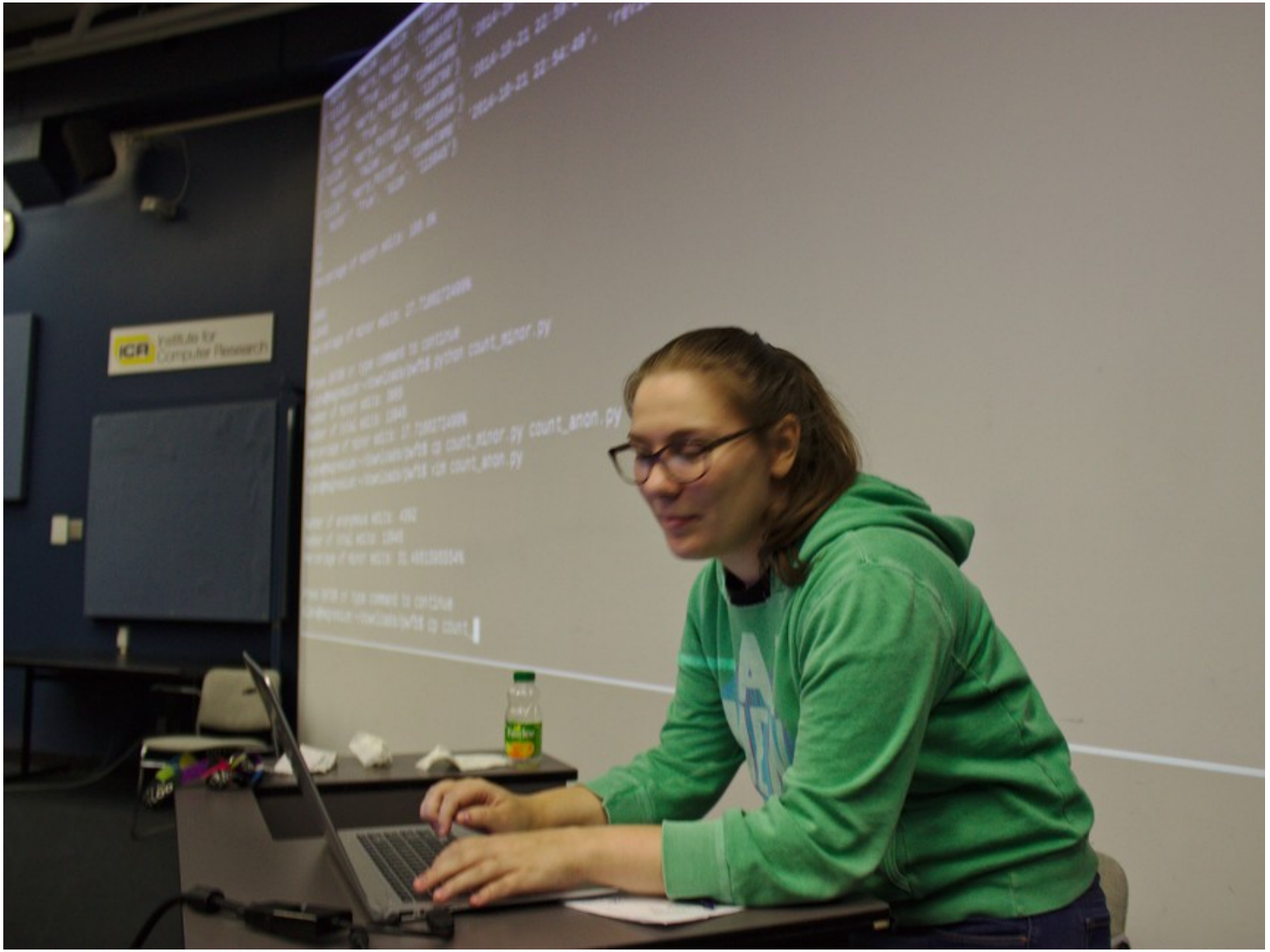
Teaching Python: The Hard Parts

Elana Hashman
Rackspace
PyCon 2016 – Portland, OR

Background

Community Data Science Workshops

Python Workshops for Beginners



Total Beginners

Platform Diversity

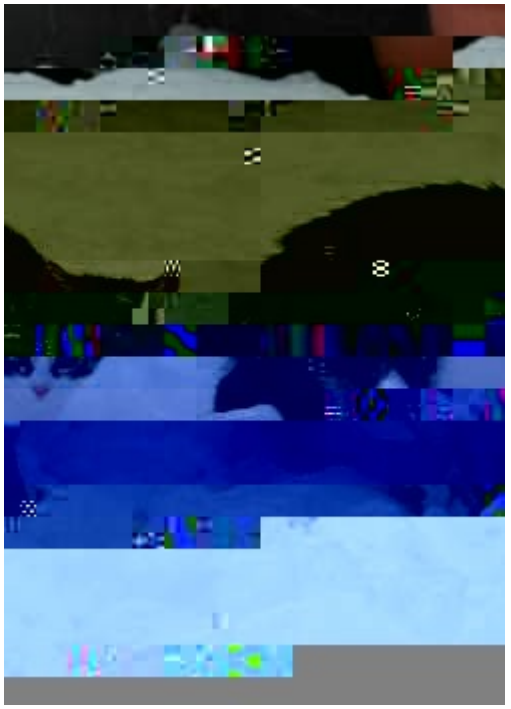
- Majority of mentors use OS X or Linux machines
- Majority of students use Windows machines
- Mentors may not have the experience to diagnose common Windows-specific issues:
  - "python.exe not found" (PATH problems)
  - Binary files and line ending conversions
  - String encoding for unicode on the console

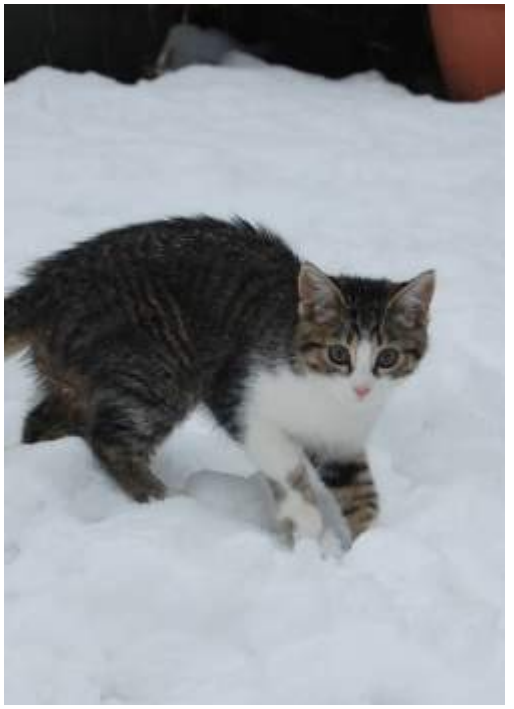
Platform Diversity

- 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





```
# demon kitty!

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

```
# 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 the two shells, typing OS commands into the Python shell and vice versa

```
$ 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:**
  - If you are going to teach your students to interact with the Python shell, first spend some time talking about the OS shell
  - Teach students specific commands for each
  - Teach students how to differentiate between the two shells

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
- SEO 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"
- Students don't have the tools to understand this yet
- Trying to explain this to beginners can overwhelm them

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

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?

Putting Together Modules

- This is my cat analyzer Python package, called 'catan'

```
.
├── 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

- Another gotcha: what if we need to import an external module called `analyzer` within `analyzer.py`?
  - Older versions of Python attempt to resolve the name locally first
  - Errors are confusing and demotivating to students
- We can fix this with

```
from __future__ import absolute_import
```

Putting Together Modules

- There are not a lot of walkthrough-style resources available for structuring Python code
- 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?
- Well...

Scoping

```
cat = 'meow'

def cat_changer():

    cat = 'purr'

    print 'inside cat: ', cat

cat_changer()  # doesn't work!

print 'outside cat: ', cat

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

Scoping

```
cat = 'meow'

def cat_changer():
    global cat
    cat = 'purr'
    print 'inside cat: ', cat

cat_changer()  # works!

print 'outside cat: ', cat

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

Scoping

- **Putting Together Modules:** sharing state between two files can be tricky because of global/function-level scoping rules
- Can't use the **`global`** keyword
  - This only works at the namespace level, i.e. across that file
- *Advice:* reference shared state using fully-qualified namespaces. Good pattern: have your students create a `config.py` package that stores all shared global state

Scoping

```
# 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 scenarios where students will run into trouble
  - Give advice for best practices for sharing global 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
  - When it comes to deployment, walk your students through an option that works well for your them and your own background

Questions?

Thank you!

Thanks to: Peter Barfuss, Fatema Boxwala, …
‹your name here›