Funkce

Tento text si v žádném případě neklade za cíl být podrobným popisem funkcí v jazyku Python. Python totiž nabízí mj. mnohem více způsobů práce s parametry funkcí než tento text popisuje. V tomto textu je kladem důraz spíše na pochopení podstaty funkcí, na vysvětlení globality a lokality proměnných i na předávání argumentů.

Pojem a podstata funkce

Funkce jsou univerzálně používaný nástroj pro strukturování kódu. Představují a umožňují:

- **Zapouzdření kódu** – umožňují vyjmout a „zabalit“ část kódu, kterou chceme opakovaně používat.
- **Procedurální dekompozici** – umožňují rozdělit program na jasně definované části a těmito částmi pak pracovat jako s celky.

V tradičních programovacích jazycích se používá pro část kódu pojem **podprogram**. Podprogramy se pak dělily na **procedury a funkce.** Procedury se od funkcí liší voláním. Procedury jsou akce s proměnnými, takže se volají jako část kódu a nemají návratovou hodnotu. Funkce se volají podobně jako funkce v matematice, mají návratovou hodnotu, takže mohou být při volání užity na pravé straně přiřazovacího příkazu.

V moderních jazycích se užívají pouze funkce, které zpravidla mají návratovou hodnotu, není to však podmínkou.

Příkazy Pythonu pro práci s funkcemi:

- **definice funkce:** *def, return*
- pro **globální proměnné:** *global*
- **volání funkcí:** identifikátor funkce se skutečnými argumenty v závorce.

Principy definice funkcí v Pythonu:

- **Def vytváří objekt typu funkce a přiřazuje jej jménu –** jméno funkce je referencí na objekt (zde typu funkce).
- **Return vrací návratovou hodnotu.** Return určuje, která proměnná vrací návratovou hodnotu funkce.
- **Argumenty ani typ návratové hodnoty nemusíme deklarovat dopředu.**

Lokální a globální proměnné

Programovací jazyky, které deklarují proměnné, je deklarují jako **lokální**, tedy platné pouze v určité proceduře či funkci, a **globální**, tedy platné v nadřazené proceduře či funkci, popř. platné v celém programu. V Pythonu se proměnné nedeklarují a jsou vždy chápány jako lokální, tedy platné pouze v příslušné funkci, jedině pokud má proměnná mít platnost i po opuštění funkce, musí být výslovně označena jako globální příkazem ***global*.** Globální je rovněž proměnná, které je přiřazena hodnota mimo funkci a uvnitř funkce je užita jen na pravé straně přiřazovacího příkazu. Kdyby však tato proměnná byla užita na levé straně přiřazovacího příkazu a nebyla by označena jako globální, byla by globální proměnná zastíněná lokální. Takže platí zásady, že:

- **modul je globální obor platnosti**
- **každé volání funkce je vytvořením nového lokálního oboru platnosti**

- **bez global jsou námi definovaná jména vždy jen lokální**
- **všechna ostatní jména v lokálním oboru** (tedy nedefinovaná v rámci funkce a nenamapovaná přes global) **by měla být globální nebo standardní,** pokud nejsou, Python je nenajde a ohlásí chybu.

Příklad:

```
# globalni obor platnosti
X = 99              # X a funkce v globalnim prostoru platnosti

def funkce(Y):      # Y a z prirazujeme v ramci funkce
    # lokalni obor platnosti
    z = X + Y
    return z

funkce(1)           # volani funkce vraci hodnotu 100
```

Globální: X, funkce

X, je globální jméno definované na nejvyšší úrovni modulu. Bez deklarace global lze X ve funkci používat jen na pravé straně přiřazovacího příkazu. Jméno funkce je definováno na nejvyšší úrovni modulu.

Lokální: Y, z

Obě jména jsou definována v těle funkce takže přestanou existovat v okamžiku ukončení běhu funkce. z proto, že je definováno prostým přiřazením a X díky tomu, že argumenty jsou předávány přiřazením (viz dále).

Předávání argumentů

Vedle globality a lokality proměnných řeší programovací jazyky i **předávání argumentů procedurám.** Rozlišujeme **argumenty:**

- **formální –** jsou zapsány v závorce v záhlaví definice funkce a jejich skutečná hodnota je jim dodána teprve při volání jedním ze dvou základních způsobů popsaných v následujícím odstavci.
- **skutečné** – jsou uvedeny v závorce příkazu volání procedury nebo funkce**.**

V zásadě existuje předávání argumentů hodnotou a odkazem. **Předání hodnotou** znamená, že pokud předáváme hodnotu nikoli konstantou (literálem), ale proměnnou, předáváme pouze *obraz* obsahu proměnné a procedura nemůže měnit obsah původní předávané proměnné. **Předání odkazem** znamená, že proměnná je dána proceduře *„k dispozici“* a ta může měnit obsah proměnné.

V Pythonu:

- **Argumenty jsou předávány přiřazením (odkazem na objekt).** Podstatou je, že volající kód i kód funkce sdílí jeden objekt přes dvě různá jména.
- **Můžeme přiřazovat formálním argumentům výchozí hodnoty** (formální argumenty jsou proměnné zapsané v závorce definované funkce).

Jak již bylo řečeno na začátku, Python je velice mocný a variabilní ve způsobech předávání argumentů, existuje ještě několik dalších způsobů, které zde nejsou popsány.

Příklad:

```
# Vypocet faktorialu -- iteracne (klasicky)
```

```
def faktorial(n):
    fakt = 1.0
    for i in range(2, n+1):      # Je to do N vcetne, proto +1
      fakt = fakt * i
    return fakt

print "10! = ", faktorial(10)
print "66! = ", faktorial(66)
```

Lambda funkce

Lze užívat i nepojmenované funkce lambda. Tyto funkce lze definovat na místě použití, takže odpadne nutnost deklarace funkce na jiném místě, než kde se funkce použije. Tělem této funkce musí být jediný výraz (ne blok příkazů). Lambda má tvar:

lambda argument1, argument2, …, argumentN : výraz

Příklad:

```
 >>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
 >>> f(1)
43
```

Funkce lambda bývá zpravidla na pravé straně přiřazovacího příkazu. V tomto případě je návratovou hodnotou funkce make_incrementator(n).

Literatura:

[1] Rubeš, J.: Nebojte se programovat, ComputerMedia, Bedihošť 2001
[2] Lutz, M., Ascher, D.: Naučte se Python, Grada, Praha 2003
[3] Beazley, D. M.: Python, Neocortex, Praha 2002
[4] Python Reference Manual
[5] Švec, J.: Létající cirkus, Python tutoriál, 2003