Marzena Nowakowska, Wydział Zarządzania i Modelowania Komputerowego, PŚk

**Temat 3. Iteracje**

Zad. 1.

Napisać program *Zad03_01*, który pobiera od użytkownika dowolny napis a następnie na podstawie treści napisu tworzy dwie listy: jedna składa się ze znaków które są samogłoskami a druga lista to te znaki w napisie, które nie są samogłoskami. Występujące w obu listach litery mają być duże.

```
# -*- coding: utf-8 -*-
napis=input(„Podaj dowolny napis: „)
for el in napis:
    print(el)
    samogloski=[] nie_samogloski=[] for el in napis:
    if el.lower() in [’a’, ‘ą’, ‘e’, ‘ę’, ‘i’, ‘o’, ‘ó’,’u’, ‘y’]:
    samogloski.append(el.upper()) else:
    nie_samogloski.append(el.upper())
    samogloski1=[el.upper() for el in napis if el.lower() in [’a’, ‘ą’, ‘e’, ‘ę’, ‘i’, ‘o’, ‘ó’, ‘u’, ‘y’] ]
 nie_samogloski1 =[el.upper() for el in napis if el.lower() not in [’a’, ‘ą’, ‘e’, ‘ę’, ‘i’, ‘o’, ‘u’, ‘y’] ]
```

Sprawdzić i objaśnić (wykład, Internet), jak działa polecenie złożenia (dwie ostatnie instrukcje programu) wykorzystywane do tworzenia zmiennych: *samogloski1* i *nie_samogloski1*.

Zad. 2.

Utworzyć program *Zad03_02* , który będzie modyfikacją programu *Zad03_01* jak następuje. Na podstawie łańcucha pobranego od użytkownika utworzyć trzy zmienne listowe o podanej niżej zawartości:

- *Samogłoski* – lista zawierająca małe litery napisu będące samogłoskami (wykorzystać złożenie z warunkiem filtrującym samogłoski)
- *Cyfry* – lista zawierająca cyfry występujące w napisie (wykorzystać złożenie z warunkiem filtrującym cyfry za pomocą przedziału domkniętego [*’a’, ‘z’*]
- *Spolgloski* – lista zawierająca małe litery napisu będące spółgłoskami (wykorzystać złożenie lub pętlę).

Do rozwiązania zadania można wykorzystać kody liter (np. małych), w tym polskich znaków diakrytyzowanych: ą, ć, ę, ł, ń, ó, ś, ź, ż. Funkcja wbudowana *ord* zwraca kod Unicode dla jednego znaku, np. *ord('ą')* → 268.

Wyznaczyć, ile jest wszystkich znaków w napisie, ile wśród nich jest samogłosek, spółgłosek i cyfr.

Zad 3.

Napisać program *Zad03_03*, który pobiera od użytkownika ciąg liczb rzeczywistych oddzielonych spacją i wyznacza proste statystyki dla liczb dodatnich i ujemnych, jak podano w programie poniżej.

```
    s = input(„Podaj ciąg liczb rzeczywistych oddzielonych spacjami: „) print(s)
liczby = list(map(float, s.split()))      # 1
print (liczby)

    # Wyznaczenie sredniej liczb dodatnich w zbiorze ‘liczby’
    sumaDod=sum([l if l>0 else 0 for l in liczby])
    liczbaDod=sum([1 if l>0 else 0 for l in liczby])
    sredDod=sumaDod/liczbaDod
# Wyznaczenie sredniej liczb ujemnych w zbiorze ‘liczby’
# sumaUj= <-- uzupełnić i usunąć #
# liczbaUj = <-- uzupełnić i usunąć #
# sredUj = <-- uzupełnić i usunąć #

    # Wyprowadzić wyniki poprzedzone odpowiednim komentarzem
```

1. Wprowadzić w programie zabezpiuecznie przed dzieleniem przez zero.
2. Omówić, jak działa metoda *split*() wywołana na rzecz napisu. Jaki znak jest domyślnym separatorem znaków łańcuch w wynikowej liście?
3. Omówić funkcję *map*(). Objaśnić działanie instrukcji opatrzonej numerem 1.
4. Rozwiązać zadanie w programie *Zad03_03a* wykorzystując pętlę *for* zamist złożenia (wybrac jeden wariant dla liczb dodatnich lub ujemnych).

Zad. 4.

Napisać program *Zad03_04*, który pobiera od użytkownika ciąg liczb rzeczywistych separowanych przecinkiem i tworzy z tych liczb listę *liczby*. Wykonać na liście operacje:

a) Utworzyć w zmiennej *liczbySort* kopię listy *liczby*, po czym uporządkować listę *liczbySort* rosnąco.
b) Wyznaczyć miary statystyczne w liście *liczbySort* :
   - *wartoscMin*, *wartoscMaks* – wartości największą i najmniejszą; odpowiednio pierwszy element w *liczbySort* (lub wykorzystać funkcję *min*) i ostatni element w *liczbySort* (lub wykorzystać *max*)
   - *wartoscMed* – medianę czyli wartość środkową; jej wartość zależy od liczby elementów ciągu, jak zilustrowano poniżej:

nieparzysta liczba elementów (wartość w środku listy):

1 3 5 5 5 **7** 8 9 9 10 11

parzysta liczba elementów (średnia arytmetyczna elementów najbliżej środka) :

1 3 5 5 5 7 8 9 9 10

**6**

wartoscMod – wartość modalną (dominantę) czyli wartość występującą najczęściej

```
liczbySort=[1,3,5,5,5,7,8,9,9,10,11]#  przykładowa uporządkowana lista liczbyZb=set(liczbySort)
  krotnosci={}                          # 1
  for el in liczbyZb:
    krotnosci[el]=liczbySort.count(el)  # 2
    krotMod=max(krotnosci.values())
    for klucz in krotnosci.keys():      # 3
    if krotnosci[klucz]==krotMod:
wartoscMod =klucz
break
```

Objaśnić instrukcje opatrzone numerami: 1, 2, 3.
Jak rozwiązać ww. punkt zadania, gdy jest kilka takich wartości (wielomodalność)?

   - *wartoscRoz* – wartość rozstępu, czyli różnicę między maksimum i minimum

Zad. 5.

Napisać program *Zad03_05*, który koduje zdanie w języku angielskim wykorzystując szyfr Cezara. Szyfr Cezara polega na zastąpieniu każdej litery tekstu jawnego literą występującą w alfabecie o 3 pozycje później. Np. literę A szyfruje się jako D, B jako E itd. W programie operować szyfrem Cezara tylko dla dużych liter. Znaki inne niż litery alfabetu pozostają w zakodowanym zdaniu niezmienione. Np. zdanie *I like #Python and C++ very much* jest zaszyfrowane następująco: *L OLNH #SBWKRQ DQG F++ YHUB PXFK*. Wykorzystać słownik (por. początek rozwiązania poniżej).

```
# Szyfrowanie kodem Cezara tekstów w j. angielskim
kody={'A': „D" , 'B': „E" , 'C': „F" , 'D': „G" , 'E': „H" , 'F': „I" ,
'G': „J" , 'H': „K" , 'I': „L" , 'J': „M" , 'K': „N" , 'L': „O" ,
'M': „P" , 'N': „Q" , 'O': „R" , 'P': „S" , 'Q': „T" , 'R': „U" ,
'S': „V" , 'T': „W" , 'U': „X" , 'V': „Y" , 'W': „Z" , 'X': „A" ,
'Y': „B" , 'Z' : „C"}
zdanie=input(„Podaj zdanie do zaszyfrowania:").upper()
zakodowane=""
# Skończyć zadanie; można rozwiązać wykorzystując pętlę for lub złożenie
```