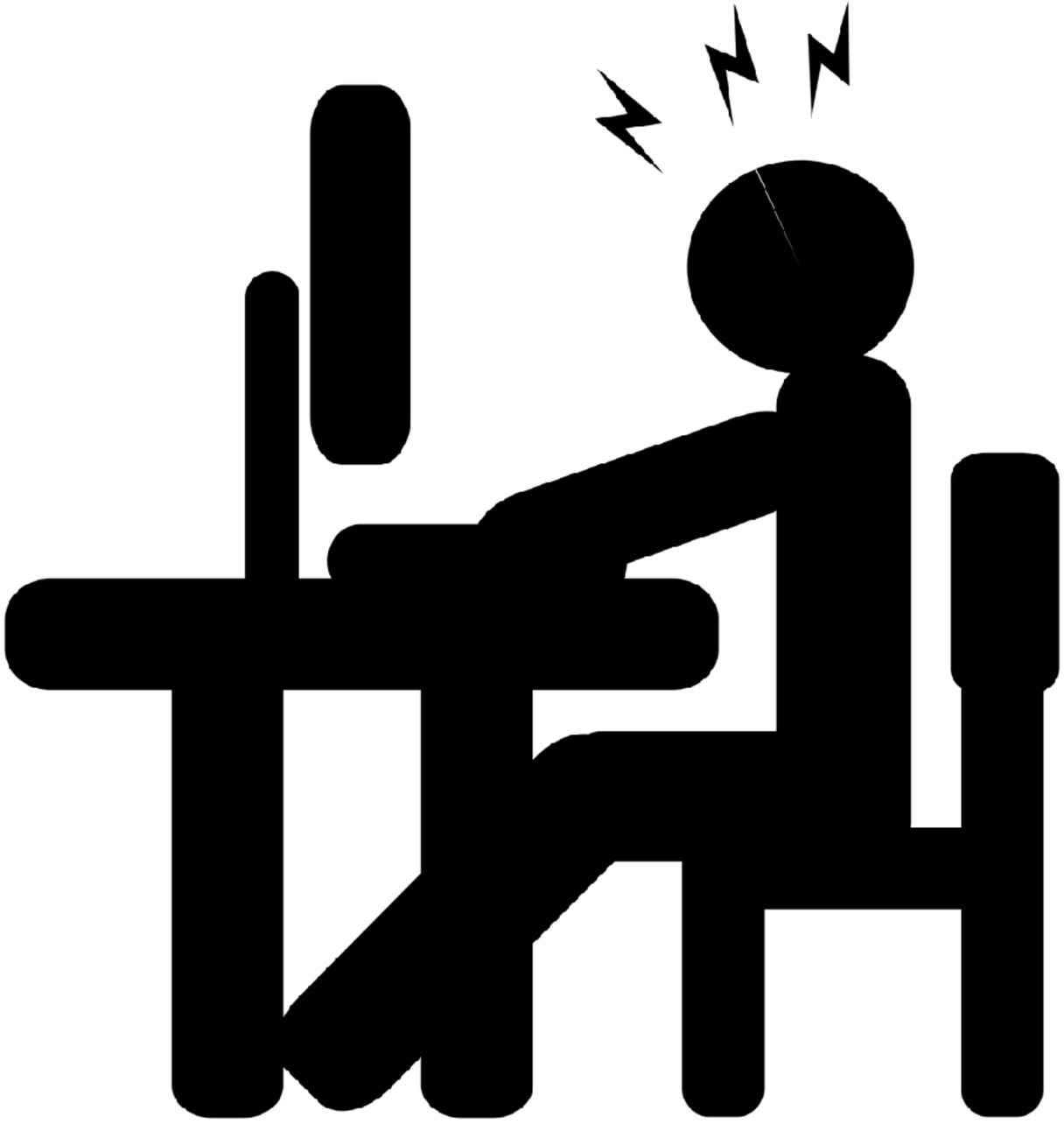
Python, základní kameny až skály I

Tomáš Svoboda

B4B33RPH, 2021-10-12, 2021-10-19

ČVUT, FEL, Katedra kybernetiky



https://gitlab.fel.cvut.cz/RPH-student-materials
http://cw.fel.cvut.cz/wiki/courses/b4b33rph/prednasky/start

https://www.root.cz/knihy/pro-git/

platnost 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

```
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)
```

> `python3 variables_scope.py`

Co vytiskne řádek 8:

A: 12

B: 1

C: nic, nastane chyba za běhu

variables_scope.py

platnost 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)
```

> `python3 variables_scope.py`

Co vytiskne řádek 8:

A: 12

B: 1

C: nic, nastane chyba za běhu

Co vytiskne řádek 13:

A: `2 14 3`

B: `1 14 3`

C: nastane chyba, `C` neznámé

D: nastane chyba už na řádku 12

`variables_scope.py`

platnost 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)
```

> `python3 variables_scope.py`

Co vytiskne řádek 8:

A: 12

B: 1

C: nic, nastane chyba za běhu

Co vytiskne řádek 13:

A: `2 14 3`

B: `1 14 3`

C: nastane chyba, `C` neznámé

D: nastane chyba už na řádku 12

`variables_scope.py`

platnost 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)
```

> `python3 variables_scope.py`

Co vytiskne řádek 8:

A: 12

B: 1

C: nic, nastane chyba za běhu

Co vytiskne řádek 13:

A: `2 14 3`

B: `1 14 3`

C: nastane chyba, `C` neznámé

D: nastane chyba už na řádku 12

`variables_scope.py`

platnost 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)
```

```
> python3 variables_scope.py
```

Co vytiskne řádek 8:

A: 12

B: 1

C: nic, nastane chyba za běhu

Co vytiskne řádek 13:

A: `2 14 3`

B: `1 14 3`

C: nastane chyba, `C` neznámé

D: nastane chyba už na řádku 12

variables_scope.py

platnost 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)
```

```
> python3 variables_scope.py
```

Co vytiskne řádek 8:

A: 12

B: 1

C: nic, nastane chyba za běhu

Co vytiskne řádek 13:

A: `2 14 3`

B: `1 14 3`

C: nastane chyba, `C` neznámé

D: nastane chyba už na řádku 12

variables_scope.py

function_vs_method.py

program structure - basic blocks

```
 1 import math
 2
 3 class MyClass:
 4     '''class for '''
 5     def __init__(self):
 6         '''MyClass constructor'''
 7         pass # nothing at the moment
 8
 9 def my_function(a,b):
10     '''compute sum a+b'''
11     pass # nothing at the moment
12
13 if __name__ == "__main__":
14     # actual program starts here
15     c = MyClass() # don't forget the parantheses! I will show!
16
```

function_vs_method.py

program structure - basic blocks

```
 1 import math
 2
 3 class MyClass:
 4     '''class for '''
 5     def __init__(self):
 6         '''MyClass constructor'''
 7         pass # nothing at the moment
 8
 9 def my_function(a,b):
10     '''compute sum a+b'''
11     pass # nothing at the moment
12
13 if __name__ == "__main__":
14     # actual program starts here
15     c = MyClass() # don't forget the parantheses! I will show!
16
```

imports

function_vs_method.py

program structure - basic blocks

```
 1 import math
 2
 3 class MyClass:
 4     '''class for '''
 5     def __init__(self):
 6         '''MyClass constructor'''
 7         pass # nothing at the moment
 8
 9 def my_function(a,b):
10     '''compute sum a+b'''
11     pass # nothing at the moment
12
13 if __name__ == "__main__":
14     # actual program starts here
15     c = MyClass() # don't forget the parantheses! I will show!
16
```

imports

Definitions:
classes
functions

function_vs_method.py

program structure - basic blocks

```
 1 import math
 2
 3 class MyClass:
 4     '''class for '''
 5     def __init__(self):
 6         '''MyClass constructor'''
 7         pass # nothing at the moment
 8
 9 def my_function(a,b):
10     '''compute sum a+b'''
11     pass # nothing at the moment
12
13 if __name__ == "__main__":
14     # actual program starts here
15     c = MyClass() # don't forget the parantheses! I will show!
16
```

imports

Definitions:
classes
functions

main program

function_vs_method.py

program structure - basic blocks

```
 1 import math
 2
 3 class MyClass:
 4     '''class for '''
 5     def __init__(self):
 6         '''MyClass constructor'''
 7         pass # nothing at the moment
 8
 9 def my_function(a,b):
10     '''compute sum a+b'''
11     pass # nothing at the moment
12
13 if __name__ == "__main__":
14     # actual program starts here
15     c = MyClass() # don't forget the parantheses! I will show!
16
```

imports

Definitions:
classes
functions

main program

V krátkých ukázkách budeme někdy ukazovat jen kód bez hlaviček a spol.

funkce vs. metoda

```
 1 import math
 2
 3 class MyClass:
 4     '''class for '''
 5     def __init__(self):
 6         '''MyClass constructor'''
 7         pass # nothing at the moment
 8     def my_class_method(self):
 9         print('nothing to report')
10
11
12 def my_function(a,b):
13     '''compute sum a+b'''
14     pass # nothing at the moment
15
16 if __name__ == "__main__":
17     # actual program starts here
18     c = MyClass() # don't forget the parantheses! I will show!
19     c.my_class_method()
20
```

funkce vs. metoda

```
1  import math
2
3  class MyClass:
4      '''class for '''
5      def __init__(self):
6          '''MyClass constructor'''
7          pass # nothing at the moment
8      def my_class_method(self):
9          print('nothing to report')
10
11
12 def my_function(a,b):
13     '''compute sum a+b'''
14     pass # nothing at the moment
15
16 if __name__ == "__main__":
17     # actual program starts here
18     c = MyClass() # don't forget the parantheses! I will show!
19     c.my_class_method()
20
```

funkce vs. metoda

```
1  import math
2
3  class MyClass:
4      '''class for '''
5      def __init__(self):
6          '''MyClass constructor'''
7          pass # nothing at the moment
8      def my_class_method(self):
9          print('nothing to report')
10
11
12 def my_function(a,b):
13     '''compute sum a+b'''
14     pass # nothing at the moment
15
16 if __name__ == "__main__":
17     # actual program starts here
18     c = MyClass() # don't forget the parantheses! I will show!
19     c.my_class_method()
20
```

```
1 a = 0.1
2 b = 0.3
3 c = 3*a
4 d = b==c
```

```
1 a = 0.1
2 b = 0.3
3 c = 3*a
4 d = b==c
```

Po vykonání řádku 4 bude proměnná **d** rovna:

A: True

B: False

C: řádek 4 se nevykoná, skončí chybou

není číslo jako číslo

```
1 a = 0.1
2 b = 0.3
3 c = 3*a
4 d = b==c
```

Po vykonání řádku 4 bude proměnná **d** rovna:

A: True

B: False

C: řádek 4 se nevykoná, skončí chybou

- vizualizace
- https://docs.python.org/3/tutorial/floatingpoint.html
- http://floating-point-gui.de/formats/binary/
- opatrnost při testování rovnosti (float) čísel
- pokud opravdu potřeba: `abs(a-b) < threshold`

`floating_point_surprise.py`

decimal vs binary

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

decimal-vs-binary-integers.png

decimal vs binary

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

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

decimal-vs-binary-integers.png

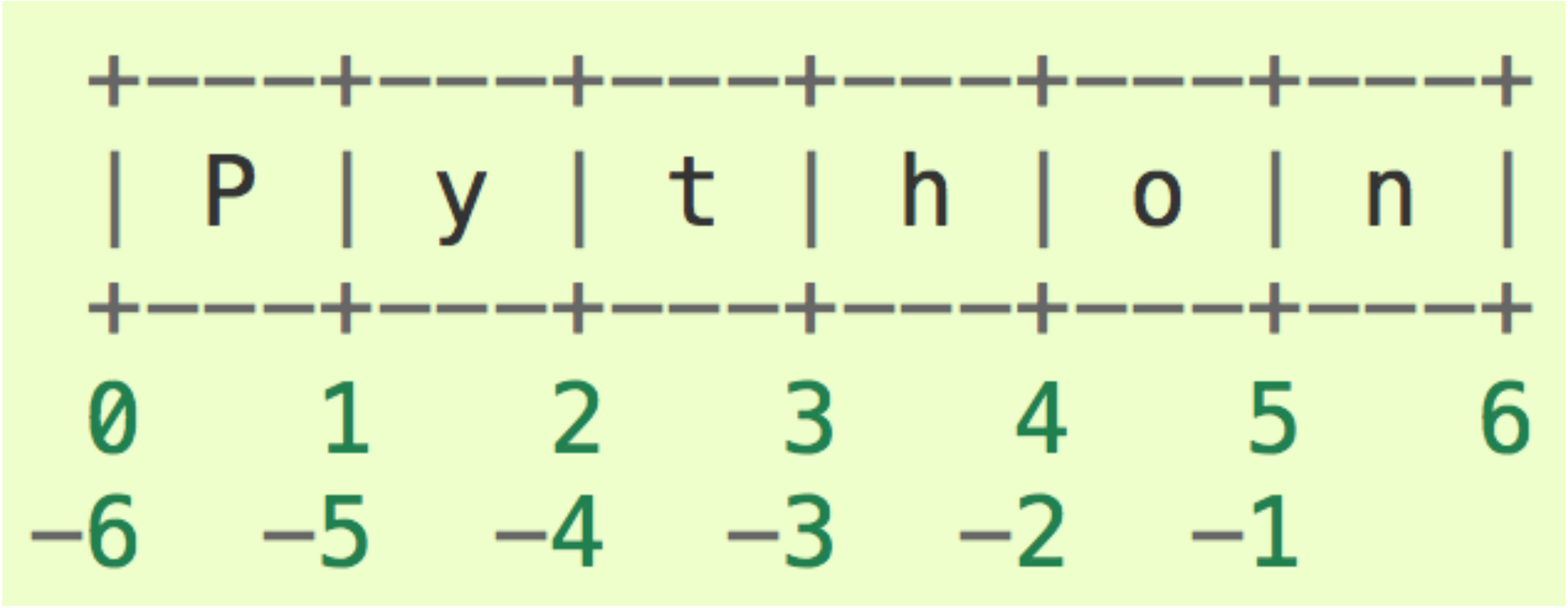
strings.py

řetězce neboli stringy

```
1  a = 'ahoj'
2  b = 'svete'
3  c = a+b
4  for i,item in enumerate(c):
5      print(i,'-',item)
6  banner = ['ahoj','svete']
7  for i,item in enumerate(banner):
8      print(i,'-',item)
9  for i,item in enumerate(banner):
10     for j,elem in enumerate(item):
11         print(i,'-',item,'***',j,':',elem)
```

visualizace

python indexing, slicing, …



dokončeme teď R-P-S
lepší hráč, s pamětí

- `player_skewed` některé tahy preferuje na úkor jiných
- nevíme které to jsou
- nejprve hrajme náhodně a uchovávejme soupeřovy tahy
- analyzujme historii tahů
- hrajme optimálně (využijme nedokonalost protihráče)

is vs ==

```
1 a = [1,2,3]
2 b = [1,2,3]
3 c = a
4
5 d1 = a==b
6 d2 = a is b
7 d3 = a==c, a is c
```

is_vs_equal.py

`is` vs `==`

```
1 a = [1,2,3]
2 b = [1,2,3]
3 c = a
4
5 d1 = a==b
6 d2 = a is b
7 d3 = a==c, a is c
```

A: `d1 True, d2 False`

B: `d1 False, d2 False`

C: `d1 True, d2 True`

D: chyba

`is_vs_equal.py`

`is` vs `==`

```
1 a = [1,2,3]
2 b = [1,2,3]
3 c = a
4
5 d1 = a==b
6 d2 = a is b
7 d3 = a==c, a is c
```

A: `d1 True, d2 False`

B: `d1 False, d2 False`

C: `d1 True, d2 True`

D: chyba

`d3` bude:

A: `True, False`

B: `True, True`

C: `False, False`

D: skončí chybou

`is_vs_equal.py`

pointers

```
1 a = [1,2,3]
2 b = a
3 a[1] = 9
4 d0 = a == [1,9,3]
5 d1 = b == [1,2,3]
6 d2 = b == [1,9,3]
7 d3 = b is a
```

pointers

```
1 a = [1,2,3]
2 b = a
3 a[1] = 9
4 d0 = a == [1,9,3]
5 d1 = b == [1,2,3]
6 d2 = b == [1,9,3]
7 d3 = b is a
```

Proměnné `d0, d1, d2, d3` budou:

A: `True, True, False, False`

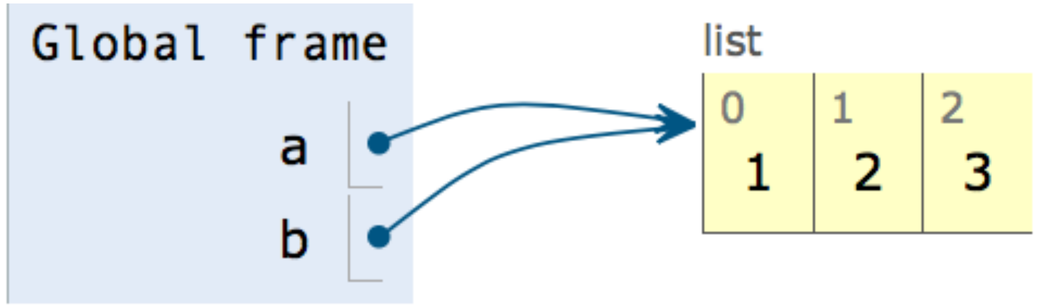
B: `True, False, True, True`

C: `True, False, False, True`

D: `True, False, True, False`

pointers

```
1  a = [1,2,3]
2  b = a
3
```



visualisation

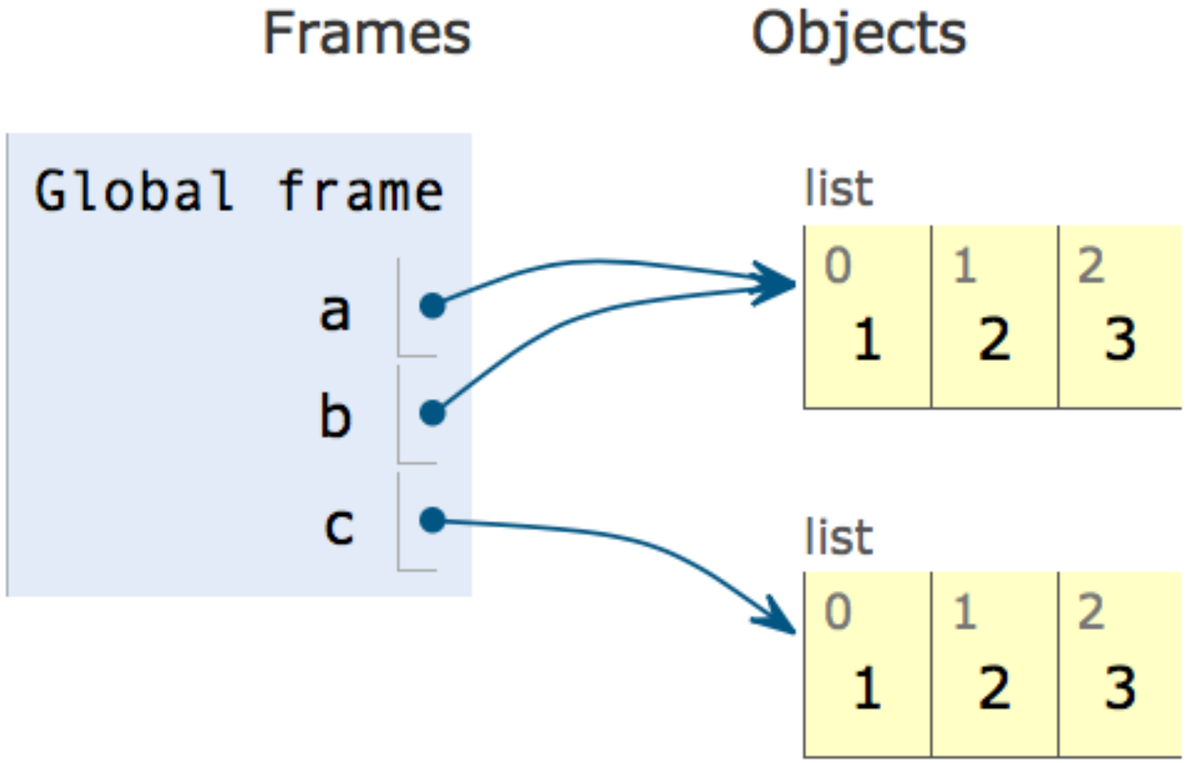
`is` vs `==`

making copy, a[:], rychle, ale …

Write code in Python 3.3

(drag lower right corner to resize code editor)

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

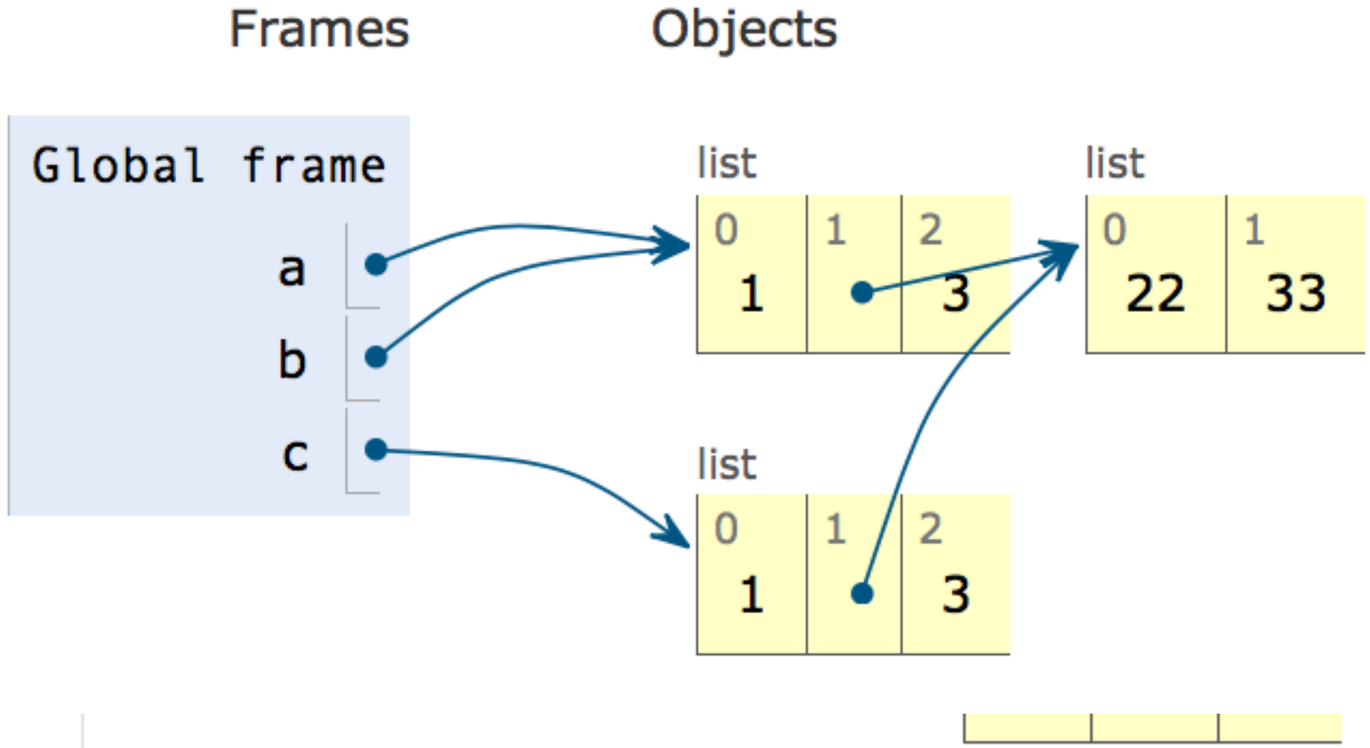


making copy, a[:], rychle, ale ...

Write code in Python 3.3

(drag lower right corner to resize code editor)

```
1  a = [1,[22,33],3]
2  b = a
3  c = a[:]
4
5
```



`import copy` and go deep

http://docs.python.org/3.4/library/copy.html

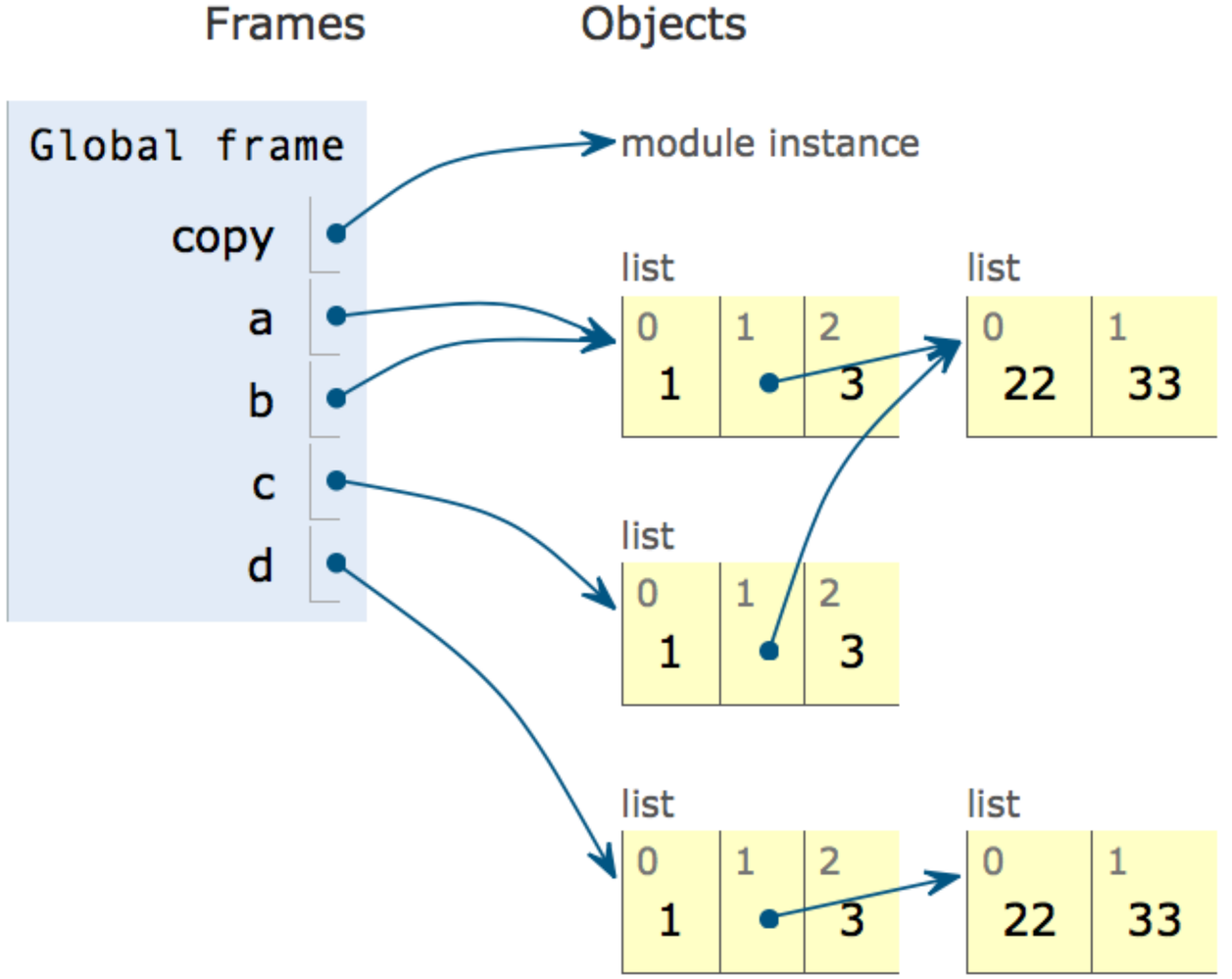
Write code in Python 3.3

(drag lower right corner to resize code editor)

```
1  import copy
2  a = [1,[22,33],3]
3  b = a
4  c = a[:]
5  d = copy.deepcopy(a)
6
7
```

line that has just executed

next line to execute

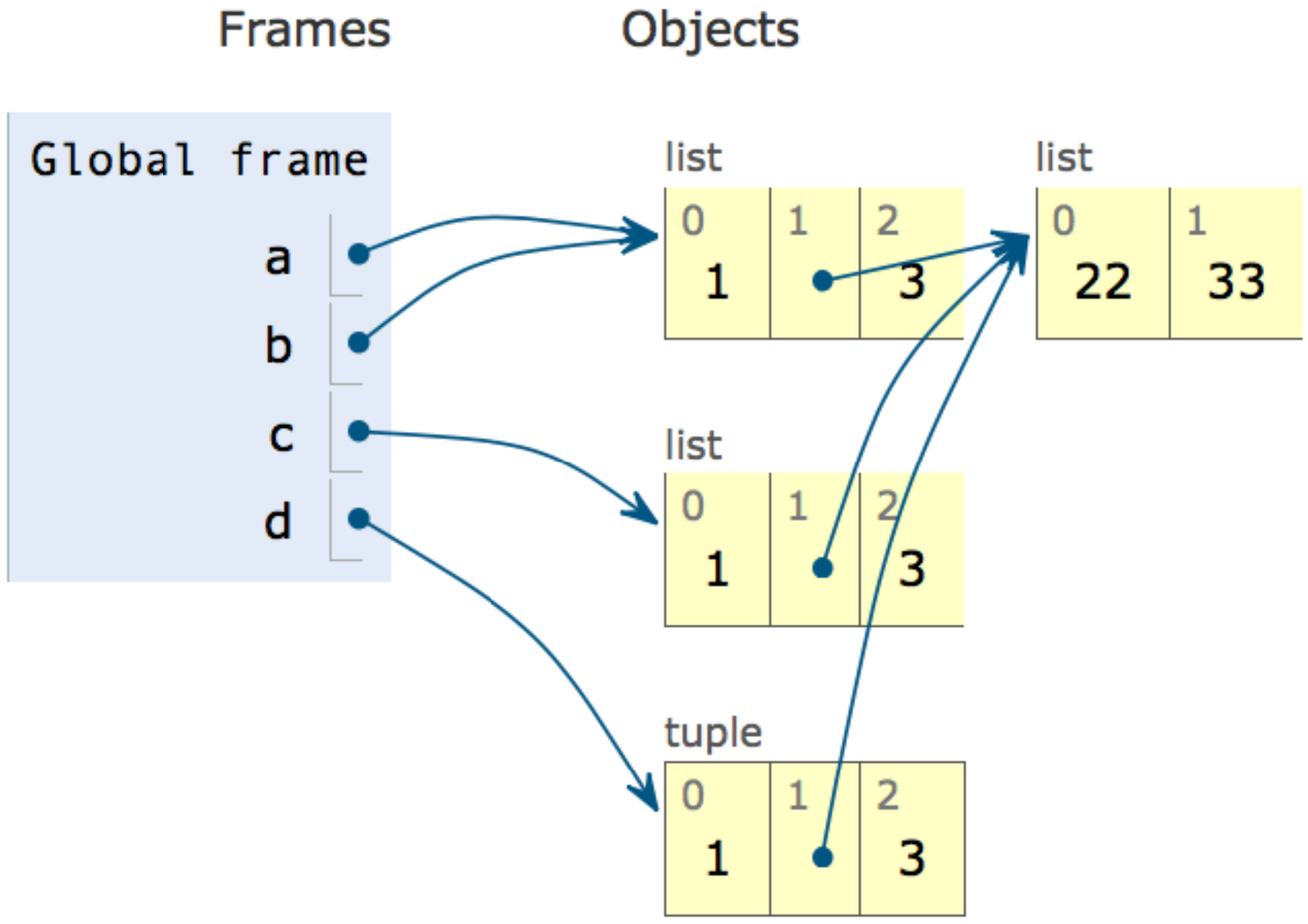


pozor na mělkost kopií

Write code in Python 3.3

(drag lower right corner to resize code editor)

```
1  a = [1,[22,33],3]
2  b = a
3  c = list(b)
4  d = tuple(a)
5
```



Funkce: ryzí (pure) vs. modifikátory (modifiers)

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

def increment_modifier(x):
    for i in range(len(x)):
        x[i] = x[i]+1
    return(x)

if __name__ == "__main__":
    a = [1,2,3]
    b1 = increment_pure_function(a)
    d0 = a == b1
    b2 = increment_modifier(a)
    d1 = b1 == b2
    d2 = a == b1
```

```python
def increment_pure_function(x):
    v = []
    for item in x:
        v.append(item+1)
    return(v)

def increment_modifier(x):
    for i in range(len(x)):
        x[i] = x[i]+1
    return(x)

if __name__ == "__main__":
    a = [1,2,3]
    b1 = increment_pure_function(a)
    d0 = a == b1
    b2 = increment_modifier(a)
    d1 = b1 == b2
    d2 = a == b1
```

Hodnoty `d0, d1, d2` budou:

A: `False, True, True`

B: `False, True, False`

C: `False, False, False`

funkce pravé a modifikátory

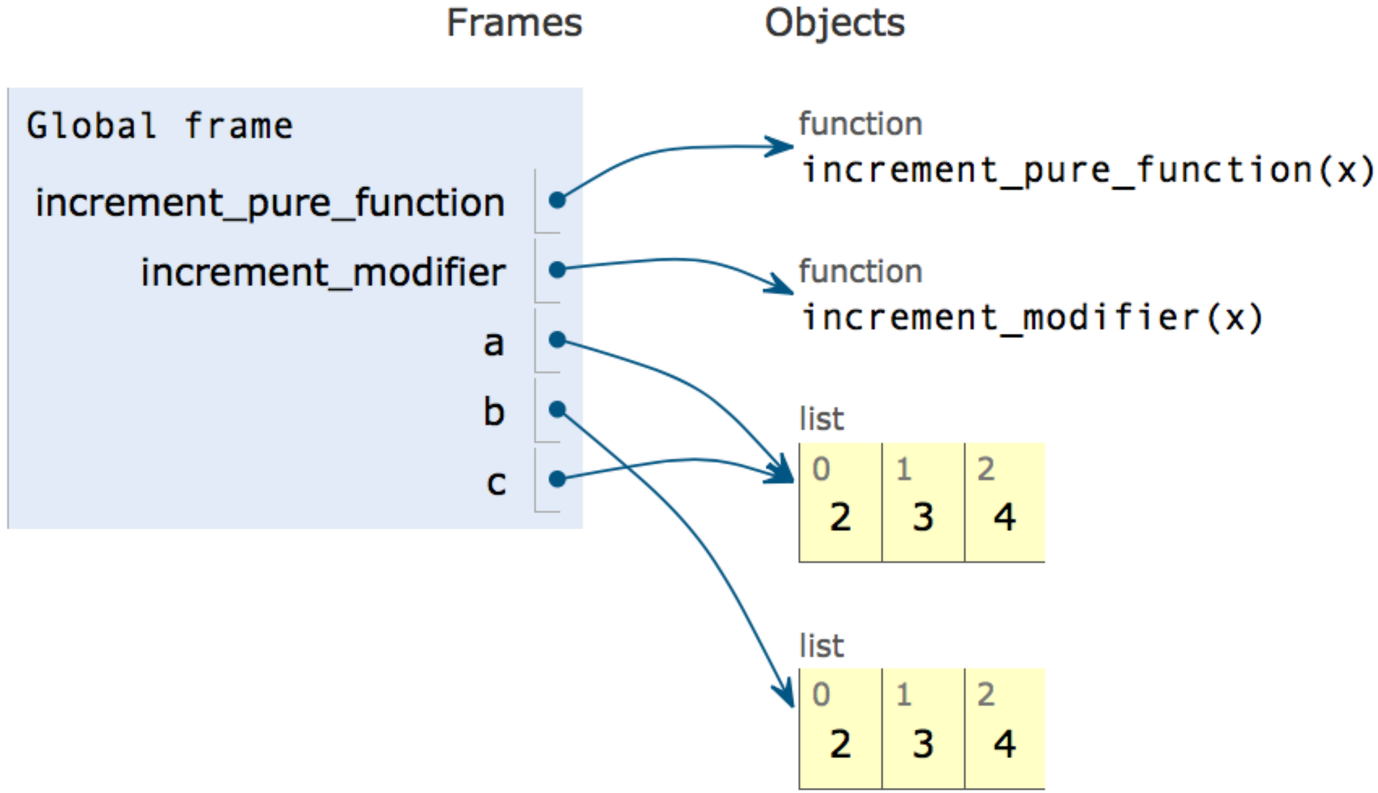
Write code in Python 3.3

(drag lower right corner to resize code editor)

```
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] = x[i]+1
10     return(x)
11
12 a = [1,2,3]
13 b = increment_pure_function(a)
14 print(a,',',b)
15 c = increment_modifier(a)
16 print(a,',',b,',',c)
```

Print output (drag lower right corner to resize)

```
[1, 2, 3] , [2, 3, 4]
[2, 3, 4] , [2, 3, 4] , [2, 3, 4]
```



function_pure_vs_modifier.py

- Použij funkci ryzí, pokud je tvé srdce ryzí (a nechceš hledat špatně odhalitelné chyby)
- Pokud návrh vede na potřebu funkce modifkátoru, zvaž objektový návrh.
- Nikdy nevracej data, která modifikuješ a hlavně, nechť volání modikátoru nemá pravou stranu!

Nebezpečí implicitních parametrů

```
1 def fn(x=[0,0]):
2     x[0] = x[0]+1
3     return x+[1]
4
5 a = fn()
6 b = fn()
```

```
1 def fn(x=[0,0]):
2     x[0] = x[0]+1
3     return x+[1]
4
5 a = fn()
6 b = fn()
```

Hodnoty proměnných `a, b` budou:

A: `[1,0,1], [1,0,1]`

B: `[2,1], [2,1]`

C: `[1,0,1], [2,0,1]`

D: dojde k chybě za běhu programu

```
1 def fn(x=[0,0]):
2     x[0] = x[0]+1
3     return x+[1]
4
5 a = fn()
6 b = fn()
7 c = fn([0,0])
```

```
1 def fn(x=[0,0]):
2     x[0] = x[0]+1
3     return x+[1]
4
5 a = fn()
6 b = fn()
7 c = fn([0,0])
```

Hodnota proměnné **c** bude:

A: **[1,0,1]**

B: **[2,0,1]**

C: **[3,0,1]**

D: dojde k chybě za běhu programu

```
1 def fn(x=[0,0]):
2     x[0] = x[0]+1
3     return x+[1]
4
5 a = fn()
6 b = fn()
7 c = fn([0,0])
```

Hodnota proměnné **c** bude:

A: `[1,0,1]`

B: `[2,0,1]`

C: `[3,0,1]`

D: dojde k chybě za běhu programu

`implicit_mutable_parameters_simple.py`

https://docs.python-guide.org/writing/gotchas/

implicitní parametry definovány, resp. vzniknou pouze 1x!

A: při prvním volání funkce
B: v okamžiku definování funkce

Vyzkoušejte/odkrokujte v pythontutor.com

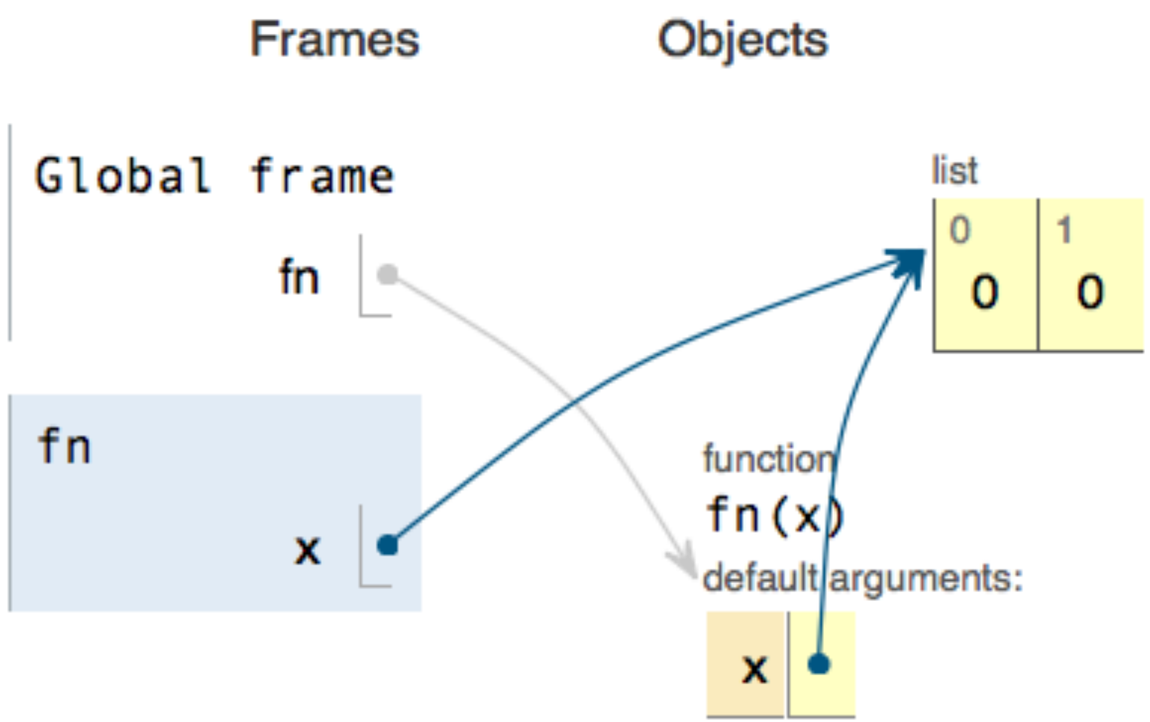
Python 3.6
(known limitations)

```
1  def fn(x=[0,0]):
2      x[0] = x[0]+1
3      return x+[1,1]
4
5  a = fn()
6  b = fn()
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>



Kompetence

- oblasti platnosti proměnných *(čím kratší život, tím lépe)*
- struktura programu *(všechno má své místo)*
- funkce vs. metoda *(jen názvosloví)*
- binární representace čísel má svá úskalí *(don't float==float)*
- is vs. == *(většinou chceme ==)*
- mělkost kopií *(na mělčině číhá nebezpečí)*
- funkce ryzí vs. modifiktátory *(rrrrr, ryzí, return)*
- implicitní parametry - měnitelné složité datové typy *(just don't)*

běžte a programujte!

- http://pythontutor.com/visualize.html#mode=edit
- http://openbookproject.net/thinkcs/python/english3e/index.html