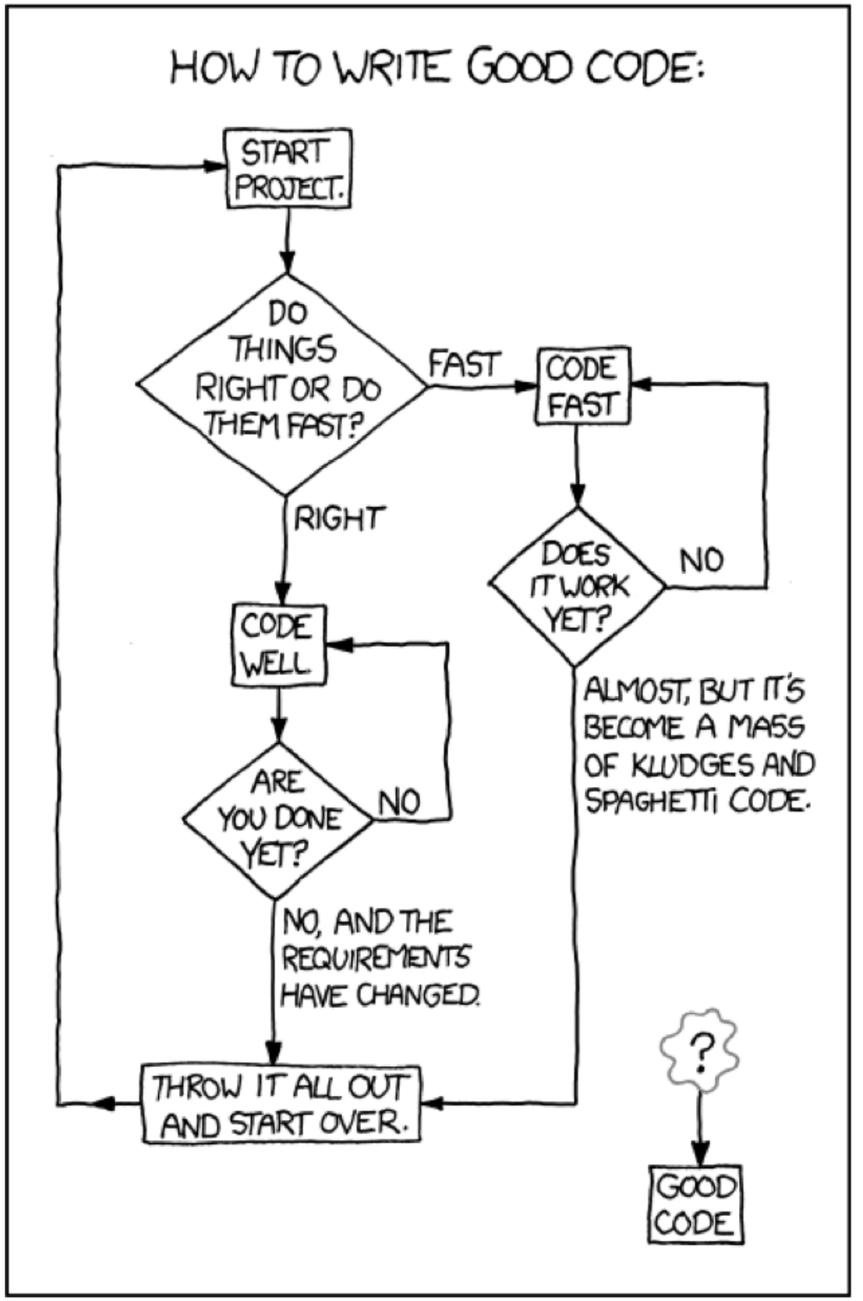
Write good code


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
RIGHT
DOES IT WORK YET?
NO
CODE WELL
ARE YOU DONE YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
NO, AND THE REQUIREMENTS HAVE CHANGED.
?
THROW IT ALL OUT AND START OVER.
GOOD CODE

Why should you care?
An example.

Largest prime factor

Problem 3

The prime factors of 13195 are 5, 7, 13 and 29.

What is the largest prime factor of the number 600851475143 ?

“Here! I’ve solved it.”

```
def fac(n):¬
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)¬
    maxq = long(floor(sqrt(n)))¬
    d = 1¬
    q = n % 2 == 0 and 2 or 3¬
    while q <= maxq and n % q != 0:¬
        q = step(d)¬
        d += 1¬
    return q <= maxq and [q] + fac(n//q) or [n]¬
```

Source: Rosetta Code

- How does this function work?
  - What are those less-than signs doing??
  - What does the variable `d` do?
  - What kinds of input can it take?
  - What kinds of output does it produce?
- What if we needed to modify this function somehow?

```
def fac(n):
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)
    maxq = long(floor(sqrt(n)))
    d = 1
    q = n % 2 == 0 and 2 or 3
    while q <= maxq and n % q != 0:
        q = step(d)
        d += 1
    return q <= maxq and [q] + fac(n//q) or [n]
```

Why should you care?

- The ultimate goal of all Data Science is **communication**.
- This doesn't mean that all Data Science achieves that goal.

```
def fac(n):
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)
    maxq = long(floor(sqrt(n)))
    d = 1
    q = n % 2 == 0 and 2 or 3
    while q <= maxq and n % q != 0:
        q = step(d)
        d += 1
    return q <= maxq and [q] + fac(n//q) or [n]
```

Why should you care?

- The ultimate goal of all Data Science is **communication**.
- Between
  - You and the engineers who will deploy and maintain your models.

```
def fac(n):¬
····step = lambda x: 1 + (x<<2) - ((x>>1)<<1)¬
····maxq = long(floor(sqrt(n)))¬
····d = 1¬
····q = n % 2 == 0 and 2 or 3¬
····while q <= maxq and n % q != 0:¬
····|···q = step(d)¬
····|···d += 1¬
····return q <= maxq and [q] + fac(n//q) or [n]¬
```

Why should you care?

- The ultimate goal of all Data Science is **communication**.
- Between
  - You and the engineers who will deploy and maintain your models.
  - You and other Data Scientists who will interact with and maintain your code.

```
def fac(n):¬
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)¬
    maxq = long(floor(sqrt(n)))¬
    d = 1¬
    q = n % 2 == 0 and 2 or 3¬
    while q <= maxq and n % q != 0:¬
        q = step(d)¬
        d += 1¬
    return q <= maxq and [q] + fac(n//q) or [n]¬
```

Why should you care?

- The ultimate goal of all Data Science is **communication**.
- Between
  - You and the engineers who will deploy and maintain your models.
  - You and other Data Scientists who will interact with and maintain your code.
  - You and your bosses/company leadership/whomever you will communicate results to.

```
def fac(n):¬
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)¬
    maxq = long(floor(sqrt(n)))¬
    d = 1¬
    q = n % 2 == 0 and 2 or 3¬
    while q <= maxq and n % q != 0:¬
        q = step(d)¬
        d += 1¬
    return q <= maxq and [q] + fac(n//q) or [n]¬
```

Why should you care?

- The ultimate goal of all Data Science is **communication**.
- Between
  - You and the engineers who will deploy and maintain your models.
  - You and other Data Scientists who will interact with and maintain your code.
  - You and your bosses/company leadership/whomever you will communicate results to.
  - You and yourself, months from now, when you will need to revisit this code.

```
def fac(n):¬
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)¬
    maxq = long(floor(sqrt(n)))¬
    d = 1¬
    q = n % 2 == 0 and 2 or 3¬
    while q <= maxq and n % q != 0:¬
        q = step(d)¬
        d += 1¬
    return q <= maxq and [q] + fac(n//q) or [n]¬
```

Why should you care?

- The ultimate goal of all Data Science is **communication**.

Good code *isn't* written for a computer to run.

Good code *is* written for other people to understand.

```
def fac(n):¬
    step = lambda x: 1 + (x<<2) - ((x>>1)<<1)¬
    maxq = long(floor(sqrt(n)))¬
    d = 1¬
    q = n % 2 == 0 and 2 or 3¬
    while q <= maxq and n % q != 0:¬
        q = step(d)¬
        d += 1¬
    return q <= maxq and [q] + fac(n//q) or [n]¬
```

How do we write good code?

How do we write good code?

- Use a style guide.
  - Seriously, click on one of the links below and read. Refer back to it.
  - Recommended: Google Python Style Guide
  - Others
    - PEP8: the official Python Style Guide (not maintained)
    - PEP257: Docstring Style Guide

nasa

How do we write good code?

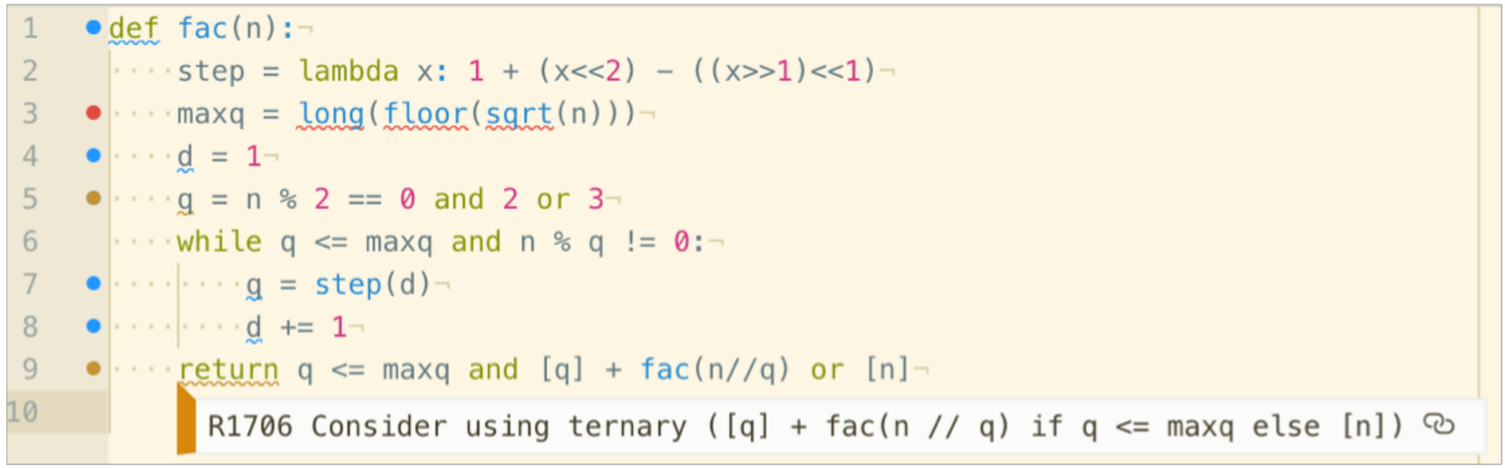
- Use a style guide.
- Use a Linter.
  - Linters scan your code and mark any problems. It’s like spell check for code.
  - Great for catching errors as well as style problems.
  - Pylint is recommended by Google Style Guide but there are many others.
  - Can be run from command line

```
[> pylint test.py                                                                10:29:51]
************* Module test
test.py:1:0: C0111: Missing module docstring (missing-docstring)
test.py:1:0: C0103: Argument name "n" doesn't conform to snake_case naming style (invalid-name)
test.py:1:0: C0111: Missing function docstring (missing-docstring)
test.py:3:11: E0602: Undefined variable 'long' (undefined-variable)
test.py:3:16: E0602: Undefined variable 'floor' (undefined-variable)
test.py:3:22: E0602: Undefined variable 'sqrt' (undefined-variable)
test.py:4:4: C0103: Variable name "d" doesn't conform to snake_case naming style (invalid-name)
test.py:5:4: R1706: Consider using ternary (2 if n % 2 == 0 else 3) (consider-using-ternary)
test.py:5:4: C0103: Variable name "q" doesn't conform to snake_case naming style (invalid-name)
test.py:7:8: C0103: Variable name "q" doesn't conform to snake_case naming style (invalid-name)
test.py:8:8: C0103: Variable name "d" doesn't conform to snake_case naming style (invalid-name)
test.py:9:4: R1706: Consider using ternary ([q] + fac(n // q) if q <= maxq else [n]) (consider-usin
g-ternary)

--------------------------------------------------------------------
Your code has been rated at -16.67/10 (previous run: -16.67/10, +0.00)
```

How do we write good code?

- Use a style guide.
- Use a Linter.
  - Linters scan your code and mark any problems. It's like spell check for code.
  - Great for catching errors as well as style problems.
  - Pylint is recommended by Google Style Guide but there are many others.
  - Can be run from command line
  - Find a plug-in for your favorite text editor to get feedback as you code.



How do we write good code?

- Use a style guide.
- Use a Linter.
- Use a Beautifier.
  - A beautifier (aka formatter) automatically makes changes to your code to improve style.
  - Only makes “safe” changes that will not affect code. You’ll have to rename the variables.
  - Can be used from command line or as a plug-in.

Style Guide Highlights

Let’s dig into the Google Style Guide for a second.

Style Guide Highlights

- Docstrings
  - A function must have a docstring, unless it meets all of the following criteria:
    - not externally visible
    - very short
    - obvious
  - Python tools automatically use the docstrings you write for custom functions.
    - Shift-tab in jupyter

```
1  def my_func(first_arg, second_arg=10):
2      """Adds together the first argument and the second argument.
3
4      Args:
5          first_arg: this can be any number
6          second_arg: this can be any number (default 10)
7
8      Returns:
9          args_sum: the sume of the two arguments
10     """
11
12     return first_arg + second_arg
```

executed in 7ms

```
1  my_func()
```

```
Signature: my_func(first_arg, second_arg=10)
Docstring:
Adds together the first argument and the second argument.

Args:
    first_arg: this can be any number
    second_arg: this can be any number (default 10)

Returns:
    args_sum: the sume of the two arguments
```

Style Guide Highlights

- Docstrings
- Inline comments
  - Do: Explain tricky parts of the code. If you'll have to explain it at the next code review, use a comment now.
  - Don't: describe the code. Assume the person reading the code knows Python (though not what you're trying to do) better than you do.

```
# We use a weighted dictionary search to find out where i is in
# the array.  We extrapolate position based on the largest num
# in the array and the array size and then do binary search to
# get the exact number.

if i & (i-1) == 0:  # True if i is 0 or a power of 2.
```

```
# BAD COMMENT: Now go through the b array and make sure whenever i occurs
# the next element is i+1
```

Style Guide Highlights

- Docstrings
- Inline comments
- Good code speaks for itself
  - No need to write separate documentation.
  - No need to for author to explain what's going on.

Public domain

Code Style in the Bootcamp

Code Style in the Bootcamp

- Jupyter
  - Jupyter is our main tool for teaching and sharing code.
  - Jupyter makes it difficult to use code style tools.
  - Recommendations:
    - Pylint doesn't work, though nblint is a command line stand in.
    - Jupyter extensions includes two beautifiers.

nasa

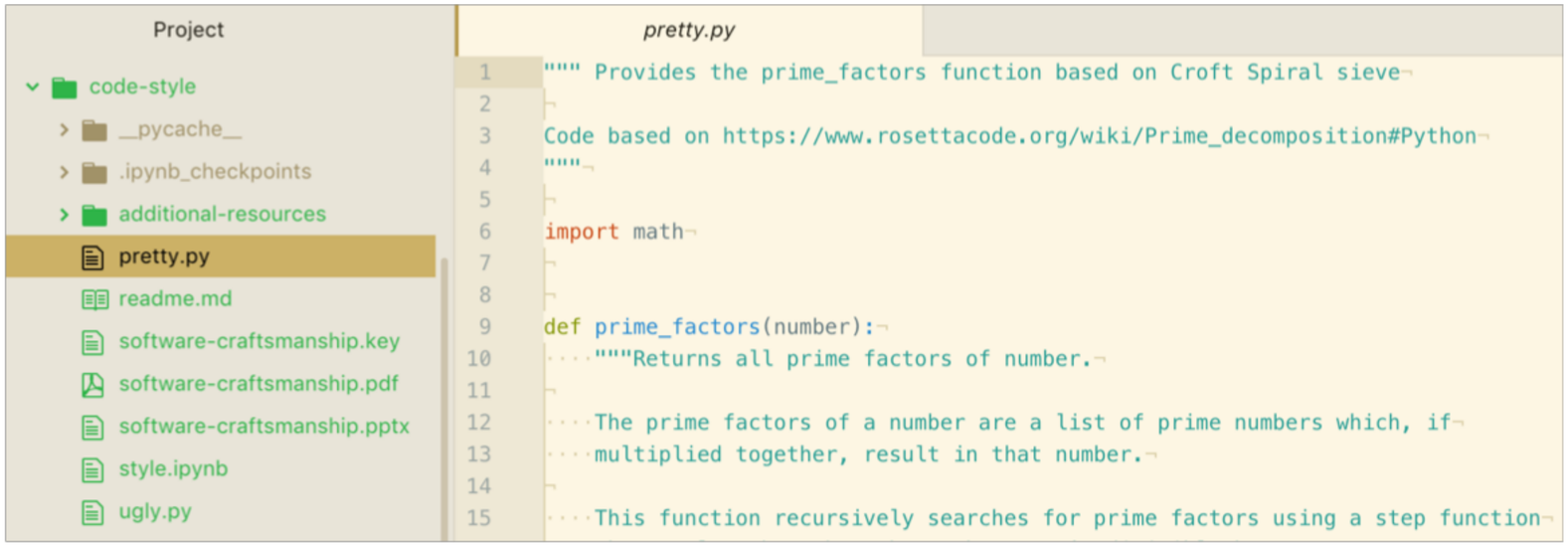
Code Style in the Bootcamp

- Jupyter
- Jupyter vs External code
  - Use Jupyter for tying together a workflow in an an easy to read format.
  - Use external code for writing complex functions that you will call from Jupyter.

nasa

Code Style in the Bootcamp

- Jupyter
- Jupyter vs External code
- External code
  - If a function you're writing is > 10 lines, consider breaking it out into an external .py file.



Code Style in the Bootcamp

- Jupyter
- External code
  - If a function you're writing is > 10 lines, consider breaking it out into an external .py file.
  - Then you can import it to your Jupyter notebook

```
1  from pretty import prime_factors
```

executed in 11ms

```
1  prime_factors(873)
```

executed in 19ms

```
[3, 3, 97]
```

Write good code

Write good code

- Code is for communication
- Use a style guide and style tools
- Find a balance between Jupyter and external code
- Good code speaks for itself