Software Engineering with Python

Isaac Laughlin

October 30, 2017

Outline

Objectives

Development tools

Writing clean code

Pair Programming

Objectives

At the end of this lecture you should.

- ☐ Be able to correctly choose different development tools, IDEs, Text Editors, Linters
- ☐ Use commands in various environments to get help
- ☐ Name several aspects of clean code.
- ☐ Know best uses of built-in types for efficient code.
- ☐ Be able to start a paired programming exercise.

Development tools

Terminal

- How to get help? `man command`
- Sometimes you'll get `No manual entry for command`. Then try `command --help`

I do

How do I find out what the `xargs` program does?

We do

How do we find out what the command `cat -n ~/.emacs.d/init.el` does?

You do

How would you find out what are the subcommands available for the `jupyter` command? Try it!

IPython

IPython is an enhanced interactive shell for Python.

You can think of it as a special program that communicates to a Python process for you, and gives you extra functionality.

How to get help?

- `help([object])`
- `?object` (only for IPython, but better)
- Tab completion!
- Online docs

I do

I know `map` takes an iterable and a function, but which one comes first?

We do

I know `itertools` has a function for doing combinations with replacement, what's it called?

You do

What kinds of things can I pass as the first argument to `pandas.Series`?

Atom

Or another editor that you're already good at.

Really useful things that you should install in your editor:

- `linter-pycodestyle`
- `autocomplete`
- Others according to your taste! Some people like Hydrogen.

Jupyter Notebooks

Great for:

- Exploratory data analysis (EDA)
- Demonstrations

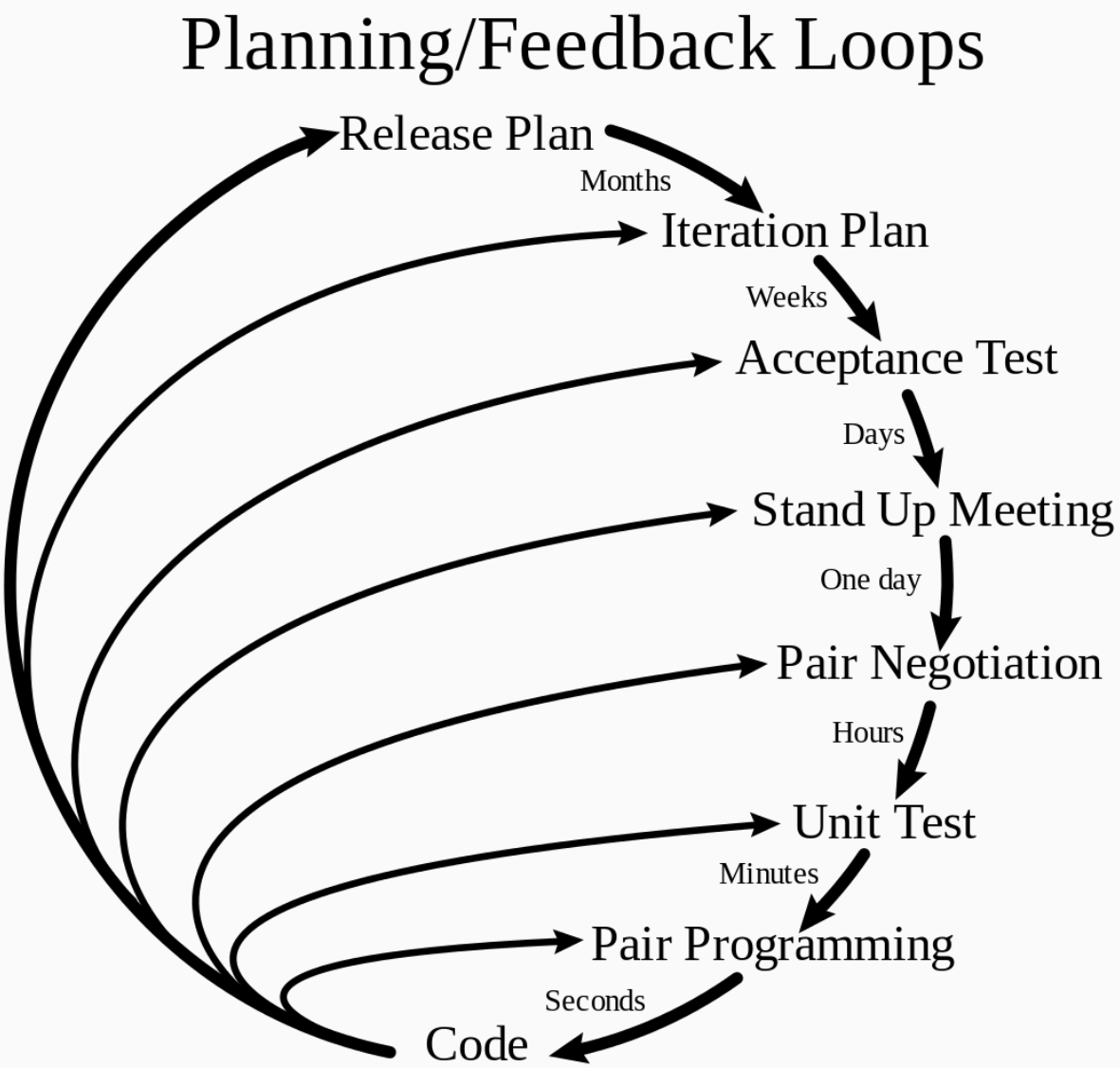
Bad for:

- Robust, reusable, code that works.

Conclusions:

- Only use notebooks for exploration or explanation.
- Write `.py` files and run them using `python` or Ipythons `%run`.

Feedback!


Planning/Feedback Loops
Release Plan
Months
Iteration Plan
Weeks
Acceptance Test
Days
Stand Up Meeting
One day
Pair Negotiation
Hours
Unit Test
Minutes
Pair Programming
Seconds
Code

Tests

Why test?

- Tests are an important tool for verifying your code performs as expected.
- Can be unit tests or integration tests.
- Usually used with a test framework like `nosetests` or `unittests`

Kinds of tests

Unit tests

- Done in isolation, tests the smallest possible functionality.
- e.g. does adding work? does adding negative numbers work? does adding float to int work? etc.
- Every major open source project definitely uses these.

Integration tests

- Tests that the object or module works as required in conjunction with other code.
- May not be used as frequently.

Unit vs. Integration Tests

Tests here

- We write some to verify (or for you to verify) your work. USE THEM.
- You could write some too!
- For heavy duty data-driven code, it can be hard to write tests, but still useful.

Writing clean code

Style and Structure

```
t=1*10**-10**2 m=1*10**2
def f(f, f1, q, t, m):
    i=0
    while ((f(q) > t) and (i < m)):
        i,q=i+1,q-f(float(q))/f1(q)
    return q
f2 = lambda x:x**2
f3 = lambda x:2*x
print( f(f2, f3, 10, t, m) )
```

vs.

```
def find_zero(f, f_prime, x,
              threshold=1E-100, max_iter=1E100):
"""Finds the zero of a function f, given its derivative
function f_prime, using the Newton-Raphson method:
"""
    x = float(x)
    iterations = 0
    while f(x) > threshold and iterations < max_iter:
        iterations += 1
        x = x - f(x)/f_prime(x)
    return x
if __name__ == "__main__":
    def f(x): return x**2
    def f_prime(x): return 2*x
    initial_guess = 10
```

Principles of Style and Structure:

- Code is read more than it is written (even by you)
- Good code is reusable, therefore is structured into components that would support that.
- DRY (don't repeat yourself) is better than WET (we enjoy typing)
- Good programmers do this, and recognize one another by their doing
- By following conventions, code is comprehensible to everyone
    - The convention almost everyone has agreed to follow is called "pep8"

Writing efficient code

- Efficient code is sometimes at odds with the readability objectives.
- Optimizing code can be the difference between code that executes immediately or that never finishes.
- "Runtime Analysis" is popular interview topic.
- Good programmers know the basics of efficiency and use them by default.

Generators

- What is the difference between `range` and `xrange` in Python 2?
- What is the difference between `range` in Python 2 and `range` in 3?
- Would you use `range` or `xrange` by default? Why?

Loops

Critique these loops

```
for item in collection:
    do_stuff(item)

for position, item in enumerate(collection):
    award_medals(position, item)

for i in xrange(len(collection)):
    you_should_feel_bad(collection[i])

for first_name, last_name in zip(first_names, last_names):
    fullname = ' '.join([first_name, last_name])
```

Dictionaries

Why is one of these better than the other?

```
homestate = [("giovanna", "maine"), ("ryan", "california"),
             ("katie", "michigan"), ("zack", "new york")]
[x[1]] for x in homestate if x[0]=='ryan']
```

```
homestate = {"giovanna": "maine", "ryan": "california",
             "katie": "michigan", "zack": "new york"}
homestate['ryan']
```

Dictionary operations

Looping

Good

- `for key in dictionary`
- `for key, value in dictionary.iteritems()`

Bad

- `for key in dictionary.keys()`
- `for key, value in dictionary.items()`

Equality

- `dict1 == dict2`

You tell me

Checking Membership

Good or Bad?

- `key in dictionary`
- `key in dictionary.keys()`

Sets

- Like a dictionary with no values
- Useful for deduplication and checking membership.
- Sets and Dictionaries are often the secret to high performance solutions.

Mutability

- Mutable objects can change their value but keep the same `id()`
  - lists
  - sets
  - dictionaries
- Immutable objects cannot be altered after creation
  - Especially important in places where constant hashes are needed (like for keys in a `dict`)
- Only immutable types are hashable so only immutable types can be used is sets or as dictionary keys.

Lambda Functions

Function

```
def add(x, y):
    return x + y
```

Lambda Function

```
lambda x, y: x + y
```

An familiar example

Remember anagrams?

Important modules

- Everything we've discussed today is a part of "base python"
- Other functionality is available via the standard library which is included with most distributions.
- Other other functionality is included via 3rd-party packages.
- Important modules in standard library
  - `itertools` Contains combinatoric functionality to create complex groups of things from iterable
  - `collections` Contains useful extensions of base types like `dict`, `list`, `set`. Use one of the ways you know how to get help to find out exactly what's included.
  - `argparse` Tools for reading parameters passed to your script from the command line as variables.
  - `timeit` Module for timing how long your code takes to run.
  - `json` Module for dealing with JSON files.

Pair Programming

What is it?

- One computer, 2 keyboards and 2 monitors.
- One Driver and One Navigator

Why?

- Learn more
- Practice speaking data science
- Fewer errors
- Working with different skill sets and personalities

Go do it!

NOW