

Software Design for Data Science

Style & Documentation

*Melissa Winstanley
University of Washington
February 27, 2025*



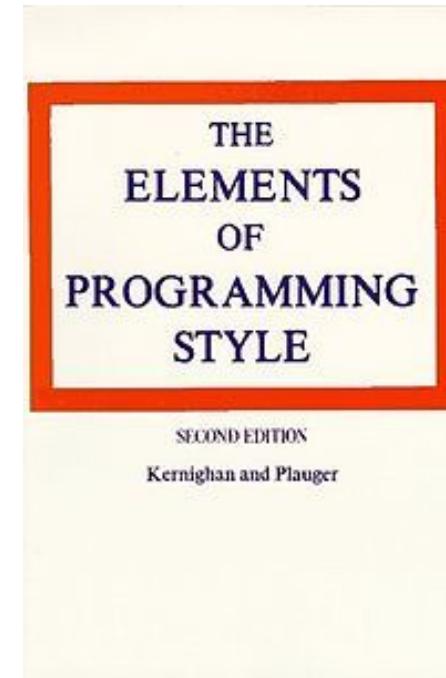
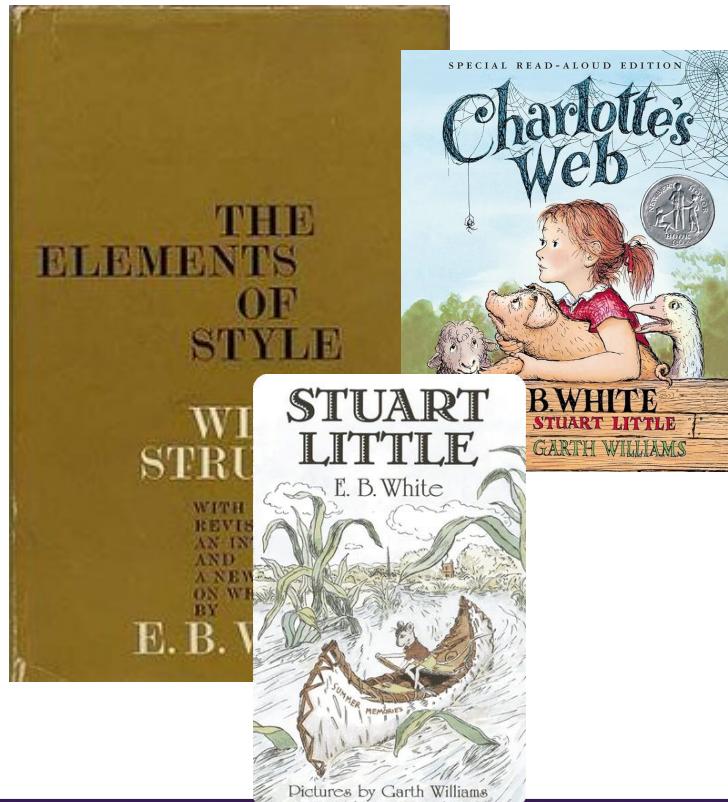
Style and documentation...



BUT! Everyone loves animals



How to write vs how to write effectively



SURF-ing!

- Why is it important that people can read your code? (SURF)
 - Sustainable
 - New version of Python (4.X)
 - Understandable
 - Is it doing what you claim?
 - Reusable
 - Can it be incorporated into a larger project?
 - Fixable

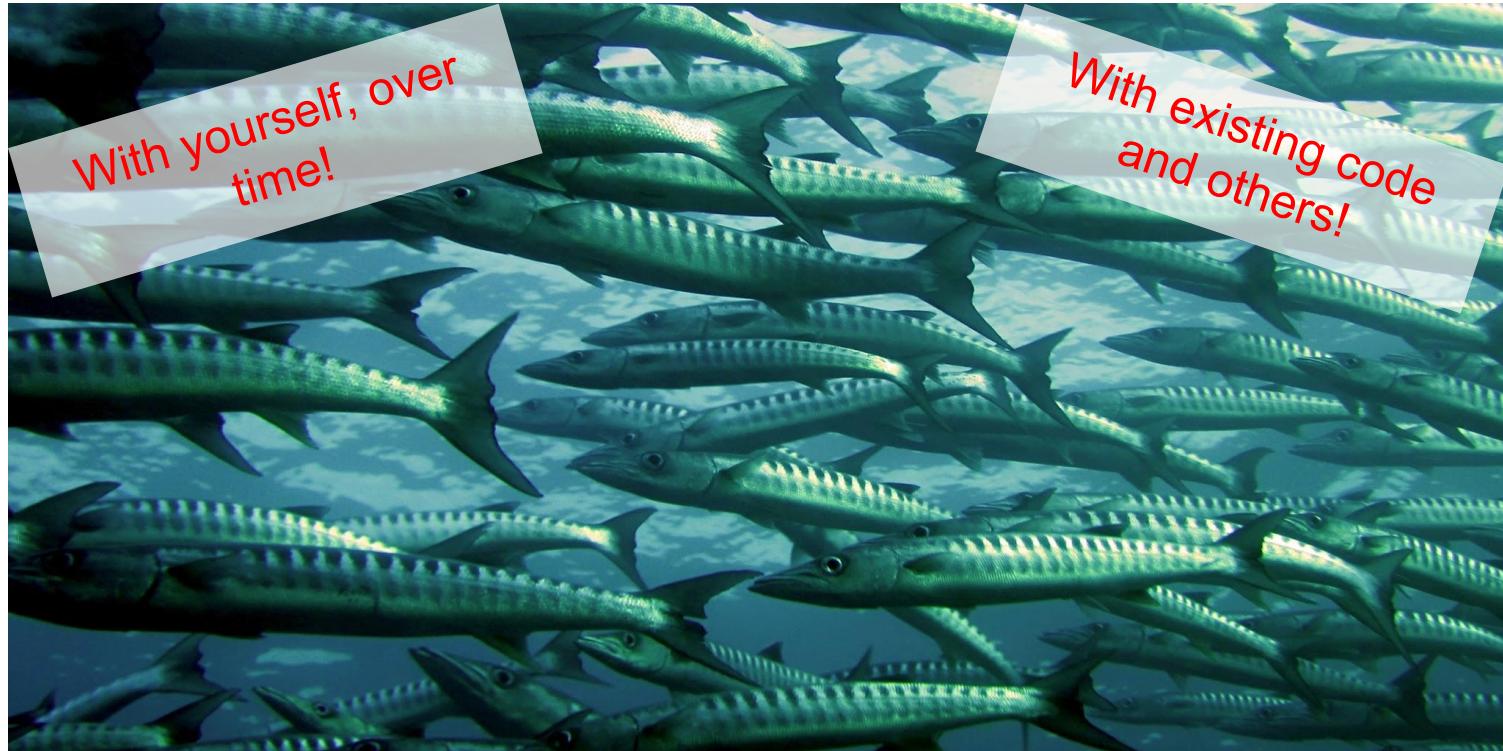


Style

Ultimately, it doesn't change how things work, just how approachable it is



Most important rule: consistency



Many different styles...

- PEP8 (most common)
 - “Python Enhancement Proposal #8”
- Google Python Style Guide
 - “More complete” PEP8



Use color! Use an editor!

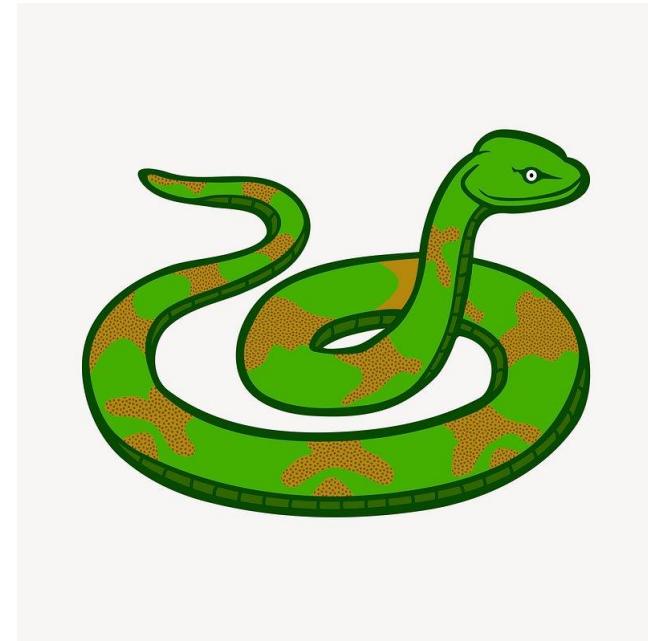
```
1  from statistics import mean
2  import numpy.random as nprnd
3  from statistics import stdev
4  def MyFuNcTiOn(ARGUMENT):
5      m = mean(ARGUMENT)
6      s = stdev(ARGUMENT)
7      gt3sd = 0
8      lt3sd = 0
9      for m in ARGUMENT:
10          if m > m + (s * 2):
11              gt3sd += 1
12          elif m < m - (s * 2):
13              lt3sd += 1
14      return(gt3sd, lt3sd)
15  def AnotherFunction(anumber = 1000, anothernumber = 1000):
16      l = nprnd.randint(anothernumber, size = anumber)
17      return(MyFuNcTiOn(l))
18  a,b=AnotherFunction(anumber = 1000, anothernumber = 1000)
19  print('found %d random values greather than 2 * sd and %d less than 2 * sd' % (a, b))
```



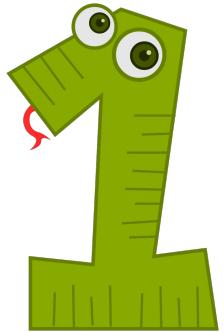
Tools for style checking

- Pylint
 - Extends from the *lint* tool for C

```
conda install pylint
pylint filename.py
pylint directory_name
```
- Also
 - pycodestyle
 - flake8



Exercise Part 1: Get Set Up



Install pylint

```
conda install pylint
```

Clone the demo repository (linked from the syllabus)

```
git clone  
https://github.com/UWDATA515/style_documentation_example.git
```

Enter the repository directory

```
cd style_documentation_example
```

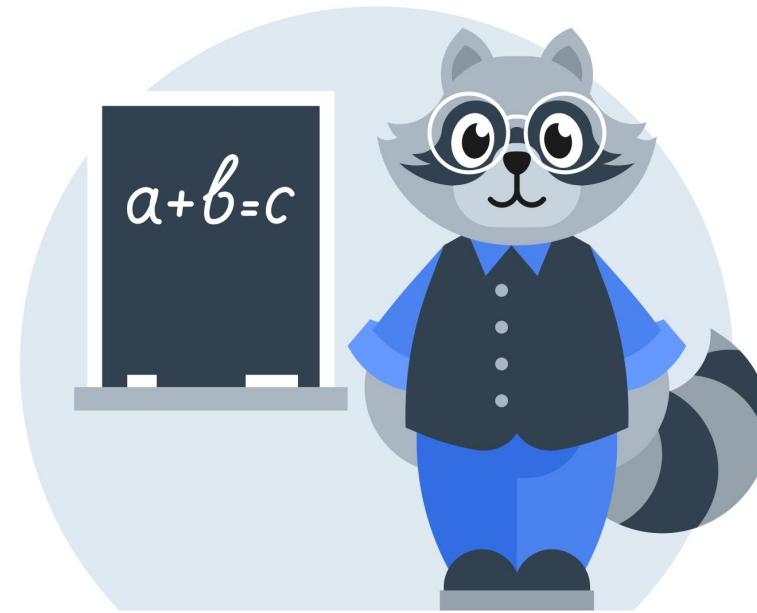
Run pylint

```
pylint style_demo.py
```

Don't try to improve it yet!

The core pieces of style

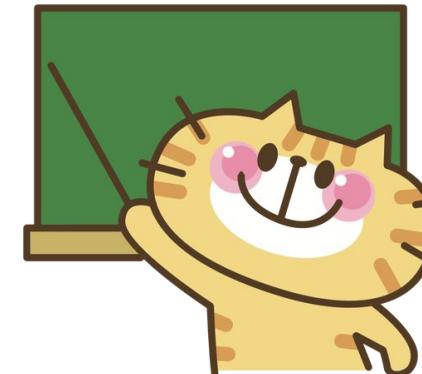
- Spacing/indentation
- Line length
- Imports
- Code constructions
- Naming



Made by FREE-VECTORS.NET

Spacing

- Indentation is so important in Python!
- Four spaces per indentation (not tabs!)
- Line wrapping - let's see!
- Set your editor to convert tabs into spaces!



YES! indentation

```
4 # Aligned with opening delimiter.  
5 foo = long_function_name(var_one, var_two,  
6 |var_three, var_four)  
7  
8 # More indentation included to distinguish this from the rest.  
9 def long_function_name(  
10 |var_one, var_two, var_three,  
11 |var_four):  
12     print(var_one) --  
13             39 # Add a comment, which will provide some distinction in editors  
14             40 # supporting syntax highlighting.  
15             41 if (this_is_one_thing and  
16             |that_is_another_thing):  
17                 42 # Since both conditions are true, we can frobnicate.  
18                     43 do_something()  
19  
20             44  
21             45  
22             46 # Add some extra indentation on the conditional continuation line.  
23             47 if (this_is_one_thing  
24             |and that_is_another_thing):  
25                 48 do_something()  
26  
27             49
```



NO! Indentation

```
27 ▼ def long_function_name(  
28     var_one, var_two, var_three,  
29     var_four):  
30     print(var_one)  
31
```

```
33  
34     if (this_is_one_thing and  
35         that_is_another_thing):  
36         do_something()  
37
```

```
23     foo = long_function_name(var_one, var_two,  
24     var_three, var_four)  
25
```



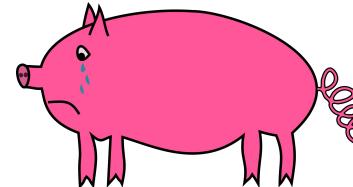
Line spacing

- Two blank lines around top-level functions
- Two blank lines around classes
- One blank line between functions in a class
- One blank line between logical groups in a function (*sparingly*)
- Extra blank lines between groups of groups of related functions (*why are they in the same file?*)



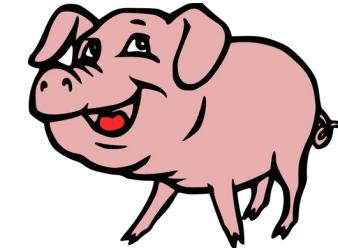
Whitespace

NO



```
79  spam( ham[ 1 ], { eggs: 2 } )
```

Brackets & braces



```
78  spam(ham[1], {eggs: 2})
```

```
83  if x == 4 : print x , y ; x , y = y , x      :: ,
```

```
82  if x == 4: print x, y; x, y = y, x
```

```
93  i=i+1
94  submitted +=1
95  x = x * 2 - 1
96  hypot2 = x * x + y * y
97  c = (a + b) * (a - b)
```

Mathy stuff

```
86  i = i + 1
87  submitted += 1
88  x = x*x - 1
89  hypot2 = x*x + y*y
90  c = (a+b) * (a-b)
```

```
104
105  def complex(real, imag = 0.0):
106      return magic(r = real, i = imag)
107
```

Function parameters

```
101  def complex(real, imag=0.0):
102      return magic(r=real, i=imag)
103
```

- Trailing spaces at the end of a line

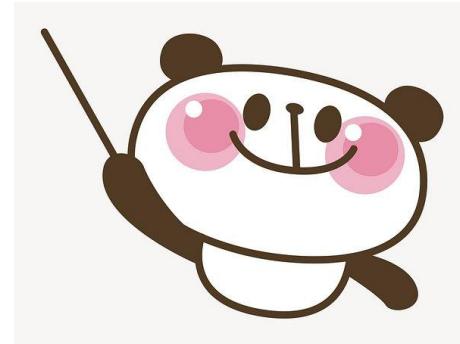
Line length



- Coding lines? Keep it to 79 characters
 - Most editors can show you the line position
- Comments & doc strings?
 - 72 characters
- Why? My monitor is big!
 - Open two files side by side? History?
 - Some teams choose to use a different max
 - Python core library is 79/72

Imports

At the top of the file!
(after any comments)



```
1 import os           ← First: standard library imports
2 import sys          ← Blank line
3 import pandas as pd ← Second: third party imports
4 import numpy as np  ← Blank line
5 from phylodist.constants import TAXONOMY_HIERARCHY, PHYLODIST_HEADER
6
7
```

Last: local imports

Code constructions

Avoid compound statements!



Always include returns!

```
128 def foo(x):  
129     if x >= 0:  
130         return math.sqrt(x)  
131  
132 def bar(x):  
133     if x < 0:  
134         return  
135     return math.sqrt(x)  
136
```

```
139 def foo(x):
140     if x >= 0:
141         return math.sqrt(x)
142     else:
143         return None
144
145 def bar(x):
146     if x < 0:
147         return None
148     return math.sqrt(x)
149
```

Naming

- Choose variable names that won't be confused.

```
1 from statistics import mean
2 import numpy.random as nprnd
3 from statistics import stdev
4 def MyFuNcTiOn(ARGUMENT):
5     m = mean(ARGUMENT)
6     s = stdev(ARGUMENT)
7     gt3sd = 0
8     lt3sd = 0
9     for m in ARGUMENT:
10         if m > m + (s * 2):
11             gt3sd += 1
12         elif m < m - (s * 2):
13             lt3sd += 1
14     return(gt3sd,lt3sd)
15 def AnotherFunction(anumber, anothernumber):
16     l = nprnd.randint(anothernumber, size = anumber)
17     return(MyFuNcTiOn(l))
18 a,b=AnotherFunction(anumber = 1000, anothernumber = 1000)
19 print('found %d random values greather than 2 * sd and %d less than 2 * sd' % (a, b))
```

- How you name functions, classes, and variables can have a huge impact on readability



Naming



Avoid the following variable names:

- Lower case L (l)
- Upper case O (O)
- Upper case I (I)

These are unacceptable, terrible, and awful.
Why?

- Can be confused with 1 and 0 in some fonts
- Can be confused with each other (i.e. I and l)

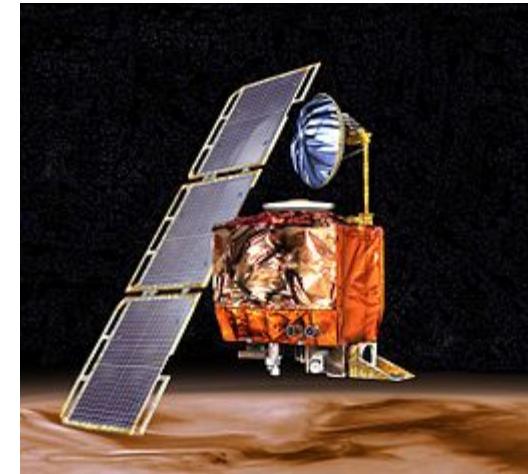
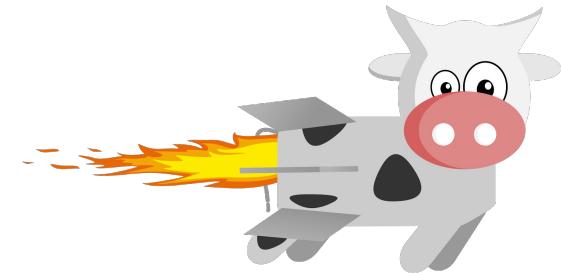
Naming

Functions

- Where units matter, append them...
 - Variable names also
 - Things go wrong...
 - Mars Climate Orbiter

... on September 23, 1999, communication with the spacecraft was lost as the spacecraft went into orbital insertion, due to ground-based computer software which produced output in non-SI units of pound-seconds ($\text{lbf}\times\text{s}$) instead of the metric units of newton-seconds ($\text{N}\times\text{s}$) specified in the contract between NASA and Lockheed.

- Wikipedia



655.2 million dollar mistake!

Naming

When naming functions and variables...

- Be consistent about pluralization / type IDs

```
122  list_of_prime = [...]
123  primes = [...]
124  list_of_primes = [...]
125  prime_list = [...]
```

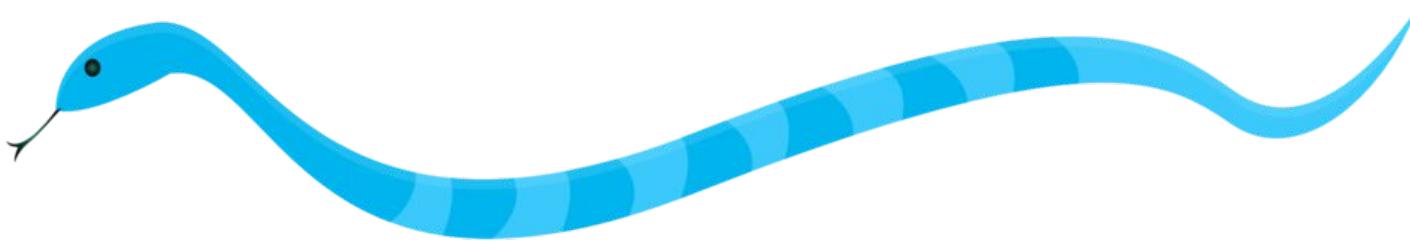


- Which do you prefer and why?
- Given your choice for the above, what would I name a variable that contained a collection of times? Users?

Naming

Conventions

- Module names should be short, lowercase
 - Underscores are OK if it helps with readability
- Package names should be short, lowercase
 - Underscores are frowned upon and people will speak disparagingly behind your back if you use them



Naming

Functions

- Some prefixes that can be used to clarify
 - compute **Reserve for functions that compute something sophisticated, e.g. not average**
 - get/set
 - find
 - is/has/can **What kind of return value would you expect from a function with this kind of prefix?**
 - Use complement names for complement functions
 - get/set, add/remove, first/last



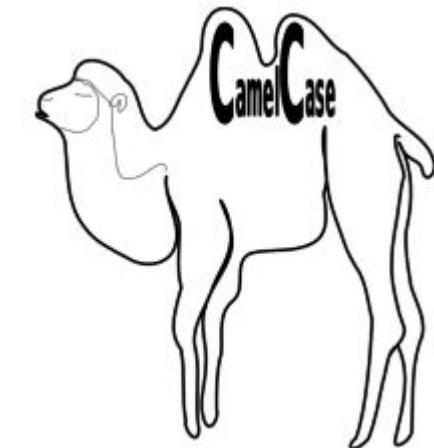
VectorStock®

VectorStock.com/13416663

Naming

Class names should be in CapWords

- SoNamedBecauseItUsesCapsForFirstLetterInEachWord
- Also known as CamelCase
- Notice no underscore!
- Much hate on the internet for Camel_Case



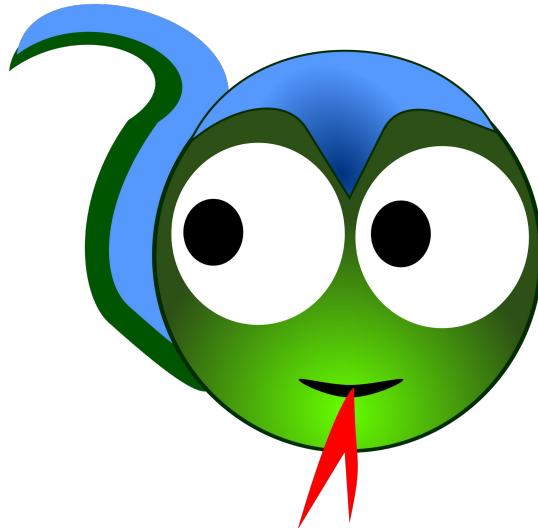
Naming

What naming convention should I use for exceptions?



WHY?

Naming



Functions

- Lowercase, with words separated by underscores as necessary to improve readability
- mixedCase is permitted if that is the prevailing style (some legacy pieces of Python used this style)
- Be consistent

Naming

- Naming conventions



Guidelines derived from Guido's Recommendations

Type	Public
Packages	
Modules	
Classes	
Exceptions	
Functions	
Global/Class Constants	
Global/Class Variables	
Instance Variables	
Method Names	
Function/Method Parameters	
Local Variables	

Exercise Part 2: Style

Run pylint on the style demo file:

```
pylint style_demo.py
```

Use an editor to get it to at least 8/10!

In addition, try to improve the following:

- Naming of functions and variables
- Pull common constants out into CONSTANTS

Ignore documentation-related issues for now.



Most important rule: consistency



UNIVERSITY *of* WASHINGTON

Documentation

W

Documentation

Documentation is for two types of people

Code comments



Code readers

README.md



Users

Documentation: README

- What kind of stuff goes in a repositories README.md?

https://github.com/kallisons/NOAH_LSM_Mussel_v2.0



Documentation: Comments

In Python:

```
# comment here
```

In shell scripting/bash:

```
# comment here
```

Always followed by a space
before the rest of the comment.



Documentation: Comments

Some examples of bad comments (from the 'net)

```
# For the brave souls who get this far: You are the chosen ones,  
# the valiant knights of programming who toil away, without rest,  
# fixing our most awful code. To you, true saviors, kings of men,  
# I say this: never gonna give you up, never gonna let you down,  
# never gonna run around and desert you. Never gonna  
# never gonna say goodbye. Never gonna tell a lie and
```

Don't Rick Roll
your readers!

```
# drunk, fix later
```

Uhm... Sigh.



Documentation: Comments

```
# Dear maintainer:  
#  
# Once you are done trying to 'optimize' this routine,  
# and have realized what a terrible mistake that was,  
# please increment the following counter as a warning  
# to the next guy:  
#  
# total_hours_wasted_here = 42
```



```
true = false  
# Happy debugging suckers
```

<http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered>

Funny is funny, but don't troll. And what was the issue the writer encountered?

At least it is logical?

Documentation: Comments

Good comments...

- ...make the code easy to read
- ...are written in English
- ...discuss the function parameters and results



```
211 % parameters:  
212 %   sequence = character string of nucleotide letters (ATCG)  
213 % returns:  
214 %   geneStarts = vector of start index into sequence of start codon  
215 %   geneEnds = vector of stop index into sequence of stop codons  
216 %           value is the first base of the stop codon  
217 function [ geneStarts, geneEnds ] = callGenesFromSequence(sequence)  
218 ...  
219 end
```

Documentation: Comments

Don't comment bad code, rewrite it!

```
223 % this is so terrible
224 % I can't find the bug here but, just subtract the length of x from
225 % the result and divide by the length
226 function meanX = averageX(x)
227     meanX = sum(x) + length(x)
228 end
```

Then comment it

```
230 % parameters:
231 % x = a vector of numerics
232 % returns:
233 % meanX = the average of the vector x
234 function meanX = averageX(x)
235     meanX = sum(x) / length(x)
236 end
```



Documentation: Inline Comments

- Comments on the same line as code

```
177 x = x + 1 # Increment x
```

- Use sparingly
 - Make sure it's actually useful!
 - Make sure the line isn't too long
 - Make sure the code is still readable



Documentation: Comments in Code

- Wrong comments are bugs

```
179     # Unit test for the sweepFiles function to test bounds
180     # checking on metadata parameters.
181     def test_sweepFiles_metadataType(self):
182         with self.assertRaises(TypeError):
183             io.sweepFiles('examples', metadata=41)
```

- Don't forget to update the comments!



Documentation: Comments in Code

- Don't insult the reader!



```
240 % compute the square root of the square of the distance
241 distance = sqrt(distanceSquared);
```

- If they're reading your code, they aren't that dumb

Corollary:

- Don't comment every line

```
187 # Find the square of the 3D distance.
188 distance_squared = (x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2
189 # Compute the square root of the square of the distance.
190 distance = math.sqrt(distance_squared);
191 # Make sure the distance is less than 3.5 Angstroms or error.
192 if distance < 3.5:
193     # Throw an error.
194     error('interatomic distance is less than 3.5 Angstroms')
195 else:
196     # Add the distance to the list of distances.
197     distances.append(distance)
```

Documentation: Comments in Code

Example: what's not great about this, other than too many comments?

```
187 # Find the square of the 3D distance.  
188 distance_squared = (x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2  
189 # Compute the square root of the square of the distance.  
190 distance = math.sqrt(distance_squared);  
191 # Make sure the distance is less than 3.5 Angstroms or error.  
192 if distance < 3.5:  
193     raise Exception('distance violation',  
194                      'interatomic distance is less than 3.5 Angstroms')  
195 else:  
196     # Add the distance to the list of distances.  
197     distances.append(distance)
```

What if I want to change the cutoff distance? There are 3 places to change it!



Documentation: Comments in Code

Note: English!
Full sentences!

Better

```
199 # Find the square of the 3D distance and compute the sqrt,
200 #      then compare the distance to our cutoff (defined elsewhere)
201 #      and throw an error if the distance is too small
202 #      otherwise add the distance to our vector of distances.
203 distance_squared = (x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2
204 distance = math.sqrt(distance_squared);
205 if distance < DISTANCE_CUTOFF:
206     raise Exception('distance violation',
207                      'interatomic distance is less than %.2f Angstroms'
208                      % DISTANCE_CUTOFF)
209 else:
210     distances.append(distance)
211
```

- Obvious constant
- Comments are consolidated and don't mention the cutoff



Documentation: Docstrings

When you provide a plain old string as the FIRST thing in a (function or class or module)

```
57  def sorted_list_difference(expected, actual):
58      """Finds elements in only one or the other of two, sorted input lists.
59
60      Returns a two-element tuple of lists.    The first list contains those
61      elements in the "expected" list but not in the "actual" list, and the
62      second contains those elements in the "actual" list but not in the
63      "expected" list.    Duplicate elements in either input list are ignored.
64      """
```

Use triple quotes



Documentation: Docstrings

What are they for?

- Documenting things users will need to know how to use
 - Functions
 - Modules
 - Classes



Documentation: Docstrings

Why?

- They can be processed by the *docutils* package into HTML, LaTeX, etc. for high quality code documentation (that makes you look smart).

Try it out when you've written some docstrings:

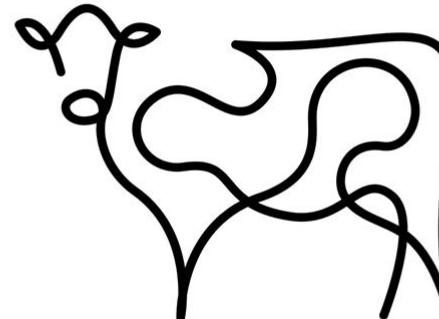
```
python -m pydoc my.module
```



Documentation: Docstrings

One line for simple stuff

```
115     def three_way_cmp(x, y):  
116         """Return -1 if x < y, 0 if x == y and 1 if x > y"""  
117         return (x > y) - (x < y)
```



Documentation: Docstrings

Multiline for more complicated things



```
1 """
2 A multiline doc string begins with a single line summary (<72 chars).
3
4 The summary line may be used by automatic indexing tools; it is
5 important that it fits on one line and is separated from the rest of
6 the docstring by a blank line.
7 """
8
9 import sys
```

Documentation: Docstrings

For scripts intended to be called from the command line, the docstring at the top of the file should be a usage message for the script.



```
....  
Usage: my_program.py [-hso FILE] [--quiet | --verbose] [INPUT ...]  
  
-h --help      show this  
-s --sorted    sorted output  
-o FILE       specify output file [default: ./test.txt]  
--quiet        print less text  
--verbose      print more text  
....
```

Documentation: Docstrings

For modules, list the classes, exceptions and functions (and any other objects) that are exported by the module, with a one-line summary of each.

https://github.com/numpy/numpy/blob/master/numpy/_init_.py



Documentation: Docstrings

For functions, include all of the good interface specification stuff we've talked about:

- Name
- Arguments
- Return values
- Side effects
- Edge cases
- Exceptions
- Examples



https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/cluster/_dbscan.py

Exercise Part 3: Documentation

Run pylint on the style demo file:

```
pylint style_demo.py
```

Use an editor to get it to 10/10!



Lastly: Ignoring recommendations

- Sometimes you disagree with a PEP8 recommendation
- Or you made a very conscious choice to do it differently
- You can tell pylint to ignore the issue with a comment on the line before:

```
# pylint: disable=consider-using-f-string
```

- NOT OFTEN!
 - We WILL deduct points on your project if you use this without a good reason

Exercise 4: Adding Style to Continuous Integration

You *must* add linting to your continuous integration for your project:

https://github.com/UWDATA515/ci_example/blob/536d2f84c2a18dd3d82647cfb4d6dfd38ab52068/.github/workflows/build_test.yml#L95-L100

However, it probably won't pass the first time.

- Take this time to fix style and add documentation to your project
- Then add linting to your continuous integration GitHub Actions Workflow