Some modest advice about open-source software development *

Jamie Collins
Woods Hole Oceanographic Institution

james.r.collins@aya.yale.edu
@jamesrco
http://jamesrco.github.io

* Presumes you intend to develop something for public consumption at project outset, but most advice useful in retroactive scenario as well

Perspective

- Mostly R (own recent experience developing LOBSTAHS package)
  - https://github.com/vanmooylipidomics/LOBSTAHS
  - http://bioconductor.org/packages/LOBSTAHS
- Includes some resources specific to R development
- But… most advice applies to package development in Python as well

At the beginning…

- Ask yourself up front:
  - Will a loose collection of scripts suffice?
  - Do I really need/want to develop a package?
  - Target audience (size and type) instructive here

If a package it is…

- Again, think about audience
- Version control (Git, Subversion) is critical (adopt early)
  - All your code is probably nicely organized in a repository anyway, right?
  - Facilitates collaboration
  - Makes it easy to walk back mistakes & float new features without breaking everything

If a package it is…

- Code hygiene will be important; don't wait until later to clean up your mess
- **Documentation** may be most time-consuming component, so start early
  - Build docs/manual pages for functions as you go
  - Make them useful (don't do the minimum only to spoof a package checker)
  - Good documentation will make your software more appealing and useful

As you work toward the goal

- Unit tests (make lots of them, use lots of them)
- Trial and error (on your development branch, of course)
- The #opensource, #openaccess community is ready to help; find a good listserv or forum and ask away
  - …but pay it forward and assist in the future when you're the one who knows the answer
- If in R: Run R CMD check, R CMD BiocCheck early & often

Once it's out there

- Support your work & be willing to fix bugs (especially early on)
- What happens after you leave: Do you have a **software sustainability** plan?
  - Some repositories will sunset or mothball your package after years of inactivity, but it might still be useful to someone

Once it's out there

- Consider creating a companion data package containing a *validated* example/demonstration dataset that you understand
  - Helps users learn your software
- Promote: A published paper helps, but so does accession to a repository (CRAN or Bioconductor for R)
  - Social media?
- Get credit: Can use Zenodo to archive and obtain DOI for each release of your package; new solutions on horizon

Advice specific to R

- Functionalize everything as soon as you can (another reason to start thinking about the future early)
- The `apply` functions are far better (and faster) than most if… then… statements
- Maintaining a Git repo and a parallel package instance in Bioconductor's Subversion system is nontrivial; happy to share some pointers if/when you get there: james.r.collins@aya.yale.edu

Some general resources

- https://software-carpentry.org/ (if very new to the idea of open-source development or coding, sign yourself up for a workshop)
- ESIP Software Guidelines, draft October 2016: https://esipfed.github.io/Software-Assessment-Guidelines/guidelines.html (thanks, Stace)
- A few useful papers (credit C. Titus Brown, Stace Beaulieu):
  - http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002303
  - https://arxiv.org/pdf/1609.00037v2.pdf

Some R resources

- https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/
- https://cran.r-project.org/doc/manuals/r-release/R-exts.html
- Bioconductor best practices (even if not submitting to Bioconductor):
  - https://www.bioconductor.org/developers/how-to/buildingPackagesForBioc/
  - https://www.bioconductor.org/developers/how-to/coding-style/
- Anything by Hadley Wickham, but particularly:
  - http://r-pkgs.had.co.nz/man.html (guidance on documentation)