Python – Více ze syntaxe

Objekty, funkce, generátory, iterátory, dekorátory, …

Michal Čihař

mcihar@suse.cz

Seznam

Uspořádané pole libovolných objektů

Práce se seznamem

- Definice

```
>>> seznam = [1, 2, 4]
```

- Přístup

```
>>> seznam[0]
1
>>> seznam[-1]
4
```

- Procházení

```
>>> for x in seznam:
...     print(x)
```

Vyřezávání seznamu

- Vyříznutí část seznamu

```
>>> seznam = [0, 3, 6, 9, 12]
>>> seznam[1:3]
[3, 6]
>>> seznam[:3]
[0, 3, 6]
>>> seznam[-2:]
[9, 12]
```

Přidávání položek

- Spojení seznamů

```
>>> seznam + [42, 43]
```

- Sloučení seznamů

```
>>> seznam.extend([52, 53])
```

- Na konec

```
>>> seznam.append(666)
```

- Na pozici

```
>>> seznam.insert(0, 666)
```

Vyhledávání v seznamu

- Spočítat počet

```
>>> seznam.count(666)
2
```

- Test existence

```
>>> 666 in seznam
True
```

- Index

```
>>> seznam.index(666)
0
>>> seznam.index(777)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 666 is not in list
```

Odstraňování položek

- Pozice

```
>>> seznam = [0, 3, 6, 9, 12]
>>> del seznam[1]
```

- Prvek

```
>>> seznam.remove(9)
```

- Zásobník

```
>>> seznam.pop()
12
```

Rozsahy

- Seznam čísel

```
>>> range(20)
```

- Rozsah čísel

```
>>> range(10, 20)
```

- Krok

```
>>> range(10, 20, 2)
```

- Záporný krok

```
>>> range(20, 10, -2)
```

N-tice

- Seznam jen pro čtení
- Podobné operace
- S neměnnými prvky lze použít jako klíč slovníku

N-tice a více hodnot

- Návrácení více hodnot z funkce

```
def vicenavrat():
    return (1, 2)
```

- Přiřazení více hodnot najednou

```
>>> (a, b) = vicenavrat()
```

- Kombinace z rozsahy

```
>>> (ONE, TWO, THREE) = range(3)
```

Množina

Neuspořádaná kolekce jedinečných hodnot

Vytvoření množiny

- Z prvků (Python 2.7 a 3.x)

```
>>> mnozina = {1, 2, 3}
```

- Ze seznamu nebo n-tice

```
>>> mnozina = set(seznam)
>>> mnozina = set([1, 2, 3])
```

- Prázdná množina

```
>>> mnozina = set()
```

Manipulace s prvky množiny

- Jednotlivé prvky

```
>>> mnozina.add(4)
```

- Další množiny

```
>>> mnozina.update({1, 5, 6})
>>> mnozina.update({7, 8}, {8, 9, 10})
```

- Seznamy nebo n-tice

```
>>> mnozina.update([1, 3, 5])
```

- Vyčištění

```
>>> mnozina.clear()
```

Odstranění prvků z množiny

- Bez ohledu na existenci prvku

```
>>> mnozina.discard(4)
```

- S kontrolou existence

```
>>> mnozina.remove(4)
>>> mnozina.remove(66)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 66
```

- Zásobník (jako seznamy, jen náhdoný prvek)

```
>>> mnozina.pop()
```

Množinové operace

```
>>> a_set = {2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195}
>>> b_set = {1, 2, 3, 5, 6, 8, 9, 12, 15, 17, 18, 21}
>>> a_set.union(b_set)
{1, 2, 195, 4, 5, 6, 8, 12, 76, 15, 17, 18, 3, 21, 30,
51, 9, 127}
>>> a_set.intersection(b_set)
{9, 2, 12, 5, 21}
>>> a_set.difference(b_set)
{195, 4, 76, 51, 30, 127}
>>> a_set.symmetric_difference(b_set)
{1, 3, 4, 6, 8, 76, 15, 17, 18, 195, 127, 30, 51}
```

Slovník

Neuspořádaná kolekce dvojic klíč-hodnota

Základní práce se slovníkem

- Vytvoření

```
>>> slovnik = {'jedna': 1, 'dva': 2}
```

- Přístup k prvkům

```
>>> slovnik['dva']
2
```

- Neexistující prvek

```
>>> slovnik['nic']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nic'
```

Úpravy slovníku

- Změna prvku

```
>>> slovnik['jedna'] = 42
```

- Přidání prvku

```
>>> slovnik['test'] = 42
```

- Klíče můžou být různé

```
>>> slovnik[(1, 2)] = 'text'
```

- Hodnota může být libovolná

```
>>> slovnik['test'] = [1, 2, 3]
```

Procházení slovníku

- Test existence klíče

```
>>> 'jedna' in slovnik
```

- Procházení po klíčích

```
>>> for klic in slovnik:
...     print(slovnik[klic])
```

Generátorová notace

Aneb jak jinak definovat seznamy nebo slovníky

Generátorová notace seznamu

- Pokud chceme prvky pole vypočítat

```
>>> [2 ** x for x in range(5)]
[1, 2, 4, 8, 16]
```

- Můžeme také filtrovat

```
>>> [2 ** x for x in range(10) if x % 2 == 0]
[1, 4, 16, 64, 256]
```

- Výběr souborů

```
>>> [f for f in glob.glob('*.py')
      if os.stat(f).st_size > 6000]
```

Generátorová notace slovníku

- Dostupné od Pythonu 2.7 a 3.x
- Podobně jako seznamy

```
>>> {f: os.stat(f) for f in glob.glob('*test*.py')}
```

- Můžeme také filtrovat

```
>>> {f: os.stat(f)
        for f in glob.glob('*test*.py')
        if len(f) > 5}
```

Generátorová notace množin

- Dostupné od Pythonu 2.7 a 3.x
- Opět podobně jako seznamy

```
>>> {x ** 2 for x in range(10)}
set([0, 1, 4, 81, 64, 9, 16, 49, 25, 36])
```

- Můžeme také filtrovat

```
>>> {x ** 2 for x in range(10) if x % 2 == 0}
set([0, 16, 4, 64, 36])
```

Generátory

Generování seznamů

Generování hodnot za běhu

- Generátorová notace vytvoří celý seznam najednou
- Co když ho vůbec nepotřebujeme alokovat?
- Generátor je funkce a příkazem yield vrací hodnoty

Fibonacci

- Typický příklad generátoru

```
def fib(max):
    a, b = 0, 1
    while a < max:
        yield a
        a, b = b, a + b

>>> fib(100)
<generator object fib at 0x7fb605ddaa50>

>>> for x in fib(10):
...     print(x)
```

Třídy

Trochu hlouběji do anatomie tříd

Třídy v Pythonu

- Základní stavební prvek
  - Všechno v Pythonu jsou objekty
- Mnohdy neviditelný
  - Čísla jsou objekty
  - Funkce jsou objekty

Staré a nové třídy

- V Pythonu 2.x jsou dva druhy tříd
  - Ať žije zpětná kompatibilita
- Staré třídy se občas chovají divně
  - Třeba nefunguje vícenásobná dědičnost
  - Jiný způsob definování atributů

```
class NovaTrida(object):
    pass

class StaraTrida:
    pass
```

Dědičnost

- Explicitní volání
  - Pokud používáme staré třídy
  - Parent.method(self, ...)
- Použití funkce super()
  - Pokud možno vždy
  - V Pythonu 3 je možné vynechat parametry

```
class C(B):
    def method(self, arg):
        super(C, self).method(arg)
```

Speciální rozhraní

- Třída může implemetovat různá standardní rozhraní
- Používají se speciální metody
  - Můžeme tímto způsobem implementovat jakýkoliv typ
  - Jmenují se __něco__
  - Občas se používá název dunder (Double UNDERscore)
  - Např. __call__, __init__, __unicode__, __iter__, …

Konstruktor

- Inicializace třídy

```
class Trida(object):
    def __init__(self, param):
        print('init')
```

Před konstruktorem

- Konstruktor jen nastavuje vlastnosti objektu
- Můžeme však ovlivnit i vytváření objektu
  - Metoda __new__
  - Obvykle jsou lepší cesty jak dosáhnout toho samého

```
class A(object):
    def __new__(cls, *args, **kwargs):
        new_instance = object.__new__(
            cls, *args, **kwargs
        )
        return new_instance
```

Atributy

- Definice v třídě

```
class A(object):
    param = 'value'
```

- Definice v konstruktoru

```
class A(object):
    def __init__(self):
        self.param = 'value'
```

- Definice za běhu
  - Možné definovat parametr kdykoliv
  - Nedoporučované, protože znepřehledňuje kód

Vlastní atributy 1

- Můžeme definovat vlastní chování atributů
  - Manipulace metodami __getattr__, __setattr__ a __delattr__
  - Zjištění seznamu metodou __dir__

```
class Attr(object):
   def __getattr__(self, name):
       if name == 'special':
            return '666'
       raise AttributeError()
```

Vlastní atributy 2

- Jak se předchozí případ chová s reálnými atributy?

```
>>> a = Attr()
>>> a.normal = 'text'
>>> a.normal
'text'
>>> a.special
'666'
>>> a.special = 42
>>> a.special
42
```

Vlastní atributy 3

- Můžeme však změnit i toto
  - Metoda __getattribute__
  - Použije se pro všechny atributy (kromě speciálních metod)

```
class Attr(object):
   param = 'value'
   def __getattribute__(self, name):
       raise AttributeError()

>>> a = Attr().param
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in __getattribute__
AttributeError
```

Funkce

Jak volat, co volat

Parametry funkce 1

- Pozicí

```
def x(foo, bar):
    print('%s %s' % (foo, bar))

>>> x(1, 2)
1 2
```

- Klíčovým slovem

```
def x(foo, bar):
    print('%s %s' % (foo, bar))

>>> x(bar=1, foo=2)
2 1
```

Parametry funkce 2

- Výchozí hodnota

```
def x(foo, bar='nic'):
    print('%s %s' % (foo, bar))

>>> x(3)
3 nic
>>> x(foo='text')
text nic
```

Parametry funkce 3

- Zpracování libovolných parametrů
  - Do proměnnné s * se přiřadí poziční parametry
  - Do proměnnné s ** se přiřadí klíčové parametry

```
def x(foo, bar=1, *args, **kwargs):
    print(repr(args))
    print(repr(kwargs))

>>> x(1, 2, 3)
(3,)
{}
>>> x(1, 2, test=3)
()
{'test': 3}
```

Návratová hodnota

- Volitelný příkaz return
  - Pokud se nepoužije, funkce vrací None
  - Příkazová řádka návratovou hodnotu None nevypisuje

```
def x():
    return 1

def y():
    pass

>>> x()
1
>>> y()
>>>
```

Třídy jako jiné typy

Co ještě může třída předstírat

Množina

- Podpora operátoru in
  - Pomocí metody __contains__
- Podpora funkce len()
  - Pomocí metody __len__

Množina - příklad

- Objekt který obsahuje 42

```
class Fixed(object):
    def __contains__(self, val):
        return val == 42
    def __len__(self):
        return 1

>>> fixed = Fixed()
>>> len(fixed)
1
>>> 42 in fixed
True
>>> 666 in fixed
False
```

Slovník nebo pole

- Získání hodnoty x[key]
  - Metoda __getitem__
- Nastavení hodnoty x[key] = value
  - Metoda __setitem__
- Smazání hodnoty del x[key]
  - Metoda __delitem__
- Výchozí hodnota pro neznámé klíče
  - Metoda __missing__

Slovník - příklad

- Slovník který na klíči test vždy obsahuje 42

```
class FixedDict(dict):
    def __getitem__(self, key):
        if key == 'test':
            return 42
        return super().__getitem__(key)

>>> fixed = FixedDict()
>>> fixed['test']
42
>>> fixed['test'] = 22
>>> fixed['test']
42
>>> fixed['new'] = 1
>>> fixed['new']
1
```

Číselné operace

- Můžeme definovat chování pro všechny operátory
  - U binárních operátorů můžeme upravit chování pro obě strany
    - __add__, __mul__, … / __radd__, __rmul__, …
  - Unární operátory
    - __neg__, __abs__, __invert__, …
  - Operace nad proměnnou (x += 1)
    - __iadd__, __imul__, …

Převody typů

- Podobně fungují převody typů
  - Na řetězec
    - __str__, __bytes__ (Python 3) / __str__, __unicode__ (Python 2)
  - Na číslo
    - __int__, __float__, __complex__
  - Na pravdivostní hodnotu
    - __bool__

Porovnávání

- Pro objekty můžeme nadefinovat i porovnávání
  - __eq__, __lt__, __gt__, __le__, __ge__

Třída jako iterátor

- Opět stačí naimplementovat metody
  - __iter__ se volá na začátku, vrací iterátor
  - __next__ vrací další hodnoty
  - ukončí se výjimkou StopIteration

Třída jako iterátor - příklad

- Opět Fibonacci

```
class Fib(object):
    def __init__(self, max):
        self.max = max

    def __iter__(self):
        self.a = 0
        self.b = 1
        return self

    def __next__(self):
        fib = self.a
        if fib > self.max:
            raise StopIteration
        self.a, self.b = self.b, self.a + self.b
        return fib
```

Třída jako kontext

- Operační kontext se definuje pomocí bloku with
- Snadná práce s objekty vyžadující úklid
  - Soubory (automatické uzavření)
  - Zámky (odemčení)
  - Přesměrování výstupu
- Implementace ve třídě
  - Metody __enter__ a __exit__

Třída jako kontext - příklad

- Přesměrování výstupu

```
class Redirect(object):
    def __init__(self, out_new):
        self.out_new = out_new

    def __enter__(self):
        self.out_old = sys.stdout
        sys.stdout = self.out_new

    def __exit__(self, *args):
        sys.stdout = self.out_old

>>> with open('log', 'w') as soubor, Redirect(soubor):
...    print('B')
```

Dokumentace

Každý kód si zaslouží dokumentaci

Dokumentační řetězce

- Docstring

```
'''
Dokumentace modulu.
'''

class Foo(object):
    '''
    Dokumentace tridy.
    '''

    def bar(self):
        '''
        Dokumentace metody.
        '''
```

Dokumentace

- Dokumentace je dostupná v atributu __doc__
- Takto jí také získává příkaz help()
- Existují nástroje na generování dokumentace

```
>>> print(Foo.__doc__)

    Dokumentace tridy.
```

Komentáře

- Když už mluvíme o dokumentaci

```
# Vypise HELLO
print('HELLO')
```

Další zdroje informací

Dokumentace

- Oficiální dokumentace
  - Popis jazyka, tutorial, popis standardní knihovny, API
  - http://docs.python.org/2/
  - http://docs.python.org/3/
- Knihy
  - Dive into Python 3 / Ponořme se do Pythonu 3
  - Expert Python Programming
- Vestavěná nápověda v příkazové řádce
- Prezentaci naleznete na
  - http://cs.cihar.com/talks/2013/python/

Nějaké dotazy?

Thank you.

Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org