

Software Engineering for Data Scientists

Style & Documentation

Bernease Herman^{1,2}, Joseph Hellerstein^{1,2}

¹eScience Institute

²Computer Science Engineering

April 21, 2020

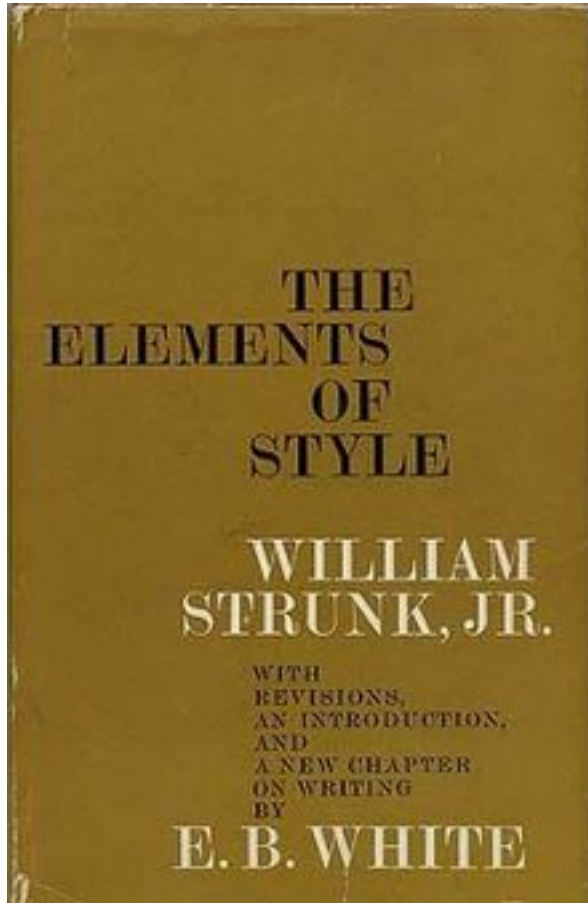


Agenda

- Style
 - Intro
 - Survey of PEP8
- Documentation



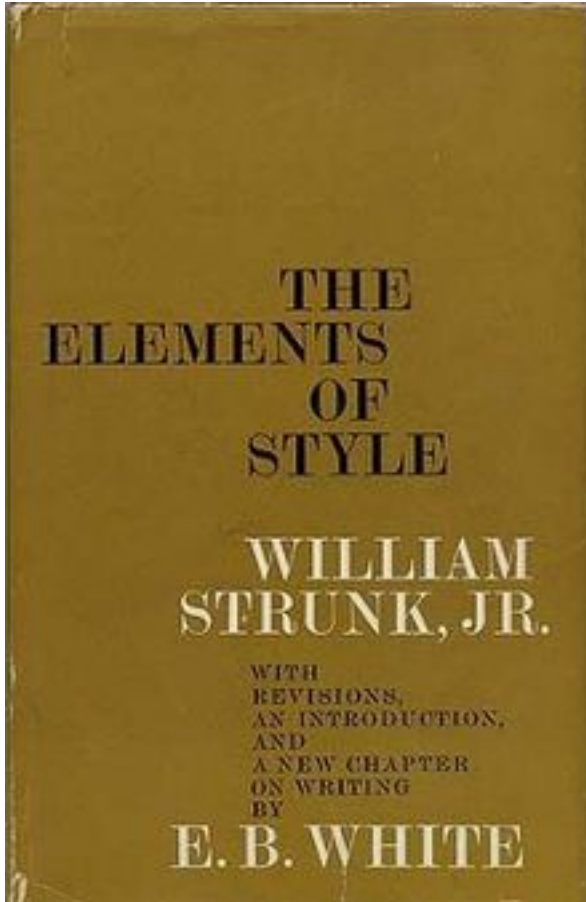
Elements of Style



- AKA Strunk & White
- 1918 (Strunk), 1959
- Doesn't describe how to write in English
- Describes how to write effectively in English
- Writing is understandable and efficient



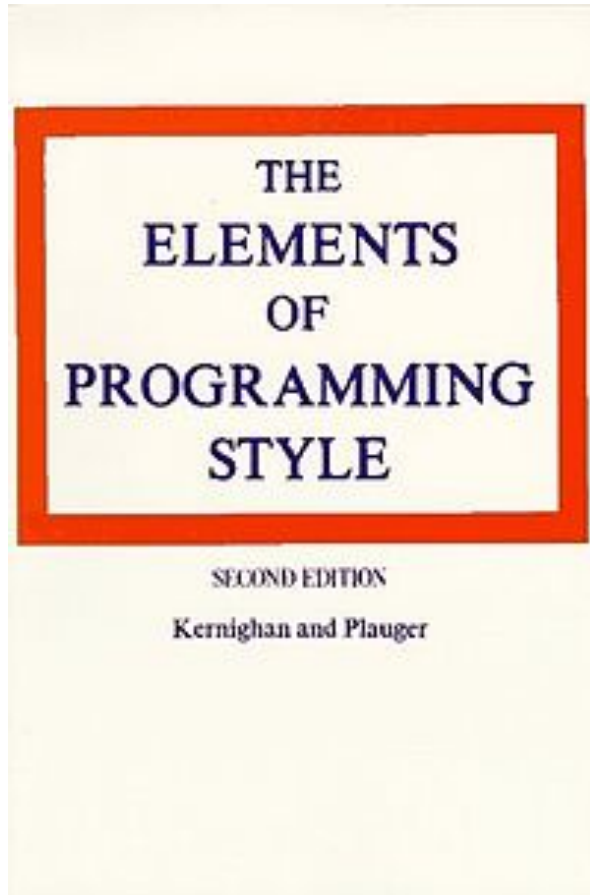
Elements of Style



- AKA Strunk & White
- “Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts. This requires not that the writer make all his sentences short, or that he avoid all detail and treat his subjects only in outline, but that he make every word tell.”



Elements of Programming Style



- 1974
- Doesn't describe how to write programs
- Describes how to write efficient programs that can be read by computers and people



Programming Style

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



Style

- Analogies for describe programming style
 - It is like putting the toilet seat down after you done using the toilet
 - It doesn't change how a toilet works
 - It makes it nicer/easier for the person who comes after you



Programming Style

- Like debugging
 - Programming style is an important part of professional software development
 - Life changing experience and habit forming
 - Converts your code from hack-ish one-offs to reusable gems of useful intellectual property



Programming Style

- Most important rule of any style
 - Consistency
 - If you make particular decision about a style guide, use it consistently



Programming Style

- There isn't only one 'style'
 - Most common: PEP8
 - Python Enhancement Proposal #8
 - Tools for checking the style adherence
 - `pip install pep8`
 - Google Python Style Guide
 - *Python (was?) is the main scripting language used at Google. This style guide is a list of dos and don'ts for Python programs.*
 - Vim & Emacs 'plugins' for style adherence



Programming Style

- Another tool...
 - *pylint* (<http://www.pylint.org>)
 - Extends from a tool called *lint* introduced for C from Bell Labs in 1976
 - <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.1841>

```
256 ▼ def complex(real=0.0, imag=0.0):
257     """Form a complex number.
258
259     Keyword arguments:
260     real -- the real part (default 0.0)
261     imag -- the imaginary part (default 0.0)
262     """
263 ▼     if imag == 0.0 and real == 0.0:
264         return complex_zero
```



PEP8

- Indentation
 - Four spaces
 - Most editors can be set to convert a tab that you type to four spaces in the file
 - What about lines that wrap? Two options...
 - Wrap and indent to opening of parens
 - Hanging indent (put nothing after parens and indent only once)



- Indentation

- 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
14 # Hanging indents should add a level.
15 foo = long_function_name(
16 |     var_one, var_two,
17 |     var_three, var_four)
18
```



PEP8

- Indentation

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

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



PEP8

- Indentation

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

```
38  
39  # Add a comment, which will provide some distinction in editors  
40  # supporting syntax highlighting.  
41  if (this_is_one_thing and  
42      that_is_another_thing):  
43      # Since both conditions are true, we can frobnicate.  
44      do_something()  
45  
46  # Add some extra indentation on the conditional continuation line.  
47  if (this_is_one_thing  
48      and that_is_another_thing):  
49      do_something()  
50
```



PEP8

- Indentation

```
55  
56 my_list = [  
57     1, 2, 3,  
58     4, 5, 6,  
59 ]  
60 # Versus  
61 my_list = [  
62     1, 2, 3,  
63     4, 5, 6,  
64 ]  
65
```

Equivalent, no specific recommendation



PEP8

- Maximum line length?
 - Coding lines? Keep it to 79 characters
 - Most editors can show you the line position
 - E.g. vim, Sublime
 - 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



PEP8

- 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?*)



PEP8

- Imports

- *Some discussion of this already*
- Imports go at the top of a file after any comments
- Imports for separate libraries go on separate lines

```
71  import os
72  import sys
73  # Versus
74  import os, sys
```



PEP8

- Imports

- Imports should be grouped with a blank line separating each group in the following order:

- Standard library imports

- os, sys, ...

- Related third party imports

- matplotlib, seaborn, numpy, etc...

- Local application / library specific imports

- pronto_utils

```
1 import os
2 import sys
3
4 import pandas as pd
5 import numpy as np
6
7 from phylodist.constants import TAXONOMY_HIERARCHY, PHYLODIST_HEADER
8
```



PEP8

- Imports
 - Avoid wildcard imports
 - Be explicit about namespaces when necessary

```
11  from ubermod import *
12
13
14  from ubermod import namespace_collision
15  from sillymod import namespace_collision
16
17
18
19  import ubermod
20  import sillymod
21
22  ubermod.namespace_collision(...)
23  sillymod.namespace_collision(...)
```



PEP8

- Quotes
 - When should I use single?
 - When should I use double?



PEP8

- Quotes
 - PEP8 has no recommendation about single vs. double
 - Except for triple quotes strings, use double
 - Multiline strings, docstrings, etc.



PEP8

- Whitespace
 - No trailing spaces at end of a line
 - Do not pad ([{ with spaces, e.g.

```
78 spam(ham[1], {eggs: 2})  
79 spam( ham[ 1 ], { eggs: 2 } )  
80
```

- Do not pad before : ; , , e.g.

```
82 if x == 4: print x, y; x, y = y, x  
83 if x == 4 : print x , y ; x , y = y , x  
84
```

What else is wrong with the above?



PEP8

- Whitespace

- Always surround =, +=, -=, ==, <, >, !=, <>, <=, >=, in, not in, is, is not, and, or, not with a single space

```
86  i = i + 1
87  submitted += 1
88  x = x*2 - 1
89  hypot2 = x*x + y*y
90  c = (a+b) * (a-b)
```

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

Having fun yet?



PEP8

- Whitespace
 - Never surround = with a space as a function parameter argument

```
101 def complex(real, imag=0.0):  
102     return magic(r=real, i=imag)  
103  
104  
105 def complex(real, imag = 0.0):  
106     return magic(r = real, i = imag)  
107
```



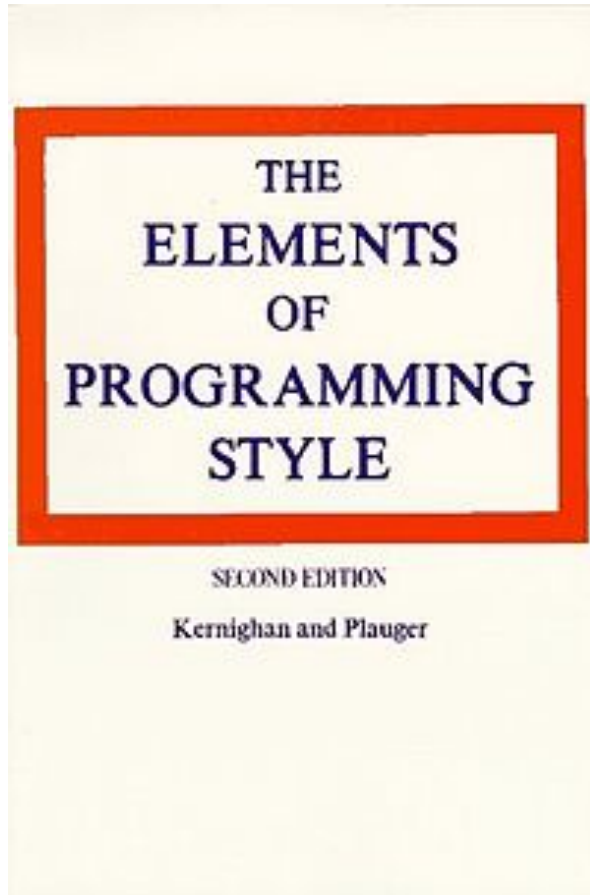
PEP8

- Compound statements

```
110 if foo == 'blah': do_blah_thing()
111 else: do_non_blah_thing()
112
113 try: something()
114 finally: cleanup()
115
116 do_one(); do_two(); do_three(long, argument,
117                               list, like, this)
118
119 if foo == 'blah': one(); two(); three()
```



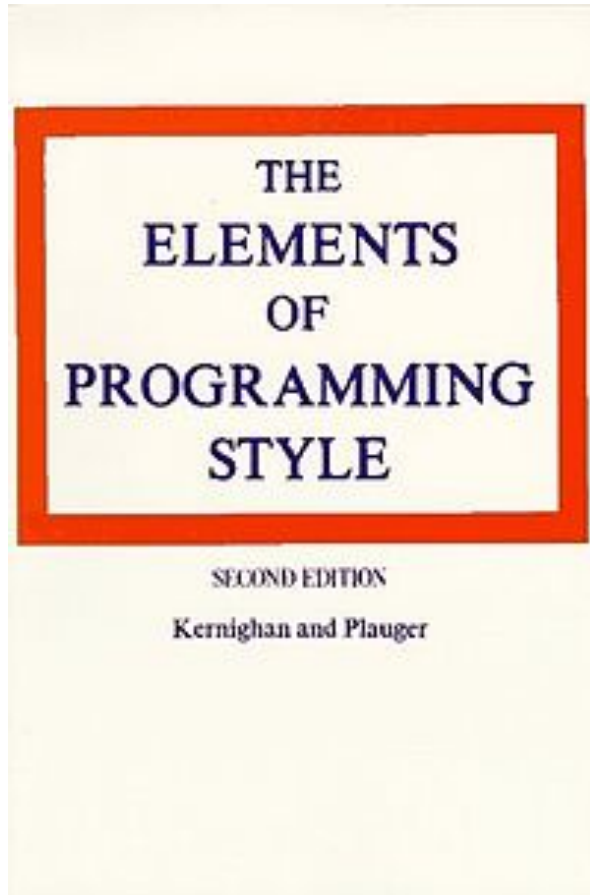
Elements of Programming Style



- 1974
- Fortran & PL/1¹
- Most of the lessons are language free, e.g.
 - *Replace repetitive expressions by calls to a common [f]unction.*
 - *Choose variable names that won't be confused.*



Elements of Programming Style



- Also addresses
 - Software design
 - Common pitfalls, e.g.

10.0 times 0.1 is hardly ever 1.0.

Don't sacrifice clarity for small gains in "efficiency."



Elements of Programming Style

- *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))
```



PEP8

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

```
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))
```

PEP8

- Naming conventions
 - Avoid the following variable names:
 - Lower case l (l)
 - Upper case O (O)
 - Upper case I (I)
 - Why?
 - Can be confused with 1 and 0 in some fonts
 - Can be confused with each other (i.e. l and I)



PEP8

- 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



PEP8

- Naming conventions

- Class names should be in CapWords

- SoNamedBecauseItUsesCapsForFirstLetterInEachWord
 - Also known as CamelCase

- Notice no underscore!
 - Much hate on the internet for
Camel_Case



PEP8

- Naming conventions
 - What naming convention should I use for exceptions?



WHY?



PEP8

- Naming conventions

- 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)
 - Easy habit to fall into... Very common in style guides for other languages, e.g. R
 - If this is your thing, then be consistent



PEP8

Why be consistent from day 1?



PEP8

- Naming conventions
 - Functions

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	



Programming Style

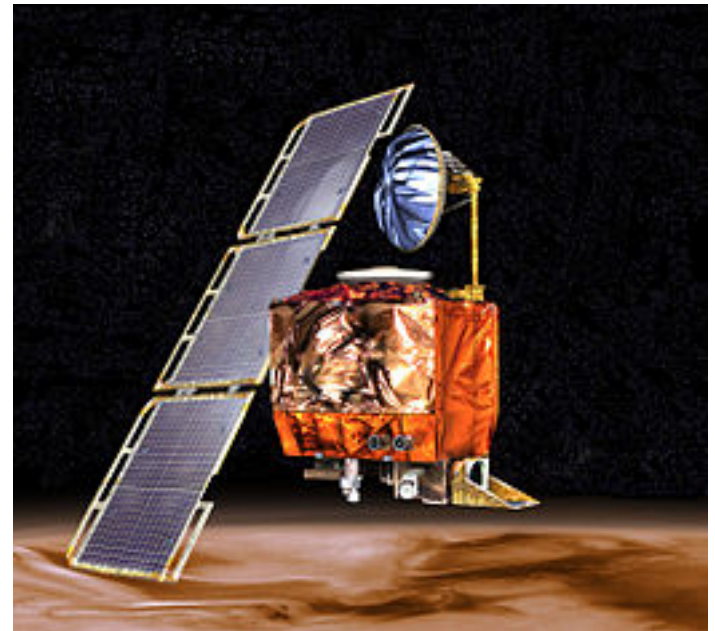
- Functions

- 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 (lbf×s) instead of the metric units of newton-seconds (N×s) specified in the contract between NASA and Lockheed.

- Wikipedia



655.2 million dollar mistake!



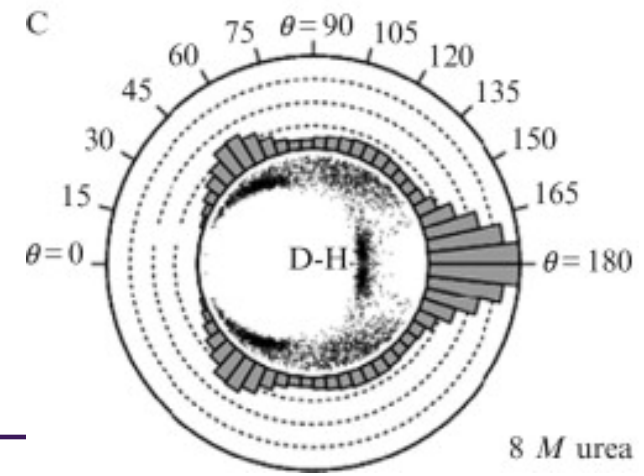
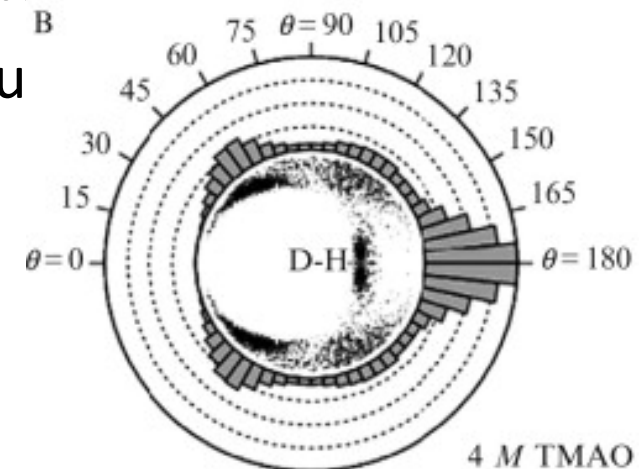
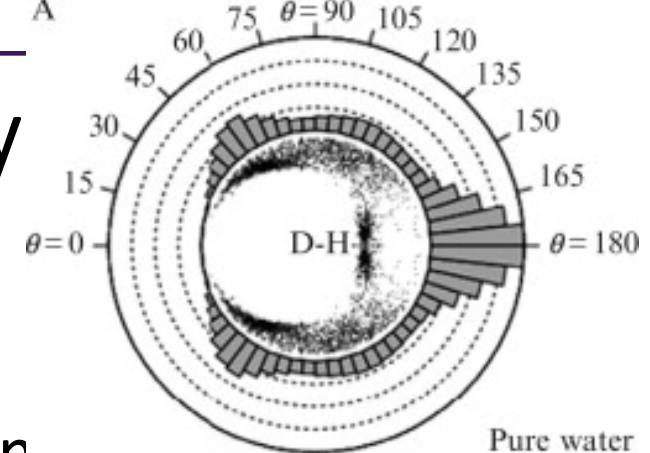
Programming Sty

- Functions
 - Where units matter, append them
 - Why do you need to use different u

Example fromwork in
molecular mechanics

```
math.cos(x)
```

Return the cosine of x radians.



Programming Style

- 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



PEP8

- Naming conventions
 - Global variables
 - These shouldn't be truly global, just global to a module's namespace
 - Function variables
 - Function / method arguments
 - All of the above use lower case with words separated by underscores



PEP8

- Naming conventions

```
163 EXPECTED_RECORD_LENGTH = 5
164 TAXONOMY_FIELD = 4
165 EXPECTED_TAXONOMY_LENGTH = 8
166 TAXONOMY_HIERARCHY = [
167     "kingdom", "phylum", "class", "order",
168     "family", "genus", "species", "strain"
169 ]
170 PHYLODIST_HEADER = [
171     "locus_tag", "subject_locus", "subject_locus_tag", "pident"
172 ]
173 IMG_DATA_DIRECTORY_NAME = "IMG_Data"
174 PHYLODIST_FILE_SUFFIX = "phylodist"
```

- Python has no specific capability for making a variable constant; all variables are reassignable
- To indicate a variable is a constant, use all CAPS, e.g.

```
1 import os
2 import sys
3
4 import pandas as pd
5 import numpy as np
6
7 from phylodist.constants import TAXONOMY_HIERARCHY, PHYLODIST_HEADER
8
```



PEP8

- Naming conventions

- Constants

- Remember this ugly mess? What could be a constant?

```
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))
```



PEP8

- Naming conventions
 - 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 choose for the above, what would I name a variable that contained a collection of times? Users?



PEP8

- There is a lot more detail in the PEP8 spec (on the syllabus)... E.g. Why is this not only bad style but potentially buggy?

```
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
```



PEP8

- Correct version

```
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
```



PEP8

- Programming Recommendations
 - This section is highly recommended
 - <https://www.python.org/dev/peps/pep-0008/#id46>
 - E.g. Use `".startswith()"` and `".endswith()"` instead of string slicing to check for prefixes or suffixes.

```
151 if filename.endswith('zip'):
```

```
153 if filename[3:] == 'zip':
```

- E.g. For sequences, (strings, lists, tuples), use the fact that empty sequences are false.

```
156 if not seq:  
157 if seq:
```

```
159 if len(seq)  
160 if not len(seq)
```



Documentation



Documentation

- Two types
 - Code readers
 - What the code is doing and why
 - E.g. **Code comments**
 - Users
 - How to use your code
 - E.g. **README.md, docstrings**



Documentation

- .md
 - .md files are Markdown
 - Markdown is a lightweight text formatting language for producing mildly styled text
 - Ubiquitous (github.io, README.md, etc.)



Documentation

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

<https://github.com/apoorva-sh/zodiac-experiments>

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



Documentation

- Comments
 - Shell script
 - #
 - Python
 - #



• Comments Documentation

– 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 make you cry,
% never gonna say goodbye. Never gonna tell a lie and hurt you.
%
```

Don't Rick Roll
your readers!

```
% drunk, fix later
```


Uhm... Sigh.

```
%
% 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
%
```

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

```
true = false;
% Happy debugging suckers
```

At least it is
logical

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

Documentation

- Good comments
 - Make the comments easy to read
 - 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

- Good 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

- Good comments
 - Some languages have special function headers



Documentation

- Good comments
 - Some languages have special function headers
 - This example is fantastic!
 - It describes
 - Calling synopsis (example usage)
 - The input parameters
 - The output variables
 - Aimed at coders and users



Documentation

- Good comments
 - Some languages have special function headers
 - These comments should also describe **side effects**
 - Any global variables that might be altered
 - Plots that are generated
 - Output that is piked



Documentation / PEP8

- Good comments
 - Inline comments
 - Comments inline with the code

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

- Generally unnecessary (as above)
- Inhibit readability



Documentation

- Good comments
 - 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)
```

- When updating code, don't forget to update the comments



Documentation

- Good comments
 - Don't insult the reader

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

- If they are reading your code... they aren't that dumb
- Corollary: don't comment every line!



Documentation

- Good comments
 - 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

- Good comments
 - Problems with this code (other than excessive 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)
```



Documentation

- Good comments
 - Problems with this code (other than excessive comments?)
 - What happens if I want to change the cutoff distance
 - I have to change the code (in 2 places)
 - I have to change the comment

```
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)
```

Documentation

- Good comments

```
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)
```

- Note how the block is commented
- The code itself reads clearly enough
- We used an obviously marked constant whose value is displayed if an error is encountered



Documentation / PEP8

- Good comments
 - Comments should be sentences. They should end with a period. There should be a space between the # and the first word of a comment.

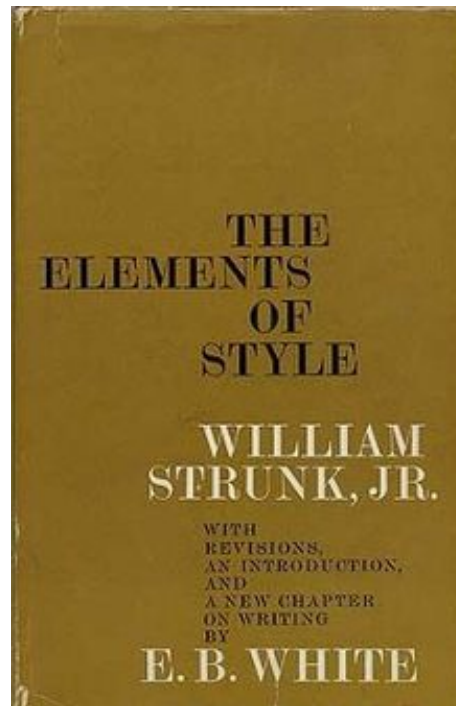
```
199 # Find the square of the 3D distance and compute the sqrt,  
200 #     then compare the distance to our cutoff (defined elsewhere)  
201 #     and throw an error if the distance is too small  
202 #     otherwise add the distance to our vector of distances.
```

- You should use two spaces after a sentence-ending period. Although Microsoft Word has abandoned this in April 2020!



Documentation / PEP8

- Good comments
 - Most standard to have comments written in English, and follow Strunk and White.



Documentation / PEP 0257

- Docstrings
 - String literal as the first statement in
 - Modules
 - Functions
 - Classes

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

If you violate these conventions, the worst you'll get is some dirty looks. But some software (such as the [Docutils \[3\]](#) docstring processing system [\[1\]](#) [\[2\]](#)) will be aware of the conventions, so following them will get you the best results.



Documentation / PEP 0257

- Docstrings
 - They are triple quoted strings
 - What kind of quotes to use?
 - They can be processed by the *docutils* package into HTML, LaTeX, et. for high quality code documentation (that makes you look smart).
 - They should be phrases (end in period).



Documentation / PEP 0257

- Docstrings
 - One line doc strings are OK for simple stuff.

```
213 def kos_root():  
214     """Return the pathname of the KOS root directory."""  
215     global _kos_root  
216     if _kos_root: return _kos_root  
217
```

- This example (taken from PEP 0257) is a little more questionable...



Documentation / PEP 0257

- Docstrings
 - Multiline docstrings are more of the norm

```
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 / PEP 0257

- 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 / PEP 0257

- Docstrings
 - For modules and packages, list the classes, exceptions and functions (and any other objects) that are exported by the module, with a one-line summary of each.
 - Looking at scikit learn and seaborn (as examples) this didn't seem to be the norm. However,

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



Documentation / PEP 0257

- Docstrings
 - **Most importantly...** For functions and methods, it should summarize its behavior and document its arguments, return value(s), **side effects**, exceptions raised.
 - Example from scikit learn:

https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/cluster/dbscan_.py

