Metodika na štýl kódu - Python

Pri vytváraní nasledujúcej metodiky som čerpal z týchto zdrojov:

- https://www.python.org/dev/peps/pep-0008/
- http://www.voidspace.org.uk/python/articles/python_style_guide.shtml#standards-and-style

V pythone odsadenie ohraničuje bloky kódu (for, if,...)

```
if x == 3:
    print "we have 3"            # patrí do podmineky
print "we also have x"           # nepatrí do podmineky
```

Na odsadenie používame 4 medzery. Nepoužívame tab.

Ak máme na jednom riadku príliš dlhé volanie funkcie:

Správne:

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

Nesprávne:

```
foo = long_function_name(var_one, var_two,
        var_three, var_four)
```

Pri podmienke if to môže vyzerať takto ak je príliš dlhá na jeden riadok:

```
if (very_long_stuff and
    this_is_quite_long):
    print "we fulfilled both"
```

Vyššie uvedený príklad je dosť nečitateľný. Odporúčané riešenie je pridať riadok komentáru.

```
if (very_long_stuff and
    this_is_very_long_too):
    # If we fulfilled both
    print "we fulfilled both"
```

Ak vytvárame nejaký objekt, ktorého definícia je na viac riadkov, koncovú zátvorku uvádzame na samostatný riadok:

Správne:

```
list = [
    1, 2, 3,
    4, 5, 6,
    ]
```

Nesprávne:

```
list = [
    1, 2, 3,
    4, 5, 6, ]
```

Ak v kóde uvádzame nejaké matematické operácie, operátor uvádzame na nový riadok.Tento spôsob zápisu umožní ľahšie priradenie operandu k operátoru.

Správne:

```
result = (food
          + drinks
          + sweets
          - donations)
```

Nesprávne:

```
result = (food +
          Drinks +
          Sweets -
          donations)
```

Maximálna dĺžka riadku kódu je 79 znakov.
Pre komentáre a docstring-y je maximálna dĺžka riadku 72 znakov.

Definície tried obaľujeme troma prázdnymi riadkami.

Definície metód obaľujeme dvoma prázdnymi riadkami.

Premenné nie je potrebné oddeľovať prázdnymi riadkami, ale je možné to urobiť ak chceme naznačiť logické rozdiely v skupinách premenných. Taktiež v kóde môžeme využiť prázdne riadky na oddelenie logických častí, čo zlepší zrozumiteľnosť kódu.

Import-y uprednostňujeme globálne oproti lokálnym. Mali by sa nachádzať na začiatku kódu.
Jeden import na jeden riadok.
V jazyku python nie je rozdiel v jednoduchých a dvojitých úvodzovkách pri uvádzaní reťazcov. My budeme používať pre uvádzanie reťazcov dvojité úvodzovky.

Príklad:

```
string = "This is a string"
```

Ak však reťazec obsahuje dvojité úvodzovky, uprednostňujeme použitie jednoduchých úvodzoviek oproti spätných lomítok.

Správne:

```
string = 'Peter said: "Hello mom".'
```

Nesprávne:

```
string = "Peter said: \"Hello mom\"."
```

Snažíme sa písať kód tak, aby sme nemuseli písať komentáre. Teda chceme aby bol kód samovysvetľujúci.
Ak však už komentáre píšeme, tak na samostatný riadok. Snažíme sa vyhnúť komentárom na rovnakých riadkoch s kódom.

Správne:

```
# x is radius
x = 3
```

Nesprávne:

```
x = 3    # x is radius
```

Dokumentačný reťazec, je taká štruktúra, ktorá nám umožní vygenerovať dokumentáciu pre projekt z reťazcov, ktoré v nej uvedieme.Dokumentačný reťazec by mal mať každý **modul, trieda, funkcia, metóda**.

Dokumentačné reťazce píšeme vždy s dvojitými úvodzovkami. Ak je jednoriadkový, vyzerá nasledovne:

```
"""Documentation string on one line"""
```

Ak je na viac riadkov, vyzerá nasledovne :

```
"""
This documentation string is not on one line.
It also has a second one.
"""
```

Príklad pre funkciu:

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

        Keyword arguments:
        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
        ...
```

Názvy premenných, funkcií, modulov a tried majú nasledovné pravidlá:

- funkcie, premenné a moduly: malé písmená, slová oddelené podčiarovníkom

  napr. -> certification_scheme_name

- triedy: veľké písmená na začiatku slov

  napr. -> CertificationScheme

Kód a názvy premenných vytvárame v anglickom jazyku. Taktiež komentáre.

HTML

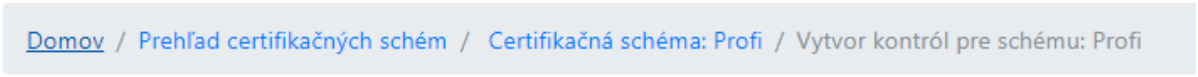
Každá HTML stránka bude obsahovať menu, ktoré sa nachádza v súbore deafult.html. Treba ho teda includnuť do vytváranej stránky, tým že na začiatok vytváraného html súboru vložíme: `{% extends "default.html" %}`

Stránka bude obsahovať navigáciu, pomocou ktorej sa používateľ dostane naspäť na predchádzajúcu stránku. Je teda potrebné vložiť blok breadcrumbs. Tento blok bude obsahovať minimálne domovskú stránku, na ktorú sa bude dať prekliknúť a vašu aktívnu stránku. Príklad:

```
{% block breadcrumbs %}
   <ol class="breadcrumb">
       <li class="breadcrumb-item"><a href="{% url 'index'%}">{%
trans "Homepage" %}</a></li>
       <li class="breadcrumb-item"><a href="{% url 'overview' %}">{%
trans "Certification scheme overview" %}</a></li>
       <li class="breadcrumb-item active"><a href="/scheme/{{
scheme.pk }}"> {% trans "Certification scheme" %}: {{
scheme.scheme_name }}</a></li>
       <li class="breadcrumb-item active" aria-current="page">{%
trans "Control create for scheme" %}: {{ scheme.scheme_name }}</li>
   </ol>
```

```
{% endblock %}
```

Na tomto príklade je možné vidieť, že ako prvá je domovská stránka, ďalej sa nachádza Prehľad schém, ďalej Vytváranie Controlu. Vytváranie Controlu je aktívna stránka, preto nie je možné na ňu kliknúť a nikam používateľa nepresmeruje. Ukážka kódu vytvorí:



Každá stránka by mala obsahovať nadpis s názvom danej stránky. Nadpis bude v tagu <h1>. Príklad je uvedený nižšie.
HTML kód musí byť správne odtabovaný a odriadkovaný. Vnútornejší tag je o jeden tab ďalej ako ten nad ním. To isté platí aj pre Django bloky. Obsah Django bloku je o jeden tab ďalej ako daný blok.
Príklad:

```
{% block content %}
   <h1>{% trans "Certification scheme" %}</h1>
   <table>
        <tr>
            <th>{% trans "Scheme's name" %}</th>
            <th>{% trans "Publisher" %}</th>
            <th>{% trans "Identifier" %}</th>
            <th>{% trans "Version" %}</th>
            <th>{% trans "Number of controls" %}</th>
        </tr>
   <a href="{% url 'index' %}">{% trans "Homepage" %}</a>
</form>

{% endblock %}
```

Linky nepíšeme v html súboroch statické ale vo formáte **{% url %}**

- Nesprávne

```
<a href="/control/{{ control.pk }}/control_objective">
      {% trans "Add control objective" %}
</a>
```

- Správne

```
<a href="{% url 'create-control-objective' control.pk %}">
      {% trans "Add control objective" %}
</a>
```

Štýl pomocou css nastavujeme v html cez css súbory a nie priamo v kóde html. Ak chceme vložiť odkaz na css súbor, vkladáme ho do bloku na to určeného. Tento blok sa vkladá pred blok **breadcrumbs**.
Príkaz {% load static %} načíta potrebné súbory pre každý html template. Následne je možné vložiť odkazy na vlastné css súbory. Css súbory je potrebné umiestniť v štruktúre projektu do priečinku **static** a v ňom do priečinku **css**.

Príklad:

```
{% block stylesheets %}
    {% load static %}
    <link rel="stylesheet" href="{% static 'css/overview.css' %}">
{% endblock %}
```

Podobne ako štýl css sa vkladajú aj vlastné javascript súbory. Nepíšeme javascript priamo do html (existujú výnimky), vkladáme ho do bloku na to určeného. Tento blok sa vkladá na koniec html súboru.
Javascript súbory je potrebné umiestniť v štruktúre programu priečinku **static** a v ňom do priečinku **javascript**.

Príklad:
```
{% block scripts %}
    <script src="{% static 'javascript/linking.js' %}"></script>
{% endblock %}
```

Django

Project v Django ma niekoľko častí. Na najvyššej úrovni sa delí na **test_app** a **my_app**.

- test_app obsahuje globálne súbory pre celý projekt
- my_app obsahuje súbory pre našu aplikáciu

Priečinok **my_app** obsahuje priečinky a súbory.

- Priečinok **fixtures** obsahuje **.json** súbory, pomocou ktorých napĺňame databázu cvičnými údajmi
- Priečinok **locale** obsahuje súbory využívané na preklad aplikácie do rôznych jazykov
- Priečinok **migrations** obsahuje automaticky generované migrácie databázy (vytvorenie tabuliek)
- Priečinok **templates** obsahuje **.html** súbory reprezentujúce formuláre aplikácie
- Súbor **forms.py** obsahuje definície tried, ktoré sa využívajú na generovanie obsahu html formulárov v prípadoch vytvárania alebo upravovania záznamov v databáze
- Súbor **models.py** obsahuje definície entít pre našu aplikáciu (podklad pre tabuľky databázy)
- Súbor **tests.py** obsahuje testy pre aplikáciu
- Súbor **urls.py** obsahuje definície url vzorov a ich preklad na controlleri formulárov
- Súbor **views.py** obsahuje definície funkcií, ktoré slúžia ako controlleri pre apllikáciu