
# Getting started in open source software

Trish Gillett-Kawamoto

Pyladies Montreal

May 27, 2016

PyLadies beamer template by PyLadies Barcelona

# Dissecting Github communities

# Anatomy of a github project



In addition to code, each Github project also has:

- An issues board
- A pull request board
- A README*
- A wiki*

* optional

# What is the issues board?



People create issue tickets in order to report possible bugs or request new features. All issues are public, and can be created by anyone and commented on by anyone.

**Issue discussion covers:**

- user perspectives
- is it a bug or not? is it a feature worth adding?
- how to approach fixing it?
- how to prioritize it? who wants to work on it?

An issue is closed after it's been fixed or when the group concludes no action is needed.

# Fun fact!



Due to a misbehaving keyboard, the first comment I ever made on an open source project issue was "Hey guys, I'm going to try working on thisttttttttttttttttttjklsatjskdlj".

Let's set the bar there. If you're making more intelligent comments than that, congratulations, you're winning at Github!

# Issue wrap up



We can think of the issue board as a communal to do list plus comments.

To finish with issues, let's look at one...

Actually, this is a hands on workshop, *let's create one.*

* I prepared a real issue in advance, don't submit fake issues for practice. ;)

# What is the pull request board?



A pull request (PR) asks for changes on a branch of one repo to be applied to a branch of another repo.

In our case, when your work is ready, you'll make a PR from a working branch of your repo to the master branch of the OSS project's main repo.

When a pull request is made, it creates a PR ticket which is public and can be commented on.

**Comments on PRs cover:**

- discussion of what you've done and how
- cross-references to relevant issues
- suggestions for tweaks on the code

# Tip: Pull requests are iterative!



Your pull requests can be updated by pushing updated code to the branch the PR is based on!

**Some nice things:**

- That branch is your territory, people will make suggestions but you make any corrections yourself.
- For very drastic changes, you can make a PR mid-process to show where you're at and get feedback. Put [WIP] in the title.
- If the PR's goal is good and you're able to make the changes asked for, your PR can have a happy ending.

# Pull request wrap up



The pull request board shows all the pull requests awaiting action by either the core devs or their creators.

**Let's look at a couple pull requests:**

- *A straight forward one*
- *One with discussion and revision*

# Pop quiz: What's missing from the Github community as presented so far?

# Answer: A place for general conversation



An organized community needs a place for open discussion of general topics:

- coordinating big picture plans
- asking for advice and sharing tips
- sharing news and making general inquiries

This element is an important part of the ecosystem for a large OSS project. If it's not built into Github, where is it?

# Tip: Figure out where the project talks



Because sometimes you just need to ask a question.

**Groups might use one or more of:**

- chat rooms, ex. Gitter (*Matplotlib*)
- mailing lists / newsgroups
- social media
- e-mail between core devs
- in person sprints or summits

* Large projects may use several of these, very small projects may only use personal communication.

# Tip: Engage in some light stalking (projects, not people!)



**Start listening to the conversations that are happening among users and developers:**

- 'Watch' the project on Github (if it's not too active)
- Sign up for any mailing lists / Google groups
- Join any chat rooms

* These are also great ways to become a power user!

# Some OSS projects are more open than others



Even if the code is open source, not every OSS project owner *wants* to manage a development community.

That's fair, but it's easier and more rewarding to work on a project that has resources to support you.

**Good signs a project is ready for you:**

- A readme, wiki, install guide, developer guide, etc.
- Thorough use of issue labels: **easy, newcomer-friendly, help-welcome, low-hanging-fruit**
- It's possible to talk directly with a core dev.

# Other qualities to look for in a project



- Responsiveness
- A product you're willing to learn about
- Overlap with your interests and skills

# Finding tasks to match your skills

# Work exists at all levels



Developing mature software involves work at all levels!

**Every project wants to have:**

- Consistent style and *PEP8* compliance
- Well documented code
- Well tested code
- User guides, tutorials, installation instructions, developer guides, etc.

Everyone agrees this work is important, and it can actually be very newcomer-friendly.

# The paradox



**In a typical large project, which will be completed first?**

a bug fix requiring time, intense effort,
and deep knowledge of the project,
or
a documentation improvement initiative
requiring time but no specific skills?

**The bug fix. Why?**

- The docs task is non-urgent and open ended
- The core devs have to prioritize by urgency
- Newer less experienced devs probably haven't heard of the docs initiative

# Can I be a valuable contributor?



Don't ask yourself this question:

X "Is there something I can do that the other devs can't?"

Ask yourself this one:

✓ "Is there something the group wants to get done that I know how to do (or can learn to do)?"

Show a desire to be involved and be helpful, follow through on what you say you'll do, and in return regular contributors will see mentoring you as a good investment.

# So how can I help if I don't know how to calibrate a hyperwave nanoshell chamber?



## The gist of it is:

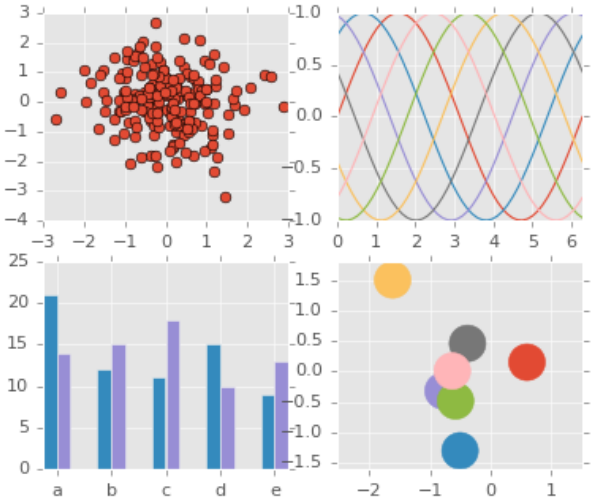
- Find something you can handle:
  - Browse issues by labels, most/least recently active
  - Ask in chat: "Hi, I'd like to start contributing to this project. Can you suggest any relatively newbie-friendly issues?"
- Find something just out of reach and ask for advice:
  - "I'd like to try fixing this but I'm not sure how to _____. Could someone give me some tips to get started?"
- Do some detective work to figure out if there are any goals or initiatives other than those listed as issues, then design your own task.

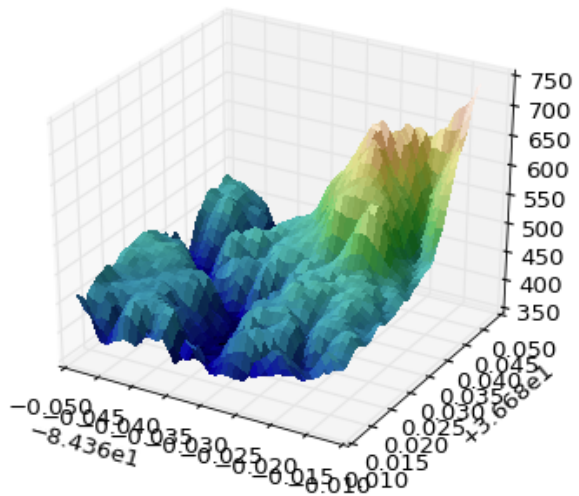
# Matplotlib and MEP12

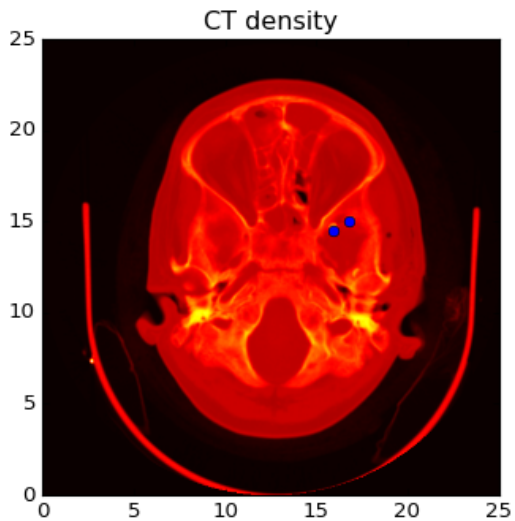
# What is Matplotlib?



‘A python 2D plotting library which produces publication quality figures’.








* http://matplotlib.org/gallery.html

# Tell me more...



- Downloads in the last month (according to PyPI): ?
- All-time forks on github: ?
- All-time contributors to git project: ?
- Contributors in the last year: ?

* As of about April 30, 2016.

# Tell me more...



- Downloads in the last month (according to PyPI): 180k
- All-time forks on github: ?
- All-time contributors to git project: ?
- Contributors in the last year: ?

* As of about April 30, 2016.

# Tell me more...



- Downloads in the last month (according to PyPI): 180k
- All-time forks on github: 1.8k
- All-time contributors to git project: ?
- Contributors in the last year: ?

* As of about April 30, 2016.

# Tell me more...



- Downloads in the last month (according to PyPI): 180k
- All-time forks on github: 1.8k
- All-time contributors to git project: 446
- Contributors in the last year: ?

* As of about April 30, 2016.

# Tell me more...



- Downloads in the last month (according to PyPI): 180k
- All-time forks on github: 1.8k
- All-time contributors to git project: 446
- Contributors in the last year: 50

* As of about April 30, 2016.

# Demographics?



**Approximate gender breakdown of one year of contributors:**

- Presenting female: 3
- Gender-neutral: 6
- Presenting male: 41

(non-scientific count guessing by gender presented in profile picture or use of a typically gendered name)

# Anatomy of Matplotlib



**Other than git issues, where is the action at?**

- *Gitter chat room*
- Mailing list

**And what other community resources exist?**

- *Wiki on Github*
- *MEPs*: Matplotlib Enhancement Proposals
- *Developer Guide*
  (including sections for things like *Documentation*)
- *README*, *INSTALL*

# Tonight's project: MEP12



*MEP12* is a Matplotlib initiative to improve *this example collection*. The collection is huge (500 examples!), but not standardized.

MEP12 sets guidelines for 'cleaning up' examples, and it's really about improving educational content by making it easier to learn from the examples. This means doing things like standardizing style, reorganizing code, and adding comments and descriptions.

*Let's look at a cleanup example!*

# Why MEP12?



## Why choose MEP12 for this hack night?

- Examples are isolated, low risk.
- They don't require much domain-specific knowledge.
- It's rewarding: when users come looking for guidance, they'll see your words!
- Educational value is something many of us understand.

## If it's so great, why is it still ongoing after 3+ years?

- It requires time, thoughtfulness, and subjective judgement calls.
- It's non-urgent, easily forgotten when you're busy with bug fixes and new features.
- You have to dig into the website to find out it exists, so new contributors might not notice it.

# Hack night instructions

# General flow



1. Follow the steps on the meetup event page!
2. Fork matplotlib on github, then clone it to your computer and create a branch.
3. Choose an example to work on from the examples folder. (Coordinate to avoid collisions! ;))
4. Follow the guidelines on *MEP12* and the template on the next slide and edit your example until you're happy with it. Try to get some feedback from the people around you.
5. Pylint your example! `pylint path-to-your-file.py`
6. Commit your code, push to github and make a pull request to matplotlib!

# File header template



```
# -*- noplot -*-          <- only if this line already exists

"""
First line summary.

Other details here.
"""

from future import __xx__          <- only if some future imports already exist

import xxxxxx as xx
from yyyyy import zzz

# rest of the code starts here
```

# Let’s get hacking! :)