7. Objekty

BAB37ZPR – Základy programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

- Část 1 – Objekty
  - Metody
  - Vytvoření objektu
  - Přístup k atributům
  - P7.1 Body ve 2D prostoru
  - P7.2 Obdélník
  - P7.3 Seznam knih
  - P7.4 Studenti a kurzy
  - Základní techniky objektově orientovaného programování

Část I

Objekty

Záznam (Record)

Záznam (obecně)

- strukturovaný/složený datový typ
- obsahuje položky *(fields)* / prvky *(elements)*/ členy
- položek je (obvykle) pevný počet, mají každá svůj typ
- položky jsou identifikované jménem
- typ záznamu má své jméno

Příklady

- Datum = rok, měsíc, den
- Osoba = jméno, příjmení, datum narození
- Adresa = jméno ulice, číslo popisné, město, PSČ
- Bod = x, y
- Kruh = střed, poloměr

Záznam (Record)

Záznam (obecně)

- strukturovaný/složený datový typ
- obsahuje položky *(fields)* / prvky *(elements)*/ členy
- položek je (obvykle) pevný počet, mají každá svůj typ
- položky jsou identifikované jménem
- typ záznamu má své jméno

Příklady

- Datum = rok, měsíc, den
- Osoba = jméno, příjmení, datum narození
- Adresa = jméno ulice, číslo popisné, město, PSČ
- Bod = x, y
- Kruh = střed, poloměr

Abstrakce

- **funkční abstrakce** *(function abstraction)*
  - Klient ví, co funkce dělá. *(rozhraní, interface)*
  - Klient nemusí vědět, jak je funkce implementována.
- **datová abstrakce** *(data abstraction)*
  - Klient ví, co datový typ představuje, jak ho vytvořit a jaké operace podporuje.
  - Klient nemusí vědět, jaká je vnitřní struktura.
- Klient musí znát rozhraní *(interface)*.
- Implementace je skrytá, lze ji změnit.
- Implementace je rozdělena mezi klientský kód a knihovny (např. ve formě modulů).
- Znovupoužitelnost kódu, možnost nezávislého vývoje.
- Zjednodušení psaní klientského kódu.

Záznam v Pythonu

V Pythonu záznamy nejsou ... ale jsou zde (dynamické) **objekty**.

objekt = *záznam* + *metody*

Objekt

- Obsahuje **datové položky** / **atributy** / **proměnné**
- Může obsahovat **metody** – funkce pracující s datovými položkami
- Strukturu definuje **třída**, **objekty** jsou instance třídy.
- **Třída** ve své podstatě definuje nový, **složený datový typ**.
- Základ **objektově orientovaného programování – OOP)**.

OOP je významně komplexnější, než je prezentováno v tomto předmětu.

Slovníček pojmů

třída	obecný uživatelsky definový typ, charakterizován atributy
objekt	konktétní instance třídy
atribut	vlastnost konkrétního objektu
metoda	funkce, která je vázaná na danou třídu
konstruktor	inicializační metoda, vytváří objekt

Intuitivní ilustrace

- třída: Pes
- instance: Alík
- datové atributy: rasa, jméno, věk, poloha
- metody: štěkej, sedni, lehni, popoběhni

Slovníček pojmů

třída	obecný uživatelsky definový typ, charakterizován atributy
objekt	konktétní instance třídy
atribut	vlastnost konkrétního objektu
metoda	funkce, která je vázaná na danou třídu
konstruktor	inicializační metoda, vytváří objekt

Intuitivní ilustrace

- třída: Pes
- instance: Alík
- datové atributy: rasa, jméno, věk, poloha
- metody: štěkej, sedni, lehni, popoběhni

Objekty v Pythonu

- Definice třídy:

```
>>> class Osoba:
...     pass # prázdná třída
...
```

- Vytvoření objektu (instance):

```
>>> o = Osoba()
```

- Přidání a použití datových položek (atributů):

```
>>> o.jmeno = "Karel"
>>> o.prijmeni = "Novak"
>>> print(o.jmeno + " " + o.prijmeni)
Karel Novak
```

- V Pythonu lze objektům přidávat atributy *dynamicky*.

  Tyto atributy jsou ale pak platné pouze pro danou instanci.

- K atributům přistupujeme pomocí *tečkové notace*.

Objekty v Pythonu

- Definice třídy:

```
>>> class Osoba:
...     pass # prázdná třída
...
```

- Vytvoření objektu (instance):

```
>>> o = Osoba()
```

- Přidání a použití datových položek (atributů):

```
>>> o.jmeno = "Karel"
>>> o.prijmeni = "Novak"
>>> print(o.jmeno + " " + o.prijmeni)
Karel Novak
```

- V Pythonu lze objektům přidávat atributy *dynamicky*.

  Tyto atributy jsou ale pak platné pouze pro danou instanci.

- K atributům přistupujeme pomocí *tečkové notace*.

Objekty v Pythonu

- Definice třídy:

```
>>> class Osoba:
...     pass # prázdná třída
...
```

- Vytvoření objektu (instance):

```
>>> o = Osoba()
```

- Přidání a použití datových položek (atributů):

```
>>> o.jmeno = "Karel"
>>> o.prijmeni = "Novak"
>>> print(o.jmeno + " " + o.prijmeni)
Karel Novak
```

- V Pythonu lze objektům přidávat atributy *dynamicky*.

  Tyto atributy jsou ale pak platné pouze pro danou instanci.

- K atributům přistupujeme pomocí *tečkové notace*.

Objekty v Pythonu

- Definice třídy:

```
>>> class Osoba:
...     pass # prázdná třída
...
```

- Vytvoření objektu (instance):

```
>>> o = Osoba()
```

- Přidání a použití datových položek (atributů):

```
>>> o.jmeno = "Karel"
>>> o.prijmeni = "Novak"
>>> print(o.jmeno + " " + o.prijmeni)
Karel Novak
```

- V Pythonu lze objektům přidávat atributy *dynamicky.*

  Tyto atributy jsou ale pak platné pouze pro danou instanci.

- K atributům přistupujeme pomocí *tečkové notace.*

I. Objekty

Instanční metody

- Metody jsou obecně funkce definované uvnitř třídy.

```
>>> class Osoba:
...     def print(self):
...         print(self.jmeno + " " + self.prijmeni)
...
>>> o = Osoba()
>>> o.jmeno = "Karel"
>>> o.prijmeni = "Novak"
```

- **Instanční metody** pracují s konkrétním objektem.
- Mají přístup k atributům objektu pomocí prvního parametru.

self není klíčovým slovem, jen silnou konvencí.

- Voláme je na objekt tečkovou notací, první argument je implicitní.

```
>>> o.print()
Karel Novak
```

Instanční metody

- Metody jsou obecně funkce definované uvnitř třídy.

```
>>> class Osoba:
...     def print(self):
...         print(self.jmeno + " " + self.prijmeni)
...
>>> o = Osoba()
>>> o.jmeno = "Karel"
>>> o.prijmeni = "Novak"
```

- **Instanční metody** pracují s konkrétním objektem.
- Mají přístup k atributům objektu pomocí prvního parametru.

self není klíčovým slovem, jen silnou konvencí.

- Voláme je na objekt tečkovou notací, první argument je implicitní.

```
>>> o.print()
Karel Novak
```

Statické metody

- **Statická metoda** je obyčejná funkce, definovaná uvnitř třídy.

```
>>> class Auto:
...     def info(): print('Třída popisuje automobil')
...
>>> Auto.info()
Třída popisuje automobil
```

- Nemá implicitní parametr → nemá přístup k atributům instance.
- S použitím dekorátoru `@staticmethod` ji lze volat i v rámci instance

```
>>> class Auto:
...     @staticmethod
...     def info(): print('Třída popisuje automobil')
...
>>> a = Auto()
>>> a.info()
Třída popisuje automobil
```

Statické metody

- **Statická metoda** je obyčejná funkce, definovaná uvnitř třídy.

```
>>> class Auto:
...     def info(): print('Třída popisuje automobil')
...
>>> Auto.info()
Třída popisuje automobil
```

- Nemá implicitní parametr → nemá přístup k atributům instance.
- S použitím dekorátoru `@staticmethod` ji lze volat i v rámci instance

```
>>> class Auto:
...     @staticmethod
...     def info(): print('Třída popisuje automobil')
...
>>> a = Auto()
>>> a.info()
Třída popisuje automobil
```

Metody tříd

- **Metoda třídy** definuje chování celé třídy, nikoli objektu.
- Metoda má přístup ke statickým atributům a metodám, nemá přístup k atributům instance.
- Je uvedena vestavěným dekorátorem `@classmethod`.

```
>>> class Auto:
...
...     text = 'Trida popisuje automobil'
...
>>>     @classmethod
  File "<stdin>", line 1
        @classmethod
    ^
IndentationError: unexpected indent
>>>     def info(obj):
  File "<stdin>", line 1
        def info(obj):
```

I. Objekty

Konstruktor

- Instance třídy mívají zpravidla stejné atributy, dynamické přidávání je nepraktické
- Pro zavedení atributů se používá inicializační metoda – **konstruktor**

```
>>> class Osoba:
...     def __init__(self, jmeno, prijmeni):
...         self.jmeno = jmeno
...         self.prijmeni = prijmeni
...     def print(self):
...         print(self.jmeno + " " + self.prijmeni)
...
```

- Konstruktor má speciální jméno `__init__`.
- Je volán implicitně (automaticky) při vytvoření objektu.

```
>>> o = Osoba("Karel", "Novak")
>>> o.print()
Karel Novak
```

Konstruktor

- Instance třídy mívají zpravidla stejné atributy, dynamické přidávání je nepraktické
- Pro zavedení atributů se používá inicializační metoda – **konstruktor**

```
>>> class Osoba:
...     def __init__(self, jmeno, prijmeni):
...         self.jmeno = jmeno
...         self.prijmeni = prijmeni
...     def print(self):
...         print(self.jmeno + " " + self.prijmeni)
...
```

- Konstruktor má speciální jméno `__init__`.
- Je volán implicitně (automaticky) při vytvoření objektu.

```
>>> o = Osoba("Karel", "Novak")
>>> o.print()
Karel Novak
```

I. Objekty

Přístup k atributům

Přímý

- přistupujeme přímo k atributu, můžeme ho přímo měnit
- `person.name`

Nepřímý

- pomocí metod (getters and setters)
- `person.get_name()`, `person.set_name()`

Který zvolit?

- závisí na použití
- objekt jen pro udržení dat – přímý přístup OK
- skrytí implementace (zapouzdření) – metody

Přístup k atributům

Přímý

- přistupujeme přímo k atributu, můžeme ho přímo měnit
- `person.name`

Nepřímý

- pomocí metod (getters and setters)
- `person.get_name()`, `person.set_name()`

Který zvolit?

- závisí na použití
- objekt jen pro udržení dat – přímý přístup OK
- skrytí implementace (zapouzdření) – metody

Veřejné a privátní atributy a metody

- V Pythonu chybí systematická ochrana objektů, vše je implicitně veřejné
- Má-li být položka privátní, začíná její název dvěma podtržítky

```
>>> class A:
...     def __fun(self):
...         print("Private method")
...     def info(self):
...         self.__fun()
...
>>> obj = A()
>>> obj.info()
Private method
>>> obj.__fun()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute '__fun'
```

I. Objekty

- definice třídy

```
>>> class Point:
...     def __init__(self, x = 0, y = 0):
...         self.x = x; self.y = y
...
```

- inicializace proměnných

```
>>> r = Point(10, 20)
>>> s = Point(13, 24)
>>> print("r.x =", r.x)
r.x = 10
```

- změna atributu

```
>>> s.y = 20
>>> print("s.y =", s.y)
s.y = 20
```

P7.1 Body ve 2D prostoru

- definice třídy

```
>>> class Point:
...     def __init__(self, x = 0, y = 0):
...         self.x = x; self.y = y
...
```

- inicializace proměnných

```
>>> r = Point(10, 20)
>>> s = Point(13, 24)
>>> print("r.x =", r.x)
r.x = 10
```

- změna atributu

```
>>> s.y = 20
>>> print("s.y =", s.y)
s.y = 20
```

- definice třídy

```
>>> class Point:
...     def __init__(self, x = 0, y = 0):
...         self.x = x; self.y = y
...
```

- inicializace proměnných

```
>>> r = Point(10, 20)
>>> s = Point(13, 24)
>>> print("r.x =", r.x)
r.x = 10
```

- změna atributu

```
>>> s.y = 20
>>> print("s.y =", s.y)
s.y = 20
```

- Funkce s argumenty datového typu `Point`:

```
>>> import math

>>> def distance(r,s):
...     """ Vypocita vzdalenost dvou bodu 'r','s' typu Point """
...     return math.sqrt((r.x-s.x)**2 + (r.y-s .y)**2)
...
>>> r = Point(10, 20)
>>> s = Point(13, 24)
>>> print(distance(r, s))
5.0
```

I. Objekty

P7.2 Obdélník

```
>>> class Rectangle:
...     def __init__(self, pozice, w, h):
...         self.corner = pozice
...         self.width = w
...         self.height = h
...
>>> bod = Rectangle(Point(), 100, 200)
>>> bod.width, bod.height
(100, 200)
>>> bod.corner.x, bod.corner.y
(0, 0)
```

Kopírování objektů 1/2

- Při kopírování třídy nebo její instance použijeme vždy metodu `copy` nebo `deepcopy` z modulu copy .
- Pro objekty tříd, které neobsahují vnořené objekty, postačí tak zvaná mělká kopie (shallow copy), která vytvoří nezávislou kopii nevnořených objektů.

```
>>> import copy
>>>
>>> p1 = Point(3,4)
>>> p2 = copy.copy(p1)
>>> # mají stejná ID ?
>>> p1 is p2
False
>>> # mají stejné souřadnice?
>>> p1.x == p2.x and p1.y == p2.y
True
```

Kopírování objektů 2/2

- Příkladem vnořeného objektu je atribut `corner` třídy `Rectangle`

```
>>> b1 = Rectangle(Point(), 100, 200)
>>> b2 = copy.copy(b1)
>>> b1 is b2
False
```

- Při provedení mělké kopie je atribut `corner` společný

```
>>> b1.corner is b2.corner
True
```

- Nezávislá hluboká kopie (deep copy) i vnořených objektů

```
>>> b3 = copy.deepcopy(b1)
>>> b1.corner is b3.corner
False
```

Kopírování objektů 2/2

- Příkladem vnořeného objektu je atribut `corner` třídy `Rectangle`

```
>>> b1 = Rectangle(Point(), 100, 200)
>>> b2 = copy.copy(b1)
>>> b1 is b2
False
```

- Při provedení mělké kopie je atribut `corner` společný

```
>>> b1.corner is b2.corner
True
```

- Nezávislá hluboká kopie (deep copy) i vnořených objektů

```
>>> b3 = copy.deepcopy(b1)
>>> b1.corner is b3.corner
False
```

Kopírování objektů 2/2

- Příkladem vnořeného objektu je atribut `corner` třídy `Rectangle`

```
>>> b1 = Rectangle(Point(), 100, 200)
>>> b2 = copy.copy(b1)
>>> b1 is b2
False
```

- Při provedení mělké kopie je atribut `corner` společný

```
>>> b1.corner is b2.corner
True
```

- Nezávislá hluboká kopie (deep copy) i vnořených objektů

```
>>> b3 = copy.deepcopy(b1)
>>> b1.corner is b3.corner
False
```

I. Objekty

- práce se seznamem knih
- atributy knihy: název, autor, rok
- chceme seznam načítat/ukládat do souboru
- implementace třídy:

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

- vytvoření záznamů

```
>>> a = Book("Dva roky prazdnin", "Jules Verne", "1888")
>>> b = Book("Honzikova cesta", "Bohumil Riha", "1954")
```

P7.3 Seznam knih

- práce se seznamem knih
- atributy knihy: název, autor, rok
- chceme seznam načítat/ukládat do souboru
- implementace třídy:

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

- vytvoření záznamů

```
>>> a = Book("Dva roky prazdnin", "Jules Verne", "1888")
>>> b = Book("Honzikova cesta", "Bohumil Riha", "1954")
```

P7.3 Seznam knih

- práce se seznamem knih
- atributy knihy: název, autor, rok
- chceme seznam načítat/ukládat do souboru
- implementace třídy:

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

- vytvoření záznamů

```
>>> a = Book("Dva roky prazdnin", "Jules Verne", "1888")
>>> b = Book("Honzikova cesta", "Bohumil Riha", "1954")
```

- načítání ze souboru

```
>>> 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
...
```

- ukládání do souboru

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

- načítání ze souboru

```
>>> 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
...
```

- ukládání do souboru

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

- použití

```
>>> save_library("library.csv", [a, b])
>>> books = load_library("library.csv")
>>> for b in books:
...     print(b.title)
...
Jules Verne
Bohumil Riha
```

Pokud ještě neznáte CSV

- CSV = Comma-separated values
- formát pro reprezentaci tabulkových dat
- jednoduchý textový formát, položky odděleny čárkami (nebo podobnými znaky)

Speciální metody a atributy 1/2

- Speciální (také magické) jsou všechny předdefinované metody a atributy tříd, jejichž název je ohraničen dvěma podtržítky
- Výpis předdefinovaných metod a atributů pro určitý objekt získáme příkazem dir()
- Speciální metody jsou interně evokované při explicitním volání příslušných operátorů a vestavěných funkcí.

```
>>> "ariel".__eq__("alias")
False
>>> (2).__mul__(4)
8
```

- Pokud definujeme nový datový typ (tj. třídu), je pro různé účely třeba definovat speciální funkce – např. při používání vestavěných třídicích metod je třeba mít definovaný operátor pro poeovnání dvou prvků

Speciální metody a atributy 2/2

```
>>> class Point:
...     def __init__(self, x=0, y=0):     # speciální metoda (instanční)
...         self.x = x
...         self.y = y
...     def __add__(self, other):         # speciální metoda (instanční)
...         return self.x + other.x, self.y + other.y
...
>>> p1 = Point(1, 2)
>>> p2 = Point(3, 4)
>>> p1 + p2                                  # interně  p1.__add__(p2)
(4, 6)
```

I. Objekty

P7.4 Studenti a kurzy

- reprezentace studentů a kurzů
- kurz je seznamem zapsaných studentů
- implementace třídy `Student`

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

P7.4 Studenti a kurzy

- reprezentace studentů a kurzů
- kurz je seznamem zapsaných studentů
- implementace třídy `Student`

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

```
>>> class Course:
...     def __init__(self, code):
...         ''' konstruktor '''
...         self.code = code
...         self.students = []
...     def add_student(self, student):
...         ''' pridani studenta '''
...         self.students.append(student)
...     def print_students(self):
...         ''' tisk informace o studentovi '''
...         i = 1
...         for s in self.students:
...             print(str(i) + ".", str(s.id), s.name, sep="\t")
...             i += 1
...
```

- Použití

```
>>> jimmy = Student(555007, "James Bond")
>>> ib111 = Course("ZPR")
>>> 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
```

- Co kdybychom chtěli řadit?
  - `sorted(self.students)` nefunguje – není definováno
  - definovat `__lt__(self, other)`
  - `sorted(self.students, key=lambda s: s.name)`

- Použití

```
>>> jimmy = Student(555007, "James Bond")
>>> ib111 = Course("ZPR")
>>> 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
```

- Co kdybychom chtěli řadit?
  - `sorted(self.students)` nefunguje – není definováno
  - definovat `__lt__(self, other)`
  - `sorted(self.students, key=lambda s: s.name)`

I. Objekty

Metody

Vytvoření objektu

Přístup k atributům

P7.1 Body ve 2D prostoru

P7.2 Obdélník

P7.3 Seznam knih

P7.4 Studenti a kurzy

Základní techniky objektově orientovaného programování

Dědění

- Dědění (inheritance) se používá při odvození nové třídy z třídy stávající.
- Odvozená třída přebírá atributy a metody (bázové) třídy výchozí.

```
>>> class Rodič:
...     pass                    # super class, rodič, předek
...
>>> class Potomek(Rodič):
...     pass                    # sub class, sub typ, potomek
...
```

Možnosti dědění

- **simple inheritance** – dědění po jediném předkovi, viz příklad
- **multiple inheritance** – prosté dědění po různých předcích
  - `class Potomek(Rodic1, Rodic2)`
- **multilevel inheritance** - zanořené dědění po více předcích
  - `class Potomek(Rodic), class Vnuk(Potomek)`

Dědění – příklad

```
>>> class Savci:
...     at = "atribut třídy"
...     def __init__(self, druh):
...         self.ai = "atribut třídy"
...         print(druh, 'je teplokrevný savec.')
...
>>> class Pes(Savci):
...     def __init__(self):
...         super().__init__('Pes') # Varianta: Savci.__init__(self, 'Pes')
...         print('Pes je přítel člověka.')
...
>>> pajda = Pes()
Pes je teplokrevný savec.
Pes je přítel člověka.
>>> pajda.at, pajda.ai
('atribut třídy', 'atribut třídy')
```

Přepsání metod a atributů při dědění

```
>>> class A:
...     pes = "Dingo"
...     def zobraz(self):
...         print ('Jsem třída A.')
...
>>> class B(A):
...     pes = "Voříšek"
...     def zobraz(self):
...         print ('Jsem třída B.')
...
>>> obj = B()
>>> obj.pes
'Voříšek'
>>> obj.zobraz()
Jsem třída B.
```

Propojení tříd – kompozice 1/2

- Propojení se uskuteční prostřednictvím instance třídy uvnitř jiné třídy

```
>>> class Raketa:
...     def __init__(self, name, destinace):
...         self.name = name
...         self.destinace = destinace
...     def launch(self):
...         return "%s dosedl na %s" % (self.name, self.destinace)
...
>>> class Lunochod():
...     def __init__(self, name, destinace, maker):
...         self.raketa = Raketa(name, destinace)   # deklarace instance
...         self.maker = maker
...     def report(self):
...         return "%s byl vyspuštěn %s" % (self.raketa.name, self.maker)
...
```

Propojení tříd – kompozice 1/2

```
>>> z = Lunochod("Lunochod", "Měsíc", "ESA")
>>> z.raketa.launch()
'Lunochod dosedl na Měsíc'
>>> z.report()
'Lunochod byl vyspuštěn ESA'
```