Chotipat Pornavalai

PSIT2016

Lecture 4: PEP (Coding Standard)

PEP (Python Enhancement Proposal)

- A design document providing information to the Python community, or describing a new feature for Python or its processes or environment.
- **PEP 20 (The Zen of Python),** http://legacy.python.org/dev/peps/pep-0020/
- **PEP 8 (Style Guide for Python Code),** http://legacy.python.org/dev/peps/pep-0008/
- **PEP 257 (Docstring Conventions),** http://legacy.python.org/dev/peps/pep-0257/
- **Pylint**
- **Code Like a Pythonista: Idiomatic Python,** http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html

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.
- 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.
- 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.
- 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!

- **Discussion: http://stackoverflow.com/questions/228181/the-zen-of-python**

PEP 8: White Space 1

- 4 spaces per indentation level.
- No hard tabs.
- **Never** mix tabs and spaces.

  This is exactly what IDLE and the Emacs Python mode support. Other editors may also provide this support.
- One blank line between functions.
- Two blank lines between classes.

PEP 8: White Space 2

- Add a space after "," in dicts, lists, tuples, & argument lists, and after ":" in dicts, but not before.
- Put spaces around assignments & comparisons (except in argument lists).
- No spaces just inside parentheses or just before argument lists.
- No spaces just inside docstrings.

PEP 8: Naming 1

- **joined_lowe**r for functions, methods, attributes
- **joined_lower** or **ALL_CAPS** for constants
- **StudlyCaps** for classes
- **camelCase only** to conform to pre-existing conventions

PEP 8 Naming 2

```
def make_squares(key, value=0):
    """Return a dictionary and a list..."""
    d = {key: value}
    l = [key, value]
    return d, l
```

PEP 8 Long Lines & Continuations 1

- Keep lines below 80 characters in length.
- Use implied line continuation inside parentheses/brackets/braces:

```
def __init__(self, first, second, third,
             fourth, fifth, sixth):
    output = (first + second + third
              + fourth + fifth + sixth)
```

PEP 8 Long Lines & Continuations 2

- Use backslashes as a last resort:

```
VeryLong.left_hand_side \
    = even_longer.right_hand_side()
```

- Backslashes are fragile; they must end the line they're on. If you add a space after the backslash, it won't work any more. Also, they're ugly.

PEP 8 Long String 1

- Adjacent literal strings are concatenated by the parser:

```
>>> print 'o' 'n' "e"
one
```

- The spaces between literals are not required, but help with readability. Any type of quoting can be used:

```
>>> print 't' r'\/\/' """o"""
t\/\/o
```

- The string prefixed with a "r" is a "raw" string. Backslashes are not evaluated as escapes in raw strings. They're useful for regular expressions and Windows filesystem paths.

PEP 8 Long String 2

```
text = ('Long strings can be made up '
        'of several shorter strings.')
```

- The parentheses allow implicit line continuation.
- Multiline strings use triple quotes:

```
"""Triple
double
quotes"""

'''\
Triple
single
quotes\
'''
```

- In the last example above (triple single quotes), note how the backslashes are used to escape the newlines. This eliminates extra newlines, while keeping the text and quotes nicely left-justified. The backslashes must be at the end of their lines.

PEP 8 Compound Statement

- **Good:**

```
if foo == 'blah':
    do_something()
do_one()
do_two()
do_three()
```

- **Bad:**

```
if foo == 'blah': do_something()
do_one(); do_two(); do_three()
```

- Whitespace & indentations are useful visual indicators of the program flow. The indentation of the second "Good" line above shows the reader that something's going on, whereas the lack of indentation in "Bad" hides the "if" statement.
- Multiple statements on one line are a cardinal sin. In Python, *readability counts*.

PEP 8 Docstring & Comments 1

- Docstrings = **How to use** code
- Comments = **Why** (rationale) & **how code works**
- Docstrings explain **how** to use code, and are for the **users** of your code. Uses of docstrings:
- Explain the purpose of the function even if it seems obvious to you, because it might not be obvious to someone else later on.
- Describe the parameters expected, the return values, and any exceptions raised.
- If the method is tightly coupled with a single caller, make some mention of the caller (though be careful as the caller might change later).

PEP 8 Docstring & Comments 2

- Comments explain **why**, and are for the **maintainers** of your code. Examples include notes to yourself, like:

```
# !!! BUG: ...

# !!! FIX: This is a hack

# ??? Why is this here?
```

- Docstrings are useful in interactive use (help()) and for auto-documentation systems.
- False comments & docstrings are worse than none at all. So keep them up to date! When you make changes, make sure the comments & docstrings are consistent with the code, and don't contradict it.

PEP 257 Docstring Conventions: One-line Docstrings

- For example:

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    if _kos_root: return _kos_root
    ...
```

- Triple quotes are used even though the string fits on one line. This makes it easy to later expand it.
- The closing quotes are on the same line as the opening quotes. This looks better for one-liners.
- There's no blank line either before or after the docstring.
- The docstring is a phrase ending in a period. It prescribes the function or method's effect as a command ("Do this", "Return that"), not as a description; e.g. don't write "Returns the pathname …".

PEP 257 Docstring Conventions: One-line Docstrings

- The one-line docstring should NOT be a "signature" reiterating the function/method parameters. Don't do:

```
def function(a, b):
    """function(a, b) -> list"""
```

- Do

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

PEP 257 Docstring Conventions: Multi-line Docstrings

- Multi-line docstrings consist of a summary line just like a one-line docstring, followed by a blank line, followed by a more elaborate description.

```
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
    ...
```

Pylint

- http://www.pylint.org/
- http://pylint-messages.wikidot.com/
- http://docs.pylint.org/features.html