Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 6 – Paradygmat funkcyjny (cz. 1): podstawy

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

1. Cel i zakres zajęć

Celem zajęć jest:

- wprowadzenie podstaw paradygmatu funkcyjnego,
- zrozumienie różnic między podejściem imperatywnym a funkcyjnym,
- poznanie pojęć: **funkcja czysta**, **funkcje wyższego rzędu**, **map**, **filter**, **reduce**,
- napisanie prostych programów funkcyjnych w Pythonie,
- przygotowanie gruntu pod bardziej zaawansowane techniki (rekurencja, leniwa ewaluacja – jednostka 7).

Zakres tematyczny:

- definicja paradygmatu funkcyjnego,
- czyste funkcje, unikanie efektów ubocznych,
- przetwarzanie list funkcjami map/filter/reduce,
- lambda i funkcje anonimowe,
- porównanie z wersją imperatywną.

---

2. Wprowadzenie teoretyczne

2.1 Paradygmat funkcyjny – definicja

W programowaniu funkcyjnym **podstawowym elementem są funkcje matematyczne**, które:

- nie zmieniają stanu programu (brak efektów ubocznych),
- dla tych samych danych wejściowych **zawsze zwracają ten sam wynik**,
- mogą być przekazywane jako argumenty i zwracane jako wyniki.

Najważniejsze cechy:

- **Deklaratywność** – opisujemy „co chcemy osiągnąć”, a nie „jak krok po kroku”.
- **Brak mutacji** – nie zmieniamy zmiennych ani struktur danych.
- **Funkcje wyższego rzędu** – przyjmują funkcje jako argumenty lub je zwracają.
- **Czyste funkcje** – deterministyczne, bez efektów ubocznych.
- **Rekurencja** – często zamiast pętli.

2.2 Czyste funkcje

Czysta funkcja:

- nie korzysta z danych zewnętrznych (poza parametrami),
- nie modyfikuje zmiennych globalnych, plików, stanu systemu,

- zawsze daje ten sam wynik dla tych samych danych wejściowych.

Przykład czystej funkcji:

```
def dodaj(a, b):
    return a + b
```

Przykład nieczystej funkcji:

```
x = 10
def dodaj_do_x(a):
    return a + x    # zależy od zmiennej globalnej
```

2.3 Funkcje wyższego rzędu

Funkcje mogą przyjmować inne funkcje jako argumenty, np.:

```
def zastosuj_operacje(f, x, y):
    return f(x, y)

def dodaj(a, b): return a+b
def mnoz(a, b): return a*b

print(zastosuj_operacje(dodaj, 2, 3)) # 5
print(zastosuj_operacje(mnoz, 2, 3)) # 6
```

---

3. Tutorial 1 – map / filter / reduce (Python)

3.1 map – przekształcanie list

```
liczby = [1, 2, 3, 4, 5]
kwadraty = list(map(lambda x: x*x, liczby))
print(kwadraty)  # [1, 4, 9, 16, 25]
```

3.2 filter – filtrowanie danych

```
parzyste = list(filter(lambda x: x % 2 == 0, liczby))
print(parzyste)  # [2, 4]
```

3.3 reduce – redukcja listy do jednej wartości

```
from functools import reduce

suma = reduce(lambda a,b: a+b, liczby)
print(suma)  # 15
```

3.4 Porównanie z podejściem imperatywnym

Imperatywnie:

```
suma = 0
for x in liczby:
    suma += x
```

Funkcyjnie:

```
from functools import reduce
suma = reduce(lambda a,b: a+b, liczby)
```

✅ Funkcyjnie kod jest **krótszy**, **deklaratywny**, nie modyfikuje zmiennych.

---

4. Tutorial 2 – Funkcje wyższego rzędu

4.1 Tworzenie własnych funkcji wyższego rzędu

```
def przetworz_liste(lista, operacja):
    return [operacja(x) for x in lista]

def potroj(x): return 3*x

wynik = przetworz_liste([1,2,3], potroj)
print(wynik)  # [3, 6, 9]
```

4.2 Zagnieżdżanie funkcji

```
def stworz_mnoznik(k):
    def mnoz(x):
        return x * k
    return mnoz

mnoz5 = stworz_mnoznik(5)
print(mnoz5(10))  # 50
```

---

5. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Napisz funkcję czystą `przetworz_liczby(lista)`, która:

- zwraca nową listę zawierającą kwadraty liczb parzystych z `lista`,
- używa wyłącznie funkcji `map` i `filter` (bez pętli i zmiennych pomocniczych).

Zadanie 2 (obowiązkowe)

Napisz funkcję `agreguj(lista, funkcja_agregujaca)` będącą ogólną wersją `reduce`. Następnie użyj jej do:

- obliczenia sumy,
- obliczenia iloczynu,
- znalezienia maksimum.

Zadanie 3 (dodatkowe)

Napisz funkcję wyższego rzędu `stworz_filtr(min_val, max_val)`, która zwróci funkcję filtrującą liczby spoza danego zakresu. Użyj jej z `filter` do przetworzenia dużej listy losowych liczb.

---

6. Pytania kontrolne

1. Co to jest czysta funkcja?
2. Jak funkcje funkcyjne różnią się od imperatywnych?
3. Co oznacza „funkcja wyższego rzędu”?
4. Czym różnią się `map`, `filter` i `reduce`?
5. Jakie zalety ma styl funkcyjny?

---

7. Checklisty

Funkcyjny

- Brak zmiennych globalnych.
- Brak mutacji list i obiektów.
- Użyto funkcji wyższego rzędu.
- Kod jest krótki i deklaratywny.
- Wyniki funkcji zależą tylko od parametrów.

---

8. Wzór sprawozdania

Dodatkowo:

- Porównanie rozwiązań imperatywnych i funkcyjnych (np. tabelka lub komentarz).

- Analiza zalet i wad stylu funkcyjnego w Twoim kodzie.
- Wskazanie fragmentów kodu, które są czysto funkcyjne.

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć podstawy paradygmatu funkcyjnego,
- umieć pisać i stosować czyste funkcje,
- znać funkcje wyższego rzędu i podstawowe operatory map/filter/reduce,
- potrafić porównać styl funkcyjny i imperatywny na prostych przykładach.