Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 7 – Paradygmat funkcyjny (cz. 2): Rekurencja i leniwa ewaluacja

Kierunek: Informatyka, semestr 5
Prowadzący: dr inż. Krzysztof Malczewski

1. Cel i zakres zajęć

Celem zajęć jest:

- poznanie **rekurencji** jako podstawowego narzędzia paradygmatu funkcyjnego,
- zrozumienie różnicy między podejściem iteracyjnym a rekurencyjnym,
- nauczenie się rozwiązywania klasycznych problemów rekurencyjnie,
- wprowadzenie pojęcia **leniwej ewaluacji** i generatorów,
- zastosowanie tych technik w praktycznych zadaniach.

Zakres tematyczny:

- definicja i zasada działania rekurencji,
- przypadek bazowy i rekurencyjny,
- porównanie z pętlami,
- rekurencja ogonowa (tail recursion) – pojęcie, ograniczenia,
- leniwa ewaluacja i generatory w Pythonie,
- infinite sequences, map/filter na generatorach.

---

2. Wprowadzenie teoretyczne

2.1 Rekurencja

Rekurencja to technika, w której funkcja wywołuje samą siebie do momentu osiągnięcia **przypadku bazowego**.

Przykład: silnia

$$n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1)! & n > 0 \end{cases} \quad n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1)! & n = 0 \end{cases} \quad n > 0$$

```
def silnia(n):
    if n == 0:
        return 1
    else:
        return n * silnia(n-1)
```

Rekurencja jest fundamentem wielu języków funkcyjnych (np. Haskell), gdzie często **zastępuje pętle**.

2.2 Struktura funkcji rekurencyjnej

Każda funkcja rekurencyjna powinna mieć:

- **przypadek bazowy** – warunek zatrzymania rekurencji,
- **krok rekurencyjny** – wywołanie samej siebie z mniejszym problemem,

- **brak efektów ubocznych** (w stylu funkcyjnym).

2.3 Rekurencja vs iteracja

- Rekurencja: często bardziej deklaratywna i czytelna.
- Iteracja: bardziej wydajna w niektórych językach, ale mniej elegancka.
- W językach funkcyjnych często stosuje się optymalizacje rekurencji ogonowej, aby uniknąć przepełnienia stosu.

2.4 Leniwa ewaluacja

Leniwa ewaluacja (lazy evaluation) polega na **odraczaniu obliczeń** do momentu, gdy wynik jest rzeczywiście potrzebny.
Pozwala to na:

- pracę z potencjalnie nieskończonymi strukturami,
- zwiększenie efektywności (nie obliczamy nieużytych danych),
- pisanie eleganckich, deklaratywnych programów.

W Pythonie leniwość można osiągnąć m.in. przez **generatory** i wyrażenia generatorowe.

---

3. Tutorial 1 – Rekurencja w Pythonie

3.1 Klasyczne przykłady

Silnia

```
def silnia(n):
    if n == 0:
        return 1
    return n * silnia(n-1)

print(silnia(5))  # 120
```

Ciąg Fibonacciego

```
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

print([fib(i) for i in range(10)])  # [0,1,1,2,3,5,8,13,21,34]
```

3.2 Rekurencja ogonowa (tail recursion)

Przykład silni w wersji ogonowej:

```
def silnia_tail(n, acc=1):
    if n == 0:
        return acc
    return silnia_tail(n-1, acc*n)
```

👉 Python nie optymalizuje rekurencji ogonowej, ale pojęcie to jest kluczowe w językach funkcyjnych takich jak Haskell, Scheme czy OCaml.

---

4. Tutorial 2 – Leniwa ewaluacja i generatory

4.1 Prosty generator liczb naturalnych

```
def liczby_naturalne():
    n = 0
    while True:
        yield n
        n += 1

gen = liczby_naturalne()
for i in range(10):
    print(next(gen))
```

4.2 Generatory + map/filter

```
def kwadraty(seq):
    for x in seq:
        yield x*x

def parzyste(seq):
    for x in seq:
        if x % 2 == 0:
            yield x

g = liczby_naturalne()
wynik = kwadraty(parzyste(g))
for i in range(5):
    print(next(wynik))  # 0, 4, 16, 36, 64
```

👉 Nie generujemy całych list – obliczenia zachodzą „na żądanie”.

---

5. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Napisz funkcję rekurencyjną `suma_lista(lista)`, która zwraca sumę elementów listy **bez użycia pętli**.

Zadanie 2 (obowiązkowe)

Zaimplementuj funkcję rekurencyjną `fib_tail(n)` w stylu ogonowym. Porównaj jej działanie z `fib(n)` dla dużych n.

Zadanie 3 (obowiązkowe)

Napisz generator `liczby_pierwsze()`, który generuje kolejne liczby pierwsze. Użyj go do wypisania 20 pierwszych liczb pierwszych.

Zadanie 4 (dodatkowe)

Napisz generator nieskończonego ciągu Fibonacciego i zastosuj na nim `map` i `filter`, aby wypisać pierwsze 10 kwadratów liczb parzystych z tego ciągu.

---

6. Pytania kontrolne

1. Na czym polega rekurencja?
2. Co to jest przypadek bazowy?
3. Na czym polega rekurencja ogonowa i dlaczego jest ważna?
4. Co to jest leniwa ewaluacja?
5. Czym różnią się listy od generatorów w Pythonie?
6. Jak połączyć rekurencję z leniwym przetwarzaniem?

---

7. Checklisty

Rekurencja

- Funkcje mają przypadek bazowy.
- Nie użyto pętli w zadaniach rekurencyjnych.
- Kod jest czysty i bez efektów ubocznych.

Leniwa ewaluacja

- Użyto `yield` i generatorów.
- Nie tworzono niepotrzebnie dużych list.
- Obliczenia wykonywane są na żądanie.

---

8. Wzór sprawozdania

Dodatkowo:

- Porównanie rekurencji i iteracji dla wybranego problemu.
- Diagram drzewa wywołań dla rekurencji (np. dla fib(5)).
- Zrzuty ekranu z działania generatorów.

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć rekurencję i jej rolę w programowaniu funkcyjnym,
- umieć pisać funkcje rekurencyjne, w tym w stylu ogonowym,
- znać ideę leniwej ewaluacji i umieć korzystać z generatorów,
- rozumieć różnicę między podejściem eager a lazy,
- potrafić zastosować rekurencję i leniwe przetwarzanie w praktycznych zadaniach.