Lekce 5

Funkce a algoritmy

Funkce, lambda funkce a rekurze. Výjimky. Algoritmizace a základy složitosti. Příklady základních algoritmů: největší společný dělitel, prvočísla. Debugování.

C2184 Úvod do programování v Pythonu
podzim 2016

Stanislav Geidl
Národní centrum pro výzkum biomolekul
Masarykova univerzita

Funkce

- funkce je **část programu** (je podprogram, angl. subroutine), která lze volat opakovaně z různých části programu, příkladem může být funkce print()
- každá funkce má svůj **identifikátor** (podobně jako proměnná), pomocí kterého ji můžeme **volat**
  `moje_funkce()`
- požadavky na funkci můžeme specifikovat pomocí **parametrů**
  `moje_funkce(parametr1, parametr2)`
- výsledek funkce může být předán jako **návratová hodnota**
  `navratova_hodnota = moje_funkce()`

Definice funkce

- pomocí klíčového slova `def` a bloku (: a odsazení)

```
def moje_funkce(parametr):
    pass
```

- návratovou hodnotu definujeme klíčovým slovem `return`, POZOR: po jejím zavolání se již v provádění funkce nepokračuje

```
def moje_funkce(parametr):
    print()
    return True
    print() # nedosažitelný kód
```

- pomocí `return` můžeme navracet hodnotu, hodnotu proměnné nebo také nic (`None`)

```
def moje_funkce(parametr):
    return
```

- defaultní hodnoty a názvy parametrů

Proměnné a funkce

- **parametry**

```
a = 1
def moje_funkce(parametr):
    print(parametr)
moje_funkce(a)
```

- **lokální proměnné** - existují pouze v rámci konkretního volání funkce a mají přednost před globální proměnnou

```
a = 1
def moje_funkce():
    a = 10
    b = 20
print(a)
```

- **globální proměnné** - existují mimo funkci, ale můžeme ji v rámci funkce používat

```
a = 1
b = 2
def moje_funkce():
    print(a)
    b = 20 # lokální proměnná
print(b)
```


Rekurze a rekurzivní volání funkce

- volání sama sebe

```
def moje_funkce():
    moje_funkce()
```

- ukázka výpočtu faktoriálu

```
def faktorial(a):
    if a < 2:
        return 1
    else:
        return a * faktorial(a - 1)
```


Anonymní lambda funkce

- v případě, kdy potřebuje funkci volat "jednorázově"

```
square = lambda x: x**2
square(5) # 25
```

- příklad ve funkci filter:

```
a = filter(lambda x: x%2,[0,1,2,3,4])
```


Chyby v kódu

- **syntaktické chyby (errors)** – skript/program není vůbec spuštěn
- **výjimky (exceptions)** – skript/program běží, ale někdy skončí v průběhu vykonávání
- **systematické chyby** – skript/program běží bez chyb, ale nedává očekávaný výsledek

Výjimky

- obsluha stavů skriptu, které ovlivní předčasné ukončení chodu programu

```
print("Hello World!")
10/0
print("Bye bye")
```

```
Hello World!
Traceback (most recent call last):
  File "python", line 2, in <module>
ZeroDivisionError: division by zero
```

- vestavěné výjimky:

  https://docs.python.org/3/library/exceptions.html

- obsluha výjimky pomocí bloku `try ... except`

try ... except

- konkrétní výjimka

```
try:
    10/0
    """zde už se žádný kód neprovede"""
    print("Pěkný den!")
except ZeroDivisionError:
    print("Nulou nelze dělit")
```

- všeobecná výjimka

```
try:
    a/b
    int(input())
except:
    print("Chyba!")
```


try ... except (pokračování)

- kombinace

```
try:
    a/b
    int(input())
except ZeroDivisionError:
    print("Nulou nelze dělit")
except ValueError:
    print("Nemůžu prevést zadané číslo")
except (RuntimeError, TypeError, NameError):
    pass """přeskočí chybu, jako by se nestala, nic
    se nestane a program bude pokračovat"""
except:
    print("Chyba!")
    raise """výjimka se znovu vyvolá"""
```


Python "Ask forgiveness not permission"

- ptáme se na svolení

```
if muzu_udelat_operaci():
    udelej_operaci()
else:
    obsluz_zakaz()
""" ukázka """
if b <> 0.0:
    print(a/b)
else:
    print("Nulou nelze delit")
```

- ptáme se na odpuštění

```
try:
    udelej_operaci()
except nemoznost_udelat_operaci:
    obsluz_odpusteni()
""" ukázka """
try:
    print(a/b)
except ZeroDivisionError:
    print("Nulou nelze delit")
```


Algoritmus

- Algoritmus je schematický postup pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků.
- V běžném životě se s algoritmy setkáváme ve formě kuchyňských receptů, návodů a postupů ...
- Algoritmus má tyto vlastnosti:
  1. konečnost - má konečný počet kroků
  2. určitost - všechny kroky jsou přesně definovány
  3. koreknost - pro správné zadání skončí se správným výsledkem
  4. obecnost - řeší všechny úlohy a vstupy, pro které je navržen
- Algoritmy dělíme na interaktivní a rekurzivní. Interaktivní používá cykly - opakuje konkretní bloky či podprogramy. Rekurzivní algoritmy používájí rekurzivní funkce - volají samy sebe.
- Asymptotická složitost algoritmu charakterizuje počet provedených operací v závislosti na velikosti dat (například počet prvků v seznamu). Například pokud procházíme pole, pak složitost bude lineární (na každý prvek připadá konstantní množství operací). Pokud ale budeme potřebovat pole seřadit, složitost algoritmu poroste.

Formy zápisu algoritmů

- graficky
  - Vývojové diagramy
  - Strukturogramy
  - UML diagramy (aktivitní diagramy)
- textově
  - Přirozený jazyk
  - Pseudokód
  - Formální jazyk a programovací jazyk

Obecné zásady psaní kódu - proč?

- čitelnost
- udržitelnost - udržba a rozšiřování
- zdroj hotových funkcí či jiných částí programu -> tvorba univerzálnějších knihoven

Obecné zásady psaní kódu

- odsazení řádků musí odpovídat vnořeným blokům (v Pythonu automaticky)
- jeden řádek - jeden příkaz (značka) (v Pythonu automaticky)
- logické celky odděleny mezerou (code style)
- dodržujte jednotné názvy objektů
- identifikátory by měly co nejpřesněji popisovat význam
- dodržovat formát pro konstrukce
- každá funkčnost naprogramovaná pouze jednou (DRY - don't repeate yourself)
- kód nesmí obsahovat magická čísla
- jedna obrazovka - jedna funkcionalita (jedna funkce)
- POUŽÍVAT KOMENTÁŘE

Obecné zásady psaní komentářů

- komentáře nesmí duplikovat kód
- komentář musí jednoznačně osvětlit popisovanou část kódu
- komentuje se hlavička každého souboru, se kterým se pracuje
- komentuje se hlavička všech objetků, funkcí a konstrukcí procedur
- komentují se všechny netriviální konstrukce

Jak na to

- na internetu můžeme najít "hotový kód"[1]
- vymyslet to:
  - snažit se problem rozložit na jednodušší úkony - funkce
  - nakreslit vývojový diagram
  - ...

Více v předmětu C2142 Návrh algoritmů pro přírodovědce
https://is.muni.cz/auth/predmet/sci/jaro2015/C2142

[1] např. http://www.algoritmy.net/

Debugování

- označované také jako ladění nebo "odvšivování"je proces zbavování kódu chyb a nevhodného chování
- nástroje:
  - čteme chyby
  - breakpointy
  - krokování

Příklad

1. Napište funkci pro převod čísel (celých i float) pomocí bloku `try ... except`.

```
Vstup: 10
Výstup: 10
Vstup: 10.5
Výstup: 10.5
Vstup: 10e2
Výstup: 1000.0
Vstup: 10a
Výstup: None
```


Příklad - řešení

```
a = input()

""" fce na prevod retezce na cislo """
def preved_na_cislo(text):
    try:
        cislo = int(text)
        return cislo
    except ValueError:
        print("Nejedna se o cele cislo")
    try:
        cislo = float(text)
        return cislo
    except ValueError:
        print("Nejedna se o desetinne cislo")
    return None

print(preved_na_cislo(a))
```