Funkce.

Funkce, metody. Předávání parametrů. Mutable/Immutable objekty. Poziční argumenty.

Tomáš Bayer | bayertom@fsv.cvut.cz

Katedra geomatiky, fakulta stavební ČVUT.

Obsah přednášky

1. Procedura, funkce, metoda

Samostatná část programu konající specializovanou funkci.
Umístěny mimo hlavní program.
Lze opakovaně spouštět z hlavního programu či jiného podprogramu.

Tvořeny:

- procedurami,
- funkcemi,
- metodami.

Výhoda použití:
Zvýšení přehlednosti a čitelnosti programu.
Urychlují vývoj a ladění programu.
Možnost opakovaného provádění výpočtů.

Volání (spuštění):
Volány se seznamem parametrů, kterým předány hodnoty potřebné pro výpočet.
Většinou vrací nějaký výsledek, tzv. *návratová hodnota*.
Avšak nemusíme předávat žádné údaje popř. žádné výsledky nevrací.

Procedura vs. funkce vs. metoda:
Některé jazyky nerozlišují funkce/procedury.
Procedura nevrací výsledek, funkce ano.
Metoda obdobou procedury/funkce v OOP.

2. Funkce a její deklarace

Před použitím funkce nutná její deklarace.
Tvořena hlavičkou funkce a tělem funkce.

```
def jmeno_funkce(form_param1, form_param2,...):
    telo funkce
```

Hlavička funkce:
Jméno funkce, typ návratové hodnoty, seznam formálních parametrů (FP).
V Pythonu netřeba uvádět datové typy (avšak můžeme, Type Hints).
Python neumí přetěžovat funkce.

Tělo funkce:
Představuje blok.
Obsahuje výkonný kód funkce.

Jméno (identifikátor) funkce zpravidla psáno malými písmeny.
Voleno tak, aby vyjadřovalo funkcionalitu.

Podpora Type Hints:
Lze specifikovat typy formálních parametrů i návratový typ, kontrola typů.

```
def jmeno_funkce(form_param1 : typ, form_param2 : typ,...)->typ:
```

Pouze informační charakter.

3. Volání funkce

= spuštění funkce, tvoří ho:

- jméno funkce,
- seznam skutečných parametrů (SP).

Funkce bez návratové hodnoty:
Volání funkce

```
funkce(sp1, sp2,...)     #Pass n parameters
```

V některých jazycích deklarace funkce musí předcházet volání.
V Pythonu nehraje roli.

Funkce s návratovou hodnotou/hodnotami:
Funkce v Pythonu schopna vrátit *více hodnot,* netypické pro většinu programovacích jazyků.
Volání součástí přiřazovacího příkazu

```
prom1, prom2, prom3 = funkce(sp1, sp2, sp3...)  #Return 3 parameters
prom = funkce(sp1, sp2, sp3,...)                #Return n parameters as tuple
```

V těle funkce klíčové slovo `return`.
Za ním uvedena/y hodnota/y vracená/é funkcí.

```
def addsubtract(a, b):          #Declaration
   return a+b, a-b              #Return addition/multiplication of 2 variables
   a=7                          #Unreachable code
..
x1 = 5                          #First variable
x2 = 7                          #Second variable
x3, x4 = addsubtract(x1, x2)    #Function call and assignment
>>> 12
```

Za klíčovým slovem `return` nesmí následovat žádný kód.
Kód byl by nedostupný ⇒ unreachable kód.

4. Funkce main()

Funkce volána při spuštění programu automaticky (C/C++, Java).

Můžeme jí předávat argumenty z příkazového řádku.
Nemá žádnou návratovou hodnotu.

Výhodou lepší organizace a čitelnost kódu.

```
def main(args):  #Arguments from command line
    pass
```

Proměnné deklarované uvnitř `main()` jsou lokální.
Proměnné deklarované vně `main()` jsou globální (Python).

```
if __name__ == '__main__':
    main(sys.argv)
```

Soubor test.py lze používat dvěma způsoby:

1. Import ve formě modulu:
   Pak `__name__= test` a blok se nevykoná.
   ```
   import test
   ```
2. Přímé spuštění:
   Pak `__name__= __name__` a blok se vykoná.
   ```
   python test.py
   ```

5. Formální a skutečné parametry funkce

Skutečné parametry:
Použity při volání funkce.
Jejich hodnoty předávány formálním parametrům.

Formální parametry:
Lokální proměnné deklarovány v hlavičce funkce.
Vznikají v okamžiku volání funkce.
Zanikají při ukončení běhu funkce.

FP stejného datového typu jako SP.
Jinak provedena implicitní konverze (nelze -li, chyba).

Dva způsoby předávání formálních parametrů:

- *hodnotou (Pass by Value)*
  Předávána kopie hodnoty FP.
  V Pythonu podporováno.
- *odkazem (Pass by Reference)*
  Předávána reference (tj. “originální” proměnná).
  Python nepodporuje.
  Lze “obejít”.

6. Předávání parametrů hodnotou

FP kopií SP.
Jakákoliv změna hodnoty FP neovlivní hodnotu SP.
Originální data nelze ve funkci “přepsat” (read only).

V Pythonu FP předávány **pouze hodnotou**.
Předávání hodnot SP do FP prováděno při volání metody.
Hodnoty jsou předávány *postupně:*

Hodnotě 1. FP předána hodnota 1. SP.
Hodnotě 2. FP předána hodnota 2. SP.

```
form_param1=skut_param1;
form_param2=skut_param2;
```

...

V Pythonu trochu komplikovanější.
Odlišné chování pro immutable/mutable objects.

Mutable objects: mohou být modifikovány v těle funkce.
Immutable objects: nemohou být modifikovány v těle funkce.

7. Immutable/Mutable objekty

V Pythonu vše objektem.
Každý objekt má unikátní identifikátor.
Zjištění identifikátoru objektu: příkaz `id()`

```
x=1
id(x)
>>> 1613424592
```

Mutable objekty:
Mohou měnit stav/obsah.
V Pythonu: `list, set, dict`.

Immutable objekty:
Nemohou měnit stav/obsah.
V Pythonu vše ostatní: `int, float, complex, string, tuple, frozen set`.
Při pokus o “modifikaci” immutable vytvořen nový objekt.

```
x=x+1
id(x)
>>> 1613424608          #ID has changed, x modified (immutable)
```

U mutable objektů nenastává:

```
L=[1, 2, 3]
id(L)
>>> 22010856
L[0]=3
>>>id(L)
>>> 22010856            #ID unchanged, L modified (mutable)
```

Odlišné chování těchto objektů jako parametrů funkcí.

8. Ukázka předávání SP do FP

Ukázka1: Funkce `dist()`, formální parametry x1, y1, x2, y2, IO,

```
def dist (x1, y1, x2, y2):              #Evaluate distance
   dx = x2 - x1                         #Coordinate diff. for x
   dy = y2 - y1                         #Coordinate diff. for y
   return (dx*dx + dy*dy)**0.5          #Return distance
```

Použití TypeHint:

```
def dist (x1 : float, y1 : float, x2 : float, y2 : float)->float:
```

Volání funkce:

```
xa = 0, ya = 0, xb = 10, yb = 10;
d  = dist(xa, ya, xb, yb);
```

Přiřazení FP do SK:

```
provede se: x1 = xa, y1 = ya, x2 = xb, y2 = yb;
```

Ukázka 2: předání seznamu jako parametru, MU:

```
def f (L):                              #Pass list
   L.append(1)                          #Append item to the list
   L.append(2)                          #Append item to the list
```

Volání funkce:

```
S = []                                  #Create empty list
f(S)                                    #Call function f
...
```

Přiřazení FP do SP:

```
provede se: L = S
```

9. Předání IO/MO podrobněji...

IO/MO objects jako formální parametry předávány hodnotou.

Immutable objekty:
Při pokusu o modifikaci IO v těle funkce se vytvoří jeho kopie.
Jeho změna neovlivní původní objekt.

```
def f(x):
   print(id(x))

...
x = 1
print(id(x))
f(x)
print(x)
>> 1946910832
>> 1946910832
>> 1
```

```
def f(x):
   x=3              #Local copy
   print(id(x))     #New ID

x = 1
print(id(x))
f(x)                #x can not be modified inside f
print(x)
>> 1946910824
>> 1946910832
>> 1                #Value has not changed
```

Mutable objekty:
Mohou měnit stav/obsah.
V těle funkce mohou být modifikovány.

```
def f(L):
   print(id(L))

.
L = [1, 2]
print(id(L))
f(L)
print(L)
>> 1946910817
>> 1946910817
>> [1, 2]
```

```
def f(L):
   L[0]=3           #Modify object
   print(id(l))     #ID has not changed

L = [1, 2]
print(id(L))
f(L)                #L was modified inside f
print(L)
>> 1946910817
>> 1946910817
>> [3, 2]           #Values have changed
```

10. Předávání parametrů odkazem

Při předávání nevzniká kopie skutečného parametru.
Formální i skutečný parametr pak odkazují na stejnou proměnnou.
V Pythonu nepodporováno, lze však "obejít" s mutable objekty.

Změna hodnoty FP uvnitř fce změní hodnoty SP.
Možnost "přepsání" hodnot volajících dat.
Menší nároky na HW, netřeba vytvářet kopie objektů.

Praktická realizace: "obejít" lze předání parametrů v seznamu.
Ve skutečnosti opět předáváme referenci, modifikujeme prvky seznamu.

Ukázka: prohození dvou čísel (využití mutable)

```
def function swap(x):
    temp = x[0]          #Create temp. variable, store first
    x[0] = x[1]          #Copy second to first
    x[1] = temp          #Copy first
```

Volání funkce:

```
a = [1, 2]               #Create list of 2 items
swap(a)                  #Swap items
```

11. Výchozí hodnoty parametrů

FP mohou mít definovány výchozí hodnoty.
Ukázka:

```
def dist1d (dx, dy = 0):       #1 FP is default
   return (dx * dx + dy * dy)**0.5
```

Ukázka:

```
def dist2d (dx = 0, dy = 0):  #2 FPs are default
   return (dx * dx + dy * dy)**0.5
```

Funkce lze spustit s:

1. hodnotami skutečných parametrů
   Mají vyšší prioritu než výchozí hodnoty FP.

   ```
   d = dist1d(5, 6) #dx = 5, dy = 6
   d = dist2d(5, 6) #dx = 5, dy = 6
   ```

2. výchozími hodnotami formálních parametrů
   SP dosazeny za FP:

   ```
   d = dist2d()     #dx = 0, dy = 0
   ```

3. Kombinace 1) + 2)

   ```
   d = dist1d(5)    #dx = 5, dy = 0
   d = dist2d(5)    #dx = 5, dy = 0
   ```

12. Předávání funkcí jako parametrů

Funkce mohou být předávány jako parametry jiné funkci.
Předaná funkce volána se stejným počtem FP (podobný princip jako u proměnných).
Tento mechanismus není běžný u jiných programovacích jazyků (C++, pointer to function).

Příklad: Výpočet kombinačního čísla:

$$C = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{(n-k)!k!}.$$

Faktoriál:

```
def fact(n):          #fact has one parameter
   f = 1
   while n > 1:
      f *= n
      n -= 1
   return f
```

Kombinační číslo:

```
def comb(n, k, f)
   return f(n) / (f(n - k) * f(k))   #f has 1 parameter as fact
```

Volání:

```
c=comb(5, 3, fact)
```

13. Poziční argumenty

V Pythonu může být libovolný počet FP zastoupen

```
*args     **kwargs
```

Používáno v případě, kdy není znám počet FP.
Po předání vytvořena *n*-tice * nebo slovník **.

```
def sum(*nums):
    s = 0
    for num in nums:
        s += num
    return s
```

```
def write(**items)
    for it in items:
        print(it, items[it])  #Key, value
```

Volání funkce:

```
s = sum(1, 2, 3, 4)                                  #Add 4 numbers
write (name='Jan', surname='Novak', age=25)   #Print database
```

Poziční argumenty mohou být kombinovány s "obyčejným"i:

```
def f(a, b=100, *c, **d):
    ...
```

Volání funkce:

```
f(1, 2, 3, 4, e=5, f=6, g=7)
```

Pak:

```
>>a=1, b=100, c = (2,3,4), d = {e:5, f:6, g:7}
```