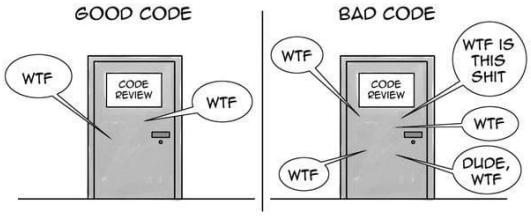
Jak programovat v Pythonu pro středně pokročilé

(B3B33LAR – Laboratoře robotiky)

David Koníček
david.konicek@email.cz
2023 březen/duben

Jak programovat? Čistě!



THE ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE

Jak programovat v Pythonu pro středně pokročilé

- Úvod
- Motivace
- Python čistý kód
- Python PEP8
- Komentáře
- Identifikátory
- Struktura souboru
- Dokumentace v kódu
- Programovací techniky
- Typové kontroly
- GIT – netradiční pohled
- Testování
- Zdroje, další studium

Motivace k čistému kódu

Jak programovat čistě? Jaký kód?

- čistý kód
- čitelný kód
- znovupoužitelný kód
- udržovatelný kód
- v průmyslové kvalitě
- *clean code*
- *readable code*
- *reusable code*
- *maintainable code*
- *industry-strength programming*

Motto (3x)

„Program is written once, but read 50 times“
Anonymous

„Programs are meant to be read by humans and only incidentally for computers to execute.“
Donald Knuth

„Eighty percent of the lifetime cost of a piece of software goes to maintenance.“
Nicolas Zakas in Maintainable JavaScript

Čistý kód je novinka?

- Vždy u tvorby software byla snaha o správný a čitelný kód
- Rozvoj složitosti programů a řešených oblastí ➔

- Strukturované programování
- Modulární programování
- Objektové programování
- Čistý kód, návrhové vzory a další paradigmata

Proč programovat čistě? 1/2

- **Komplexnost** kódu roste
- Přesto kód musí být:
  - čitelný
  - pochopitelný
  - udržovatelný

  (včetně hledání chyb i rozvoje kódu)

Proč programovat čistě? 2/2

- **Tým - práce v týmu**
- Předávání kódu „v čase" a mezi týmy/projekty

- Kód v týmu musí být:
  - čitelný
  - pochopitelný
  - udržovatelný

  (včetně hledání chyb i rozvoje kódu)

Výhody a přínosy čistého kódu

- Snadný k **porozumění**
- Snadný k porozumění, co **bude kód dělat**
- Snadný k porozumění i **za 6 měsíců nebo 6 let**
- Snadný k porozumění **pro ostatní v týmu**
- Snadný k porozumění **mimo tým** (API, knihovny, …)
- Snadný/jednodušší pro **testování**

Kdy programovat čistě?

- **Vždy!** Platí pro oblasti:
  - business
  - univerzity
  - open source
- **Výjimky:**
  - pokusný a jednorázový kód („trvanlivost" < 1 den)
  - starý kód
  - výukové materiály pro promítání (prostorové omezení)
  - programátorské soutěže

Jak programovat čistě

- Změna přístupu – jednotlivec, tým, management
- Standardní/zvyková pravidla a konvence

- Sebedisciplína – jednotlivce i týmu
- Pravidla programovacího jazyka
- Využití nástrojů statické kontroly

Subjektivnost

- Měřítkem "čistoty" je vnímání kódu člověkem, tedy subjektivní pohled.
- Standardní/zvyková pravidla a konvence ho sjednocují. *(good practices)*

Univerzálnost pravidel

- Neexistují 100% univerzální pravidla/konvence
- Existují obvykle dodržované pravidla/konvence
- Firmy/projekty/atd. definují vlastní modifikace
- Doktrinářský versus volný přístup
  - Doktrinářský = žádné výjimky
  - Volný přístup = možné výjimky, nutno ale popsat

Čistý kód - oblasti

- Formátování kódu (*style guide*)
- Identifikátory – smysluplné, dobře popisné
- Komenáře a dokumentace (vč. dokumentace v kódu)
- Uspořádání souborů a adresářů
- Programátorské praktiky – pochopitelný kód
- Testy – automatizované
- Pokročilá paradigmata (koncepty)

Programovací jazyky – čistý kód

- Čistý kód je obecný princip aplikovatelný na všechny programovací jazyky:
  - (*statické*) Pascal, C, C++, C#, Java, Ada, ...
  - (*dynamické*) Python, PHP, JavaScript, R, MATLAB, ...
  - (*databázové*) SQL, PL/SQL, T-SQL, CQL (Cassandra), ...
  - (*skriptovací*) Shell (bash, csh, sh), PowerShell, AWK, SED, ...
  - (*značkovací*) HTML, CSS, XML, ...

Programovací jazyky – čistý kód

Python – čistý kód

Python – základy čistého kódu

- **PEP 8** - Style Guide for Python Code
- **PEP 484** - Type Hints + **PEP 485** - The Theory of Type Hints
- **PEP 257** - Docstring Conventions
- kvalitně pojmenované **identifikátory**
- čtivé a smysluplné **komentáře**
- **PEP 20** - The Zen of Python
- **PEP 440** - Version Identification

Další zdroje/inspirace

- Google Python Style Guide

  https://github.com/google/styleguide/blob/gh-pages/pyguide.md

  https://google.github.io/styleguide/pyguide.html

- Reitz & Schlusser:
  Hitchhiker's Guide to Python Style Guide

  https://docs.python-guide.org/writing/style/

Další zdroje/inspirace

- Další (obecná) paradigmata
- Další programovací jazyky
  - C++
  - Java
  - JavaScript
  - ...

PEP = Python Enhancement Proposal

- Původně = návrhy na vylepšení jazyka a knihoven
- Fakticky = schválené vylepšení plus návrhy:
  - Schválené/aktivní/finální
  - Dočasně schválené
  - Zastaralé
  - Otevřené/zvažované
  - Odmítnuté/stažené
- PEP 0 ... https://peps.python.org/ ... seznam všech

PEP = Python Enhancement Proposal

- Schválené/aktivní/finální:
  - Standardizační (*standards track*) ... normativní, jsou součástí jazyka a/nebo standardní knihovny
  - Procesní (*process*) ... řízení procesu schvalování PEP
  - Informační, historické atd. ... nejsou normativní

PEP 8 - Style Guide for Python Code

jako základ pro čistý styl

PEP 8 - Style Guide for Python Code

- Tj. definice pravidel formátování
- Web … https://peps.python.org/pep-0008/
- Čitelný web … https://pep8.org/
- Srozumitelní vysvětlení: https://realpython.com/python-pep8/
- IDE –obvykle silná podpora
- Možno použít také CLI nástroje (pycodestyle, flake8, …)

PEP 8
základy formátování kódu

PEP 8 – znaková sada

- Python pravidla: **UTF-8** (Python 3); ASCII (Python 2)
- Tj. netřeba: `# -*- coding: utf-8 -*-`
- PEP 8 pravidla:
  - pouze anglická abeceda a číslice: **a .. z, A .. Z, 0 .. 9**
  - speciální znaky: **!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~**
  - mezery (\x20) a konce řádku (\n, \x0A, \r, \x0D)
  - nikdy TAB (\t, \x09) a FormFeed (\f, \x0C, Ctrl + L)
  - nikdy Unicode "exotické" mezery (\x2000 .. \x200A atd.)

PEP 8 – znaková sada

- Důsledek: „neanglické" znaky pouze:
  - vlastní jména (lidi, instituce, místa, …)
  - výstupní text (textové konstanty) pro uživatele
  - textové konstanty pro analýzu dat

PEP 8 - odsazování

- Odsazování … **4 mezery**
  (chytrý editor/IDE konvertuje z klávesy TAB)
- Rozdělené řádky – varianty:
  - Zarovnání pod otevírací závorku
  - Odsazení 4/8 mezery + za otevírací závorkou konec řádky
- Nejrychlejší řešení … automatické formátování:
  - PyCharm : Ctrl + Alt + L
  - Visual Studio Code: XXX

PEP 8 – délka řádky

- Délka řádky < 80 znaků
- Často výjimky (99 nebo 119 nebo 120 znaků)
- Python Standard Library < 79 znaků
- Pozn. pro nastavení: *vertical edge; hard wrap; rules*

PEP 8 – rozdělování řádky

- Pokračovací řádky – varianty:
  - Zarovnání pod otevírací závorku
  - Odsazení 4/8 mezery & otevírací závorka jen s koncem řádky
- Pokračovací řádky … předchozí řádka končí znakem `\` (nedoporučuji, dtto Google)
- Pokračovací řádky … **otevřené závorky** `()[]{}` (preferované)
- Pokračovací řádky … začínají případným binárním operátorem (*Knuth's method*)

PEP 8 – složené příkazy

- Vždy složené příkazy na více řádků
- Tj. nikdy 2 příkazy na jednu řádku; nikdy `;`

- Pozn. PEP 8 povoluje výjimku pro jednoduché případy; nedoporučuji

ANO:
```
if color == "red":
    stop_vehicle()
```

NE:
```
if color == "red": stop_vehicle()
```


PEP 8 – prázdné řádky

- Mezi funkcemi/třídami … 2 řádky → 3 řádky
- Mezi metodami … 1 řádka → 2/3 řádky
- Mezi a vnořenými funkcemi … 1 řádka → 2/3 řádky
- Oddělení úseků souboru … 1 řádka → 2 řádky
- Oddělení logických úseků kódu … 1 řádka
- Pro zvýšená čitelnosti kódu … 1 řádka
- PEP8 povoluje, ale nedoporučuji: Ctrl/L
- Výjimky

PEP 8 - Řetězce

- Jednoduché řetězce: uvozovky `"` nebo apostrof `'`
- Doporučuji pro čitelnost: uvozovky `"`

- Víceřádkové řetězce: 3x apostrof `''' text '''` (!!!)
- Docstring: 3x uvozovky `""" text """` (!!!)

```
"""\
text
"""
```


PEP 8 - Mezery ve výrazech a příkazech

PEP 8 – ANO, používat mezery

- **za** čárkou, ~~středníkem~~, dvojtečkou

- **oboustranně** u binárních operátorů
  = + - * / < == > <> <= >= != : :: in not in is is not and or not ->

- složité výrazy – obvykle jen kolem nejméně prioritních operátorů

PEP 8 – ANO, používat mezery

Speciální případ (výjimka z "NE" pravidla):

- oboustraně u rovnítka `=` pro implicitní hodnoty parametrů funkce s (!!!) type hintingem

PEP 8 – NE, nepoužívat mezery

- na konci řádky
- **za** otevírací závorkou `( [ {`
- **před** zavírací závorkou `) ] }`
- vč. čárky před zavírací závorkou `(1,)`
- před čárkou `y, x = x, y` a středníkem

PEP 8 – NE, nepoužívat mezery

- oboustraně u rovnítka `=` pro implicitní hodnoty parametrů funkce bez type hinting

→ **výjimka … ANO, používat mezery**

- oboustraně u rovnítka `=` u pojmenovaných argumentů funkce

→ **výjimka … ANO, používat mezery**

PEP 8 – NE, nepoužívat mezery

- před dvojtečkou u příkazů `if x == 4: y = 5`
- před dvojtečkou u type hinting
- před závorkou u volání funkce/metody `fce(x)`
- před závorkou u indexování a řezů `array[1:3]`
- vícenásobné mezery

→ **výjimka "tabulka" (deklarace/volání funkce)**
→ **výjimka "tabulka" (definice strukturovaných konstant)**
→ **výjimka "tabulka" type hinting (deklarace funkce)**

PEP 8 – ukončující čárky

- Tj. ukončující závorka před uzavírací závorkou
- Povinně – odlišení závorek u výrazu a inicializaci jednočleného tuple

```
FILES = ("setup.cfg",)
```

- Volitelně – definice seznamů, tuple, slovníků, parametrů, ale musí být víceřádkové

Komentáře (nejen PEP 8)

Kdy komentáře

- Nejasný kód; obtížně pochopitelný
- Zvýšení čitelnosti
- Dokumentace kódu
- Ideálně: jen Docstring + speciální příznaky

PEP 8 - komentáře

- Blokové:
  - `#` a mezera
  - na samostatném řádku (!!!)
  - předchozí řádek před příslušným příkazem (!!!)
  - obvykle jeden prázdný řádek před komentářem
- Jednořádkové (inline)
  - nejméně 2x mezera `#` a mezera
  - na konci řádku

Blokové a inline komentáře

```
# We use a weighted dictionary search to find
# out where i is in the array.  We extrapolate
# position based on the largest number
# in the array and the array size and then
# do binary search to get the exact number.


if i & (i-1) == 0:  # True if it's 0 or power of 2.
```


PEP 8 - komentáře

- Dokumentační řetězce = docstring
- PEP 257 - Docstring konvence
- Docstring: 3x uvozovky `""" text """` (!!!)

```
"""\
text
"""
```


Komentáře

- Anglicky
- Gramaticky správně včetně spellingu, gramatiky a interpunkce
- Komentář musí být normálně čitelný text
- Nekomentujte to, co je přímo v příkazu

NE:
```
# The total number of elements is count + 1.
count = count + 1
```


Komentáře

- Nikdy – "odstranění" kódu, "zakomentování" kódu
- Udržovat komentář v souladu s kódem (!!!)
- Často lze nahradit komentář vhodným názvem proměnné/funkce/metody
- Často lze nahradit komentář voláním vhodně pojmenované (nově vytvořené) funkce/metody realizující celý blok komentovaného kódu

Komentáře – speciální příznaky

A. řídící příkazy pro systémy statické analýzy
B. speciální komentáře

 # TODO: blabla … popis toho, co je nutné dodělat
 # REALLY: blabla …. krátce, co je vlastně správně (ačkoliv kód vypadá, že je chybný, resp. podezřelý)

```
except Exception:   # REALLY: catch all exceptions
```

```
pass # REALLY: do nothing
```


Komentáře – speciální příznaky

 # MAGIC: blabla …. triky, obtížné na pochopení, atd.

 # HACK (for … browser/OS/library …): blabla… popis speciálních postupů … (nebo nic, je-li to jasné z kódu)

 # WORKAROUND (for … what …): … popis chyby a jejího napravení … (nebo nic, je-li to jasné z kódu)

Komentáře – speciální příznaky

REQUIRES: blabla … popis, co je vyžadováno

 # ASSUMPTION: blabla … dtto (lépe – příkaz `assume` nebo zahrnout do komentáře celé funkce/metody/třídy/modulu).

 # SIDE-EFFECT: blabla …. popsat používaný nebo využívaný vedlejší efekt deklarované nebo používané funkce

 # DIRTY: blabla …. Zdůraznit a okomentovat opravdu nevhodný kód, který je ale nezbytný

Jmenné konvence, identifikátory atd. (nejen PEP 8)

PEP 8 – jmenné konvence

- Popisné (!!!) identifikátory
- Popis podle funkcionality, ne podle implementace (povinně pro veřejné API, volitelně pro interní funkce a metody)
- Nikdy matoucí/zaměnitelné:
  - klíčová slova
  - velké íííí
  - malé eeeelll
  - velké óóó

I
l
O

PEP 8 – jmenné konvence

- Povinně ASCII
- Preferovaně anglicky
  - včetně spellingu
  - ale bez gramatické preciznosti (členy, "to", "of", "'s")
- Proměnné, funkce, metody, konstanty, soubory, adresáře, moduly, balíčky

PEP 8 - jmenné konvence

- **Konstanta** CAPITAL_LETTERS, popř. podtržítka
- **Proměnná** lowercase_name, popř. podtržítka
- **Funkce a metoda** lowercase_name, popř. podtržítka
- **Třída** (class) CapitalisedWords

PEP 8 - Jmenné konvence

- **Vyjímka** (exception) CapitalisedWords**Error**
- **Typy** CapitalisedWords
- **Balíček** (package) lowercasename
  tj. jméno adresáře
- **Modul** lowercase_name, popř. podtržítka
  tj. jméno souboru

PEP 8 – jmenné konvence

- **Speciální globální proměnné** __xyz__
  (*"dunder"*)
- **Speciální metody tříd** __xyz__
  (*"dunder"*)
- **Privátní metody a členy tříd a modulů** _alfa
- **Metody instancí – první argument** `self`
- **Metody tříd – první argument** `cls`

Proměnné a atributy - konvence

- Začínají **podstatným jménem**,
  případná příčestí následují; bez členů
- **Číselné hodnoty** nejlépe:
  xxx_count xxx_length xxx_size, ...
- **Řetězcové hodnoty** nejlépe:
  xxx_name xxx_title message_xxx, ...

Funkce a metody - konvence

- Začínají **slovesem**, nejlépe akční sloveso:
  load_xxx fetch_xxx run_xxx compute_xxx, ...
- **Predikáty**, nejlépe sloveso navozující odpověď ANO/NE:
  is_xxx has_xxx was_xxx, ...
- **Atributy metod**, slovesa get, set, has:
  get_xxx set_xxx has_xxx
- Konzistentí použití všech slov (!!!)

Nevhodná jména

- Nevyslovitelná a nesmyslná slova včetně zkratek
- Neslušná slova
- Slangová slova
- Obsahující datový typ (vč. "maďarské notace")
- Názvy obvyklých knihoven a jejich častých funkcí
- Matoucí název neodpovídá významu

Matematika, fyzika, MATLAB, …

- Obvykle pracují s jednopísmennými proměnnými
- Dále si "pomáhají" dalšími abecedami, indexy, …
- Nahradit plnohodnými identifikátory
- Fyzikální jednotky … vše v jednotkách podle SI, jinak jednotku uvést v názvu, obvykle na konci

Jmenné konvence - vyjímky

- Zkratky, které jsou opravdu široce známé, popř. normalizované (ISO/EN/ČSN):
  - měny, státy, ekonomika … (CZK, EUR, … VAT, ID, KPI, …)
  - IT (TCP, UDP, ID, HTTP, URI, …)
  - aplikační doména
- Jednopísmené/zkrácené v opravdu krátkém (!!!) kódu :
  - pro indexaci bez dalšího významu (i, j, k, …)
  - pomocné proměnné (tmp, txt, len, …)

Příkaz import, použití modulů a balíčků

Příkazy `import`

- Python je modulární jazyk:
  - Rozdělení problému na menší části
  - Znovupoužitelný kód
  - Využití kódu jiných autorů
- Modul v Pythonu:
  1. Python soubor
  2. C modul
  3. Vestavěné (built-in) moduly

Příkazy `import`

- Balíček (*package*)
  - více modulů
  - tečková notace
  - adresářový strom
  - povinně soubor `__init__.py`
- Příkaz import – zpřístupnění modulů/balíčků:
  - Moduly – hledá soubory
  - Balíčky – hledá adresáře se souborem `__init__.py`

Příkazy `import` – pořadí hledání

- Aktuální adresář
- PYTHONPATH proměnná operačního systému
- Pevná místa instalace Python (podle operačního systému)
- Pozn: Anaconda a VENV (Vitual Environments) mění tyto cesty

PEP 8 - Příkazy `import`

- Vždy na začátku souboru za docstring a komentáře, tj. před konstanty a globální proměnné

- Absolutní import – preferovaný protože čitelnější
- Relativní import – akceptovatelné, ale nedoporučuji
  - `from .some_module import some_class`
  - `from ..some_package import some_function`
  - `from . import some_class`

PEP 8 – Pořadí příkazů `import`

- Každý modul/balíček na samostatné řádce
- Doporučuji stejně, je-li import více funkcí, objektů, atd.

- Příkazy import rozděleny na sekce podle druhů modulů/balíčků
- Sekce oddělené prázdnou řádkou
- Popř. každá sekce abecedně řazená

PEP 8 – Pořadí příkazů `import`

`from __future__ import alfa_beta` — 0

- standardní knihovna — 1
- dobře známé knihovny třetích stran — 2
- ostatní knihovny třetích stran — 2
- vlastní obecné knihovny — 3
- moduly projektu — 3

Druhy modulů a balíčků

- standardní knihovna

  https://docs.python.org/3/library/index.html

- knihovny třetích stran
  - příkaz `pip install xxxxx`
  - příkaz `python -m pip install xxxxx`
  - příkaz `conda install xxxxx`
  - PyPi = Python Package Index … *https://pypi.org/*

PEP 8 - Příkazy `import`

- Nikdy import všech pomocí hvězdičky

NE: `from xxx import *`

- Výjimka: moduly projektu

ANO: `from parking_turtle_constants import *`

Příkazy `import` (Google)

- Používat pouze pro balíčky a moduly
- Nepoužívat pro individuální třídy, funkce, konstanty

ANO:
```
import pathlib
import matplotlib.pyplot
```

NE:
```
import pathlib.Path
from pathlib import Path
from matplotlib import pyplot
from matplotlib. pyplot import show
```

Použití modulů

- Důsledek:

ANO:

```
import myextraclass
import foo.bar.myclass
```

Explicit is better than implicit.
(Zen of Python)

A pak:

```
data_object = myextraclass.MyExtraClass()
data_object = foo.bar.myclass.MyClass()
```


Příkazy `import` (Google)

- Výjimky:
  - typing
  - collections.abc
  - typing_extensions
- Příklady:

```
from typing import List
from collections.abc import Iterable
```


Příkazy `import` (Google)

- Použití plných jmen pro moduly z balíčků
- Tj. preferovat tečkovou notaci před `from … import`

ANO:

```
import matplotlib.pyplot
import package2.subpackage1.module5
```

NE:

```
from matplotlib import pyplot
from package2.subpackage1 import module5
```


Příkazy `import .. as` (Google)

- Nepřejmenovávat při importu pomocí `as`
- Výjimky:
  - Obvyklé zkratky pro dobře známé knihovny třetích stran (numpy, pandas, matplotlib.pyplot, …)
  - Opravdu dlouhá jména
  - Řešení nekompatibilit různých verzí

ANO:

```
import matplotlib.pyplot as plt
import numpy as np
```

NE:

```
import utils.terminal_width as terminal_width
```


Souvislosti příkazu `import`

- PEP 20 - The Zen of Python

„Explicit is better than implicit."

← Použití plných jmen pro moduly z balíčků
← Nepřejmenovávat při importu pomocí `as`

Struktura souboru s kódem (nejen PEP)

Python struktura souboru

Shebang
Kódování (není nepovinné)
Docstring – dokumentace modulu
Komentář k souboru (není povinné)
Hlavička souboru (autor atd.)
Import modulů a balíčků
Konstanty a globální proměnné
Třídy a funkce
Spuštění programu

Python shebang

- Povinné
- První řádek souboru
- Pro Linux/Unix/POSIX určuje interpreter

ANO: `#!/usr/bin/env python3`

NE: `#!/usr/bin/python3`
NE: `#!/usr/bin/env python`
NE: `#!/usr/bin/python`

Python kódování souboru

- Volitelné
- Druhý řádek souboru
- Obecně: **UTF-8** (Python 3); ASCII (Python 2)
- Tj. netřeba: `# -*- coding: utf-8 -*-`

Docstring modulu (souboru)

- Povinné
- Dokumentace vnořená v programu
- PEP 257 - Docstring Conventions
- Tento bude zobrazován jako nápověda v IDE
- Docstring: 3x uvozovky `""" text """` (!!!)

```
"""\
text
"""
```


Komentář na úrovni modulu

- Volitelné
- Nebude součástí nápovědy zobrazované v IDE
- Varianty:
  - Blokový komentář
  - Víceřádkový řetězec

- Blokový komentář - `#` a mezera

- Víceřádkové řetězce: 3x apostrof `''' text '''` (!!!)

Python hlavička souboru

```
__author__      = "David Koníček"
__maintainer__  = "David Koníček"
__email__       = "david.konicek@cvut.cz"
__copyright__   = "\xa9 2022 CIIRC CTU, Prague."\
                  " All rights reserved."
__license__     = "MIT"
__version__     = "0.1.0"
__date__        = "2023/01/02"
__status__      = "Development"
__credits__     = [""]
__all__         = []
```

Python hlavička souboru

- Metadata programu/modulu.
- „Nepsaný standard" pro všechnu programátorskou práci, nejen Python
- Každý programovací jazyk má svoji syntaxi
- Python vychází z pravidel pro Docstring dle Epydoc

Python hlavička souboru

- Epydoc Fields

https://epydoc.sourceforge.net/manual-fields.html#module-metadata-variables

- Rob Knight: Python Coding Guideliness (for Cogent project)

https://web.archive.org/web/20111010053227/http://jaynes.colorado.edu/PythonGuidelines.html#module_formatting

Python hlavička souboru

- `__author__` – řetězec (!!!), obsahuje jedno či více jmen (oddělovat čárkami nebo středníkem)
- `__maintainer__` – řetězec (!!!), obsahuje jedno či více jmen (oddělovat čárkami nebo středníkem)
- `__email__` – řetězec (!!!), obsahuje jedno či více emailových adres (oddělovat čárkami nebo středníkem)

Autorská práva a licence

- **Copyright = právo autora**, vzniká automaticky
- **Licence = právní ujednání,** definuje (1) práva užití a obvykle (2) omezuje odpovědnost za škodu
- *What's the difference between Copyright and Licensing?*

https://opensource.stackexchange.com/questions/297/whats-the-difference-between-copyright-and-licensing

Autorská práva a licence

- **Copyright = právo autora**; vzniká automaticky ze zákona (zákon 121/2000 Sb., popř. předpisy EU, popř. předpisy země užití, popř. celosvětové dohody)
- Pozn. právo USA a slovo **copyright** a značka **©** jsou zvykově používány, ale v ČR/EU nemají právní význam
- Pro laiky je přijatelný zdroj informací česká Wikipedie

https://cs.wikipedia.org/wiki/Autorský_zákon_(Česko,_2000)

Python hlavička souboru

- **Copyright = právo autora**; vzniká automaticky ze zákona ➔ je deklaratorní/informativní
- Obvykle: držitel práv, rok/roky a deklarace

```
__copyright__ = "\xa9 2022 CIIRC CTU, Prague."\
" All rights reserved."
```

Python hlavička souboru

- Viz ustanovení §60 zákona o školních dílech
  ➔rok/roky, fyzický autor, jméno školy, popř. fakulty
- Viz ustanovení §58 zákona o zaměstnaneckých dílech
  ➔rok/roky, jméno zaměstnavatele

(*Pozn. nebylo konzultováno s právním oddělením*)

- Viz případně uzavřené smlouvy
- Ostatní případy – fyzický autor díla

Autorská práva a licence

- **Licence = právní ujednání**, byť může být ve formě jednostranné deklarace
- Licence … (1) definuje práva užití a obvykle (2) omezuje odpovědnost autora za způsobenou škodu
- Doporučuji neměnit text a držet se toho, co právníci vymysleli; text vesměs univerzální ve smyslu různých národních úprav
- Praxe přinesla jiné texty pro software (GNU/GPL, MIT, APL, …) a pro ostatní autorská díla (CC = Cretive Commons, …)

Licence = právní ujednání

- Pro laiky je přijatelný zdroj informací a zejména **porovnání** anglická Wikipedie:

https://en.wikipedia.org/wiki/Software_license

- Obsáhlý výčet licencí volně dostupného software (GNU, MIT, Apache, ….): **OSI Approved Licenses**

https://opensource.org/licenses/

- Proprietární software přináší mnoho dalších typů licencí

Licence = právní ujednání

- Další možnost licence: WTFPL, ale pozor, neřeší zřeknutí se odpovědnosti za škody:
  http://www.wtfpl.net/
  https://en.wikipedia.org/wiki/WTFPL
- Doporučuji se řídit dle **OSI Approved Licenses** (mezinárodní)
  https://opensource.org/licenses/?categories=international
- Protože jsme členský stát EU (všech 23 jazyků, 27 zemí):
  https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12

Python hlavička souboru

- Základ: váš právní status (student, zaměstnanec, zcela volný),
- Doporučuji zvážit licence: MIT, GPL, EUPL
- Obvykle:
  - standardní zkratka podle webu OSI Approved Licenses
  - přiložený textový soubor s názvem LICENSE nebo LICENSE.TXT s doslovným (!!!) zněním licence

```
__license__    = "MIT"
```


Python hlavička souboru

- `__date__` poslední změny; řetězec (!!!), ideálně automaticky doplňuje verzovací systém (GIT atd.)
- `__version__`
- Status by měl odpovídat logice číslu verze (alfa, beta, release candidate atd.)
- `__status__` – řetězec (!!!), obvykle jedno z
  `"Prototype"`
  `"Development"`
  `"Production"`

Python číslování verzí

- PEP 440 - Version Identification
- Inspirováno – sémantické verzování https://semver.org/
- Inspirováno – datumové/kalendářové verzování https://calver.org/

Obvyklé sémantické verzování

- N.N.N .... číslo.číslo.číslo ... např. 1.43.15
- [major].[minor].[release].[build]
- MAJOR verze ... mění se při změně architektury nebo změně narušující zpětnou kompatibilitu
- MINOR verze ... mění se při přidání funkcionality se zachováním zpětné kompatibility
- RELEASE/PATCH/MICRO verze ... opravy chyb při zachováním zpětné kompatibility
- BUILD ... neustále rostoucí číslo reprezentující vnitřní verzi

Číslování verzí podle PEP 440

`v?[N!]N(.N)*[{a|b|rc}N][(-)?.postN][(-|_|\.)?.devN]`

`v?` nepovinné písmeno „v“
`N!` nepovinné číslo epochy (=změna logiky číslování)
`N(.N)*` jedno nebo více čísel, obvykle N.N.N
`aN` alfa (pod)verze = neobsahuje všechnu funkcionalitu
`bN` beta (pod)verze = veškerá funkcionalita, ale netestováno
`rcN` release candidate (podverze) = interně testováno, „skoro“ dobré, probíhá externí testování

- Pozn.: *alfa - beta- RC - produkční* verze odpovídá standardním fázím vývojového cyklu software

Python hlavička souboru

- `__credits__` – seznam řetězců (!!!), **ne**obsahuje autory importovaných modulů, ale obsahuje autory m.j.:
  - doporučení a rad
  - kteří nahlásili chyby
  - použitých fragmentů kódu
  - inspirativních textů

```
__credits__ = ["Rob Knight", "Peter Maxwell"]
```


Python hlavička souboru

- `__all__` seznam (!!!) řetězců jmen tříd, funkcí, konstant atd., které modul **exportuje**
- Tj. jediná část metadat používaná jazykem Python
- Zároveň zvyšuje čtivost kódu

Python hlavička souboru

- `from xxx import *` a modul neobsahuje `__all__` ➔ importuje opravdu všechno
- `from xxx import *` a modul obsahuje `__all__` ➔ importuje jen objekty z __all__
- `import xxx` a modul obsahuje použití objektů z modulu xxx ➔ kontroluje, že to jsou exportované objekty z __all__

Import, konstanty, proměnné

- Importy – členění do sekcí podle PEP8
- Konstanty – názvy podle PEP8; popisné
- (Popř.) globální proměnné – názvy podle PEP8; popisné; preferovaně s podtržítkem an začátku

Třídy, funkce

- Třídy
- Funkce

➔ V celém modulu není žádný přímo vykonávaný kód

Patička souboru

```
if __name__ == '__main__':
    main()
```

- *What Does if __name__ == "__main__" Do in Python?*

https://realpython.com/if-name-main-python/

"Přímý kód" v souboru

- V celém modulu není žádný přímo vykonávaný kód
- Výjimky:
  - Příkazy import
  - Definice konstant
  - Inicializace globálních proměnných
  - Patička s případným voláním main()
- Další výjimky:
  - Testovací program
  - Grafické programy

PEP 20 - The Zen of Python

- Principy psaní „pythonic" kódu
- Velmi stručné, velmi hutné, často obtížné dodržet

- https://peps.python.org/pep-0020/
- https://pep20.org/

PEP 20 - The Zen of Python

- **Beautiful** is better than ugly.
- **Explicit** is better than implicit.
- **Simple** is better than complex.
- Complex is better than complicated.
- **Flat** is better than nested.

PEP 20 - The Zen of Python

- **Sparse** is better than dense.
- **Readability** counts.
- Special cases are **not special enough to break the rules**.
- Although **practicality** beats purity.
- **Errors should never pass silently**.
- Unless explicitly silenced.

PEP 20 - The Zen of Python

- In the face of ambiguity, refuse the temptation to guess.
- There should be one - and preferably only one - obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.

PEP 20 - The Zen of Python

- Although never is often better than *right* now.
- 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.
- **Namespaces** are one honking great idea -- let's do more of those!

Nástroje

Doporučené nástroje

- Tzv. statická analýza kódu, tj. bez spuštění kódu
- Původně – samostatná nástroje, obvykle CLI (**C**ommand **L**ine **I**nterface = příkazová řádka)
- Moderně – integrace do grafických IDE (**I**ntegrated **D**evelopment **E**nvironmnet)

Python IDE

- **Visual Studio** Community/Professional/Enterprise (Microsoft)
- **Visual Studio Code** (Microsoft)
- **PyCharm** (JetBrains): Community/Professional
- **Jupyter**
- **Spyder**
- **PyDev**/LiClipse/Eclipse
- **PyScripter**
- Omezeně: editory (vim, Emacs, Atom, Sublime, Notepad++, ...)
- Omezeně: (Python) IDLE

Výhody (Python) IDE

- Zvýraznění syntaxe
- Našeptávání/dokončování
- Ladění
- Statická analýza zobrazená v kódu
- Integrace GIT
- Nástroje pro další jazyky a soubory (HTML, CSS, Markdown, make, shell, …)

Nástroje kontroly kódu

- PEP8 formátování / styl → pycodestyle/pep8, **Flake8**
- Datové typy (*type checker*) → **Mypy**, Pyright, Pytype
- Detekce chyby (*error linter*) → **Pylint**, pyflakes, **Flake8**
- Nepoužitý / mrtvý kód → Vultare, eradicate
- Komplexita / nepřehlednost → McCabe, Radon
- Bezpečnost (hlavně web) → Bezpečnost: Bandit

Nečistý kód (*code smells*)

Další nástroje

- Balíčkování (*packaging):* Pyroma
- Formátování kódu podle PEP8: black, autopep8
- Formátování Docstgring: pydocstringformatter, docformatter

Doporučené nástroje – pep8

- pycodestyle (= pep8) kontrola stylu podle PEP 8
- Integrován v PyCharm
- Dokumentace: https://pycodestyle.pycqa.org/en/latest/intro.html
- Seznam pravidel/chyb – vhodné pro konfiguraci výjimek: https://pycodestyle.pycqa.org/en/latest/intro.html#error-codes https://pypi.org/project/pep8-naming/

Doporučené nástroje – flake8

- flake8 (= PyFlakes + pycodestyle + McCabe)
  - Pycodestyle = pep8 = kontrola stylu
  - Pyflakes = kontrola syntaxe a hrubých chyb programování
  - McCabe = kontrola cyklomatické komplexity
- Dokumentace https://media.readthedocs.org/pdf/flake8/latest/flake8.pdf
- Seznam pravidel/chyb – vhodné pro konfiguraci výjimek: https://flake8.pycqa.org/en/latest/user/error-codes.html https://www.flake8rules.com/

Doporučené nástroje

- Pylint
- MyPy
- SonarLint
- pydocstyle

- PyCharm & VisualStudioCode – nutné instalovat

Komplexita / nepřehlednost

- Špatná čitelnost, testovatelnost a udržovatelnost pro příliš složitou strukturu kódu ➔ měření složitosti
- **Cyklomatická komplexita**
  (Thomas McCabe, 1976. DoD)
- Cca počítá definice funkcí/metod, větvení a cykly a jejich vnořování

Komplexita / nepřehlednost

- **Kognitivní komplexita**
  (G. Ann Campbell, 2016?, SonarSource)
- Cca počítá narušení lineárního toku algoritmu (shora dolů, zprava doleva)
- Více penalizuje vnořené struktury
- Nepenalizuje definice funkcí, metod a tříd
- Lépe vyhovuje moderním programovacím jazykům a moderním paradigmatům

Řešení příliš vysoké komplexity

- Vysoká komplexita (obvykle) správně indikuje nepřehledný kód
- Řešení ➔ refaktorovat kód, např.:
  - jednodušší podmínky
  - vnořené if ➔ řetězec if – elif – elif …
  - Vnořené cykly a podmínky rozdělit na více menších funkcí/metod