Assignment 1: Packaging Greengraph

11 January 2016

Ernest Lo
110006646
MPHYG001

Greengraph

Description:

This is a software that plots the number of green pixels between a start point, an end point, given the number of steps and the name of the output file you want the graph to be saved as.

Usage:

```
greengraph [-h] [--from FROMCITY] [--to TOCITY] [--steps STEPS] [--outOUT]

optional arguments:
    -h, --help   show this help message and exit
    --from FROMCITY, -f FROMCITY
    --to TOCITY, -t TOCITY
    --steps STEPS, -s STEPS
    --out OUT, -o OUT
```

Difficulties

Creating useful tests and utilising the mock module were the most challenging parts of this task. Understanding how to use mocks to restrict internet usage when testing individual methods and devising tests that are as comprehensive as possible. Despite attempting a few tests using mocks ultimately they were not a part of the package.

Publishing

There are many things to consider when publishing your software. First you need to consider the purpose of why you are releasing it, the license associated with the code and a business model if relevant, how you will maintain the code and whether you are prepared for it to be used to by others and whether it is meant for a certain community or niche market.

Software that is generally released to the wider public will be much cleaner in terms of layout and readability, as well as comments and documentation. There is the implication that released code will be generally more bug-free however that is a misconception as code authors generally use their own code in a very narrow fashion. Code that is written purely for the code author will in the common usage for the author can be comparatively bug free as he/she has written the code for that narrow purpose. When the wider community stretches the capabilities of a piece of software, they will reveal more bugs and limitations. This is however also an advantage of releasing code, as it allows far more human users as testers of your software.

Depending on the license attached to the released software. Releasing it to the wider public can allow business opportunities, where the author is compensated with donations, purchases of premium versions of software, add-ons, customer service etc.

Scientific software that is released may also be used and cited by the wider scientific community. This can become a part of the code author's scientific credentials as well as increasing his recognition.

The use of package managers like pip and indexes like PyPI allows a quick and easy way for others to install your software without spending too much time reading through your code to figure out how it is

installed and used. Package managers streamline the process, and allows a very efficient way to add modules and libraries to your current python package. An index allows the code author to be familiar with what is already out there, preventing repeated efforts to solve a solved problem. It also gives the code author exposure to a large community of users that may be interested in his/her software.

The cost of preparing software for package managers and indexes is the extra work the author has to put in to ensure everything is well documented and every module and library is appropriately linked up within the package. It forces the author to have a well organised structure, whether that's using appropriate folder hierarchy, separating classes, functions and scripts, as well as tests and documentation. The time cost of this can be very significant for a very small project and therefore if the code is intended for a small circle, such as a research group, it may not be worth the time for the author to create a package out of it. Similarly this applies to code that is frequently updated and modified, such as research code, keeping versions up to date is an added time cost for the author maintaining his software that is on a package index.

Building a community

To build a communiy of users, you first need to know whether there is a community that is interested in the application of your software. You then need to be aware of how popular your desired coding language is among that community.

For scientific software, it is useful to have a small group of known colleagues and collaborators that can give you more immediate constructive feedback and stress testing. Very often scientists release their software on their own websites and sometimes require that they are used for academic purposes only. While this may reduce exposure as compared to a package index and using more liberal licenses, it might create a smaller but more tight knit community of like minded individuals.

For example there are plenty of materials simulation software for semiconductors, and while it may be useful for a wide range of fields, from condensed matter physics to solid state to electrical engineering. An electrical engineer will write software that is more purpose built for his field. If he is spending time catering and maintaining his code for people that it wasn't made for, it may be counter productive to his code development.