Politechnika Wrocławska

Python
wstęp do programowania
dla użytkowników WCSS

Dr inż. Krzysztof Berezowski
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

Dane i operacje na danych

SEKWENCJE I PĘTLA FOR

Typy sekwencji

Typ	Reprezentacja
‘str’	Łańcuch znaków: ‘Ala ma kota’, ”Ala ma kota”
‘unicode’	Łańcuch unicode: u’Ala ma kota’, u”Ala ma kota”
‘list’	Modyfikowalna lista: [”Ala”, ”ma”, ”kota”]
‘tuple’	Niemodyfikowalna krotka: (”Ala”, ”ma”, ”kota”)
‘buffer’	Zwracane przez funkcję buffer(). Podobne do łańcuchów ale bez konkatenacji czy powtórzeń.
‘xrange’	Zwracane przez f-cję xrange(). Podobne do listy ale bez podsekwencji (slicing), konkatenacji i powtórzeń. Niektóre operatory są na nieefektywne na obiektach ‘xrange’

Listy

- Lista pusta

```
>>> l0 = []
>>> print l0
[]
>>> print len(l0)
0
```

- Przykłady list

```
>>> l1 = [10, 2, 4, 5]
>>> print l1
[10, 2, 4, 5]
>>> print len(l1)
4
```

Listy

- Lista łańcuchów

```
>>> l2 = [ "Chemiczny", "Elektryczny", "Budownictwa" ]
>>> print l2
['Chemiczny', 'Elektryczny', 'Budownictwa']
>>> print len(l2)
3
```

- Lista list

```
>>> l3 = [l0, l1, l2]
>>> print l3
[[], [10, 2, 4, 5], ['Chemiczny', 'Elektryczny', 'Budownictwa']]
>>> print len(l3)
3
```

Iteracja po indeksach

`range([start, ] stop [, step])`

produkuje listę kolejnych liczb całkowitych:

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

Generacja list (list comprehensions)

Listy mogą być generowane algorytmicznie

```
>>> print [ x*x*x for x in range(8) ]
[0, 1, 8, 27, 64, 125, 216, 343]
```

Np. generacja listy trójek pitagorejskich:

```
>>> print [ (a, b, c ) \
    for a in range(1, 15) \
    for b in range(1, 15) \
    for c in range(1, 15) \
    if a*a + b*b == c*c ]
[(3, 4, 5), (4, 3, 5), (5, 12, 13), (6, 8, 10), (8, 6, 10), (12, 5, 13)]
```

Albo liczb trójkątnych:

```
>>> print [ sum(range(a)) for a in range(2, 11) ]
[1, 3, 6, 10, 15, 21, 28, 36, 45]
```

Zaawansowana generacja list

Comprehensions mogą być zagnieżdżane:

```
>>> print [[r[i] for r in [[0, 1, 2], [3, 4, 5], [6, 7, 8]]] \
           for i in [0, 1, 2]]
[[0, 3, 6], [1, 4, 7], [2, 5, 8]]
```

Co się wydarzyło?

```
m = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

print "------"
for row in m: print row

t = []
for i in [0, 1, 2]:
    t += [[]]            # albo: t.append([])
    for r in m:
        t[i] += [r[i]]   # albo: t.append(r[i])

print "------"
for row in t: print row
```

Problems | Console

<terminated> C:\Users\Krzysztof Berezowski\workspace\lists\src\lists.py

```
------
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
------
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
```

Dane i operacje na danych

ZBIORY I SŁOWNIKI

Zbiory

- Nieuporządkowane kolekcje unikalnych elementów:

```
>>> a = set(['apple', 'orange', 'apple', 'pear', 'orange', 'banana'])
set(['orange', 'pear', 'apple', 'banana'])
>>> b = set(['apple', 'lemon'])
```

- Przykładowe operacje na zbiorach

```
>>> a | b # suma
set(['lemon', 'apple', 'orange', 'pear', 'banana'])
>>> a & b # iloczyn (przekrój)
set(['apple'])
>>> a - b # różnica
set(['orange', 'pear', 'banana'])
>>> a ^ b # różnica symetryczna
set(['orange', 'pear', 'lemon', 'banana'])
```

Zbiory

- Operator `in`:

```
>>> 'orange' in a ^ b
True
>>> 'orange' not in a ^ b
False
>>> not 'orange' in a ^ b
False
```

- Iteracje po zbiorach:

```
>>> for elem in a ^ b:
        print elem
orange
pear
lemon
banana
```

Operacje na zbiorach I

Operator	Interpretacja algebraiczna
`x | y` `x.union(y)`	Suma zbiorów
`x & y` `x.intersection(y)`	Przecięcie (iloczyn) zbiorów
`x - y` `x.difference(y)`	Różnica zbiorów
`x ^ y` `x.symmetric_difference(y)`	Różnica symetryczna zbiorów
`e in x`	Testowanie czy element e jest w zbiorze x
`e not in x` `not e in x`	przeciwnie
`x <= y` `x.issubset(y)`	Testowanie czy x jest podzbiorem y
`x >= y`	Testowanie czy x jest nadzbiorem y

Operacje na zbiorach II

Operator	Interpretacja algebraiczna
`x |= y` `x.union_update(y)`	Suma zbiorów w miejscu
`x &= y` `x.intersection_update(y)`	Przecięcie (iloczyn) zbiorów w miejscu
`x -= y` `x.difference_update(y)`	Różnica zbiorów w miejscu
`x ^= y` `x.symmetric_difference_update(y)`	Różnica symetryczna zbiorów w miejscu

Operacje na zbiorach III

Operator	Interpretacja algebraiczna
s.add(x) s |= set([x])	Dodanie elementu do zbioru
s.remove(x)	Usunięcie elementu ze zbioru; KeyError jeżeli element nie istnieje
s.discard(x) s - set([x])	Usunięcie elementu ze zbioru jeżeli istnieje
s.pop()	Pobranie (jakiegoś) elementu ze zbioru; KeyError jeśli zbiór jest pusty
s.clear()	Wyczyszczenie zbioru

Zbiory - uwagi końcowe

- Konstrukcja:

```
>>> s = set([0, 4, 8])
>>> f = frozenset([0, 4, 8])

>>> s = {0, 4, 8}      # ta składnia istnieje od pythona 2.7
```

- Uwaga na różnicę między `in` a `<=` lub `<`

```
>>> s1 = set([0, 4, 8])
>>> s2 = set([0, 4])
>>> s2 in s1            # dlaczego? Bo testuje czy zbiór s2 jest elementem s1
False                   # (czym nota bene być nie może)
>>> s2 <= s1
True
>>> s2 < s1
True
```

Słowniki

- Nieuporządkowane kolekcje par *unikalny klucz → wartość*:

```
# skladnia
print {'nazwisko':'Kowalski', 'imie':'Wisniewski'}

# z listy list
print dict([[0, 'Polska'], [1, 'Francja'], [2, 'Dania']])

# z zipa
print dict(zip(range(5), ['Warszawa', 'Poznan', 'Krakow','Gdansk', 'Wroclaw']))

# z list comprehension
print dict([ [x, y] for x in range(10) for y in range(10) if y == x*3 ])
```

Problems | Console

```
<terminated> C:\Users\Krzysztof Berezowski\workspace\looping\src\dicts.py
{'nazwisko': 'Kowalski', 'imie': 'Wisniewski'}
{0: 'Polska', 1: 'Francja', 2: 'Dania'}
{0: 'Warszawa', 1: 'Poznan', 2: 'Krakow', 3: 'Gdansk', 4: 'Wroclaw'}
{0: 0, 1: 3, 2: 6, 3: 9}
```

Operacje na słownikach

Operator	Interpretacja algebraiczna
`d[key]`	Pobranie wartości przypisanej kluczowi. *KeyError* jeśli nie istnieje.
`d[key] = val`	Ustawienie wartości
`del d[key]`	Skasowanie wpisu (całego)
`key in d`	Testowanie czy słownik zawiera klucz
`key not in d` `not key in d`	Testowanie czy słownik nie zawiera klucza
`d.fromkeys(seq[, val])`	Inicjalizacja słownika z sekwencji
`d.get(key)`	Pobranie wartości przypisanej kluczowi. None jeśli nie istnieje.
`d.has_key(key)`	Testowanie czy klucz jest w słowniku.
`d.keys(), d. values(), d.items()`	Lista kluczy, lista wartości, lista par klucz-wartość.

Właściwe dla pythona techniki iteracji

```
d = {0:"zero", 1:"jeden", 2:"dwa", 3:"trzy", 4:"cztery", \
     5:"piec", 6:"szesc", 7:"siedem", 8:"osiem", 9:"dziewiec"}
# iterowanie po krotkach
for k, v in d.items():
    print k, v

# lista list
l = [ [x, x^2, x^3] for x in range(10) ]
print l

# rozpakowanie listy
for v1, v2, v3 in [ [x, x^2, x^3] for x in range(10) ]:
    print v1, v2, v3

# "zipowanie" list
for v in zip(range(10), reversed(range(10))):
    print v

# porzadkowanie list nieuporzadkowanych
for k1, k2 in zip(sorted(d.keys()), reversed(sorted(d.keys()))):
    print k1, k2
```