IB111 Úvod do programování skrze Python
Přednáška 9

Složené datové typy, objekty

Nikola Beneš

13. listopad 2017

Motivace

Udržování dat pohromadě

- příklad: chceme uchovávat záznamy o knihách
  - název knihy, autor, ISBN, apod.

Jak reprezentovat?

- ntice (tuples)
  - neměnné, nepojmenované
- seznamy
  - měnitelné, nepojmenované (je název první nebo autor první?)
- slovníky

Jde to i jinak?

Motivace

Schovávání škaredých detailů

- příklad: vzpomeňte si na dvojrozměrné matice z přednášky o datových typech
  - 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

Vlastní datové struktury

- co kdybychom chtěli mít nějakou vlastní datovou strukturu?
- např. zřetězený seznam, nějaký druh stromové struktury, ...

Operátor „tečka“

- už jsme viděli u seznamů, slovníků apod.

```
s = [7, 14, 42, 0]
s.sort()
s.append(9)

d = {"a": 1, "b": 2}
s = d.items()
```

- co to znamená, jak se to liší od volání funkcí?

Složené datové typy

Záznamy, struktury

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

Objekty

- často rozšíření struktur
- kombinují data a funkce (metody)
- C++, Java: `class`
- mnohem komplikovanější
  - dědičnost, polymorfismus

Objekty v Pythonu

Varování

Pozor!

- toto není objektově orientované programování (OOP)
- OOP je *podstatně* složitější
- zde používáme objekty jen *velmi jednoduchým* způsobem
  - náhrada za záznamy (struktury)

Objekty v Pythonu

Objekty

- atributy
- typ objektu: třída (`class`)
  - definuje metody – funkce, které pracují s objekty

Třídy

```
class MyObject:
    # definice metod
```

Metody

```
    def do_something(self, some_parameter):
        # nějaký kód
```

- `self` je povinný první parametr
  - odkaz na aktuální objekt

Objekty v Pythonu

Inicializace objektu – speciální metoda `__init__`

- pomocí ní inicializujeme atributy objektů
- k atributům přistupujeme pomocí tečkové notace

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

Vytvoření objektu

```
homer = Person("Homer Simpson", 34);

print(homer.name)    # Homer Simpson
print(homer.age)     # 34
```

Objekty v Pythonu

Použití metod

- opět 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.
```

Objekty v Pythonu

Použití metod

- metody mohou mít i další parametry
- metody mohou modifikovat objekt

```
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
```

Objekty v Pythonu

Přístup k atributům

- přímý: `homer.name`
  - atributy tak můžeme i měnit
- nepřímý: pomocí metod

Který zvolit?

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

Příklad 1 – knihovna

- chceme si pamatovat seznam knih
- kniha má
  - název
  - autora
  - ISBN
- chceme seznam načítat/ukládat do souborů

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

dune = Book("Frank Herbert", "Dune", "978-0441172719")
temno = Book("Bohuslav Balcar & Petr Štěpánek",
             "Teorie množin", "80-200-0470-X")
```

```
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(a, t, i))
    return book_list

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

save_library("library.csv", [dune, temno])
```

Příklad 2 – databáze studentů a předmětů

- chceme jednoduchou databázi předmětů a studentů
- student má
  - UČO
  - jméno
- předmět má
  - kód
  - seznam studentů

Příklady použití objektů v Pythonu

studenti

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

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) + "\t" + s.name)
            i += 1
```

Příklady použití objektů v Pythonu

```
jimmy = Student(555007, "James Bond")

ib111 = Course("IB111")

ib111.add_student(Student(555000, "Jan Novák"))
ib111.add_student(jimmy)
ib111.add_student(Student(555555, "Kryštof Harant"))

ib111.print_students()
# 1. 555000        Jan Novák
# 2. 555007        James Bond
# 3. 555555        Kryštof Harant
```

Příklad 3 – práce s časem

```
class Time:
    def __init__(self, h, m, s):
        # zde by mohla být kontrola vstupů
        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
```

```
class Time:
    # ... pokračování ...
    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()     # 1:31:12
t.add_seconds(107)
t.pretty_print()     # 1:32:59
```

Příklad 4 – matice

- 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)]
```

```
class Matrix:
    # ... pokračování ...
    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
```

```
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
```

Příklad 5 – jednosměrně zřetězený seznam

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class List:
    def __init__(self):
        self.first = None
    def insert_at_start(self, data):
        node = Node(data)
        node.next = self.first
        self.first = node
    def delete_first(self):
        if self.first != None:
            self.first = self.first.next
```

```
    def output(self):
        node = self.first
        while node != None:
            print(node.data)
            node = node.next

s = List()
s.insert_at_start("Hello")
s.insert_at_start("Ahoj")
s.output()
# Ahoj
# Hello
s.delete_first()
s.output()
# Hello
```

Příklad 6 – rodokmen

- chceme vytvořit rodokmen
- každý objekt bude reprezentovat člena rodiny
- každý člen rodiny má
  - jméno
  - seznam dětí
  - rodiče
- pro zjednodušení:
  - každý má v rodokmenu jen jednoho rodiče
  - typické pro rodokmeny s jedním prapředkem

```
class Person:
    def __init__(self, name):
        self.name = name
        self.parent = None
        self.children = []
    def add_child(self, child):
        self.children.append(child)
        child.parent = self
```

```
abe = Person("Abraham Simpson")
homer = Person("Homer Simpson")
abe.add_child(homer)
homer.add_child(Person("Bart Simpson"))
homer.add_child(Person("Lisa Simpson"))
homer.add_child(Person("Maggie Simpson"))
```

```
bart, lisa, maggie = homer.children

print(bart.name)          # Bart Simpson
print(bart.parent.name)   # Homer Simpson
```

- chceme funkci `siblings(a,b)`, která zjistí, jestli jsou `a`, `b` sourozenci

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

print(siblings(bart, lisa))    # True
print(siblings(homer, bart))   # False
print(siblings(maggie, abe))   # False
```

- chceme najít nejstaršího předka v rodokmenu: `oldest_ancestor`
- jak na to?

```
def oldest_ancestor(person):
    while person.parent != None:
        person = person.parent
    return person

print(oldest_ancestor(maggie).name)   # Abraham Simpson
```

(pokročilejší)

- chceme spočítat celkový počet všech (přímých i nepřímých) potomků zadané osoby
- jak na to? rekurzí!

```
def count_offspring(person):
    count = 0
    for child in person.children:
        count += 1 + count_offspring(child)
    return count

print(count_offspring(abe))  # 4
```

- iterativní řešení by bylo komplikovanější
  - dalo by se řešit pomocí zásobníku

(pokročilejší)

- chceme rodokmen vykreslit (např. textově) - jak? rekurzí!

```
def draw_family_tree(person, level = 0):
    print(("    " * level) + person.name)
    for child in person.children:
        draw_family_tree(child, level + 1)

draw_family_tree(abe)

# Abraham Simpson
#     Homer Simpson
#         Bart Simpson
#         Lisa Simpson
#         Maggie Simpson
```

(pokročilejší)

- vykreslení graficky
  - formát GraphViz
  - webová verze na `http://www.webgraphviz.com`

```
def add_transitions(person):
    for child in person.children:
        print('"' + person.name + '" -> "' + child.name + '"')
        add_transitions(child)

def create_family_tree_graph(person):
    print("digraph {")
    add_transitions(person)
    print("}")
```

Na co si dát pozor

Statické atributy

- definovány přímo ve třídě
- patří samotné třídě, ne objektům
- 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 (ve skutečnosti je to MyClass.x)
```

Na co si dát pozor

V čem je problém?

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

mirek = Student("Mirek Dušín")
mirek.add_hobby("pomáhání slabším")
mirek.add_hobby("světový mír")

bidlo = Student("Dlouhé Bidlo")
bidlo.add_hobby("alkohol")

print(mirek.hobbies)     # ???
```

Tečková notace

Jaký je rozdíl mezi běžnou funkcí a metodou objektu?

- v Pythonu nemůžu mít dvě funkce, které se jmenují stejně
- ale různé objekty mohou mít stejně pojmenované metody

```
class Dog:
    def make_sound(self):
        print("Whoof!")

class SmallFurryAnimalFromAlphaCentauri:
    def make_sound(self):
        print("Qwxrq!")

x = Dog()
x.make_sound()

x = SmallFurryAnimalFromAlphaCentauri()
x.make_sound()
```

Shrnutí

Záznamy/struktury

- používáme pro seskupení souvisejících dat
- v Pythonu přímo nejsou

Objekty

- složitější než záznamy
- mají metody
- v tomto předmětu je používáme jen jako záznamy
- více se o nich dozvíte v jiných předmětech