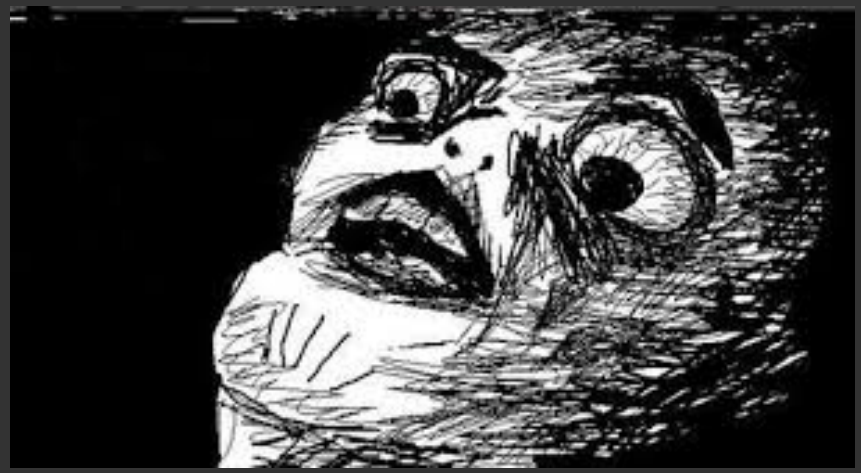
Writing clean code

By Thomas Martin

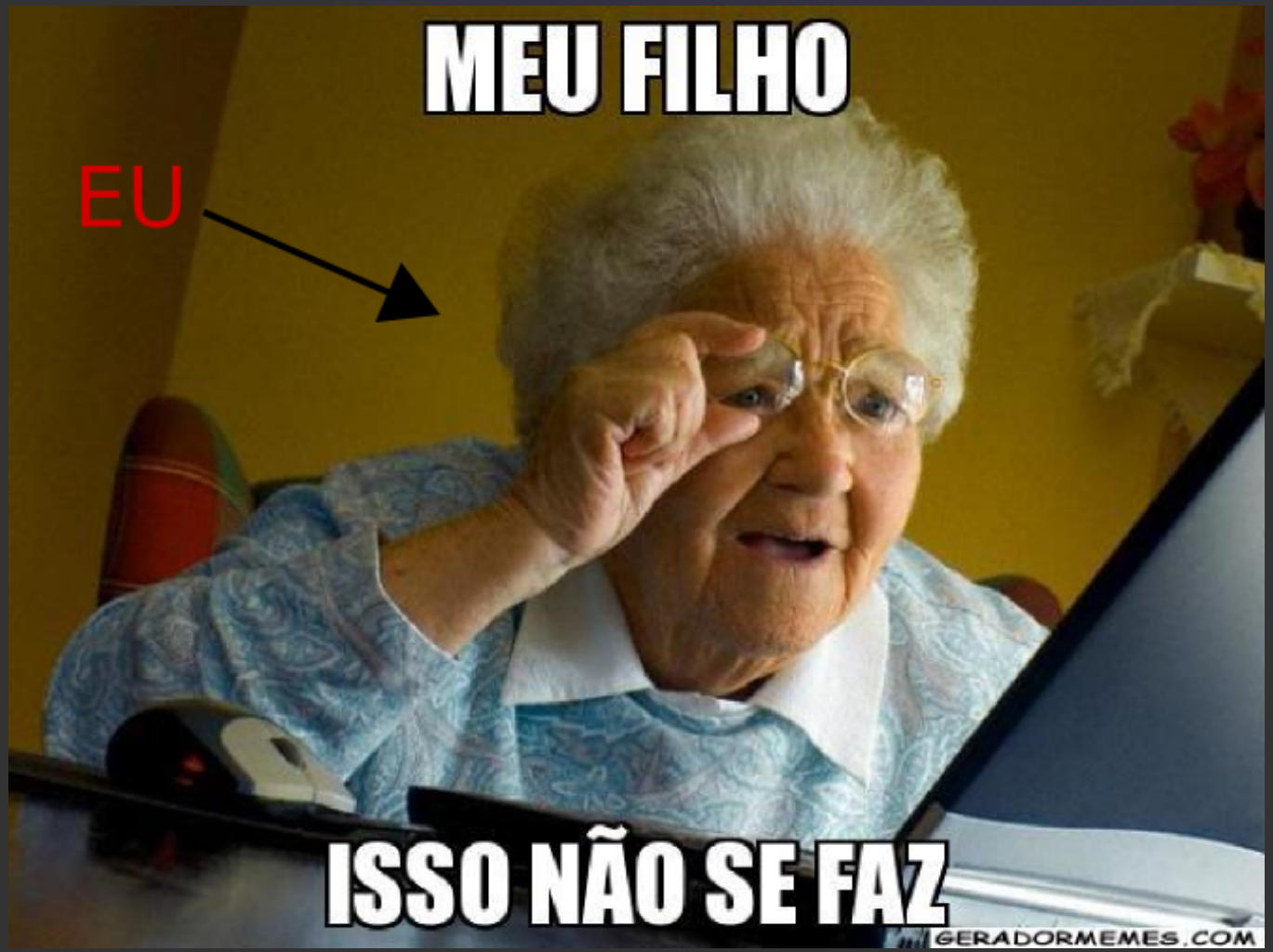
IAG Python Bootcamp 2015

me: "Hey Humberto, could you give me your python code to visualise this data?"

```
from pyratobil import *; import matplotlib as matpl
class HoveErmoLIler():def __init__(self,networkstat
"""I CREATED THIS CLASS WHICH DO A LOT OF THINGS
   self.net=  __net__
    self.__paARTAgetdara__=self.net.pPartef(opa)
        self."Bywind":'1H',    "Byvar":'1H', # This
 'pos':'Lon','attributnumber2':2} # this is a a dic
def set_p_A___ra___(self,parameter,Value): ; self.p
        print(str(parameter)+' has been set to but
-> '+ str(value))
```



```
from pyratobil import *; import matplotlib as matpl
class HoveErmoLIler():def __init__(self,networkstat
"""I CREATED THIS CLASS WHICH DO A LOT OF THINGS
   self.net=  __net__
    self.__paARTAgetdara__=self.net.pPartef(opa)
```


MEU FILHO
EU
ISSO NÃO SE FAZ
GERADORMEMES.COM

Why clean code?

"Code is read much more often than it is written!"
- *Guido*
or in other word, **"Readability counts"**

Clean code will make you and your colleagues happy and productive!

Why clean code?

In 1999, when Guido submitted a funding proposal for Programming for Everybody (DARPA), in which he further defined his goals for Python as:

1. An easy and intuitive language just as powerful as major competitors
2. Open source, so anyone can contribute to its development
3. Code that is as understandable as plain English
4. Suitability for everyday tasks, allowing for short development times

source:wikipedia

Why clean code?

In 1999, when Guildo submitted a funding proposal for Programming for Everybody (DARPA), in which he further defined his goals for Python as:

1. An easy and intuitive language just as powerful as major competitors
2. Open source, so anyone can contribute to its development
3. Code that is as understandable as plain English
4. Suitability for everyday tasks, allowing for short development times

source:wikipedia

Readability is at the very foundation of the Python objectives

Outline

- Zen of Python
- Indentation
- Spacing
- Maximum line length
- Blank lines
- Import modules
- Comments
- Naming Conventions

Conventions and tips from PEP8 and PEP20
PEP = Python Enhancement Proposal

The Zen of Python

```
import this
```

- 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.
- ...

Tim Peters

Indentation

- 4 spaces indentation is preferred to tab
- Python 3 disallows mixing the use of tabs and spaces for indentation.

Indentation

- Aligned with opening delimiter.

Bad

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

Good

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

Indentation

Add more indentation to distinguish from the rest

Bad

```
def long_function_name(
    var_one, var_two, var_three,
    var_four, var_five, var_six
    var_seven, var_height):
    print(var_one)
```

Good

```
def long_function_name(
        var_one, var_two, var_three,
        var_four, var_five, var_six
        var_seven, var_height):
    print(var_one)
```

Spacing

- Put a single space **after** every comma
- put a single space **before and after** any operators

Bad

```
var=0
list=[1,2,3,4,5]
```

Good

```
var = 0
list = [1, 2, 3, 4, 5]
```

Spacing

- multiple level operator: space around lowest priority operator
- Use your own judgement ... Yes you can !

Bad

```
i=i+1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Good

```
i = i + 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Spacing

- Do not use space for arguments

Bad

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

Good

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

Maximum Length

- No more than 79 characters

Bad

```
var=['this','ligne','is','too','long','to','be','ea
```

Good

```
var = ['better', 'to',
       'make', 'several', 'lines',
       'when', 'it', 'is', 'too', long']
```

Blanck lines

- Two blank lines between class and top-level function definitions
- One blank line between methods in a class

Bad

```
def function():
    pass
class Class(object):
    def __init__(self, arg):
        self.arg = arg
    def multiply(self, var):
        return self.arg * var
```

Blanck lines

Good

```
def function():
    pass


class MyClass(object):

    def __init__(self, arg):
        self.arg = arg

    def multiply(self, var):
        return self.arg * var
```

Import modules

- Always at the top of the file
- import should be in separate lines

Bad

```
import numpy, pandas, matplotlib
```

Good

```
import numpy
import pandas
import matplotlib
```

Import modules

- Explicit import is better
- If the object name already exist it will overwrite it

Bad

```
from my_module import *
```

Good

```
from my_module import func1, func2, func3
```

Comments

- Contradictory comments are the worst
- Keep them up to date
- You should write your comments in English

Block comments

```
foo()

# This is a block comments
#
# Each paragraph should be separated by a "#"

bar()
```

Inline comments

```
zinho()  # This is an inline comment
bar()  # Separated by 2 spaces from the code
barzinho()  # Use inline comment only if necessary!
```

Documentation string

- See PEP 257 to write "docstrings"

```
def function():
    """
    Any function that's longer than a few lines
    should have a docstring that explains:
      1. Inputs
      2. Output
    Use triple-quote to make it
    """

    def __init__(self):
        pass
```

Naming Conventions

- Variables give code meaning and context
- Variables names should be as explicit as possible

Bad

```
list_of_number = [73, 68, 78, 89]
list_of_string = ['Julio', 'Malu', 'Elvis', 'Bruno
```

Good

```
age_tutor = [73, 68, 78, 89]
name_tutor  = ['Julio', 'Malu', 'Elvis', 'Bruno']
```

Naming Conventions

- **Package and Module**: All-lowercase names
  - **Class**: CapWords convention
  - **Golbal variable, Method and Attribute**: lowercase separated by underscore
  - **Constant**: Capital letters

Good

```
import matplotlib

CONSTANT = 2

class MyClass(MyOtherClass):

    def my_methods(self, var):
        return var * CONSTANT
```

Naming Conventions

- `_leading`: *Internal use indicator*
- `trailing_` *used to avoid conflict with Python keyword. class HomemElegante(nome,* `class_`*= True)*
- `__double_leading`: *Strictly internal use. Invokes name mangling (inside class FooBar,* `__boo` *becomes* `_FooBar__boo`*).*
- `__double_leading_and_trailing__` : *"magic" objects. E.g.* `__init__`

Explicit Code

Bad

```
def make_complex(*args):
    x, y = args
    return dict(**locals())
```

Good

```
def make_complex(x, y):
    return {'x': x, 'y': y}
```

One statement by line

Bad

```
print 'one'; print 'two'

if x == 1: print 'one'
```

Good

```
print 'one'
print 'two'

if x == 1:
    print 'one'
```

Tips to keep you code clean

- Write testing code
- Use wisely Exception
- Their is a lot more tips to write clean code, look for them!

References

PEP8 by Guido van Rossum, Barry Warsaw, Nick Coghlan
https://www.python.org/dev/peps/pep-0008/

Writing clean python code by Erik Romijn
https://speakerdeck.com/erik/writing-clean-python-code

A Quick Primer on Writing Readable Python Code for New Developers by Hartley Brody
https://blog.hartleybrody.com/python-style-guide/