Python – podstawy

Maciej Chmiel
Maciej Ogrodniczek

Wbudowane typy danych

- LICZBY

  - Int (liczby całkowite)
  - Float (liczby zmiennoprzecinkowe podwójnej precyzji)
  - Long (tylko w Pythonie 2.x)
  - Complex (liczby zespolone)

  DZIAŁANIA: dodawanie, odejmowanie, mnożenie, dzielenie …

Podstawowe operacje na liczbach

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x // y`	(floored) quotient of *x* and *y*
`x % y`	remainder of `x / y`
`-x`	*x* negated
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`long(x)`	*x* converted to long integer
`float(x)`	*x* converted to floating point
`complex(re,im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.
`c.conjugate()`	conjugate of the complex number *c*. (Identity on real numbers)
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

Wbudowane typy danych

- SEKWENCJE
  - String (napisy, 8 bit/znak w Python 2.x, Unicode w Python 3.x)
  - Byte (tylko w Python 3.x)
  - List (lista)
  - Tuple (krotka)
- ZBIORY
  - Set (matematyczny zbiór)
- MAPY
  - Dict (słownik, mapa)

Napisy – najczęściej używane metody

- METODY

```
split(s) – podział względem s, zwraca listę
replace(old, new)- zastępuje stary podciąg nowym
strip() – usuwa końcowe białe znaki
```

- FORMATOWANIE – operator {} i metoda format

```
‘{0} jest {1}’.format(‘Python’, ‘super’)
‘{jezyk} jest {cecha}’.format(jezyk=‘Python’, cecha=‘super’)
```

Operacje na napisach

```
>>> word = „Hello Python”
>>> len(word)
>>> word[0] # H
>>> word[1] # e
>>> word[2] # l
>>> word[3] # l
>>> word[4] # o
>>> word[-1] # n
>>> word[-2] # o
```

```
>>> word[0:5]
>>> word[:5]
>>> word[6:]
>>> word[-6:]
>>> word[:-7]
>>> word[0:10000]
>>> word[100:]
>>> word[-0]
>>> word[:]
>>> word[::-1]
```

Listy

Czym są listy?

- Lista jest tablicą, kontenerem, …
- Przechowywanie różnych typów danych
- Kolejność ma znaczenie

Listy

Tworzenie listy

```
L = []
L = [1, ‘napis’, obiekt1, obiekt2, inna_lista]
L = list()
L = list(sequence)
```

Ta sama lista pod różnymi nazwami

```
A = B = []
A = []
B = A
```

niezależne listy

```
A = []; B = []
```

Listy

Dostęp do danych - indeksowanie

L = [‘a’, ‘b’, ‘c’, ‘d’]

L[0] to ‘a’ L[1] to ‘b’ L[2] to ‘c’ L[3] to ‘d’ L[-1] to ‘d’

L[start : stop : step]

L[0:2] to [‘a’, ‘b’] L[:2] to [‘a’, ‘b’]

Listy

Operacje na listach

list.**append**(x)
list.**extend**(L)
list.**insert**(i, x)
list.**remove**(x) lub del L[index]
list.**index**(x)
list.**count**(x)
list.**reverse**()
list.**pop**([i])

list.**sort**() lub sorted(L)

X **in** L (sprawdzenie czy X jest na liście L)

Tuple (krotka)

- Niezmienna lista
- Krotki są niemodyfikowalna
  - Zawartość ustalana podczas tworzenia krotki – nie można jej później zmienić

- Operacje szybsze niż na liście
- Dane tylko do odczytu

Tuple (krotka)

Tworzenie krotki

```
empty = ()
singleton = 'hello',

t = 12345, obj, 'hello!‘

u = t, (1, 2, 3, 4, 5)

t = (‘a’, ‘b’, ‘c’)

tuple('abc')

('a', 'b', 'c')
```

Tuple (krotka)

Operacje na krotkach

krotka.**count**(x) - zliczanie wystąpień X
krotka.**index**(x) - indeks X

I to wszystko …

Słownik

Czym jest słownik (mapa)

- Mapa (tablica asocjacyjna, słownik haszujący) typu klucz - wartość
- Modyfikowalny

- Nieuporządkowany zbiór danych!!!
- Unikatowe (niepowtarzalne) klucze

- Klucz musi być niemodyfikowalny (liczba, napis, krotka)
- Wartością może być dowolny obiekt

- Bardzo szybki dostęp do danych

Słownik

Tworzenie słownika

D = {klucz: wartość, … }

D = dict()
D = dict(one=1, two=2, three=3)

D = {[(k1, v1), (k2, v2), (k3, v3)]}

Słownik

Najważniejsze metody

dict.**get**(*key*, [*default*])
dict.**has_key**(*key*)
dict.**items**()
dict.**keys**()
dict.**values**()
dict.**update**([*other*])

Zbiory

Czym są zbiory?

- W ujęciu matematycznym: zestaw elementów
- Wsparcie dla operacji matematycznych
  - suma, przecięcie, różnica, iloczyn, selekcja, …

- Podobnie jak słownik:
  - implementacja mapy haszującej
  - nieuporządkowany zbiór danych
  - unikatowe (niepowtarzalne) elementy

Zbiory

Tworzenie zbioru

```
s = Set()
s = Set(L)

a = set('abracadabra')
set(['a', 'r', 'b', 'c', 'd'])
```

Zbiory

Najważniejsze metody

set.**add**(elem)
set.**remove**(elem)
set.**discard**(elem)
set.**pop**()
set.**clear**()
set.**union**(other, ...)
    set | other | …
set.**intersection**(other, ...)
    set & other &
set.**difference**(other, ...)
    set - other - …
set. **symmetric_difference**(other)
    set ^ other

Prawda i Fałsz

- Prawda – 1
- Fałsz – 0, None, [], {}

- AND (zwraca pierwszą fałszywą wartość)

```
'a' and 'b'     # 'b'
[]  and 'b'     # []
```

- OR (zwraca pierwszą prawdziwą wartość)

```
'a' or 'b'     # 'a'
[]  or'b'      # 'b'
```

Kontrola przepływu danych

IF

```
if <statement>:
    <do something>
elif:
    <do something else>
else:
    <do something different>
```

Kontrola przepływu danych

FOR

```
for <item> in <sequence>:
    <do something>
    <break or continue>
```

```
for <item> in <sequence>:
    <do something>
    <break or continue>
else:
   <do something if no break>
```

Kontrola przepływu danych

WHILE

```
while <statement>:
    <do something>
    <break>
```

Funkcje

Tworzenie własnych funkcji

- Deklaracja funkcji

```
def function_name(parameters):
    <do something>
    <return>
```

- Wywołanie funkcji

```
function_name(arguments)
```

Funkcje

Przykład

```
def add(a, b):
    return a + b

X = 3
Y = 2

print add(X, Y)
```

Funkcje

Parametry

- Normalne (mają nazwę i pozycję)
- Nazwane (mają nazwę)

```
add(a, b=10)
```

- Zmienne (mają *, mają pozycję)

```
add(a, b=10, *args)
```

- Zmienne nazwane (mają **, mają nazwę, dowolna liczba argumentów)

```
add(a, b=10, **kwargs)
```

Obiektowość/Klasy

- Wszystko jest obiektem
- Dynamiczne typowanie (obiekty posiadają typ)
- Każdy obiekt posiada
  - Tożsamość (lokalizacja obiektu w pamięci)
  - Typ (reprezentacja obiektu)
  - Wartość (dane przechowywane w obiekcie)

- Klasy również są obiektami

```
class Klasa(object): pass
```

Metody i atrybuty klasy

```
class Adder(object):
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2

    def result(self):
        return self.param1 + self.param2


adder_object = Adder(2, 3)
print adder_object.result()
```