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 – Boolean, priority operátorů

Booleovské operátory

Pravdivostní operátory `and` a `or` v Python'u používají zkrácené vyhodnocování, tzn. že je-li možné určit hodnotu výrazu po vyhodnocení jeho prvního operandu, druhý operand není vůbec uvažován. Souhrnně tedy platí:

- `x or y` : je-li `x` nepravdivé, vrať `y`, jinak vrať `x`
- `x and y` : je-li `x` pravdivé, vrať `y`, jinak vrať `x`

Algoritmus vyhodnocování je vskutku takovýto – tyto operace vrací právě jeden ze svých operandů, nikoli booleovskou hodnotu (ale viz následující slajd):

```
>>> a = []     # tedy False
>>> b = [1]    # tedy True

>>> a and b
[]                   # tedy False
>>> a or b
[1]                  # tedy True
```

K dispozici je samozřejmě i operátor negace `not`. Má vyšší prioritu než `and` a to má zase vyšší prioritu než `or`.

Jelikož prakticky všechny ostatní operátory mají prioritu vyšší než operátory pravdivostní, tak je závorkování nejen dobrým nápadem pro lepší čitelnost.

Pravdivostní hodnoty objektů

Ačkoli je kvůli čitelnosti dobrým zvykem vypisovat podmínky v plné formě, často můžeme využít toho, že libovolný objekt v Python'u (tedy prakticky cokoliv) je možno testovat na pravdivostní hodnotu. Přitom následující hodnoty jsou pokládány za `False`, cokoliv jiného za `True`:

- `None`
- `False`

- nula v libovolném číselném typu (např. `0`, `0.0`, `0j`)
- libovolná prázdná sekvence (např. `''`, `()`, `[]`)
- libovolné prázdné mapování (např. `{}`)
- instance uživatelem zavedené třídy, která definuje „magickou“ metodu `__bool__`, resp. `__len__`, vrací-li tato metoda boolovské `False`, resp. celočíselnou `0`

Poznámka: Python sám od sebe vrací ve většině případů pravdivostní hodnoty ve formě `True/False`, resp. `1/0`. Specifickou výjimkou jsou booleovské operace `or` a `and`, které vrací jeden ze svých argumentů (viz předchozí slajd).

Porovnávání

K dispozici je standardní sada porovnávacích operátorů: `< <= > >= == !=`

→ Ve starších kódech můžete zakopnout o již nepodporovanou variantu nerovnosti `<>`.

Sekvenční typy se porovnávají lexikograficky a rekurzivně:

```
# protože 3 < 4
>>> [1, 2, 3] < [1, 2, 4]
True

# protože ('a', 'b') < ('ab', 'b'), protože 'a' < 'ab'
>>> [1, ('a', 'b'), 3] < [1, ('ab', 'b'), 4]
True

# protože druhá sekvence je „delší“
>>> [1, ('a', 'b'), 3] < [1, ('a', 'b'), 3, 'abcd']
True
```

→ Řazení řetězců používá *code point* jednotlivých znaků v *Unicode*.

Pro objekty máme navíc k dispozici test jejich (ne)identity `is` a `is not`:

```
>>> x = y = ['a']
>>> x == y
True
>>> x is y
True

>>> x = ['b']
>>> y = ['b']
>>> x == y
True
>>> x is y
False
```

Pro **sekvenční typy** je v Python'u ještě dvojce operátorů zjišťujících (ne)výskyt hodnoty v sekvenci `in` a `not in`:

```
>>> xl = 'abcdefgh'
>>> 'd' in xl
True
>>> 'd' not in xl
False
```

Tabulka priorit operátorů

operátor	popis
`lambda`	lambda-výraz
`or`	booleovské OR
`and`	booleovské AND
`not`	booleovské NOT
`in, not in, is, is not, <, <=, >, >=, !=, ==`	test na přítomnost hodnoty v sekvenci, test na identitu objektů, porovnávání
`|`	bitové OR
`^`	bitové XOR
`&`	bitové AND
`<< >>`	levý a pravý posun
`+ -`	sčítání, odčítání
`* / // %`	násobení, dělení, modulo
`-x +x ~x`	unární minus a plus, bitové NOT
`**`	umocňování

`x[index], x[index:index], x(arguments...), x.attribute`	hodnota, výřez, volání, odkaz na atribut
`(expressions...), [expressions...], {key:datum...}`	„výroba“ nebo zobrazení tuplu, zobrazení seznamu či slovníku