Wykład 4

- Strukturalne typy danych w języku Python
- Procedury i funkcje

Krotki

- Krotka (tuple) analogiczna do listy ale bez możliwości zmian zawartości

Przykłady:

```
k1 = ( )
k2 = ( 'ala', 'ola', 'ula' )

a,b,c = k2
lista = list(k2)
k3 = tuple(lista)

x1,x2 = rk(a,b,c)    # funkcja zwraca krotkę
```

Krotki

Przykład:

```
rm = (-1,0,3,3,6,1,4,6,2,5,0,3,5)
dni = ('niedziela','poniedziałek','wtorek','sroda','czwartek','piatek','sobota')

d  = int(input("dzien 1-31      : "))
m = int(input("miesiąc 1-12   : "))
r  = int(input("rok 1900-2099 : "))

dt = d + rm[m] + (r-1900) + (r-1900)//4
if r%4==0 and m<3: dt=dt-1
dt = dt%7

print(dni[dt])
```

Zbiory

- Nieuporządkowana kolekcja elementów
- Elementy zbioru mogą być różnych typów
- Podstawowe operacje na zbiorach:

```
s = set()              # twórz zbiór pusty
s2 = set([2,3,5])      # twórz zbiór z listy
len(s)                 # moc zbioru
s.add(el)              # wstaw element do zbioru
s.remove(el)           # usuń istniejący element ze zbioru
s.discard(el)          # usuń element ze zbioru
s.clear()              # opróżnij zbiór
if el in s:            # czy należy
if el not in s:        # czy nie należy
s1.union(s2)           # suma mnogościowa zbiorów s1 i s2
s1.intersection(s2)    # część wspólna zbiorów  s1 i s2
```

Słowniki

- Słownik, mapa, tabela, tablica asocjacyjna, tablica z haszowaniem
- Nieuporządkowana kolekcja par klucz/wartość
- Klucz najczęściej jest liczbą lub napisem
- Klucz musi być unikalny
- Wartość może być dowolnego typu

Przykład:

```
t1 = { }
t2 = { 'ala':6, 'ola':12, 'jan':23 }
```

Słownik – funkcje, metody

- h[klucz] – wartość dla klucza
- h[klucz]=wartosc – przypisanie wartości
- del h[klucz] – usuń wartość ze słownika
- h.clear() – usuwa słownik
- len(h) – liczba pozycji w słowniku
- h.get(klucz,default) – wartość dla klucza lub default
- h.keys() – lista kluczy
- h.values() – lista wartości
- h.items() – lista par
- h.copy() – kopia słownika

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)
```

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 )
```

Procedury i funkcje

Cel stosowania:

- dekompozycja problemu
- wielokrotne wykonanie
- poziomy abstrakcji
- oddzielna kompilacja
- możliwość użycia rekurencji

Przekazywanie parametrów

Deklaracja funkcji:
- def func(name) - Argument normalny: dopasowanie po pozycji lub nazwie
- def func(name=value) - Argument domyślny, o ile nie przekazany w wywołaniu
- def func(*name) - Zbiera pozostałe argumenty pozycyjne w krotkę
- def func(**name) - Zbiera pozostałe argumenty nazwane w słownik

Wywołanie funkcji:
- func(value) - argument normalny pozycyjny
- func(name=value) - argument nazwany
- func(*sequence) - przekazuje elementy listy jako kolejne argumenty pozycyjne
- func(**dict) - przekazuje elementy słownika (klucz/wartośd) jako kolejne argumenty nazwane

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
```

Wartości domyślne, parametry nazwane

```
def f(a,b,c=0):
  …

> f(1,2,3)
> f(1,2)


def f(a=1,b=2,c=3):
  …

> f()
> f(b=5)
> f(b=6, c=7, a=8)
```

Przykład:

```
print( x1, x2, end='', sep='' )
```

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
```