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 – Množiny

Úvod

Množina je neuspořádaná kolekce jedinečných a neměnných (*immutable*) hodnot. K jejímu vytvoření slouží konstruktor `set()` nebo přímá notace za pomoci složených závorek:

```
>>> xs = { 1, 2, 'ahoj', 3, 'b', }

>>> xs
{3, 1, 2, 'b', 'ahoj'}

>>> type(xs)
<class 'set'>
```

> → Takže se o množinách dá uvažovat trošku jako o „slovnících bez hodnot“ – klíče ve slovníku jsou také jedinečné, musí být neměnné a jejich pořadí není určené.

Stejně tak je možné vytvořit množinu z jiných datových typů, např. seznamu (který obsahuje pouze neměnné datové typy):

```
>>> xs = [ 1, 2, 'ahoj', 3, 'b', ]
>>> set(xs)
{1, 2, 3, 'ahoj', 'b'}
```

Množiny v Python'u jsou jinak množinami v matematickém smyslu slova, takže na nich můžeme provádět standardní množinové operace jako je sjednocení, průnik nebo rozdíl.

Základní vlastnosti

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

```
>>> xs = { 1, 2, 'ahoj', 3, 'b', }

# dotaz na počet prvků
>>> len(xs)
5

# dotaz na výskyt prvku
>>> 'b' in xs
True
>>> 5 in xs
False

# přirozený cyklus přes prvky množiny
>>> for x in xs:
...     print(x)
...
1
2
3
ahoj
b
```

→ Cyklus *for in* projde všechny prvky množiny, ale ve víceméně náhodném pořadí. Použití má tedy skutečně převážně pro „množinové“ operace, kde nás zajímají vlastnosti či funkce prvků.

Přidávání prvků

Přidat prvky do množiny se dá dvěma způsoby:

I. Metoda `add(PRVEK)` přebírá jeden prvek (libovolného typu) a pokud ještě v množině není, přidá ho tam:

```
>>> xs = {1, 2.0, 'tři', }

# přidání nového prvku
>>> xs.add('four')
>>> xs
{'four', 1, 2.0, 'tři'}

# pokus o přidání již existujícího prvku
>>> xs.add(1)
>>> xs
{'four', 1, 2.0, 'tři'}
```

II. Metoda `update(YS[, ZS, ...])` přebírá na vstupu jednu (nebo více) množin a postupně je prvek po prvku zanese do hlavní množiny:

```
>>> xs = {'four', 1, 2.0, 'tři'}

# přidání prvků z jedné množiny
>>> xs.update( {5, 'šest'} )
>>> xs
{'four', 1, 2.0, 'šest', 'tři', 5}

# přidání prvků ze dvou množin
>>> xs.update( {'sedm', 8.0,}, {9,} )
>>> xs
{'four', 1, 2.0, 'šest', 'tři', 9, 8.0, 'sedm', 5}
```

Kromě toho dokáže zpracovat na vstupu i typy, které je možno na množiny převést, a s jejich prvky provede stejnou operaci. Například pro seznam:

```
>>> xs = {1, 2.0, 'tři', }

>>> xs.update( ['four', 5, 1, 'tři',] )
>>> xs
{'four', 1, 2.0, 'tři', 5}
```

Odebírání prvků

Odebrat prvky z množiny je možné třemi rozdílnými způsoby:

I. Metoda `discard(PRVEK)` se pokusí daný prvek odstranit z množiny. Pokud tam není, nic se nestane:

```
>>> xs = {1, 2.0, 'tři', }

# odstranění prvku
>>> xs.discard(2.0)
>>> xs
{1, 'tři'}

# pokus o odstranění neexistujícího prvku
>>> xs.discard(2.0)
>>> xs
{1, 'tři'}
```

> → Tudíž není-li pro vás případná neexistence odstraňovaného prvku v množině důležitá, použijete *discard()*.

II. Metoda `remove(PRVEK)` dělá úplně to samé jako metoda *discard()*, ale zavoláte-li ji pro neexistující prvek, bude se na vás „zlobit“:

```
>>> xs = {1, 2.0, 'tři', }

# odstranění prvku
>>> xs.remove(2.0)
>>> xs
{1, 'tři'}

# pokus o odstranění neexistujícího prvku
>>> xs.remove(2.0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2.0
```

→ Tudíž potřebujete-li se při odstraňování prvku z množinu při jeho případné neexistenci zachovat jinak, použijete metodu *remove()*. Nebo se nejdřív pomocí operátoru `in`, resp. `not in`, zeptáte, zda daný prvek v množině je, resp. není.

III. Metoda `x = xs.pop()` odstraňuje a zároveň vrací náhodný prvek z množiny. Pokus o odstranění prvku z prázdné množiny selže:

```
>>> xs = {'tři', 1, 2.0, }
>>> xs
{1, 2.0, 'tři'}

# postupné odstraňování (a vracení) náhodných prvků množiny
>>> xs.pop()
1
>>> xs.pop()
2.0
>>>
>>> xs.pop()
'tři'

# pokus o odstranění (a vrácení) prvku z prázdné množiny
>>> xs.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'pop from an empty set'
```

Množinové operace

Množiny podporují tradiční sadu množinových operací. K dispozici jsou ve dvou podobách:

I. Více bližší jazyku matematiky (a tak nejspíš průhlednější) jsou množinové operace pomocí binárních operátorů:

```
# zaveďme dvě množiny jako uskupení jedinečných písmen ze dvou řet
>>> xs = set('abracadabra')
>>> ys = set('alabama')
>>> xs
{'a', 'r', 'b', 'c', 'd'}
>>> ys
{'a', 'b', 'm', 'l'}

# PRŮNIK: písmena, která jsou zároveň v xs i ys
>>> xs & ys
{'a', 'b'}

# SJEDNOCENÍ: písmena, která jsou v xs a v ys
>>> xs | ys
{'a', 'c', 'b', 'd', 'm', 'l', 'r'}

# ROZDÍL: písmena z xs, která nejsou v ys
>>> xs - ys
{'c', 'r', 'd'}

# SYMETRICKÝ ROZDÍL: písmena, která jsou buď pouze v xs nebo pouze
>>> xs ^ ys
{'c', 'd', 'm', 'l', 'r'}
```

→ (Upraveno podle množinového příkladu z dokumentace.)

Máme také k dispozici standardní dotazy na množiny:

```
>>> xs = {'tři', 1, 2.0, }
>>> ys = {'tři', }

# Je/není (vlastní) podmnožinou?
>>> ys <= xs
True
>>> ys < xs
True

>>> ys <= ys
True
>>> ys < ys
False

# Je/není (vlastní) nadmnožinou?
>>> xs >= ys
True
>>> xs > ys
True

# Je/není prvkem?
>>> 2.0 in xs
True
>>> 2.0 not in xs
False
```

II. Druhou variantu představují metody objektů typu *množina*. Největším rozdílem oproti operátorovým verzím je, že jako své argumenty akceptují libovolné typy, ze kterých je možno „vyrobit“ množiny.

Následují příklady odpovídající výše uvedeným operátorovým verzím:

```
>>> xs = set('abracadabra')
>>> ys = set('alabama')

# PRŮNIK: písmena, která jsou zároveň v xs i ys [odpovídá operátor
>>> xs.intersection(ys)
{'a', 'b'}

# SJEDNOCENÍ: písmena, která jsou v xs a v ys [odpovídá operátoru
>>> xs.union(ys)
{'a', 'c', 'b', 'd', 'm', 'l', 'r'}

# ROZDÍL: písmena z xs, která nejsou v ys [odpovídá operátoru -]
>>> xs.difference(ys)
{'c', 'r', 'd'}

# SYMETRICKÝ ROZDÍL: písmena, která jsou buď pouze v xs nebo pouze
>>> xs.symmetric_difference(ys)
{'c', 'd', 'm', 'l', 'r'}
```

```
>>> xs = {'tři', 1, 2.0, }
>>> ys = {'tři', }

# Je ys podmnožina xs/ys? [odpovídá operátoru <=]
>>> ys.issubset(xs)
True
>>> ys.issubset(ys)
True

# Je xs nadmnožina ys? [odpovídá operátoru >=]
>>> xs.issuperset(ys)
True
```

Navíc je k dispozici operace `isdisjoint()`, která nemá operátorovou variantu, a vrací *True*, pokud množiny nemají žádné společné prvky:

```
>>> {'tři', 1, 2.0, }.isdisjoint( {'tři', } )
False

>>> {'tři', 1, 2.0, }.isdisjoint( {'four', 5, } )
True
```

→ Neboli `isdisjoint()` vrátí *True*, pokud množiny mají prázdný průnik, tj. platí `{'tři', 1, 2.0, }.isdisjoint( {'four', 5, } ) == ( {'tři', 1, 2.0, }.intersection( {'four', 5, } ) == set() )`

Další operace

I. Množinu můžeme smazat pomocí její metody `clear()`:

```
>>> xs.clear()
>>> xs
set()
```

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

III. Operace za metodou `update()` je v podstatě *sjednocení* více množin. Dá se tudíž zapsat oběma následujícími způsoby – pomocí metody a pomocí operátorů:

```
# sjednocení metodou
>>> xs = {'tři', 1, 2.0, }
>>> xs.update( {'four', 5, 1, 'tři'} )
>>> xs
{'four', 1, 2.0, 'tři', 5}

# sjednocení operátorem
>>> xs = {'tři', 1, 2.0, }
>>> xs |= {'four', 5, 1, 'tři'}
>>> xs
{'four', 1, 2.0, 'tři', 5}
```

Obdobné metody i operátorové zkratky máme k dispozici i pro další množinové operace – *průnik*, *rozdíl* a *symetrický rozdíl*:

```
# A1) společný průnik množin; operátorově
>>> xs = {'tři', 1, 2.0, }
>>> xs &= {'four', 5, 'tři', 1} & {'tři'}
>>> xs
{'tři'}
# A2) společný průnik množin; metodou
>>> xs = {'tři', 1, 2.0, }
>>> xs.intersection_update( {'four', 5, 'tři', 1}, {'tři'} )
>>> xs
{'tři'}

# B1) rozdíl množin; operátorově
>>> xs = {'tři', 1, 2.0, 'four', 5, 6.0, 'sedm', }
>>> xs -= {'four', 5} | {2.0}
>>> xs
{1, 6.0, 'sedm', 'tři'}
# B2) rozdíl množin; metodou
>>> xs = {'tři', 1, 2.0, 'four', 5, 6.0, 'sedm', }
>>> xs.difference_update( {'four', 5}, {2.0} )
>>> xs
{1, 6.0, 'sedm', 'tři'}

# C1) symetrický rozdíl množin; operátorově
>>> xs = {1, 'dva', 3.0, 'four', }
>>> xs ^= {1, 2.0, 'tři', 'four'}
>>> xs
{2.0, 3.0, 'tři', 'dva'}
# C2) symetrický rozdíl množin; metodou
>>> xs = {1, 'dva', 3.0, 'four', }
>>> xs.symmetric_difference_update( {1, 2.0, 'tři', 'four'} )
>>> xs
{2.0, 3.0, 'tři', 'dva'}
```

Generátorová notace

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

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

→ Jako u seznamu vnitřní závorky zde označují nepovinnou část výrazu. Důležité jsou složené závorky vnější, které (spolu s identifikátorem *HODNOTA*) určují, že jde o generátor množiny. První část *HODNOTA* může (ale také nemusí) být funkcí prvků procházené sekvence.

Jeden příklad přímo z dokumentace:

```
>>> xs = {x for x in 'abracadabra' if x not in 'abc'}
>>> xs
{'r', 'd'}
```

→ Tedy: Z uvedeného řetězce '*abracadabra*' vybereme zástupce písmen (tj. každé písmeno právě jednou), ale pouze takové, které se nevyskytují v řetězci '*abc*'.