Tytuł ćwiczenia: Jednostka 6 - Paradygmat funkcyjny
Autor: Joanna Dagil
Grupa: TCH-1
Data: 18.11.2025

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawami paradygmatu funkcyjnego i porównanie go z programowaniem imperatywnym.

2 Przebieg ćwiczenia

W tym ćwiczeniu posłużę się językiem Python w edytorze Visual Studio Code. W celu wykonania zadań zapoznaję się z tutorialami, a następnie przechodzę do stworzenia pliku:

lab06.py

```python
from functools import reduce
import random

def przetworz_liczby(lista):
    return list(map(lambda x: x*x, (list(filter(lambda x: x % 2 == 0, lista)))))

def suma(x, y): return x + y
def iloczyn(x, y): return x * y
def maksimum(x, y): return x if x > y else y

def agreguj(lista, funkcja_agregujaca):
    return reduce(funkcja_agregujaca, lista)

def stworz_filtr(min, max):
    def filtr(x):
        return min <= x and x <= max
    return filtr

if __name__ == "__main__":
    lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print("Oryginalne liczby:    ", lista)
    print("-------------- ZADANIE 1 --------------")
    print("Przetworzone liczby:  ", przetworz_liczby(lista))
    print("-------------- ZADANIE 2 --------------")
    print("Agregacja sumowaniem: ", agreguj(lista, suma))
    print("Agregacja iloczynem:  ", agreguj(lista, iloczyn))
    print("Agregacja maksowaniem:", agreguj(lista, maksimum))
    print("-------------- ZADANIE 3 --------------")
    duza_lista = [random.randint(1,100) for _ in range(10000)]
    print("Przed filtrowaniem: ", len(duza_lista))
    print("Po filtrowaniu:     ", len(list(filter(stworz_filtr(30,70), duza_lista))))
```

Program uruchamiam poleceniem

```
python lab06.py
```

3 Wyniki działania

Dla przykładowych danych

```
Oryginalne liczby:     [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
-------------- ZADANIE 1 --------------
Przetworzone liczby:   [4, 16, 36, 64, 100]
-------------- ZADANIE 2 --------------
Agregacja sumowaniem:  55
Agregacja iloczynem:   3628800
Agregacja maksowaniem: 10
-------------- ZADANIE 3 --------------
```

```
Przed filtrowaniem:  10000
Po filtrowaniu:      4052
```

4 Wnioski

4.1 Porównanie rozwiązań imperatywnych i funkcyjnych

Kryterium	Imperatywne	Funkcyjne
Paradygmat	Skupia się na sekwencji instrukcji zmieniających stan programu	Skupia się na definiowaniu funkcji i przepływie danych między nimi
Stan programu	Zmienny – dane są modyfikowane w trakcie działania	Niezmienny – dane są niemutowalne, funkcje nie zmieniają stanu
Podejście do obliczeń	Wykonuje polecenia krok po kroku	Deklaruje, co ma być obliczone, bez opisywania kolejnych kroków
Efekty uboczne	Dozwolone i często wykorzystywane	Ograniczane lub eliminowane
Powtarzanie	Implementowane przy użyciu pętli	Implementowane przy użyciu rekurencji
Zmienne, listy, obiekty	Wykorzystywane	Jedyne jako parametry funkcji

4.2 Zalety i wady stylu funkcyjnego

Zalety:

- Kod jest bardziej zwięzły i deklaratywny – opisuje *co* ma być wykonane, a nie *jak*.
- Łatwo tworzyć czyste funkcje, które nie modyfikują danych wejściowych, co ułatwia testowanie.
- Zastosowanie `map`, `filter` i `reduce` pozwala pisać kod modularny
- Funkcje można łatwo przekazywać jako argumenty, co zwiększa elastyczność programu.

Wady:

- Styl funkcyjny może być mniej czytelny dla osób przyzwyczajonych do programowania imperatywnego.
- Trudniejsze może być śledzenie przepływu danych i debugowanie złożonych funkcji anonimowych.
- Niektóre operacje (np. zagnieżdżone `lambda`) mogą być mniej wydajne lub trudniejsze do optymalizacji.

Czysto funkcyjne są wszystkie funkcje poza mainem.