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 – Slovníky

Úvod

Slovníky jsou nejdůležitějším (a po dlouhou dobu také jediným) zástupcem tzv. *mapovacích typů*. Jejich struktura je následující:

```
slovník = {
  KLÍČ1: HODNOTA1,
  KLÍČ2: HODNOTA2,
  ...
}
```

Přitom – jak je v Python'u zvykem – jak hodnoty, tak klíče mohou být najednou nejrůznějších typů. Přesněji *hodnotou* může být cokoliv, *klíč* však musí být **neměnný** datový typ (*immutable*). Typicky se indexuje řetězcem nebo číslem, ale stejně tak dobře můžete použít například právě n-tici, pokud se sama skládá z neměnných typů.

> → Slovníky jsou v podstatě hash-tabulky optimalizované na vyhledávání hodnot podle klíčů.

Nejsnadněji (bez využití jiné datové struktury a speciálních metod) slovník vytvoříme přímou notací:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }
>>> type(ds)
<class 'dict'>
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3]}
```

> → Už vidíme, proč (mimo jiné) slovníky nepatří mezi sekvence – není u nich zaručeno pořadí prvků.

Základní vlastnosti

Ačkoli slovníky nepatří mezi sekvence, některé z vlastností mají – díky své struktuře „balíků na hodnoty“ – společné:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

# dotaz na počet dvojic klíč-hodnota
>>> len(ds)
3

# dotaz na výskyt klíče
>>> 'a' in ds
True
>>> 123 in ds
False

# přirozený cyklus přes klíče slovníku
>>> for key in ds:
...     print(key)
...
a
c
b

# dotaz na hodnotu odpovídající klíči 'b'
>>> d['b']
[1, 2, 3]
```

→ Všimněte si obzvláště, že slovník je struktura optimalizovaná na přístup podle klíčů – základní „přirozené“ operace se slovníkem se odehrávají na úrovni klíčů, nikoli hodnot.

Přístup k prvkům

Existují dva základní způsoby jak přistupovat k prvkům slovníku:

- známe-li klíč, můžeme pomocí něj získat odpovídající hodnotu (viz `slovník[klíč]` z předchozího slajdu)
- pomocí smyčky přes vybraný „pohled“ na slovník probrat prvky slovníku jeden po druhém (ovšem s víceméně náhodným pořadím)

Co se „pohledů“ (*views*) na slovník (označme ho *ds*) týká, jsou tři různé:

1. `ds.keys()` – pohled přes klíče
2. `ds.values()` – pohled přes hodnoty
3. `ds.items()` – pohled přes n-tice *(klíč, hodnota)*

…a sdílejí stejné vlastnosti:

- každého pohledu se můžeme zeptat na počet jeho členů pomocí funkce `len()`
- přes každý pohled můžeme iterovat (např. pomocí smyčky `for in`)
- každý pohled můžeme testovat na přítomnost prvků pomocí operátoru `in`

I. Pohled přes klíče je pohledem „základním“ a vlastně jsme ho viděli v akci už na předchozím slajdu. Je totiž:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

>>> ds.keys()
dict_keys(['a', 'c', 'b'])

>>> len( ds.keys() )
3

>>> 'a' in ds.keys()
True

>>> for key in ds.keys():
...     print(key)
...
a
c
b
```

→ Čili pro většinu praktických aplikací část `.keys()` vůbec nemusíme uvádět, je brána jako implicitní.

→ Jelikož přístup k hodnotám slovníku je realizován přes klíče, máme při tomto základním způsobu získávání informací o slovníku přístup jak ke klíčům, tak k jim odpovídajícím hodnotám.

II. Pohled přes hodnoty je doplňkovým k pohledu přes klíče:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

>>> ds.values()
dict_values([1, 'ahoj', [1, 2, 3]])

>>> len( ds.values() )
3

>>> 'ahoj' in ds.values()
True

>>> for val in ds.values():
...     print(val)
...
1
ahoj
[1, 2, 3]
```

- → Oba pohledy prochází prvky slovníku ve stejném pořadí (i když víceméně náhodném).
- → Narozdíl od pohledu přes klíče se při pohledu přes hodnoty na odpovídající prvek v páru nedostaneme – heš-tabulka je jednosměrná.

III. Posledním – a nejúplnějším – pohledem na slovník je pohled přes páry klíč-hodnota:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

>>> ds.items()
dict_items([('a', 1), ('c', 'ahoj'), ('b', [1, 2, 3])])

>>> len( ds.items() )
3

>>> ('a', 1) in ds.items()
True

>>> for (key, val) in ds.items():
...     print(key, val)
...
a 1
c ahoj
b [1, 2, 3]
```

- → Vidíme, že prvky slovníku jsou opět procházeny ve stejném pořadí. Tentokrát jsou ale získávány ve formě n-tice tvaru *(klíč, hodnota)*, resp. *(klíč, ds[klíč])*.

Základní práce se slovníky

Pro nejběžnější práci se slovníky si vystačíme s několika základními „triky“:

IV. Novou dvojici klíč-hodnota přidáme do slovníku velmi podobným způsobem, jako se ptáme na existující:

```
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3]}

>>> ds['d'] = (1, 2,)
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3], 'd': (1, 2)}
```

V. Pokud provedeme totéž pro již existující klíč, dojde k nahrazení jeho hodnoty:

```
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3], 'd': (1, 2)}

>>> ds['d'] = 'svět'
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3], 'd': 'svět'}
```

VI. Již nepotřebné dvojce klíč-hodnota se zbavíme pomocí univerzálního `del`:

```
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3], 'd': 'svět'}

>>> del ds['d']
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3]}
```

VII. Počáteční slovník můžeme vytvořit buď ručně..

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3]}
```

..nebo z vhodné datové struktury pomocí funkce/konstrukturu `dict()`:

```
>>> ds = dict( [('a', 1), ('b', [1, 2, 3]), ('c', 'ahoj')] )

>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3]}
```

Mimochodem prázdný slovník zavedeme také buď pomocí `dict()` nebo přímou notací:

```
>>> ds = {}

>>> ds
{}
>>> type(ds)
<class 'dict'>

>>> ds[123] = 'ahoj'
>>> ds
{123: 'ahoj'}
```

A ještě znovu upozorním na to, že ani klíče nemusí být stejného typu:

```
>>> ds = { 'a': 'ahoj', 123: 'svět', (12, 'fokus'): 666, }

>>> ds
{'a': 'ahoj', (12, 'fokus'): 666, 123: 'svět'}

>>> ds['a']
'ahoj'
>>> ds[123]
'svět'
>>> ds[ (12, 'fokus') ]
666
```

Pomocné metody

Metody z předchozího slajdu vám dost pravděpodobně vystačí skoro na všechno, co kdy budete kolem slovníků potřebovat. Ale slovníky navíc poskytují spoustu užitečných dalších metod:

VIII. Do slovníku můžeme přidat více dvojic klíč-hodnota najednou pomocí metody `update(SLOVNÍK)`:

```
>>> ds = {}
>>> ds
{}

>>> ds.update( {'a': 1, 'b': [1, 2, 3]} )
>>> ds
{'a': 1, 'b': [1, 2, 3]}
```

Podobně jako u tvorby slovníku však parametrem může být i jiná vhodná datová struktura:

```
>>> ds.update( [('c', 'ahoj'), (666, 'lokus')] )
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3], 666: 'lokus'}
```

IX. Jelikož dotaz na neexistující klíč vrátí výjimku *KeyError*..

```
>>> ds
{'a': 1, 'c': 'ahoj', 'b': [1, 2, 3], 666: 'lokus'}

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

..měli bychom před každým dotazem kontrolovat, zda příslušný klíč ve slovníku vůbec je (nebo odchytávat výjimku a zachovat se podle toho). S pomocí metody `get(KLÍČ[, výchozí_hodnota])` se to však dá zjednodušit:

```
>>> x = ds.get('c')
>>> x
'ahoj'

>>> x = ds.get('d')
>>> x

>>> x = ds.get('d', 0)
>>> x
0
```

> → Jinými slovy: Pro existující klíč se get() chová jako standardní získání hodnoty příslušející danému klíči, pro neexistující prvek automaticky odchytí výjimku a vrátí buď nic (výchozí hodnota) nebo to, co si sami určíme.

X. Trochu podobná a poněkud kriptická metoda `setdefault(key[, default])` pro existující klíč vrací jeho hodnotu, pro neexistující ho ve slovníku zavede (s hodnotou *default* nebo výchozím *None*):

```
>>> x = ds.setdefault('c')
>>> x
'ahoj'

>>> x = ds.setdefault(123, 777)
>>> x
777
>>> ds
{'a': 1, 123: 777, 'c': 'ahoj', 'b': [1, 2, 3], 666: 'lokus'}
```

XI. Slovník můžete smazat pomocí jeho metody `clear()`:

```
>>> ds.clear()
>>> ds
{}
```

XII. Tvorbu slovníku s výchozí hodnotou nám může často usnadnit metoda `fromkeys(seq[, value])`, která zavede slovník s klíči ze sekvence *seq* a případnou výchozí hodnotou *default* (jinak *None*):

```
>>> ds = {}.fromkeys( range(4), '' )
>>> ds
{0: '', 1: '', 2: '', 3: ''}
```

XIII. Slovník můžeme také prvek po prvku „zničit“. K dispozici jsou na to dvě různé metody. První z nich, `pop(key[, default])`, odstraní prvek s klíčem *key* a vrátí jeho hodnotu (nebo *default*):

```
>>> ds = { 'a': 'ahoj', 123: 'svět', (12, 'fokus'): 666, }
>>> ds
{'a': 'ahoj', (12, 'fokus'): 666, 123: 'svět'}

>>> ds.pop('a')
'ahoj'
>>> ds
{(12, 'fokus'): 666, 123: 'svět'}
```

→ Pokud *default* neuvedete a *key* není platným klíčem, dočkáte se výjimky *KeyError*.

V podobném duchu operuje metoda `popitem()`, která ovšem vrací dvojici *(klíč, hodnota)* a navíc v náhodném pořadí (vhodné pro množinové operace):

```
>>> ds = { 'a': 'ahoj', 123: 'svět', (12, 'fokus'): 666, }
>>> ds
{'a': 'ahoj', (12, 'fokus'): 666, 123: 'svět'}

>>> ds.popitem()
('a', 'ahoj')
```

→ Pokus o zavolání `popitem()` na prázdném slovníku vrátí *KeyError*.

XIV. Jako jedna z mála datových struktur nabízí slovník i vlastní metodu pro tvorbu mělké kopie: `copy()`

„Řazení“

Ve výchozím stavu prochází cyklus prvky slovníku v náhodném pořadí:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

>>> for key in ds:
...     print(key)
...
a
c
b
```

Pomocí zabudované funkce `sorted()` si ale můžeme *aktuální* průchod pro *navzájem porovnatelné* klíče vynutit seřazený:

```
>>> ds = { 'a': 1, 'b': [1,2,3], 'c': "ahoj", }

>>> for key in sorted(ds):
...     print(key)
...
a
b
c
```

Generátorová notace

Python 3 rozšířil generátorovou notaci – dostupnou dříve pouze u seznamů – na další dva typy. A jedním z nich je právě slovník. Obecný konstruktor vypadá velmi podobně, jako u seznamu:

```
{ KLÍČ:HODNOTA for I in SEKVENCE [if PODMÍNKA] }
```

→ Jako u seznamu vnitřní závorky zde označují nepovinnou část výrazu. Na první části je důležité, že má strukturu *KLÍČ: HODNOTA*, ne už tak, že oba prvky mohou být funkcí prvků procházené sekvence.

XV. Lepší radši rovnou příklad – jak dosáhnout stejného výsledku jako metoda *fromkeys*:

```
>>> { i:'' for i in range(4) }
{0: '', 1: '', 2: '', 3: ''}
```

XVI. A něco složitějšího:

```
>>> xs = "Ahoj, světe!"

>>> { i:x for (i, x) in enumerate(xs) }
{0: 'A', 1: 'h', 2: 'o', 3: 'j', 4: ',', 5: ' ', 6: 's', 7: 'v', 8
```

→ Aneb jak velmi komplikovaně přiřadit k pozici v řetězci

odpovídající písmeno, když totéž zvládá přímo xs[i] ^_^

```
>>> xs = "Abcd abcd"

>>> { x:ord(x) for (i, x) in enumerate(xs) }
{'A': 65, ' ': 32, 'c': 99, 'b': 98, 'd': 100, 'a': 97}
```

→ Aneb jak pro každé písmeno v řetězci získat jeho unicodový *code-point*. (Jelikož klíč slovníku je jedinečný, proiterujeme sice všechna písmena řetězce, ale ve slovníku skončí každé pouze jednou.)