rozdział 5:

LISTY

ostatnia modyfikacja: 04.12.18

Podstawy programowania

sortowanie:

ustawianie danych w kolejności wynikającej z pewnego kryterium

- jedna z elementarnych czynności algorytmicznych
- wiele sposobów o różnej złożoności i różnej wydajności
- wdzięczny temat badań i rozważań

Podstawy programowania

sortowanie: 1, 13, 0, 9, 5

- rosnące (niemalejące) → 0, 1, 5, 9, 13
- malejące (nierosnące) → 13, 9, 5, 1, 0
- naturalne
- nienaturalne
- ...

Podstawy programowania

zadanie trywialne:

posortować niemalejąco dwie liczby

```
a=int(input())
b=int(input())
#
# ?
#
```

Podstawy programowania

posortować niemalejąco dwie liczby

```
a=int(input())
b=int(input())
if a<b:
    print(a,b)
else:
    print(b,a)
```

Podstawy programowania

zadanie ciekawe:

posortować niemalejąco trzy liczby

```
a=int(input())
b=int(input())
c=int(input())

#
# ?
#
```

Podstawy programowania

zadanie ciekawe:

posortować niemalejąco trzy liczby

```
a=int(input())
b=int(input())
c=int(input())
if a <= b and a <= c:
    print(a,end=" ")
    if b > c:   print(c,b)
    else:       print(b,c)
elif b <= a and b <= c:
    print(b,end=" ")
    if a > c:   print(c,a)
    else:       print(a,c)
elif c <= a and c <= b:
    print(c,end=" ")
    if(b > a):  print(a,b)
    else:       print(b,a)
else:  print(a,b,c)
```

Niespodzianka!
Można tak pisać, jeśli za **if** stoi tylko jedna instrukcja… ale nie jest to zalecany styl, bo kod traci na czytelności

Podstawy programowania

zadanie mordercze:

posortować niemalejąco cztery liczby

?

Podstawy programowania

zadanie niewykonalne:

posortować niemalejąco tysiąc liczb

?!!

Podstawy programowania

co z tego wynika?

- sortowanie oraz inne czynności podobnej natury **nie dadzą** się realizować na pojedynczych zmiennych
- potrzebny jest kontener, który potrafi przechowywać jednocześnie **więcej niż jedną daną**

Podstawy programowania

co to jest **lista?**

- **lista** to **agregat** danych
- w innych językach znana pod nazwą **tablica**
- zmienna zmiennych
- zmienna → jedna dana (np. liczba)
- lista → dowolnie wiele (w tym zero) **ponumerowanych** danych
- liczba danych w liście może zmieniać się w czasie

Podstawy programowania

zmienna o nazwie „zmienna“ przechowuje liczbę całkowitą o wartości 12

```
zmienna = 12
lista = [1,3,7,15]
```

lista o nazwie „lista“ przechowuje cztery liczby całkowite (a wartości jakie są, każdy widzi)

Podstawy programowania

```
pusta = [ ]
```

ta lista jest pusta – nie zawiera żadnych danych

```
zera = [0] * 10
```

ta lista zawiera dziesięć zer; zauważ, że tym kontekście operator * nie oznacza mnożenia, a **powielanie** → jeszcze do tego wrócimy

Podstawy programowania

listy:

- elementy listy są **numerowane**
- pierwszy element listy ma zawsze **numer 0**
- ostatni – o jeden mniejszy, niż liczba elementów w liście
- numer elementu w liście → **indeks**

Podstawy programowania

```
lista[indeks]
```

w ten sposób uzyskujemy dostęp do elementu o indeksie „indeks“ → tym samym spośród wielu elementów zgromadzonych w liście możemy wybrać ten, który ma interesujący nas numer

Podstawy programowania

```
a = lista[0]
b = lista[a]
c = lista[2 * a + 1]
d = lista[ lista[0] ]
e = lista[ int(input()) ]
```

indeks może być literałem, zmienną, a nawet dowolnie złożonym wyrażeniem!

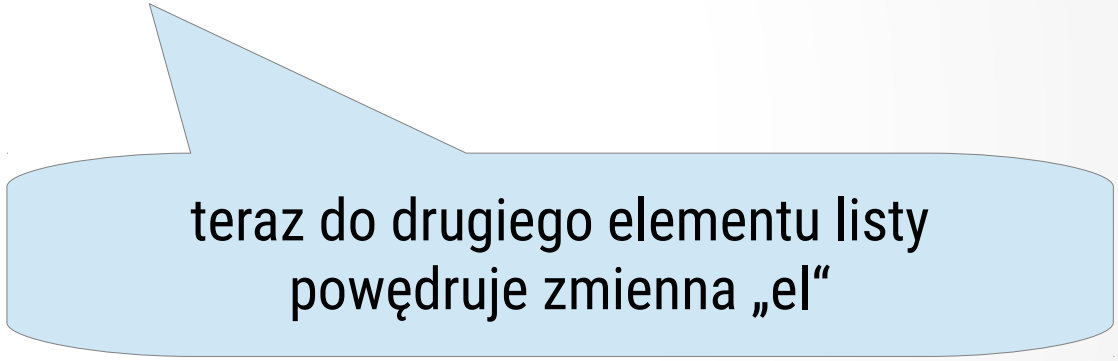
Podstawy programowania

```
lista = [1, 10, 4, 5, 7]
el = lista[0]
```

do zmiennej „el“ powędruje pierwszy element listy

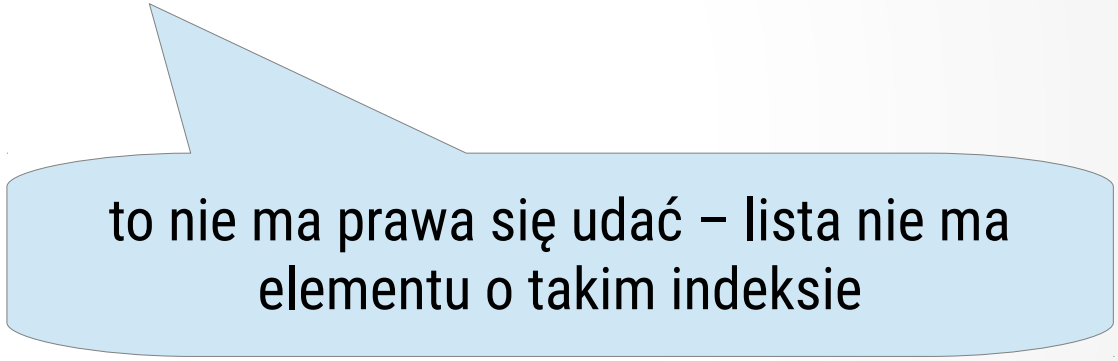
Podstawy programowania

```
lista[1] = el
```



Podstawy programowania

```
lista = [1, 10, 4, 5, 7]
el = lista[10]
```




```
Traceback (most recent call last):
  File "z.py", line 2, in <module>
    el = lista[10]
IndexError: list index out of range
```

Podstawy programowania

```
lista = [1, 10, 4, 5, 7]
lista[10] = 1
```

to również się nie uda – z tej samej przyczyny co poprzednio

```
Traceback (most recent call last):
  File "z.py", line 2, in <module>
    lista[10] = 1
IndexError: list assignment index out of range
```

Podstawy programowania

```
lista = [1, 10, 4, 5, 7]
el = lista[-1]
```

niespodzianka – to się uda!
indeksowanie liczbami ujemnymi oznacza **indeksowanie od końca**, czyli:

`lista[-1]` – ostatni element
`lista[-2]` – przedostatni element
etc.

Podstawy programowania

listy:

- elementy listy (jeśli nie są listami!) zachowują się jak zwykłe zmienne
- tak też można ich używać

Podstawy programowania

- **posumowanie wszystkich elementów listy:**

```
lista = [0, 3, 12, 8, 2]
suma = 0
for i in range(5):
    suma = suma + lista[i]
print(suma)
```

dlaczego to działa?

pytanie: co trzeba tu zmienić, żeby zamiast sumy liczyć iloczyn?

Podstawy programowania

- **oto odpowiedź:**

```
lista = [0, 3, 12, 8, 2]
iloczyn = 1
for i in range(5):
   iloczyn = iloczyn * lista[i]
print(iloczyn)
```

Podstawy programowania

nowość do zapamiętania:

- Python oferuje pewną liczbę tzw. **operatorów skróconych**
- na przykład operacja postaci:

```
suma = suma + lista[i]
```

może zostać zapisana krócej:

```
suma += lista[i]
```

Podstawy programowania

postać ogólna:

- jeżeli symbol **OP** jest operatorem, to wyrażenie:

```
A = A OP B
```

może zostać zapisane jako:

```
A OP= B
```

Podstawy programowania

czyli:

- `A = A + B` → `A += B`
- `A = A - B` → `A -= B`
- `A = A * B` → `A *= B`
- `A = A / B` → `A /= B`
- `A = A // B` → `A //= B`
- `A = A ** B` → `A **= B`

Podstawy programowania

oraz:

- `A = A << B` → `A <<= B`
- `A = A >> B` → `A >>= B`
- `A = A & B` → `A &= B`
- `A = A | B` → `A |= B`
- `A = A ^ B` → `A ^= B`

Podstawy programowania

ale nie mają postaci skróconej operatory poniższe:

- `A = A and B`
- `A = A or B`

Podstawy programowania

pętla **for** – trzecia postać (*):

```
for x in lista:
    kod
```

- zmienna *x* będzie po kolei przybierać wszystkie wartości elementów listy

(*) tak naprawdę ta postać jest równoważna pierwszej

Podstawy programowania

pętla **for** – czwarta postać (*):

```
for x in lista:
    kod1
else:
    kod2
```

- *kod2* wykona się, gdy wyczerpią się wartości z *listy*

(*) tak naprawdę ta postać jest równoważna drugiej

Podstawy programowania

- **sumowanie raz jeszcze**

```
lista = [0, 3, 12, 8, 2]
suma = 0
for el in lista:
   suma += el
print(suma)
```

Podstawy programowania

- **mnożenie raz jeszcze**

```
lista = [0, 3, 12, 8, 2]
iloczyn = 1
for el in lista:
   iloczyn *= el
print(iloczyn)
```

Podstawy programowania

- **zachowanie listy jako argumentu funkcji print()**

```
lista = [0, 3, 12, 8, 2]
print(lista)
```

```
[0, 3, 12, 8, 2]
```

Podstawy programowania

operator **in:**

```
wartość in lista
```

odpowiada na pytanie:
czy wartość **występuje** na liście?
(True - False)

Podstawy programowania

operator **not in:**

```
wartość not in lista
```

odpowiada na pytanie:
czy wartość **nie występuje** na liście?
(True – False)

- dla porównania:

```
not (wartość in lista)
```

Podstawy programowania

- na przykład:

```
lista = [0, 3, 12, 8, 2]
print(5 in lista)
print(5 not in lista)
print(12 in lista)
```

```
False
True
True
```

Podstawy programowania

wycinek listy:

```
lista[ skąd : dokąd ]
```

- tworzy **nową listę** z wycinka **istniejącej** listy
- uwaga – mamy tu tę samą pułapkę, co w funkcji range()
- możemy używać indeksów ujemnych

Podstawy programowania

- na przykład:

```
lista = [10, 8, 6, 4, 2]
nowa_lista = lista[1:3]
print(nowa_lista)
```

```
[8, 6]
```

Podstawy programowania

wynika stąd, że:

```
lista[ skąd : dokąd ]
```

- **skąd** – indeks pierwszego elementu listy, który **wejdzie** do wycinka
- **dokąd** – indeks pierwszego elementu listy, który **nie wejdzie** do wycinka

Podstawy programowania

- można również używać indeksów ujemnych

```
lista = [10, 8, 6, 4, 2]
nowa_lista = lista[1:-1]
print(nowa_lista)
```

```
[8, 6, 4]
```

SIC!

Podstawy programowania

- a nawet indeksów mocno podejrzanych

```
lista = [10, 8, 6, 4, 2]
nowa_lista = lista[-1:1]
print(nowa_lista)
```

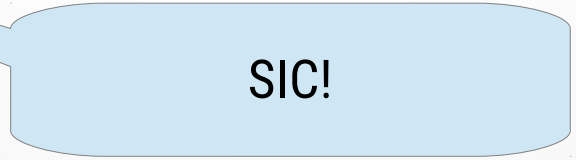
```
[ ]
```

Podstawy programowania

- specjalne przypadki wycinania

```
lista = [10, 8, 6, 4, 2]
nowa_lista = lista[:3]
print(nowa_lista)
```

```
[10, 8, 6]
```



Podstawy programowania

zapamiętaj:

```
lista[ : dokąd ]
```

jest równoważne

```
lista[ 0 : dokąd ]
```

Podstawy programowania

ważkie pytanie:
jak dowiedzieć się, ile elementów ma lista?

Podstawy programowania

funkcja len():

jak length → długość

- ustala długość listy (liczbę jej elementów)
- `len(x)`:
  - argument: pewna lista
  - wynik: długość tej listy
  - efekt: żaden
- np:

```
print(len([ ])) → 0
```

Podstawy programowania

- specjalne przypadki wycinania

```
lista = [10, 8, 6, 4, 2]
nowa_lista = lista[3:]
print(nowa_lista)
```

```
[4, 2]
```

Podstawy programowania

zapamiętaj:

```
lista[ skąd : ]
```

jest równoważne

```
lista[ skąd : len(lista) ]
```

Podstawy programowania

- specjalne przypadki wycinania

```
lista = [10, 8, 6, 4, 2]
nowa_lista = lista[:]
print(nowa_lista)
```

```
[10, 8, 6, 4, 2]
```

Podstawy programowania

zapamiętaj:

```
lista[ : ]
```

jest równoważne

```
lista[ 0 : len(lista) ]
```

Podstawy programowania

zapamiętaj:

```
nowalista = lista[ : ]
```

jest czymś zupełnie innym niż:

```
nowalista = lista
```

Podstawy programowania

- **eksperyment z zaskakującym wynikiem:**

```
lista = [1,3,5]
lista1 = lista[:]
lista2 = lista
lista[1] = 0
print(lista)
print(lista1)
print(lista2)
```

```
[1, 0, 5]
[1, 3, 5]
[1, 0, 5]
```

Podstawy programowania

kolejne ważkie pytanie:
jak usunąć jakiś element z listy?

Podstawy programowania

Dygresja fonetyczna:

$$,,delete“ \rightarrow /dɪ'li:t/$$

Podstawy programowania

instrukcja (nie funkcja!) **del:**

jak delete → usuń

- usuwa element z listy (o ile istnieje)
- `del l[x]`:
  - argument: element *x* listy *l*
  - efekt: usunięcie elementu
- np:

```
del lista[0]
```

Podstawy programowania

- różne przypadki usuwania elementów listy

```
lista = [10, 8, 6, 4, 2]
del lista[1]
print(lista)
```

```
?
```

Podstawy programowania

```
lista = [10, 8, 6, 4, 2]
del lista[1]
print(lista)
```

```
[10, 6, 4, 2]
```

Podstawy programowania

- różne przypadki usuwania elementów listy

```
lista = [10, 8, 6, 4, 2]
del lista[1:3]
print(lista)
```

```
?
```

Podstawy programowania

```
lista = [10, 8, 6, 4, 2]
del lista[1:3]
print(lista)
```

```
[10, 4, 2]
```

Podstawy programowania

- różne przypadki usuwania elementów listy

```
lista = [10, 8, 6, 4, 2]
del lista[:]
print(lista)
```

```
?
```

Podstawy programowania

```
lista = [10, 8, 6, 4, 2]
del lista[:]
print(lista)
```

```
[ ]
```

Podstawy programowania

- jesteśmy blisko niebezpiecznej pułapki!

```
lista = [10, 8, 6, 4, 2]
del lista
print(lista)
```

```
?
```

Podstawy programowania

```
lista = [10, 8, 6, 4, 2]
del lista
print(lista)
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print(lista)
NameError: name 'lista' is not defined
```

zapamiętaj:

```
del lista[:]
```

usuwa **całą zawartość** listy
(usuwa to, co jest w kontenerze)

```
del lista
```

usuwa **listę** jako taką
(usuwa cały kontener)

Podstawy programowania

a teraz nie sposób nie zadać pytania:
jak dodać jakiś element do listy?

sposób pierwszy – wstawić na koniec listy:

ale zanim o tym opowiemy, musimy wprowadzić nowe pojęcie:
metoda

Podstawy programowania

metoda

specyficzny rodzaj **funkcji**, wywoływanej w specyficzny sposób

Podstawy programowania

funkcja vs metoda:

tak wywołujemy funkcję: `funkcja(argument)`

tak wywołujemy metodę: `dana.metoda(argument)`

funkcja → otrzymuje argument i używa go do obliczeń, których wynik zwróci
metoda → otrzymuje argument i używa go do wykonania czynności modyfikujących daną

dalsze szczegóły w ramach przedmiotu „Programowanie obiektowe“

Podstawy programowania

funkcja vs metoda:

- aby wywołać **funkcję**, musimy mieć pewnośc, że taka funkcja **istnieje**
- aby wywołać **metodę**, musimy mieć pewność, że pewna **dana ma taką metodę**

Podstawy programowania

zapamiętaj:

każda lista ma metodę **append()**

```
lista.append(element)
```

- powoduje dodanie elementu element na koniec listy lista

Podstawy programowania

- na przykład:

```
lista = [10, 8, 6, 4, 2]
lista.append(0)
print(lista)
```

```
[10, 8, 6, 4, 2, 0]
```

Podstawy programowania

```
lista = [ ]
for x in range(5):
   lista.append(x * x)
print(lista)
```

```
?
```

Podstawy programowania

- na przykład:

```
lista = [ ]
for x in range(5):
   lista.append(x * x)
print(lista)
```

```
[0, 1, 4, 9, 16]
```

Podstawy programowania

zapamiętaj:

każda lista ma metodę **insert()**
jak insert → wstawiać

```
lista.insert(gdzie,element)
```

- powoduje dodanie elementu element na pozycję gdzie listy lista; pozostałe elementy zostają „rozepchnięte“

Podstawy programowania

- na przykład:

```
lista = [10, 8, 6, 4, 2]
lista.insert(0, 0)
print(lista)
```

```
[0, 10, 8, 6, 4, 2]
```

Podstawy programowania

- a teraz w środek listy:

```
lista = [10, 8, 6, 4, 2]
lista.insert(3, 111)
print(lista)
```

```
[10, 8, 6, 111, 4, 2]
```

Podstawy programowania

- to się również uda:

```
lista = [10, 8, 6, 4, 2]
lista.insert(len(list), 111)
print(lista)
```

```
[10, 8, 6, 4, 2, 111]
```

Podstawy programowania

- i ku wielkiemu zdumieniu, to też działa:

```
lista = [10, 8, 6, 4, 2]
lista.insert(1000, 111)
print(lista)
```

```
[10, 8, 6, 4, 2, 111]
```

Podstawy programowania

- i to też!

```
lista = [10, 8, 6, 4, 2]
lista.insert(-1000, 111)
print(lista)
```

```
[111, 10, 8, 6, 4, 2]
```

Podstawy programowania

- ćwiczymy: jak znaleźć największy element listy?

```
lista = [17,3,11,5,1,9,7,15,13]
#
# ?
#
print(max)
```

Podstawy programowania

- ćwiczymy: jak znaleźć największy element listy?

```
lista = [17,3,11,5,1,9,7,15,13]
max = lista[0]
for i in range(1,len(lista)):
   if lista[i] > max:
      max = lista[i]
print(max)
```

Podstawy programowania

- ćwiczymy: jak sprawdzić, czy pewna wartośc występuje na liście i na której pozycji?

```
lista = [1,2,3,4,5,6,7,8,9,10]
szukany = 5
#
# ?
#
```

Podstawy programowania

- propozycja rozwiązania:

```
lista = [1,2,3,4,5,6,7,8,9,10]
szukany = 5
jest = False
for i in range(len(lista)):
    jest = lista[i] == szukany
    if jest:
        break
if jest:
    print("jest na pozycji ", i)
else:
    print("nie ma")
```

Podstawy programowania

- ćwiczymy: jak **efektywnie** sprawdzić, czy pewna wartośc występuje na liście i na której pozycji?

```
lista = [1,2,3,4,5,6,7,8,9,10]
szukany = 5
#
# ?
#
```

Podstawy programowania

- propozycja rozwiązania:

```
lista = [1,3,5,7,9,11,13,15,17]
szukany = 1
jest = False
lewy = 0
prawy = len(lista) - 1
while lewy <= prawy:
    centralny = (lewy + prawy) // 2
    jest = lista[centralny] == szukany
    if jest:
        break
    if szukany > lista[centralny]:
        lewy = centralny + 1
    else:
        prawy = centralny - 1
if jest:
    print("jest na pozycji ", centralny)
else:
    print("nie ma")
```

funkcja randint() z modułu random

- „losuje“ liczbę całkowitą z podanego przedziału
- `randint(min,max)`:
  - argument: min – najmniejsza oczekiwana liczba
    max – największa oczekiwana liczba
  - wynik: pseudolosowa liczba z podanego przedziału
  - efekt: żaden
- np:
  `print(random.randint(0,10))` → ???

Podstawy programowania

- ćwiczymy: jak wylosować liczby do totolotka?

```
numerki = []
ile = 6
max = 49
#
# ?
#
print(numerki)
```

Podstawy programowania

- propozycja rozwiązania:

```
from random import randint

numerki = [ ]
ile = 6
max = 49
while len(numerki) < ile:
    numerek = randint(1,max)
    if numerek in numerki:
        continue
    numerki.append(numerek)
print(sorted(numerki))
```

Podstawy programowania

- ćwiczymy: jak sprawdzić wyniki totolotka?

```
wylosowano = [5, 11, 9, 42, 3, 49]
nasze_typy = [3, 7, 11, 42, 34, 49]
#
# ?
#
print(ile_trafionych)
```

Podstawy programowania

- propozycja rozwiązania:

```
wylosowano = [5, 11, 9, 42, 3, 49]
nasze_typy = [3, 7, 11, 42, 34, 49]
ile_trafionych = 0

for numerek in nasze_typy:
    if numerek in wylosowano:
        ile_trafionych += 1
print(ile_trafionych)
```

Podstawy programowania

- ćwiczymy: jak posortować listę niemalejąco?

```
lista = [17,3,11,5,1,9,7,15,13]
#
# ?
#
print(lista)
```

Podstawy programowania

- zanim rozwikłamy ten problem, rozwiążmy pewien podproblem – jak zamienić wartości w zmiennych?

```
a = 1
b = 2

# i co dalej?
```

Podstawy programowania

- rozwiązanie klasyczne, jedyne możliwe w wielu językach programowania:

```
a = 1
b = 2

pomocnik = a
a = b
b = pomocnik
```

Podstawy programowania

- ale w Pythonie można to zrobić tak!

```
a = 1
b = 2

a,b = b,a
```

Podstawy programowania

propozycja rozwiązania: „bąbel“ w formie klasycznej

```
lista = [17,3,11,5,1,9,7,15,13]

while True:
    zamiana = False
    for i in range(len(lista) - 1):
        if lista[i] > lista[i + 1]:
            lista[i+1],lista[i] = lista[i],lista[i+1]
            zamiana = True
    if not zamiana:
        break
print(lista)
```

Podstawy programowania

- propozycja rozwiązania: „bąbel“ zmodyfikowany

```
lista = [17,3,11,5,1,9,7,15,13]

for i in range(len(lista)):
    for j in range(i + 1,len(lista)):
        if lista[i] > lista[j]:
            lista[i],lista[j]=lista[j],lista[i]

print(lista)
```