

Programování 1 pro matematiky

11. cvičení, 15-12-2021

tags: Programování 1 2021, středa

Obsah:

0. Farní oznamy
1. Domácí úkoly
2. Opakování: Třídy
3. Soubory a výjimky

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**
 - O víkendu jste dostali novou trojici domácích úkolů.
 - Další trojice bude následovat
 - Dáme nějakou nápovědu
3. **Opakování** velice rychle třídy a datové objekty

Kde se nacházíme

Dnešní hlavní téma bude čtení a zápis do souborů a obsluha výjimek.

Opakování: 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  # 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", "Hauff!")
```

```

17 >>> pes
18 <__main__.Zvire object at 0x000001A01A391B80>
19 >>> pes.slysi_na("Miau")
20 False
21 >>> pes.ozvi_se()
22 Puntča říká: Haffff!
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
13 >>> k = Kocka("Příšerka", "Mňauuu")
14 >>> k.slysi_na("Příšerka") (speciální kočičí verze)
15 False
16 >>> k.ozvi se() (původní zvířecí metoda)
17 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 >>> isinstance(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

Soubory: čtení a zápis

Souborem myslíme nějakou skupinu bajtů, uloženou pod svým názvem v souborovém systému.

Budeme se zabývat **textovými soubory**, v nichž bajty reprezentují znaky v nějakém kódování.

- *ASCII* ("anglická abeceda" o 95 znacích)
- *iso-8859-2* (národní znaky východoevropských jazyků)
- *cp1250* (něco podobného specifického pro Windows)
- *UTF-8* (všechny znaky, pokrývá všechny jazyky světa)

Kódování je všude přítomné, nevyhnete se problémům s explicitním uvedením kódování nebo s převodem. Neexistuje nic takového jako defaultní kódování textového souboru, i když například pro kód v Pythonu je defaultním kódováním UTF-8.

Python má rozsáhlou podporu kódování a většinu problémů jde jednoduše řešit.

Python načte textový soubor jako kolekci řádků.

```
1 f = open("soubor.txt", "w")
2 f.write("Hej, mistře!\n")
3 f.close()
```

Protože komunikujeme se systémovými službami a operačním systémem, může se při zápisu nebo čtení lehce stát něco neočekávaného - nejde vytvořit soubor, do kterého chcete zapisovat, soubor na čtení neexistuje tam, kde ho hledáte a podobně. Pokud chceme, aby nám v takovýchto situacích neskončil program s chybou, ale nějak se s situací graciálně vypořádal, potřebujeme nástroje na *obsluhu výjimek* a o nich budeme mluvit za chvíli.

U čtení zápisu bychom rádi měli jistotu, že ať se stane cokoli, soubor se zavře. Proto standardně obsluhujeme soubory pomocí **kontextového manažera** takto:

```
1 with open("soubor.txt", "w") as f:
2     f.write("Hej, mistře!\n")
```

`f.close()` se zavolá automaticky po opuštění bloku `with` a to i v případě, že se stane něco neočekávaného.

Metody souborů

`f.write(text)` – zapíše text

`f.read(n)` – přečte dalších `n` znaků, na konci " ".

`f.read()` – přečte zbývající znaky souboru

`f.readline()` – přečte další řádek (včetně "\n") nebo " ".

`f.seek(...)` – přesune se na další pozici v souboru

Další operace:

`print(..., file=f)`

`for line in f:` – cyklus přes řádky souboru

Pozor, řádky končí "\n", hodí se zavolat `rstrip()`.

Vždy je k dispozici:

`sys.stdin` – standardní vstup (odtud čte `input()`)

`sys.stdout` – standardní výstup (sem píše `print()`)

`sys.stderr` – standardní chybový výstup

Chyby a výjimky

```
1 def divide(x, y):
2     return x/y
3
4 divide(1, 0)
5
6 Traceback (most recent call last):
7   File "<pyshe11#73>", line 1, in <module>
8     divide(1,0)
9   File "<pyshe11#72>", line 2, in divide
10    return x/y
11 ZeroDivisionError: division by zero
```

Chyba vygeneruje výjimku, např.

`ZeroDivisionError` – dělení nulou

`ValueError` – chybný argument

`IndexError` – přístup k indexu mimo rozsah

`KeyError` – dotaz na hodnotu neexistujícího klíče ve slovníku

`FileNotFoundError` – pokus o otevření neexistujícího souboru ke čtení

`MemoryError` – vyčerpání dostupné paměti

`KeyboardInterrupt` – běh programu byl přerušen stiskem `Ctrl-C`

`StopIteration` – žádost o novou hodnotu z vyčerpaného iterátoru

```
1 try:
2     x, y = map(int, input().split())
3     print(x/y)
4 except ZeroDivisionError:
5     print("Nulou dělit neumím.")
6 except ValueError as x:
7     print("Chyba:", x)
8     print("Zadejte prosím dvě čísla.")
```

Výjimky jsou objekty, jejich typy jsou třídy.

Výjimka se umí vypsat příkazem `print`

Atributy výjimky obsahují dodatečné informace o tom, co a kde se stalo.

Výjimky tvoří hierarchie, například `FileNotFoundError` je potomkem `IOError`. Můžeme zachytit obecnější typ a doptat se, o kterého potomka se jedná.

```
1 >>> raise RuntimeError("Jejda!")
2 Traceback (most recent call last):
3   File "<pyshell#75>", line 1, in <module>
4     raise RuntimeError("Jejda!")
5 RuntimeError: Jejda!
6
7 >>> assert 1 == 2
8 Traceback (most recent call last):
9   File "<pyshell#78>", line 1, in <module>
10    assert 1 == 2
11 AssertionError
12
13 >>> assert 1 == 2, "Pravda už není, co bývala!"
14 Traceback (most recent call last):
15   File "<pyshell#79>", line 1, in <module>
16     assert 1 == 2, "Pravda už není, co bývala!"
17 AssertionError: Pravda už není, co bývala!
```