

# **Real World Python**

for map makers

# Objective

Create a data set of Target store locations in the U.S.



# Steps

1. Find Target store location info on the web
2. Develop a strategy
3. Get our python environment ready
4. Write python code to parse and store post office location data

# Finding Target store location info

Target has a [Store Locator](#). Promising!

But...

No map.

You cannot download a file.

*And what's worse:*

There are no coordinates attached to the locations, only addresses.



# all locations

Target

Find a Store

All Locations

## All Target Locations

### Select a State\*

We have a store near you.

Alabama  
Alaska  
Arizona  
Arkansas  
California  
Colorado  
Connecticut  
Delaware  
Florida  
Georgia  
Hawaii  
Idaho  
Illinois  
Indiana  
Iowa  
Kansas  
Kentucky

Louisiana  
Maine  
Maryland  
Massachusetts  
Michigan  
Minnesota  
Mississippi  
Missouri  
Montana  
Nebraska  
Nevada  
New Hampshire  
New Jersey  
New Mexico  
New York  
North Carolina  
North Dakota

Ohio  
Oklahoma  
Oregon  
Pennsylvania  
Rhode Island  
South Carolina  
South Dakota  
Tennessee  
Texas  
Utah  
Virginia  
Washington  
Washington DC  
West Virginia  
Wisconsin  
Wyoming

\*Currently, there are no Target stores in Vermont.

So we need to find ways to:

- 1) scrape the 'locations' from the web site
- 2) geocode these 'locations'.

**Python is great for this!**

# Scraping

Sucking information from web sites that are not set up to give it to you the way you want it.

An astounding amount of (spatial) information is hidden away on the web that way.

Python has awesome tools to help you get to it.

# Our python toolbelt

'[Requests](#) is an elegant and simple HTTP library for Python, built for human beings.'





'You didn't write that awful page. You're just trying to get some data out of it. [Beautiful Soup](#) is here to help'



# Geocoding

## Two strategies

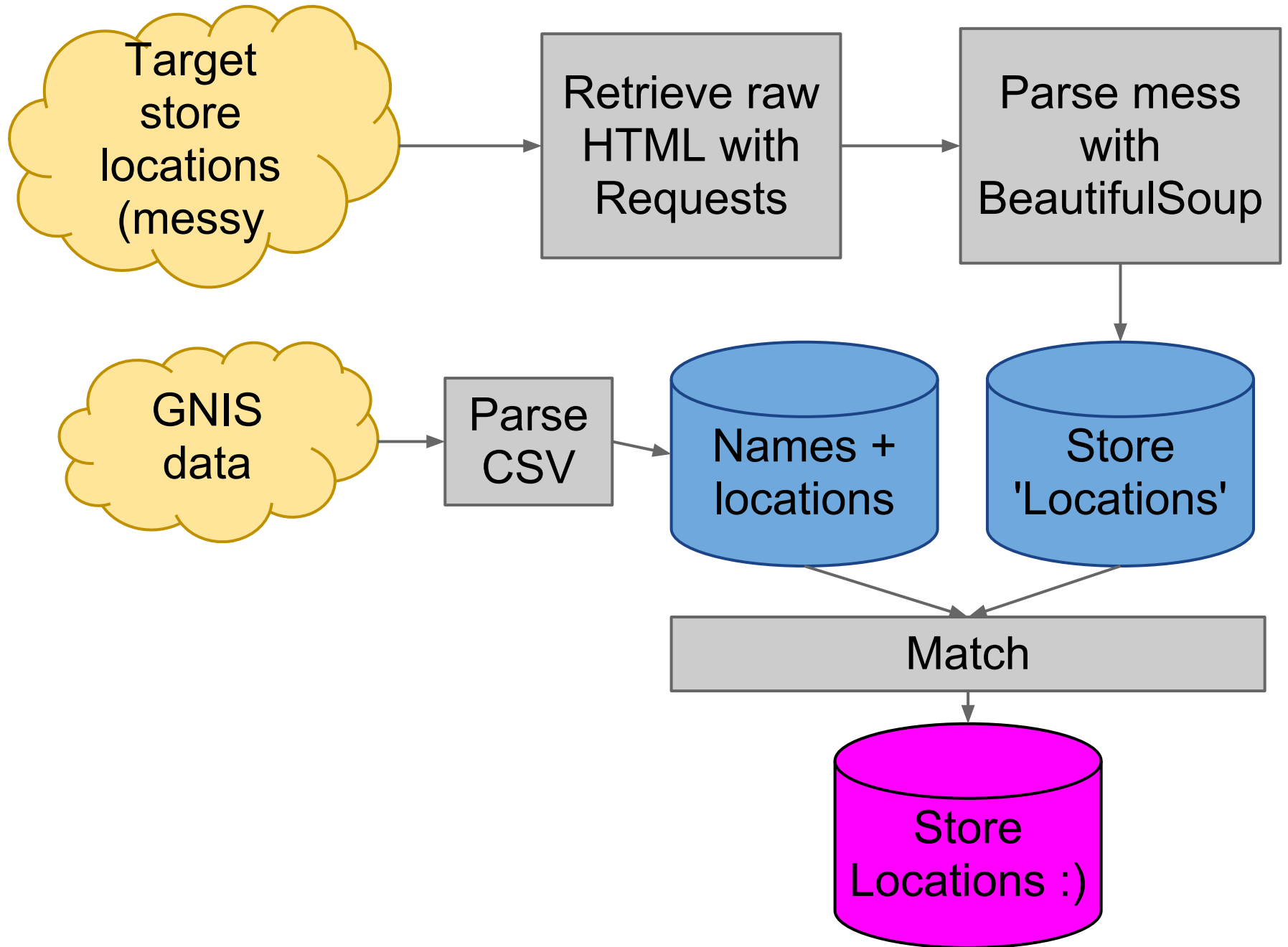
- 1) Use an online geocoding service (through [geopy](#) for example)
- 2) Match with USGS [GNIS](#) data or [ZIP codes](#)

Option 2 is better because the data is free (as in beer and as in speech).

1	FEATURE_ID	FEATURE_NAME	FEATURE_CLASS	STATE_ALPHA	STATE_NUMERIC	COUNTY_NAME	COUNTY_NUMERIC	PRIMARY_LAT_DMS	PRIM_LONG_DMS	PRIM_LAT_DEC	PRIM_LONG_DEC
2	1397658	Ester	Populated Place	AK	02	Fairbanks North Star	090	645950N	1480052W	64.8472222	-148.0144444
3	1397685	Two Rivers	Populated Place	AK	02	Fairbanks North Star	090	645220N	1470218W	64.8722222	-147.0383333
4	1397926	Afognak	Populated Place	AK	02	Kodiak Island	150	580120N	1524555W	58.0221236	-152.7652264
5	1397967	Aguikchuk	Populated Place	AK	02	Bethel (CA)	050	602830N	1642608W	60.475	-164.4355556
6	1397989	Aiaktalik	Populated Place	AK	02	Kodiak Island	150	564225N	1540654W	56.7069444	-154.115
7	1398007	Akhiok	Populated Place	AK	02	Kodiak Island	150	565644N	1541013W	56.9455556	-154.1702778
8	1398011	Akiachak	Populated Place	AK	02	Bethel (CA)	050	605434N	1612553W	60.9094444	-161.4313889
9	1398012	Akiak	Populated Place	AK	02	Bethel (CA)	050	605444N	1611250W	60.9122222	-161.2138889
10	1398027	Akulurak	Populated Place	AK	02	Bethel (CA)	050	601639N	1622629W	60.2775	-162.4413889
11	1398028	Akulurak	Populated Place	AK	02	Wade Hampton (CA)	270	623308N	1643310W	62.5522222	-164.5527778
12	1398029	Akumsuk	Populated Place	AK	02	Wade Hampton (CA)	270	624323N	1642820W	62.7230556	-164.4722222
13	1398042	Alakanuk	Populated Place	AK	02	Wade Hampton (CA)	270	624120N	1643655W	62.6888889	-164.6152778
14	1398055	Alatna	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	663350N	1524000W	66.5638889	-152.6666667
15	1398091	Aleknagik	Populated Place	AK	02	Dillingham (CA)	070	591623N	1583704W	59.2730556	-158.6177778
16	1398093	Aleksashkina	Populated Place	AK	02	Kodiak Island	150	574648N	1522119W	57.78	-152.3552778
17	1398095	Aleut Village	Populated Place	AK	02	Kodiak Island	150	580104N	1524556W	58.0178	-152.7655
18	1398097	Alexander	Populated Place	AK	02	Matanuska-Susitna	170	612502N	1503549W	61.4172222	-150.5969444
19	1398102	Alexanders Village	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	664718N	1452613W	66.7883333	-145.4369444
20	1398129	Allakaket	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	663356N	1523844W	66.5655556	-152.6455556
21	1398235	Anaktuvuk Pass	Populated Place	AK	02	North Slope	185	680836N	1514409W	68.1433333	-151.7358333
22	1398242	Anchorage	Populated Place	AK	02	Anchorage	020	611305N	1495401W	61.2180556	-149.9002778
23	1398245	Anderson	Populated Place	AK	02	Denali	068	642039N	1491113W	64.3441667	-149.1869444
24	1398261	Saint Marys	Populated Place	AK	02	Wade Hampton (CA)	270	620311N	1630957W	62.0530556	-163.1658333
25	1398286	Aniak	Populated Place	AK	02	Bethel (CA)	050	613442N	1593120W	61.5783333	-159.5222222
26	1398315	Anogok	Populated Place	AK	02	Bethel (CA)	050	594916N	1640143W	59.8211111	-164.0286111
27	1398335	Anvik	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	623922N	1601224W	62.6561111	-160.2066667
28	1398352	Apokak	Populated Place	AK	02	Bethel (CA)	050	600748N	1621153W	60.13	-162.1980556
29	1398382	Arctic Village	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	680737N	1453216W	68.1269444	-145.5377778
30	1398409	Artesian Village	Populated Place	AK	02	Anchorage	020	611339N	1494653W	61.2275	-149.7813889
31	1398469	Auke Bay	Populated Place	AK	02	Juneau	110	582300N	1343935W	58.3833333	-134.6597222
32	1398480	Aurora	Populated Place	AK	02	Fairbanks North Star	090	645125N	1474531W	64.8569444	-147.7586111
33	1398485	Aurora Lodge	Populated Place	AK	02	Fairbanks North Star	090	642814N	1465619W	64.4705556	-146.9386111
34	1398551	Baker	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	645801N	1502735W	64.9669444	-150.4597222
35	1398627	Barrabara (historical)	Populated Place	AK	02	Nome (CA)	180	632304N	1621953W	63.3844444	-162.3313889
36	1398635	Barrow	Populated Place	AK	02	North Slope	185	711726N	1564719W	71.2905556	-156.7886111
37	1398672	Batzulnetas (historical)	Populated Place	AK	02	Valdez-Cordova (CA)	261	623653N	1434617W	62.6147222	-143.7713889
38	1398764	Bearpaw	Populated Place	AK	02	Denali	068	640458N	1504100W	64.0827778	-150.6833333
39	1398776	Beaver	Populated Place	AK	02	Yukon-Koyukuk (CA)	290	662134N	1472347W	66.3594444	-147.3963889

# Strategy

- Load GNIS data (memory? database?)
  - Look at number of records, total file size
  - Memory is much less hassle but can get slow for querying (not indexed)
- Retrieve the Target locations
  - One page for each state
  - What is the page HTML structure? Find the pattern.
- Match found places with GNIS
  - Match on hopefully unique combination of place + state code.
  - Where are the coordinates?
- Output
  - CSV for easy loading in ArcMap
  - Other options.. GeoJSON, KML, Shapefile, database.



# Python Environment

First, we will create and activate a python **virtual environment**.

This is a fully self-contained python environment that we can use as a sandbox without messing up our systemwide python environment (which may be used for other, more important tasks)

```
C:\Users\mvexel>virtualenv class
New python executable in class\Scripts\python.exe
Installing setuptools.....done.
Installing pip.....done.

C:\Users\mvexel>class\Scripts\activate.bat
(class) C:\Users\mvexel>
```

# Python environment

Next, we will install the modules we will use. This is made easy by the Python Package Index ([PyPI](#)) and the [pip](#) tool.

You'll find the package names in their installation instructions.

```
(class) C:\Users\mvexel>pip install beautifulsoup4
Downloading/unpacking beautifulsoup4
  Downloading beautifulsoup4-4.1.3.tar.gz (131Kb): 131Kb downloaded
  Running setup.py egg_info for package beautifulsoup4

Installing collected packages: beautifulsoup4
  Running setup.py install for beautifulsoup4

Successfully installed beautifulsoup4
Cleaning up...
```

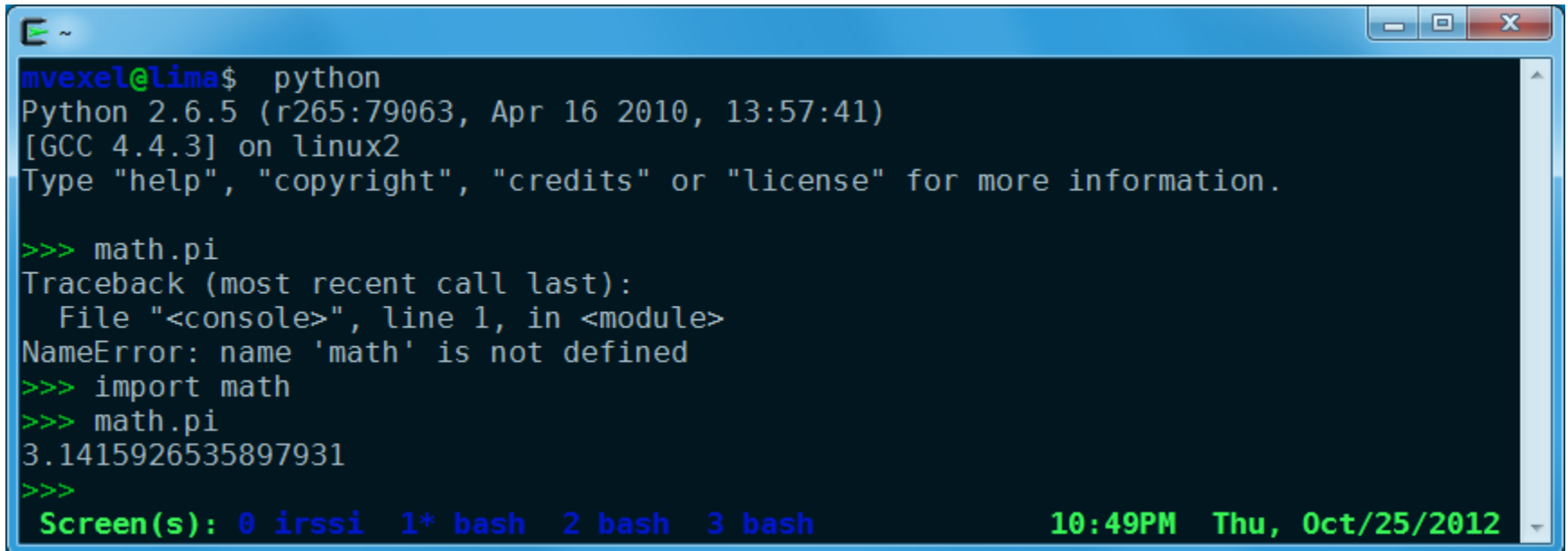
# Write Python code

1. import modules
2. load GNIS populated places
3. for each state:
  - a. Get web page
  - b. Find the relevant elements on the page (place names)
  - c. for each placename found:
    - i. Match to GNIS place names
    - ii. add (lon,lat) from matched GNIS place to record
    - iii. write to file



# Importing

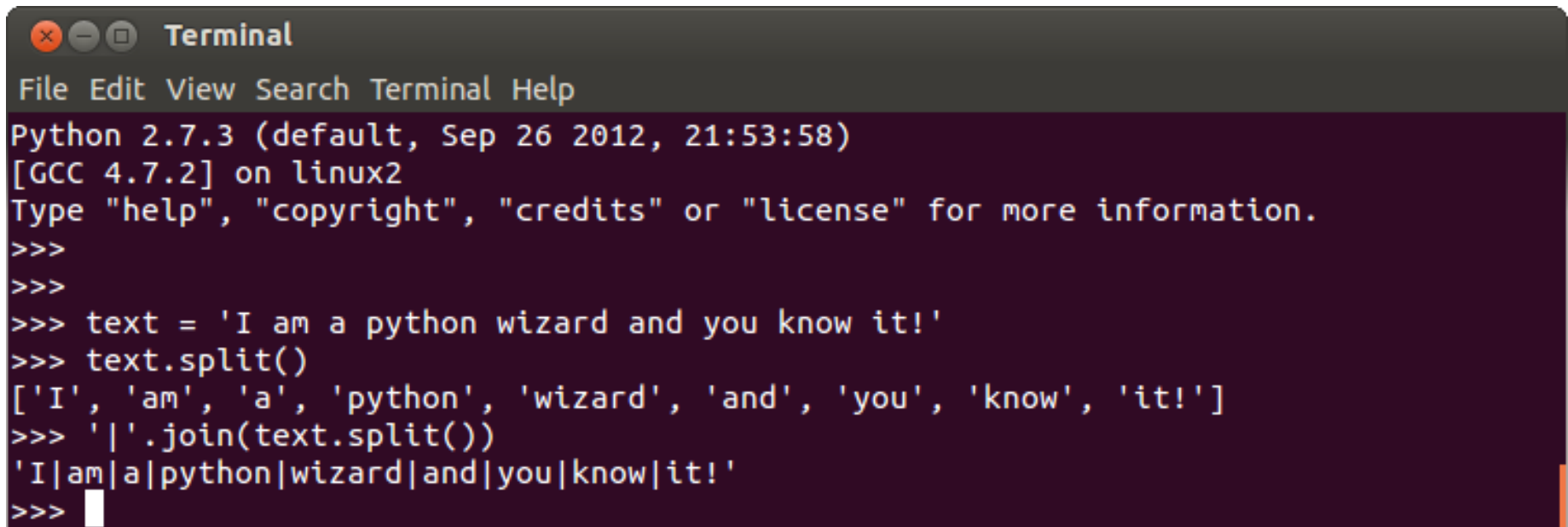
If you want to use any python module, you need to import it first.

A terminal window with a blue title bar and standard window controls. The text inside shows a user running 'python' in a shell. It displays the Python version (2.6.5), GCC version (4.4.3), and OS (linux2). The user then enters 'math.pi', which results in a 'NameError: name 'math' is not defined'. After entering 'import math', the user enters 'math.pi' again, which successfully returns the value '3.1415926535897931'. The bottom status bar shows 'Screen(s): 0 irssi 1\* bash 2 bash 3 bash' and the time/date '10:49PM Thu, Oct/25/2012'.

```
mvexel@lima$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> math.pi
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'math' is not defined
>>> import math
>>> math.pi
3.1415926535897931
>>>
Screen(s): 0 irssi 1* bash 2 bash 3 bash 10:49PM Thu, Oct/25/2012
```

# Doing stuff with strings

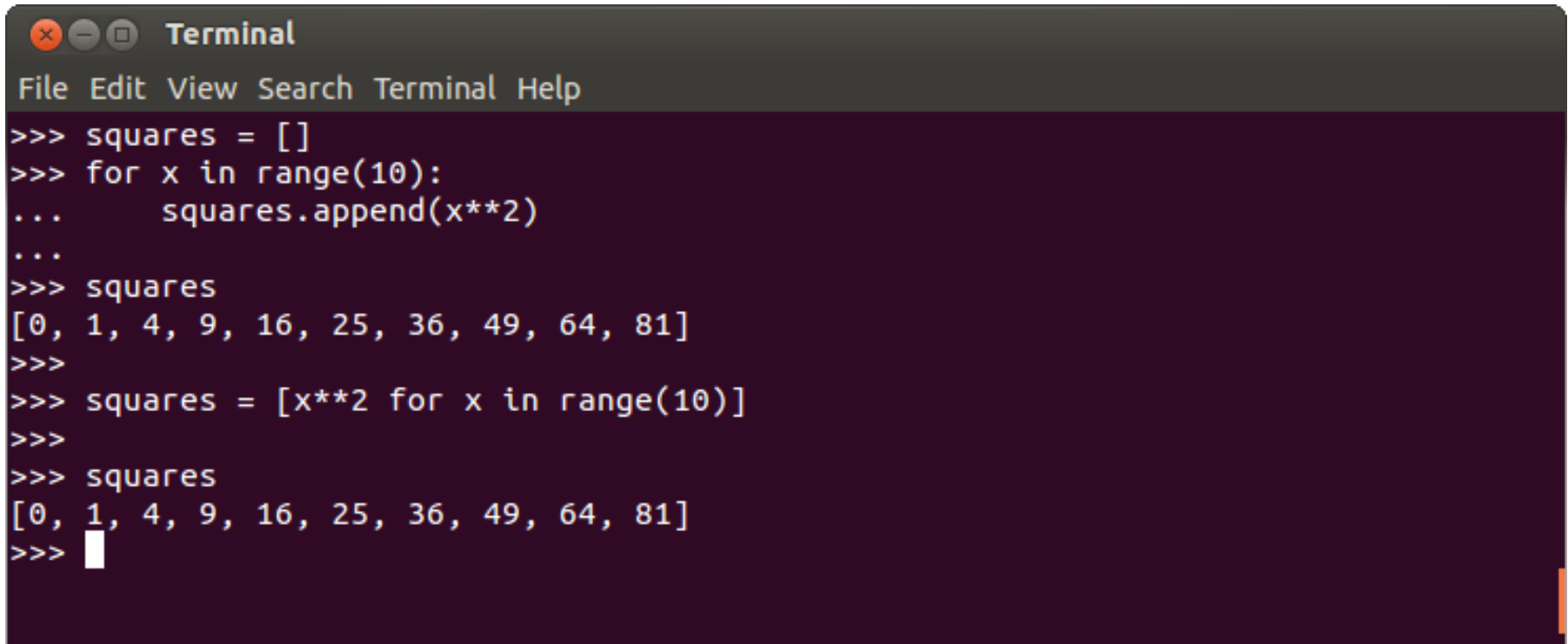
A terminal window titled "Terminal" with a dark background and light-colored text. The window shows the Python 2.7.3 prompt and several lines of code demonstrating string splitting and joining. The code defines a string 'text' and uses the split() and join() methods to manipulate it. The output of the join operation is displayed on the next line.

```
Terminal
File Edit View Search Terminal Help
Python 2.7.3 (default, Sep 26 2012, 21:53:58)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> text = 'I am a python wizard and you know it!'
>>> text.split()
['I', 'am', 'a', 'python', 'wizard', 'and', 'you', 'know', 'it!']
>>> '|'.join(text.split())
'I|am|a|python|wizard|and|you|know|it!'
>>>
```

# Data Structures: Lists

```
Terminal
File Edit View Search Terminal Help
0
>>> list = [0, 1, 'two', 'three', [4, 5, 6]]
>>> list[2]
'two'
>>> list[4][2]
6
>>> len(list)
5
>>> list.append([7, 8, 9])
>>> list
[0, 1, 'two', 'three', [4, 5, 6], [7, 8, 9]]
>>> list.extend(list)
>>> list
[0, 1, 'two', 'three', [4, 5, 6], [7, 8, 9], 0, 1, 'two', 'three', [4, 5, 6], [7, 8, 9]]
>>> list.count(1)
2
```

# Data Structures: Lists

A terminal window with a dark background and light-colored text. The window title is "Terminal". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The code is entered line by line, and the output is shown immediately. The code creates a list of squares from 0 to 9 using two different methods: a loop with append and a list comprehension. Both methods result in the same list: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81].

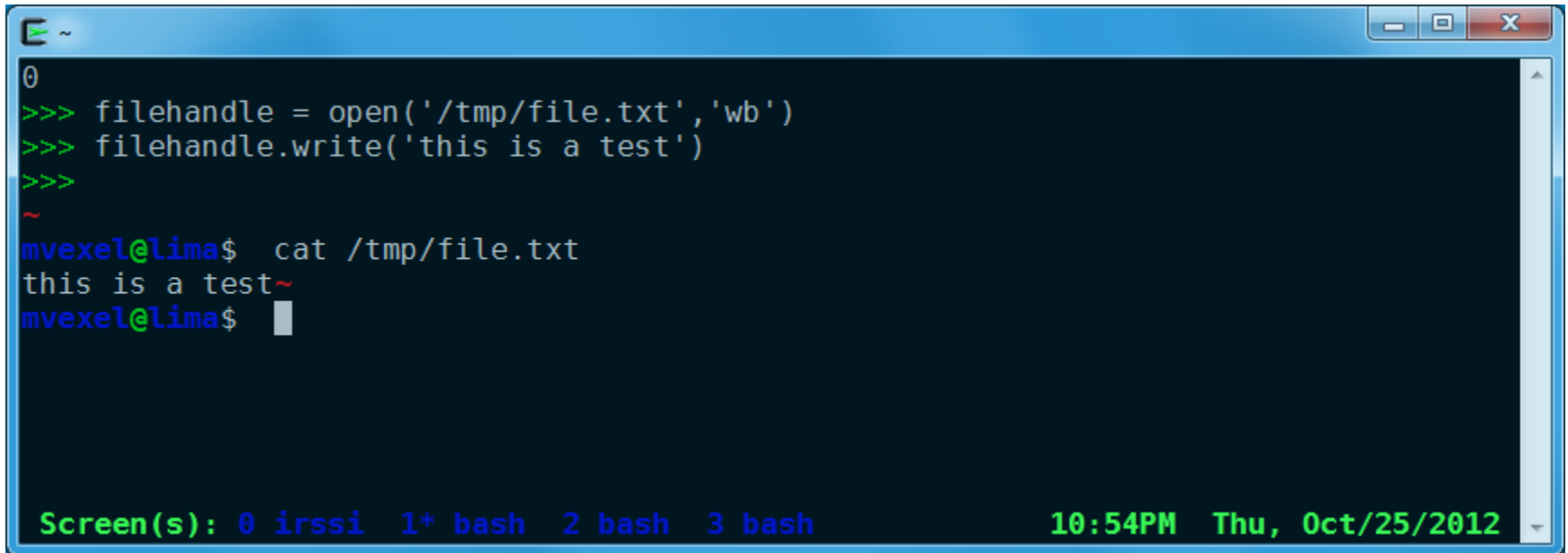
```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
>>> squares = [x**2 for x in range(10)]
>>>
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

# Data Structures: Dictionaries

```
Terminal
File Edit View Search Terminal Help
0
>>> dict = {'utes': 102, 'cal': 13, 'byu': 400}
>>> dict.keys()
['byu', 'utes', 'cal']
>>> for k,v in dict.items():
...     print '%s --- %s' % (k,v)
...
byu --- 400
utes --- 102
cal --- 13
>>>
```

# Working with files

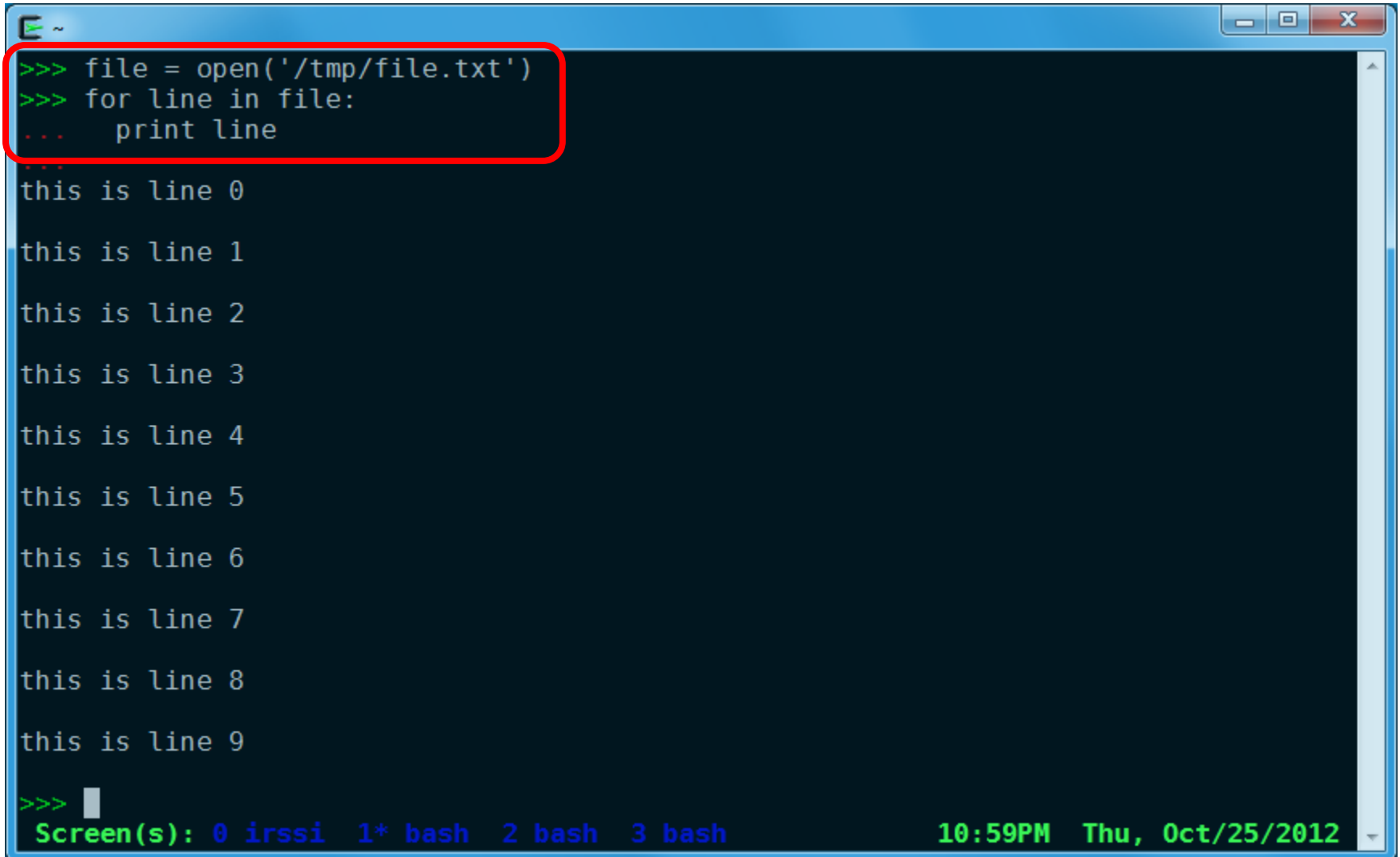
you can write to and read from a file handle, which is what `open()` returns:

A terminal window with a blue title bar and standard window controls. The terminal shows a Python script being executed in a shell. The script opens a file in write mode, writes a test string, and then the shell uses 'cat' to display the file's contents. The status bar at the bottom shows screen information and the current date and time.

```
0
>>> filehandle = open('/tmp/file.txt','wb')
>>> filehandle.write('this is a test')
>>>
~
mvexel@lima$ cat /tmp/file.txt
this is a test~
mvexel@lima$
```

Screen(s): 0 irssi 1\* bash 2 bash 3 bash 10:54PM Thu, Oct/25/2012

# Reading from a file line by line

A terminal window with a blue title bar and standard window controls. The background is dark blue. The text is in a monospaced font. The first three lines of code are enclosed in a red rounded rectangle. The output lines are indented. The status bar at the bottom is green and black.

```
>>> file = open('/tmp/file.txt')
>>> for line in file:
...     print line
...
this is line 0
this is line 1
this is line 2
this is line 3
this is line 4
this is line 5
this is line 6
this is line 7
this is line 8
this is line 9
>>> 
```

Screen(s): 0 irssi 1\* bash 2 bash 3 bash 10:59PM Thu, Oct/25/2012

# What can you do with this?

The resulting file can be loaded directly into ArcMap ([Display XY data...](#))

Instead of CSV, we could also have written the output as [GeoJSON](#), [KML](#) or [Shapefile](#) for further GIS processing.

And even make an interactive web map using [Leaflet](#)



# Try it yourself

[Rite-Aid Locations](#)

[Chase locations](#)

[CVS locations](#)

[Tea Party groups locations](#)

[Five Guys locations](#)

.....more? just [google](#)

# Thanks folks

Martijn van Exel

m@rtijn.org

@mvexel

Everything is on Github:

<https://github.com/mvexel/pythonclass>