Tablice w Pythonie

```
t =[]
for i in range(100): t.append(0)
t[6] = 23
```

```
t =[]
for i in range(8):
  t.append([])
  for j in range(8): t[i].append(0)
t[6][5] = 23
```

```
t ={}
for i in range(100): t[i] = 0
t[6] = 23
```

```
t = {}
for i in range(8):
  for j in range(8): t[i,j] = 0
t[5,6] = 29
```

Wyrażenia listowe

```
[ expr for x in collection ]
[ expr for x in collection if warunek]

kwadraty = [ i**2 for i in range(100) ]
kwadraty_zakonczone_6 = [ i**2 for i in range(100) if i**2%10==6 ]

osoby = [ ('ola',18), ('ula',16), ('ala',19), … ]
pełnoletni = [os for (os,wiek) in osoby if wiek>=18 ]

lx = [ x for x in range(-10,11) ]
ly = [ f(x) for x in lx ]

la = [ 2,3,5 ]
lb = [ 7,11,13 ]

iloczyn_kartezjanski = [ (a,b) for a in la for b in lb ]

LL = [ [3,5,7], [2,4,8] ]
[ x**2 for L in LL for x in L]
```

Wyrażenia słownikowe

```
{ key : value for (key,value) in collection }

osoby = [ (’ola’,18), (’ula’,16), (’ala’,19) ]

dic1 = {key : value for (key,value) in osoby}
dic2 = {key : value for (value,key) in dic1.items()}



t = ( i**3 for i in range(10) )

for i in t:
   print(i)
```

Tablice w Pythonie

```
t = []
for i in range(100):
  t.append(0)
t[6] = 23

t = [i for i in range(100)]

t = [0]*100
```

```
t = []
for i in range(8):
  t.append([])
  for j in range(8): t[i].append(0)
t[6][5] = 23

w,k = 3,2

q1 = [ [0]*k for _ in range(w) ]
q1[2][1]=7
print(q1) # [[0,0], [0,0], [0,7]]

q2 = [ [0]*k ]*w      !!! źle
q2[2][1]=7
print(q2) # [[0,7], [0,7], [0,7]]
```

Pytania i zadania

Dany jest program:

```
begin
  read(n);
  while n<>rewers(n) do
    n := n+rewers(n)          { dodawanie arytmetyczne }
end.
```

rewers(n) to liczba n zapisana od końca
Czy powyższy program zakończy się dla każdej liczby naturalnej?
Proszę sprawdzić to dla wszystkich liczb n<200.

Pliki

- f = open(*filename* [, *tryb* [, *buffersize*])
  - tryb: "r", "w", "a" ; default "r‟
  - buffersize: 0=unbuffered; 1=line-buffered; buffered
- metody:
  - read([*nbytes*])
  - readline() – pojedyńcza linia
  - readlines() – wszystkie linie jako lista
  - write(*string*)
  - writelines(*list*)
  - seek(*pos*)
  - flush()
  - close()

Pliki - przykład

Przykład:

```
t = {}

f=open("pap.txt","r")
for line in f:
   line=line.strip('\n').lower()
   for z in line:
      t.setdefault(z,0)
      t[z]+=1
   # end for
# end for
f.close()

for k in t:
   print(k,t[k])
```

Pliki - przykład

Można inaczej:

```
from collections import defaultdict

t = defaultdict(int)
with open("pap.txt","r") as f:
  for line in f:
     line=line.strip('\n').lower()
     for z in line:
        t[z]+=1
     # end for
   # end for
# end with

for k in t:
   print(k,t[k])
```

Pliki - przykład

Plik w postaci:
Nowak Jan ; Informatyki ; profesor ; 180

```
from collections import namedtuple

Osoba = namedtuple('Osoba', 'katedra stanowisko pensum')
osoby = {}
with open(„dane.csv","r") as f:
  for line in f:
     li=line.strip('\n').split(';')
     osoby[lista[0]] = Osoba( lista[1], lista[2], lista[3] )
  # end for
# end with

for naz in osoby:
   print( naz, t[naz].katedra, t[naz].stanowisko, t[naz].pensum )
```

Wyjątki

```
try:
   f=open("pap.txt","r")
   try:
      line = f.readline()
      print(line)
   finally:
      f.close()
   # end
except IOError:
   print("Coś jest nie tak!")
# end
```

Przekazywanie parametrów

- Argumenty typów niemodyfikowalnych (np. integer, string, krotka) są przekazywane przez wartość.

- Przykład

- ```
  def cube(x):
      x = x*x*x
      return x
  # end def

  n=2
  w=cube(n)
  print(n,w)        # 2 8
  ```
  ```

Przekazywanie parametrów

- Argumenty typów modyfikowalnych (np. zbiory, listy, słowniki) są przekazywane przez referencję.

- Przykład

- ```
  def zeruj(lista):
      for i in range(len(lista)):
          lista[i] = 0
      return
  # end def

  l=[2,3,5,7]
  zeruj(l)
  print(l)        # [0,0,0,0]
  ```
  ```

Przekazywanie wielu argumentów

```
>>> def f(*args): print(args)
...
> f()
()
> f(1)
(1,)
> f(1, 2, 3, 4)
(1, 2, 3, 4)
```

```
def srednia(*arg):
    suma = 0
    licz = 0
    for el in arg:
        suma += el
        licz += 1
    # end
    return suma/licz
# end
```

Zmienne globalne i lokalne

```
X = 88          # global X

def f():
  X = 99        # local X
# end def

f()
print(X)        # 88
```

```
X = 88          # global X

def f():
  global X
  X = 99        # global X
# end def

f()
print(X)        # 99
```

```
X = 99

def f(Y):
  Z = X + Y     # X is a global
  return Z
# end def

print( f(1) )   # 100
```

```
X = 99

def f(Y):
  global X
  Z = X + Y     # X is a global
  return Z
# end def

print( f(1) )   # 100
```

Zmienne nonlocal

```
def zewn():
    x = "ala"

    def wewn():
        nonlocal x
        x = "ula"
        print("wewn:", x)

    wewn()
    print(„zewn:", x)


zewn()
```

```
wewn: ula
zewn: ula
```

Funkcja lambda

```
def f(x, y):
   return x + y

>>>f(2,3)
>>>5
```

lambda arg1, arg2,... argN : wyrażenie zbudowane z argumentów arg1 .. argN

```
g = lambda x, y: x + y

>>>g(2,3)
>>>5

def row_kw(a,b,c):
  return lambda x : a*x**2+b*x+c

>>> f = row_kw(1,2,3)
>>> f(2)
>>>11
```

Funkcja lambda

```
>>> def make_incrementor (n):
        return lambda x: x + n

>>> f = make_incrementor(2)
>>> g = make_incrementor(6)

>>> print( f(42), g(42) )
44 48

>>> print( make_incrementor(22)(33) )
55
```

Funkcja lambda

```
osoby = [ ('ola',18), ('ula',16), ('ala',19) ]


L1 = sorted(osoby)
print(L1)

def pole(x):
  return x[1]
# end

L2 = sorted(osoby,key=pole)
print(L2)

L3 = sorted(osoby,key=lambda x : x[1])
print(L3)
```

Funkcje: filter, map, reduce

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]

>>> print( filter(lambda x: x % 3 == 0, foo) )
[18, 9, 24, 12, 27]

>>> print( map(lambda x: x * 2 + 10, foo) )
[14, 46, 28, 54, 44, 58, 26, 34, 64]

>>> print( reduce(lambda x, y: x + y, foo) )
139
```

Moduły

Moduły są plikami z rozszerzeniem py, w których zawarto pewien zestaw funkcji. Moduły importujemy do swojego programu za pomocą komendy import.

Plik my_mod.py

```
def suma_cyfr( n ):
   suma = 0
   while n>0:
     suma += n%10
     n /= 10
   # end
   return suma
# end

if __name__ == '__main__':
  if suma_cyfr(123456)==21):
    print(' OK ')
  else:
    print(' Fail ')
# end
```

Plik prog.py

```
form my_mod import suma_cyfr

n = int(input( 'liczba' ))
s = suma_cyft(n)
print('suma cyfr wynosi: ',s)
```

Pakiety

Pakiety to przestrzenie nazw, które zawierają w sobie wiele modułów.
Są po prostu katalogami, ale muszą zawierać specjalny plik nazwany __init__.py

Aby zbudować pakiet, tworzymy katalog o nazwie my_libs, która jest nazwą pakietu, wewnątrz katalogu tworzymy moduł nazwany np. my_mod.

Aby używać modułu my_mod, możemy go zaimportować na dwa sposoby:

```
import my_libs.my_mod
```

albo:

```
from my_libs import my_mod
```