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 – Funkce I

Úvod

Funkci v Python'u zavedeme pomocí klíčového slova `def`:

```
>>> def nic_nedělající_funkce():
...     pass
...
>>> nic_nedělající_funkce()
>>>
```

→ Klíčové slovo `pass` je prázdná operace. Jelikož odsazování slouží v Python'u k určení těla bloku, použijeme `pass` často právě na místě, kde teprve kód dopíšeme.

Trošku rozumnější příklad:

```
def celociselne_deleni_se_zbytkem(co, cim):
    """Vrať výsledek celočíselného dělení a též zbytek po něm."""
    return co // cim, co % cim

>>> vysledek = celociselne_deleni_se_zbytkem(13, 4)
>>> vysledek
(3, 1)
```

Dokumentační řetězce

Je dobrým zvykem na prvním místě těla funkce uvádět tzv. *dokumentační řetězec*. Má-li více než jednu řádku, je jeho ustálená podoba následující:

```
def f():
    """
    SUBJECT
    ↵
    BODY
    """
```

Tyto *dokumentační řetězce* jsou součástí pythonovského kódu a dá se na ně programově dostat (čehož využívají některé nástroje a mimo jiné vestavěná funkce `help`):

```
>>> def f():
...     """
...     Nedělej nic, ale zdokumentuj to.
...
...     Skutečně, tato funkce nic nedělá. Ale její dokumentační čá
...     více než jednu řádku.
...     """
...
>>> print( f.__doc__ )

        Nedělej nic, ale zdokumentuj to.

        Skutečně, tato funkce nic nedělá. Ale její dokumentační čá
        více než jednu řádku.
```

Zatímco *dokumentační řetězec* je částí kódu a můžeme ho tudíž použít mimo jiné pro vytvoření jinak (zatím) prázdného těla funkce (tedy místo `pass`), s komentářem tomu tak není:

```
>>> def f():
...     # prázdná funkce
...
  File "<stdin>", line 3

    ^
IndentationError: expected an indented block
```

Komentář z hlediska interpretru neexistuje – místo bloku jsme tak vlastně nenapsali nic:

```
>>> def f():
...
  File "<stdin>", line 2

    ^
IndentationError: expected an indented block
```

> → Jinými slovy – tam, kde je očekáván blok, musí blok být! Byť by byl vyroben právě jen pomocí `pass` nebo *dokumentačního řetězce*.

Návratová hodnota funkce

V Python'u všechny funkce, které explicitně nic nevrací, vrací `None`:

```
>>> def f1():
...     pass
...
>>> f1()
>>> print( f1() )
None

>>> def f2():
...     return
...
>>> f2()
>>> print( f2() )
None
```

→ V rámci výstupu interpretru je toto `None` potlačeno, pokud si ho explicitně pomocí `print` nevyžádáme.

Pro srovnání zde je chování při explicitním vrácení jen a pouze právě *None*:

```
>>> def f3():
...     return None
...
>>> f3()
>>> print( f3() )
None
```

Jinak ovšem funkce v Python'u mohou vracet prakticky cokoliv:

```
def f1():
    return "Ahoj, světe!"

def f2():
    return 2+2, "Baf!", [1, 'a', 3]
```

Argumenty poziční

Funkce mohou mít argumenty. Ty se standardně předávají pozičně:

```
def f(a, b):
    print( "Argument 1: ", a )
    print( "Argument 2: ", b )

>>> f(1, 2)
Argument 1: 1
Argument 2: 2

>>> f('ahoj', 333)
Argument 1: ahoj
Argument 2: 333

>>> f('ahoj', (333, 'svět') )
Argument 1: ahoj
Argument 2: (333, 'svět')
```

→ Jak je vidět, do argumentu může vstupovat libovolný pythonovský objekt.

Argumenty pojmenované

Kromě *pozičních* argumentů můžeme s výhodou využít argumenty *pojmenované*. Zadávají se ve formě *keyword* = *value* a v nejjednodušším (a nejméně užitečném) případě slouží „pouze“ pro zpřehlednění volání funkce:

```
def celociselne_deleni_se_zbytkem(co, cim):
    return co // cim, co % cim

>>> celociselne_deleni_se_zbytkem(co = 13, cim = 4)
(3, 1)
```

Užitečnějším použitím jsou výchozí (*defaultní*) hodnoty argumentů:

```
def mocneni(cislo, exponent=2):
    return cislo**exponent

>>> mocneni(3)
9
>>> mocneni(3, 3)
27
>>> mocneni(3, exponent=3)
27
>>> mocneni(cislo=3, exponent=3)
27
```

Ale pozor – otočit pořadí pozičních a pojmenovaných argumentů už nemůžete:

```
>>> mocneni(cislo=3, 3)
  File "<stdin>", line 1
SyntaxError: non-keyword arg after keyword arg
```

Uvedený „trik“ používá spousta zabudovaných funkcí, např. funkce `round` pro zaokrouhlování reálných čísel na uvedený počet míst za desetinnou čárkou – neuvedete-li počet míst, výchozí hodnotou je 0:

```
>>> x = 3.141592753
>>> round(x)
3
>>> round(x, 2)
3.14
```

→ Výchozí hodnotou pojmenovaného argumentu může být i odkaz na proměnnou, ale tyto odkazy vyhodnucují ve chvíli definice funkce v aktuálním *defining scope*. Dalším „průšvihem“ je, pokud jako parametr použijete proměnný (*mutable*) typ, ale to probereme až později.