



Jython: Funkcie a spôsoby odovzdávania parametrov

Jython, hoci je objektovo orientovaný, nepodporuje preťažovanie funkcií. Toto je vynahradené širokou paletou možností, ako odovzdávať parametre. Tieto ďalej opísané vlastnosti Java nepodporuje.

Funkcie s nepovinnými argumentmi

Okrem štandardného spôsobu definovania funkcie možno funkciu definovať s premenným počtom argumentov tak, že pre nepovinné argumenty nadefinujeme ich defaultné hodnoty.

```
def fceNepovArg(x, a=2, b='tretí'):
    print x, a, b
```

Spôsoby volania definovanej funkcie sú nasledujúce:

```
>>> fceNepovArg(0)
0 2 tretí
>>>
>>> fceNepovArg(0, 111)
0 111 tretí
>>>
>>> fceNepovArg(0, 'Ahoj')
0 Ahoj tretí
>>>
>>> fceNepovArg(0, a='Cau')
0 Cau tretí
>>>
>>> fceNepovArg(0, b='III')
0 2 III
>>>
>>> fceNepovArg(x=0, a=123, b='III')
0 123 III
```

Výrazy definujúce defaultné hodnoty nepovinných parametrov sa nevyhodnocujú pri každom volaní funkcie, ale iba raz, a to vtedy, keď sa vykonáva príkaz definície funkcie.

```
>>> a = 1
>>>
>>> def f(x = a):          #zaciatok definície
    funkcie
    ...     print x
    ...
>>>
>>> f()                   #pouzijem defaultnu
    hodnotu
1
>>>
>>> f(123)               #zadam svoju hodnotu
123
>>>
>>> a = 999
>>> f()                   #defaultna hodnota
    pouzije povodnu hodnotu
1
>>>
```

Voliteľný zoznam argumentov

Pokiaľ je zoznam argumentov ukončený argumentom, ktorý obsahuje ako prvý znak *, napríklad *volitelny, potom argument dostane tuple obsahujúci všetky zvyšujúce parametre pri volaní funkcie:

```
>>> def f(x, a='druhý', *volitelne):
    ...     print x, a, volitelne
    ...
>>>
>>> f(1)
1 druhý ()
>>>
>>> f(1,2)
1 2 ()
>>>
>>> f(1,2, 3, 4)
1 2 (3, 4)
>>>
>>> f(1, 'backora', 999, 'koniec')
1 backora (999, 'koniec')
```

Definícia funkcie, ktorá môže obsahovať ľubovoľný počet argumentov, ale nemusí mať žiadny, sa môže začínať takto:

```
>>> def fce(*params):
    ...     print params
    ...
>>> fce()
()
>>>
>>> f('a', [1,2,3], {1: "ichi", 2: "ni"})
('a', [1, 2, 3], {2: 'ni', 1: 'ichi'})
>>>
```

Konečná syntax definície funkcie je nasledujúca:

```
def nazovFunkcie([ arg, ..., *zbytok, **argKlicSlov ]):
    blok
```

Ďalší príklad osvetlí význam posledného typu parametra. Pokiaľ ho chceme použiť, **musí** nasledovať za voliteľným parametrom *zbytok.

```
>>> def f(a, b=0, c='ahoj', *d, **e):
    ...     print a, b, c, d, e
    ...
>>> f(1)
1 0 ahoj () {}
>>>
>>> f(1, 111)
1 111 ahoj () {}
>>>
>>> f(1, 2, 3, 4, 5, 6)
1 2 3 (4, 5, 6) {}
>>>
>>> f(1, 2, 3, typ='Wokenice', program='Wod
    Bila')
1 2 3 () {'program': 'Wod Bila', 'typ': 'Wo-
    kenice'}
>>>
>>> f(1, 2, 3, 4, 5, typ='Wokenice', pro-
    gram='Wod Bila')
1 2 3 (4, 5) {'program': 'Wod Bila', 'typ':
    'Wokenice'}
```

Funkcionálne programovanie

Ukážky kódu, s ktorými sme sa doposiaľ stretávali, boli písané spôsobom blízky procedurálnemu programovaniu. Na Jython je sympatické, že vám nevnucuje, akým spôsobom máte svoje programy písať. Môžete používať procedurálny, objektovo orientovaný či funkcionálny spôsob.

Lambda výrazy

Kľúčové slovo lambda vám umožní vytvárať malé anonymné funkcie. Termín lambda výraz je výstižný, pretože lambda výraz nesmie obsahovať príkazy. Syntax lambda výrazu je nasledujúca:

```
lambda zoznam_parametra: vyraz
```

Ukážme si rozdiel medzi procedurálnym a funkcionálnym spôsobom naprogramovania funkcie, ktorá nám povie, či jej argument je párne alebo nepárne číslo:

```
>>># proceduralny sposob programovania
>>>
>>> parne_neparne(cislo):
    ...     if cislo%2:
    ...         return 'neparne'
    ...     else:
    ...         return 'parne'
    ...
>>> parne_neparne(6)
'parne'
>>>
>>> parne_neparne(3)
'neparne'
>>>
>>>#####
>>># vyuziti lambda vyrazu
>>>
>>> parne_neparne = lambda cislo: cislo % 2
    and "neparne" or "parne"
>>>
>>> parne_neparne (6)
'parne'
>>> parne_neparne (3)
```

```
'neparne'
>>>
>>>
>>>#####
>>>##
>>>#co mozno tiez zapisat takto
>>>
>>> parne_neparne (cislo):
    ...     return cislo % 2 and "neparne" or
    "parne"
    ...
>>> parne_neparne (6)
'parne'
>>> parne_neparne (3)
'neparne'
>>>
```

Zoznam argumentov pre lambda výrazy má rovnaké vlastnosti ako v prípade používateľsky definovaných funkcií.

```
>>>#funkce vrati definici funkce, ktoru po-
    uziju dalej
>>> def objemValca(vyska):
    ...     return lambda r, pi=3.14,
    vyska=vyska: pi*r**2 * vyska
    ...
>>> #zaujimaju ma objemy valca s roznyh polo-
    merom a s vyskou 10
>>> objemy_pri_vyske_10 = objemValca(10)
>>>
>>> for r in range(1, 6):
    ...     print 'polomer:', r, 'objem:', ob-
    jemy_pri_vyske_10(r)
    ...
polomer: 1 objem: 31.400000000000002
polomer: 2 objem: 125.60000000000001
polomer: 3 objem: 282.6
polomer: 4 objem: 502.40000000000003
polomer: 5 objem: 785.0
>>>
```

Pretože lambda výrazy nemôžu obsahovať príkazy (ako napr. if/else, ...), často sa v nich používajú operátory and/or na implementáciu logiky vyhodnocovania výrazov.

```
>>> faktorial = lambda cislo: cislo==1 or
    cislo * faktorial(cislo-1)
>>> faktorial(5)
120
```

Lambda funkcia je účinný nástroj v spolupráci s ďalšími funkciami definovanými v Jython. Príkladom môže byť funkcia filter, ktorú možno výhodne použiť pri práci so zoznamami. Jej syntax je takáto:

```
filter(nazov_funkcia, zoznam)
```

Túto funkciu všeobecne použijete tam, kde potrebujete zo zoznamu vybrať prvky na základe určitého kritéria.

```
>>> #parne cisla zo zoznamu
>>> filter(lambda x: x%2 == 0, range(10))
[0, 2, 4, 6, 8]
```

Na záver uvedieme pekný príklad použitia funkcie filter, ktorého úloha je nájsť prienik dvoch množín:

```
>>> set1 = range(0,200, 7)
>>> set2 = range(0,200, 3)
>>>
>>> filter(lambda x: x in set1, set2)
[0, 21, 42, 63, 84, 105, 126, 147, 168, 189]
```



■ ŠTEFAN HAVLÍČEK,
Sales engineer, InterSystems B. V.