Geography 378: Introduction to Geocomputing
Lab 1: Introduction to Python

Assigned: 9/13
Due: 9/27
24 points

Hand-in

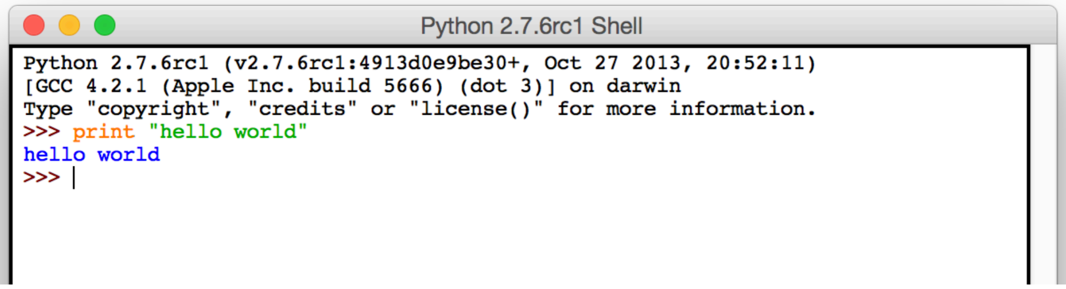
- Please collect your answers for each task in a separate .py file. Compress your Python files into a single .zip file and name your zip file as **lab1_yourname.zip.**
- Submit the zip file to the assignment folder called "Lab 1".
- Include appropriate comments to explain what each line or block of code accomplishes. **You must comment your code for full credit**.

Getting Started with IDLE

The simplest interface for writing and executing Python scripts is the IDLE environment. If you follow the instruction of ArcGIS installation on Learn@UW, you should already have Python/IDLE installed. We'll be using Python 2.7 for this lab (although 2.5 – 3.0 should work equally as well). Documentation for the Python standard library can be found at http://docs.python.org/2/library/index.html.

Now, follow these steps to create a simple Python script:

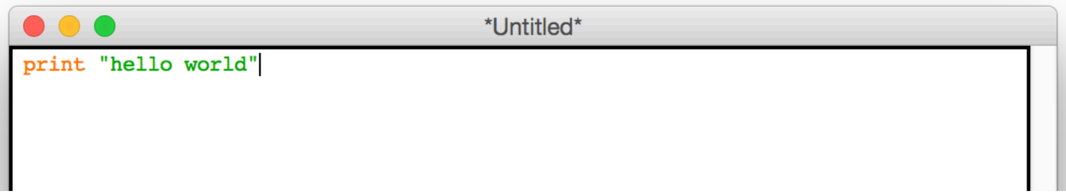
- Start IDLE by searching for IDLE in your program menu. Opening IDLE will bring you to a command line interface on which you can type single commands and hit "Enter" to execute. Try it out with the `print` command: `print "hello world"`



```
Python 2.7.6rc1 (v2.7.6rc1:4913d0e9be30+, Oct 27 2013, 20:52:11)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print "hello world"
hello world
>>> |
```

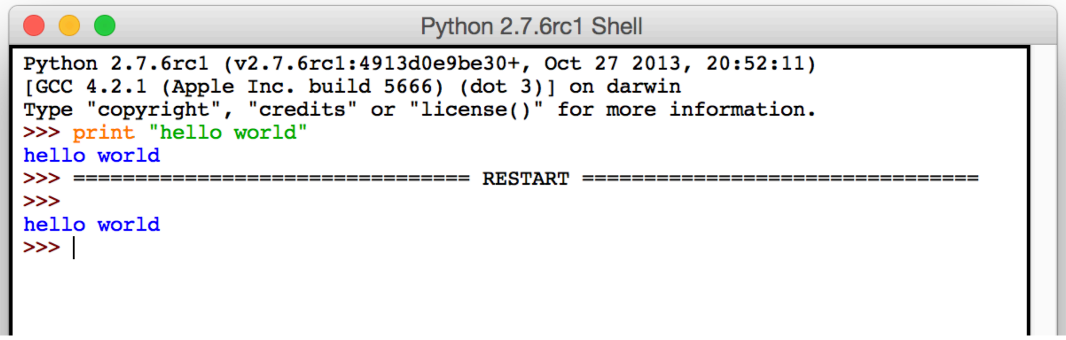
- To actually create your Python scripts, you will need to use the script editor. In the IDLE Python Shell, go to "File->New Window" (or hit CTRL-N). This will bring up an editing window for a new file to hold your Python script. The statements (or commands) in the file won't be executed until you run the program.

- Type the earlier print command in the new editing window.



```
print "hello world"
```

- Before running the script, you will first need to save it. Go to “File->save” (or hit CTRL-S) to save it. Note the file extension is .py.
- Now hit F5 to run the script (It will prompt you for saving if you haven’t save it). You will see the output in the Python Shell window (not the Editor).



```
Python 2.7.6rc1 (v2.7.6rc1:4913d0e9be30+, Oct 27 2013, 20:52:11)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print "hello world"
hello world
>>> ================================ RESTART ================================
>>>
hello world
>>> 
```

- Just as we saw with batch files, you can also create or edit (but not execute) your Python scripts in Notepad or Notepad++, but these files will not be integrated with the Python shell. You can execute them from the CLI (start it using cmd) by navigating to the directory in which you saved the files and typing “python *filename*.py”

Python Basics

You can download four sample Python scripts from Learn@UW. These scripts provide examples as a resource for you to use. It is recommended that you read through and experiment with them in the following order:

1. python_basics.py
2. function_if_example.py
3. while_statement_example.py
4. try_except_example.py

You should also make liberal use of the Python documentation, linked above, while completing the lab.

Lab Assignments

Now apply what you learned and complete the following tasks:

1. Let's start with some basic geometry. The Python script *triangle_area.py* calculates the area of a triangle. Comment the code, replacing each instance of `[explain]` with your own explanation as to what the block of code does, and save the modified file. (2 pts)

2. Modify the code from Task 1 so that it first asks the user to either calculate the area of a triangle or a trapezoid, prompts the user to enter the necessary values to perform the right calculations, then performs the calculations and outputs the result. Make sure your code is commented appropriately, and save it as ***triortrap_area.py***. (4 pts)

3. In the previous program, imagine that you are a user in a hurry that types 2O (oh) rather than 20 (zero) as the height. What happens to your program? In "try_except_example.py", you can see an example that shows how to avoid such problem.

   Now let's try to combine the example with the code from Task 2. If the user enters a value that could not be converted to numeric type, allow 3 additional opportunities to enter a new value. If the user fails to enter a correct value after 4 attempts, inform them of such failure and allow the program to end without crashing. Save your new code as **triortrap_advanced.py**. (4 pts)

4. Write a Python script that asks the user to input the latitude and longitude of a location. After the user has entered the information, the script should display one of the following messages to describe the latitude entered:

Your program should display:	**If the latitude entered:**
`That location is on the equator.`	Is 0
`That location is north of the equator.`	Is between 0 and 90
`That location is south of the equator.`	Is between -90 and 0
`That location does not have a valid latitude!`	Is greater than 90 or less than -90

and one of the following messages to describe the longitude entered:

Your program should display:	**If the longitude entered:**
`That location is on the prime meridian.`	Is 0
`That location is east of the prime meridian.`	Is between 0 and 180

`That location is west of the prime meridian.`	Is between -180 and 0
`That location does not have a valid longitude!`	Is greater than 180 or less than -180

If the user enters something other than a number, give them an error message and restart the program. Save your script as **latlon.py**. (6 pts)

5. Time for a GIS task. Write a Python script that calculates the distance between any two points on the Earth's surface, given their latitude and longitude.

   The script should first prompt the user to enter the coordinates of each location, then calculate the spherical distance along the Earth's surface (use the following distance formula).

   To make the problem simpler, you *may* assume that numbers entered are valid latitude or longitude coordinates in decimal degrees, but feel free to experiment with responding to violations of this assumption.

   Finally, save your script as ***great_circle_dist.py***. (8 pts)

   Spherical distance formula:

$$d = \mathrm{acos}(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\lambda_1 - \lambda_2))$$

   whereby **d** is the angle between two points, λ_1 and λ_2 are the longitudes of the points, and ϕ_1 and ϕ_2 are the latitudes of the points. . To convert the angle to a spherical distance, multiply by the radius of the earth, which is ~6300 km.

   Hints:

   - To use trigonometric functions in Python, you will need to import the `math` module at the beginning of the script.
   - Python's math module assumes that angular values are given in *radians*. Latitude and longitude values are always given in *degrees*, so you will need to convert the values entered by the user from degrees to radians.
   - Standard convention is to express northern latitudes and eastern longitudes as positive numbers, and southern latitudes and western longitudes with negative numbers.

   Sample outputs:

Here are some sample locations to test your code with:

Madison: 43.13972 ºN, 89.3375 ºW

New York: 40.77725 ºN, 73.87261 ºW

Jakarta: 6.12555 ºS, 106.6558 ºE

The output distances should be:

Madison - New York: 1289 km

Madison - Jakarta: 15437 km

New York - Jakarta: 15982 km

For a listing of latitude and longitudes of US and world cities, see http://www.realestate3d.com/gps/latlong.htm. You can also find coordinates for any location using http://www.robogeo.com/latlonfinder/map.asp.