`Programování` 1 pro matematiky

10. cvičení, 9-12-2021

tags: `Programovani 1 2021`, `čtvrtek`

Obsah:

0. Farní oznamy
1. Opakování: Slovníky a množiny
2. Třídy

Farní oznamy

1. **Materiály k přednáškám** najdete v GitHub repozitáři https://github.com/PKvasnick/Programovani-1. Najdete tam také kód ke cvičením a pdf soubory textů cvičením.
2. **Domácí úkoly**
    - Dnes dostanete nové úkoly, budou všechny označeny jako bonusové, abychom nezvyšovali laťku.
    - 2 kategorie:
        - lehčí pro ty, kteří potřebují nabrat před koncem semestru nějaké body
        - těžší pro ty, kteří chtějí dělat něco zajímavějšího.
3. **Opakování** velice rychle n-tice, slovníky a množiny

Kde se nacházíme

Dnes začneme mluvit o třídách v Pythonu.

Opakování: n-tice, množiny, slovníky

n-tice

n-tice je neměnná (immutable) struktura, která obsahuje několik objektů, které logicky patří k sobě, například souřadnice x, y bodu v rovině, den, měsíc a rok v datumu a pod.

```
>>> a = 1
>>> b = 2
>>> t = (a,b) # sbalení
>>> t
(1, 2)
>>> a = 2
>>> t
(1, 2)
>>> t[0]
1
>>> t[1]
2
```

```
>>> t[0] = 3
Traceback (most recent call last):
  File "<pyshell#292>", line 1, in <module>
    t[0] = 3
TypeError: 'tuple' object does not support item assignment
>>> x, y = t # rozbalení
>>> x
1
>>> y
2
```

Funkce `enumerate` a `zip`

Abychom se vyhnuli iterování přes index a vyhledávání v seznamu/množině/slovníku v každém cyklu:

`enumerate(seznam) dává (0, seznam[0]), (1, seznam[1])...`

```
>>> mesta = ["Praha", "Brno", "Ostrava"]
>>> for i in range(len(mesta)):
    print(i, mesta[i])


0 Praha
1 Brno
2 Ostrava
>>> for i, mesto in enumerate(mesta):
    print(i, mesto)


0 Praha
1 Brno
2 Ostrava
>>> for u in enumerate(mesta):
    print(u)


(0, 'Praha')
(1, 'Brno')
(2, 'Ostrava')
```

`zip(seznam1, seznam2) dává (seznam1[0], seznam2[0]), (seznam1[1], seznam2[1])...`

```
>>> text = """
Praha -2 0
Brno 0 -1
Ostrava 1 1
"""
>>> mesta = []
>>> x = []
>>> y = []
>>> for radek in text.split("\n"):
    if len(radek) == 0:
        continue
    veci = radek.split()
    mesta.append(veci[0])
```

```
    x.append(float(veci[1]))
    y.append(float(veci[2]))


>>> mesta, x, y
(['Praha', 'Brno', 'Ostrava'], [-2.0, 0.0, 1.0], [0.0, -1.0, 1.0])

# Standardní způsob:
>>> for i in range(len(mesta)):
    print(mesta[i], x[i], y[i])


Praha -2.0 0.0
Brno 0.0 -1.0
Ostrava 1.0 1.0

# S využitím funkce zip:
>>> for mesto, x, y in zip(mesta, x, y):
    print(mesto, x, y)


Praha -2.0 0.0
Brno 0.0 -1.0
Ostrava 1.0 1.0
>>>
```

Množiny

Množiny jsou vysoce optimalizované kontejnery s rychlým vyhledáváním:

```
>>> zvířata = {"kočka", "pes", "lev", "pes", "lev", "tygr"}
>>> zvířata
{'pes', 'tygr', 'lev', 'kočka'}
>>> "tygr" in zvířata # O(1)
True
>>> set(["a", "b", "c"])
{'b', 'c', 'a'}
set("abrakadabra")
{'d', 'b', 'a', 'r', 'k'}
>>> set() # prázdná množina
set()
>>> {} # není prázdná množina!
{}
>>> type({})
<class 'dict'>
```

Množiny využívají stromové struktury a algoritmy pro rychlé vyhledávání a modifikaci. Vytváření množin a operace:

```
set("abrakadabra")
{'d', 'b', 'a', 'r', 'k'}
>>> a=set("abrakadabra")
>>> b=set("popokatepetl")
>>> "".join(sorted(a))
'abdkr'
>>> a & b # průnik, také a.intersection(b)
```

```
{'k', 'a'}
>>> a | b # sjednoceni, také a.union(b)
{'d', 'b', 'o', 'l', 'p', 'e', 'a', 'r', 't', 'k'}
>>> a - b # rozdíl, také a.difference(b)
{'d', 'b', 'r'}
>>> a.remove("r")
>>> a
{'d', 'b', 'a', 'k'}
>>> b.add("b")
>>> b
{'o', 'b', 'l', 'p', 'e', 'a', 't', 'k'}
>>> a == b
False
```

Podrobněji třeba tady

Slovníky

```
>>> teploty = { "Praha": 17, "Dill´ı": 42,
"Longyearbyen": -46 }
>>> teploty
{'Praha': 17, 'Dill´ı': 42, 'Longyearbyen': -46}
>>> teploty["Praha"]
17
>>> teploty["Debrecen"]
Traceback (most recent call last):
  File "<pyshell#387>", line 1, in <module>
    teploty["Debrecen"]
KeyError: 'Debrecen'
>>> teploty["Debrecen"] = 28
>>>
>>> del teploty["Debrecen"]
>>> "Debrecen" in teploty
False
>>> teploty["Miskolc"]
Traceback (most recent call last):
  File "<pyshell#394>", line 1, in <module>
    teploty["Miskolc"]
KeyError: 'Miskolc'
>>> teploty.get("Miskolc")
None
>>> teploty.get("Miskolc", 20)
20

# Iterujeme ve slovníku:
>>> for k in teploty.keys():
    print(k)

Praha
Dill´ı
Longyearbyen
>>> for v in teploty.values():
    print(v)

17
42
-46
```

```
>>> for k, v in teploty.items():
    print(k, v)

Praha 17
Dill´ı 42
Longyearbyen -46
>>>
```

Podrobněji třeba tady

Comprehensions pro seznany, množiny a slovníky.

```
>>> [i % 7 for i in range(50)]
[0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0]
>>> {i % 7 for i in range(50)}
{0, 1, 2, 3, 4, 5, 6}
>>> {i : i % 7 for i in range(50)}
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 0, 8: 1, 9: 2, 10: 3, 11: 4, 12: 5, 13: 6, 14: 0, 15: 1, 16: 2, 17: 3, 18: 4, 19: 5, 20: 6, 21: 0, 22: 1, 23: 2, 24: 3, 25: 4, 26: 5, 27: 6, 28: 0, 29: 1, 30: 2, 31: 3, 32: 4, 33: 5, 34: 6, 35: 0, 36: 1, 37: 2, 38: 3, 39: 4, 40: 5, 41: 6, 42: 0, 43: 1, 44: 2, 45: 3, 46: 4, 47: 5, 48: 6, 49: 0}
>>>
```

`defaultdict` - slovník s defaultní hodnotou *pro počítání*

```
>>> from collections import defaultdict
>>> pocet = defaultdict(int)
>>> pocet['abc']
0
>>> from collections import defaultdict
>>> pocet = defaultdict(int)
>>> pocet["abc"]
0
# počítáme slova
>>> for w in "quick brown fox jumps over lazy dog".split():
    pocet[w] += 1
>>> pocet
defaultdict(<class 'int'>, {'abc': 0, 'quick': 1, 'brown': 1, 'fox': 1, 'jumps': 1, 'over': 1, 'lazy': 1, 'dog': 1})
>>> list(pocet.items())
[('abc', 0), ('quick', 1), ('brown', 1), ('fox', 1), ('jumps', 1), ('over', 1), ('lazy', 1), ('dog', 1)]

# počítáme délky slov
>>> podle_delek = defaultdict(list)
>>> for w in "quick brown fox jumps over lazy dog".split():
    podle_delek[len(w)].append(w)

>>> podle_delek
defaultdict(<class 'list'>, {5: ['quick', 'brown', 'jumps'], 3: ['fox', 'dog'], 4: ['over', 'lazy']})
>>>
```

Třídy

Třídy nám umožňují seskupit data a funkce, které na nich operují a zpřístupňují je, a zároveň "schovat" detaily implementace. Třída je datový typ, od kterého si vytváříme instance.

```
class Zvire():
    pass

>>> pes = Zvire()
>>> pes
<__main__.Zvire object at 0x000001A01A376460>
>>> kocka = Zvire()
>>> kocka
<__main__.Zvire object at 0x000001A01A391B80>
```

Vidíme, že máme dva různé objekty. Takovýto objekt by ale nebyl moc užitečný, pokud neumíme definovat nějaké vlastnosti objektu.

```
# Třídy

class Zvire():

    def __init__(self, jmeno, zvuk):
        self.jmeno = jmeno
        self.zvuk = zvuk

    def slysi_na(self, jmeno):
        return self.jmeno == jmeno

    def ozvi_se(self):
        print(f"{self.jmeno} říká: {self.zvuk}")

...
>>> pes = Zvire("Punťa", "Hafff!")
>>> pes
<__main__.Zvire object at 0x000001A01A391B80>
>>> pes.slysi_na("Miau")
False
>>> pes.ozvi_se()
Punťa říká: Hafff!
>>> kocka = Zvire("Mourek", "Miau!")
>>> kocka.ozvi_se()
Mourek říká: Miau!
```

`self` nás odkazuje na instanci třídy.

`__init__()` je metoda, která vytváří instanci ze vstupních dat - *konstruktor*.

Metod s dvojitými podtržítky existuje mnoho. Jsou to metody, které definují standardní aspekty objektů.

Vlastnosti a metody

```
>>> azor = Zvire("Azor", "Haf!")
>>> azor
<__main__.Zvire object at 0x00000214E4303D00>
>>> azor.jmeno
'Azor'
>>> azor.zvuk
'Haf!'
>>> azor.zvuk = "Haffff!"
>>> azor.slysi_na("azor")
False
>>> azor.ozvi_se()
Azor říká: Haffff!
```

Identita objektu

```
>>> jezevcik = Zvire("˘Spagetka", "haf")
>>> bernardyn = Zvire("Bernard", "HAF!!!")
>>> maxipes = bernardyn
>>> maxipes.jmeno = "F´ık"
>>> bernardyn.jmeno
'F´ık'
>>> type(jezevcik)
<class 'Zvire'>
>>> id(jezevcik), id(bernardyn), id(maxipes)
(737339253592, 737339253704, 737339253704)
>>> bernardyn is maxipes
True
>>> bernardyn is jezevcik
False
```

Znaková reprezentace objektu

`__str__()` je to, co používá funkce `print`

`__repr__()` je to, co vypíše Pythonská konzole jako identifikaci objektu.

```
class Zvire():

    def __init__(self, jmeno, zvuk):
        self.jmeno = jmeno
        self.zvuk = zvuk

    ...

    def __str__(self):
        return self.jmeno

    def __repr__(self):
        return f"Zvire({self.jmeno}, {self.zvuk})"

    ...
>>> pes = Zvire("Punta", "haf!")
>>> pes
Zvire(Punta, haf!)
>>> print(pes)
Punta
```

Protokoly pro operátory

```
class Zvire():

    def __init__(self, jmeno, zvuk):
        self.jmeno = jmeno
        self.zvuk = zvuk

...

    def __eq__(self, other):
        return self.jmeno == other.jmeno and \
            self.zvuk == other.zvuk

...
>>> pes = Zvire("Punta", "haf!")
>>> kocka = Zvire("Mourek", "Miau!")
>>> pes == kocka
False
```

Podobně lze předefinovat řadu dalších operátorů:

- Konverze na bool, str, int, float
- Indexování `objekt[i], len(i)`, čtení, zápis, mazání.
- Přístup k atributům `objekt.klíč`
- Volání jako funkce `objekt(x)`
- Iterátor pro `for x in objekt:`

Dokumentační řetězec

```
class Zvire():
    """Vytvoří zvíře s danými vlastnostmi"""

    def __init__(self, jmeno, zvuk):
        self.jmeno = jmeno
        self.zvuk = zvuk

...

>>> help(Zvire)
>>> lenochod = Zvire("lenochod", "Zzzz...")
>>> help(lenochod.slysi_na)

```

Dědičnost

```
class Kocka(Zvire):

    def __init__(self, jmeno, zvuk):
        Zvire.__init__(self, jmeno, zvuk)
        self._pocet_zivotu = 9 # interní

    def slysi_na(self, jmeno):
        # Copak ko˘cka sly˘s´ı na jm´eno?
```

```
        return False
...

>>> k = Kocka("Příšerka", "Mňauuu")
>>> k.slysi na("Příšerka") (speciální kočičí verze)
False
>>> k.ozvi se() (původní zvířecí metoda)
Příšerka říká: Mňauuu
```

Typy

```
>>> type(k) is Kocka
True
>>> type(k) is Zvire
False
>>> isinstance(k, Kocka)
True
>>> isinstance(k, Zvire)
True
>>> issubclass(Kocka, Zvire)
True
```

Prostory a rozsahy platnosti

Co dělá Python, když chce zjistit, kterou metodu třídy má volat?

Prostory jmen, **namespaces**:

- Zabudované funkce (print) `builtins`
- Globální jména - proměnné a funkce, definované mimo jakoukoli funkci nebo třídu `globals`
- Lokální jména definovaná při aktuálním volání uvnitř aktuální funkce `locals`
- Jména definovaná v aktuální třídě
- Jména definovaná v aktuálním objektu

Oblasti platnosti, **scopes**

Obyčejné jméno se hledá ve všech prostorech jmen, které jsou z daného kontextu vidět - lokální, globální, zabudované proměnné.

`objekt.jméno` se hledá

- mezi atributy objektu
- mezi atributy třídy
- mezi atributy nadřazených tříd