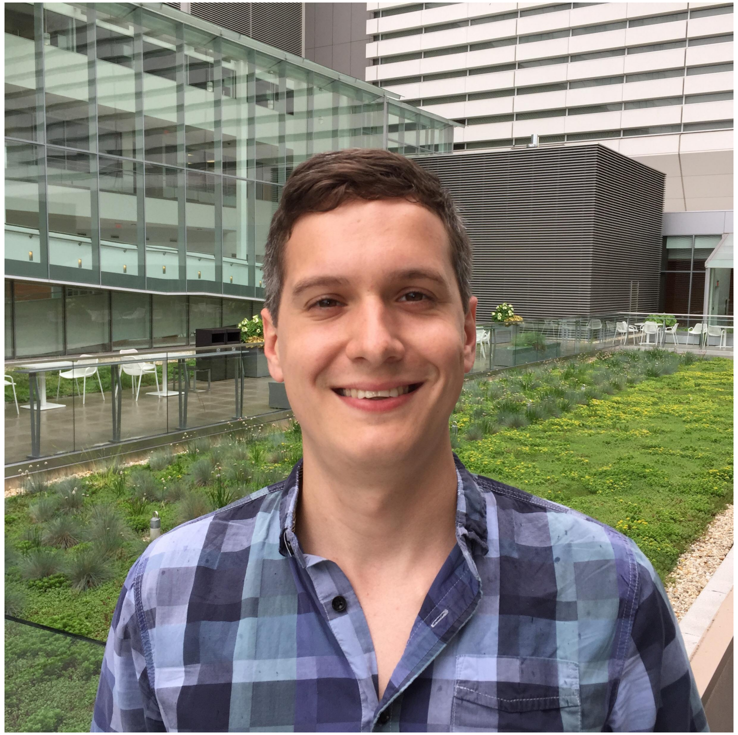
Code Clarity and Variable Naming in Python

Kurt Wheeler

About Me



Twitter: @datawheeler

Github: https://github.com/kurtwheeler/

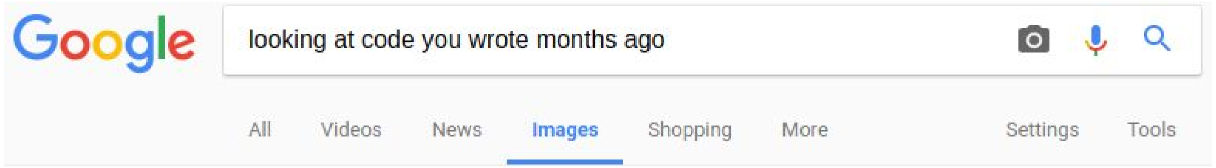
The Childhood Cancer Data Lab

“Empowering scientists and doctors to harness the power of Big Data”

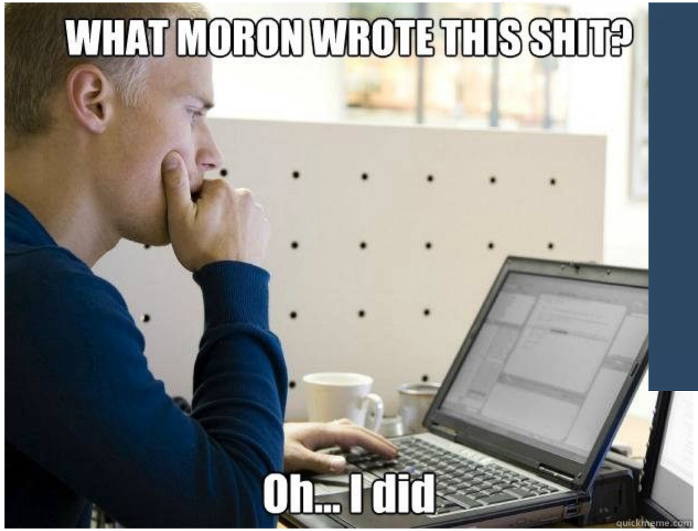
The Data Refinery

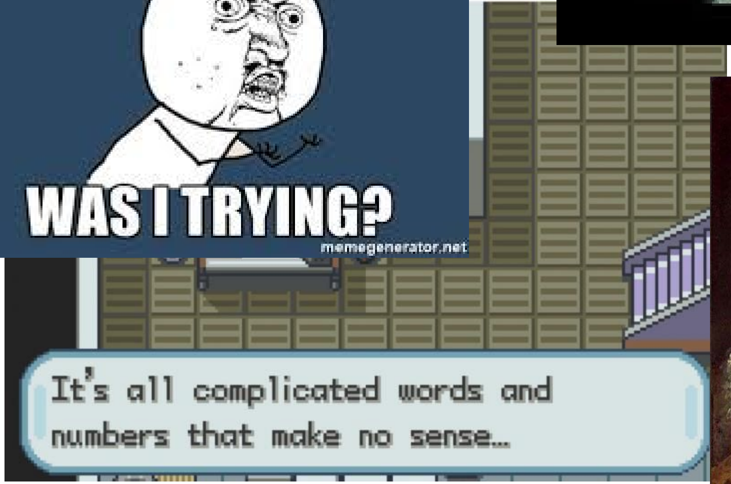
- 1.5 million publicly available genomic samples
- Multiple sources of data
- Growing number of processing options
- Distributed cloud architecture

Code Clarity: Why it's important

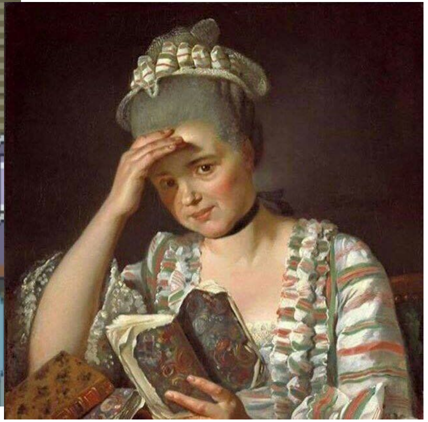

Google

looking at code you wrote months ago



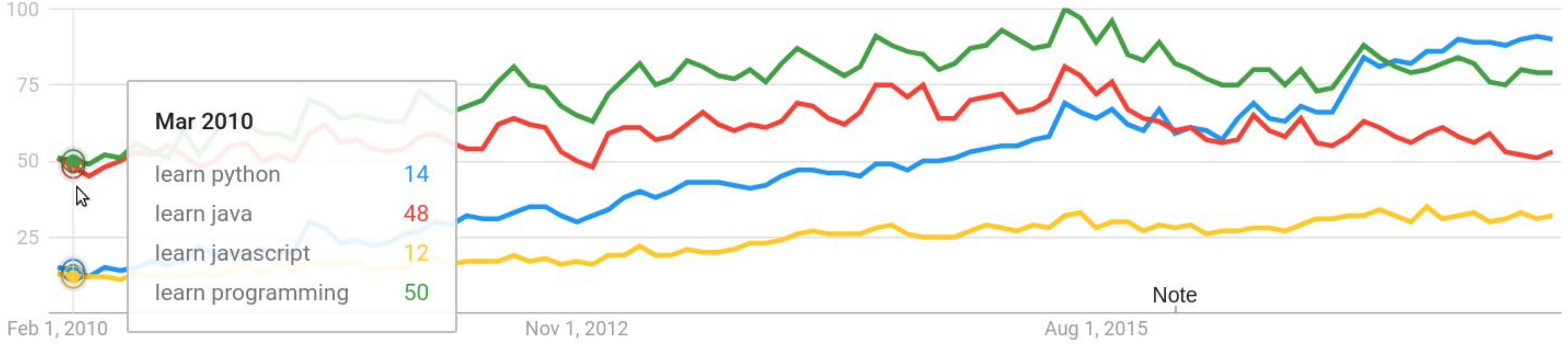

It's all complicated words and numbers that make no sense...

LOOKING AT SOME CODE YOU WROTE A WEEK AGO
I HAVE NO MEMORY OF THIS PLACE
memegenerator.net



Why Python?!?

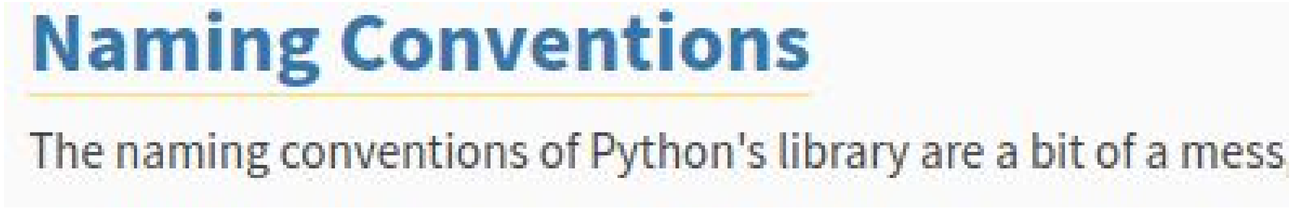
Python is a lot of programmers’ first language



*Graphic created using Google Trends

There’s a lot of bad examples of Python code

From pep8:

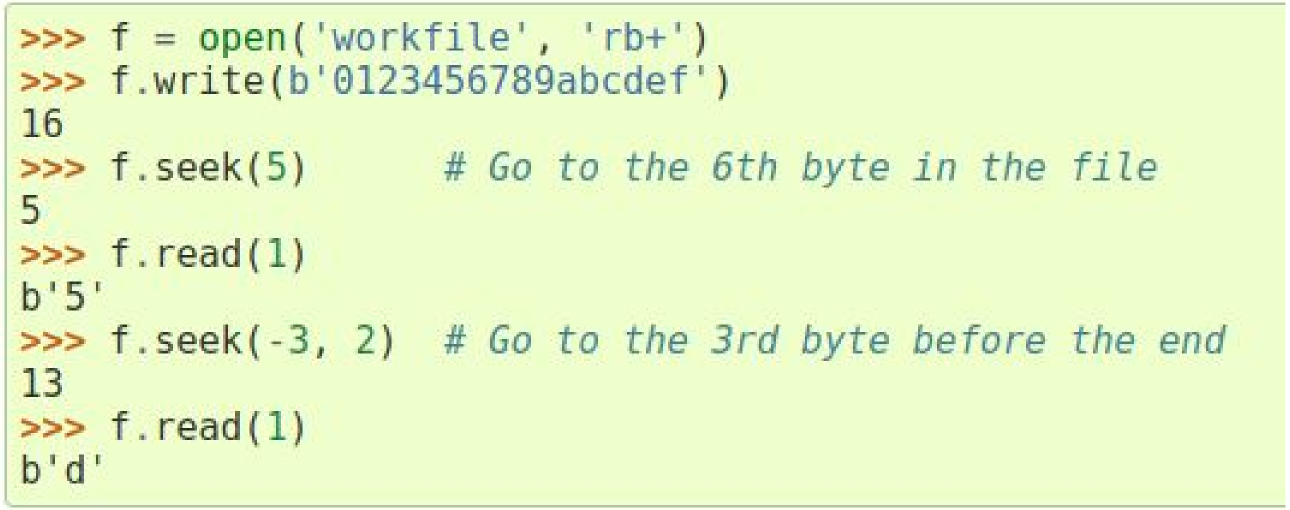


Naming Conventions

The naming conventions of Python's library are a bit of a mess

There are bad examples of Python...

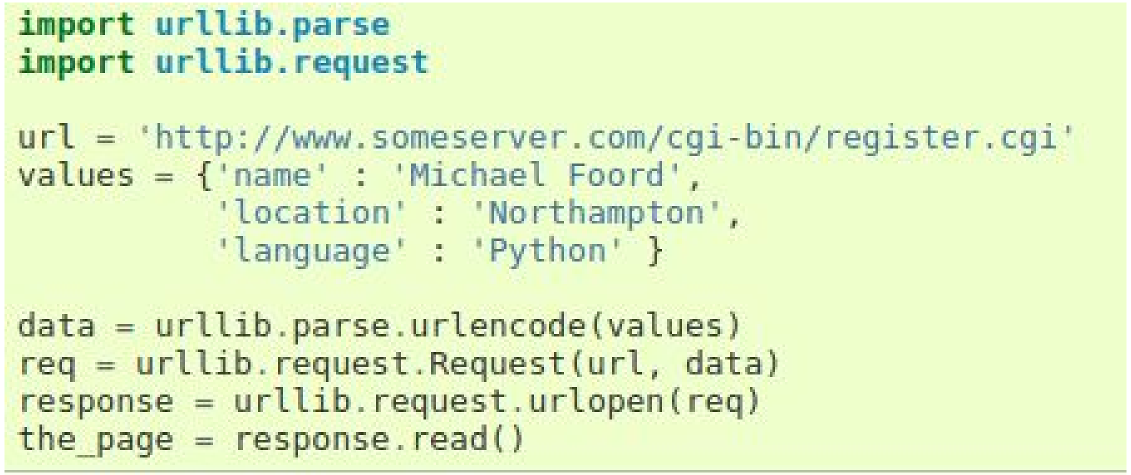
in official Python Tutorials



```
>>> f = open('workfile', 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)      # Go to the 6th byte in the file
5
>>> f.read(1)
b'5'
>>> f.seek(-3, 2)  # Go to the 3rd byte before the end
13
>>> f.read(1)
b'd'
```

There are bad examples of Python...

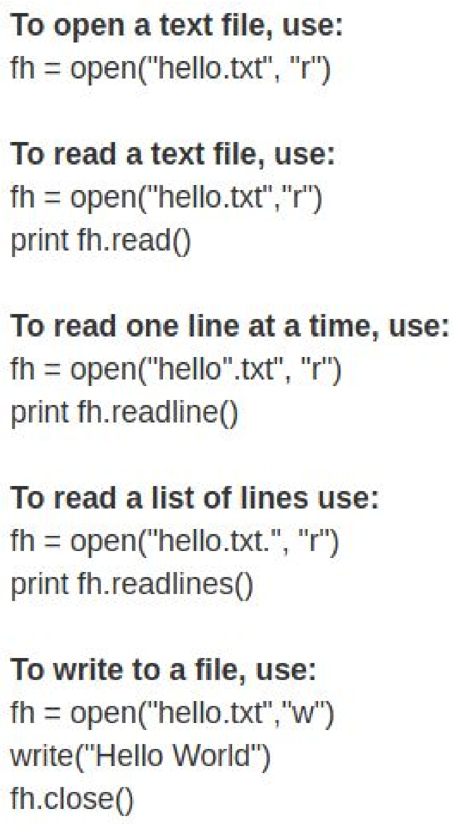
in other official Python tutorials



```python
import urllib.parse
import urllib.request

url = 'http://www.someserver.com/cgi-bin/register.cgi'
values = {'name' : 'Michael Foord',
          'location' : 'Northampton',
          'language' : 'Python' }

data = urllib.parse.urlencode(values)
req = urllib.request.Request(url, data)
response = urllib.request.urlopen(req)
the_page = response.read()
```

There are bad examples of Python...

in blogs meant for beginners



To open a text file, use:

```
fh = open("hello.txt", "r")
```

To read a text file, use:

```
fh = open("hello.txt","r")
print fh.read()
```

To read one line at a time, use:

```
fh = open("hello".txt", "r")
print fh.readline()
```

To read a list of lines use:

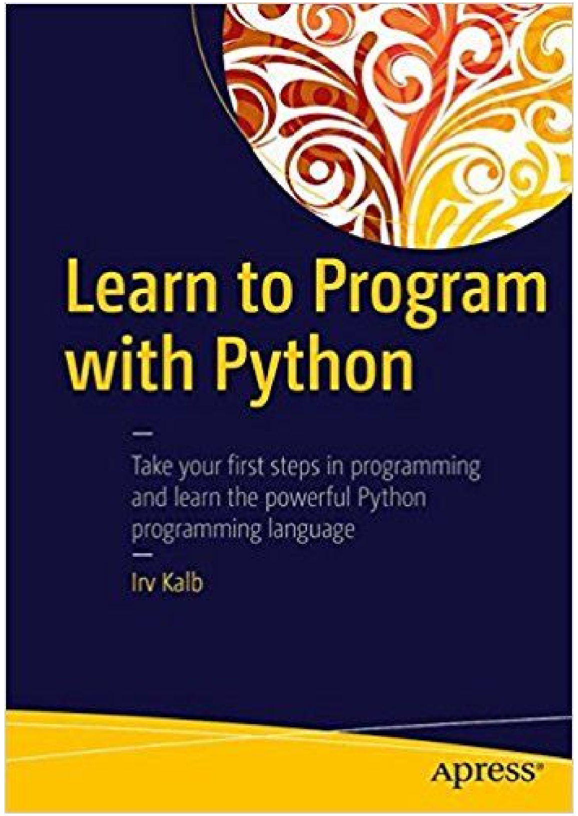
```
fh = open("hello.txt.", "r")
print fh.readlines()
```

To write to a file, use:

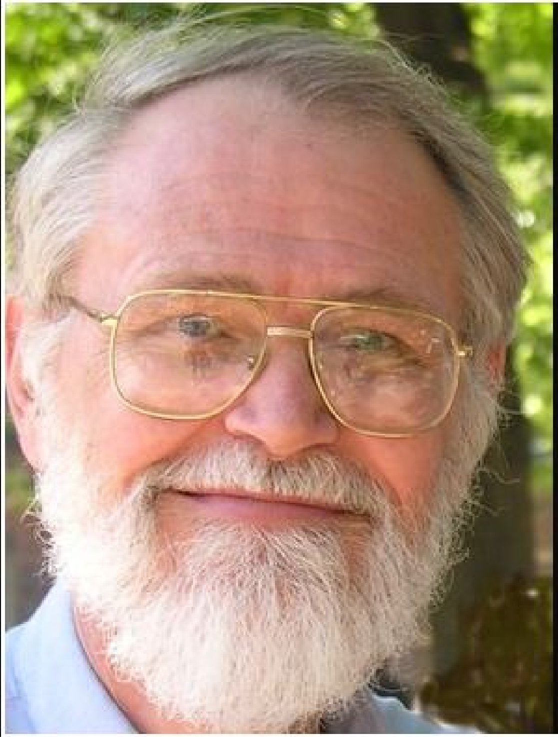
```
fh = open("hello.txt","w")
write("Hello World")
fh.close()
```

There are bad examples of Python...

in books meant for beginners


Learn to Program with Python
Take your first steps in programming and learn the powerful Python programming language
Irv Kalb
Apress

Writing Clear Python



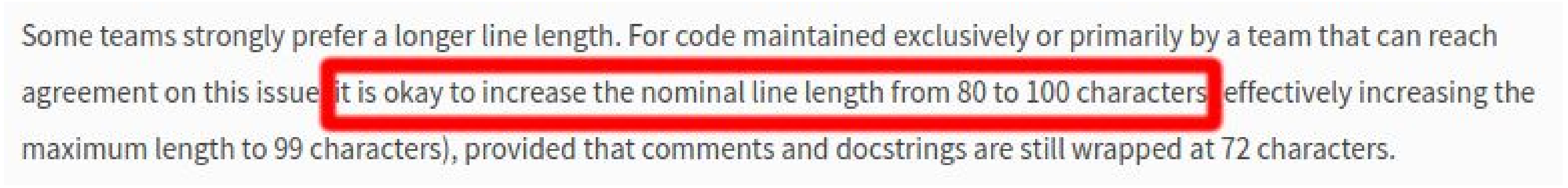
Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

— Brian Kernighan —

AZ QUOTES

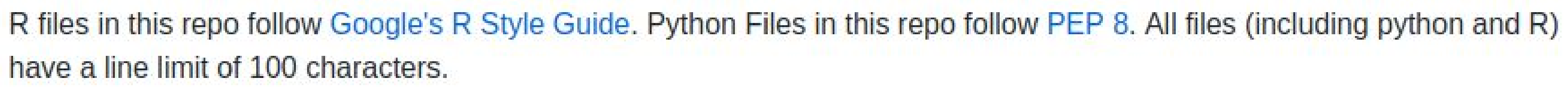
Longer lines can be helpful

From pep8:



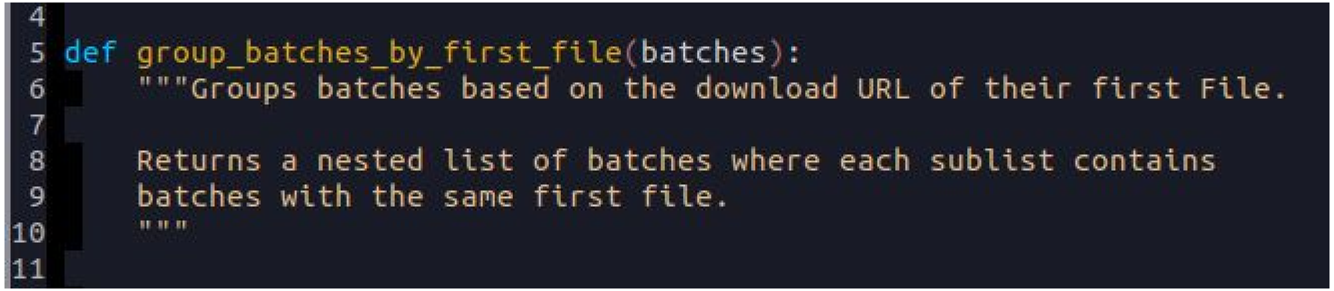
Some teams strongly prefer a longer line length. For code maintained exclusively or primarily by a team that can reach agreement on this issue it is okay to increase the nominal line length from 80 to 100 characters effectively increasing the maximum length to 99 characters), provided that comments and docstrings are still wrapped at 72 characters.

From the data-refinery/README.md:



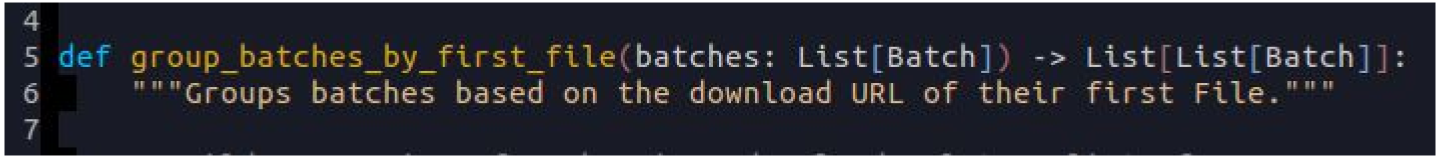
R files in this repo follow Google's R Style Guide. Python Files in this repo follow PEP 8. All files (including python and R) have a line limit of 100 characters.

Use Type Hints



```python

def group_batches_by_first_file(batches):
    """Groups batches based on the download URL of their first File.

    Returns a nested list of batches where each sublist contains
    batches with the same first file.
    """

```

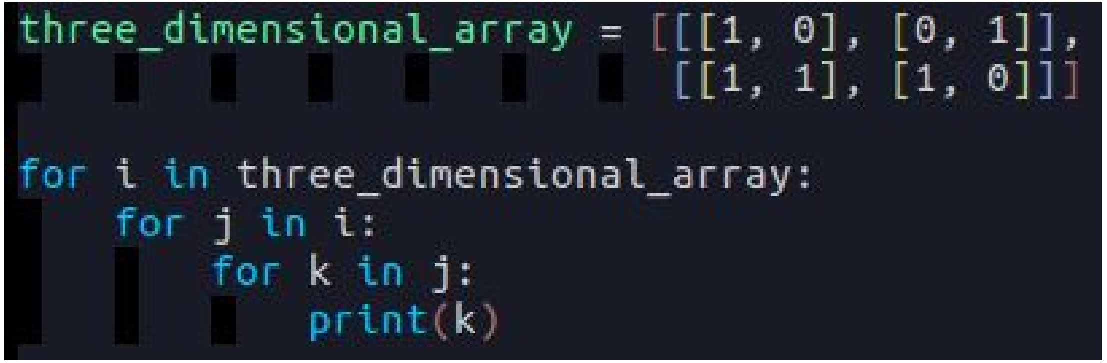
Becomes:



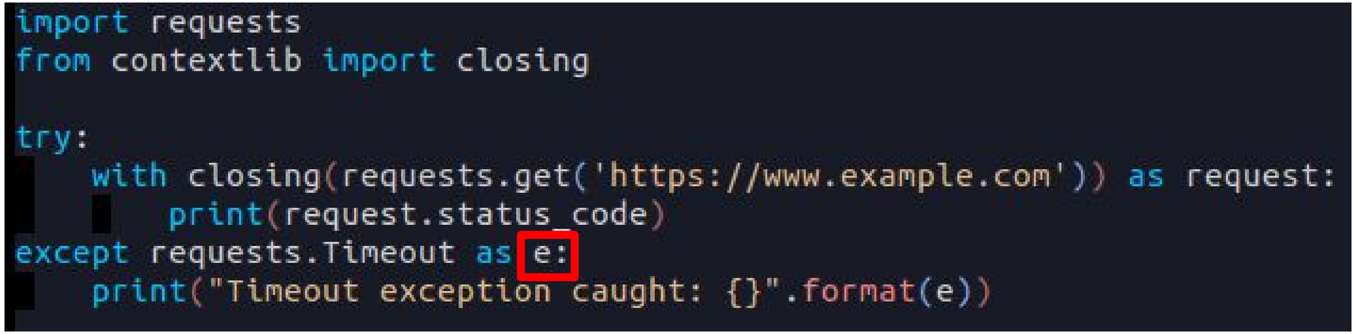
```python

def group_batches_by_first_file(batches: List[Batch]) -> List[List[Batch]]:
    """Groups batches based on the download URL of their first File."""

```

- Requires Python >= 3.5
- mypy: can perform static type checking[2]

Single letter variable names

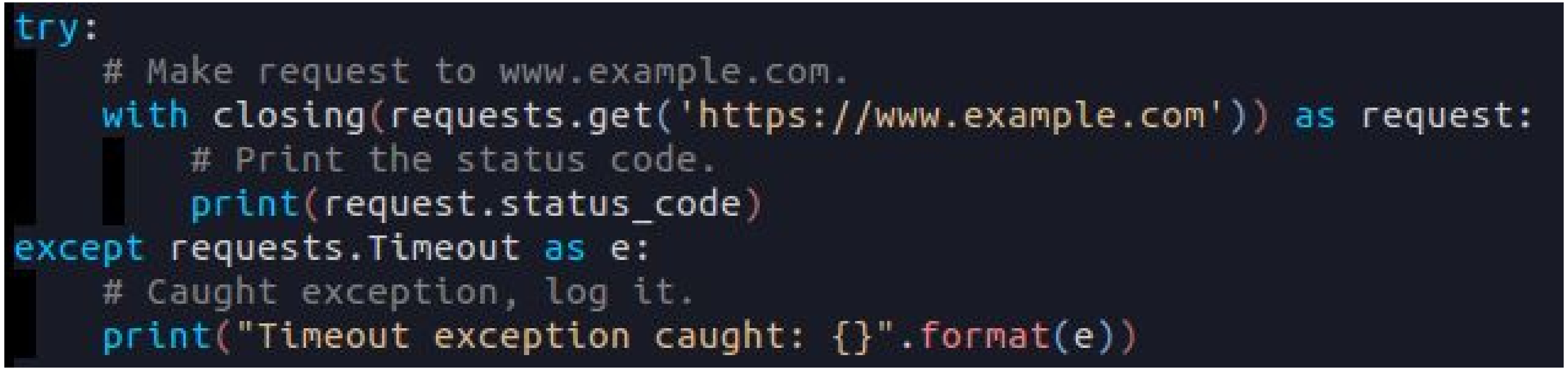


```
three_dimensional_array = [[[1, 0], [0, 1]],
                           [[1, 1], [1, 0]]]

for i in three_dimensional_array:
    for j in i:
        for k in j:
            print(k)
```

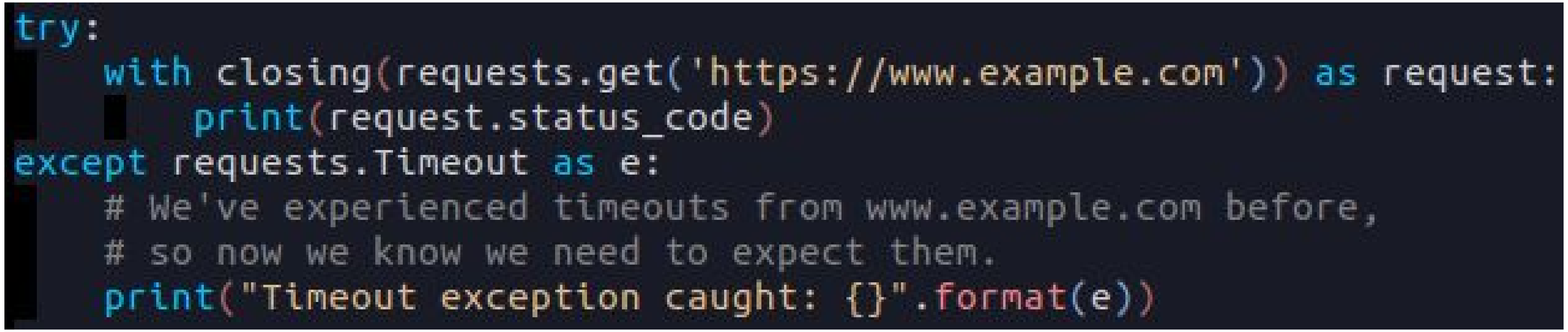


```
import requests
from contextlib import closing

try:
    with closing(requests.get('https://www.example.com')) as request:
        print(request.status_code)
except requests.Timeout as e:
    print("Timeout exception caught: {}".format(e))
```

Self Documenting Code

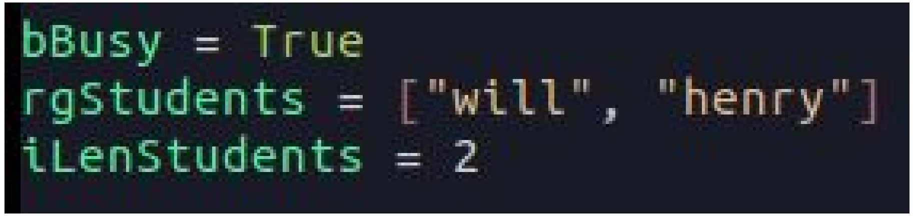


```python
try:
    # Make request to www.example.com.
    with closing(requests.get('https://www.example.com')) as request:
        # Print the status code.
        print(request.status_code)
except requests.Timeout as e:
    # Caught exception, log it.
    print("Timeout exception caught: {}".format(e))
```



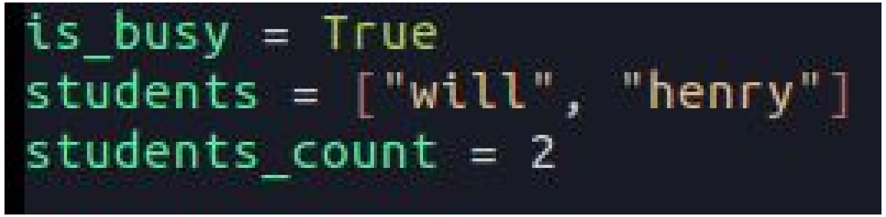
```python
try:
    with closing(requests.get('https://www.example.com')) as request:
        print(request.status_code)
except requests.Timeout as e:
    # We've experienced timeouts from www.example.com before,
    # so now we know we need to expect them.
    print("Timeout exception caught: {}".format(e))
```

Don’t use hungarian notation[4]



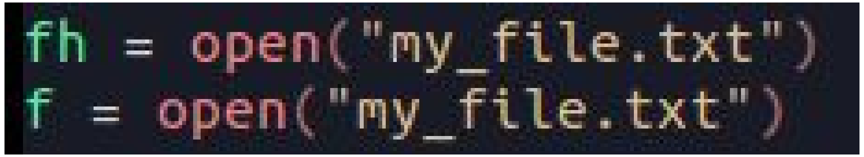
```
bBusy = True
rgStudents = ["will", "henry"]
iLenStudents = 2
```

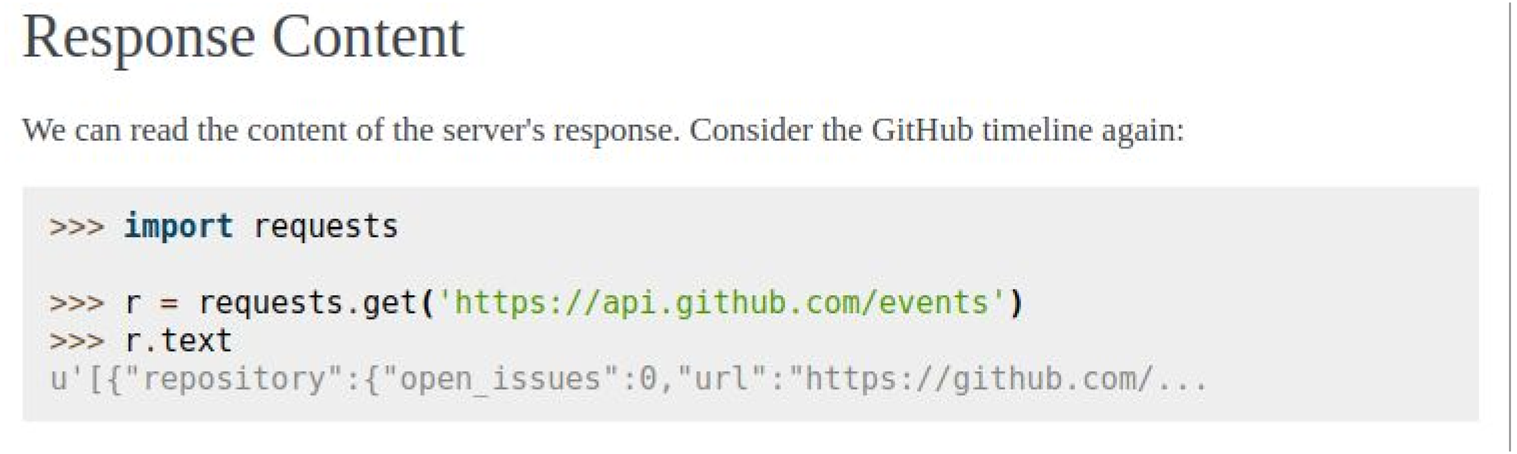
Better:



```
is_busy = True
students = ["will", "henry"]
students_count = 2
```

Don’t make up conventions

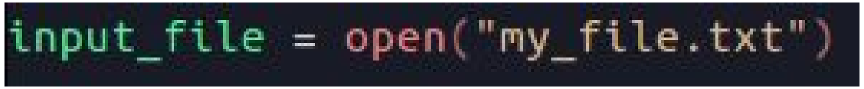

```
fh = open("my_file.txt")
f = open("my_file.txt")
```


Response Content

We can read the content of the server's response. Consider the GitHub timeline again:

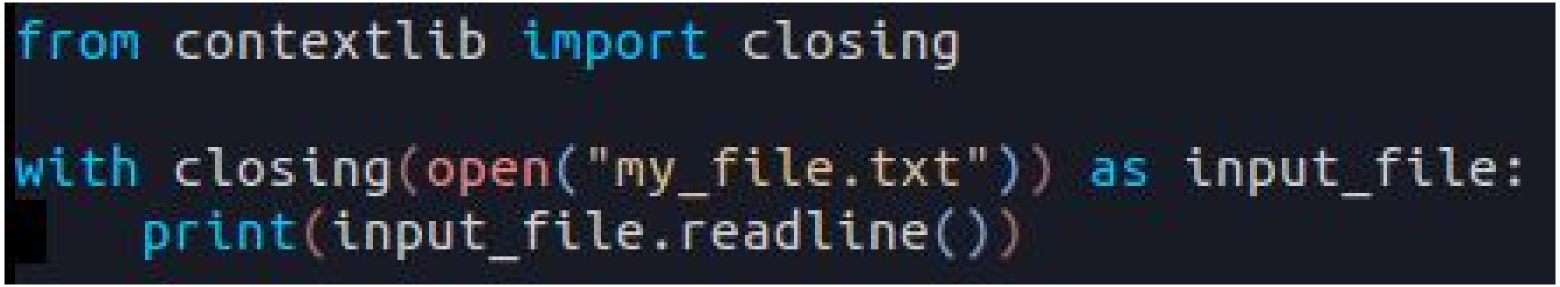
```
>>> import requests

>>> r = requests.get('https://api.github.com/events')
>>> r.text
u'[{"repository":{"open_issues":0,"url":"https://github.com/...
```

Better:



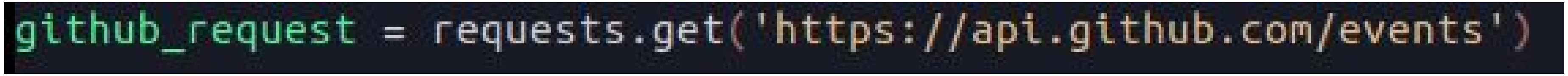
```python
input_file = open("my_file.txt")
```

Best:



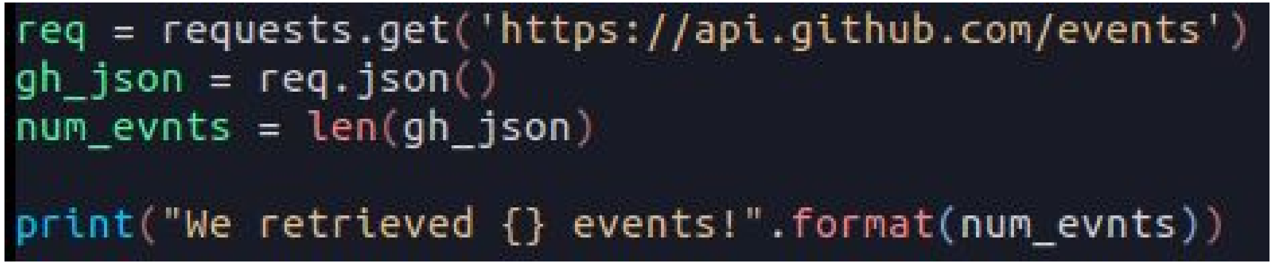
```python
from contextlib import closing

with closing(open("my_file.txt")) as input_file:
    print(input_file.readline())
```

Also better:



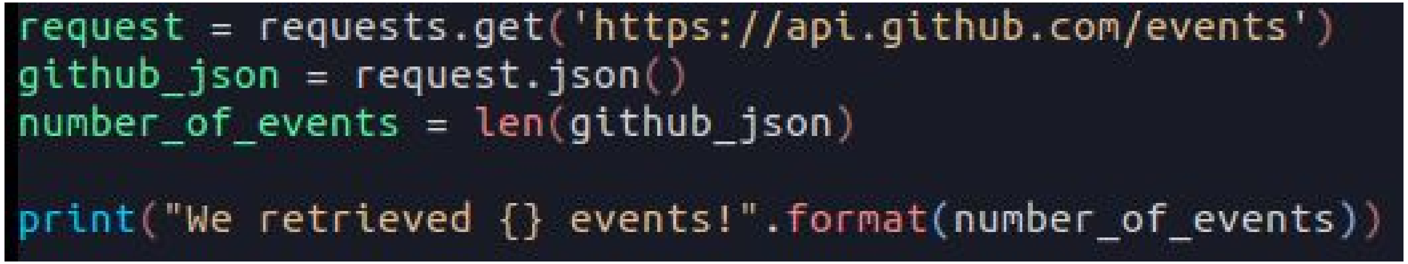
```python
github_request = requests.get('https://api.github.com/events')
```

Don’t shorten words



```
req = requests.get('https://api.github.com/events')
gh_json = req.json()
num_evnts = len(gh_json)

print("We retrieved {} events!".format(num_evnts))
```

Better:



```
request = requests.get('https://api.github.com/events')
github_json = request.json()
number_of_events = len(github_json)

print("We retrieved {} events!".format(number_of_events))
```

However...

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

- pep8

Questions?

References

1. https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext
2. http://mypy-lang.org/
3. https://books.google.com/books?id=tiDkDAAAQBAJ&pg=PA28&lpg=PA28&dq=python+shorthand+conventions&source=bl&ots=lz5lBvhC-h&sig=Kso6GpCbeCfr7YCvlxPOxJvVSeQ&hl=en&sa=X&ved=0ahUKEwjehOaIqMvYAhXHl-AKHZvgCk0Q6AEIUDAE#v=onepage&q=python%20shorthand%20conventions&f=false
4. https://en.wikipedia.org/wiki/Hungarian_notation#Disadvantages
5. https://docs.python.org/3.5/tutorial/inputoutput.html#reading-and-writing-files
6. https://docs.python.org/3.1/howto/urllib2.html#data
7. http://www.pythonforbeginners.com/cheatsheet/python-file-handling
8. http://docs.python-requests.org/en/master/user/quickstart/
9. https://www.python.org/dev/peps/pep-0008/

Further Reading

- These slides can be found at https://github.com/kurtwheeler/talks/python-clarity
- Ottinger's Rules for Variable and Class Naming: http://www.maultech.com/chrislott/resources/cstyle/ottinger-naming.html
- pep8: https://www.python.org/dev/peps/pep-0008/
- JonesComplexity: https://github.com/Miserlou/JonesComplexity