Programování 1.12: (Ne)Standardní knihovny

Petr Čermák
cermak@mag.mff.cuni.cz

Katedra Fyziky Kondenzovaných Látek
MFF UK Praha

2020
forked from https://gitlab.kam.mff.cuni.cz/mj/prm1

Moduly a importování

Programy mohou být členěné na **moduly**.

Provedeme-li:

```
>>> import math
>>> math.sin(1)
```

načte se soubor `math.py` (hledá se v aktuálním adresáři a pak v knihovnách).

Formálně: **math** je objekt, jako jeho atributy vidíme funkce a proměnné definované uvnitř modulu.

Alternativně:

```
>>> from math import sin
>>> from sys import *
```

navíc zkopíruje do aktuálního modulu definice z importovaného modulu (takže pak **sin** znamená totéž co **math.sin** apod.).

Už jsme potkali…

- **math** – matematické funkce a konstanty
- **operator** – funkční podoba operátorů (třeba **add** pro $+$)
- **collections** – typ **defaultdict**
  - také tu je **deque** (double-ended queue) – seznam s rychlým přidáváním a odebíráním na obou koncích

Binární řetězce

Typ **bytes** obsahuje neměnnou posloupnost bajtů (8-bitových hodnot).

Literály: `b'Brum' b'\x62\x72\x75\x6d'`

```
>>> "číslo".encode('utf-8')
b'\xc4\x8d\xc3\xadslo'
>>> b'\xc4\x8d\xc3\xadslo'.decode('utf-8')
'číslo'
>>> bytes([1, 2, 3])
b'\x01\x02\x03'
```

Při práci s binárními soubory čteme/zapisujeme bytes.

Binární seznamy

Typ **bytearray:**

- jako `bytes`, ale lze modifikovat
- seznam bajtů (prostorově efektivnější než běžný seznam)
- `bytearray(n)` – seznam *n* nul
- `bytearray([1,2,3])`
- `bytearray('řetězec', 'utf-8')`

Homogenní pole

```
>>> import array
>>> a = array.array('i', [1, 2])
>>> a.itemsize
4
```

'i' je kód typu položek, například:

- **i** – integer: aspoň 32-bitové číslo se znaménkem
- **I** – totéž bez znaménka
- **b** – bajt se znaménkem
- **q** – aspoň 64-bitové číslo se znaménkem
- **f** – float: aspoň 32-bitové desetinné číslo
- **d** – double: aspoň 64-bitové desetinné číslo

Lepší pole s NumPy

pip install numpy

```
>>> import numpy as np
>>> a = np.array([1, 2])
>>> a.itemsize
4

>>> a.dtype
dtype('int32')

>>> a = np.array([1,2.2])
>>> a.dtype
dtype('float64')

>>> a = np.array([1,22145354345345345345345345473])
>>> a.dtype
dtype('O')
```

Zlomky

```
>>> from fractions import Fraction
>>> Fraction(1, 2) + Fraction(1, 3)
Fraction(5, 6)

>>> Fraction(1/3)
Fraction(6004799503160661, 18014398509481984)

>>> Fraction(1/3).limit_denominator(100000)
Fraction(1, 3)

>>> Fraction("1/3")
Fraction(1, 3)

>>> print(Fraction(1, 3))
'1/3'
```

Pseudonáhodný generátor

```
>>> import random
>>> random.random()      (Mersenne Twister)
0.2894785770291432 6     (z intervalu [0, 1])

>>> random.uniform(0, 1000)
50.64122748168531      (z intervalu [a, b])

>>> random.randrange(0, 1000)
524     (celé číslo od a do b − 1, dolní mez lze vynechat)

>>> random.seed(12345)
>>> random.random()
0.4166198725453411 6     (vyjde vždy stejně)
```

Pseudonáhodný výběr

```
>>> random.choice(['pátek','sobota','neděle'])
'neděle'
```
(náhodný prvek seznamu)

```
>>> random.choices("0123456789", k=10)
['3','4','9','8','1','3','2','1','8','6']
```
(výběr s vracením zpět)

```
>>> random.sample("0123456789", k=10)
['3','0','6','1','2','7','4','9','5','8']
```
(výběr bez vracení)

```
>>> random.sample(range(1000000), k=5)
[301599, 128323, 695182, 627325, 967424]
```
(vzorkuje přímo rozsah, nepřevádí na seznam)

Pseudonáhodný výběr

Pozor, Mersenne twister algoritmus je sice robustní (perioda 2**19937-1), ale není vhodný pro kryptografii. Statistickou analýzou lze odhadnout, kde v sekvenci jsme.

Proto Python ještě umí:

```
>>> import secrets
>>> secrets.randbelow(10)      (Crypto-secure os-based)
6    (celé číslo do a-1)
```

Pro authentifikaci lze použít například:

```
>>> secrets.token_hex(16)
f9bf78b9a18ce6d46a0cd2b0b86df9da
```

Další zajímavé moduly

- **datetime**, **calendar** – práce s datem a časem
- **copy** – rekurzivní kopírování
- **cmath** – počítání s komplexními čísly
- **statistics** – základní statistické funkce
- **pickle** – (de)serializace datových struktur
- **re** – analýza textu pomocí regulárních výrazů

Úkoly na poslední hodinu

- Simulujte 1000 hodů kostkou. Spočítejte výskyty každého čísla. Kolik jich je nejméně a kolik nejvíce?
- Generujte náhodně body ve čtverci [-1,1] × [-1,1]. Počítejte, kolik z nich padlo do jednotkového kruhu, a tím aproximujte π.
- Generujte náhodně permutace slov věty ”Kobyla má malý bok”.
- Pomocí Malé Fermatovy věty testujte, zda je nějaké velké číslo pravděpodobně prvočíslem. Hodí se tříparametrová funkce pow(a, b, c), která efektivně spočítá a^b mod c.
- Simulujte náhodnou procházku po celých číslech od 0 do N. Začínáme v 0, v každém kroku náhodně buď zvýšíme nebo snížíme o 1 (v 0 jen zvyšujeme). Po kolika krocích se dostaneme do N?