: my[py]

Clean code & code style ~ Jan Havlík `xhavlik@mendelu.cz`

PEP 8

- Python Enhancement Proposals
- základní pravidla psaní kódu
- předpoklad - kód se více čte, než píše ~ hlavní je **čitelnost**
- konzistence pomáhá s čitelností kódu (kdy se nemusí vyplatit být konzistentní - držet se za každou cenu PEP8?)

Zen of Python (PEP 20)

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
```

PEP 8 - Struktura kódu

Odsazení

- 4 mezery

```
# Zarovnání s oddělovačem (závorkou)
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Přidání 4 dalších mezer (extra úroveň zanoření) pro oddělení od kódu
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

PEP 8 - Struktura kódu

- `if` podmínka

```
# Bez indentace
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

# S komentářem - podpoří rozdělení kódu a podmínky
if (this_is_one_thing and
    that_is_another_thing):
    # Since both conditions are true, we can frobnicate.
    do_something()
```

PEP 8 - Struktura kódu

Odsazení

- inicializace dat

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

PEP 8 - Struktura kódu

Taby nebo mezery?

- vždy mezery, taby pouze v případě, že už to tak někdo *špatně* napsal
  - nelze mixovat mezi sebou

Délka řádku

- 79 znaků (pro komentáře 72)

PEP 8 - Struktura kódu

Importy

- v následujícím pořadí:
  1. Standardní knihovny
  2. Knihovny třetích stran (`pypi`)
  3. Lokální moduly
- samostatně, na každém řádku

```
import os
import sys
```

PEP 8 - Struktura kódu

Importy

- nepoužívat wildcart import (všeho)

```
from os import *
```

  možná kolize atd.

- lépe abosultní, než relativní cesty

```
from tensorflow import keras
from tensorflow.keras import layers
```

PEP 8 - Struktura kódu

Whitespaces

- pozor na extra znaky

```
# Spravne
spam(ham[1], {eggs: 2})

# Spatne
spam( ham[ 1 ], { eggs: 2 } )
```

PEP 8 - Struktura kódu

Whitespaces

- mezera ihned za čárkou, dvojtečkou, středníkem

```
if x == 4: print(x, y); x, y = y, x
```

- indexace, funkce

```
# Spravne
dct['key'] = lst[index]

# Spatne
dct ['key'] = lst [index]
```

PEP 8 - Struktura kódu

Whitespaces

- trailing comma (tuple s jedním prvkem)

```
# Spravne:
foo = (0,)

# Spatne:
bar = (0, )
```

- pozor na **trailing whitespace**, soubor končit **novým řádkem**

PEP 8 - Struktura kódu

Whitespaces

- nezarovnávat hodnoty proměnných

```
# Spravne
x = 1
y = 2
long_variable = 3

# Spatne
x             = 1
y             = 2
long_variable = 3
```

PEP 8 - Struktura kódu

Komentáře

- **Neaktuální komentáře jsou horší než žádné komentáře**
- Každý komentář - celá věta
- **vždy** anglicky

In-line komentáře

- `print(x) # FIXME: remove debug print`
- not too obvious
- 2 mezery za kódem, 1 za mříží

PEP 8 - Naming conventions

- pozor na klíčová slova (`sum` , ...), zaměnitelná písmena v názvu funkcí (O, I, ...)

Názvy metod, funkcí, proměnných

- vystihující snake case

Názvy modulů

- krátké, vše lowercase

PEP 8 - Naming conventions

Názvy tříd

- CapWords (metody ale *snake case*)

Názvy výjmek (Exceptions)

- CapWords, ale končící suffixem `Error`

Globání proměnné

- pls don't

PEP 257 - Docstring

- komentáře na úrovni modulu, funkcí, tříd, metod, ...
- možnost generování dokumentace (`sphinx`)
- vyžadováno některými tooly (`pylint`)
- jednořádkové docstringy (trojité uvozovky vždy - jednoduše rozšiřitelné):

```
def function(a, b):
    """Do X and return a list."""
    pass
```

PEP 257 - Docstring

- multi-line docstring

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments: <- pripadne numpy/google style parametry
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """

    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

Pylint

- statická analýza kódu - bez nutnosti spuštění
- hledání chyb, mrtvého kódu, `code smells`
- navrhuje refaktor kódu

1. Zformátujte soubor `pylint/simplecaesar.py` pomocí návrhů `pylintu`
2. Upravte soubor `pylint/return.py` tak, aby se nezobrazovala hláška `function-redefined`
3. Upravte soubor `pylint/return.py` tak, aby `pylint` ignoroval chybu `too-many-return-statements`

`isort`

- automatické řazení importů
- použití na konkrétní adresář, soubor
- `python -m isort .`

1. Vytvořte soubor `isort.py`, kam přidáte 2 systémové importy, 2 importy třetích stran a 2 vlastní moduly (např. jedna funkce). Seřaďte tento soubor pomocí nástroje `isort`

`black`

- automatické formátování kódu
- použití na konkrétní adresář, soubor
- `python -m black {source_file_or_directory}...`

1. Zformátujte soubor `black/simplecaesar.py` pomocí nástroje `black` - jak se liší výstup oproti pylintu?
2. Zformátujte soubor `black/black2.py` pomocí nástroje `black` - proč dojde k chybě?

Mypy

- statická kontrola typů
- na `mypy` založeno `PEP 484 – Type Hints` - přidání typů k proměnným, se kterými pracujeme

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

Mypy

Základní typy

- `str`, `int`, `float`, `bool`, `None` - podobnost s datovými typy v `numpy`

Mypy

Datové struktury

- z modulu `typing`:

```
from typing import TypeVar, Iterable, List, Callable
def feeder(get_next_item: Callable[List[str], str]) -> None:
    pass
```

Mypy

Generika

```
from typing import TypeVar, Any

T = TypeVar('T', int, float, complex)

def func(x: T = 1.0, y: Any) -> None:
    pass
```

Mypy

Specifikace parametrů

```
from typing import Union, Optional, List, Dict
def func(x: Union[str, int], y: Optional[Dict[str, List[int]]]):
    pass
```

Mypy - použití

- instalace přes `pip`
- `mypy <file>/<folder>`

1. Napište program, který načte od uživatele vstup a přičtěte k němu číslo `1`. Tento program zkontrolujte pomocí `mypy`.
2. Zkontrolujte soubor `mypy/ex1.py`, kde může nastat chyba? Jak byste soubor upravili?

Mypy - použití

3. Napište funkci, která přijmá pouze `float` a `int` a tyto dvě čísla sečte. Zkontrolujte pomocí `mypy`.

- Kontrola typů z jiného modulu:

```
from pathlib import Path

def load_template(template_path: Path, name: str) -> str:
    # Mypy knows that `file_path` has a `read_text` method that returns a str
    template = template_path.read_text()
    # ...so it understands this line type checks
    return template.replace('USERNAME', name)
```