Exercise session 11

Object-oriented programming. Classes, inheritance and polymorphism. Modules and packages.

Advanced Programming - SISSA, UniTS, 2024-2025

Pasquale Claudio Africa

03 Dec 2024

Exercise 1: generators for the solution of ODEs

Solving the differential equation $u' = -\sin(u)$ by applying the explicit Euler method results in the recursion:

$$u_{n+1} = u_n - h\sin\left(u_n\right).$$

Starting from `hints/ex1.py`, given initial value $u_0 = 1$ and a given value of the time step $h = 0.1$, implement a decorator `step_counter` that counts and prints how many time steps have been performed. Perform 10 steps.

Exercise 2: `Polynomial` class (1/3)

Starting from `hints/ex2.py`, you are tasked with implementing a Python class called `Polynomial` that represents polynomials. The class should have the following features:

1. **Constructor**: The class should have a custom constructor that takes variable coefficients as arguments. The coefficients should be provided in increasing order of degree ($a_0 + a_1 x + \cdots + a_n x^n$).
2. **String representation**: Implement the `__repr__` method to provide a string representation of the polynomial. The string should display the polynomial in a human-readable form. For example, for the polynomial with coefficients `[1, 2, 3]`, the string representation should be `"1 + 2x + 3x^2"`.
3. **Addition and multiplication**: Implement the `__add__` and `__mul__` methods to allow addition and multiplication of polynomials. The methods should return a new polynomial.

Exercise 2: `Polynomial` class (2/3)

4. **Class method to create from string**: Implement a `@classmethod` called `from_string` that creates a Polynomial object from a string representation. Assume that the input string will be a polynomial in the form of `"a + bx + ... + cx^(n-1) + dx^n"`.
5. The base class `Polynomial` should be extended by two subclasses:
    - `StandardPolynomialEvaluator`: Implements the standard polynomial evaluation method:

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$$

    - `HornerPolynomialEvaluator`: Implements Horner's rule for polynomial evaluation:

$$P(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \ldots + x \cdot (a_{n-1} + x \cdot a_n) \ldots))$$

Exercise 2: `Polynomial` class (3/3)

6. Implement a `measure_time` decorator, which measures the time taken by a function to execute.
7. Instantiate objects of both `StandardPolynomialEvaluator` and `HornerPolynomialEvaluator` with the same set of coefficients.
8. Apply the `measure_time` decorator to a function that takes a `PolynomialEvaluator` object and evaluates it at a given list of points.
9. Evaluate the polynomial at the same 1000 points using both methods and compare the results. Raise an assertion error if the results do not match.
10. Use the decorated function to evaluate the polynomial using both the standard method and Horner's rule, and observe the logged results and execution times.

Exercise 3: modular data processing package

Refactor the existing data processing code provided in `hints/ex3.py` into a modular package with multiple modules, functions, classes.

1. **Refactoring:** Refactor the code into a modular package `dataprocessor` with the following modules:
   - `__init__.py`: Entry point for the package, import necessary functions, classes, and data. Implement `__all__`.
   - `operations.py`: Contains functions for data processing and analysis.
   - `data_analysis.py`: Introduce a class `DataAnalyzer` that encapsulates data processing and analysis functionalities.
2. **Documentation:** Provide docstrings for functions and classes. Explain the purpose and usage of each function and configuration option.
3. **Test cases:** Create a test `main.py` script to demonstrate the usage of the package.