Python Tutorial Series
Your codes pretty and healthy

PhD Bruno C. Quint
bquint@ctio.noao.edu
Resident Astronomer at SOAR Telescope

What does make scientific software usable for long time?

What does make scientific software usable for long time?

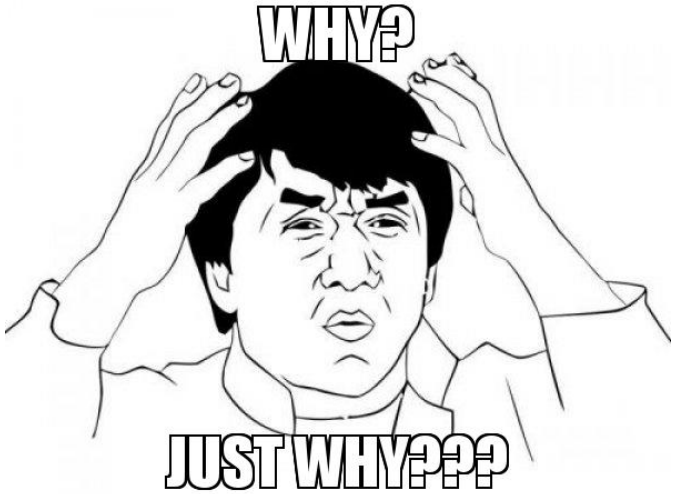
1. Code documentation
2. Final user documentation
3. Friendly interface
4. Version coherence
5. Code readability
6. Error handling
7. Version Control

What does make scientific software usable for long time?

1. Code documentation
2. Final user documentation
3. Friendly interface
4. Version coherence
5. Code readability
6. Error handling
7. Version Control
8. Scientific Impact

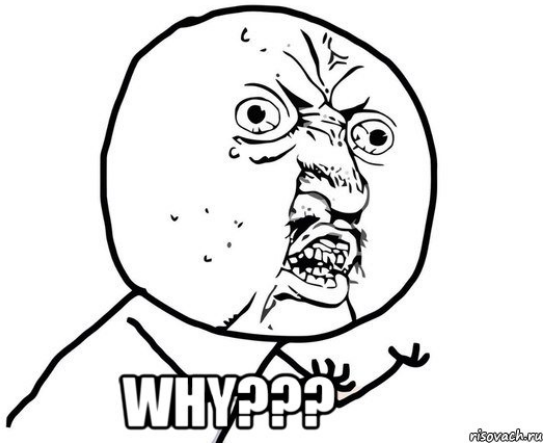
Why do we keep NOT doing these?

Documentation



Clear Code

Error & Version Track



The Zen of Python

```
>>> import this
```

The Zen of Python

```
>>> import this
```

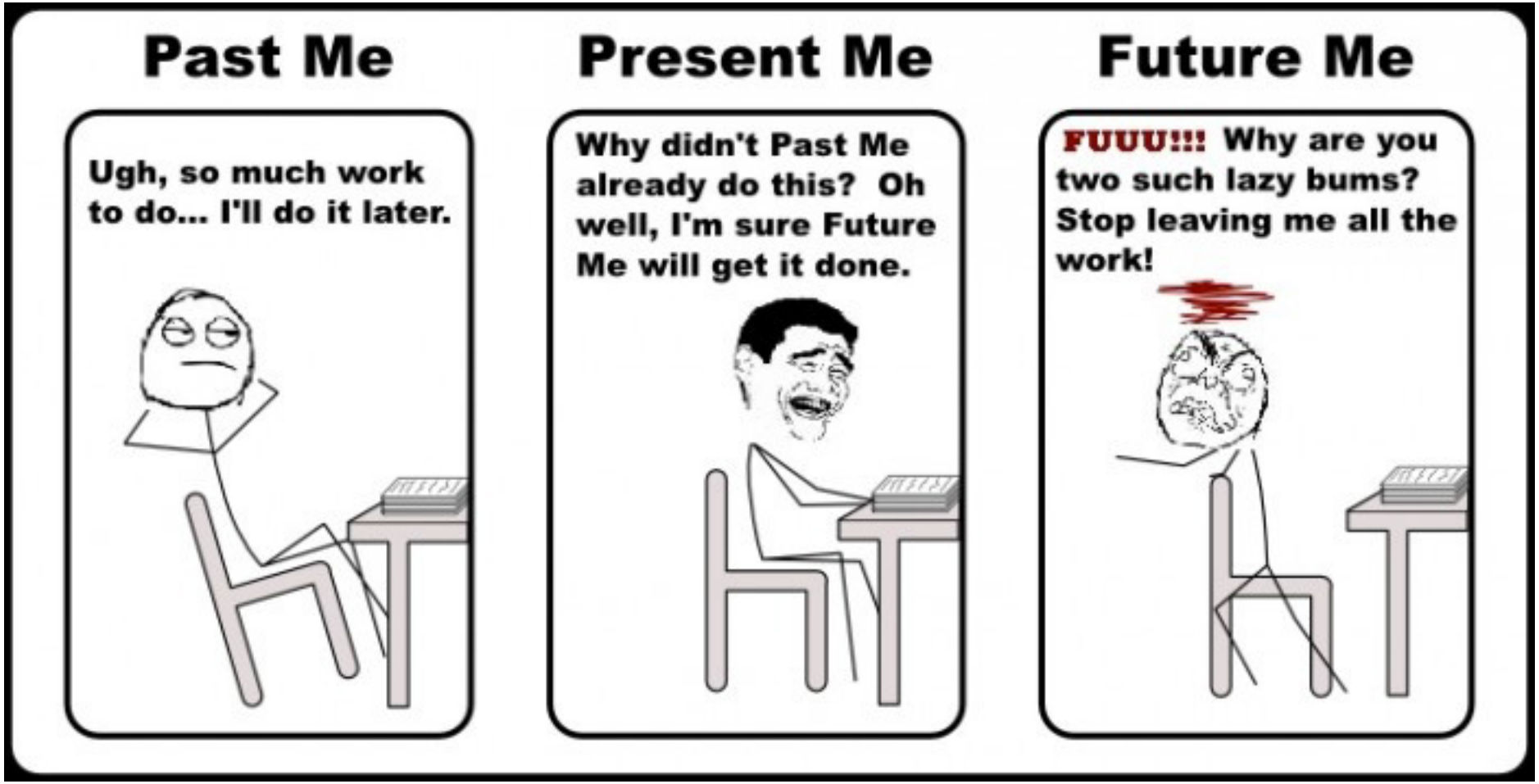
```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special
enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity,
refuse the temptation to guess.
...
```

The Zen of Python

```
>>> import this
```

```
...
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!
```

Do the docstring for your life!



Do the docstring for your life!

Do the docstring for your life!

Add documentation to your files

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
My Sample Code

This is such a simple documentation! The only thing I
have to do is to add three single quotes (') or double
quotes (") together to start a docstring block! Then
just write down in a few words what you want to do in
your file.

by J. Bond - Feb 31, 202X
"""
```

Do the docstring for your life!

Add documentation to your files

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""”The docstring can also be just one line"""
```

Do the docstring for your life!

Add documentation to methods/classes

```
def say_hello(name):
        """
        This method simply says hello to the person
        whose name is given as parameter.
        """
        print("Hello, {0}!".format(name))
```

Do the docstring for your life!

Documentation styles

PEP 257

```
def say_hello(name):
        """
        This method simply says hello to the person
        whose name is given as parameter.

        :param name: name of the person that to be greeted.
        :type name: string
        """
        print("Hello, {0}!".format(name))
```

Do the docstring for your life!

Documentation styles

[Sphinx Napoleon]

```
def say_hello(name):
    """
    This method simply says hello to the person
    whose name is given as parameter.

    Parameters
        name (string): The name of the person that
                       will be greeted.
    """
    print("Hello, {0}!".format(name))
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Indentation

```
for i in range(10):
    x = i * 2
    print x
```

Name Notation

```
number_of_states = 5
def sum_integers(x, y):
    return (x, y)
```

```
THIS_IS_A_GLOBAL_VARIABLE = None
class PhaseMap:
    def __init__(self, **kwargs):
        self.input_filename = kwargs['filename']
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Code Layout

```
def and_a_new_method_after_two_spaces():
    """
    Of course, you remember the docstring. We know that it is important but you want to
    """
    return "I am a method with a laaaarge docstring"


class Person:

    def __init__(self, an_argument, and_other_argument_, and_another_argument, fourth_ar
        """This person has several arguments... """
        self.x = an_argument
        self.y = and_other_argument_
        self.w = and_another_argument
        self.z = fourth_argument

    def single_space_between_methods_within_a_class(self):
        """Docstrings. Always"""
        return "Oh! Yeah!" + self.x
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Code Layout – Wrap text within 80 characters (72 for docstrings).

```
you_can_have_a_variable = "like this"


def and_a_new_method_after_two_spaces():
    """
    Of course, you remember the docstring. We know that it is important but you
    want to avoid more than 80 characters wide.
    """
    return "I am a method with a laaaarge docstring"
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Code Layout – Wrap text within 80 characters (72 for docstrings).

```
you_can_have_a_variable = "like this"


def and_a_new_method_after_two_spaces():
    """
    Of course, you remember the docstring. We know that it is important but you
    want to avoid more than 80 characters wide.
    """
    return "I am a method with a laaaarge docstring"
```

```
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Give some space and use self-explanatory names

```
import numpy as np

def SnellLaw(t,n):
    return np.deg(np.asin(np.sin(np.rad(t))/n))
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Give some space and use self-explanatory names

```
import numpy as np

def snell_law(incident_angle, refractive_index):
    """Calculates the snell's law."""

    theta = np.rad(incident_angle)
    n     = refractive_index

    temp = np.sin(theta) / n
    temp = np.asin(temp)
    output_angle = np.deg(temp)

    return output_angle
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Give some space and use self-explanatory names

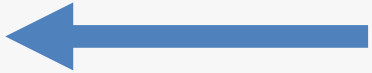
```
import numpy as np

def snell_law(incident_angle, refractive_index):
    """Calculates the snell's law."""

    theta = np.rad(incident_angle)
    n     = refractive_index

    temp = np.sin(theta) / n
    temp = np.asin(temp)
    output_angle = np.deg(temp)

    return output_angle
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Give some space and use self-explanatory names

```
import numpy as np

def snell_law(incident_angle, refractive_index):
    """Calculates the snell's law."""

    theta = np.rad(incident_angle)
    n = refractive_index

    temp = np.sin(theta) / n
    temp = np.asin(temp)
    output_angle = np.deg(temp)

    return output_angle
```



Why write if others can't read?

Code Style: PEP-8 and/or Google Style

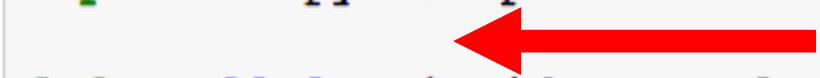
Give some space and use self-explanatory names

```
import numpy as np

def snell_law(incident_angle, refractive_index):
    """Calculates the snell's law."""

    theta = np.rad(incident_angle)
    n = refractive_index

    temp = np.sin(theta) / n
    temp = np.asin(temp)
    output_angle = np.deg(temp)

    return output_angle
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Give some space and use self-explanatory names



```
import numpy as np

def snell_law(incident_angle, refractive_index):
    """Calculates the snell's law."""

    theta = np.rad(incident_angle)
    n = refractive_index

    temp = np.sin(theta) / n
    temp = np.asin(temp)
    output_angle = np.deg(temp)

    return output_angle
```

Why write if others can't read?

Code Style: PEP-8 and/or Google Style

Give some space and use self-explanatory names

```
import numpy as np


def snell_law(incident_angle, refractive_index):
    """Calculates the snell's law."""

    theta = np.rad(incident_angle)
    n = refractive_index

    temp = np.sin(theta) / n
    temp = np.asin(temp)
    output_angle = np.deg(temp)

    return output_angle
```

Questions?