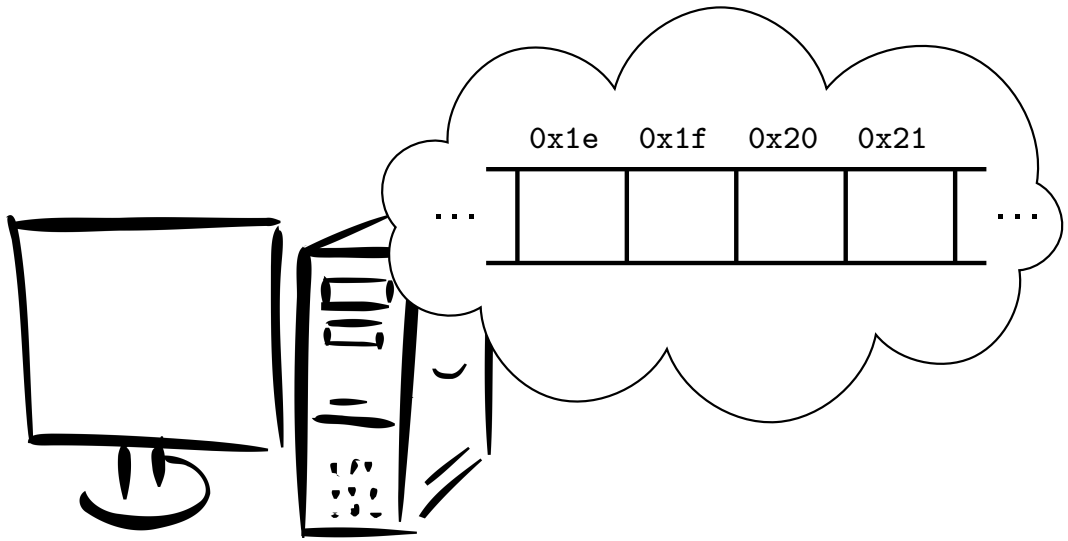
IB111 Úvod do programování skrze Python
Přednáška 8

Správa paměti (proměnné podrobněji)
Práce se soubory

Nikola Beneš

13. listopad 2015

Proměnné a paměť



Proměnné

Proměnné

- něco, co drží hodnotu
- jejich hodnota se může během výpočtu měnit

Názvy proměnných (v Pythonu)

- posloupnost písmen, číslic a znaků `'_'`
- bez mezer, více slov pomocí:
  - podtržítek: `dlouhy_nazev_promenne`
  - střídání velikosti písmen: `dlouhyNazevPromenne`
- nelze používat rezervovaná klíčová slova jazyka

```
and       def       finally   in        print     yield
as        del       for       is        raise     None
assert    elif      from      lambda    return
break     else      global    not       try
class     except    if        or        while
continue  exec      import    pass      with
```

Globální a lokální proměnné

Globální proměnné

- definovány globálně (tj. ne uvnitř funkce)
- jsou viditelné kdekoli v programu

Lokální proměnné

- definovány uvnitř funkce
- jsou viditelné jen ve své funkci

Obecněji:

- proměnné jsou viditelné v rámci svého bloku
- blokem můžou být:
  - moduly (soubory se zdrojovým kódem)
  - funkce
  - třídy (o těch se dozvíme později)
  - a jiné (závisí na konkrétním jazyce)

Globální a lokální proměnné v Pythonu

Příklad 1

```
a = "This is global."

def example1():
    b = "This is local."
    print a
    print b

example1()  # This is global.
            # This is local.
print a     # This is global.
print b     # CHYBA!

# NameError: name 'b' is not defined
```

Globální a lokální proměnné v Pythonu

Příklad 2

```
a = "Think global."

def example2():
    a = "Eat local."
    print a

print a      # Think global.
example2()   # Eat local.
print a      # Think global.
```

- vytváříme novou lokální proměnnou, neměníme tu globální

Globální a lokální proměnné v Pythonu

Příklad 3

- jak měnit globální proměnné?

```
a = "Think global."

def example3():
    global a
    a = "Eat local."
    print a

print a      # Think global.
example3()   # Eat local.
print a      # Eat local.
```

Globální a lokální proměnné v Pythonu

Příklad 4

- lokální proměnná vzniká přiřazením **kdekoli uvnitř funkce**

```
a = "Think global."

def example4(change_opinion=False):
    print a
    if change_opinion:
        a = "Eat local."
        print "Changed opinion:", a

print a      # Think global.
example4()   # CHYBA!

# UnboundLocalError: local variable 'a' referenced before
# assignment
```

Proměnné podrobněji

Proměnné v různých jazycích

- pojmenované místo v paměti
- odkaz na místo v paměti *(Python)*
- kombinace obou možností

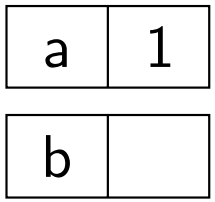
Přiřazení

- proměnné ve stylu C: změna obsahu paměti
- proměnné ve stylu Pythonu: změna odkazu na jiné místo v paměti

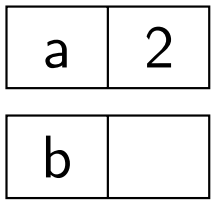
Proměnné podrobněji

Ilustrace přiřazení

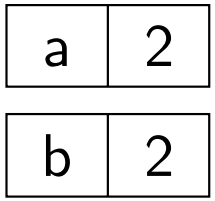
```
int a, b;
a = 1;
```




```
a = 2;
```



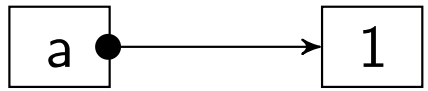

```
b = a;
```



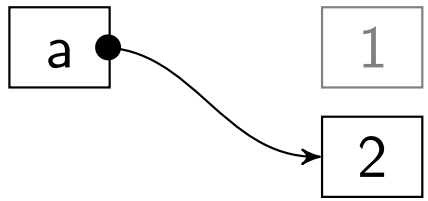
Jazyk C

Proměnné jako hodnoty

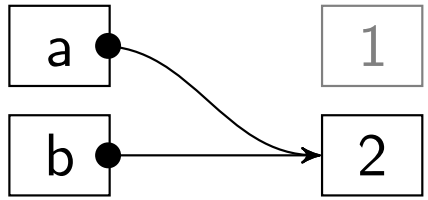
```
a = 1;
```




```
a = 2;
```




```
b = a;
```



Jazyk Python

Proměnné jako odkazy

Proměnné podrobněji

Příklad

```
a = 1000
b = a

print a, b
print id(a), id(b)

b += 1

print a, b
print id(a), id(b)
```

```
a = [1]
b = a

print a, b
print id(a), id(b)

b.append(2)

print a, b
print id(a), id(b)
```

[další ukázky, použití `http://pythontutor.com`]

Předávání parametrů funkcím

Způsoby předávání parametrů

- hodnotou (call by value)
  - předá se hodnota proměnné (kopie)
  - standardní v C, C++, apod.
- odkazem (call by reference)
  - předá se odkaz na proměnnou
  - lze použít v C++
- jiné možnosti (jménem, hodnotou-výsledkem, ...)
- jazyk Python: něco mezi voláním hodnotou a referencí
  - podobně funguje např. Java
  - někdy nazýváno *call by object sharing*

Předávání parametrů funkcím

Předávání parametrů hodnotou

- parametr je vlastně lokální proměnná
- funkce má svou vlastní lokální kopii předané hodnoty
- funkce nemůže změnit hodnotu předané proměnné

Předávání parametrů odkazem

- nepředává se hodnota, ale odkaz na proměnnou
- změny parametru jsou ve skutečnosti změny předané proměnné

Příklad předávání v C++

```
#include <iostream>

void funkce(int a, int & b) {
    a = a + 1;
    b = b + 1;
}
int main() {
    int a = 1;
    int b = 1;

    std::cout << "a: " << a << ", b: " << b << "\n";

    funkce(a,b);

    std::cout << "a: " << a << ", b: " << b << "\n";
}
```

Předávání parametrů funkcím

Předávání parametrů v Pythonu

- paramater drží odkaz na předanou proměnnou
- změna parametru změní i předanou proměnnou
- pro neměnné typy tedy v podstatě funguje jako předávání hodnotou
  - čísla, řetězce, ntice (tuples)
- pro měnitelné typy jako předávání odkazem
- ale pozor! přiřazení znamená změnu odkazu

```
def fun(s):
    s.append(3)
    s = [42, 17]
    s.append(9)
    print s

t = [1,2]
fun(t)       # [42, 17, 9]
print t      # [1, 2, 3]
```

Předávání parametrů funkcím

Operátor +=

- různé chování pro neměnné typy a pro seznamy

```
def increment(x):
    print x, id(x)
    x += 1
    print x, id(x)

p = 2015
increment(p)
print p, id(p)
```

```
def add_to_list(seznam):
    print seznam, id(seznam)
    seznam += [1]
    print seznam, id(seznam)

s = [1, 2, 3]
add_to_list(s)
print s, id(s)
```

Předávání parametrů funkcím

Pozor na rozdíl mezi = a += u seznamů

```
def add_to_list1(seznam):
    print seznam, id(seznam)
    seznam += [1]
    print seznam, id(seznam)

s = [1, 2, 3]
add_to_list1(s)
print s, id(s)

# vysledek je [1, 2, 3, 1]
```

```
def add_to_list2(seznam):
    print seznam, id(seznam)
    seznam = seznam + [1]
    print seznam, id(seznam)

s = [1, 2, 3]
add_to_list2(s)
print s, id(s)

# vysledek je [1, 2, 3]
```

Správa paměti

Různé přístupy ke správě paměti

- manuální – funkce pro přidělení/uvolnění paměti
- automatická – počítání referencí
  - kolik částí programu ještě s danou pamětí pracuje
  - pokud už nikdo, paměť je uvolněna
- automatická – garbage collection
  - jednou za čas se uklidí nepoužívaná paměť

Správa paměti v Pythonu

- automatická – počítání referencí + někdy i větší úklid
- počet referencí `sys.getrefcount(object)`

Správa paměti v Pythonu

Zvýšení počtu odkazů

- vytvoření — `a = "Hello!"`
- vytvoření aliasu — `b = a`
- předání funkci — `fun(a)`
- vložení do složeného prvku — `s = [42, a, -7]`

Snížení počtu odkazů

- ukončení platnosti lokální proměnné — konec funkce
- smazání proměnné — `del a`
- přiřazení jiné hodnoty aliasu — `b = "Aloha!"`
- odstranění ze složeného prvku — `s.remove(a)`
- odstranění složeného prvku — `del s`

Správa paměti v Pythonu

Pamatování některých hodnot

- některé hodnoty si Python automaticky udržuje v paměti a nevytváří je znovu
- kvůli vyšší rychlosti / výkonu

- často používané hodnoty
- čísla mezi -5 a 256
- prázdný řetězec, jednotlivé znaky
- krátké řetězce se vytváří jen jednou

```
import sys
a = 1
print sys.getrefcount(a)
a = ""
print sys.getrefcount(a)
```

Kopírování objektů

Vytvoření aliasu `b = a`

- odkaz na stejnou věc

Mělká kopie `b = a[:]` nebo `b = list(a)`

- vytváříme nový seznam, ale prvky tohoto seznamu jsou aliasy
- obecně i pro jiné typy než seznamy (knihovna `copy`)
  - `b = copy.copy(a)`

Hluboká kopie

- kompletní kopie všech dat – jak?
- obecné řešení (opět knihovna `copy`)
  - `b = copy.deepcopy(a)`

Práce se soubory

Práce se soubory v Pythonu

Způsob práce

- otevření souboru
- práce se souborem (čtení / zápis)
- zavření souboru

```
>>> soubor = open("/tmp/my_file", "w")
>>> print soubor
<open file '/tmp/my_file', mode 'w' at 0x7fc8f75d8420>
>>> soubor.write("Hasta la vista!\n")
>>> soubor.close()
```

Práce se soubory v Pythonu

Otevření souboru

- `open(jmeno, zpusob)`
- jméno souboru: řetězec (pozor na Windows a `'\\'`)
- způsob otevření:
  - čtení (`"r"`)
  - zápis (`"w"`) – přepíše soubor, pokud není, vytvoří jej
  - přidání na konec (`"a"`)
  - čtení i zápis (`"r+"`, `"w+"` nebo `"a+"`)
  - binární režim (přidat `"b"` k některému způsobu)

Práce se soubory v Pythonu

Čtení ze souboru

- `read(pocet)` – přečte daný počet znaků
- `read()` – přečte celý soubor, vrací řetězec
- `readline()` – přečte celý jeden řádek
- `readlines()` – přečte všechny řádky, vrací seznam řádků

Zápis do souboru

- `write(text)` – zapíše řetězec do souboru
  - neukončuje řádky, je třeba explicitně použít `'\n'`

Jiné

- `tell()` – aktuální pozice v souboru
- `seek(pozice)` – přesun pozice v souboru

Práce se soubory v Pythonu

Iterace po řádcích

```
for line in my_file:
    print line
```

Speciální blok `with`

- při jeho použití není třeba soubor zavírat
- zavře se sám při ukončení bloku

```
with open("/tmp/my_file", "r") as my_file:
    lines = my_file.readlines()

print lines
```

Shrnutí

Proměnné

- lokální vs. globální
- v Pythonu: drží odkaz na hodnotu

Předávání parametrů

- v Pythonu: něco mezi hodnotou a odkazem

Správa paměti

- v Pythonu: počítání referencí, automatický úklid

Kopírování objektů

- mělká vs. hluboká kopie
- užitečná knihovna copy

Práce se soubory

- otevření / zavření souboru; užitečný blok `with`
- zápis, čtení