Tytuł ćwiczenia: Jednostka 2 - Paradygmat proceduralny i strukturalny
Autor: Joanna Dagil
Grupa: TCH-1
Data: 14.10.2025

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadami paradygmatów proceduralnych i strukturalnych. A następnie refaktoryzacja programu imperatywnego z Jednostki 1 do wersji proceduralnej.

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 plików:

zad1.py (dla zadania 1 i 2)

```python
def wczytaj_dane():
    # pobranie ilości liczb przez instrukcję wejścia i przypisanie jej do zmiennej N
    N = int(input("Podaj liczbę elementów: "))
    # zainicjowanie pustej listy numbers
    numbers = []
    # iteracja instrukcją sterującą pętlą for i wczytanie N liczb do listy numbers przez instrukcję wejścia
    for i in range(N):
        number = float(input(f"Podaj liczbę {i+1}: "))
        numbers.append(number)
    # zwrócenie listy numbers
    return numbers

def oblicz_sume(numbers):
    # zwrócenie sumy elementów listy dane
    return sum(numbers)

def oblicz_srednia(numbers):
    # zwrócenie średniej elementów listy dane lub 0 jeśli lista jest pusta
    return oblicz_sume(numbers)/len(numbers) if numbers else 0

def znajdz_min_max(numbers):
    # zwrócenie krotki (min, max) elementów listy dane lub (None, None) jeśli lista jest pusta
    return (min(numbers), max(numbers)) if numbers else (None, None)

def znajdz_mediane(numbers):
    # posortowanie listy numbers
    sorted_numbers = bubble_sort(numbers)
    n = len(sorted_numbers)
    if n == 0:
        return None
    mid = n // 2
    # zwrócenie mediany elementów listy numbers
    if n % 2 == 0:
        return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2
    else:
        return sorted_numbers[mid]

def bubble_sort(numbers):
    # iteracja instrukcją sterującą pętlą for
    for i in range(len(numbers)-1):
        # iteracja zagnieżdzoną instrukcją sterującą pętlą for
        for j in range(len(numbers)-1-i):
            # porównanie dwóch sąsiednich elementów listy przy pomocy instrukcji warunkowej if
            if numbers[j] > numbers[j+1]:
                # zamiana miejscami dwóch sąsiednich elementów listy za pomocą przypisania wielokrotnego
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
```

```
    # zwrócenie posortowanej listy numbers
    return numbers

def drukuj_wyniki(dane):
    # wypisanie posortowanej listy numbers instrukcją wyjścia
    print("Posortowane dane:", dane)

def main():
    # wczytanie danych
    dane = wczytaj_dane()
    if not dane:
        print("Brak danych.")
        return
    # obliczene sumy, średniej, min i max, mediany
    srednia = oblicz_srednia(dane)
    min_val, max_val = znajdz_min_max(dane)
    mediana = znajdz_mediane(dane)
    # sortowanie danych
    posortowane = bubble_sort(dane)
    # drukowanie posortowanych danych
    drukuj_wyniki(posortowane)
    # drukowanie reszty wyników
    porównanie = '>' if mediana > srednia else '<' if mediana < srednia else '='
    print(f"Mediana vs średnia: {mediana} {porównanie} {srednia}")
    print(f"Średnia: {srednia}, Min: {min_val}, Max: {max_val}")

# wywołanie funkcji main
if __name__ == "__main__":
    main()
```

zad3.py (dla zadania 3)

```
def wczytaj_dane():
    N = int(input("Podaj liczbę ocen: "))
    oceny = []
    for i in range(N):
        ocena = int(input(f"Podaj ocenę {i+1} (0-100): "))
        oceny.append(ocena)
    return oceny

def oblicz_sume(numbers):
    # zwrócenie sumy elementów listy numbers
    return sum(numbers)

def oblicz_srednia(numbers):
    # zwrócenie średniej elementów listy dane lub 0 jeśli lista jest pusta
    return oblicz_sume(numbers)/len(numbers) if numbers else 0

def bubble_sort_malejący(numbers):
    # iteracja instrukcją sterującą pętlą for
    for i in range(len(numbers)-1):
        # iteracja zagnieżdzoną instrukcją sterującą pętlą for
        for j in range(len(numbers)-1-i):
            # porównanie dwóch sąsiednich elementów listy przy pomocy instrukcji warunkowej if
            if numbers[j] < numbers[j+1]:
                # zamiana miejscami dwóch sąsiednich elementów listy za pomocą przypisania wielokrotnego
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
    # zwrócenie posortowanej listy numbers
    return numbers

def drukuj_wyniki(oceny, srednia):
    # wypisanie ocen powyżej średniej malejaco i ich liczby
    print("Oceny powyżej średniej posortowane malejąco:", end=' ')
    liczba = 0
    for ocena in oceny:
        if ocena > srednia:
            print(ocena, end=' ')
            liczba += 1

```

```
    print("")
    print("Liczba ocen powyżej średniej:", liczba)

def main():
    # wczytanie ocen
    oceny = wczytaj_dane()
    if not oceny:
        print("Brak danych.")
        return

    # obliczenie średniej ocen
    srednia = oblicz_srednia(oceny)

    # posortowanie listy ocen malejąco
    oceny_sorted = bubble_sort_malejący(oceny)

    # drukowanie wyników
    drukuj_wyniki(oceny_sorted, srednia)

# wywołanie funkcji main
if __name__ == "__main__":
    main()
```

Pliki uruchamiam poleceniem

```
python zad{N}.py
```

3 Wyniki działania

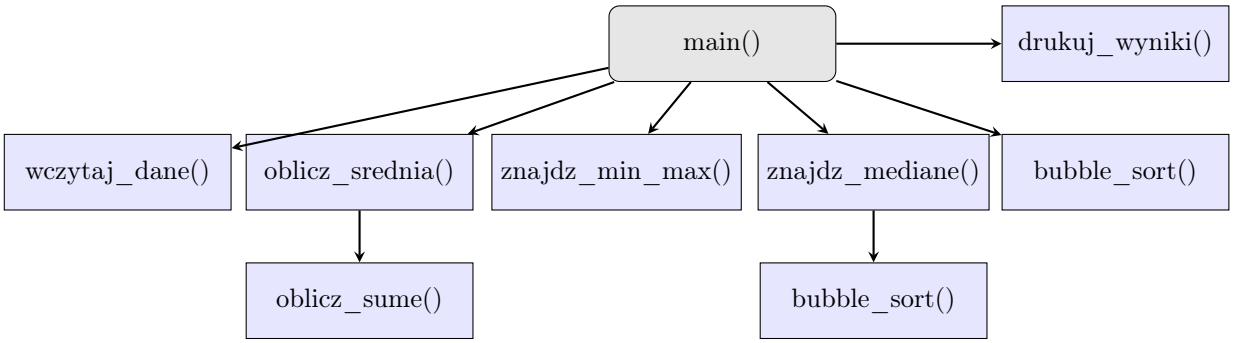
Dla Zadania 1 i 2:

```
PS C:\Users\asiad\Desktop\paradygmaty\lab02> python zad1.py
Podaj liczbę elementów: 4
Podaj liczbę 1: 4.6
Podaj liczbę 2: 10.6
Podaj liczbę 3: 5
Podaj liczbę 4: 7.3
Posortowane dane: [4.6, 5.0, 7.3, 10.6]
Mediana vs średnia: 6.15 < 6.875
Średnia: 6.875, Min: 4.6, Max: 10.6
```

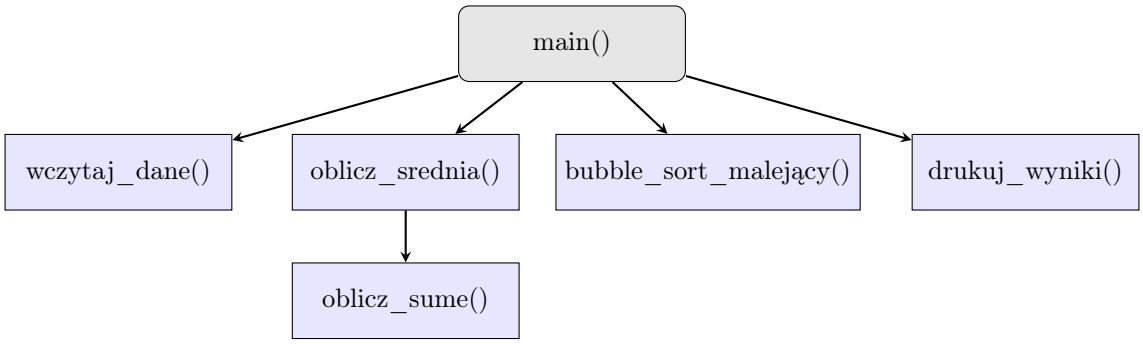
Dla Zadania 3:

```
PS C:\Users\asiad\Desktop\paradygmaty\lab02> python zad3.py
Podaj liczbę ocen: 3
Podaj ocenę 1 (0-100): 40
Podaj ocenę 2 (0-100): 30
Podaj ocenę 3 (0-100): 10
Oceny powyżej średniej posortowane malejąco: 40 30
Liczba ocen powyżej średniej: 2
PS C:\Users\asiad\Desktop\paradygmaty\lab02> python zad3.py
Podaj liczbę ocen: 4
Podaj ocenę 1 (0-100): 50
Podaj ocenę 2 (0-100): 40
Podaj ocenę 3 (0-100): 30
Podaj ocenę 4 (0-100): 40
Oceny powyżej średniej posortowane malejąco: 50
Liczba ocen powyżej średniej: 1
PS C:\Users\asiad\Desktop\paradygmaty\lab02> python zad3.py
Podaj liczbę ocen: 0
Brak danych.
```

4 Schemat wywołań funkcji



Rysunek 1: Schemat blokowy wywołań funkcji w programie proceduralnym zad1.py.



Rysunek 2: Schemat blokowy wywołań funkcji w programie strukturalnym zad3.py.

5 Porównanie wersji imperatywnej i proceduralnej

Wersja imperatywna programu opiera się na sekwencyjnym wykonywaniu instrukcji – program składa się głównie z ciągu poleceń modyfikujących zmienne i sterujących przepływem wykonania (np. poprzez pętle i instrukcje warunkowe). W takiej formie kod często ma charakter liniowy, co utrudnia jego czytelność, ponowne wykorzystanie fragmentów oraz utrzymanie w przypadku rozbudowy programu. Logika działania jest bezpośrednio „zaszyta" w głównym bloku programu.

Natomiast wersja strukturalna (proceduralna) wprowadza podział programu na mniejsze, logiczne jednostki – funkcje (procedury). Każda z nich realizuje określone zadanie, a główna funkcja `main()` koordynuje ich wywołania. Dzięki temu kod staje się bardziej przejrzysty, modularny i łatwiejszy w testowaniu oraz modyfikacji. W przypadku błędów lub potrzeby zmian wystarczy poprawić jedną funkcję, bez konieczności ingerencji w cały program.

Pod względem działania obie wersje realizują ten sam cel, jednak strukturalna forma pozwala na lepszą organizację kodu i w teorii eliminuje powtarzanie fragmentów.

6 Wnioski

Paradygmat proceduralny jest rozszerzeniem paradygmatu imperatywnego. Zwiększa czytelność i ułatwia podział zadań. Pozwala także na łatwiejszy przyszły rozwój oprogramowania - można edytować tylko poszczególną funkcję, bez ingerencji w cały kod.

Jednocześnie nie jest to wziąć paradygmat wprowadzający niepotrzebnie złożoną strukturę, nawet w najprostszych programach i jest wygodny do małych i średnich projektów.