

Geocoding and Bounding

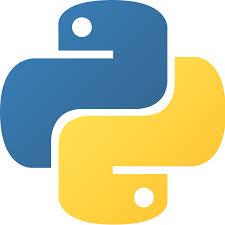
in Object Oriented Python (3.6.1)

(with special appearance by the Geocoder package)

by

Anthony E. Rutledge

5/10/2017 @ 5:00 P.M.

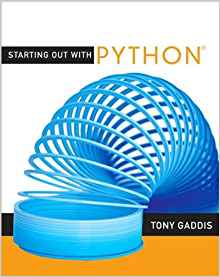
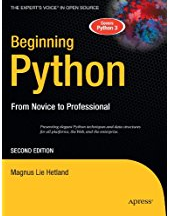
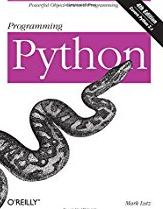


Contents

1. Introduction
2. Installation and configuration
3. Design and Architecture
4. Testing
5. Conclusions

I. Introduction

I hope this is what you were expecting. My girlfriend was like “just do the basics.” But, then I got rolling after I finished reviewing. Python, a splash of cold water to the face! I have not touched it since 2011 (2.7.x). I decided that I would attempt this exercise in Python 3.6.1, even though that meant almost starting from scratch in a situation where speed might mean something. I was quite busy last week, but by Friday I had finished reviewing and upgrading my knowledge to Python 3.6.1. Final basic testing happened yesterday and today.

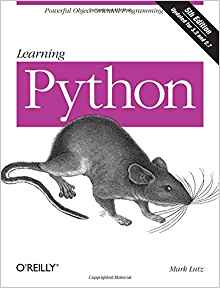


Figure 1: Python books I skimmed + official Python 3 library/documentation.

I figured the problem out easy enough, but I had no geocoding or 3rd party package experience. I like Visio. I made a basic scratch flow chart (GeoGrid Flow in the Visio doc), but the actual solution is a bit more detailed. See **GeoGrid::main.**

Also, I do not actively practice automated software testing. So, without more time to study, I devised a way to throw mass addresses at the program via the Captain Kirk method. I changed the conditions of the exercise. The program allows you to specify multiple files, not just one. Therefore, the output is delimited by the coordinates of the bounding box (you will see I have a different name for this) and dashes.

*In my opinion, the bulk of the program comes down to separation of concerns, input validation (command line, file, and geocoder), and exception handling*. There are many ins and outs of securing an application like this that I did not go into (such as preventing NUL byte injection), but I did try to make the program robust. Additionally, I tried to use some features I never had before in Python (Enums). Oh, and Python 3 is new to me, too!

(continued)

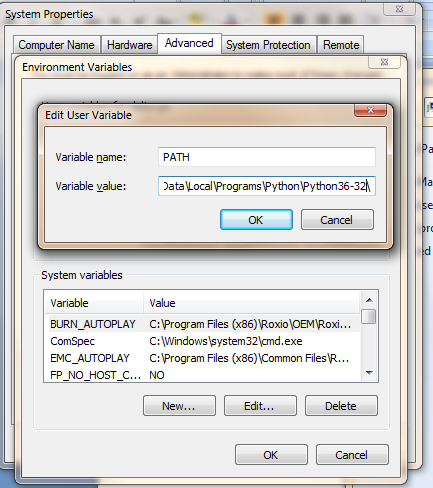
II. Installation and Configuration

I started from scratch.

1. Fresh install of Python 3.6.1: <https://www.python.org/downloads/>

2. Fresh install of Wingware Pro: <https://wingware.com/>

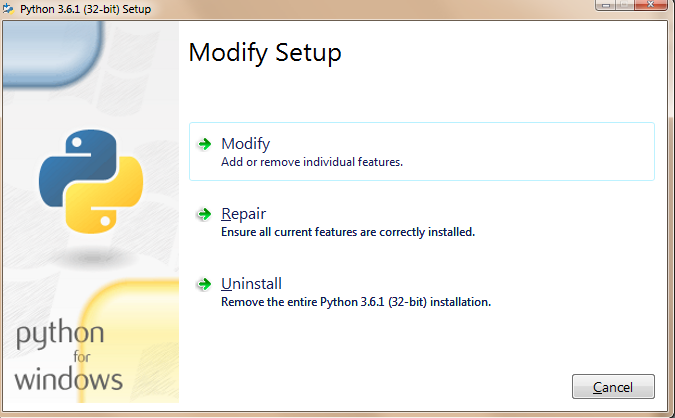
3. Configured various PATH stuff in Windows so that I could run Python command from the Windows command line (so much easier in FreeBSD/Linux).



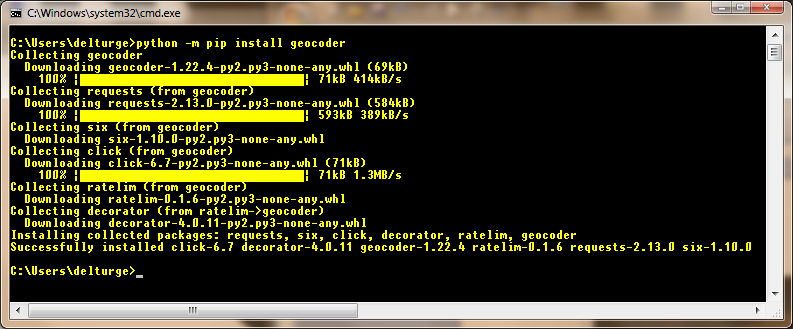
Change the PATHEXT environmental variable. Add **;.PY** to the end.

(continued)

4. Actually the path thing messed up during installation, so I had to run “modify” from the installer.



5. I installed Geocoder using the Python documentation and the official documentation about how to configure Python.



III. Design and Architecture

The *Grid* tab in the Visio UML diagram says it all.

It is an object-oriented MVC “wanna be.” I tried to emphasize separation of concerns (save for creating a Sanitizer module for inputs, which I implemented as various methods to save time), modularity, dependency injection, input validation (command line, files, setting grid coordinates, getting data back from geocoder, etc…). The *UML static class diagrams in the Visio document* probably say 1,000,000 words. So, have a gander at those.

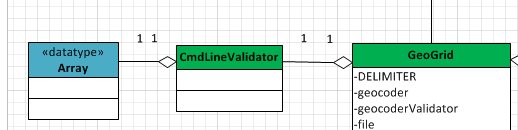
Aside from multiple inheritance, Python’s object model is functional, but seemingly primitive. I kept getting slapped in the face with the “Pythonic” way of doing things. Well, more power to the Python world, but I feel that my objects are not as “tight” as they could be if I had private (not mangled), protected, and public options for visibility (almost in the same vein as ECMAScript 3). That said.

**Private members**: self.\_\_foo

**Protected members**: self.\_foo

**Public members**: self.foo

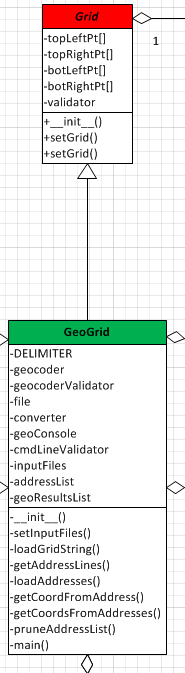
**~Request Hanlder~**

****

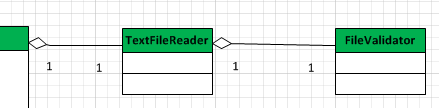
CmdLineValidator extends FileValidator, which extends Validator. Command line validator is a generic name, but it will do for this exercise. This class validates the input given at the command line. It passes the file name in put to the GeoGrid “controller”.

**~Controller~**

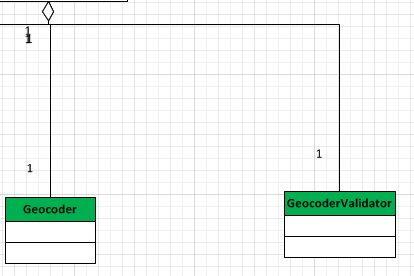
*GeoGrid* extends *Grid*. It manages everything. Maybe it’s not an essential separation, but is does leave the door open for there to be different kinds of Grids. Anyway, the idea is to program to an interface. Since abstract classes are apparently a pain in the neck in Python 3, no base classes are abstract (but, that’s not my preference). Yes, yes, there is no controller base class. However, I did not start out trying to emulate MVC.



**~Model~**

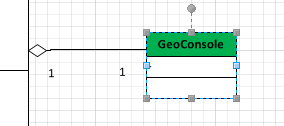


(Service layer)



What can I say? TextFileRead is a wrapper class for opening text files and reading to them. I do not do much with iterators and generators in PHP, but they are all over Python. So, with those efficiencies, I implement modern file handling when practical.

**~View~**

****

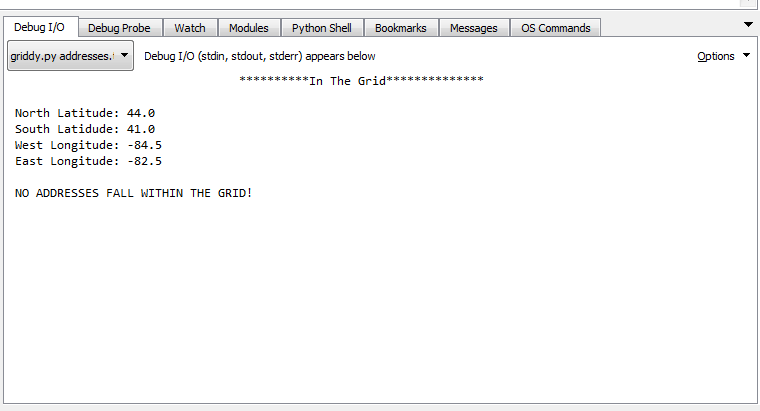
Now, *GeoConsole* actually extends *View*. I took the display code out of the controller and put it here so that the controller would not be coupled to the display implementation. That said, views would have to have the same interface to be interchangeable. Stands to reason, I should have done this with the model, too (in case the data comes from a database), but I wasn’t thinking like that in the beginning of the program.

And, if I had been thinking about it, all tasks in the controller necessary for managing input from Geocoder and the file would have been put into one model. See the “A model idea” tab in the Visio UML Diagram. I spent too much time catching up with Python 3 to think about overall architecture.

IV. Testing

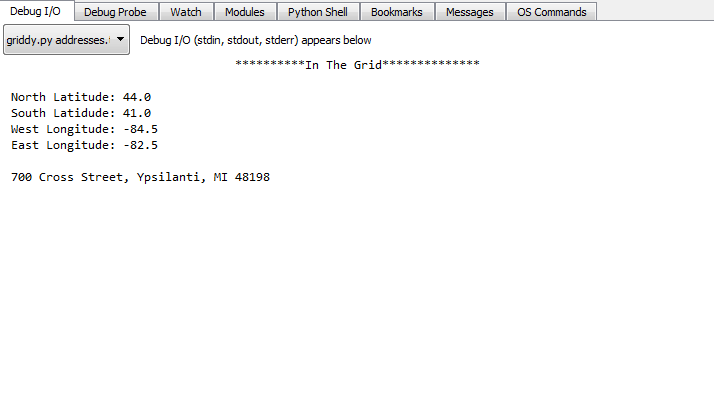
The grid coordinates target southeastern Michigan (44.0 N, 41.0 S, -84.5 W, -82.5 E)

**Target Address:** 500 Riverside Drive, Augusta, ME 04330



When there are no addresses in the grid.

**Target Address:** 700 Cross Street, Ypsilanti, MI 48198



When there is only one address in a file and it is within the grid.

**Target Addresses**:

700 Cross Street, Ypsilanti, MI 48198

1501 Grant Road, West Chester, PA 19380

1250 Holmes Road, Ypsilanti, MI 48198

500 Riverside Drive, Augusta, ME 04330

Detroit, MI

Ann Arbor, MI

Columbus, OH

New York, NY

Miami, FL

Seattle, WA

990834098309

Houston, TX

{}{|}{;;340- 09re09 -09r ,

Philadelphia, PA

Bangor, ME

Tigers

Southfield, MI

Chicago, IL

Boston, MA

\*&#\*($&)#\*\_%

Chapel Hill, NC

Juno, Alaska

Big Bird

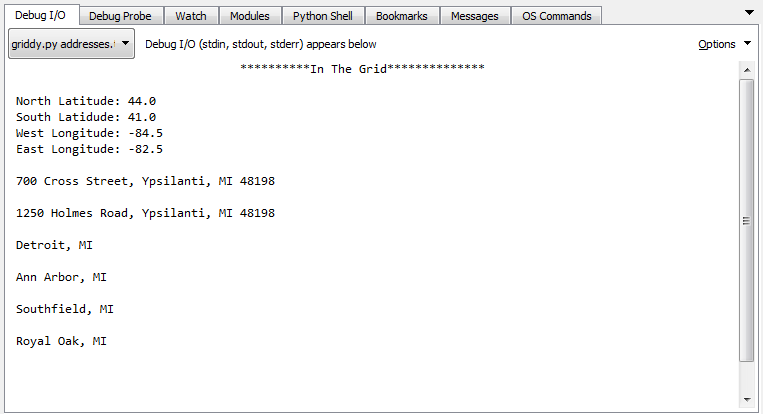
Cookie Monster

Austin, TX

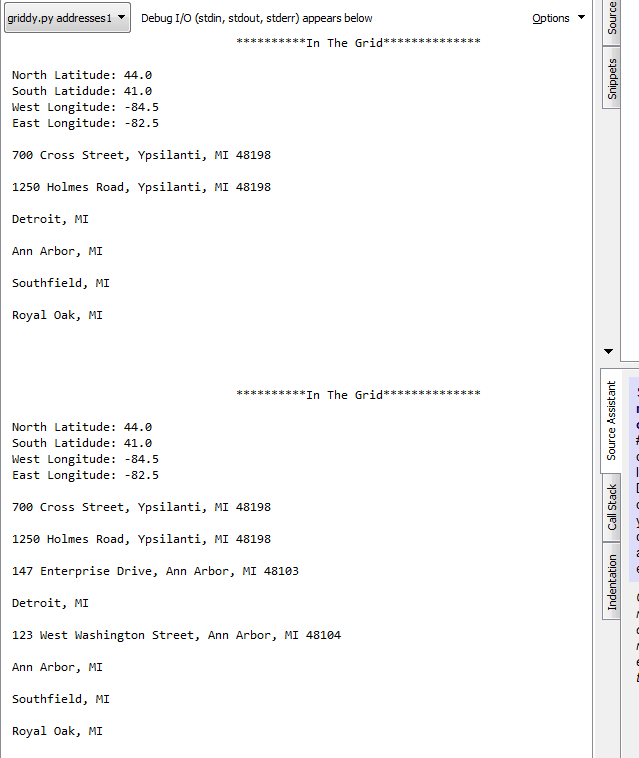
Bombay, India

Los Angeles, CA

San Francisco, CA



**Using more than one (2) file.**



V. Conclusions

With more time the solution could be refined more. But, with the time I have, this is what I can do. Software engineering takes time. You have to study, and I am always learning more as I go. This program could have been web based, but that was not my task. Hopefully, you enjoyed the diagrams. I do not have automated software testing experience, but I can use a loop! ;-)

There are flaws in the program, but I have no time to fix them all right now. I posted this question on stackoverflow.com. If I ever get the answer, the program will be complete.

http://stackoverflow.com/questions/43904389/why-is-a-python-3-6-1-program-working-in-wingware-pro-6-but-failing-at-the-wind?noredirect=1#comment74842705\_43904389

That is all for now.

Anthony

(nothing follows.)