysis you conducted to evaluate the effectiveness of different strategies. Include visualizations if applicable.

1 Exercise 1: Abstract Base Classes and Concrete Implementations

Task Description

Define an abstract class named **Shape**, which declares two abstract methods: `area()` and `perimeter()`. Then, implement three concrete classes that inherit from **Shape**:

1. **Rectangle** - Takes two side lengths as parameters.
2. **Triangle** - Takes three side lengths as parameters.
3. **RegularPolygon** - Takes the radius of the circumscribed circle (the distance from the center to a vertex) and the number of sides as parameters.

Implement the `area()` and `perimeter()` methods according to the geometric definitions of each shape. Submit your solution in a Jupyter notebook.

2 Exercise 2: Monitoring Function Calls

Task Description

Propose several generic approaches to count how many times a given function is used within a Python script. Demonstrate your solutions by applying them to the following code snippet, which evaluates a polynomial using two methods: a basic approach and Horner's method. Your submission should include both code and Markdown cells, discussing the advantages and disadvantages of each method you propose.

```python
def addition(a, b):
    return a + b

def multiplication(a, b):
    return a * b

def evaluate_polynomial_basic(coefficients, x):
    result = 0
    for i, coefficient in enumerate(coefficients):
        term = coefficient
        for _ in range(i):
            term = multiplication(term, x)
        result = addition(result, term)
    return result

def evaluate_polynomial_horner(coefficients, x):
    result = 0
    for coefficient in reversed(coefficients):
        result = addition(multiplication(result, x), coefficient)
    return result

# Example usage
coefficients = [3, 2, 1]  # Represents the polynomial 3x^2 + 2x + 1
x_value = 2
basic_method_value = evaluate_polynomial_basic(coefficients, x_value)
horner_method_value = evaluate_polynomial_horner(coefficients, x_value)
```

3 Problem: Maximizing Expected Score in a Card Game

Problem Description

Consider a card deck with N cards, evenly divided between red and black. In this one-player game, the player draws cards one by one without replacement. Drawing a red card awards $+1$ point, while a black card results in -1 point. The player can stop drawing cards at any time, aiming to maximize the expected score.

Task

Using object-oriented principles, model the card game. Implement:

1. A **Card** class which inherits from `enum.Enum` to deal with the two colors.
2. A **Deck** class for shuffling and drawing cards.
3. A **Player** class to model the stopping strategy and the score – Hint: stopping strategies should depend on the remaining number of black and red cards in the deck.
4. A **Game** class to manage gameplay, allowing the player to draw cards based on their strategy.

Submit a Python module containing your class definitions and a Jupyter notebook demonstrating their use with several strategies.

Bonus points: could you find the optimal strategy? – Info: for a game with 32 cards, the optimal strategy yields an expected score of $\simeq 2.05$.

Conclusion

This assignment combines object-oriented programming with strategic thinking and problem-solving. It challenges you to apply your programming skills to model systems and explore various strategies within those systems. Good luck!