

# Programování 1 pro matematiky

## 10. cvičení, 8-12-2021

tags: Programování 1 2021, středa

### 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.

```
1 >>> a = 1
2 >>> b = 2
3 >>> t = (a,b) # sbalení
4 >>> t
5 (1, 2)
6 >>> a = 2
7 >>> t
8 (1, 2)
9 >>> t[0]
10 1
11 >>> t[1]
12 2
```

```

13 >>> t[0] = 3
14 Traceback (most recent call last):
15   File "<pyshe11#292>", line 1, in <module>
16     t[0] = 3
17 TypeError: 'tuple' object does not support item assignment
18 >>> x, y = t # rozbalení
19 >>> x
20 1
21 >>> y
22 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])...`

```

1 >>> mesta = ["Praha", "Brno", "Ostrava"]
2 >>> for i in range(len(mesta)):
3     print(i, mesta[i])
4
5
6 0 Praha
7 1 Brno
8 2 Ostrava
9 >>> for i, mesto in enumerate(mesta):
10     print(i, mesto)
11
12
13 0 Praha
14 1 Brno
15 2 Ostrava
16 >>> for u in enumerate(mesta):
17     print(u)
18
19
20 (0, 'Praha')
21 (1, 'Brno')
22 (2, 'Ostrava')

```

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

```

1 >>> text = """
2 Praha -2 0
3 Brno 0 -1
4 Ostrava 1 1
5 """
6 >>> mesta = []
7 >>> x = []
8 >>> y = []
9 >>> for radek in text.split("\n"):
10     if len(radek) == 0:
11         continue
12     veci = radek.split()
13     mesta.append(veci[0])

```

```

14     x.append(float(veci[1]))
15     y.append(float(veci[2]))
16
17
18 >>> mesta, x, y
19 ([ 'Praha', 'Brno', 'Ostrava'], [-2.0, 0.0, 1.0], [0.0, -1.0, 1.0])
20
21 # Standardní způsob:
22 >>> for i in range(len(mesta)):
23     print(mesta[i], x[i], y[i])
24
25
26 Praha -2.0 0.0
27 Brno 0.0 -1.0
28 Ostrava 1.0 1.0
29
30 # S využitím funkce zip:
31 >>> for mesto, x, y in zip(mesta, x, y):
32     print(mesto, x, y)
33
34
35 Praha -2.0 0.0
36 Brno 0.0 -1.0
37 Ostrava 1.0 1.0
38 >>>

```

## Množiny

Množiny jsou vysoce optimalizované kontejnery s rychlým vyhledáváním:

```

1 >>> zvířata = {"kočka", "pes", "lev", "pes", "lev", "tygr"}
2 >>> zvířata
3 {'pes', 'tygr', 'lev', 'kočka'}
4 >>> "tygr" in zvířata # O(1)
5 True
6 >>> set(["a", "b", "c"])
7 {'b', 'c', 'a'}
8 set("abrakadabra")
9 {'d', 'b', 'a', 'r', 'k'}
10 >>> set() # prázdná množina
11 set()
12 >>> {} # není prázdná množina!
13 {}
14 >>> type({})
15 <class 'dict'>

```

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

```

1 set("abrakadabra")
2 {'d', 'b', 'a', 'r', 'k'}
3 >>> a=set("abrakadabra")
4 >>> b=set("popokatepetl")
5 >>> "".join(sorted(a))
6 'abdkr'
7 >>> a & b # průnik, také a.intersection(b)

```

```

8  {'k', 'a'}
9  >>> a | b # sjednocení, také a.union(b)
10 {'d', 'b', 'o', 'l', 'p', 'e', 'a', 'r', 't', 'k'}
11 >>> a - b # rozdíl, také a.difference(b)
12 {'d', 'b', 'r'}
13 >>> a.remove("r")
14 >>> a
15 {'d', 'b', 'a', 'k'}
16 >>> b.add("b")
17 >>> b
18 {'o', 'b', 'l', 'p', 'e', 'a', 't', 'k'}
19 >>> a == b
20 False

```

Podrobněji třeba [tady](#).

## Slovníky

```

1  >>> teploty = { "Praha": 17, "Dillí": 42,
2  "Longyearbyen": -46 }
3  >>> teploty
4  {'Praha': 17, 'Dillí': 42, 'Longyearbyen': -46}
5  >>> teploty["Praha"]
6  17
7  >>> teploty["Debrecen"]
8  Traceback (most recent call last):
9    File "<pyshe11#387>", line 1, in <module>
10     teploty["Debrecen"]
11  KeyError: 'Debrecen'
12 >>> teploty["Debrecen"] = 28
13 >>>
14 >>> del teploty["Debrecen"]
15 >>> "Debrecen" in teploty
16 False
17 >>> teploty["Miskolc"]
18 Traceback (most recent call last):
19   File "<pyshe11#394>", line 1, in <module>
20     teploty["Miskolc"]
21  KeyError: 'Miskolc'
22 >>> teploty.get("Miskolc")
23 None
24 >>> teploty.get("Miskolc", 20)
25 20
26
27 # Iterujeme ve slovníku:
28 >>> for k in teploty.keys():
29     print(k)
30
31 Praha
32 Dillí
33 Longyearbyen
34 >>> for v in teploty.values():
35     print(v)
36
37 17
38 42
39 -46

```

```

40 >>> for k, v in teploty.items():
41     print(k, v)
42
43 Praha 17
44 Dillí 42
45 Longyearbyen -46
46 >>>

```

Podrobněji třeba [tady](#).

**Comprehensions** pro seznamy, množiny a slovníky.

```

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

```

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

```

1 >>> from collections import defaultdict
2 >>> pocet = defaultdict(int)
3 >>> pocet['abc']
4 0
5 >>> from collections import defaultdict
6 >>> pocet = defaultdict(int)
7 >>> pocet["abc"]
8 0
9 # počítáme slova
10 >>> for w in "quick brown fox jumps over lazy dog".split():
11     pocet[w] += 1
12 >>> pocet
13 defaultdict(<class 'int'>, {'abc': 0, 'quick': 1, 'brown': 1, 'fox': 1,
14 'jumps': 1, 'over': 1, 'lazy': 1, 'dog': 1})
15 >>> list(pocet.items())
16 [('abc', 0), ('quick', 1), ('brown', 1), ('fox', 1), ('jumps', 1), ('over',
17 1), ('lazy', 1), ('dog', 1)]
18 # počítáme délky slov
19 >>> podle_delek = defaultdict(list)
20 >>> for w in "quick brown fox jumps over lazy dog".split():
21     podle_delek[len(w)].append(w)
22 >>> podle_delek
23 defaultdict(<class 'list'>, {5: ['quick', 'brown', 'jumps'], 3: ['fox',
24 'dog'], 4: ['over', 'lazy']})
25 >>>

```

# 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.

```
1 class Zvire():
2     pass
3
4 >>> pes = Zvire()
5 >>> pes
6 <__main__.Zvire object at 0x000001A01A376460>
7 >>> kocka = Zvire()
8 >>> kocka
9 <__main__.Zvire object at 0x000001A01A391B80>
```

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

```
1 # Třídy
2
3 class Zvire():
4
5     def __init__(self, jmeno, zvuk):
6         self.jmeno = jmeno
7         self.zvuk = zvuk
8
9     def slysi_na(self, jmeno):
10        return self.jmeno == jmeno
11
12    def ozvi_se(self):
13        print(f"{self.jmeno} říká: {self.zvuk}")
14
15    ...
16 >>> pes = Zvire("Puntča", "Hafff!")
17 >>> pes
18 <__main__.Zvire object at 0x000001A01A391B80>
19 >>> pes.slysi_na("Miau")
20 False
21 >>> pes.ozvi_se()
22 Puntča říká: Hafff!
23 >>> kocka = Zvire("Mourek", "Miau!")
24 >>> kocka.ozvi_se()
25 Mourek říká: Miau!
```

`self` nás odkazuje na instanci třídy.

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

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

## Vlastnosti a metody

```

1  >>> azor = Zvire("Azor", "Haf!")
2  >>> azor
3  <__main__.Zvire object at 0x00000214E4303D00>
4  >>> azor.jmeno
5  'Azor'
6  >>> azor.zvuk
7  'Haf!'
8  >>> azor.zvuk = "Haffff!"
9  >>> azor.slysi_na("azor")
10 False
11 >>> azor.ozvi_se()
12 Azor říká: Haffff!

```

## Identita objektu

```

1  >>> jezevcik = Zvire("Špagetka", "haf")
2  >>> bernardyn = Zvire("Bernard", "HAF!!!")
3  >>> maxipes = bernardyn
4  >>> maxipes.jmeno = "Fík"
5  >>> bernardyn.jmeno
6  'Fík'
7  >>> type(jezevcik)
8  <class 'Zvire'>
9  >>> id(jezevcik), id(bernardyn), id(maxipes)
10 (737339253592, 737339253704, 737339253704)
11 >>> bernardyn is maxipes
12 True
13 >>> bernardyn is jezevcik
14 False

```

## Znaková reprezentace objektu

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

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

```

1  class Zvire():
2
3      def __init__(self, jmeno, zvuk):
4          self.jmeno = jmeno
5          self.zvuk = zvuk
6
7      ...
8
9      def __str__(self):
10         return self.jmeno
11
12     def __repr__(self):
13         return f"Zvire({self.jmeno}, {self.zvuk})"
14
15     ...
16 >>> pes = Zvire("Punta", "haf!")
17 >>> pes
18 Zvire(Punta, haf!)
19 >>> print(pes)
20 Punta

```

## Protokoly pro operátory

```
1 class Zvire():
2
3     def __init__(self, jmeno, zvuk):
4         self.jmeno = jmeno
5         self.zvuk = zvuk
6
7     ...
8
9     def __eq__(self, other):
10         return self.jmeno == other.jmeno and \
11             self.zvuk == other.zvuk
12
13     ...
14 >>> pes = Zvire("Punta", "haf!")
15 >>> kocka = Zvire("Mourek", "Miau!")
16 >>> pes == kocka
17 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

```
1 class Zvire():
2     """vytvoří zvíře s danými vlastnostmi"""
3
4     def __init__(self, jmeno, zvuk):
5         self.jmeno = jmeno
6         self.zvuk = zvuk
7
8     ...
9
10 >>> help(Zvire)
11 >>> lenochod = Zvire("lenochod", "zzzz...")
12 >>> help(lenochod.slysi_na)
13
```

## Dědičnost

```
1 class Kocka(Zvire):
2
3     def __init__(self, jmeno, zvuk):
4         Zvire.__init__(self, jmeno, zvuk)
5         self._pocet_zivotu = 9 # interní
6
7     def slysi_na(self, jmeno):
8         # Copak kočka slyší na jméno?
```



```

9         return False
10     ...
11
12 >>> k = Kocka("Příšerka", "Mňauuu")
13 >>> k.slysi na("Příšerka") (speciální kočičí verze)
14 False
15 >>> k.ozvi se() (původní zvířecí metoda)
16 Příšerka říká: Mňauuu

```

## Typy

```

1 >>> type(k) is Kocka
2 True
3 >>> type(k) is Zvire
4 False
5 >>> isinstance(k, Kocka)
6 True
7 >>> isinstance(k, Zvire)
8 True
9 >>> issubclass(Kocka, Zvire)
10 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