EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE

Python Coding Style

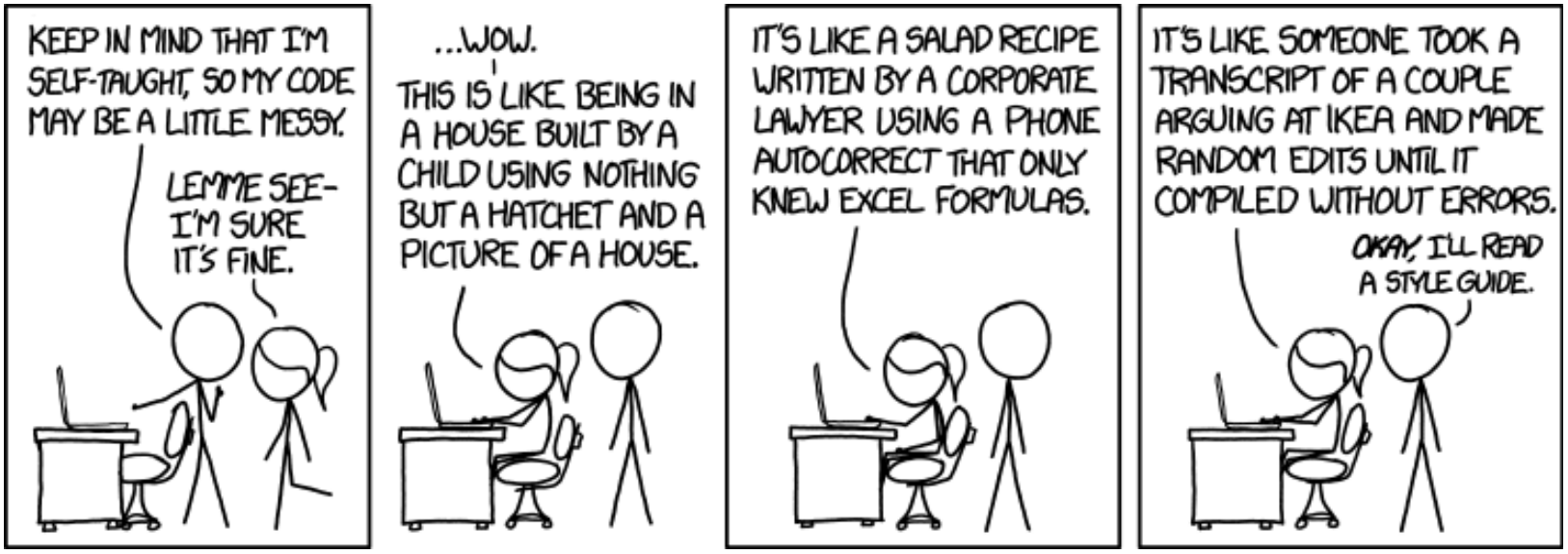
PEP8

Motivation

- Code is more often read than written
- Readability is one of pythons main strengths
- Style guides help enforce for **consistency** within packages, modules, projects

- Main concepts
  - One statement per line
  - Explicit is better than implicit
  - Return values in consistent manner
  - If the implementation is hard to explain, it’s probably a bad idea
  - **Readability counts**

- Examples
  - http://docs.python-guide.org/en/latest/writing/style/
  - http://artifex.org/~hblanks/talks/2011/pep20_by_example.pdf



https://xkcd.com/1513/

PEP 8

- Official python style guide
- https://www.python.org/dev/peps/pep-0008

- 4 spaces per indentation level
- 79 characters per line (longer makes code hard to read)
  - Long lines can be wrappd by using Python's implied line continuation inside parentheses, brackets and braces or ‘\’
- Imports should be at top of file and one import per row
- Absolute imports are recommended and wildcard imports avoided
  - from mypkg import sibling
  - import mypkg as m
- Comments
  - should be up-to-date
  - should be complete sentences
  - should be in English

Docstrings

- A documentation string should be written for all public
  - modules, functions, classes, methods
- Private functions do not need a docstring, but should have a comment above the **`def`** that says what the function does
- The docstring should be the first statement of the module, function, class or method definition

```
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
```

https://www.python.org/dev/peps/pep-0257/

Variable names

- Possibly indistinguishable letters should be avoided as single character name
  - **`I, l, O`**
- Package and module names:
  - short, all lowercase
  - underscores can be used in modules if it improves readability
- Class names
  - CapWords convention
- Functions
  - names: lowercase, words separated by underscore
  - arguments: lowercase
- Constants
  - should be defined at module level
  - all capital letters
- Append trailing _ to avoid collision with built-in names (e.g. def_)
- Ignored variable: **`bn, __, ext = filename.rpartition('.')`**

Function arguments

- Positional arguments
  - mandatory
  - no default values
  - examples:

```
send(message, recipient)
```

- Keyword arguments
  - optional
  - have default values
  - examples

```
send(message, to, cc=None, bcc=None)
```

- Arbitrary argument list (`*args`) and arbitrary keyword argument dictionary (`**kwargs`)

Function should be easy to read and easy to change

http://docs.python-guide.org/en/latest/writing/style/

White spaces

- Indentations should be either whitespace OR tab (ideally 4 spaces)
- Only one space around assignment: `a = 1`
- No trailing whitespaces anywhere

- White spaces should be avoided
  - immediately inside parentheses
  - immediately before a comma, semicolon or colon
  - immediately before the open parenthesis that start argument list, indexing or slicing
  - around the '=' of keyword arguments **`send(to=<name>)`**

- Always surround the following operators with whitespace
  - assignment (=) and augmented assignment (+= , -= etc.)
  - comparisons (== , < , > , != , <> , <= , >= , in , not in , is , is not)
  - Booleans (and , or , not).

https://www.python.org/dev/peps/pep-0008/#id26

Working with variables

- Comparisons to singletons like None should always be done with **is** or **is not** , never the equality operators**.**

```
if variable:
      do sth
if variable is None:
    do sth
```

- Check if element in list

```
if 'el' in ['ab', 'cd', 'ef']:
    do sth
```

- Accessing elements in dictionary

```
d = {'world': 'hello'}
print d.get('world', 'bye')   # prints 'hello'
print d.get('thingy', 'bye')  # prints 'bye'
# or
if 'world' in d:
    print d['world']
```

Python comprehensions

- List comprehension

  Single loop

  ```
  [x ** 2 for x in range(10)]
  ```

  Nested loop

  ```
  [el for il in ol for el in il]
  ```

- Dictionary comprehensions

  ```
  {k: v.lower() for k, v in prev_dict}
  ```

- Lambda : create small anonymous functions
  - Do not use to bind to identifier, better use def statement
  - Useful for map, filter, reduce, and getting items using itemgetter

  ```
  c = map(lambda x: x + 2, a)
  d = reduce(lambda x, y: x + y, [47, 11, 42, 13])
  e = filter(lambda x: x > 4, a)
  ```

List comprehensions

```
# Filter elements greater than 4
a = [3, 4, 5]

Bad example:
b = []
for i in a:
    if i > 4:
        b.append(i)

Good examples:
b = [i for i in a if i > 4]
or
b = filter(lambda x: x > 4, a)
```

IPython Magics

Making some simple tasks even simpler

What is it?

- Fast access to commonly used tasks
- Simple integration of code from other languages
- Capture and load external files and output

- **Line magics**
  - prefixed with %
  - like OS command line calls
  - work on one line
- **Cell magics**
  - prefixed with %%
  - takes lines below as argument

- System calls
  - prefixed with !
  - output can be captured as python variable

- Info about object: <object>? or for more details <object>??

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Use other languages in a cell

In [12]:
```
%%bash
echo "HELLO WORLD!"
```

```
HELLO WORLD!
```

In [4]:
```
%%HTML
<button type="button" id="loading-example-btn" data-loading-text="Loading..." class="btn
 btn-primary">
  Loading state
</button>
```

In [5]:
```
%%javascript
console.log('hello world');
```

```
<IPython.core.display.Javascript at 0x105f51810>
```

Also available

- R, Perl, Ruby
- Latex, …

Run arbitrary Unix command

! allows to run an arbitrary command from within notebook

```
In [17]: !ls
```

```
CGM_DEMO.ipynb              LICENSE                 myfile.txt
IPythonOverview.ipynb       README.md               zenofpython.py
IPythonOverview.slides.html ca_website.png          zenofpython.pyc
```

```
In [18]: !cat CGM_DEMO.ipynb | sort | uniq -c | sort -r | head
```

```
     41     {
     41      "metadata": {},
     40     },
     21      ]
     21      "source": [
     21      "cell_type": "markdown",
     20      ],
```

Some have built-in direct magic:
%ls
%cd

Time your functions and code

In [10]:
```
%timeit np.linalg.eigvals(np.random.rand(100,100))
```

```
100 loops, best of 3: 11.2 ms per loop
```

In [11]:
```
%%timeit
a = np.random.rand(100, 100)
np.linalg.eigvals(a)
```

```
100 loops, best of 3: 11 ms per loop
```

Additional Kernel extensions

Some magics extensions need explicit installation

- required for R-magic
- pep8
- autoreload modules after changes
- https://github.com/ipython/ipython/wiki/Extensions-Index

- Installed with %install_ext
- Loaded with %load_ext

- More about the different magics and how to use them

%lsmagic and

http://ipython.readthedocs.io/en/stable/interactive/magics.html

Export and convert

- Notebooks can be exported as
  - python files
  - HTML sites
  - PDF documents
  - Markdown

- Export from notebook
  - File ➔ Download As ➔ Select your format of choice

- Convert using nbconvert
  - jupyter nbconvert --to <output format> <input notebook>