Objekty, strukturovaná data, styl

IB113
Radek Pelánek

2021

Dnešní přednáška

- objekty a strukturovaná data – rychlý úvod
- ukázka na živo – objekty + opakování
- styl, rozbor odevzdaných příkladů z Umíme programovat

Objekty: Motivace

udržování dat pohromadě

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

Jak reprezentovat?

Pokus 1: 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

Pokus 2: Využití konstant

```
TITLE = 0
AUTHOR = 1

book = ["Godel, Escher, Bach: An Eternal Golden Braid",
        "Douglas R. Hofstadter",
        "978-0465026562",
        1999]
print(book[TITLE])
print(book[AUTHOR])
```

lepší jak předchozí, ale pořád škaredé

Pokus 3: Slovníky

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

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

toto není špatné, ale . . .

Motivace II

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

Motivace II

- vlastní datové typy: nejen data, ale i funkcionalita
- 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
  - kontrolovat 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, ...

Nezbytné využití objektů

```
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()
```

Proč `t.upper()` a ne `upper(t)`?
Bez povědomí o objektech to nedává smysl.

Složené datové typy

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

- 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 (Java, C++)

Úrovně porozumění objektům

1. pochopení základních konceptů, porozumění vestavěným typům
2. použití objektů jako záznamy/struktury (shlukování dat)
3. využití objektů s metodami
4. plné objektově orientované programování

v IB113 pouze bod 1.
body 2. a 3. lze zvládnout samostudiem
na bod 4. se určitě hodí kurz

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 *instance* 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

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
- textové obrázky

Příklad: knihy

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

Knihy

```
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, neverwhere])
books = load_library("library.csv")
for b in books: print(b.title)
```

vytvořený / načítaný soubor library.csv:

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

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")
ib113 = Course("IB113")
ib113.add_student(Student(555000, "Luke Skywalker"))
ib113.add_student(jimmy)
ib113.add_student(Student(555555, "Bart Simpson"))
ib113.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 add_seconds(self, sec):
        self.seconds += sec
        self.validate()

    def pretty_print(self):
        print("{}:{:02}:{:02}".format(self.hours,
                                      self.minutes,
                                      self.seconds))



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

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)]
```

Matice: 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
```

Matice: 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
```

Matice: poznámky

čistě ilustrativní příklad

reálnější verze by využívala:

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

Textové obrázky

- objekt reprezentující obrázek
- metody: přidej bod, přidej čtverec, zkombinuj s jiným obrázkem, ...
- „textové“ vykreslení

$\Rightarrow$ demo ukázka programování

Varování: Statické atributy

- definovány přímo ve třídě
- patří samotné třídě, ne objektům
- mají svůj smysl, ale při začátcích spíše zdroj chyb

```
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("drinking beer")

print(mirek.hobbies)
```

Objekty a globální proměnné

příklad: herní plán (piškorky, dáma, apd)

- přirozená reprezentace objektem, nahrazení globálních proměnných
- datové atributy: velikost plánu, stav hracího plánu, figurky, ...
- metody: vykresli, zanes tah, zkontroluj výhru, ...

Vestavěné typy jako objekty

```
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()
```

Dědičnost

jeden z pokročilejších prvků objektového programování

ilustrace myšlenky na příkladech:

- typický „školní“ příklad: geometrické objekty
- praktický příklad: výpis heterogenních statistik z výukových systémů

Styl psaní programů

Při programování jde nejen o korektnost a efektivitu, ale i čitelnost a čistotu kódu:

- snadnost vývoje, testování
- údržba kódu
- spolupráce s ostatními (sám se sebou po půl roce)

Styl psaní programů

obecná doporučení:

- nepoužívat copy&paste kód
- dekompozice na funkce, „funkce dělá jednu věc“
- rozumná jména proměnných, funkcí

specifická doporučení (závisí na programovacím jazyce, příp. společnosti) – důležitá hlavně konzistence:

- odsazování
- bílá místa (mezery v rámci řádku)
- styl psaní víceslovných názvů

PEP8 – doporučení pro Python

`https://www.python.org/dev/peps/pep-0008/`

- obecný „duch“ doporučení celkem univerzální
- částečně však specifické pro Python (pojmenování, bílá místa, ...)

dnes na ukázkách, trochu více detailů na poslední přednášce

Komentované ukázky kódů

- řešení z programátorských cvičení z `umimeprogramovat.cz`
- automatické hodnocení, hodnotí pouze korektnost
- ilustrované stylistické problémy ale časté i jinde

upozornění: následující ukázky jsou všechny nějakým způsobem problematické

Rámeček

Napište funkci frame(text, symbol), která vypíše text v rámečku tvořeném znakem symbol (předpokládejte, že symbol je řetězec tvořený právě jedním znakem).

```
def frame(text, symbol):
    print(symbol*(len(text)+2))
    print(symbol + text + symbol)
    print(symbol*(len(text)+2))
```

Rámeček

```
def frame(text, symbol):
    n=len(text)
    for i in range(n+2):
            print(symbol,end="")
    print()
    print(symbol+text+symbol)
    for i in range(n+2):
            print(symbol,end="")
    print()
```

Rámeček

```
def frame(text, symbol):
    lenght = len(text)
    new_l = lenght +2
    print("{}".format(symbol*new_l))
    for i in range(0,new_l):
        if i == 0:
            print(symbol, end = '')
        elif i == new_l -1:
            print(symbol, end = '')
            print()
        else:
            print(text[i-1], end = '')
    print("{}".format(symbol*new_l))
```

Součin nenulových

Napište funkci nonzero_product(numbers), která pro zadaný seznam čísel numbers vrátí součin všech nenulových čísel v seznamu.

```
def nonzero_product(numbers):
    soucet = 1
    for i in numbers:
        if i != 0:
            soucet *= i
    return soucet
```

Maximální sousedi

Napište funkci max_pair_sum(num_list), která pro zadaný seznam kladných čísel num_list vypočítá nejvyšší součet dvou po sobě jdoucích čísel.

```
def max_pair_sum(num_list):
    new_list=[]
    for i in range(len(num_list)):
        if i<len(num_list)-1:
            new_list.append(num_list[i]+num_list[i+1])
        else:
            new_list.append(num_list[i-1]+num_list[i])
    for number in new_list:
        return max(new_list)
```

```
def max_pair_sum(num_list):
    count=0
    max=0
    sum=0
    sudy=0
    for number in num_list:
        sum+=number
        count+=1
        if count%2==1:
            lichy=number
        if count%2==0:
            sudy=number
        if sum > max:
            max=sum
        if count%2==0:
            sum=sum-lichy
        if count%2==1:
            sum=sum-sudy
    return max
```

Maximální sousedi

```
def max_pair_sum(num_list):
    x=num_list[0]+num_list[1]
    y=num_list[1]+num_list[2]
    z=num_list[2]+num_list[3]
    u=num_list[3]+num_list[4]
    q=num_list[4]+num_list[5]
    p=num_list[5]+num_list[6]
    return(max(x,y,z,u,q,p))
```

Nejdelší slovo v seznamu

Napište funkci find_longest_word(words_list), která vrátí nejdelší slovo ze zadaného seznamu slov words_list. Pokud je nejdelších slov více, vrátí to z nich, které je v seznamu uvedeno jako první.

```
def find_longest_word(words_list):
    a = 0
    for i in words_list:
        if len(i) > a:
            a = len(i)
            b = i
    return b
```

Nejdelší slovo v seznamu

```
def find_longest_word(words_list):
    delka=[]
    for i in range(len(words_list)):
        delka.append(len(words_list[i]))
    for j in range(len(delka)):
        if delka[j]==max(delka):
            return words_list[j]
```

Nejdelší slovo v seznamu

```
def find_longest_word(words_list):
    longest=words_list[0]
    long=len(words_list[0])
    for i in range(len(words_list)):
        for j in words_list:
            if len(words_list[i])>long:
                long=len(words_list[i])
                longest=words_list[i]
            else:
                continue
    return longest
```

Nejdelší slovo v seznamu

```
def find_longest_word(words_list):
    counter = 0
    result = " "
    for word in words_list:
        if len(word) > counter:
            counter = len(word)
    for word in words_list:
        if len(word) == counter:
            result = word
            break
    return result
```

Nejdelší slovo v seznamu

```
def find_longest_word(words_list):
    length = sorted(words_list, key=len)
    if len(length) == 1:
        return(length[0])
    elif len(length[-2]) == len(length[-1]) and len(len
        return(length[-2])
    elif len(length[-3]) == len(length[-2]) == len(leng
        return(length[-3])
    else:
        return(length[-1])
```

Kontrolní otázky

- Uveďte příklady dat, pro jejichž reprezentaci se přirozeně hodí objektový zápis.
- Jaký je rozdíl mezi pojmy třída a objekt?
- Jaký je význam pojmů atribut, metoda, konstruktor?
- Co je to „metoda“ a jak ji zapisujeme?
- Jak přistupujeme k metodám a atributům objektu?
- Jak vytvoříme objekt?
- Co znamená `self` v zápisu metod?

Shrnutí

- záznamy (struktury) – používáme pro seskupení souvisejících dat, v Pythonu přímo nejsou
- objekty
  - složitější než záznamy, data + metody
  - vestavěné typy používáme jako objekty
- dobrý programátorský styl je důležitý

příště: obrázky