České vysoké učení technické v Praze FIT

Programování v Pythonu

Jiří Znamenáček

Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.

Praha & EU: Investujeme do vaší budoucnosti

Python – Seznamy

Úvod

Se seznamy se setkáme v Python'u úplně všude, málem stejně často jako s iterátory :) Je to jeden z nejdůležitějších sekvenčních typů – umožňuje schraňovat pohromadě libovolné (a proměnlivé) množství nejrůznějších objektů.

Z daných objektů vytvoříme seznam nejsnadněji jejich uzavřením do hranatých závorek:

```
>>> xs = [ 'Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 6
>>> xs
['Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 666']
```

Kromě toho je možné za pomoci funkce `list()` převádět jiné typy na seznam, například řetězce:

```
>>> xs = "Ahoj, světe!"
>>> xs
'Ahoj, světe!'
>>> list(xs)
['A', 'h', 'o', 'j', ',', ' ', 's', 'v', 'ě', 't', 'e', '!']
```

Seznamy jako sekvence

Jelikož seznamy patří mezi sekvence, máme k dispozici celou armádu sekvenčních operací:

```
>>> xs = ['Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 66

# délka sekvence
>>> len(xs)
5

# konkrétní prvek
>>> xs[3]
['hokus', 'pokus']
>>> xs[-3]
(12, 'tři')

# různé výřezy
>>> xs[1:3]
[123, (12, 'tři')]
>>> xs[1::2]
[123, ['hokus', 'pokus']]
>>> xs[3:]
[['hokus', 'pokus'], 'lokus 666']
>>> xs[-3:]
[(12, 'tři'), ['hokus', 'pokus'], 'lokus 666']

# dotaz na výskyt prvku
>>> 123 in xs
True
>>> (12, 'tři') in xs
True
>>> 'fokus' in xs
False

# dvě spojené kopie
>>> xs * 2
['Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 666', 'Ahoj
```

→ Komentáře zde pochopitelně nejsou součástí výstupu interaktivního interpretru, slouží pouze k popisu zde zapsaného kódu.

Po sekvencích se navíc přirozeně prochází smyčkou:

```
>>> for x in xs:
...     print(x)
...
Ahoj!
123
(12, 'tři')
['hokus', 'pokus']
lokus 666
```

Případně pokud je důležitá i pozice výskytu prvku:

```
>>> for (i, x) in enumerate(xs):
...     print(i, x)
...
0 Ahoj!
1 123
2 (12, 'tři')
3 ['hokus', 'pokus']
4 lokus 666
```

Seznamy jako měnitelné sekvence

Narozdíl např. od řetězců seznamy patří mezi sekvence **měnitelné** (*mutable*), což znamená, že jejich obsah po vytvoření můžeme libovolně upravovat:

```
>>> xs = ['Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 66

>>> xs[3]
['hokus', 'pokus']
>>> xs[3] = 'fokus'
>>> xs[3]
'fokus'
>>> xs
['Ahoj!', 123, (12, 'tři'), 'fokus', 'lokus 666']
```

Dotazování na prvky

Kromě již dříve uvedeného a všem sekvencím společného operátoru `in`:

```
>>> xs = ['Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 66

>>> 123 in xs
True
>>> (12, 'tři') in xs
True
>>> 'fokus' in xs
False
```

...mají seznamy k dispozici i metodu `index(PRVEK)`, která slouží k navrácení pozice prvního výskytu prvku s uvedenou hodnotou:

```
>>> xs = ['jedna', 'dvě', 'tři', 'jedna', 'pět', 'šest', ]

>>> xs.index('jedna')
0
>>> xs.index('pět')
4
```

→ Není-li takový prvek v seznamu k dispozici, obdržíme výjimku *ValueError*.

Kromě toho můžeme i počítat výskyty zadaného prvku pomocí metody `count(PRVEK)`:

```
>>> xs = ['jedna', 'dvě', 'tři', 'jedna', 'pět', 'šest', ]

>>> xs.count('jedna')
2
>>> xs.count('dvě')
1
```

Přidávání prvků

Přidat nové prvky do existující seznamu se dá celkem čtyřmi způsoby. Pro ukázku budeme pracovat nad počátečním seznamem `xs = [1, 2.0, 'tři']`:

I. Předně můžeme využít metody seznamu `append(PRVEK)` pro přidání jednoho nového prvku na konec seznamu:

```
>>> xs.append('four')
>>> xs
[1, 2.0, 'tři', 'four']
```

→ Ač to tak možná na první pohled nevypadá, ekvivalentní kód zařizující totéž jest: `xs[ len(xs): ] = ['four']`

II. Jeden prvek také můžeme pomocí metody `insert(POZICE_PRVKU_ZA, PRVEK)` vložit na vybrané místo seznamu:

```
>>> xs.insert(3, 3.5)
>>> xs
[1, 2.0, 'tři', 3.5, 'four']
```

→ Jinak řečeno `xs.insert(0, x)` přidá prvek *x* na začátek seznamu, zatímco `xs.insert(len(xs), x)` na konec (takže provede to samé jako `xs.append(x)`).

III. Na konec seznamu můžeme také přidat více prvků seznamu jiného, a to buď pomocí metody `extend(SEZNAM)`:

```
>>> xs.extend( [5, 'šest'] )
>>> xs
[1, 2.0, 'tři', 3.5, 'four', 5, 'šest']
```

IV. ..nebo pomocí „obyčejného“ slepování sekvencí pomocí operátoru `+`:

```
>>> xs += ['sedm', 8.0, 9]
>>> xs
[1, 2.0, 'tři', 3.5, 'four', 5, 'šest', 'sedm', 8.0, 9]
```

→ Z technického hlediska je nejdříve v paměti vytvořen nový dočasný seznam, do nějž jsou oba počáteční seznamy (tj. zde *xs* a *['sedm', 8.0, 9]*) „slepeny“, a teprve potom je tento přiřazen zpět identifikátoru *xs*. Vadit by vám to asi mohlo, kdyby slučované seznamy byly extrémně veliké.

Pozor na rozdíl mezi `append()` a `extend()`:

```
>>> xs = [1, 2, 3]
>>> xs
[1, 2, 3]

>>> xs.append( [4, 5, 6] )
>>> xs
[1, 2, 3, [4, 5, 6]]
>>> len(xs)
4
```

versus

```
>>> xs = [1, 2, 3]
>>> xs
[1, 2, 3]

>>> xs.extend( [4, 5, 6] )
>>> xs
[1, 2, 3, 4, 5, 6]
>>> len(xs)
6
```

Odebírání prvků

Odebrat prvky ze seznamu se dá několika způsoby. Pro ukázku budeme

pracovat nad počátečním seznamem `xs = ['jedna', 'dvě', 'tři', 'čtyři', 'pět', 'šest']`:

V. Odebrat prvek, resp. prvky, na základě jeho, resp. jejich, pozice (indexu) nám umožňuje univerzální příkaz `del`:

```
>>> del xs[1]
>>> xs
['jedna', 'tři', 'čtyři', 'pět', 'šest']

>>> del xs[3:]
>>> xs
['jedna', 'tři', 'čtyři']
```

→ Stačí tedy pomocí *slice*-notace označit, čeho se chceme zbavit, a *del* se již postará o ostatní.

VI. Pro jeden prvek víceméně totéž zařídí metoda seznamu `pop([INDEX])`. Rozdíl je v tom, že odebraný prvek je metodou vrácen:

```
>>> xs = ['jedna', 'dvě', 'tři', 'čtyři', 'pět', 'šest']

>>> xs.pop()     # ostraň poslední prvek
'šest'
>>> xs
['jedna', 'dvě', 'tři', 'čtyři', 'pět']

>>> xs.pop(0)    # odstraň první prvek
'jedna'
>>> xs
['dvě', 'tři', 'čtyři', 'pět']

>>> xs.pop(2)    # odstraň prvek na pozici 2, tj. třetí
'čtyři'
>>> xs
['dvě', 'tři', 'pět']
```

VII. Poslední možností je odstranit prvek na základě jeho obsahu, a nikoli indexu. O to se stará metoda `remove(PRVEK)`. Odstraněn je vždy první výskyt daného prvku:

```
>>> xs = ['jedna', 'dvě', 'tři', 'dvě', 'pět', 'šest']

>>> xs.remove('dvě')
>>> xs
['jedna', 'tři', 'dvě', 'pět', 'šest']
```

→ Pokus o odstranění prvku, který se v seznamu nenachází, skončí výjimkou *ValueError*.

Řazení

Prvky seznamu můžeme „otočit“, tj. zaměnit jejich pořadí od posledního k prvnímu, metodou `reverse()`:

```
>>> xs = [1, 2, 3, 4, 5, 6]
>>> xs
[1, 2, 3, 4, 5, 6]

>>> xs.reverse()
>>> xs
[6, 5, 4, 3, 2, 1]
```

→ Metoda `reverse()` operuje nad aktuálním seznamem, tj. nic nevrací, protože zamění pořadí prvků rovnou v seznamu, na kterém ji zavoláte!

Skutečné setřídění prvků seznamu (lexikografické) se pak dá udělat buď stejně přímo v místě (takže ztratíme původní uspořádání) pomocí metody sekvencí `sort([key[, reverse]])`:

```
>>> xs = ['jedna', 'dvě', 'tři', 'čtyři', 'pět', 'šest', 'sedm']

>>> xs.sort()
>>> xs
['dvě', 'jedna', 'pět', 'sedm', 'tři', 'čtyři', 'šest']
```

→ Výchozí třídění řetězců probíhá porovnáváním unicodových *code-points* a česká nabodeníčka jsou holt až za ASCII-prvky...

...nebo alternativně pomocí vestavěné funkce `sorted()`, která ponechá původní seznam beze změny a utříděný vrátí:

```
>>> xs = ['jedna', 'dvě', 'tři', 'čtyři', 'pět', 'šest', 'sedm']

>>> ys = sorted(xs)
>>> ys
['dvě', 'jedna', 'pět', 'sedm', 'tři', 'čtyři', 'šest']
>>> xs
['jedna', 'dvě', 'tři', 'čtyři', 'pět', 'šest', 'sedm']
```

Obě třídící funkce navíc akceptují na vstupu parametr *reverse* signalizující třídění v opačném směru:

```
>>> xs = [3, 2, 7, 6, 1, 5]
>>> sorted(xs)
[1, 2, 3, 5, 6, 7]
>>> sorted(xs, reverse=True)
[7, 6, 5, 3, 2, 1]
```

Posledním – a veskrze „magickým“ :) – parametrem je parametr *key*, který slouží (v podobě funkce jednoho argumentu) k určení porovnávacího klíče nad prvky seznamu:

```
>>> names = [ "John", "Wendy", "Pete", "Jane", "David", "Amanda",

# setřídění podle délky prvků
# ~ třídění je stable, takže zachová původní pořadí prvků se stejn
>>> sorted(names, key=len)
['Ix', 'John', 'Pete', 'Jane', 'Wendy', 'David', 'Amanda']

# Pokud chceme stejně dlouhé prvky navíc setřídit i podle abecedy,
>>> sorted(names, key=lambda x: (len(x), x))
['Ix', 'Jane', 'John', 'Pete', 'David', 'Wendy', 'Amanda']
```

→ Ve druhém příkladu parametr *key* v podstatě vyrobil z původního seznamu seznam tuplů `[(4, 'John'), (5, 'Wendy'), (4, 'Pete'), (4, 'Jane'), (5, 'David'), (6, 'Amanda'), (2, 'Ix'), ]`, který následně lexikograficky setřídil a zpětně z něj vyextrahoval prvky původního seznamu. Ve starších verzích Python'u byste si tento upravený „meziseznam“ i extrakci setříděných prvků museli zařídit sami.

→ Toto je naprosto typické místo použití lambda-funkcí.

Seznam jako zásobník

S dostupnými metodami proměníte seznam velmi snadno v *zásobník*, tedy paměťovou strukturu, kde poslední přidaný prvek je také první vrácený (*LIFO*). Příklad přímo z dokumentace:

```
>>> stack = [3, 4, 5]

>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]

>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]

>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Seznam jako fronta

Téměř stejně jednoduše jako zásobník můžete naimplementovat i *frontu*, tedy paměťovou strukturu, kde první přidaný prvek je také první vrácený (*FIFO*). Příklad přímo z dokumentace:

```
>>> queue = ["Eric", "John", "Michael"]

>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.pop(0)
'Eric'
>>> queue.pop(0)
'John'
>>> queue
['Michael', 'Terry', 'Graham']
```

List comprehensions

> → Po vzoru Petra Přikryla, překladatele úžasné knihy Marka Pilgrima Dive Into Python 3, budu podle aktuálního použití překládat anglický termín *list comprehension* někdy jako „generátorová notace (seznamu)“, jindy jako „generátor seznamu“ a ještě jindy i jako „generovaný seznam“.

Bezesporu jednou z nejúžasnějších „zkratek“, které nám Python nabízí, je

generátorová notace (nejen) seznamu. Umožňuje nám často velmi elegantně a na minimu prostoru vytvořit z původního seznamu seznam nový, který se skládá třeba i z úplně nových prvků, vytvořených však na základě prvků seznamu původního a dodatečných pravidel.

Obecný konstruktor pro jednoduchý generátor seznamu vypadá takto:

```
[ Funkce(I) for I in SEKVENCE [if PODMÍNKA] ]
```

→ Zde vnější hranaté závorky vyznačují generovaný seznam, zatímco vnitřní pouze to, že část s podmínkou není povinná.

Spíše než složité vysvětlování několik snad názorných ukázek:

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

# kopie seznamu
>>> [x for x in xs]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# každý prvek seznamu zvětšený o tři
>>> [x+3 for x in xs]
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

# seznam druhých mocnin prvků původního seznamu
>>> [x*x for x in xs]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# výběr sudých prvků
>>> [x for x in xs if x % 2 == 0]
[0, 2, 4, 6, 8]

# seznam tuplů, kde je každý prvek přítomen jako číslo a zároveň i
>>> [(x, str(x)) for x in xs]
[(0, '0'), (1, '1'), (2, '2'), (3, '3'), (4, '4'), (5, '5'), (6, '

# jako předchozí, ale pouze pro liché prvky z původního seznamu
>>> [(x, str(x)) for x in xs if x % 2 != 0]
[(1, '1'), (3, '3'), (5, '5'), (7, '7'), (9, '9')]
```

Vícenásobné 'List comprehensions'

Vícenásobná generátorová notace už nevypadá tak průhledně a ne vždy dělá to, co by člověk na první pohled hádal:

```
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]

# pro každý prvek první smyčky bude znovu vykonána celá smyčka dru
>>> [x*y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]

# z tohohle by se dal vytěžit skalární součin
>>> [vec1[i]*vec2[i] for i in range(len(vec1))]
[8, 12, -54]
```

→ Převzato z dokumentace k Python'u.

Ještě poměrně průhledný, i když dosti šílený způsob jak najít prvočísla menší než 50:

```
>>> noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
>>> noprimes
[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36,

>>> primes = [x for x in range(2, 50) if x not in noprimes]
>>> primes
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

A jelikož se generátory seznamu dají zanořovat do sebe...

```
>>> [x for x in range(2, 50) if x not in [j for i in range(2, 8) f
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

→ http://www.secnetix.de/olli/Python/list_comprehensions.hawk