# Vestavěné nástroje Pythonu

Petr Pošík

Katedra kybernetiky, FEL ČVUT v Praze

OI, B4B33RPH - Řešení problémů a hry, 2016

## Credits

David Beazley (http://www.dabeaz.com/): **Builtin Superheros!**

PyData Chicago, 2016

https://www.youtube.com/watch?v=j6VSAsKAj98 (https://www.youtube.com/watch?v=j6VSAsKAj98)

> "For me, Python has always been a personal productivity tool. It solves the messy day to day problems that surround the actual problem you are trying to solve.
>
> Example: Data Munging (90/10 rule)"

## Tajné zbraně Pythonu

Vestavěné nástroje:

- datové typy `tuple`, `list`, `set`, `dict`
- modul `collections`
- různé vestavěné operace

Klíčové pro každého Python profesionála!

### Jsou všude!

Vestavěné nástroje jsou vždy dostupné:

- ve všech verzích Pythonu
- na všech operačních systémech
- ve všech distribucích Pythonu

Kromě Pythonu nepotřebujete nic instalovat!

### Jsou univerzální!

Dynamické typování je zde velmi užitečné!

- Datové struktury můžete vytvářet prakticky z čehokoliv.
- A nemusíte o tom téměř přemýšlet.
- Vše prostě funguje.

### Jsou rychlé!

Vestavěné datové typy jsou rychlé - z hlediska použití.

- Paměť je levná.
- Procesory jsou levné.
- Můj (a váš) čas je drahý!

Vestavěné nástroje poskytují základní vybavení pro rychlé zkoumání nápadů, konstrukci prototypů a řešení neobvyklých problémů. Vždy můžete své řešení později optimalizovat (pokud to je třeba).

### Jsou zábavné!

S vestavěnými nástroji Pythonu je radost pracovat!

- Odstiňují vás od otravných detailů (např. od správy paměti).
- Můžete se soustředit na to, co potřebujete vyřešit.
- Nicméně je třeba vědět, jaký nástroj kdy použít.

Je to vše o vaší osobní produktivitě.

## Vestavěné kolekce (kontajnery)

### tuple (n-tice)

Také známý jako:

> záznam (record); struktura

Typické použití:

> přiřazení více hodnot najednou; k reprezentaci řádku v databázi

Základní použití:

```
>>> record = (val1, val2, val3)
>>> a, b, c = record
>>> val = record[n]
```

Příbuzní:

> collections.namedtuple

In [1]:

```
row = ('Jan', 'Novak', 'Sokolovska 27', 'Praha')
row[2]
```

Out[1]:

```
'Sokolovska 27'
```

Není příliš čitelné mít v kódu natvrdo uvedeny všechny tyto indexy. Lepší je použít `namedtuple`, který nám umožní odkazovat se na jednotlivé prvky i jménem (a přitom se jinak chová stejně jako `tuple`).

In [2]:

```
from collections import namedtuple
Person = namedtuple('Person', 'fname lname street city')
row = Person('Jan', 'Novak', 'Sokolovska 27', 'Praha')
row.fname
```

Out[2]:

```
'Jan'
```

In [3]:

```
row.street
```

Out[3]:

```
'Sokolovska 27'
```

# list (seznam)

Také známý jako:

> proměnlivá (mutable) posloupnost, pole (array)

Typické použití:

> vynucuje pořadí

Základní použití:

```
>>> items = [val1, val2, ..., valn]
>>> x = items[n]
>>> items[n] = x
>>> del items[n]
>>> items.append(value)
>>> items.sort()
```

## set (množina)

Typické použití:

> jedinečnost, testy existence prvku v kolekci

Základní použití:

```
>>> s = {val1, val2, ..., valn}
>>> s.add(val)
>>> s.remove(val)
>>> val in s
```

In [4]:

```
names = ['Jan', 'Adam', 'Eva', 'Jan']
names
```

Out[4]:

```
['Jan', 'Adam', 'Eva', 'Jan']
```

In [5]:

```
names = {'Jan', 'Adam', 'Eva', 'Jan'}
names
```

Out[5]:

```
{'Adam', 'Eva', 'Jan'}
```

## dict (slovník)

Také známý jako:

> mapování, asociativní pole

Typické použití:

> vyhledávací tabulky; rejstříky

Základní použití:

```
>>> d = {key1: val1, key2: val2, key3: val3}
>>> val = d[key]
>>> d[key] = val
>>> del d[key]
>>> key in d
```

In [6]:

```
prices = {'ACME': 94.23, 'YOW': 45.2}
prices['ACME']
```

Out[6]:

```
94.23
```

## Counter (čítač)

také známý jako:

> `collections.Counter`

Typické použití:

> počítání výskytů, četností; histogram

Základní použití:

```
>>> c = Counter(sequence)
>>> c[key] += n
>>> c.most_commont(n)
```

In [7]:

```
from collections import Counter
c = Counter('xyzzy')
c    # A dictionary that counts the letters
```

Out[7]:

```
Counter({'x': 1, 'y': 2, 'z': 2})
```

In [8]:

```
c['a'] += 10
c['b'] += 13
c
```

Out[8]:

```
Counter({'a': 10, 'b': 13, 'x': 1, 'y': 2, 'z': 2})
```

In [9]:

```
c.most_common(2)
```

Out[9]:

```
[('b', 13), ('a', 10)]
```

## defaultdict (slovník s výchozími hodnotami)

Také známý jako:

> `collections.defaultdict`

Typické použití:

> relace 1 ku n; seskupování

Základní použití:

```
>>> d = defaultdict(list)
>>> d[key].append(val)
>>> values = d[key]

>>> d = defaultdict(set)
>>> d[key].add(val)
>>> unique_values = d[key]
```

In [10]:

```
from collections import defaultdict
d = defaultdict(list)
d['spam'].append(42)
d['blah'].append(13)
d['spam'].append(10)
d
```

Out[10]:

```
defaultdict(list, {'blah': [13], 'spam': [42, 10]})
```

# Síla a supersíla!

## Iterace přes kolekce

Cyklus:

```
for item in container: ...
```

Varianty:

```
for i, item in enumerate(container): ...

for item1, item2 in zip(container1, container2): ...
```

Agregační funkce:

```
sum(container)
min(container)
max(container)
any(container)
all(container)
```

## Comprehensions

List comprehensions:

```
[expression for x in iterable if condition]
```

Set comprehensions:

```
{expression for x in iterable if condition}
```

Dict comprehensions:

```
{key: val for key, val in iterable if condition}
```

In [11]:

```
nums = [1,2,3,4,5,6]
squares = []
for x in nums:
    squares.append(x*x)
squares
```

Out[11]:

```
[1, 4, 9, 16, 25, 36]
```

In [12]:

```
squares = [x*x for x in nums]
squares
```

Out[12]:

```
[1, 4, 9, 16, 25, 36]
```

# Generátory

Generátorový výraz:

```
(expression for x in iterable if condition)
```

Kombinovaný s agregační funkcí:

```
sum(expression for x in iterable if condition)
```

Tip:

- Generátory nám umožňují zpracovávat velká množství dat inkrementálně --- spoří obrovká množství paměti!

In [13]:

```
nums
```

Out[13]:

```
[1, 2, 3, 4, 5, 6]
```

In [14]:

```
squares = (x*x for x in nums)
squares
```

Out[14]:

```
<generator object <genexpr> at 0x05AEF600>
```