MPHYG001 Assignment 1: Packaging Greengraph

Paul Brookes
University College London

January 9, 2016

Usage

The Greengraph package comes with a command line interface called greengrapher.py. This can be used to plot the fraction of green space a variable number of points between two locations. The interface takes four optional arguments: START, END, STEPS and OUT. It's usage is summarised by the output of `python greengrapher.py --help` shown below.

```
usage: greengrapher.py [-h] [--steps STEPS] [--out OUT] [--start START] [--end END]

Plot the amount of green space between two locations.

optional arguments:
-h, --help      show this help message and exit
--steps STEPS   Number of steps plotted. Default value = 20.
--out OUT       Name of output file. "*.png" or "*.pdf"
--start START   Start location for plot. Default location is London.
--end END       End location of plot. Default location is Cambridge.
```

Problems

The most difficult problem encountered in this project was understanding the Mock class and how to patch objects within libraries using mocks. The principal reason for learning these tools was to prevent the Greengraph and Map classes from accessing the internet during tests. Greengraph accesses the internet when the geolocate method is called since it uses the geopy library to find the co-ordinates of a supplied location. The Map class uses the requests library to obtain aerial photographs from Google Maps at given co-ordinates.

In order to prevent this it was necessary to patch over objects in the geopy and requests libraries with Mocks so that I could keep track of when these objects were used, what they were called with and supply my own data to greengraph so that I could check how successful it was at identifying and counting green pixels in images I personally supplied to it.

Another difficulty was to find a way of testing greengraph is a wide range of circumstances since bugs could slip past if only a narrow range of test cases is examined. It is for this reason that I wrote the `test_tools.single_colour_speckle` and `test_tools.multi_speckle` functions. The purpose of these functions is to randomly generate arrays of pixels which can be supplied to the Map class to test the `Map.count_green`, `Map.green` and `Map.show_green methods`. Since there is a degree of randomness resulting arrays have a degree of randomness is the generation of these arrays they can cover a wide range of test cases and be representative of the kind of images which will be obtained from Google Maps during normal use of the package.

Releasing your work

There are many pros and cons to preparing your work for release which must be taken into account. Let us first consider the advantages:

- By attempting to make your code available to others you force yourself to meet certain minimum quality standards. You will be motivated to make your code clear, readable and robust. You will

check the robustness of your code by implementing tests and the development of tests will likely encourage you to think about how a user would interface with your code. This will lead to your code being generally easier to use.

- If the community takes notice of your code then you will develop a user base who will all be testing your code for you and who may contact you if they find problems. This is one of the great advantages of open source code and many code sharing websites such as GitHub have issue trackers designed for this purpose.
- If someone else uses your code and compliments it you'll have warm tingly feelings inside.

We must also consider the cons:

- If your work finds uses among the community you are then put into a position of responsibility. There is pressure upon you to maintain the code. Compatibility issues must be resolved when your package's dependencies are updated. Users will expect you to fix bugs and respond to requests for information and clarifications about your work. You may not wish to deal with these problems but without someone to maintain it your code will rot.
- Preparing your code for release will take a significant amount of time. If it is highly unlikely that anyone else will find a use for your code you may not wish to invest this time.
- You will have to choose a license. This can be a very complicated decision, especially if you wish to make money from your code. You will have to consider under what circumstances you wish to allow others to profit from your code, whether or not they can patent their own contributions to your code, whether you wish to force users of your code to make their projects open source as well, etc.

Package managers and package indexes

One of the barriers to sharing code is making your code easy to install. Luckily there is a brilliant tool to simplify the installing of packages in Python called `pip`. This can be used to download and install any package stored in the PyPI repository using only a single line in your terminal. If the desired package requires other PyPI packages to function then these will be installed automatically. If you wish to make it easy to install your package the best path is to submit it to PyPI. Alternatively `pip` can also be used with GitHub.

Building a community

To build a community of users for your project you need to make your code easy to install, easy to use and robust. You personally have to interact with the community of users by being responsive to any issues raised. However the most important thing is for your code to do something interesting and to tell people about it. Don't build new code when there's already a library which solves the same problems.