Funkce

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMI/ZPP1 Základy programování v Pythonu 1

Funkce

Motivace

- Máme zadány známky a potřebujeme spočítat průměr:

```
znamky=[[1,1.0],2,1,3,1]

soucet=0
pocet=0
for i in znamky:
    soucet+=i
    pocet+=1
prumer=soucet/pocet

print("Průměr známek je {:.2f}.".format(prumer))
```

Motivace

- Každá známka má svou váhu. Potřebujeme tedy spočítat vážený průměr:

```
znamky=[[1,1.0],[2,0.8],[1,0.8],[3,1.0],[1,0.75]]

soucet=0
pocet=0
for i in znamky:
    soucet+=i[0]*i[1]
    pocet+=i[1]
prumer=soucet/pocet

print("Průměr známek je {:.2f}.".format(prumer))
```

Motivace

- Máme známky z více předmětů:

```
znamky_CJ=[[1,1.0],[2,0.8],[1,0.8],[3,1.0],[1,0.75]]
znamky_M=[[3,1.0],[1,0.6],[2,0.9],[4,0.5],[1,1.0]]

soucet=0
pocet=0
for i in znamky_CJ:
    soucet+=i[0]*i[1]
    pocet+=i[1]
prumer=soucet/pocet

print("Průměr známek z ČJ je {:.2f}.".format(prumer))
```

Motivace

```
soucet=0
pocet=0
for i in znamky_M:
    soucet+=i[0]*i[1]
    pocet+=i[1]
prumer=soucet/pocet

print("Průměr známek z M je {:.2f}.".format(prumer))
```

- Při x předmětech musíme kód pro výpočet váženého průměru x-krát opakovat.
- Chceme tedy náš kód vykonat vícekrát s různým vstupem.
- Za tímto účelem by bylo vhodné mít možnost si část programu pojmenovat, určit co jsou vstupní proměnné a co je výstupní hodnota.
- Právě k tomu slouží uživatelské funkce.

Funkce

- osamostatněné části programu
- „komunikují“ s jinými funkcemi prostřednictvím volání těchto funkcí
- při volání funkce dojde k provedení příkazů jejího těla
- parametry funkce = vstup funkce
- argumenty funkce = reálné hodnoty parametrů při volání funkce
- návratová hodnota = výstup funkce

Definice funkce

- syntaxe:

```
def jméno_funkce(parametr_1, parametr_2, ..., parametr_n):
    příkazy
```

  - `jméno_funkce` představuje identifikátor funkce (formálně se jedná o proměnnou, která obsahuje referenci na objekt reprezentující, danou funkci),
  - `parametr_1, . . . , parametr_n` jsou parametry funkce = proměnné, jež jsou přístupné v těle funkce,
  - tělo funkce, tvořené `příkazy`, je blokem kódu a musí být korektně odsazeno.

Motivace

- Vytvoříme funkci:

```
def vazenyPrumer(znamky):
    soucet=0
    pocet=0
    for i in znamky:
        soucet+=i[0]*i[1]
        pocet+=i[1]
    prumer=soucet/pocet
    return prumer
```

Volání funkce a návratová hodnota

- funkci voláme pomocí operátoru volání funkce () uvedeným za identifikátorem dané funkce; uvnitř závorek dále uvedeme argumenty oddělené čárkou
- příklad:

```
mocnina(2,4)
```

- pro opuštění funkce s danou návratovou hodnotou slouží příkaz `return`
- není povinné mít vracet hodnotu z každého místa funkce, ale může to být zdroj chyb
- volání funkce může být součástí složitějšího výrazu, takto lze využít návratovou hodnotu dané funkce

Motivace

- Funkci zavoláme:

```
znamky_CJ=[[1,1.0],[2,0.8],[1,0.8],[3,1.0],[1,0.75]]
znamky_M=[[3,1.0],[1,0.6],[2,0.9],[4,0.5],[1,1.0]]

p=vazenyPrumer(znamky_CJ)
print("Průměr známek je {:.2f}.".format(p))
print("Průměr známek je {:.2f}.".format(vazenyPrumer(znamky_M)))
```

Funkce s vedlejším efektem

- Všechny funkce vracejí hodnotu,
- funkce, u kterých jsme nedefinovali, co mají vracet (neobsahují příkaz return), vracejí hodnotu None,
- volání funkce je chápáno jako výraz, který se při provádění kódu vyhodnotí a na jeho místo se do okolního kódu dosadí obdržený výsledek.

- Vedlejší efekt funkce
  - výsledek akce realizované funkcí, kterou voláme proto, abychom od ní získali hodnotu,
  - je považován za něco, co by se v programu nemělo vyskytovat,
  - optimální by bylo, kdyby všechny funkce byly definovány buď tak, aby jen provedly nějakou akci, anebo tak, aby vrátily nějakou hodnotu bez jakýchkoliv vedlejších efektů.

Příklad

- Nesprávně napsaná funkce s vedlejším efektem:

```
def mocnina(x,n):
    m=x**n
    print(f"Mocnina: {m}")
    return m
```

- Správně napsaná funkce bez vedlejšího efektu:

```
def mocnina(x,n):
    return x**n

cislo=int(input("Zadej číslo:"))
n=int(input("Zadej n:"))
print(f"Mocnina: {mocnina(cislo,n)}")
```

Příklad

```
def tisk(s):
    print("Seznam obsahuje tato čísla: ");
    for i in s:
        print(f"{i},",end=" ")

def soucet(s):
    suma=0
    for i in s:
        suma+=i
    return suma

cisla=[8,4,5,6,7,1,2,9,10,24]
tisk(cisla)
print(f"Součet čísel v je: {soucet(cisla)}")
mocniny=[]
for i in cisla:
    mocniny.append(i**2)

tisk(mocniny)

print(f"Součet mocnin je: {soucet(mocniny)}")
```

Příklad

```
def obsah_obdelniku(s1, s2):
    if(s1>0 and s2>0):
        return s1*s2

a=int(input("Zadej stranu a:"))
b=int(input("Zadej stranu b:"))


print(f"Obsah obdelníku je: {obsah_obdelniku(a,b)}")
```

- Co když změníme hodnotu proměnné v na -8?

Příklad

```
def obsah_obdelniku(s1, s2):
    if(s1>0 and s2>0):
        return s1*s2
    else:
        return "Chybné zadání stran."

a=int(input("Zadej stranu a:"))
b=int(input("Zadej stranu b:"))


print(f"Obsah obdelníku je: {obsah_obdelniku(a,b)}")
```

Příklad

```
def obsah_obdelniku(s1, s2):
    if(s1>0 and s2>0):
        return s1*s2
    else:
        return "Chybné zadání stran."

def obsah_ctverce(s1):
    return obsah_obdelniku(s1,s1)

a=int(input("Zadej stranu a:"))
b=int(input("Zadej stranu b:"))


print(f"Obsah obdelníku je: {obsah_obdelniku(a,b)}")
print(f"Obsah čtverce je: {obsah_ctverce(a)}")
```

Přejmenování vestavěných identifikátorů

- vestavěné identifikátory lze používat jako vlastní identifikátory:

```
range=6
def input(x):
    return x
print (input(range))
```

- přejmenované identifikátory nelze použít v původním významu:

```
>>> a=range(1,10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```

Polymorfismus funkce

- vytvoříme funkci:

```
>>> def times(x,y):
...     return x*y
...
```

- funkci zavoláme s celými čísly:

```
>>> print(times(3,5))
15
```

- ale i s číslem a řetězcem:

```
>>> print(times("ahoj",5))
```

- ale ne se dvěma řetězci (* není definován pro dva řetězce):

```
ahojahojahojahojahoj
>>> print(times("ahoj","světe"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in times
TypeError: can't multiply sequence by non-int of type 'str'
>>>
```

Příklad

```
def prunik(x,y):
    pr=[]
    for a in x:
        if a in y:
            pr.append(a)
    return pr

r=range(1,10)
s=[1,2,3,5,6]
t=[1,[2,3],5,6]
print(prunik(r,s))
[1, 2, 3, 5, 6]
print(prunik(s,r))
[1, 2, 3, 5, 6]
print(prunik(t,s))
[1, 5, 6]
print(prunik(t,"ahoj"))

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in prunik
TypeError: 'in <string>' requires string as left operand, not int
```

Příklad

```
op=input("Zadej operaci:")
x=int(input("Zadej první číslo:"))
y=int(input("Zadej druhé číslo:"))

if op=="+":
    def f(x,y):
        return x+y

if op=="-":
    def f(x,y):
        return x-y

if op=="*":
    def f(x,y):
        return x*y

if op=="/":
    def f(x,y):
        return x/y

print(f"Výsledek operace {x}{op}{y} je {f(x,y)}.")
```

Rozdělení programu do funkcí

Zadání

- Chceme vytvořit program pro loterii.
- Vstupem programu bude:
  - `n` - z kolika čísel se skládá osudí,
  - `p` – kolik čísel se losuje,
  - `schema` – výherní schéma,
  - `losy` – seznam vsazených losů.
- Program by měl vylosovat tažená čísla a vyhodnotit, které losy vyhráli a kolik.
- Příklad vstupu:

```
n = 20
p = 6
schema=[[6,1000],[5,500],[4,100]]
losy= [[1,8,17,14,5,6],[11,18,7,4,15,16],[2,4,5,6,19,20]]
```

Řešení

- Nejdříve vylosujeme tažená čísla:

```
import random

tah=[]
for i in range(p):
    tah+=[random.randint(1,n)]
```

- Potom vyhodnotíme, který los vyhrál a kolik:

```
vyhry=[]
for l in losy:
    spravne=0
    for i in l:
        if i in tah:
            spravne+=1
    v=[[l,0]]
    for s in schema:
        if s[0]==spravne:
            v[1]=s[1]
            break
    vyhry+=v
```

- Kód začíná být nepřehledný a složitý -> rozdělíme jej na funkce.

Řešení

- Začneme hlavní funkcí:

```
def loterie(n,p,schema,losy):
    tah=vylosuj(n,p)
    vyhry=vyhodnot(tah,losy,schema)
    return vyhry
```

- Vytvoříme funkci vylosuj:

```
def vylosuj(n,p):
    t=[]
    for i in range(p):
        t+=[random.randint(1,n)]
    return t
```

Řešení

- Vytvoříme funkci vyhodnot:

```
def vyhodnot(tah,losy,schema):
    vyhry=[]
    for l in losy:
        spravne=0
        for i in l:
            if i in tah:
                spravne+=1
        v=[[l,0]]
        for s in schema:
            if s[0]==spravne:
                v[1]=s[1]
                break
        vyhry+=v
    return vyhry
```

Řešení

- Funkci loterie zavoláme:

```
x = 20
y = 6
sch=[[6,1000],[5,500],[4,100]]
lo= [[1,8,17,14,5,6],[11,18,7,4,15,16],[2,4,5,6,19,20]]

seznam_vyher=loterie(x,y,sch,lo)
for s in seznam_vyher:
    print("Los {0} vyhráva {1} Kč".format(s[0],s[1]))
```

Rozsah platnosti

Příklad

```
def f1(b):
    b = b + 1
    return b

def f2(b):
    f1(b)
    return b

print (f2 (2))
```

- Co se vytiskne?

Rozsah platnosti

- **Lokální proměnná**
  - deklarujeme ji ve funkci,
  - při každém vstupu do daného bloku je proměnná vytvořena,
  - při opuštění funkce pozbývá proměnná platnosti.
- **Globální proměnná**
  - deklarujeme ji mimo funkce,
  - platnost je v celém modulu,
  - ve funkci můžeme vytvořit globální proměnnou pomocí příkazu global,
  - pozor na špatnou práci s globálními proměnnými.
- **Pravidlo LEGB**
  - identifikátor se hledá v pořadí:
    - lokální rozsah (local),
    - neukončená funkce (enclosing),
    - globální rozsah (global),
    - vestavěné názvy (built-in),
  - pokud není nalezen ani v jednom rozsahu -> chyba.

Příklad

```
x=99

def fun():
    x=88
    print(f"Ve funkci je hodnota X={x}.")

fun()
print(f"Mimo funkci je hodnota X={x}.")
```

Příklad

```
x=99

def fun():
    global x
    x=88
    print(f"Ve funkci je hodnota X={x}.")

fun()
print(f"Mimo funkci je hodnota X={x}.")
```

Úkol

- Vytvořte počáteční stav šachovnice u hry česká dáma.
- Šachovnici osm krát osm reprezentujte jako seznam řádku, kde řádek je seznam polí.
- Nulou označte prázdné pole, jedničkou bílý kámen a dvojkou černý kámen.

- Napište funkci, která vytiskne šachovnici. Prázdná pole tisknete znakem tečka, bílý kámen písmenem malé o, černý kámen hvězdičkou. Za znakem pole nechávejte mezeru.
- Doplňte z obou stran čísla řádků a písmen sloupců.

```
  a b c d e f g h
1 . o . o . o . o 1
2 o . o . o . o . 2
3 . o . o . o . o 3
4 . . . . . . . . 4
5 . . . . . . . . 5
6 . * . * . * . * 6
7 * . * . * . * . 7
8 . * . * . * . * 8
  a b c d e f g h
```