

B4B33RPH: Řešení problémů a hry

Python – základní kameny až skály

Tomáš Svoboda, Petr Pošík, Petr Štibinger

svobodat@fel.cvut.cz

8. října 2024



Katedra kybernetiky
Fakulta elektrotechnická
České vysoké učení technické v Praze

Než začneme...

Materiály z přednášek

- [CourseWare](#) – přednáškové slajdy
- gitlab.fel.cvut.cz/RPH-student-materials – zdrojové kódy
 - Verzovací systém Git – root.cz/knihy/pro-git

Platnost proměnných

variables_scope.py – co je to vlastně **scope**?

```
1 C = 3
2 a = 1
3
4 def my_function(x):
5     a = 9 + C
6     return x + a
7
8 print(a)
9
10 if __name__ == '__main__':
11     a = 2
12     b = my_function(a)
13     print(a, b, C)
```

Platnost proměnných

variables_scope.py – co je to vlastně **scope**? Scope = oblast platnosti proměnných

```
1 C = 3
2 a = 1
3
4 def my_function(x):
5     a = 9 + C
6     return x + a
7
8 print(a)
9
10 if __name__ == '__main__':
11     a = 2
12     b = my_function(a)
13     print(a, b, C)
```

Platnost proměnných

variables_scope.py – co je to vlastně **scope**? Scope = oblast platnosti proměnných

```
1 C = 3
2 a = 1
3
4 def my_function(x):
5     a = 9 + C
6     return x + a
7
8 print(a)
9
10 if __name__ == '__main__':
11     a = 2
12     b = my_function(a)
13     print(a, b, C)
```

Řádek 8 vytiskne:

- (a) 12
- (b) 1
- (c) nastane RuntimeError

Platnost proměnných

variables_scope.py – co je to vlastně **scope**? Scope = oblast platnosti proměnných

```
1 C = 3
2 a = 1
3
4 def my_function(x):
5     a = 9 + C
6     return x + a
7
8 print(a)
9
10 if __name__ == '__main__':
11     a = 2
12     b = my_function(a)
13     print(a, b, C)
```

Řádek 13 vytiskne:

- (a) 2 14 3
- (b) 1 14 3
- (c) chyba protože C je neznámé
- (d) chyba už na řádku 12

Co “hodný” program nedělá?

variables_scope.py

```
1 C = 3
2 a = 1
3
4 def my_function(x):
5     a = 9 + C
6     return x + a
7
8 print(a)
9
10 if __name__ == '__main__':
11     a = 2
12     b = my_function(a)
13     print(a, b, C)
```

example_launcher.py

```
1 import variables_scope as vs
2
3 if __name__ == '__main__':
4     print(vs.my_function(2))
```

Co “hodný” program nedělá?

variables_scope.py

```
1 C = 3
2 a = 1
3
4 def my_function(x):
5     a = 9 + C
6     return x + a
7
8 print(a)
9
10 if __name__ == '__main__':
11     a = 2
12     b = my_function(a)
13     print(a, b, C)
```

example_launcher.py

```
1 import variables_scope as vs
2
3 if __name__ == '__main__':
4     print(vs.my_function(2))
```

Po spuštění example launcher

- (a) nastane RuntimeError
- (b) vypíše 1 řádek s výsledkem
- (c) vypíše toho víc

Základní bloky

basic_blocks.py – základní struktura programu

```
1  import math
2
3  class MyClass:
4      '''class for doing important stuff'''
5
6      def __init__(self):
7          '''MyClass constructor that shows how "pass" works'''
8          pass # does nothing, adds a new line after a block header
9
10 def my_function(a, b):
11     '''function that computes sum a + b'''
12     return a + b
13
14 if __name__ == '__main__':
15     # actual program starts here
16     c = MyClass() # don't forget the ( ) !
17     result = my_function(10, 25)
```

Základní bloky

basic_blocks.py – základní struktura programu

import modulů

```
1  import math
2
3  class MyClass:
4      '''class for doing important stuff'''
5
6      def __init__(self):
7          '''MyClass constructor that shows how "pass" works'''
8          pass # does nothing, adds a new line after a block header
9
10 def my_function(a, b):
11     '''function that computes sum a + b'''
12     return a + b
13
14 if __name__ == '__main__':
15     # actual program starts here
16     c = MyClass() # don't forget the ( ) !
17     result = my_function(10, 25)
```

Základní bloky

basic blocks.py – základní struktura programu

```
1  import math
2
3  class MyClass:
4      '''class for doing important stuff'''
5
6      def __init__(self):
7          '''MyClass constructor that shows how "pass" works'''
8          pass # does nothing, adds a new line after a block header
9
10 def my_function(a, b):
11     '''function that computes sum a + b'''
12     return a + b
13
14 if __name__ == '__main__':
15     # actual program starts here
16     c = MyClass() # don't forget the ( ) !
17     result = my_function(10, 25)
```

import modulů

definice tříd a funkcí
(obecné šablony)

Základní bloky

basic blocks.py – základní struktura programu

```
1 import math
```

import modulů

```
3 class MyClass:
```

```
4     '''class for doing important stuff'''
```

definice tříd a funkcí
(obecné šablony)

```
5  
6     def __init__(self):
```

```
7         '''MyClass constructor that shows how "pass" works'''
```

```
8         pass # does nothing, adds a new line after a block header
```

```
9  
10    def my_function(a, b):
```

```
11        '''function that computes sum a + b'''
```

```
12        return a + b
```

```
14 if __name__ == '__main__':
```

```
15     # actual program starts here
```

```
16     c = MyClass() # don't forget the ( ) !
```

```
17     result = my_function(10, 25)
```

hlavní program
(vytváříme konkrétní instance,
používáme funkce)

Funkce vs. metoda

function_vs_method.py – názvosloví

```
1  class MyClass:
2      '''class for doing important stuff'''
3
4      def __init__(self):
5          pass # does nothing, adds a new line after a block header
6
7      def my_class_method(self):
8          print('started my_class_method')
9
10 def my_function(a, b):
11     print('started my_function')
12
13 if __name__ == '__main__':
14     # actual program starts here
15     c = MyClass()
16
17     my_function(10.7, 'RPH')
18
19     c.my_class_method()
```

Funkce vs. metoda

function_vs_method.py – názvosloví

```
1  class MyClass:
2      '''class for doing important stuff'''
3
4      def __init__(self):
5          pass # does nothing, adds a new line after a block header
6
7      def my_class_method(self):
8          print('started my_class_method')
9
10 def my_function(a, b):
11     print('started my_function')
12
13 if __name__ == '__main__':
14     # actual program starts here
15     c = MyClass()
16
17     my_function(10.7, 'RPH')
18
19     c.my_class_method()
```

Metoda – uvnitř třídy

Funkce vs. metoda

function_vs_method.py – názvosloví

```
1  class MyClass:
2      '''class for doing important stuff'''
3
4      def __init__(self):
5          pass # does nothing, adds a new line after a block header
6
7      def my_class_method(self):
8          print('started my_class_method')
9
10 def my_function(a, b):
11     print('started my_function')
12
13 if __name__ == '__main__':
14     # actual program starts here
15     c = MyClass()
16
17     my_function(10.7, 'RPH')
18
19     c.my_class_method()
```

Metoda – uvnitř třídy

Funkce – samostatná

Funkce vs. metoda

function_vs_method.py – názvosloví

```
1 class MyClass:
2     '''class for doing important stuff'''
3
4     def __init__(self):
5         pass # does nothing, adds a new line after a block header
6
7     def my_class_method(self):
8         print('started my_class_method')
9
10 def my_function(a, b):
11     print('started my_function')
12
13 if __name__ == '__main__':
14     # actual program starts here
15     c = MyClass()
16
17     my_function(10.7, 'RPH')
18
19     c.my_class_method()
```

Metoda – uvnitř třídy

Funkce – samostatná

Volání funkce – nevyžaduje nic extra

Funkce vs. metoda

function_vs_method.py – názvosloví

```
1  class MyClass:
2      '''class for doing important stuff'''
3
4      def __init__(self):
5          pass # does nothing, adds a new line after a block header
6
7      def my_class_method(self):
8          print('started my_class_method')
9
10 def my_function(a, b):
11     print('started my_function')
12
13 if __name__ == '__main__':
14     # actual program starts here
15     c = MyClass()
16
17     my_function(10.7, 'RPH')
18
19     c.my_class_method()
```

Metoda – uvnitř třídy

Funkce – samostatná

Volání funkce – nevyžaduje nic extra

Volání metody z **konkrétního** objektu **c**

Není číslo jako číslo

```
>>> a = 0.1  
>>> b = 0.3  
>>> c = 3*a  
>>> d = b == c
```

Co se stane?

- (a) program skončí chybou
- (b) **d je True**
- (c) **d je False**
- (d) **d je 0.3**

Desítková a dvojková soustava

Celá čísla

Decimal (base 10)		Binary (base 2)
$1 \cdot 10^1 + 3 \cdot 10^0 = 13_{10}$	=	$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
$1 \cdot 10 + 3 \cdot 1 = 13_{10}$	=	$1101_2 = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$

Demonstrační kód: [float2bin.py](#)

Desítková a dvojková soustava

Celá čísla

Decimal (base 10)		Binary (base 2)
$1 \cdot 10^1 + 3 \cdot 10^0 = 13_{10}$	=	$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
$1 \cdot 10 + 3 \cdot 1 = 13_{10}$	=	$1101_2 = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$

Desetinná čísla

Decimal (base 10)		Binary (base 2)
$6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3} = 0.625_{10}$	=	$0.101_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$
$6 \cdot 1/10 + 2 \cdot 1/100 + 5 \cdot 1/1000 = 0.625_{10}$	=	$0.101_2 = 1 \cdot 1/2 + 0 \cdot 1/4 + 1 \cdot 1/8$

Demonstrační kód: [float2bin.py](#)

Předcházíme problémům

Některá čísla se reprezentují obtížně

- Např. $2/3$ v desítkové soustavě ≈ 0.6667
- Stejně problémy i v binární!
- Omezený počet desetinných míst nebo bitů → **zaokrouhlování**

Předcházíme problémům

Některá čísla se reprezentují obtížně

- Např. $2/3$ v desítkové soustavě ≈ 0.6667
- Stejné problémy i v binární!
- Omezený počet desetinných míst nebo bitů → **zaokrouhlování**

Při práci s floaty:

- Používáme raději operátory $>$ $<$
- Hledáme alternativy pro $==$
- Zaokrouhlovací **tolerance**: $\text{abs}(a-b) < \text{tol}$
- Vestavěné funkce: `math.isclose(a,b)`

Pár příkladů z konzole

- Vědecká notace
- Počítání s extra velkými a extra malými čísly
- Formátování výpisu
- Použití `isclose`
- Speciální čísla: ∞ , NaN

Kámen nůžky papír – pokračování

```
1 import random
2
3 class MyPlayer:
4     '''A basic random player'''
5
6     OPTIONS = ('R', 'P', 'S') # class constant/variable
7
8     def __init__(self):
9         self.history = []
10
11     def play(self):
12         return random.choice(MyPlayer.OPTIONS)
13
14     def record(self, move):
15         self.history.append(move)
```

Nápady k vylepšení

- náhodný, ale vychýlený výběr
- využití paměti, analýza soupeře – je také vychýlený?
- reaktivní strategie – maximalizujeme vlastní zisk
- paměťový limit

Nerovnoměrný náhodný výběr

random_choices.py – ukázka váhovaného výběru

```
1 import random
2
3 options = ['R', 'P', 'S']
4 weights = [1,4,1]
5
6 result = random.choices(options, weights, k=15)
7 print(result)
```

Reference vs skutečná data

```
1 a = [1, 2, 3]
2 b = a
3 a[0] = 9
4 print(a, b)
```

Řádek 4 vypíše

(a) [9, 2, 3] [1, 2, 3]

(b) [9, 2, 3] [9, 2, 3]

(c) [9, 2, 3, 1, 2, 3]

(d) [9, 2, 3, 9, 2, 3]

Reference vs skutečná data

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
([known limitations](#))

```
1 a = [1,2,3]
2 b = a
3 a[0] = 9
→ 4 print(a,b)
```

[Edit this code](#)

→ line that just executed

→ next line to execute

<< First

< Prev

Next >

Last >>

Done running (4 steps)

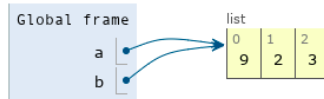
[Customize visualization](#)

Print output (drag lower right corner to resize)

[9, 2, 3] [9, 2, 3]

Frames

Objects



Kopírujeme data

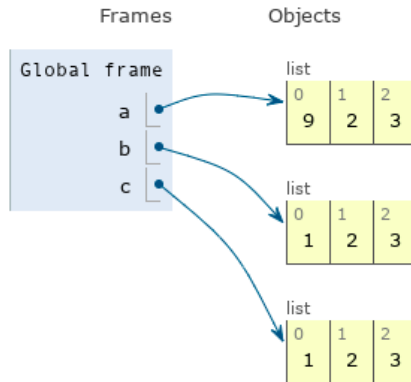
ref_vs_data_2.py

```
1 a = [1, 2, 3]
2 b = a[:]
3 c = list(a)
4 a[0] = 9
5 print(a, b, c)
```

Kopírujeme data

ref_vs_data_2.py

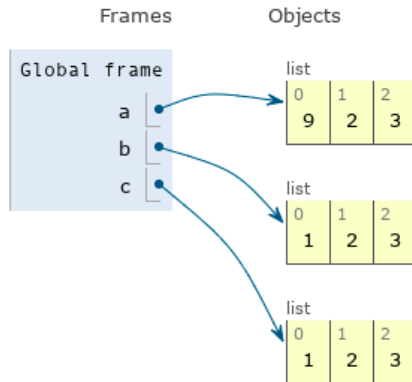
```
1 a = [1,2,3]
2 b = a[:]
3 c = list(a)
4 a[0] = 9
5 print(a, b, c)
```



Kopírujeme data

ref_vs_data_2.py

```
1 a = [1,2,3]
2 b = a[:]
3 c = list(a)
4 a[0] = 9
5 print(a, b, c)
```



Jednoduchý list – bez problémů

Kopírujeme data, ALE...

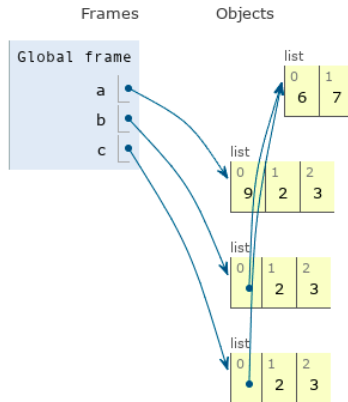
ref_vs_data_3.py

```
1 a = [[6,7],2,3]
2 b = a[:]
3 c = list(a)
4 a[0][1] = 9
5 print(a, b, c)
```

Kopírujeme data, ALE...

ref_vs_data_3.py

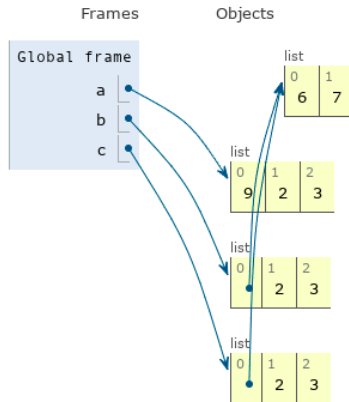
```
1 a = [[6,7],2,3]
2 b = a[:]
3 c = list(a)
4 a[0][1] = 9
5 print(a, b, c)
```



Kopírujeme data, ALE...

ref_vs_data_3.py

```
1 a = [[6,7],2,3]
2 b = a[:]
3 c = list(a)
4 a[0][1] = 9
5 print(a, b, c)
```



Problém – více úrovní seznamu, děláme pouze **mělkou kopii** (shallow copy)

Going deep

ref_vs_data_4.py

```
1 import copy
2 a = [[[1,2],7],2,3]
3 b = copy.deepcopy(a)
4 a[0][0][1] = 9
5 print(a, b)
```

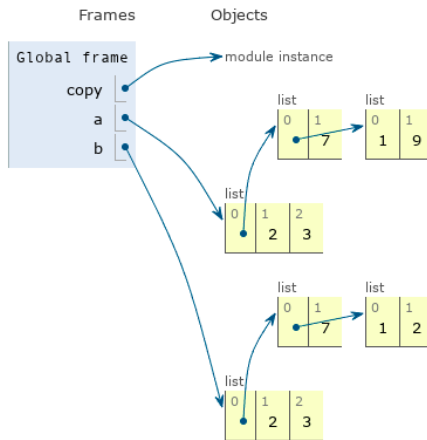
Going deep

ref_vs_data_4.py

```
1 import copy
2 a = [[[1,2],7],2,3]
3 b = copy.deepcopy(a)
4 a[0][0][1] = 9
5 print(a, b)
```

Print output (drag lower right corner to resize)

```
[[[1, 9], 7], 2, 3] [[[1, 2], 7], 2, 3]
```



Going deep

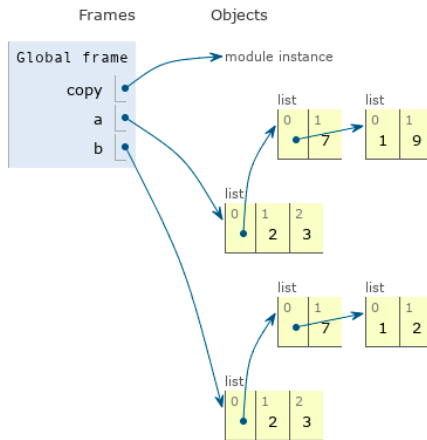
ref_vs_data_4.py

```
1 import copy
2 a = [[[1,2],7],2,3]
3 b = copy.deepcopy(a)
4 a[0][0][1] = 9
5 print(a, b)
```

Hluboká kopie (deep copy)

Print output (drag lower right corner to resize)

```
[[[1, 9], 7], 2, 3] [[[1, 2], 7], 2, 3]
```



Porovnávání dat a referencí

Operátor `==`

- Porovnává skutečné hodnoty (data)
- Dunder metoda `def __eq__(self, other):`
- Můžeme definovat vlastní chování

Operátor `is`

- Porovnává cíle reference
- Vyhodnotí zda dvě proměnné ukazují na identický objekt
- Nemá dunder implementaci, nelze předefinovat

Porovnávání dat a referencí

```
a = [1,2,3,4]
b = a
c = a[:]
```

Která možnost bude **False**?

- (a) `a == b`
- (b) `a == c`
- (c) `a is b`
- (d) `a is c`

Funkce ryzí (pure function) vs modifikátory (modifier)

pure_fn_vs_modifier.py

```
1  def increment_pure_function(x):
2      v = []
3      for item in x:
4          v.append(item + 1)
5      return v
6
7  def increment_modifier(x):
8      for i in range(len(x)):
9          x[i] += 1
10     return x
11
12  if __name__ == '__main__':
13      a = [1,2,3]
14      b1 = increment_pure_function(a)
15      d0 = a == b1
16      b2 = increment_modifier(a)
17      d1 = b1 == b2
18      d2 = a == b1
19      print(d0,d1,d2)
```

Funkce ryzí (pure function) vs modifikátory (modifier)

pure_fn_vs_modifier.py

```
1  def increment_pure_function(x):
2      v = []
3      for item in x:
4          v.append(item + 1)
5      return v
6
7  def increment_modifier(x):
8      for i in range(len(x)):
9          x[i] += 1
10     return x
11
12  if __name__ == '__main__':
13      a = [1,2,3]
14      b1 = increment_pure_function(a)
15      d0 = a == b1
16      b2 = increment_modifier(a)
17      d1 = b1 == b2
18      d2 = a == b1
19      print(d0,d1,d2)
```

Jaké budou hodnoty d0, d1, d2?

- (a) False, True, True
- (b) False, True, False
- (c) False, False, False

Co “hodný” program nedělá?

pure_fn_vs_modifier.py

```
1 def increment_pure_function(x):  
2     v = []  
3     for item in x:  
4         v.append(item + 1)  
5     return v
```

(A)

```
6  
7 def increment_modifier(x):  
8     for i in range(len(x)):  
9         x[i] += 1  
10    return x
```

(B)

```
11  
12 if __name__ == '__main__':  
13     a = [1,2,3]  
14     b1 = increment_pure_function(a)  
15     d0 = a == b1  
16     b2 = increment_modifier(a)  
17     d1 = b1 == b2  
18     d2 = a == b1  
19     print(d0, d1, d2)
```

Co “hodný” program nedělá?

pure_fn_vs_modifier.py

```
1 def increment_pure_function(x):
2     v = []
3     for item in x:
4         v.append(item + 1)
5     return v
6
7 def increment_modifier(x):
8     for i in range(len(x)):
9         x[i] += 1
10    return x
11
12 if __name__ == '__main__':
13     a = [1,2,3]
14     b1 = increment_pure_function(a)
15     d0 = a == b1
16     b2 = increment_modifier(a)
17     d1 = b1 == b2
18     d2 = a == b1
19     print(d0, d1, d2)
```

Špatně rozmyšlený kód!

- Chyby v modifikátorech se obtížněji hledají
- Pokud potřebujeme modifikátor – vhodnější objektově orientovaný přístup
- Modifikátor by neměl vracet data, která upravoval (*existují výjimky*)
- Ryzí funkce pracuje pouze ve vlastním scope a nemění existující data

Kontrolní otázka

```
a = [1,2,3]
b = [4,5,6]
c = a + b
```

Sčítání dvou seznamů:

- (a) je modifikátor
- (b) je funkce ryzí
- (c) vůbec není funkce

Shrnutí

- Oblast platnosti proměnných
- Základní struktura programu
- Funkce a metody – jen názvosloví
- Limity binární reprezentace (`float == float ...just don't`)
- Rozdíl mezi `==` a `is`
- Mělká a hluboká kopie
- Ryzí funkce a modifikátory

Díky za pozornost

Jaká byla dnes rychlost výkladu?

- (A) Příště přidejte.
- (B) Tak akorát.
- (C) Prosím zpomalte.
- (D) Ona už přednáška skončila?

Díky za pozornost

Dnešní látka pro vás byla:

- (A) Lehká až triviální.
- (B) Tak akorát.
- (C) Těžká, ztrácel/a jsem se.
- (D) ...