Kurs rozszerzony języka Python

Wykład 3.

Marcin Młotkowski

20 października 2017

Plan wykładu

Plan wykładu

Import nazw

```
import random
print(random.randint(1, 10))
```

Wszystkie nazwy musimy poprzedzić nazwą modułu.

Import wybranych funkcji

```
from random import random, randint
print(random.random())
print(random())
```

Import wybranych funkcji

```
from random import random, randint
print(random.random())
print(random())
```

```
from random import *
```

Nazw funkcji nie poprzedzamy nazwą modułu.

Jak stworzyć własny moduł

random.py

```
def random():
    ...

def randint(a, b):
    ...
```

Pakiety

Katalog `pakiet`:

__init__.py
a.py # tu jest funkcja foo()
b.py # tu jest funkcja bar()

```
import pakiet
from pakiet import a
from pakiet.b import bar

a.foo()
bar()
```

Plan wykładu

Przykłady kolekcji

- Listy: [12,3]
- Napisy: "abc", 'def', 'Zażółć gęślą żółtą jaźń'
- Krotki: (1, 'jeden', (1, 2+3j, 0x4))
- Słowniki
- Zbiory

Krotki

```
brown = 165, 42, 42
NavyBlue = (0,0,128)
htmlColor = { 'turquoise' : (64,224,208), 'NavyBlue' : NavyBlue }
r, g, b = htmlColor['NavyBlue']
```

Przypomnienie

podstawienie

```
a, b = 1, 2
```

Przypomnienie

podstawienie

(a, b)= (1, 2)

Kolekcje: operator zawierania

in

```
'bc' in 'abcd'
4 not in [2, 3, 5, 7, 11]
'pi' in { 'pi' : 3.1415, 'e' : 2.7182 }
```

Kolekcje: łączenie kolekcji

+

```
>>> [ 'jeden', 2, 3.0 ] + [ 0x4, 05 ]
['jeden', 2, 3.0, 4, 5]
>>> ('jeden', 2, 3.0) + (0x4, 05)
('jeden', 2, 3.0, 4, 5)
```

Rozmiar kolekcji

len

```
len( ['jeden', 2, 3.0] )
len( {'jeden' : 1, 'dwa' : 2 } )
len( (1, 2, 3) )
```

Odwołania do elementów kolekcji

$[1, 2, 3][2] = 3$

Odwołania do elementów kolekcji

$[1, 2, 3][2] = 3$
$'abcd'[1:3] = 'bc'$

Odwołania do elementów kolekcji

$[1, 2, 3][2] = 3$
$'abcd'[1:3] = 'bc'$
$(1, 2, 3)[1:] = (2, 3)$

Odwołania do elementów kolekcji

$[1, 2, 3][2] = 3$
$'abcd'[1:3] = 'bc'$
$(1, 2, 3)[1:] = (2, 3)$
$(1,2,3)[:1] = (1,)$

Odwołania do elementów kolekcji

$[1, 2, 3][2] = 3$
$'abcd'[1:3] = 'bc'$
$(1, 2, 3)[1:] = (2, 3)$
$(1,2,3)[:1] = (1,)$
$'Python'[:-1] = 'Pytho'$

Odwołania do elementów kolekcji

```
[1, 2, 3][2] = 3
'abcd'[1:3] = 'bc'
(1, 2, 3)[1:] = (2, 3)
(1,2,3)[:1] = (1, )
'Python'[:-1] = 'Pytho'
'Python'[-1:] = 'Python'[-1] = 'n'
```

Slicing

```
>>> 'informatyka'[::3]
'ioak'
```

Przetwarzanie kolekcji — iteratory

```
x = [1,2,3]
y = [4,5,6]
prod = 0
for i in range(len(x)):
        prod += x[i] * y[i]
```

Przetwarzanie list

```
x = [1,2,3]
y = [4,5,6]
prod = 0
for i, v in enumerate(x):
        prod += v * y[i]
print(prod)
```

Przetwarzanie list, inne rozwiązanie

```
x = [1,2,3]
y = [4,5,6]
prod = 0
for a, b in zip(x, y):
        prod += a * b
print(prod)
```

Wariacje nt. słowników

Przetwarzanie słowników

```
dict = { 'uno' : 1, 'duo' : 2, 'tre': 3 }
for key, val in dict.items():
    print(key, "=", val)
```

Wariacje nt. słowników

Przetwarzanie słowników

```
dict = { 'uno' : 1, 'duo' : 2, 'tre': 3 }
for key, val in dict.items():
    print(key, "=", val)
```

Jeszcze inaczej

```
for key in dict.items():
    print(key, "=", dict[key])
```

Plan wykładu

Stałe napisowe

Stałe

'Ala ma kota'
"Ala ma kota"

Stałe napisowe

Stałe

'Ala ma kota'

"Ala ma kota"

Stringi w unicde (Python 2)

u"Zażółć żółtą jaźń"

Stałe napisowe

Stałe

'Ala ma kota'
"Ala ma kota"

Stringi w unicde (Python 2)

u"Zażółć żółtą jaźń"

Unicode (Python 2.*)

len(u"żółty") == 5
len("żółty") == 8

Stałe napisowe

Stałe

'Ala ma kota'
"Ala ma kota"

Stringi w unicde (Python 2)

u"Zażółć żółtą jaźń"

Unicode (Python 2.*)

len(u"żółty") == 5
len("żółty") == 8

Długie napisy

"""To jest
wielolinijkowy string"""

Napisy w Pythonie 3.*

W Pythonie 3.* wszystkie napisy są w UTF-8.

Stringi

- Stringi są kolekcjami
- 'raw' strings: r'abcd\n'
- Kontynuacja napisu:
  "To jest bardzo\n\
  długi tekst\n"
- Mnóstwo funkcji bibliotecznych
- Stringi są niemutowalne, tj. 'abc'[1] = 'd'

Nowe typy "napisowe" w Pythonie 3

Niemutowalne ciągi bajtów

b"byte"
bytes([34,56,50,40])

Nowe typy "napisowe" w Pythonie 3

Niemutowalne ciągi bajtów

b"byte"
bytes([34,56,50,40])

Mutowalne ciągi bajtów

bytearray(b"byte")
bytearray([34,56,50,40])

Formatowanie stringów

Operator % (Python 2.*)

```
print "%i + %i = %i\n"  %  (2, 2, 2+2)

dict = { 'dwa' : 2, 'cztery' : 4 }
print "%(dwa)s + %(dwa)s = %(cztery)s\n" % dict
```

Formatowanie stringów

Operator % (Python 2.*)

```
print "%i + %i = %i\n"  %  (2, 2, 2+2)

dict = { 'dwa' : 2, 'cztery' : 4 }
print "%(dwa)s + %(dwa)s = %(cztery)s\n" % dict
```

```
print("{0} + {1} = {2}\n".format(2,2,2+2))

print ("{dwa} + {dwa} = {cztery}\n".format(dwa=2, cztery=4)
```

Plan wykładu

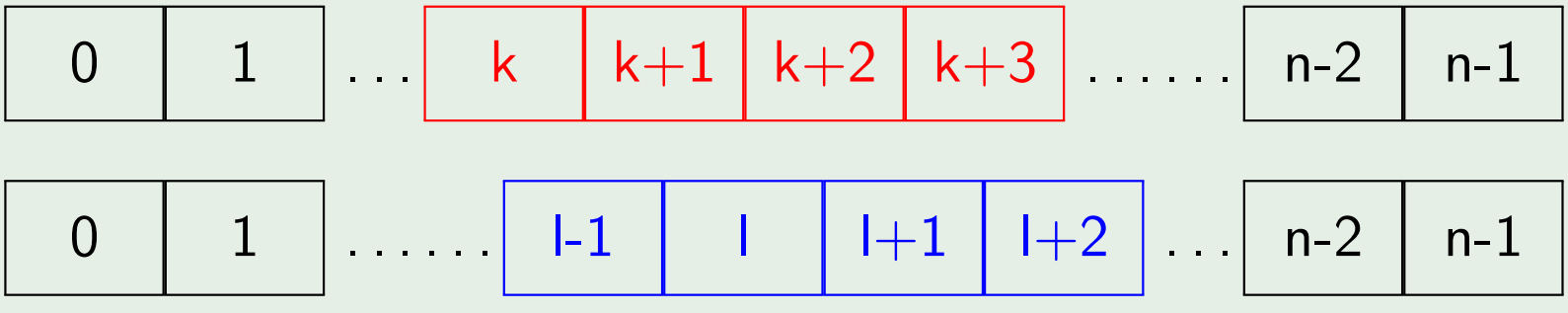
Operacje na listach

Wymiana elementów

```
lista = [1,2,3]
lista[1] = 5          # [1, 5, 3]
lista[1:] = [2,3,4]   # [1,2,3,4]
```

Zamiana podlisty

lista[zakres] = innaLista



Zamiana podlisty

Przykłady

```
lista = [0,1,2,3]
lista[1:3] = ["jeden"]    # [0, 'jeden', 3]
lista[1:1] = [1]          # [0, 1, 'jeden', 3]
lista[2:3] = [2]          # [0, 1, 2, 3]
```

Zamiana podlisty

Przykłady

```
lista = [0,1,2,3]
lista[1:3] = ["jeden"]    # [0, 'jeden', 3]
lista[1:1] = [1]          # [0, 1, 'jeden', 3]
lista[2:3] = [2]          # [0, 1, 2, 3]
```

Przykład ze slicingiem

```
lista = [0, 1, 2, 3]
lista[::2] = [4, 5]
>>> [4, 1, 5, 3]
```

Dodawanie i usuwanie elementów

Przykłady

lista = [0, 1, 2, 3]
lista[len(lista):] = [4, 5, 6]
>>> [0, 1, 2, 3, 4, 5, 6]

Dodawanie i usuwanie elementów

Przykłady

lista = [0, 1, 2, 3]
lista[len(lista):] = [4, 5, 6]
>>> [0, 1, 2, 3, 4, 5, 6]
lista = [0, 1, 2, 3, 4, 5]
lista[4:6] = []
>>> [0, 1, 2, 3]

Instrukcja del

Przykłady

```
lista = [ 'żółty', 'zielony', 'czerwony', 'niebieski' ]
del lista[3]
>>> ['żółty', 'zielony', 'czerwony']
```

Instrukcja del

Przykłady

```
lista = [ 'żółty', 'zielony', 'czerwony', 'niebieski' ]
del lista[3]
>>> ['żółty', 'zielony', 'czerwony']
del lista[1:]
>>> ['żółty' ]
```

del dla słowników

Przykład

```
htmlCol = { 'NavyBlue' : (0,0,128), 'turquoise' : (64,224,208) }
del htmlCol['turquoise']
```

Operacje na listach

Inne operacje

append, extend, insert, remove, pop, index, count, sort, reverse

Operacje na listach

Inne operacje

append, extend, insert, remove, pop, index, count, sort, reverse

Przykłady

```
lista = [0, 1, 2, 3]
lista.reverse() # Nie zwraca wyniku
```

Operacje na listach

Inne operacje

append, extend, insert, remove, pop, index, count, sort, reverse

Przykłady

```
lista = [0, 1, 2, 3]
lista.reverse() # Nie zwraca wyniku
```

Odwracanie listy: zwrócenie wyniku

```
lista = [0, 1, 2, 3]
reversed(lista)[a] # zwraca wynik
```

[a] a właściwie list(reversed(lista))

Plan wykładu

Funkcje

Przykład użycia funkcji

```
def calka(f, a, b):
    krok, suma, x = .1, 0, a
    while x + krok < b:
        suma += f(x)*krok
        x += krok
    return suma

def fun(n): return n * n

print(calka(fun, 0, 5))
```

Funkcje, cd

Inne przykłady

```
def square(n): return n*n

def double(n): return 2 * n

funList = [ square, double ]
for f in funList:
    print(f(10))
```

Lambda funkcje

```
double = lambda x: 2*x
```

Lambda funkcje

```
double = lambda x: 2*x

square = lambda x: x*x
```

Lambda funkcje

```
double = lambda x: 2*x

square = lambda x: x*x

funList = [ double, square ]
print(calka(square, 0, 10))
```

Lambda funkcje, cd

```
funList = [ lambda x: 2*x, lambda x: x*x ]

print(calka(lambda x: x*x, 0, 10))
```

Dwuargumentowe funkcje lambda

```
f = lambda x, y: 2*x + y
```

Operacje na listach

Stałe

lista = range(100)

Operacje na listach

Stałe

```
lista = range(100)

def fun(n): return n % 2 == 0
```

Operacje na listach

Stałe

```
lista = range(100)

def fun(n): return n % 2 == 0

print(filter(fun, lista))
```

Operacje na listach

Stałe

```
lista = range(100)

def fun(n): return n % 2 == 0

print(filter(fun, lista))
print(map(lambda x: 2*x, lista))
```

Operacje na listach

Stałe

```
lista = range(100)

def fun(n): return n % 2 == 0

print(filter(fun, lista))
print(map(lambda x: 2*x, lista))
print(reduce(lambda x, y: x + y, lista, 0))
```

Operacje na listach

Stałe

```
lista = range(100)

def fun(n): return n % 2 == 0

print(filter(fun, lista))
print(map(lambda x: 2*x, lista))
print(reduce(lambda x, y: x + y, lista, 0))
```

W przykładach tych w Pythonie 3 lepiej jest:

```
print(list(filter(fun, lista)))
print(list(map(lambda x: 2*x, lista)))
print(list(reduce(lambda x, y: x + y, lista, 0)))
```

Listy składane

Przykłady

```
lista = range(10)
[ 2 * x for x in lista ]
```

Listy składane

Przykłady

```
lista = range(10)
[ 2 * x for x in lista ]
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Listy składane

Przykłady

```
lista = range(10)
[ 2 * x for x in lista ]
```

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

```
[ (x, x*x*x) for x in lista if x % 3 == 0 ]
```

Listy składane

Przykłady

```
lista = range(10)
[ 2 * x for x in lista ]
```

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

```
[ (x, x*x*x) for x in lista if x % 3 == 0 ]
```

[(0, 0), (3, 27), (6, 216), (9, 729)]

Listy składane, dalsze przykłady

Przetwarzanie list stringów

lista = ["mOnty", "pyTHon's", "FlyinG", "circus"]

Listy składane, dalsze przykłady

Przetwarzanie list stringów

```
lista = [ "mOnty", "pyTHon's", "FlyinG", "circus" ]

lista = [ e[0].upper() + e[1:].lower() for e in lista ]
```

Listy składane zagnieżdżone

Kolejne potęgi dwójki

```
a = [1, 8, 64]
b = [1, 2, 3]
print [ x << y for x in a for y in b]
```

Uzupełnienie — listy

Implementacja list

Wektor wskaźników

Złożoność operacji

Czas dostępu: $O(1)$
Wstawianie/usuwanie elementów:

- na końcu: zamortyzowany czas $O(1)$
- poza tym: $O(n)$

Specjalizowane listy

Na przykład `collections.deque`; wstawianie i usuwanie z obu końców: $O(1)$

Koniec

ev