Algoritmy a programování

Funkce

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Motivační příklad

- Výpočet $1 + 2x^2 - 4x^3$ pro $x \in \{0.5, 1, 2\}$

```
1 v1 = 1 + 2*1**2 - 4*1**3
2 print("f(1)=",v1)
3 v2 = 1 + 2*1/2**2 - 4*1/2**3
4 print("f(1/2)=",v2)
5 v3 = 1 + 2*2**2 - 4*2*3
6 print("f(2)=",v3)
```

```
f(1)= -1
f(1/2)= 1.0
f(2)= -15
```

- Co lze programu vytknout?

Motivační příklad

- Výpočet polynomu provedeme ve funkci
- Vstupem funkce je hodnota x
- Výstupem funkce je hodnota $1 + 2x^2 - 4x^3$
  - Klíčové slovo `return`

```
1 def f(x):
2     return 1 + 2*(x**2) - 4*(x**3)
3
4 value = 1
5 print("f(",value, ")=", f(value) )
6 value = 1/2
7 print("f(",value, ")=", f(value) )
8 value = 2
9 print("f(",value, ")=", f(value) )
```

```
f( 1 )= -1
f( 0.5 )= 1.0
f( 2 )= -23
```

Funkce

- Funkce je způsob strukturování programu
- Funkce řeší konkrétní úkol
- Funkce lze opakovaně volat
- Použití funkcí zjednodušuje vývoj, debugování (ladění), pochopení programu
  - Stačí opravit chybu jednou (ve funkci), ne v každém volání
  - Funkce jsou (typicky) menší a tudíž jednodušší na pochopení
- Při použití funkcí nedochází k duplikaci kódu
- Funkce má jméno, vstup (argument) a poskytuje výstup (návratová hodnota)

```
1 def functionName(parameters):
2     code
```

- Pouze příkazy se stejným odsazením jsou součástí funkce

Příklady funkcí: výpis

- Funkce vypisuje vstupní argument mezi “!”

```
1 def Print(text):
2     print("!!!!!!", text, "!!!!!!")
3
4 print("Normal print")
5 Print("Access denied")
```

```
Normal print
!!!!!! Access denied !!!!!!
```

Příklady funkcí: faktoriál


```
1 def fact(n): #n is integer >=0
2     result = 1
3     for i in range(2,n+1):
4         result *= i
5     return result
6
7 for value in range(8):
8     f = fact(value)
9     print("fact(",value,")␣=␣", f)
```

```
fact( 0 ) =  1
fact( 1 ) =  1
fact( 2 ) =  2
fact( 3 ) =  6
fact( 4 ) =  24
fact( 5 ) =  120
fact( 6 ) =  720
fact( 7 ) =  5040
```

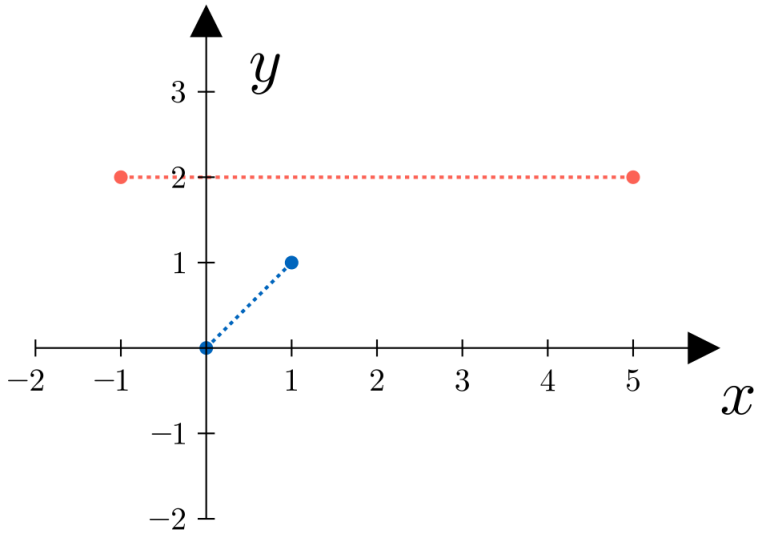
- Výpočet faktoriálu: $n! = n \cdot (n-1) \cdot (n-2) \cdots 1$
- Vstupem funkce je n, výstupem je $n!$
- Výsledek je předán klíčovým slovem `return`

Příklady funkcí: vzdálenost bodů

- Výpočet vzdálenosti 2D bodů (x_1, y_1) a (x_2, y_2)

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```
1 def distance(x1,y1,x2,y2):
2     dx = x1-x2
3     dy = y1-y2
4     return (dx**2 + dy**2)**(0.5)
5
6 print( distance(0,0,1,1) )
7 print( distance(-1,2,5,2) )
```




```
1.4142135623730951
6.0
```

Příklady funkcí: aproximace π

- Srinivasa Ramanujan odvodil

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!\,(1103 + 26390k)}{(k!)^4 396^{4k}}$$

- Napište funkci, která realizuje tento výpočet
- Jaké podpůrné operace budou k výpočetu potřeba?
- Jaké typy proměnných budeme potřebovat (float, string, int?)

Příklady funkcí: aproximace π

- Opakující se operace (faktoriál) realizujeme funkcí, sumu jako for cyklus

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{\overbrace{(4k)!\,(1103 + 26390k)}^{a}}{\underbrace{(k!)^4 396^{4k}}_{b}}$$

```
1  def fact(n):
2      res = 1
3      for i in range(2, n+1):
4          res *= i
5      return res
6
7  def piApprox():
8      s = 0           #for computing the sum
9      kmax = 6      #what is good value?
10     for k in range(kmax):
11         a = fact(4*k)* (1103 + 26390*k)
12         b = (fact(k)**4)*(396**(4*k))
13         s += a/b
14         print(k, a/b)    #debug output
15     r = (2*(2**(0.5)) / 9801)*s
16     return 1/r
17
18 print( "PI=", piApprox() )
```

Příklady funkcí: aproximace π

Výpis volání `piApprox`:

```
0 1103.0
1 2.6831974348925308e-05
2 2.2453850201136644e-13
3 1.995074994495897e-21
4 1.8393545314677564e-29
5 1.7358835411230251e-37
PI= 3.141592653589793
```

- Výsledek a/b rychle klesá, ve 4. iteraci je $\sim 1.8 \cdot 10^{-29}$
- Prakticky nemá smysl dělat více než několik iterací
- Otestujte program pro různá *kmax*. Co se stane, pokud $kmax = 1$?
- Existuje *kmax*, pro které program selže?

Příklady funkcí: výskyt písmen v textu

- Vstupem funkce je text a písmeno, úkolem je spočítat výskyt písmena v textu

```
1  def countLetter(text, letter):
2      #text,letter are strings, len(letter) = 1
3      c = 0
4      for i in range(len(text)):
5          if text[i] == letter:
6              c += 1
7      return c
8
9  text = "hippopotomonstrosesquippedaliophobia"
10
11 print( countLetter(text, "o") )
12 print( countLetter(text, "h") )
```

```
7
2
```

Příklady funkcí: výskyt písmen v textu

- Rozšíříme předchozí program na výpočet výskytu všech písmen

```
1  def countLetter(text, letter):
2      #text,letter are strings, len(letter) = 1
3      c = 0
4      for i in range(len(text)):
5          if text[i] == letter:
6              c += 1
7      return c
8
9  text = "hippopotomonstrosesquippedaliophobia"
10 for i in range(ord('a'), ord('z')+1):  #note +1 !!!
11     c = countLetter(text, chr(i))
12     if c != 0:
13         print(chr(i), ":", c)
```

```
a : 2
b : 1
d : 1
e : 2
h : 2
i : 4
l : 1
m : 1
n : 1
o : 7
p : 6
q : 1
r : 1
s : 3
t : 2
u : 1
```

Funkce: varianty

Počty parametrů

- žádný parametr
- jeden nebo více parametrů

Návratové hodnoty:

- Jedna nebo více návratových hodnot
- Pokud není explicitně uvedena, je návratová hodnota `None`

```
1  def f1():
2      a = 1
3                  #noreturn
4
5  def f2(x):
6      y = x
7                  #noreturn
8
9  def f3(c,d,a):
10     d = 2
11                 #noreturn
12
13 def f4():
14     return -1
15
16 def f5(x,y,z):
17     return y,z,y
18
19 f1()
20 f2(10)
21 f3(1,2,3)
22 v = f4()
23 a,b,c = f5(0,1,2)
```

Funkce: správné volání

- Funkci musíme volat přesně s tolika argumenty, kolik je vyžadováno v její definici
- (Neplatí v případě argumentů s defaultní hodnotou, ale to nebudeme používat)
- Případné špatné volání je detekováno až za běhu programu!

```
1 def xyz():
2     print("call xyz")
3
4 xyz()
5 xyz(12)
```

```
call xyz
Traceback (most recent call last):
  File "../functions//function2error.py", line 5, in <
     module>
    xyz(12)
TypeError: xyz() takes 0 positional arguments but 1
   was given
```

Funkce: správné volání

- Funkci musíme volat přesně s tolika argumenty, kolik je vyžadováno v její definici
- (Neplatí v případě argumentů s defaultní hodnotou, ale to nebudeme používat)
- Případné špatné volání je detekováno až za běhu programu!

```
1 def awesome(x,y):
2     return x+y
3
4 awesome(12)
```

```
    awesome(12)
TypeError: awesome() missing 1 required positional
    argument: 'y'
```

Funkce: return

- Návratová hodnota je určena klíčovým slovem `return`
- Pokud není explicitně uvedena, je návratová hodnota `None`

```
1 def f1(text):
2     print("Text: ", text)
3
4 a = f1("ahoj")
5 b = print(a)
6 print(b)
```

```
Text:  ahoj
None
None
```

Funkce: návratová hodnota

- Návratová hodnota je určena klíčovým slovem `return`
- Funkce může vracet více hodnot
- Více návratových hodnot se předává jako typ tuple

```
1 def sortAB(a,b):
2     if a < b:
3         return a,b
4     else:
5         return b,a
6
7 x,y = sortAB(10,-1)
8 print(x,y)
```

```
-1 10
```

Funkce: návratová hodnota

- Návratová hodnota je určena klíčovým slovem `return`
- Funkce může vracet více hodnot
- Více návratových hodnot se předává jako typ tuple

```
1 def sortAB(a,b):
2     if a < b:
3         return a,b
4     else:
5         return b,a
6
7 z = sortAB(10,-1) #all values into tuple z
8 print(z)
9 print(type(z))
```

```
(-1, 10)
<class 'tuple'>
```

Funkce: návratová hodnota

- Návratová hodnota je určena klíčovým slovem `return`
- Volání return bez parametru: návratová hodnota je `None`
- Stejně tak pokud je funkce ukončena bez `return`

```
1 def someFunction(x,y):
2     v = x+y
3
4 r = someFunction(10,11)
5 print(r)
```

```
None
```

Funkce: předávání parametrů hodnotou

- Datové typy immutable (int, float, string a další) se do funkcí předávají hodnotou
- Změna hodnoty argumentů se neprojeví mimo funkci

```
1  def test1(a,b):
2      print("Test1", a,b)
3      a = 0
4      b = 0
5      print("Test1", a,b)
6
7  x = 10
8  y = 20
9  print("Main:",x,y)
10 test1(x,y)
11 print("Main:",x,y)
```

```
Main: 10 20
Test1 10 20
Test1 0 0
Main: 10 20
```

Funkce: globální proměnné

- Pokud je proměnná definována ve funkci, je lokální (není přístupná mimo funkci)
- Proměnné definované mimo funkce jsou tzv. globální a funkce je mohou používat **pro čtení**

```
1 def add(x):
2     x += someValue
3     return x
4
5 someValue = 3
6 print( add(4) )
7 print( add(10) )
```

```
7
13
```

Funkce: globální proměnné

- Pokud je proměnná definována ve funkci, je lokální (není přístupná mimo funkci)
- Proměnné definované mimo funkce jsou tzv. globální a funkce je mohou používat **pro čtení**

```
1 def add(x):
2     x += someValue
3     return x
4
5 #someValue = 3
6 print( add(4) )
7 print( add(10) )
```

```
    print( add(4) )
  File "../functions//functionGlobal1.py", line 2, in add
    x += someValue
NameError: name 'someValue' is not defined
```

Funkce: globální proměnné

- Pokud je proměnná definována ve funkci, je lokální (není přístupná mimo funkci)
- Proměnné definované mimo funkce jsou tzv. globální a funkce je mohou používat **pro čtení**

```
1 def add(x):
2     someValue = 1 #new local variable
3     x += someValue  #new local variable
4     return x
5
6 someValue = 3 #global variable
7 print( add(4) )
8 print( add(10) )
```

```
5
11
```

Funkce: globální proměnné

- Proměnná je definována ve funkci ⇒ je lokální (není přístupná mimo funkci)
- Proměnné definované mimo funkce jsou tzv. globální a funkce je mohou používat **pro čtení**
- Pokud chceme měnit globální proměnnou ⇒ klíčové slovo `global`
- Toto vede na změnu stavu programu — může ovlivnit jiné části programu
- Nepoužíváme

```
1  def add(x):
2      global someValue
3      someValue += 1
4      x += someValue
5      return x
6
7  someValue = 3 #global variable
8  print("Add", add(4) )
9  print("someValue",someValue)
10 print("Add", add(4) )
11 print("someValue",someValue)
```

```
Add 8
someValue 4
Add 9
someValue 5
```

Pure function (čistá funkce)

- Její výstup závisí (jednoznačně) pouze na vstupních argumentech
- Nemá vedlejší efekty (změna globálních proměnných, souborů)

Pure

```
1  def f1():
2      return 12
3
4  def f2(x,y):
5      return x*min(x,y)
6
7  def f3(x,y,z):
8      return 123
9
10 def f4():
11     a = 1335
```

Impure

```
1  def f1():
2      return 12*a #uses global var.
3
4  def f2(x,y):
5      return x*min(x,b) #uses global var.
6
7  def f3(x,y,z):
8      global q  #changes global var.
9      q = x
10     return x+y+z+q
```

Funkce: doporučení

- Omezujeme použití globální proměnných
- Píšeme krátké funkce ($\sim$ jedna obrazovka)
- Jméno funkce vystihuje její obsah
- Jména argumentů taktéž
- Preferujeme čisté (pure) funkce

Chybná syntaxe (syntax error)

- Špatný zápis programu, který porušuje syntaktická pravidla
- Je detekován před spuštěním programu (kontroluje Python)
- Při chybě syntaxe se program nespustí

```
1 def b():
2     i = 1
3      print("hello")
```

```
  File "../errors//syntaxError1.py", line 3
    print("hello")
    ^
IndentationError: unexpected indent
```

```
1 a = 3
2 b = a**(1/2
```

```
  File "../errors//syntaxError2.py", line 3

                ^
SyntaxError: unexpected EOF while parsing
```

Chybná syntaxe (syntax error)

- Špatný zápis programu, který porušuje syntaktická pravidla
- Je detekován před spuštěním programu (kontroluje Python)
- Při chybě syntaxe se program nespustí

```
1 for i in range(5)
2     print(i)
```

```
  File "../errors//syntaxError3.py", line 1
    for i in range(5)
                    ^
SyntaxError: invalid syntax
```

```
1 s = "ahoj'
2 print(s)
```

```
  File "../errors//syntaxError4.py", line 1
    s = "ahoj'
              ^
SyntaxError: EOL while scanning string literal
```

Chyba běhu (runtime error)

- Chyba, která vznikne až při exekuci programu
- Např. dělení nulou, nevhodné typy argumentů funkcí, volání metod polí na ne-pole proměnné ...
- Tyto chyby nelze detekovat dokud nenastanou

```
1 def gcd(a,b):
2     return a
3
4 print("Program␣start")
5 print(gdc(10,2))
```

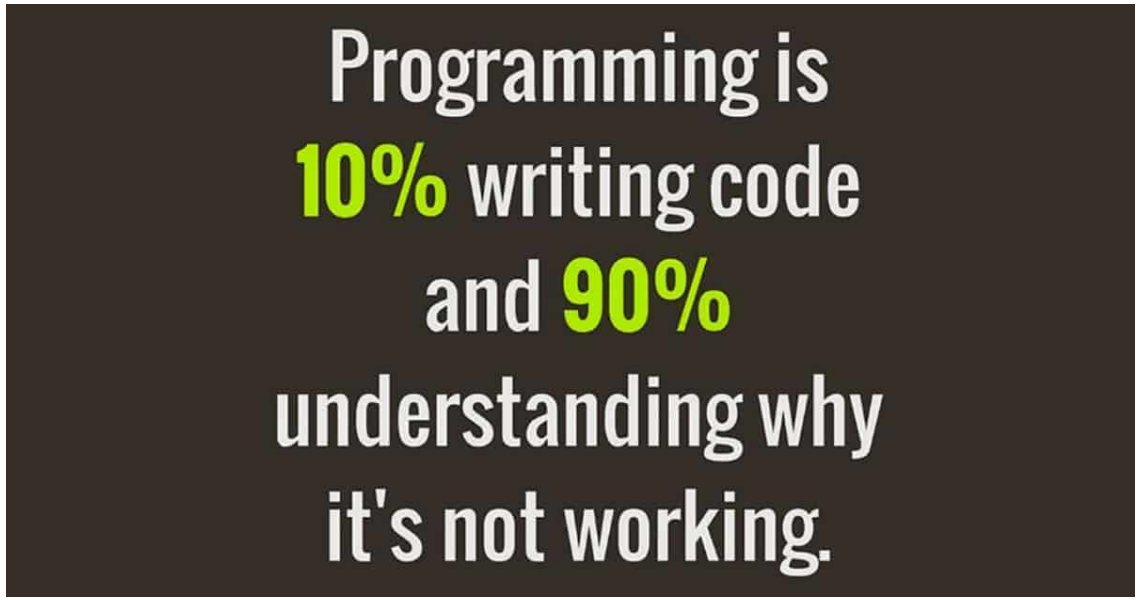
```
Program start
Traceback (most recent call last):
  File "../errors//runtimeError1.py", line 5, in <module>
    print(gdc(10,2))
NameError: name 'gdc' is not defined
```

Chyba běhu (runtime error)

- Chyba, která vznikne až při exekuci programu
- Např. dělení nulou, nevhodné typy argumentů funkcí, volání metod polí na ne-pole proměnné ...
- Tyto chyby nelze detekovat dokud nenastanou

```
1 def normalize(a,b,c):
2     s = a+b+c
3     return a/s, b/s, c/s
4
5 print( normalize(1,2,0) )
6 print( normalize(1,2,-3) )
```

```
(0.3333333333333333, 0.6666666666666666, 0.0)
Traceback (most recent call last):
  File "../errors//runtimeError5.py", line 6, in <module>
    print( normalize(1,2,-3) )
  File "../errors//runtimeError5.py", line 3, in normalize
    return a/s, b/s, c/s
ZeroDivisionError: division by zero
```


Programming is
10% writing code
and 90%
understanding why
it's not working.

Sémantická chyba (semantic error)

- Program pracuje bez chyby (bez runtime/syntax error), ale jeho výsledek neodpovídá zamýšlenému významu
- Analýza programátorem

```
1 def getMin(a,b):
2     if a<b:
3         return a
4     return b
5
6 x = input()
7 y = input()
8 print("Mensi cislo je", getMin(x,y))
```

inputFromFile.txt

```
12
5
```

```
python3 semanticError1.py < inputFromFile.txt
```

```
Mensi cislo je 12
```