Wydział: **Zarządzania i Modelowania Komputerowego**
Katedra Technologii Informatycznych
Przedmiot: **Technologie informacyjne**
Rok **I**

PYTHON - ĆWICZENIE 4

W poprzednim ćwiczeniu pojawiło się już pojęcie **lista**, teraz zostanie ono dokładniej opisane. W przypadku sekwencji wygenerowanej za pomocą **range** nie jest ona zachowywana w pamięci komputera.

1. Lista

W wielu zadaniach konieczne jest zachowanie wszystkich elementów wygenerowanej sekwencji. Aby przechowywać takie dane, w Pythonie można użyć struktury danych nazywanej **listą** (w większości języków programowania używany jest inny termin – *ciąg* lub *tablica*). Listę można zdefiniować wymieniając jej elementy w nawiasach kwadratowych, oddzielając je przecinkami. Listy mogą zawierać elementy różnych typów, ale w większości zastosowań wszystkie elementy listy są tego samego typu. Można też tworzyć listę, której elementami są inne listy. Lista jest zmiennym typem danych - można ją modyfikować zmieniając, dodając i usuwając jej elementy.

```
>>> liczby=[2,4,5,11,6]
>>> kolory=['biały','czerwony','zielony','złoty']
>>> inna=[2.4,-4,'zima',2.3e-4]
>>> nowa=[[1,2,3],'wynik',[-11.3,-2]]
>>>
```

Listę można definiować stosując **list.**

list(*sekwencja*)

Ta metoda definiowania listy była ilustrowana w poprzednim ćwiczeniu, przy omawianiu **range.**

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

Do elementu listy można się odwołać podając jego pozycję na liście (indeks), przy czym numeracja elementów zaczyna się od **0**

```
>>> print(liczby[0])
2
>>> print(liczby[3])
11
>>> print(kolory[0])
biały
>>>
```

Elementy listy mogą też mieć indeks ujemny:

```
>>> print(kolory[-1])
złoty
>>> print(kolory[-3])
czerwony
>>>
```

Indeks ujemny oznacza, że zaczynamy od ostatniego elementu (ma on indeks -1) i przechodzimy w lewo.

Liczbę elementów listy (długość listy) zwraca funkcja **len**, ta sama, która zwraca liczbę znaków w łańcuchu znaków (napisie):

```
>>> n=len(liczby)
>>> print(n)
5
>>>
```

Poniżej przedstawiono dwa sposoby wyświetlenia elementów listy **kolory**. W pierwszym przypadku do elementów listy odwołano się przez podanie indeksu elementu. Zmienna sterująca **i** przyjmuje wartości z sekwencji utworzonej za pomocą **range**, czyli kolejno 0,1,2,3. W drugim przypadku zmienna sterująca **x** wskazuje wprost na kolejne elementy listy **kolory.**

```
>>> for i in range(len(kolory)):
        print(kolory[i])


biały
czerwony
zielony
złoty
>>>
```

```
>>> for x in kolory:
        print(x)


biały
czerwony
zielony
złoty
>>>
```

Listy mogą być wczytywane na różne sposoby.

Przykład 1.

Utwórz pustą listę (taką bez elementów, jej długość to **0**), następnie wczytaj informację o liczbie elementów tworzonej listy i wczytaj kolejno te elementy. Wykorzystaj **append** – dołączanie elementu na końcu listy.

```
File  Edit  Format  Run  Options  Window  Help
a=[]
n=int(input('ile elementów ma mieć lista? n='))
for i in range(n):
    nowy=int(input('nowy element: '))
    a.append(nowy)
print(a)
```

Przykładowe wyniki:

```
================== RESTART: D:/PYTHON,
ile elementów ma mieć lista? n=5
nowy element: 2
nowy element: 3
nowy element: 5
nowy element: 7
nowy element: 11
[2, 3, 5, 7, 11]
>>>
```

Warto zauważyć, że dwie linie programu, stanowiące zakres pętli, można zastąpić jedną:

```
File  Edit  Format  Run  Options  Window  Help
a=[]
n=int(input('ile elementów ma mieć lista? n='))
for i in range(n):
    a.append(int(input('nowy element: ')))
print(a)
```

Nową listę można utworzyć przez konkatenację (złączenie) list już istniejących oraz przez zwielokrotnienie (inaczej powtarzanie, mnożenie listy przez liczbę):

```
>>> a=[1,2,3]
>>> b=[-4,-3,-2,-1]
>>> c=a+b
>>> print(c)
[1, 2, 3, -4, -3, -2, -1]
>>> d=4*a
>>> print(d)
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

Zadanie z przykładu 1 można zrealizować tworząc n-elementową listę złożoną z samych zer, a następnie zastąpić te zera kolejno wczytywanymi elementami.

```
File  Edit  Format  Run  Options  Window  Help
n=int(input('ile elementów ma mieć lista? n='))
a=[0]*n
for i in range(n):
    a[i]=int(input('nowy element: '))
print(a)
```

Przykład 2.

Listę można wczytać podając ją na wejściu w jednym wierszu. Należy tu przypomnieć, że instrukcja **input()** przypisuje wskazanej zmiennej łańcuch znaków podany na wejściu. Jeżeli wejściowy wiersz będzie zawierał elementy, które mają zostać umieszczone na liście, należy rozdzielić je umownym znakiem (domyślnie jest to spacja), a potem zastosować metodę **split:**

lista.**split**(*znak*)

która zwraca listę złożoną z fragmentów łańcucha wejściowego uzyskanych po „rozdzieleniu” wskazanym znakiem. Poniżej zilustrowano zastosowanie jako znaku rozdzielającego spacji i przecinka:

```
>>> s=input()
1 23 -4 11
>>> a=s.split()
>>> print(a)
['1', '23', '-4', '11']
>>>
>>> s=input()
1,23,-4,11
>>> a=s.split(',')
>>> print(a)
['1', '23', '-4', '11']
>>>
```

Ponieważ tak uzyskana lista składa się z napisów (łańcuchów znakowych), w celu uzyskania elementów liczbowych trzeba dokonać konwersji.

```
>>> for i in range(len(a)):
        a[i]=int(a[i])


>>> print(a)
[1, 23, -4, 11]
>>>
```

Operacja wprowadzania listy może być zapisana krócej, jak w poniższym skrypcie:

```
File  Edit  Format  Run  Options  Window  Help
print('podaj elementy listy rozdzielone spacjami ')
a=input().split()
for i in range(len(a)):
    a[i]=int(a[i])
print(a)
```

po uruchomieniu którego:

```
================= RESTART: D:/PYTHON
podaj elementy listy rozdzielone spacjami
1 3 5 7 11 -8 -6 -4
[1, 3, 5, 7, 11, -8, -6, -4]
>>>
```

2. Generator

Aby utworzyć bardziej skomplikowane listy, można użyć **generatora,** wyrażenia pozwalającego wypełnić listę zgodnie z formułą. Ogólna forma generatora jest następująca:

[*wyrażenie* **for** *zmienna* **in** *sekwencja*]

gdzie *wyrażenie* opisuje sposób tworzenia elementów listy (zazwyczaj w zależności od zmiennej), *zmienna* jest identyfikatorem pewnej zmiennej, *sekwencja* jest ciągiem wartości, które kolejno przyjmuje zmienna – może to być lista lub obiekt uzyskany za pomocą **range.**

Przykładowo, inny sposób tworzenia listy wypełnionej zerami:

```
>>> n=5
>>> a=[0 for i in range(n)]
>>> print(a)
[0, 0, 0, 0, 0]
>>>
```

Lista zbudowana z kwadratów początkowych liczb całkowitych dodatnich:

```
>>> n=7
>>> a=[i**2 for i in range(1,n+1)]
>>> print(a)
[1, 4, 9, 16, 25, 36, 49]
>>>
```

Z wykorzystaniem generatora skrypt z przykładu 1 można zapisać następująco:

```
File  Edit  Format  Run  Options  Window  Help
a=[int(input('nowy element: ')) for i in range(int(input('ile elementów n= ')))]
print(a)
```

a skrypt z przykładu 2:

```
File  Edit  Format  Run  Options  Window  Help
print('podaj elementy listy rozdzielone spacjami ')
a=[int(x) for x in input().split()]
print(a)
```

3. Operacje na listach

Na listach można wykonywać wiele różnych operacji. W przykładzie 1 została pokazana metoda **append:**

lista.**append**(*element*)

umożliwiająca dołączenie do listy, na jej końcu, nowego elementu:

```
>>> a=[2,3,5,7,11,13]
>>> a.append(17)
>>> print(a)
[2, 3, 5, 7, 11, 13, 17]
>>>
```

Nowy element listy można wstawić na dowolną pozycję, powodując „rozsunięcie" już istniejących elementów. Służy do tego metoda **insert**:

lista.**insert**(*pozycja*, *element*)

```
>>> a=[2,3,5,7,11,13]
>>> a.insert(0,-4)
>>> print(a)
[-4, 2, 3, 5, 7, 11, 13]
>>> a.insert(4,-8)
>>> print(a)
[-4, 2, 3, 5, -8, 7, 11, 13]
>>>
```

Listę można rozszerzyć o dodatkową listę, stosując metodę **extend:**

lista.**extend**(*lista_plus*)

```
>>> a=[2,3,5,7,11,13]
>>> a.extend([-12,-10,-8])
>>> print(a)
[2, 3, 5, 7, 11, 13, -12, -10, -8]
>>>
```

Z listy można, za pomocą metody **remove** usunąć wskazany element (jego pierwsze wystąpienie):

lista.**remove**(*element)*

```
>>> a=[2,3,5,7,11,13]
>>> a.remove(7)
>>> print(a)
[2, 3, 5, 11, 13]
>>>
```

Jeśli wskazany *element* nie występuje na liście, sygnalizowany jest błąd.

Metoda **pop:**

lista.**pop**(*pozycja*)

usuwa element z podanej *pozycji* na liście i zwraca go; jeśli nie podano parametru *pozycja* (jest on opcjonalny) metoda usuwa i zwraca ostatnią pozycję z listy:

```
>>> a=[2,3,5,7,11,13]
>>> a.pop(3)
7
>>> print(a)
[2, 3, 5, 11, 13]
>>>
```

```
>>> a=[2,3,5,7,11,13]
>>> a.pop()
13
>>> print(a)
[2, 3, 5, 7, 11]
>>>
```

Wszystkie elementy listy usunie metoda **clear**

*lista.***clear**()

```
>>> a=[2,3,5,7,11,13]
>>> a.clear()
>>> print(a)
[]
>>>
```

Metoda **count** zwraca liczbę wystąpień elementu na liście

*lista.***count**(*element*)

```
>>> a=[2,3,5,2,7,11,2,3,13]
>>> a.count(2)
3
>>> a.count(-7)
0
>>>
```

Pozycję, na której po raz pierwszy wystąpi *element* zwraca metoda **index**

*lista.***index**(*element*)

Powyższa metoda może mieć opcjonalnie dwa dodatkowe parametry, zawężające zakres poszukiwań:

*lista.***index**(*element, początek, koniec*)

Zwrócony indeks jest obliczany względem początku pełnej sekwencji (indeksu 0), a nie argumentu *początek*; umożliwia to wskazanie pozycji kolejnego wystąpienia szukanego *elementu.*

```
>>> a=[1,3,5,2,7,11,2,3,13]
>>> a.index(2)
3
>>> a.index(2,4,7)
6
>>>
```

Przy zastosowaniu metody **sort** można posortować elementy listy:

*lista.***sort**()

```
>>> a=[1,11,4,3,2,13,5,7,-3]
>>> a.sort()
>>> print(a)
[-3, 1, 2, 3, 4, 5, 7, 11, 13]
>>>
```

Metoda **reverse** utworzy listę złożoną z elementów ustawionych w odwrotnej kolejności

*lista.***reverse**()

```
>>> a=[2,3,5,2,7,11,2,3,13]
>>> a.reverse()
>>> print(a)
[13, 3, 2, 11, 7, 2, 5, 3, 2]
>>>
```

4. Wycinki (plasterki)

Operując na listach można posługiwać się wycinkami (**slice**). Zasady tworzenia wycinków są następujące:

Gdy **a** jest listą, to

a[i:j] jest wycinkiem o elementach a[i], a[i+1],…, a[j-1]

a[i:j:-1] jest wycinkiem o elementach a[i], a[i-1],…, a[j+1] (czyli zmiana kolejności elementów; j musi być mniejsze od i)

a[i:j:k] wycięte z krokiem **k** – elementy a[i], a[i+k], a[i+2*k],…; gdy **k<0** elementy występują w odwrotnej kolejności

Pominięcie **i** lub **j** oznacza, że w miejsce **i** przyjęto początek listy (i=0), w miejsce **j** koniec listy (j=len(a)-1 lub, używając innego odwołania, j=-1), skąd wynika, że a[:]==a

Przykład 3.

```
>>> a=[1,2,3,4,5,6]
>>> a[3:5]=[7,8,9]
>>> print(a)
[1, 2, 3, 7, 8, 9, 6]
>>>
```

Elementy a[3], a[4] zostały zastąpione nową listą trzech elementów;

```
>>> a=[1,2,3,4,5,6,7,8,9]
>>> a[::-2]=[50,40,30,20,10]
>>> print(a)
[10, 2, 20, 4, 30, 6, 40, 8, 50]
>>>
```

Tu wycinek a[::-2] to elementy a[-1], a[-3], a[-5], a[-7], a[-9] (korzystając z tego, że len(a)=9 można zauważyć, że elementy wycinaka to: a[8], a[6], a[4], a[2], a[0]) i te właśnie elementy zostały zastąpione przez elementy 50, 40, 30, 20, 10.

Uwaga Jeśli wycinkowi nieciągłemu (takiemu z krokiem k>1 lub k<-1) przypisywana jest nowa wartość, to liczba elementów zastępowanych i zastępujących musi być taka sama – w przeciwnym razie wystąpi błąd:

```
>>> a=[1,2,3,4,5,6,7,8,9]
>>> a[::-2]=[30,20,10]
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    a[::-2]=[30,20,10]
ValueError: attempt to assign sequence of size 3 to extended
slice of size 5
>>>
```

Zadanie 1.

Napisz skrypt **numery.py** w którym do listy **a** wczytywane jest **n** liczb całkowitych, wyznaczana jest średnia arytmetyczna tych liczb, a następnie podawane są indeksy elementów listy większych od obliczonej średniej.

Przykładowy skrypt:

```
File Edit Format Run Options Window Help
n=int(input('ile liczb wprowadzasz? n='))
print('wpisz te liczby w jednej linii oddzielając spacjami')
a=[int(x) for x in input().split()]
s=0
for x in a: s+=x
s=float(s)/n
print('średnia =%6.2f' % s)
indeksy=[]
for i in range(n):
    if a[i]>s: indeksy.append(i);
print('lista pozycji elementów większych od średniej: ',indeksy)
```

i jego uruchomienie:

```
========= RESTART: D:/PYTHON
ile liczb wprowadzasz? n=6
wpisz te liczby w jednej linii oddzielając spacjami
1 7 6 4 10 5
średnia =  5.50
lista pozycji elementów większych od średniej:  [1, 2, 4]
>>>
```

Inne rozwiązanie:

```
File Edit Format Run Options Window Help
print('wpisz liczby w jednej linii oddzielając spacjami')
a=[int(x) for x in input().split()]
s=sum(a)/len(a)
print('średnia =%6.2f' % s)
indeksy=[]
for i in range(len(a)):
    if a[i]>s: indeksy.append(i);
print('lista pozycji elementów większych od średniej: ',indeksy)
```

To drugie rozwiązanie wykorzystuje funkcję **len**, zwracającą liczbę elementów listy i funkcję **sum**, której wynikiem jest suma elementów listy.

Zadanie 2.

Dla danej listy liczb rzeczywistych napisz skrypt **mini.py**, który po wczytaniu liczb do listy wyznacza element minimalny , a następnie generuje listę pozycji na których w liście znajdują się elementy równe minimalnemu.

```
File Edit Format Run Options Window Help
print('wpisz elementy listy w jednej linii oddzielając spacjami')
b=[float(x) for x in input().split()]
el_min=min(b)
print('element minimalny min =%6.2f' % el_min)
indeksy=[]
for i in range(len(b)):
    if b[i]==el_min: indeksy.append(i);
print('lista pozycji elementów równych min: ',indeksy)
```

W powyższym skrypcie, którego przykładowe uruchomienie jest następujące:

```
========= RESTART: D:/PYTHON
wpisz elementy listy w jednej linii oddzielając spacjami
2.4 7.2 2.4 3.6 5.1 7.8
element minimalny min =  2.40
lista pozycji elementów równych min:  [0, 2]
>>>
```

została wykorzystana funkcja **min**, która zwraca najmniejszy element listy (istnieje analogiczna funkcja **max**, zwracająca element największy). Skrypt, który pokazuje zastosowany przez funkcję **min** algorytm wyszukiwania elementu minimalnego zamieszczono poniżej.

```
File  Edit  Format  Run  Options  Window  Help
print('wpisz elementy listy w jednej linii oddzielając spacjami')
b=[float(x) for x in input().split()]
el_min=b[0]
for x in b:
    if x<el_min:el_min=x
print('element minimalny min =%6.2f' % el_min)
indeksy=[]
for i in range(len(b)):
    if b[i]==el_min: indeksy.append(i);
print('lista pozycji elementów równych min: ',indeksy)
```

Zadanie 3.

Napisz skrypt liczby.py, który po wprowadzeniu listy liczb całkowitych podaje informacje:

- ile jest na liście liczb ujemnych
- sumę kwadratów tych liczb na liście
- średnią arytmetyczną dodatnich elementów listy

Przetestuj rozwiązanie zapisane w skrypcie:

```
File  Edit  Format  Run  Options  Window  Help
print('wpisz liczby w jednej linii oddzielając spacjami')
a=[int(x) for x in input().split()]
ile=0
for x in a:
    if x<0:ile+=1
print('liczb ujemnych jest: ', ile)
c=0
for x in a:
    c+=x*x
print('suma kwadratów =', c)
d=0
l=0
for x in a:
    if x>0:
        d+=x
        l+=1
if l>0:
    print('średnia elementów dodatnich =%6.2f' % (d/l))
else:
    print('na liście nie ma elementów dodatnich')
```

i takie, w którym zostały zastosowane konstruktory tworzące trzy listy – pierwszą złożoną z elementów ujemnych listy danej, drugą zbudowaną z kwadratów elementów i trzecią zawierającą tylko elementy dodatnie wczytanej na początku listy liczb:

```
File  Edit  Format  Run  Options  Window  Help
print('wpisz liczby w jednej linii oddzielając spacjami')
a=[int(x) for x in input().split()]
b=[x for x in a if x<0]
print('liczb ujemnych jest: ', len(b))
c=[x*x for x in a]
print('suma kwadratów =', sum(c))
d=[x for x in a if x>0]
if len(d)>0:
    print('średnia elementów dodatnich =%6.2f' % (sum(d)/len(d)))
else:
    print('na liście nie ma elementów dodatnich')
```

Czy w obu przypadkach przykładowe rozwiązanie jest następujące?

```
========== RESTART: D:/PYTHON
wpisz liczby w jednej linii oddzielając spacjami
-2 7 1 4 -1 5
liczb ujemnych jest:  2
suma kwadratów = 96
średnia elementów dodatnich =  4.25
>>>
```

Zadania dodatkowe

Napisz, uruchom i przetestuj skrypty:

- **gole.py**, który analizuje dwie listy **x** i **y** zawierające wyniki spotkań piłkarskich; na liście **x** zapisana jest liczba bramek zdobytych przez gospodarzy, na liście **y** liczba bramek strzelonych przez gości; należy wyznaczyć:
  - średnią liczbę goli zdobytych przez gości w spotkaniach zakończonych zwycięstwem gospodarzy,
  - średnią liczbę bramek strzelonych w meczu,
  - ile spotkań zakończyło się różnicą bramek większą od 1,
  - maksymalną różnicę bramek w meczu i podać w którym spotkaniu (podać pozycję tego spotkania na listach **x** i **y**) taka różnica miała miejsce
- **test.py**, który po wprowadzeniu listy liczb całkowitych nazwanej **liczby** wprowadza do zmiennej **c** jeszcze jedną liczbę całkowitą, a następnie:
  - tworzy dwie nowe listy zbudowane z elementów listy **liczby** odpowiednio podzielnych przez **c** i niepodzielnych przez **c**
  - podaje informację, która z nowo utworzonych list ma więcej elementów
  - oblicza wartość średnią elementów tej dłuższej listy