Teaching Python:
The Hard Parts

Elana Hashman
Rackspace Managed Security
PyCon 2016 – Portland, OR

Background

Community Data Science Workshops

Python Workshops for Beginners

Total Beginners

Platform Diversity

- Majority of mentors use OS X or Linux machines

- Majority of students use Windows machines

- Mentors may not have the experience to diagnose common Windows-specific issues

  - "python.exe not found" (PATH problems)

  - String encoding for unicode on the console

  - Binary files and line ending conversions

Platform Diversity

```python
# Example from PWFB

from urllib2 import urlopen

site = urlopen('http://placekitten.com/250/350')

data = site.read()

kitten_file = open('kitteh.jpg','w')

kitten_file.write(data)

kitten_file.close()
```

Platform Diversity


```python
# demon kitty!

open('kitteh.jpg','w')
```



```python
# normal kitty

open('kitteh.jpg','wb')
```

Platform Diversity

- **Takeaway:**

  – *You must anticipate cross-platform issues for your participants*

  – *Make sure you test your examples on multiple platforms, especially Windows*

The Command Line

- Most Python tutorials start by running `python` or `ipython` on the command line

- Most total beginners have never used the command line before

- We don't tend to spend a lot of time teaching about the OS shell before jumping into the Python shell

The Command Line

- Beginners get confused between shells, typing OS commands into the Python shell and vice versa

```
me@mylaptop:~$ python

Type "help", "copyright", "credits" or "license"
for more information.

>>> ls

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined
```

The Command Line

- **Platform Diversity:** Windows users may need to use `dir` instead of `ls` to list files; most OS shell tutorials don't cover these users

- **Takeaway:**
  - When teaching your students to interact with the Python shell, first spend some time talking about the OS shell
  - Teach students specific commands for each shell
  - Show students how to differentiate between them

Python 2 vs. Python 3

- "What version do I install?"

- Worse: no one asks and the whole class has installed a variety of different versions of Python

- Python 3 libraries are incompatible with Python 2 and have different documentation

- Search engine indexing is not as good for Python 3 stuff, so beginners may accidentally fetch the wrong docs

Python 2 vs. Python 3

- **Takeaway:**
    - *Making everything Python 2/3-compatible can take your time away from curriculum-building and may end up confusing beginners*
    - *Advice:* Pick one version of Python that's right for you and your group
    - *Be aware of Python versioning at install time*
    - *Make sure your entire class uses the same version of Python uniformly*

A little more advanced

Methods vs. Functions, OOP

- "Why do we write `foo.keys()` but `range(10)`? Why not `keys(foo)`?"
  - "One is a function and one is a method"
- "When do I use `foo.sort()` versus `sorted(foo)`?"
  - "One mutates foo and the other doesn't"
- Trying to explain this to beginners can overwhelm them
- Students don't have the tools to understand this yet

Methods vs. Functions, OOP

- **Takeaway:**
  - Different syntax for invoking subroutines can be confusing to beginners and mentors should be aware of this

  - This is a good point to start introducing students to documentation, to clarify what syntax to use

  - *Advice:* don't introduce OOP to students that don't have prior programming experience. Or, put it near the end of your curriculum

Testing

- Tests are a common blind spot for curriculum

- The later you introduce testing, the more optional it seems to your students

- You want to fit as much shiny and cool stuff as possible!

  - Testing = "eating your vegetables"

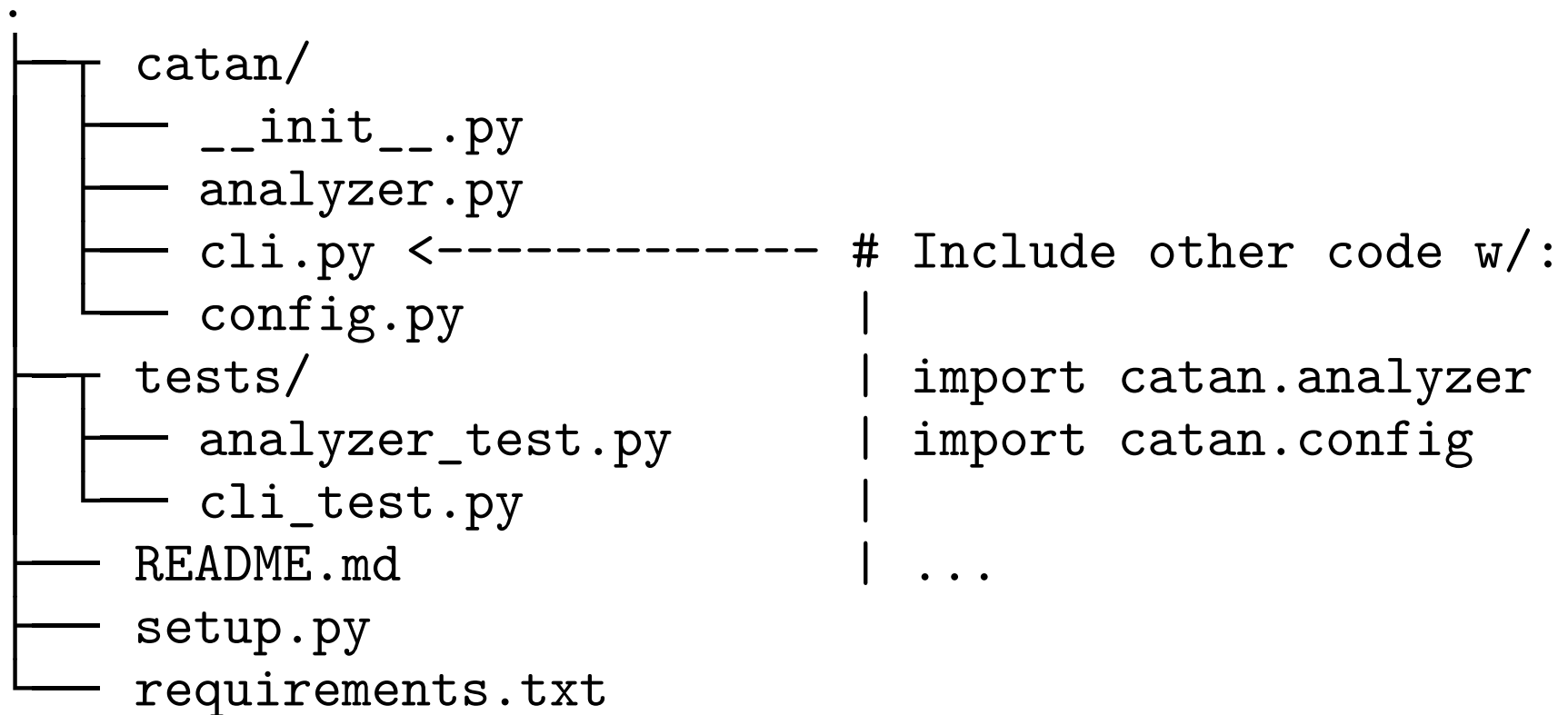- Is this something that has a place in curriculum for complete beginners?

Testing

- `unittest` is easy to set up and get working

- Tests are a powerful tool for reasoning about the correctness of code and building confidence as a developer

- **Takeaway:**

  - Consider including some curriculum on testing

  - Lead by example: include tests in your sample code

Putting Together Modules

- Students in workshops are usually taught how to work with the Python interpreter or single files

- What do you do when code gets too big to fit in a single file?
  - Ask a mentor
  - Despair

- Documentation is poor

- Version differences can make this very challenging on multiple axes

Putting Together Modules

- Teach your students by example. Provide templates:

```
.
├──┬── catan/
│  ├── __init__.py
│  ├── analyzer.py
│  ├── cli.py <------------- # Include other code w/:
│  └── config.py            |
├──┬── tests/               | import catan.analyzer
│  ├── analyzer_test.py     | import catan.config
│  └── cli_test.py          |
├── README.md               | ...
├── setup.py
└── requirements.txt
```

Putting Together Modules

- We can help ease the documentation gap/student confusion by covering this

- This is a common curriculum gap, possibly because it's so "obvious"

- **Takeaway:**

  - If you want students to walk away from your workshop with the ability to ship working software, you should cover this topic or provide future resources

Intermediate students

Scoping

- Intermediate students will ask questions about Python's scoping to learn how to reason about their code

  - Is it lexical? Dynamic? Something else?

- Let's go over a common, confusing example

Scoping

```python
cat = 'meow'

def cat_changer():

    cat = 'purr'

    print 'inside cat: ', cat

cat_changer()

print 'outside cat: ', cat

# => inside cat: purr
# => outside cat: meow
```

Scoping

- **Putting Together Modules:** sharing state between two files can be tricky because of scoping rules

- *Advice:* reference shared state using caution and fully-qualified namespaces

  - Good pattern: have your students create a `config.py` package that stores all shared global state

Scoping

```python
# catan/config.py

CAT_DB = 'postgres://localhost:5555'

CAT_LOG = '/home/catlover/var/log/cat.log'


# catan/cli.py

def main():

    # ...
    catan.analyzer.run_analyzer(catan.config.CAT_DB,
                                catan.config.CAT_LOG)
```

Scoping

- **Takeaway:**
    - Python's scoping rules can be tricky for even experienced programmers new to the language
    - Try to cover the rules in detail and cover "heads-up" scenarios where students may run into trouble
    - Guide your students on how to use global variables/state

Packaging and Deployment

- Students want to ship their code and see it in action!

  - "How do I write a web app in Python?"

  - "A mobile app?"

  - "How do I package and deploy a command-line Python application?"

  - "How do I write a Python service/daemon?"

- *Maybe abandon hope*

Packaging and Deployment

- There are lots of different moving parts to packaging and developing Python software
    - Learning to navigate `setuptools` and `setup.py`
    - Package managers: `pip/easy_install/conda`?
    - Virtual environments for development: `virtualenv` vs. `pyvenv`
- This is important operational knowledge for new Python programmers
- *Advice:* Sharing "one true way" for your students is better than confusing them with too many options

Packaging and Deployment

- Okay, we know how to develop the software and how to package it at a Python-level; let's deploy it

- How do we address dependency management?
  - At the system or user-level? What about OS-level dependencies?

- What about deployment processes?
  - git and pip?
  - Docker?
  - PEX?
  - dh-virtualenv and Debian packages?

Packaging and Deployment

- **Takeaway:**
  - If you have the time, briefly walk through `setup.py` and `setuptools` for building packages
  - If you work with external libraries and installing them is in scope of your workshop, cover virtual environments
  - Deployment: walk your students through one option that makes sense for their background and will enable them to sustainably run their software

For all levels

General advice

- Be aware of your own shortcomings

- Less is more

- Bring together a supportive educational team

- Seek feedback from your students and mentors

- Treat your students with patience, empathy, and respect

- Encourage your students to build community

Questions?

Thank you!

Thanks to: Peter Barfuss, Murphy Berzish,
Fatema Boxwala, Paul Kehrer, Rackspace

Talk links and resources can be found at
https://hashman.ca/pycon-2016/