Proměnné, paměť, rekurze

IB113
Radek Pelánek

2022

Rozcvička I

```
a = [3, 1, 7]
print(sorted(a))
print(a)
b = [4, 3, 1]
print(b.sort())
print(b)
```

Rozcvička II

```
a = ["magic"]
a.append(a)
print(a[1][1][1][1][1][0][1][0][0][0])
```

Dnes

„záludnější partie“

- globální a lokální proměnné
- reprezentace dat v paměti, kopírování
- předávání parametrů funkcím
- typy
- rekurze

základní témata obecně důležitá
detaily specifické pro Python

Dnešní programy

Varování: Dnešní ukázky programů jsou často „úchylné“.

- minimalistické ukázky důležitých jevů
- nikoliv pěkný, prakticky používaný kód

Python Tutor – vizualizace

Vizualizace použité ve slidech:

`http://www.pythontutor.com`

Odkazy na dnešní ukázky:

`https://www.fi.muni.cz/~xpelanek/IB113/tutor-codes.html`

Doporučeno projít si interaktivně.

Názvy proměnných – konvence

- konstanty – velkými písmeny
- běžné proměnné:
  - smysluplná slova
  - víceslovné názvy: `lower_case_with_underscores`
- krátké (jednopísmenné) názvy:
  - indexy
  - souřadnice: `x`, `y`
  - pomocné proměnné s velmi lokálním využitím

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

Rozsah proměnných obecněji

- proměnné jsou viditelné v rámci svého „rozsahu“
- rozsahem mohou být:
  - funkce
  - moduly (soubory se zdrojovým kódem)
  - třídy (o těch se dozvíme později)
  - a jiné (závisí na konkrétním jazyce)

relevantní terminologie: „namespace“, „scope“

Globální a lokální proměnné

```
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)      # ERROR!
# NameError: name 'b' is not defined
```

Globální a lokální proměnné

Python 3.6

```
1  a = "This is global."
2
3  def example1():
4      b = "This is local."
5      print(a)
6      print(b)
7
8  example1()    # This is global.
9                # This is local.
10 print(a)      # This is global.
11 print(b)      # ERROR!
```

Edit code | Live programming

Print output (drag lower right corner to resize)

```
This is global.
```

Frames | Objects

Global frame
- a: "This is global."
- example1: → function example1()

example1
- b: "This is local."

Globální a lokální proměnné

vytváříme novou lokální proměnnou, neměníme tu globální

```
a = "Think global."

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

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

Globální a lokální proměnné

Python 3.6

```
1  a = "Think global."
2
3  def example2():
4      a = "Act local."
5      print(a)
6
7  print(a)       # Think global.
8  example2()     # Act local.
9  print(a)       # Think global.
```

Edit code | Live programming

st executed

ecute

set a breakpoint; use the Back and Forward buttons to jump there.

Print output (drag lower right corner to resize)

```
Think global.
```

Frames Objects

Global frame

a "Think global."

example2

function example2()

example2

a "Act local."

Globální a lokální proměnné

Jak měnit globální proměnné?

```
a = "Think global."

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

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

Lokální proměnné: deklarace

lokální proměnná vzniká, pokud je přiřazení kdekoli uvnitř funkce

---

```
a = "Think global."
def example4(change_opinion=False):
    print(a)
    if change_opinion:
        a = "Act local."
        print("Changed opinion:", a)

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

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

Globální a lokální proměnné: příklad

```python
a = 5

def test1():
    print(a)

def test2():
    print(a)
    a = 8

test1()  # 5
test2()  # UnboundLocalError
```

Rozsah proměnných: for cyklus

- rozsah proměnné v Pythonu není pro „dílčí blok kódu“, ale pro celou funkci (resp. globální kód)
- záludný překlep: řídicí proměnná for cyklu použita po ukončení cyklu

---

```
n = 9
for i in range(n):
    print(i)
if i % 2 == 0:
    print("I like even length lists")
```

Globální a lokální proměnné

Doporučení:

- vyhýbat se globálním proměnným
- pouze ve specifických případech, např. globální konstanty

Proč?

„Globální proměnné jsou zlo“

- špatná modularita, horší znovupoužitelnost kódu
- horší čitelnost kódu
- náročnější testování, překvapivé interakce
- nepořádek ve jmenném prostoru
- zdroj chyb (např. kolize lokální vs globální)

obecně: „lokalita kódu“ je užitečná

více argumentů: google “global variables are bad / evil”

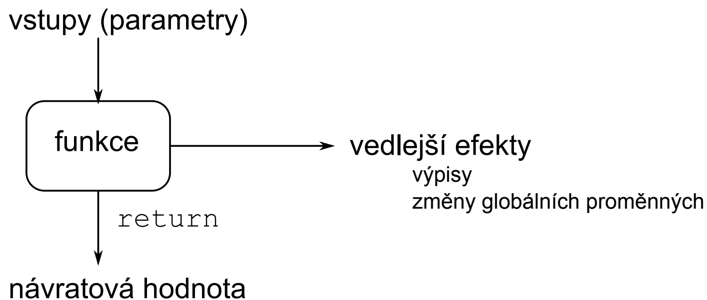
Globální proměnné: alternativy

- předávání parametrů funkcím a vracení hodnot z funkcí
- objekty (probereme stručně později)
- a další (nad rámec předmětu): statické proměnné (C, C++, Java, ...), návrhové vzory, ...

příklad: herní plán

Funkce

Připomenutí z dřívější přednášky



Funkce bez vedlejších efektů

„čistá funkce" = funkce bez vedlejších efektů

Reklama

Čisté funkce jsou vaši přátelé!

- ladění
- modularita
- přemýšlení o problému
- čitelnost kódu

Funkce: vedlejší efekty

- změna měnitelných parametrů
  - OK, ale nemíchat s návratovou hodnotou, vhodně pojmenovat, dokumentovat
- změna globálních proměnných (které nejsou parametry)
  - většinou cesta do pekla
- změna stavu systému (libovolné „výpisy“, zápis do souboru, databáze, odeslání na tiskárnu, ...)
  - „procedury“
  - nutnost, ale nemíchat chaoticky s výpočty

Proměnné a paměť

Proměnné v různých jazycích

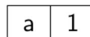
- pojmenované místo v paměti
- odkaz na místo v paměti *(Python)*
- kombinace obou možností

Přiřazení

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

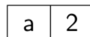
Proměnné a paměť

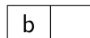
```
int a, b;
a = 1;
```



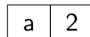
b

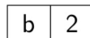
```
a = 2;
```






```
b = a;
```

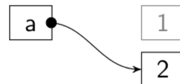




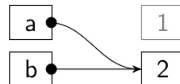
Jazyk C
Proměnné jako hodnoty

```
a = 1;
```

```
a = 2;
```




```
b = a;
```



Jazyk Python
Proměnné jako odkazy

Proměnné a paměť

funkce `id()` – vrací „identitu“ objektu (± adresa v paměti)

```python
a = 1000
b = a

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

b += 1

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

```python
a = [1]
b = a

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

b.append(2)

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

Rovnost vs. „stejná identita“

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a == b)            # True
print(a is b)            # False
print(id(a) == id(b))    # False
```

---

operátor `is` – stejná identita

Předávání parametrů v Pythonu

- paramater drží odkaz na předanou proměnnou
- změna parametru změní i předanou proměnnou
- *neměnitelné typy* (čísla, řetězce, ntice)
  - nelze „změnit“, každé přiřazení znamená vytvoření odkazu na něco nového
  - neprojeví se mimo funkci
- *měnitelné typy* (seznam, slovník)
  - lze měnit, změna typu `append`, `x[3] = 8` se projeví i mimo funkci
  - pozor: přiřazení znamená změnu odkazu, neprojeví se mimo funkci

Předávání parametrů: ukázky

číselný parametr je neměnitelný, toto nic neprovede

```
def update_param_int(x):
    x = x + 1

a = 1
print(a)  # 1
update_param_int(a)
print(a)  # 1
```

Předávání parametrů: ukázky

seznam je měnitelný, změna se projeví i mimo funkci

```
def update_param_list(x):
    x.append(3)

a = [1, 2]
print(a)  # [1, 2]
update_param_list(a)
print(a)  # [1, 2, 3]
```

Předávání parametrů: ukázky

odkaz se změní na nový seznam, původní je nezměněn – změna se neprojeví mimo funkci

---

```
def change_param_list(x):
    x = [1, 2, 3]

a = [1, 2]
print(a)  # [1, 2]
change_param_list(a)
print(a)  # [1, 2]
```

Předávání parametrů: kvíz

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

t = [1, 2]
test(t)
print(t)
```

Předávání parametrů: vizualizace

Python 3.6

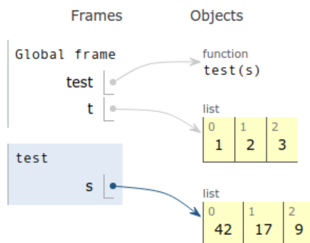
```
1  def test(s):
2      s.append(3)
3      s = [42, 17]
4      s.append(9)
5      print(s)
6
7  t = [1, 2]
8  test(t)
9  print(t)
```

Edit code | Live programming

ted

eakpoint; use the Back and Forward buttons to jump there.

Print output (drag lower right corner to resize)



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)`

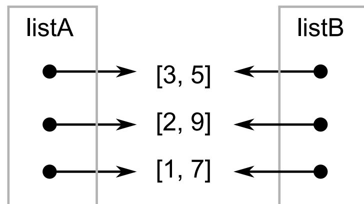
Hluboká kopie

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

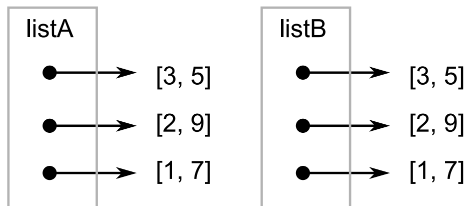
Kopírování objektů

```
listA = [[3, 5], [2, 9], [1, 7]]
```

mělká kopie



hluboká kopie



Příklad

```
import copy
a = [[3, 5], [2, 9], [1, 7]]
b = a
c = a[:]
d = copy.deepcopy(a)
a[0][0] = 100
print(b[0][0], c[0][0], d[0][0])
a[0] = [200, 200]
print(b[0][0], c[0][0], d[0][0])
```

Datové typy

Datové typy určují:

- význam dat
- operace, které lze s daty provádět
- hodnoty, kterých mohou data nabývat

Typy v Pythonu

- `bool`
- `int`, `float`, `complex` – číselné typy
- `str` – řetězec
- `list` – seznam
- `tuple` – n-tice
- `dict` – slovník
- `set` – množina

(výběr nejdůležitějších)

Typy: kvíz

```
print(type(3))
print(type(3.0))
print(type(3==0))
print(type("3"))
print(type([3]))
print(type((3,0)))
print(type({3:0}))
print(type({3}))
```

Typy: kvíz

```
type(3)           # <class 'int'>
type(3.0)         # <class 'float'>
type(3==0)        # <class 'bool'>
type("3")         # <class 'str'>
type([3])         # <class 'list'>
type((3,0))       # <class 'tuple'>
type({3:0})       # <class 'dict'>
type({3})         # <class 'set'>
```

Typy: měnitelnost

- neměnitelné (immutable):
  `bool, int, float, str, tuple`
- měnitelné (mutable):
  `list, dict, set`

Příklady, kde důležité:

- změna indexováním
- předávání parametrů funkcím
- indexování slovníku

None

- `None` – jedinečná hodnota typu `NoneType`
- význam: „prázdné“, „žádná hodnota“
- využití: např. defaultní hodnota parametrů funkcí
- implicitní návratová hodnota z funkcí (pokud nepoužijeme `return`)

Pravdivostní hodnota

```
if value:
    print("foo")
```

Pro které z těchto hodnot `value` se vypíše `foo`?

`True, False, 3, 0, 3.0, -3, [3], [], "3", "", None`

Pravdivostní hodnota

test je úspěšný ("true") vždy, kromě následujících případů:

- konstanty: `None`, `False`
- nulové hodnoty u numerických typů: `0, 0.0, 0j`
- prázdné sekvence (nulová délka měřená pomocí `len`): "", [], (), {}

(mírně zjednodušeno, např. u objektů může být komplikovanější)

Dynamická kontrola typů

- `type(x)` – zjištění typu
- `isinstance(x, t)` – test, zda je proměnná určitého typu

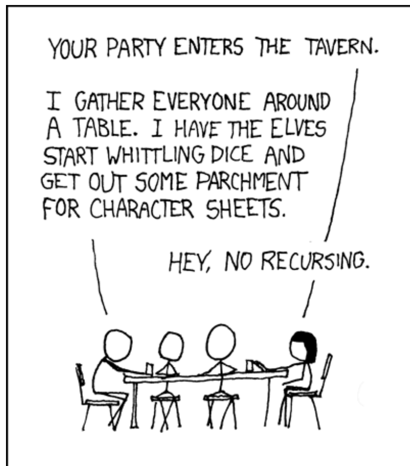
```
values = [3, 8, "deset", 4, "dva", "sedm", 6]
s = 0
for value in values:
    if isinstance(value, int):
        s += value
    else:
        print("Not int:", value)
print("Sum of ints:", s)
```

Rekurze

- použití funkce při její vlastní definici
- volání sebe sama (s jinými parametry)

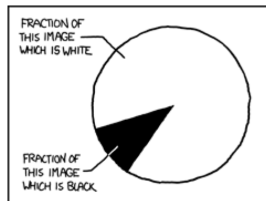
To iterate is human, to recurse divine. (L. Peter Deutsch)

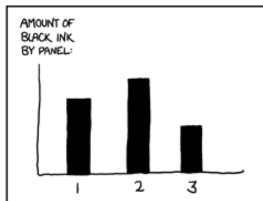
xkcd: Tabletop Roleplaying

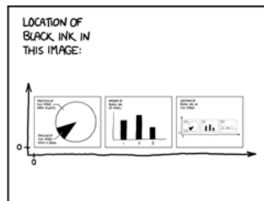


https://xkcd.com/244/

Sebereference







https://xkcd.com/688/

Sebereferenční test

1. Které písmeno není správnou odpovědí na žádnou otázku:
   (A) A (B) C (C) B
2. Odpověď na 3. otázku je:
   (A) stejná jako na 1. otázku (B) stejná jako na 2. otázku (C) jiná než odpovědi na 1. a 2. otázku
3. První otázka, na kterou je odpověď A, je otázka:
   (A) 1 (B) 2 (C) 3

Hlavolamikon

Rekurze a sebereference

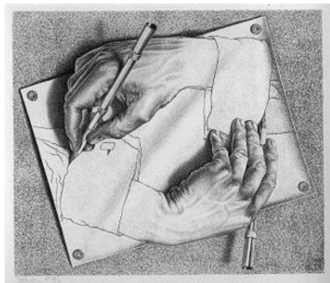
Rekurze a sebereference – klíčové myšlenky v informatice

některé souvislosti:

- matematická indukce
- funkcionální programování
- rekurzivní datové struktury (např. stromy)
- gramatiky
- logika, neúplnost
- nerozhodnutelnost, diagonalizace

Rekurze a sebereference

... nejen v informatice



M. C. Escher; Drawing Hands, 1948

Gödel, Escher, Bach: An Eternal Golden Braid; Douglas Hofstadter

Faktoriál

$$n! = 1 \cdot 2 \cdots (n-1) \cdot n$$

$$f(n) = \begin{cases} 1 & \text{pokud } n = 1 \\ n \cdot f(n-1) & \text{pokud } n > 1 \end{cases}$$

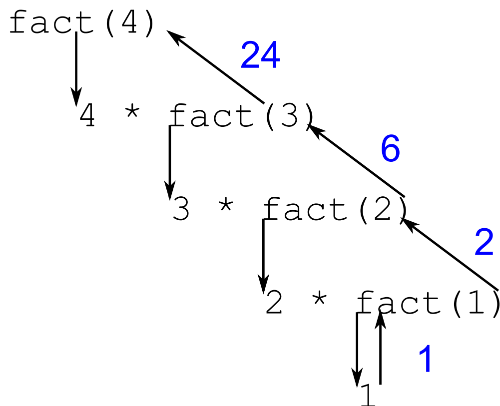
Faktoriál iterativně (pomocí cyklu)

```python
def fact(n):
    f = 1
    for i in range(1,n+1):
        f = f * i
    return f
```

Faktoriál rekurzivně

```
def fact(n):
    if n == 1: return 1
    else: return n * fact(n-1)
```

Faktoriál rekurzivně – ilustrace výpočtu



Příklad: výpis čísel

Vymyslete funkci, která:

- vypíše čísla od 1 do N
- pomocí rekurze – bez použití cyklů `for`, `while`

Příklad: výpis čísel

Vymyslete funkci, která:

- vypíše čísla od 1 do N
- pomocí rekurze – bez použití cyklů `for`, `while`

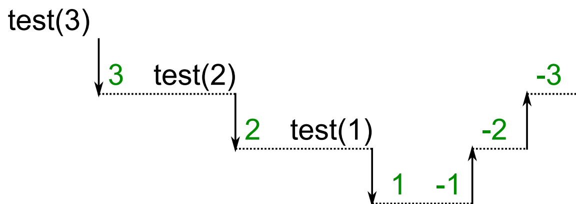
```
def sequence(n):
    if n > 1:
        sequence(n-1)
    print(n)
```

Co udělá tento program?

```
def test(n):
    print(n)
    if n > 1:
        test(n-1)
    print(-n)

test(5)
```

Ilustrace zanořování



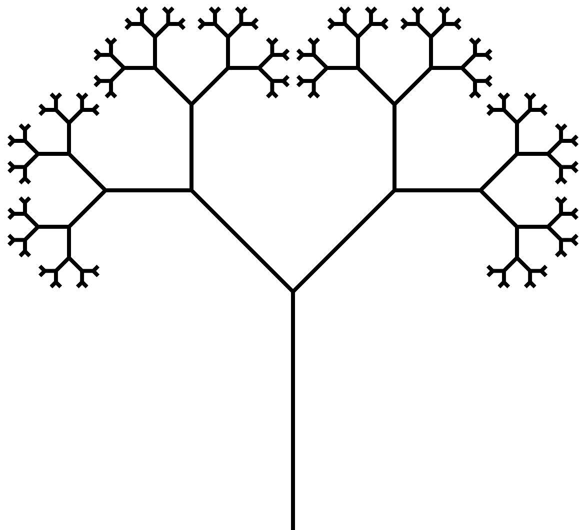
Nepřímá rekurze

```
def even(n):
    print("even", n)
    odd(n-1)

def odd(n):
    print("odd", n)
    if n > 1:
        even(n-1)

even(10)
```

Rekurzivní stromeček



Rekurzivní stromeček

nakreslit stromeček znamená:

- udělat stonek
- nakreslit dva menší stromečky (pootočené)
- vrátit se na původní pozici

Stromeček želví grafikou

```
def tree(length):
    forward(length)
    if length > 10:
        left(45)
        tree(0.6 * length)
        right(90)
        tree(0.6 * length)
        left(45)
    back(length)
```

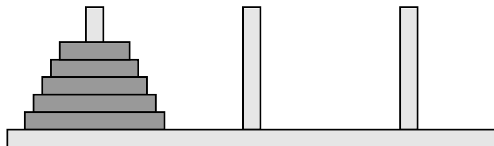
Podoby rekurze, odstranění rekurze

- koncová rekurze (tail recursion)
  - rekurzivní volání je poslední příkaz funkce
  - lze vesměs přímočaře nahradit cyklem
- „plná“ rekurze
  - „zanořující se“ volání
  - např. stromeček
  - lze přepsat bez použití rekurze za použití zásobníku
  - rekurzivní podoba často výrazně elegantnější

Hanojské věže aneb O konci světa

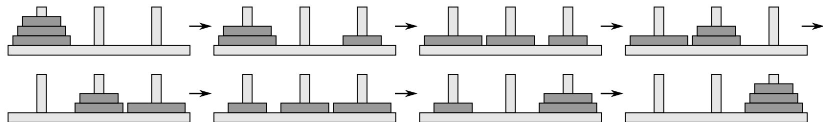
- video: `https://www.youtube.com/watch?v=8yaoED8jc8Y`
- klášter kdesi vysoko v horách u města Hanoj
- velká místnost se třemi vyznačenými místy
- 64 různě velkých zlatých disků
- podle věštby mají mniši přesouvat disky z prvního na třetí místo
- a až to dokončí ...

Hanojské věže: pravidla



- N disků různých velikostí naskládaných na sobě
- vždy může být jen menší disk položen na větším
- možnost přesunout jeden horní disk na jiný kolíček
- cíl: přesunout vše z prvního na třetí

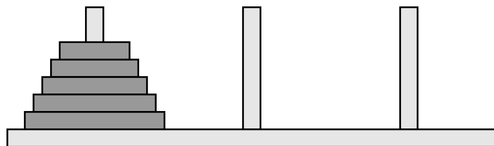
Hanojské věže: řešení



Hanojské věže: výstup programu

```
>>> solve(3, "A", "B", "C")
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
```

Hanojské věže: rekurzivní řešení



```
def solve(n, start, end, auxiliary):
    if n == 1:
        print(start, "->", end)
    else:
        solve(n-1, start, auxiliary, end)
        solve(1, start, end, auxiliary)
        solve(n-1, auxiliary, end, start)
```

Hanojské věže: souvislosti

- Sierpińského fraktál
- Pascalův trojúhelník

$\Rightarrow$ `https://www.youtube.com/watch?v=8yaoED8jc8Y`

Kontrolní otázky

- Předpokládejme, že `y` je seznam. Jaký je rozdíl mezi `x = y` a `x = y[:]`?
- Jaký je význam pojmů „globální proměnná" a „lokální proměnná"?
- Jaké jsou nevýhody použití globálních proměnných?
- Jaký je význam pojmů „hluboká kopie" a „mělká kopie"?
- Jaké jsou v Pythonu hlavní vestavěné typy? Které z nich jsou měnitelné a které neměnitelné?
- Co je to rekurze?
- Jak zapíšeme výpočet faktoriálu: a) za použití rekurze, b) bez použití rekurze?

Shrnutí

- představa o reprezentaci v paměti je potřeba
- globální, lokální proměnné
- předávání parametrů funkcím
- mělká vs. hluboká kopie
- typy, měnitelné, neměnitelné
- rekurze

příště: regulární výrazy, zpracování textu