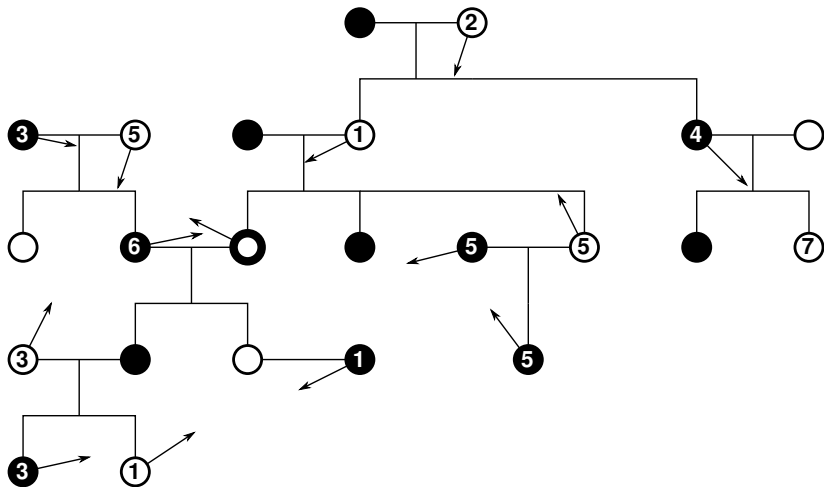


Objekty

IB111 Základy programování
Radek Pelánek

2017

Rozcvička



Řešením je identifikace místa poblíž Brna. Pokud je právě 6. 11. 2009, najdete na tomto místě další zprávu.

udržování dat pohromadě

- záznamy o knihách
 - název knihy, autor, ISBN, rok vydání, ...
- postava v počítačové hře
 - souřadnice (x, y), energie, vybavení, ...

Jak reprezentovat?

ntice, seznamy

```
book = ["Godel, Escher, Bach: An Eternal Golden Braid",  
        "Douglas R. Hofstadter",  
        "978-0465026562",  
        "1999"]  
print(book[0]) # title  
print(book[1]) # author
```

nepojmenované – nutno si pamatovat pořadí položek (v hlavě,
v komentářích, v konstantách)

toto není pěkné

Jak reprezentovat?

slovníky – pojmenované položky

```
book = {"title": "Godel, Escher, Bach: An Eternal Golden  
        "author": "Douglas R. Hofstadter",  
        "isbn": "978-0465026562",  
        "year": "1999"}
```

```
print(book["title"])  
print(book["author"])
```

toto není špatné, ale ...

- vlastní datové typy: nejen data, ale i funkcionality
- schovávání škaredých detailů

příklad: dvojrozměrné matice z přednášky o datových typech

- použitá reprezentace:
 - reprezentovány pomocí seznamu seznamů
 - nepěkný způsob zjišťování velikosti matice
- co bychom chtěli?
 - spolu s maticí si udržovat informace o její velikosti
 - mít něco, co kontroluje přístupy do matice

Motivace III

objektům se nelze vyhnout (v Pythonu a většině dalších jazyků) – zápis volání funkcí přes operátor „tečka“ (řetězce, seznamy, slovníky, soubory)

```
t = "hello"  
print(t.upper())  
s = [7, 14, 42, 0]  
s.sort()  
s.append(9)  
d = {"a": 1, "b": 2}  
s = d.keys()  
f = open("myfile.txt")  
line = f.readline()
```

Záznamy, struktury

- datový typ složený z více položek
- typicky fixní počet položek, deklarované typy
- C: struct
- Pascal: record

Objekty

- často rozšíření struktur
- kombinují data a funkce (metody)
- C++, Java, Python: class

Objektově orientované programování

Varování

Toto není objektově orientované programování.

- zde probíráme základy využití objektů v Pythonu
 - primárně náhrada za záznamy/struktury
 - jednoduché využití metod
- „objektově orientované programování“ je podstatně složitější
 - zapouzdření, veřejné/soukromé atributy
 - dědičnost
 - polymorfismus
 - metodika návrhu programů
- viz navazující kurzy

Objekty v Pythonu

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def say_hello(self):
        print(self.name + " says hello.")

    def rename_to(self, new_name):
        print(self.name + " renamed to " + new_name + ".")
        self.name = new_name
```

Slovníček pojmů

(zjednodušeně)

třída	obecný uživatelsky definovaný typ, charakterizován atributy
objekt	konkrétní <i>instance</i> třídy
atribut	„to co je za tečkou“, atributy jsou datové a funkční
datový atribut	proměnná patřící třídě/objektu (*)
metoda	funkce, která je vázaná na danou třídu
konstruktor	inicializační metoda, vytváří objekt

(*) je rozdíl mezi atributem třídy a objektu, více později

Pojmy – intuitivní ilustrace

- třída: Pes
- objekt (instance): Alík
- datové atributy: rasa, jméno, věk, poloha
- metody: štěkej, popoběhni

Objekty v Pythonu

- definice třídy: `class MyObject:`
- definice metod: `def do_something(self, parameter):`
 - `self` – povinný první parametr
 - odkaz na aktuální objekt
 - `self` není klíčovým slovem, jen silnou konvencí
- objekty
 - instance třídy, vlastní atributy
 - přístup k atributům pomocí *tečkové notace*

Vytvoření objektu

- speciální metoda (konstruktor): `__init__`
- především inicializace atributů

—

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

vytvoření objektu:

```
homer = Person("Homer Simpson", 34)
print(homer.name)
print(homer.age)
```

Metody: definice a použití

- tečková notace
- objekt před tečkou se předá jako první parametr (`self`)

```
class Person:  
    ...  
  
    def say_hello(self):  
        print(self.name + " says hello.")  
  
homer.say_hello()  
# Homer Simpson says hello.
```

Metody a modifikace objektu

```
class Person:
    ...

    def rename_to(self, new_name):
        print(self.name + " renamed to " + new_name + ".")
        self.name = new_name

homer.rename_to("Homer Jay Simpson")
# Homer Simpson renamed to Homer Jay Simpson.
print(homer.name)
# Homer Jay Simpson
```


Přístup k atributům

- přímý:
 - přistupujeme přímo k atributu, můžeme i měnit
 - `homer.name`
- nepřímý:
 - pomocí metod („getters and setters“)
 - `get_name`, `set_name`

Který zvolit?

- závisí na použití
- objekt jen pro držení dat: přímý přístup je nejspíše OK
- schováváme v objektu složitější *vnitřnosti*: pište metody

PEP8 konvence: názvy

- jména tříd:
CapWords (velká počáteční písmena slov, bez oddělovačů slov)
- atributy (datové, metody): stejně jako běžné proměnné/funkce
`lowercase, lower_case_with_underscores`

Příklady

- knihy
- studenti
- čas
- matice
- rodokmen

Příklad: knihy

- práce se seznamem knih
- kniha má: název, autora, ISBN
- chceme seznam načítat/ukládat do souborů

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn

geb = Book("Godel, Escher, Bach",
          "Hofstadter",
          "978-0465026562")
neverwhere = Book("Neverwhere",
                  "Gaiman",
                  "978-0380789016")

print(neverwhere.author)
```

Knihy: načítání a ukládání

```
def load_library(filename):
    book_list = []
    with open(filename, "r") as f:
        for line in f:
            a, t, i = line.split(";")
            book_list.append(Book(t, a, i))
    return book_list

def save_library(filename, book_list):
    with open(filename, "w") as f:
        for book in book_list:
            f.write(book.title + ";" +
                    book.author + ";" +
                    book.isbn + "\n")
```

Knihy: použití

```
save_library("library.csv", [geb, nowhere])  
books = load_library("library.csv")  
for b in books: print(b.title)
```

vytvořený / načítaný soubor library.csv:

```
Godel, Escher, Bach;Hofstadter;978-0465026562  
Nowhere;Gaiman;978-0380789016
```

CSV jsou super, ty chcete!

- CSV = Comma-separated values
- formát pro reprezentaci tabulkových dat
- jednoduchý textový formát, položky odděleny čárkami (nebo něčím podobným)
- vlastně ne úplně jednoduchý a pro reálné aplikace chcete použít knihovny jako je `csv` nebo `pandas`, každopádně CSV jsou super, viz též <https://xkcd.com/1301/>

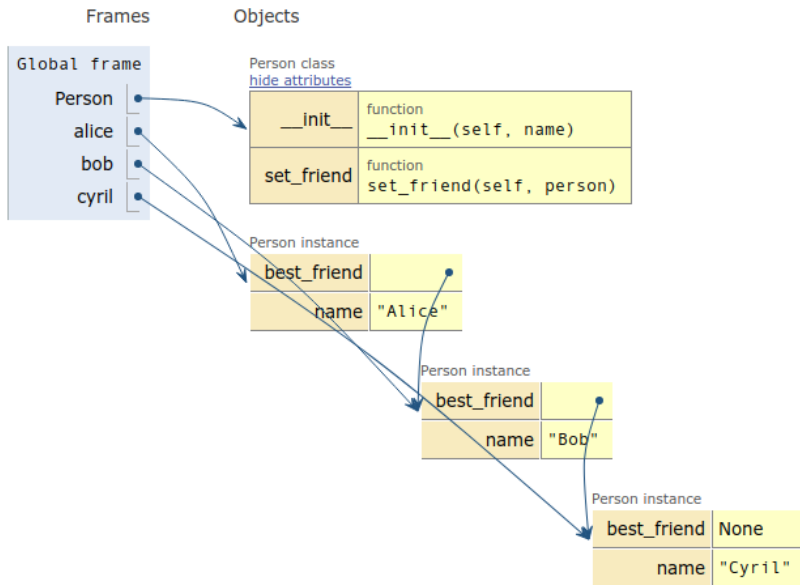
Kahoot kód: otázky 2, 3, 4, 5

```
class Person:
    def __init__(self, name):
        self.name = name
        self.best_friend = None

    def set_friend(self, person):
        self.best_friend = person
```

```
alice = Person("Alice")
bob = Person("Bob")
cyril = Person("Cyril")
alice.set_friend(bob)
bob.set_friend(cyril)
```

Python Tutor vizualizace



Kahoot kód: otázky 6, 7, 8, 9

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def shift(self, x, y):
        self.x += x
        self.y += y

a = Point(3, 2)
b = Point(1, 4)
points = [a, b, Point(0, 2)]
```

Příklad: studenti a kurzy

- reprezentace studentů a kurzů
- kurz má seznam zapsaných studentů

—

```
class Student:
    def __init__(self, uco, name):
        self.uco = uco
        self.name = name
```

Příklad: studenti a kurzy

```
class Course:
    def __init__(self, code):
        self.code = code
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def print_students(self):
        i = 1
        for s in self.students:
            print(str(i) + ". ",
                  str(s.uco),
                  s.name, sep="\t")
            i += 1
```

Studenti a kurzy: použití

```
jimmy = Student(555007, "James Bond")
ib111 = Course("IB111")
ib111.add_student(Student(555000, "Luke Skywalker"))
ib111.add_student(jimmy)
ib111.add_student(Student(555555, "Bart Simpson"))
ib111.print_students()
```

```
# 1.      555000  Luke Skywalker
# 2.      555007  James Bond
# 3.      555555  Bart Simpson
```

Studenti a kurzy: řazení

Co když chceme seřazený seznam studentů?

- přirozené `sorted(self.students)` nefunguje – není definováno uspořádání na studentech
- možnosti:
 - definovat uspořádání na studentech – metoda `__lt__(self, other)`
 - `sorted(self.students, key=lambda s: s.name)`
 - použít pomocný seznam a ten seřadit:

```
tmp = [(s.name, s.uco) for s in self.students]
```
- co uspořádání abecedně podle příjmení?

Příklad: reprezentace času

(jednoduchá naivní verze, pořádně: knihovna datetime)

```
class Time:
    def __init__(self, h, m, s):
        self.hours = h
        self.minutes = m
        self.seconds = s
        self.validate()

    def validate(self):
        if self.seconds >= 60:
            self.minutes += self.seconds // 60
            self.seconds = self.seconds % 60
        if self.minutes >= 60:
            self.hours += self.minutes // 60
            self.minutes = self.minutes % 60
```


Reprezentace času

```
class Time: # ... continued ...
    def pretty_print(self):
        print("{}:{:02}:{:02}".format(self.hours,
                                       self.minutes,
                                       self.seconds))

    def add_seconds(self, sec):
        self.seconds += sec
        self.validate()
```

```
t = Time(1, 30, 72)
t.pretty_print()
t.add_seconds(107)
t.pretty_print()
```

```
# 1:07:12
```

Příklad: matice

oproti dřívější ukázce práce s maticemi:

- chceme uchovávat i jejich velikost
- chceme bezpečný přístup k prvkům

—

```
class Matrix:
    def __init__(self, rows, cols):
        self.rows = rows
        self.cols = cols
        self.matrix = [[0 for i in range(self.cols)]
                       for i in range(self.rows)]
```

Matrice: metody

```
class Matrix: # ... continued ...
    def check(self, row, col):
        if row < 0 or row >= self.rows:
            print("Bad row index.")
            return False
        if col < 0 or col >= self.cols:
            print("Bad column index.")
            return False
        return True

    def get(self, row, col):
        if self.check(row, col):
            return self.matrix[row][col]

    def set(self, row, col, value):
        if self.check(row, col):
            self.matrix[row][col] = value
```

Malice: násobení

```
def matrix_mult(matL, matR):
    if matL.cols != matR.rows:
        print("Incompatible matrices.")
        return
    result = Matrix(matL.rows, matR.cols)
    for i in range(matL.rows):
        for j in range(matR.cols):
            for k in range(matL.cols):
                result.set(i, j, result.get(i, j) +
                            matL.get(i, k) *
                            matR.get(k, j))
    return result
```

čistě ilustrativní příklad
reálnější verze by využívala:

- přetížení operátoru krát: `__mul__`
- výjimky
- ... a nebo rovnou `numpy`

Příklad: rodokmen

- rozšíříme objekt pro osoby o rodičovské vztahy
- pro jednoduchost jen jeden rodič
 - uvažujeme jen ženy – *Mater certa, pater incertus est.*
- rodokmen \sim strom potomkyň
- pro každou ženu si pamatujeme:
 - jméno
 - matka
 - seznam dcer
- matka, seznam dcer – odkazy na objekty (abychom s nimi mohli pracovat), nikoliv „jména“

```
class Woman:
    def __init__(self, name):
        self.name = name
        self.mother = None
        self.daughters = []

    def add_daughter(self, daughter):
        self.daughters.append(daughter)
        daughter.mother = self
```

```
alice = Woman("Alice")
mary = Woman("Mary")
lisa = Woman("Lisa")
carol = Woman("Carol")
alice.add_daughter(mary)
alice.add_daughter(lisa)
mary.add_daughter(carol)
```

Základní použití

```
print(carol.mother.name)           # Mary
print(alice.mother.name)           # error message
print(len(lisa.mother.daughters))  # 2

def siblings(a, b):
    return a.mother != None and a.mother == b.mother

print(siblings(mary, alice))       # False
print(siblings(mary, lisa))        # True
```


Výpis rodokmenu

Chceme vypsat textově znázorněný rodokmen – odsazení podle generací:

```
-----  
Alice  
    Mary  
        Carol  
            Betty  
    Lisa  
        Maria  
-----
```

Jak to udělat?

Rekurze!

```
def draw_family_tree(woman, level=0):  
    print(("    " * level) + woman.name)  
    for daughter in woman.daughters:  
        draw_family_tree(daughter, level + 1)
```

Rodokmen: nejstarší předkyně

Rekurzivně:

```
def oldest_ancestor(woman):  
    if woman.mother == None:  
        return woman  
    return oldest_ancestor(woman.mother)
```

Iterativně:

```
def oldest_ancestor2(woman):  
    while woman.mother != None:  
        woman = woman.mother  
    return woman
```

Rodokmen: počet potomkyň

Chceme vypočítat počet všech potomkyň (i nepřímých).

Rodokmen: počet potomkyň

Chceme vypočítat počet všech potomkyň (i nepřímých).

rekurze s návratovou hodnotou:

- funkce `count_offspring()` vrací počet potomkyň
- hodnotu vypočítáme pomocí volání `count_offspring()` na dcerách

Rodokmen: počet potomkyň

```
def count_offspring(woman):  
    count = 0  
    for daughter in woman.daughters:  
        count += 1 + count_offspring(daughter)  
    return count
```

Pozn. `count_offspring` by též mohla být metoda třídy `Woman`

Varování: Statické atributy

- definovány přímo ve třídě
- patří samotné třídě, ne objektům
- mají svůj smysl, ale v tuto chvíli spíše zdroj chyb
- v tomto předmětu **raději nepoužívejte**

```
class MyClass:
```

```
    x = 0
```

```
    def __init__(self, n):
```

```
        self.y = n
```

```
print(MyClass.x)           # 0
```

```
my_object = MyClass(17)
```

```
print(my_object.y)        # 17
```

```
print(my_object.x)        # 0 (same as MyClass.x)
```

Statické atributy: ilustrace problému

```
class Person:
    hobbies = []
    def __init__(self, name):
        self.name = name
    def add_hobby(self, hobby):
        self.hobbies.append(hobby)
```

```
mirek = Person("Mirek Dusin")
mirek.add_hobby("running")
mirek.add_hobby("world peace")
bidlo = Person("Dlouhe Bidlo")
bidlo.add_hobby("alkohol")
```

```
print(mirek.hobbies)
```


Python Tutor vizualizace

Python 3.6

```
1 class Person:
2     hobbies = []
3     def __init__(self, name):
4         self.name = name
5     def add_hobby(self, hobby):
6         self.hobbies.append(hobby)
7
8 mirek = Person("Mirek Dusin")
9 mirek.add_hobby("running")
10 mirek.add_hobby("world peace")
11 bidlo = Person("Dlouhe Bidlo")
12 bidlo.add_hobby("alkohol")
```

[Edit code](#) | [Live programming](#)

is just executed
- execute
de to set a breakpoint; use the Back and Forward buttons to jump there.

Frames Objects

Global frame

- Person
- mirek
- bidlo

Person class
[hide attributes](#)

__init__	function	__init__(self, name)	
add_hobby	function	add_hobby(self, hobby)	
hobbies	list	0	1 2
		"running"	"world peace" "alkohol"

Person instance

name	"Mirek Dusin"
-------------	---------------

Person instance

name	"Dlouhe Bidlo"
-------------	----------------

49 / 51

Varování: kopírování objektů

Co udělá?

```
a = Matrix(2, 2)
a.set(0, 1, 6)
b = a
b.set(0, 0, 7)
print(a.matrix)
```

Vytvoření aliasu, nikoliv kopírování (podobně jako u seznamů)!
Jak kopírovat?

- knihovna `copy`: `copy` (mělká kopie), `deepcopy` (hluboká kopie)
- vytvořit speciální kopírovací konstruktor

Záznamy/struktury

- používáme pro seskupení souvisejících dat
- v Pythonu přímo nejsou

Objekty

- složitější než záznamy
- data + metody
- v tomto předmětu je používáme primárně jako záznamy s jednoduchými metodami
- více o objektově orientovaném programování (dědičnost, polymorfismus, objektový návrh) v navazujících předmětech