Tytuł ćwiczenia: Jednostka 3 - Paradygmat modularny
Autor: Joanna Dagil
Grupa: TCH-1
Data: 21.10.2025

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadami paradygmatu modularnego. A następnie refaktoryzacja programu proceduralnego z Jednostki 2 do wersji modularnej i dodanie dodatkowych funkcjonalności.

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:

main.py

```
"""
Główny moduł programu do analizy listy liczb.
"""
import utils
import sorting
import stats
import analityka
import raport

def main():
    # wczytanie danych
    dane = utils.wczytaj_dane()
    if not dane:
        print("Brak danych.")
        return
    # obliczene sumy, średniej, min i max, mediany
    srednia = stats.oblicz_srednia(dane)
    min_val, max_val = stats.znajdz_min_max(dane)
    mediana = analityka.mediana(dane)
    odchylenie = analityka.odchylenie_standardowe(dane)
    # sortowanie danych
    posortowane = sorting.sort(dane)
    # drukowanie posortowanych danych
    utils.wypisz_dane(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}")
    print(f"Mediana: {mediana}, Odchylenie standardowe: {odchylenie}")
    # zapisanie raportu do pliku
    raport.zapisz_raport(srednia, min_val, max_val, mediana, odchylenie, porównanie)

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

stats.py

```
"""
Moduł udostępnia funkcje do najbardziej podstawowej analizy statystycznej list liczb.

Publiczne funkcje:
- oblicz_srednia(numbers): zwraca średnią listy liczb.
- znajdz_min_max(numbers): zwraca krotkę (min, max) listy liczb.
"""
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)
```

sorting.py

```
"""
Moduł udostępnia funkcje do sortowania listy liczb.

Publiczne funkcje:
- sort(numbers): zwraca posortowoną listy liczb.
"""
def sort(numbers):
    # iteracja instrukcją sterującą pętlą for
    for i in range(len(numbers)-1):
        # iteracja zagnieżdżoną 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
```

utils.py

```
"""
Moduł udostępnia funkcje do wczytywania i wypisywania danych listowych.

Publiczne funkcje:
- wczytaj_dane(): wczytuje listę liczb od użytkownika.
- wypisz_dane(dane): wypisuje listę liczb na ekran.
"""
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 wypisz_dane(dane):
    # wypisanie posortowanej listy numbers instrukcją wyjścia
    print("Posortowane dane:", dane)
```

analityka.py

```
"""
Moduł udostępnia funkcje do podstawowej analizy statystycznej list liczb.

Publiczne funkcje:
- mediana(numbers): zwraca medianę listy liczb.
- odchylenie_standardowe(numbers): zwraca odchylenie standardowe listy liczb.
"""

import sorting
import stats

def mediana(numbers):
    # posortowanie listy numbers
```

```
    sorted_numbers = sorting.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 odchylenie_standardowe(numbers):
    # obliczenie średniej
    srednia = stats.oblicz_srednia(numbers)
    # obliczenie wariancji
    wariancja = sum((x - srednia) ** 2 for x in numbers) / len(numbers) if numbers
    else 0
    # zwrócenie odchylenia standardowego
    return wariancja ** 0.5
```

raport.py

```
"""
Moduł udostępnia funkcje do stworzenia raportu z analizy listy liczb.

Publiczne funkcje:
- zapisz_raport(srednia, min_val, max_val, mediana, odchylenie, porownanie, filename="
    raport.txt"): zapisuje raport do pliku tekstowego.
"""
def zapisz_raport(srednia, min_val, max_val, mediana, odchylenie, porownanie, filename
    ="raport.txt"):
    # zapisuje raport do pliku tekstowego.

    with open(filename, "w") as f:
        f.write("Raport:\n")
        f.write(f"Srednia: {srednia}\n")
        f.write(f"Min: {min_val}\n")
        f.write(f"Max: {max_val}\n")
        f.write(f"Mediana: {mediana}\n")
        f.write(f"Odchylenie standardowe: {odchylenie}\n")
        f.write(f"Mediana vs srednia: {mediana} {porownanie} {srednia}\n")
```

Program uruchamiam poleceniem

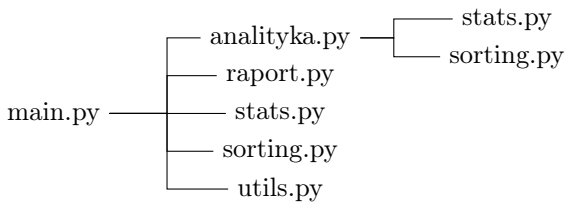
```
python main.py
```

3 Wyniki działania

Dla Zadania 1 i 2:

```
PS C:\Users\asiad\Desktop\paradygmaty\lab03> python main.py
Podaj liczbę elementów: 2
Podaj liczbę 1: 3
Podaj liczbę 2: 7
Posortowane dane: [3.0, 7.0]
Mediana vs średnia: 5.0 = 5.0
Średnia: 5.0, Min: 3.0, Max: 7.0
Mediana: 5.0, Odchylenie standardowe: 2.0
PS C:\Users\asiad\Desktop\paradygmaty\lab03> python main.py
Podaj liczbę elementów: 5
Podaj liczbę 1: 345
Podaj liczbę 2: 7657
Podaj liczbę 3: 45
Podaj liczbę 4: 4535
Podaj liczbę 5: 3
Posortowane dane: [3.0, 45.0, 345.0, 4535.0, 7657.0]
Mediana vs średnia: 345.0 < 2517.0
Średnia: 2517.0, Min: 3.0, Max: 7657.0
Mediana: 345.0, Odchylenie standardowe: 3086.76166880438
```

4 Opis modułów



Rysunek 1: Diagram zależności modułów.

4.1 Moduł main.py

Moduł główny wykorzystujący pozostałe moduły do wczytania danych, obliczenia średniej, wartości maksymalnej i minimalnej, mediany, posortowania danych, wypisania posortowanych danych oraz samodzielnie wypisuje obliczone wartości.

4.2 Moduł utils.py

Niezależny moduł zawierający funkcje do wczytania i wypisania danych.

4.3 Moduł sorting.py

Niezależny moduł zawierający funkcję sortującą.

4.4 Moduł stats.py

Niezależny moduł zawierający funkcje obliczajace sumę, średnią, wartość minimalną i maksymalną

4.5 Moduł raport.py

Niezależny moduł zawierający funkcje zapisującą raport z wyników obliczeń do pliku tekstowego.

4.6 Moduł analityka.py

Moduł zawierający funkcje obliczające medianę i odchylenie standardowe posiłkując się sortowaniem i obliczaniem wartości średniej.

5 Wnioski

Paradygmat modularny opiera się na podziale kodu na oddzielnie przechowywane moduły. Jest on kolejnym krokiem rozbudowy programu, tak jak paradygmat proceduralny podzielił paradygmat imperatywny na procedury, to paradygmat modularny rozdziela te procedury do oddzielnych plików.

Umożliwia to jeszcze sprawniejszą pracę w zespole nad jednym projektem. Zespoły mogą równolegle pracować nad różnymi modułami minimalizując konflikty. Po ustaleniu co dokładnie wejścia i wyjścia konkretnych funkcji każdy może zaczynać pracę nad swoim modułem niezależnie.

Dodatkowo, sprzyja to ponownemu użyciu już stworzonych modułów w różnych projektach. Z jasno określonym interfejsem modułów możliwe jest wielokrotne wykorzystywanie tych samych fragmentów kodu.

Ten paradygmat można stosować już nawet do dużych projektów.