

# ***# Python Language Gr.1, Project: Dense polynomials based on python lists***

***# Documentation, author: Daniel Szarek***

## **0. Treść Zadania**

Należy zaimplementować klasę obsługującą wielomiany gęste na bazie list w Pythonie. Wymagane operacje to:

- Dodawanie, odejmowanie, mnożenie.
- Obliczanie wartości wielomianu algorytmem Hornera.
- Porównywanie wielomianów ( $=$ ,  $\neq$ ).
- Wyświetlanie oraz odczyt współczynnika przy danej potęgze  $x$  za pomocą operatora  $[]$ .
- Uznanie wielomianów za równe, jeśli ich różnica jest wielomianem zerowym.

Dodatkowe funkcje pomocnicze:

- `is_zero()`: sprawdzanie, czy wielomian jest zerowy.
- `degree()`: zwracanie stopnia wielomianu.

## **1. Wprowadzenie**

Wielomiany są podstawowym narzędziem w matematyce i mają szerokie zastosowanie w naukach ścisłych i inżynierii. Niniejszy dokument opisuje projekt implementujący wielomiany gęste w języku Python na bazie list, umożliwiając wykonywanie operacji arytmetycznych, obliczanie wartości wielomianów, porównywanie ich oraz wiele innych funkcjonalności. Projekt realizuje wymagane zadania i wzbogaca implementację o dodatkowe operatory oraz funkcje.

## **2. Struktura Projektu**

Projekt składa się z następujących plików:

- **poly.py** - Zawiera implementację klasy `Poly` obsługującej wielomiany.
- **test\_poly.py** - Zawiera testy jednostkowe dla klasy `Poly`, wykorzystujące framework `pytest`.
- **documentation.pdf** - Dokumentacja projektu, zawierająca opis teoretyczny, szczegóły implementacji oraz wyniki testów.

## **3. Opis Interfejsu**

**Klasa Poly** - Reprezentacja wielomianu jako listy współczynników, gdzie indeks w liście odpowiada potęgę  $x$ .

**Atrybuty:**

- **coefficients** - Lista współczynników wielomianu, np. dla  $2x^2 - 3x + 5$ :  $[5, -3, 2]$ .
- **\_index** - Chroniona zmienna do iteracji współczynników.

**Metody i Operatory:**

**1. Konstruktor**

- **\_\_init\_\_**: Inicjalizuje wielomian klasy Poly.

**2. Operatory Arytmetyczne**

- **\_\_add\_\_, \_\_radd\_\_, \_\_sub\_\_, \_\_rsub\_\_, \_\_mul\_\_, \_\_rmul\_\_**: Dodawanie, odejmowanie i mnożenie wielomianów.
- **\_\_pow\_\_**: Potęgowanie wielomianu.
- **\_\_neg\_\_, \_\_pos\_\_**: Negacja i dodatnia wartość wielomianu.

**3. Porównywanie**

- **\_\_eq\_\_, \_\_ne\_\_**: Równość i nierówność wielomianów.

**4. Obsługa Nawiasów**

- **\_\_getitem\_\_, \_\_setitem\_\_, \_\_delitem\_\_**: Dostęp, modyfikacja i usunięcie współczynnika dla konkretnej potęgi  $x$ .

**5. Wyświetlanie**

- **\_\_str\_\_**: Czytelna reprezentacja wielomianu, np.  $x^2 - x + 1$ .
- **\_\_repr\_\_**: Techniczna reprezentacja, np. `Poly([1, -1, 1])`.

**6. Funkcje Matematyczne**

- **evaluate\_horner**: Oblicza wartość wielomianu algorytmem Hornera.
- **degree**: Zwraca stopień wielomianu.
- **is\_zero**: Sprawdza, czy wielomian jest zerowy.
- **combine**: Złożenie dwóch wielomianach Poly.
- **differentiate**: Pochodna wielomianu gęstego.
- **integrate**: Całka nieoznaczona wielomianu gęstego.

**7. Iteracja i Kopiowanie**

- **\_\_iter\_\_, \_\_next\_\_**: Iteracja po współczynnikach.
- **copy**: Tworzenie płytkiej kopii wielomianu.

## 8. Inne

- `__len__`: Zwraca długość tablicy wielomianu.
- `__call__`: Wywołanie obiektu jako klasy.

### Przykładowe Użycie:



```
poly.py test_poly.py przyklad.py X
Project > przyklad.py > ...
1 from poly import Poly
2
3 p1 = Poly([1, -2, 3]) # Reprezentuje 3x^2 - 2x + 1
4 p2 = Poly([0, 4])    # Reprezentuje 4x
5
6 print(p1 + p2)       # Wynik: 3x^2 + 2x + 1
7 print(p1.evaluate_horner(2)) # Wynik: 9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\dszar\OneDrive\Pulpit\Studia\I Stopień\Semestr\Python\Project> python -u "c:\Users\dszar\OneDrive\Pulpit\Studia\I Stopień\Semestr\Python\Project\przyklad.py"
3x^2 + 2x + 1
9
PS C:\Users\dszar\OneDrive\Pulpit\Studia\I Stopień\Semestr\Python\Project>
```

## 4. Uwagi na temat Implementacji

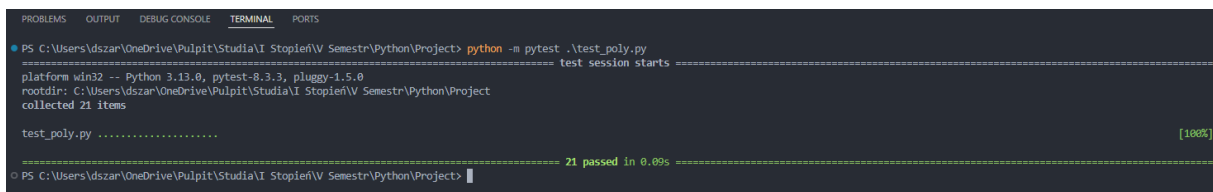
1. **Struktura Danych** - Wielomiany są przechowywane jako listy współczynników, co umożliwia efektywne operacje.
2. **Obsługa Wyjątków** - Metody są zabezpieczone przed błędami typu danych (zgłaszane `TypeError` lub `ValueError`).
3. **Komentarze** – Kod w plikach `.py` zawiera liczne komentarze np. typu docstring, aby informować użytkownika na temat identyfikacji oraz sposobie napisania danej funkcjonalności.
4. **Złożoności Obliczeniowe (Wymaganych w poleceniu zadania funkcjonalności)**
  - Dodawanie (`__add__` / `__radd__`): Złożoność to  $O(\max(n,m))$ , gdzie  $n$  i  $m$  to długości wielomianów. Dzieje się tak, ponieważ sumowanie dwóch wielomianów wymaga iteracji po współczynnikach.
  - Odejmowanie (`__sub__` / `__rsub__`): Złożoność to również  $O(\max(n,m))$ , gdyż działa podobnie jak dodawanie (operacja na współczynnikach).
  - Mnożenie (`__mul__` / `__rmul__`): Złożoność to  $O(n \cdot m)$ , gdzie  $n$  i  $m$  to długości wielomianów. Wynika to z konieczności iteracji przez wszystkie pary współczynników.
  - Obliczanie Hornerem (`evaluate_horner`): Złożoność to  $O(n)$ , gdzie  $n$  to stopień wielomianu. Dzieje się tak, ponieważ algorytm Hornera przechodzi przez wszystkie współczynniki.
  - Porównywanie (`__eq__`, `__ne__`): Złożoność to  $O(\max(n,m))$ , ponieważ porównanie dwóch wielomianów sprowadza się do odjęcia ich współczynników i sprawdzenia, czy wynik jest wielomianem zerowym.

- Wyświetlanie (`__str__`): Złożoność to  $O(n)$ , gdzie  $n$  to stopień wielomianu. Konwersja do czytelnego formatu wymaga przejścia przez wszystkie współczynniki.
- Użycie nawiasów (`__getitem__`): Złożoność to  $O(1)$ , ponieważ dostęp do elementu listy Python ma złożoność stałą.
- Stopień wielomianu (`degree`): Złożoność to  $O(1)$ , ponieważ operacja ta zwraca długość listy współczynników minus 1.
- Sprawdzanie, czy wielomian jest zerowy (`is_zero`): Złożoność to  $O(1)$ , ponieważ sprawdzenie, czy lista współczynników jest pusta, wymaga stałego czasu.

## 5. Wyniki Testów

Testy jednostkowe, zaimplementowane w `test_poly.py`, obejmują wszystkie wymagane operacje oraz funkcjonalności dodatkowe. Framework `pytest` automatyzuje proces testowania, zapewniając czytelne raporty. Zainstalowanie i odpowiednie skonfigurowanie modułu `pytest` jest niezbędne do prawidłowego uruchomienia testów.

Przykład wyniku testów:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\dszar\OneDrive\Pulpit\Studia\I Stopień\W Semestr\Python\Project> python -m pytest -v test_poly.py
===== test session starts =====
platform win32 -- Python 3.13.0, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\dszar\OneDrive\Pulpit\Studia\I Stopień\W Semestr\Python\Project
collected 21 items

test_poly.py ..... [100%]

===== 21 passed in 0.09s =====
PS C:\Users\dszar\OneDrive\Pulpit\Studia\I Stopień\W Semestr\Python\Project>

```

## 6. Podsumowanie

Projekt spełnia wszystkie wymagania zadania. Dodatkowo implementacja została wzbogacona o funkcjonalności takie jak potęgowanie wielomianów, całkowanie nieoznaczone czy składanie wielomianów.

## 7. Literatura

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Introduction to Algorithms*.
2. Wikipedia - [Horner's Method](#).
3. Dokumentacja Python: [Oficjalna strona](#).