Temat 4

**Zadanie 1.** Wariancja

Część 1. Funkcja:

**def** function(list):  
 average = 0  
 variance = 0  
 **for** x **in** list:  
 average += x / len(list)  
  
 **for** x **in** list:  
 variance += ((average - x) \*\* 2) / (len(list) - 1)  
 **return** (average, variance)

Definiuje funkcję przyjmującą listę **list** jako argument.

Przypisuje wartość zero zmiennym **avarage** i **variance** przechowujących średnią i wariancje.

Korzystam z faktu, że:

Gdzie **n = len(list)** = ilość liczb

Spostrzeżenie to umożliwia liczenie średniej tylko w pętli **for**.

*Wyciągnięcie przed nawias jest równoważne wyciągnięciu ilorazu* ***avarage*** *i* ***len(list)*** *poza pętlę* ***for****. Najpierw w pętli sumowałbym n liczb, a następnie po pętli wynik podzielił przez* ***len(list)****.*

Tworzę pętlę **for**, **x** przyjmuje wartości kolejnych zmiennych z listy.

W każdym obiegu pętli zwiększam wartość zmiennej **avarage** o **x/len(list),** czyli iloraz danego wyrazu i liczby wszystkich wyrazów.

W zmiennej **avarage** znajduje się teraz średnia elemetnów listy **list**.

Podobne podejście stosuje w liczeniu wariancji. Korzystam z zasady rozdzielności mnożenia względem dodawania:

Tworzę pętlę **for**, **x** przyjmuje wartości kolejnych zmiennych z listy.

W każdym obiegu pętli zwiększam wartość zmiennej **variance** o **((average - x) \*\* 2) / (len(list) - 1)**.

*Podobnie jak przy liczeniu średniej, dzielenie może zostać wykonane poza pętlą.*

W zmiennej **variance** znajduje się teraz wariancja elemetnów listy **list**.

Zwracam krotkę zawierającą parę liczb, będących wynikami -> **(avarage, variance)**

Część 2. Program

**def** program():  
 list = []  
 **while True**:  
 given\_number = int(input(**"Podaj liczbe: "**))  
 **if** given\_number == 0:  
 **break** list.append(given\_number)  
 print(function(list))  
 list2=[3,3,3,3]  
 print(function(list2))  
 list3=[5,6,7,8,9]  
 print(function(list3))

Definiuje funkcję **program()**

Tworzę pętlę **while**, działającą dopóki nie zostanie przerwana instrukcją **break**.

Wczytuje do zmiennej **given\_number** liczbę z klawiatury i rzutuje na typ całkowity

Jeśli podana liczba to zero, pętla kończy działanie, program przechodzi do kolejnych instrukcji.

W innym przypadku (**given\_number ≠ 0**) do listy dodajemy **given\_number**

Po zakończeniu działania pętli instrukcją **break**, program wypisuje wynik funkcji z nowo-utworzoną listą **list** jako parametr.

Program wypisuje zadane wartości jeszcze dwukrotnie dla **list** podanch w treści zadania.

**Zadanie 2.** Cyfry słowie

**def** translate(number):  
 slownik = { **"0"**:**"zero"**, **"1"**:**"jeden"**, ... , **"9"**:**"dziewięć"**}  
 result = **""  
 for** digit **in** number:  
 **if** digit **in** slownik:  
 result += slownik[digit] + **" "  
  
 return** result

Definiuje funkcję **translate(number),** gdzie number to liczba zapisana jako tekst (string)

Tworzę słownik którego kluczami są pojedyńcze znaki (cyfry typu string), zaś wartościami nazwy tych liczb (napis typu string)

Definiuje zmienną **result** będącą pustym napisem.

Tworzę pętlę **for**, **digit** jest kolejnymi znakami napisu **number**.

Jeżeli **digit** jest kluczem w słowniku (czyli jest cyfrą) do wyniku (**result**) dodaję **slownik[digit]**, czyli nazwę liczby, którą aktualnie jest digit, zakończoną spacją.

Po zakończeniu działania pętli zwracam napis zapisany w zmiennej **result**.

**Zadanie 3.** Suma krotek

**def** sum\_of\_tuples(list):  
 results\_list = []  
 **for** tuple **in** list:  
 results\_list.append(sum(tuple))  
 **return** results\_list

Definiuje funkcję **sum\_of\_tuples(list)** przyjmującą jako argument listę krotek.

Tworzę pustą tablicę w której będę trzymał wyniki.

Tworzę pętlę **for**, **tuple** jest równe kolejnym krotkom z listy **list**.

W każdym obiegu pętli dodaje do listy wyników sumę danej krotki(**tuple**) z **list**

Zwracam listę wyników (**result\_list**) jako poprawny wynik

**Zadanie 4.** Liczba z napisu

**def** translate(number):  
  
 slownik = {  
 **"dwadzieścia"**:**"2"**,  
 . . .  
 **"pięćdziesiąt"**:**"5"**,  
 **"jeden"**:**"1"**,  
 . . .  
 **"dziewięć"**:**"9"**,}  
  
 digits = number.split(**" "**)  
 result = slownik[digits[0]]  
  
 **if** len(digits) > 1:  
 result += slownik[digits[1]]  
 **else**:  
 result += **"0"  
  
 return** int(result)

Tworzę funkcję **translate(number),** przyjmującą jako argument liczbę całkowitą z zakresu **[20;59]**

Tworzę odpowiedni słownik.

Tworzę tablicę **digits** za pomocą funkcji **string.split(“ “)**, dzielącej napis na słowa względem określonego znaku.

Ustalam wartość zmiennej **result** jako napis będący “tłumaczeniem” pierwszego słowa.

Jeżeli dlugość tablicy jest większa od **1**, oznacza to, że wyrazów było więcej niż jeden czyli dwa.

Dopisuje do zmiennej **result** “przetłumaczone” drugie słowo z tablicy **digits**.

W innym przypadku (dwadzieścia, trzydzieści…) dopisuje zero.

Zwracam **result** rzutowany na typ całkowity.

**Zadanie 5.** “Książka telefoniczna”

ksiazka = {(**'Jan'**, **'Kowalski'**):**"123456789"**, (**'Adam'**, **'Nowak'**):**"987654321"** , (**'Adam'**, **'Kowalski'**): **"600300900"**}

s

**def** find\_numbers(surname):  
 numbers = []  
 **for** tuple **in** ksiazka.keys():  
 **if** tuple[1] == surname:  
 numbers.append(ksiazka[tuple])  
 **return** numbers

Definiuje funkcję **find\_numbers(surname)** przyjmującą jako argument napis, będący szukanym nazwiskiem w książce

Tworzę tablicę do której będę wpisywał numery osób o nazwisku **surname**

**dictionary.keys()** zwraca listę kluczy ze słownika

Tworzę pętlę **for**, krotka **tuple** przyjmuje wartości kolejnych krotek z listy kluczy słownika ksiazka.

Jeżeli napis (nazwisko) na drugiej pozycji w krotce **tuple** jest taki sam jak **surname** (poszukiwane nazwisko) wtedy dodaje do znalezionych numerów (**numbers**) wartość ze słownika ksiazka dla klucza **tuple**.

W innym przypadku nic się nie dzieje w danym obiegu pętli.

Po wyjściu z pętli zwracam tablice z numerami pasującymi do szukanego nazwiska.

Zgodnie z poleceniem należy wywołać:

print(ksiazka[(**'Jan'**, **'Kowalski'**)])  
print(find\_numbers(**"Kowalski"**))