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 – Třídění

Třídění sekvenčních typů

Python má zabudovaný algoritmus pro třídění *TimSort*, který je poměrně univerzální, rychlý a hlavně stabilní, takže se dá použít pro vícenásobné třídění objektů podle více kritérií. Jeho aplikaci umožňují dvě metody:

I. Metoda `xs.sort([key[, reverse]])` řadí „v místě“ objekty typu *seznam*:

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

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

→ Aby dala ještě více najevo, že objekty jsou řazeny „v místě“, vrací metoda `xs.sort()` *None*.

II. Globální funkce `xs = sorted(iterable[, key][, reverse])` dokáže nějakým způsobem setřídit každý iterovatelný typ, který se nějak setřídit dá, přičemž vrací seznam:

```
>>> xs = { 'b': 2, 'a': 1, 'c': 9, 'd': 7, }

>>> ys = sorted(xs)
>>> ys
['a', 'b', 'c', 'd']
```

Nepovinný boolean-parametr *reverse* nastavením na *True* způsobí změnu směru řazení (vlastně provádí všechna porovnávání opačně):

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

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

V dalším textu ho, a stejně tak i metodu seznamu `sort()`, budeme ignorovat a probereme důkladně pouze druhý pojmenovaný parametr *key* a logiku schovanou za vlastním tříděním.

Třídění je lexikografické

Objekty v Python'u se třídí podle svého typu:

```
# třídění čísel podle velikosti
>>> sorted( [1, 5, 3, 2, 4, ] )
[1, 2, 3, 4, 5]

# třídění řetězců podle abecedy..
>>> sorted( ['ovce', 'kočka', 'pes', 'hroznýš', 'prase', ] )
['hroznýš', 'kočka', 'ovce', 'pes', 'prase']

# ..ovšem unicodové
>>> sorted( ['jedna', 'dvě', 'tři', 'čtyři', 'pět', 'šest', 'sedm'
['dvě', 'jedna', 'pět', 'sedm', 'tři', 'čtyři', 'šest']
```

Složitější objekty se pak třídí postupně po jednotlivých částech, přičemž každý „základní/jednoduchý“ typ se třídí stejně jako výše:

```
>>> sorted( [('two', 3), ('one', 4), ('one', 2), ('three', 5), ('t
[('one', 2), ('one', 4), ('three', 5), ('two', 1), ('two', 3)]
```

→ Tzn. že n-tice v seznamu se nejdříve setřídily podle své první položky, tj. abecedně, a poté zde shodné ještě navíc i podle druhé, tj. podle velikosti.

Magický parametr „key“

Třídění můžeme vnutit, aby se chovalo jinak, než by pro daný typ bylo běžné. To zajišťuje nepovinný pojmenovaný parametr *key*, který specifikuje funkci, která bude před vlastním porovnáváním zavolána na každý prvek vstupního iterovatelného typu. Několik příkladů:

I. Využití předdefinované funkce:

```
>>> names = [ "John", "Wendy", "Pete", "Jane", "David", "Amanda",
>>> sorted( names, key=len )
['Ix', 'John', 'Pete', 'Jane', 'Wendy', 'David', 'Amanda']
```

→ Zde je každý prvek vstupního seznamu, tedy řetězec, převeden na svou délku (ve znacích) a teprve podle ní je porovnán.

II. Využití lambda-funkce:

```
>>> sorted( [('two', 3), ('one', 4), ('one', 2), ('three', 5), ('t
[('two', 1), ('one', 2), ('two', 3), ('one', 4), ('three', 5)]
```

→ Zde jsme místo výchozího lexikografického porovnání vnutili třídění podle druhé části (`x[1]`) každého prvku vstupního seznamu.

Složitější použití lambda-funkce:

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

# A) setřídění pouze podle délky slov
>>> sorted(names, key=len)
['Ix', 'John', 'Pete', 'Jane', 'Wendy', 'David', 'Amanda']

# B) setřídění navíc i podle abecedy
>>> sorted(names, key=lambda x: (len(x), x))
['Ix', 'Jane', 'John', 'Pete', 'David', 'Wendy', 'Amanda']
```

→ V příkladu B parametr *key* v podstatě vyrobil z původního seznamu seznam n-tic `[(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 (viz tzv. metoda DSO, „decorate-sort-undecorate“).

Modul „operator“

Třídění podle vybrané části složitějšího objektu je natolik častý obrat, že Python poskytuje v rámci modulu *operator* několik pomocných funkcí, které nám ušetří psaní výběrových lambda-funkcí:

```
>>> xs = [('two', 3), ('one', 4), ('one', 2), ('three', 5), ('two'

# A) výchozí lexikografické třídění
>>> sorted(xs)
[('one', 2), ('one', 4), ('three', 5), ('two', 1), ('two', 3)]

# B1) třídění za pomoci výběrové lambda-funkce
>>> sorted(xs, key=lambda x: x[1])
[('two', 1), ('one', 2), ('two', 3), ('one', 4), ('three', 5)]

# B2) totéž, ale za pomoci metody itemgetter()
>>> from operator import itemgetter
>>> sorted(xs, key=itemgetter(1))
[('two', 1), ('one', 2), ('two', 3), ('one', 4), ('three', 5)]
```

Modul *operator* poskytuje i další metody, které se uplatní zvláště u třídění instancí objektů: `operator.attrgetter()`, `operator.methodcaller()`

Třídění je stabilní

Všimněme si ještě jednou příkladu:

```
>>> names = [ "John", "Wendy", "Pete", "Jane", "David", "Amanda",
>>> sorted( names, key=len )
['Ix', 'John', 'Pete', 'Jane', 'Wendy', 'David', 'Amanda']
```

Jelikož porovnávací funkcí je `len()`, tedy délka řetězce, provede Python na vstupním objektu jen tolik změn, kolik je nezbytně nutné. Ve výsledku si čtyřpísmenná ('John', 'Pete', 'Jane') i pětipísmenná ('Wendy', 'David') slova zachovají stejné pořadí jako na vstupu.

> → Nezapomeňte – provedli jsme srovnání podle délky, nikoli abecedy!

Stability se dá využít pro třídění podle více kritérií – postupně provedeme třídění od posledního požadavku po první. Např. studenty podle známky a poté podle věku setřídíme následovně:

```
# ('jméno', 'známka', věk)
>>> xs = [ ('Petr', 'A', 23), ('Bořek', 'A', 21), ('Pavel', 'B', 2

# a) nejdříve provedeme třídění podle věku..
>>> s = sorted(xs, key=lambda x: x[2])
# b) ..a poté podle známky
>>> sorted(s, key=lambda x: x[1])
[('Bořek', 'A', 21), ('Petr', 'A', 23), ('Pavel', 'B', 21)]
```

Poznámky

- Třídění můžeme vnutit ohled na národní zvyklosti pomocí metody `key = locale.strxfrm()` modulu `locale`. Ne že by to bylo zrovna jednoduché...
- Výchozí porovnávání objektů využívá jejich metodu `__lt__()`, je§li k dispozici. Jejím definováním tudíž můžeme předepsat, jak se naše objekty budou mezi sebou porovnávat.
- Funkce vstupující do parametru *key* vůbec nemusí operovat na porovnávaných objektech. Má-li to smysl, může hodnoty pro třídění brát úplně odjinud.