INTRODUCTION TO PROGRAMMING
WEEK 12 - PART I & II CONCLUSION

CONGRATULATIONS!

You made it! Graduating from this course is no easy feat and I'm extremely proud of all of you.

DEMO FILES

I have prepared demo files for you to follow along with.

Please run the following from your `conclusion` directory:

```
cp -r /home/alberthwang/python/conclusion demo_files
```

OS MODULE

The `os` module was designed to take care of various operations at the Operating System level. The `os` module features various methods that can change permissions of directories, read files, etc. **Essentially the `os` module was designed so that a python script could do the things you do at the command line like `cd`, `mkdir`, `chmod`, etc.**

Let's take a look at 3 methods that I have found especially useful:

os.system

The `os.system()` call allows you to input a **unix command** and the python script will run it in your place.

```
import os

def main():
 os.system('touch test.txt')

if __name__ == '__main__':
 main()
```

Run the file and see what happens - is there a test.txt file now inside your week8 folder?

The `os.system()` method allows you to run a unix command (some limitations apply). The argument you pass inside `system()` is the command the script will run.

This method is especially useful when you have to batch create, delete, move, or rename dozens or hundreds of files or directories. For instance, if you work with data, for instance, you

could write a script that does calculations and then creates 5 reports or selectively deletes files out of a directory based on their content, etc.

os.chdir

One limitation of the `os.system()` method is that it needs to execute inside your current directory. You cannot run `os.system('cd nextdir')` for instance. For this, we have the `os.chdir` command that executes like a unix `cd`.

While this might not seem especially useful at first, imagine if you had to do an assignment where you had to create a dozen user folders with documents in them. This would be a real pain, but now you can harness the power of python. Change the contents of your main function to this:

```
def main():
 users = [{'ldap': 'bridgetc', 'score': '80'},
          {'ldap': 'lassetjt', 'score': '81'},
          {'ldap': 'alberthwang', 'score': '60'}]

 # Create a users directory
 os.system('mkdir users')

 # Go into the users directory
 os.chdir('users')

 for user in users:
   # For each user, create a new directory and go in
   os.system('mkdir ' + user['ldap'])
   os.chdir(user['ldap'])

   # Create a score.txt file
   os.system('touch score.txt')

   # Put the user's score into the score.txt file
   score_file = open('score.txt', 'w')
   score_file.write(user['score'])
   score_file.close()

   # Change directories back to the users directory
   os.chdir('..')
```

You should now have a **users** directory with 3 user sub-directories - bridgetc, lassetjt, and alberthwang. Inside each of these you should see a score.txt file with the user's score in it.

os.listdir

The `os.listdir` method returns a list of files (represented as strings) in a target directory. In the next example, I will show how to use `os.listdir` to batch rename all the files inside a directory.

Create a directory named **reports** and in it, create 5 text files named: hr_report_1.txt, hr_report_2.txt, hr_report_3.txt, hr_report_4.txt, hr_report_5.txt.

Now change the code in your main function to this:

```
def main():

 # Get a list of the reports in the directory and go in
 report_list = os.listdir('reports')
 os.chdir('reports')

 for report in report_list:
   # If it's an hr report, rename it
   if report[:2] == 'hr':
     new_name = report.replace('hr', 'peopleops')

     # Rename each report with its new name
     os.system('mv ' + report + ' ' + new_name)
```

This python script might have come in handy after we changed our org's name from HR to People Operations!

What's the OS module good for?

The OS Module is fantastic for automating manual work that is very easy to codify. For instance, if you have a weekly task where you need to archive a bunch of files into a directory given a certain set of criteria, you can use the OS module to do that. Another example is if you need to lock permissions to files based on a set of criteria (contains sensitive information or has region/ manager specific data), the OS module can help you do that as well.

RANDOM TIDBITS YOU SHOULD KNOW

In the process of teaching this course, I discovered that there dozens of ways to do any one thing in Python. While this provides a great deal of flexibility to the power user, it can also be very confusing for the beginner. As a result, I decided to leave all of this random (but essential) material off until the end.

Running Python Scripts without Invoking Python

If your python script is properly formatted with a shebang at the top (#!/usr/bin/python...). You can run your python script without using the python command, just by passing the name of the script. For instance:

alberthwang@gswitch:~/python$ week3/test.py

Is functionally the same as running the command:

alberthwang@gswitch:~/python$ python week3/test.py

If you want to run a python script from the current directory, you can can specify it with a "./" in front of the script (indicating the current directory).

alberthwang@gswitch:~/python/week3$ ./test.py

When reading documentation, you will often see python scripts run like this instead of invoking the python command and passing the file name.

Multi-Line Strings

Python has many different conventions for multi-line strings. I didn't teach these in the class because I thought the indentations would confuse beginners and end up creating tons of needless debugging hours.

```
  # Long string with line breaks
  long_str1 = """This is a very long multiline string that
spans many lines and is hard to work with...Good luck! and
Good Night!"""

  print long_str1

  print ''

  # Long string w/out line breaks
  long_str2 = """This is a very long multiline string that \
spans many lines and is hard to work with...Good luck! \
and Good night!"""

  print long_str2

  print ''
```

```
# Long string w/out line breaks - my preference
long_str3 = ('This is a very long multiline string that '
             'spans many lines and is hard to work with...'
             'Good luck! and Good night!')

print long_str3
```

As you can see, I strongly prefer the formatting in long_str3, using the parantheses and then placing smaller substrings inside of it on new lines. This helps to preserve the integrity of the indentation and is much more readable in my opinion.

Please use one of these conventions rather than lots of += for long strings.

String Formatting

Python offers a variety of rich string formatting and templating functions. The one I'm showing here is positional string templating. First, you create a string template with positional values (%s) in it. Next, you put a % and a tuple with the values to replace the positional values. Values are replaced left to right.

```
print 'Hi there, %s - I see you are from %s!' % ('albert', 'chicago')

print ''

loc_str = 'Hi there, %s - I see you are from %s'
print loc_str % ('john', 'dallas')

print ''
```

Use this technique in place of += and replace() when formatting a string with dynamic values.

Join()

When you have a list of string and want to create one master string from the list using a single string as a separator, you can use the join() method.

```
name_list = ['albert', 'sauwei', 'hwang']

# albert,sauwei,hwang
print ','.join(name_list)

# albert+sauwei+hwang
print '+'.join(name_list)
```

Please use this in place of for loops with +=

Rounding Floats

Since we worked almost exclusively with integers for this class, I didn't teach how to round floating point numbers. Python has a built-in `round()` function. When called by itself, it rounds the number to closest integer:

print round(10.87) # 11
print round(5.23) # 5

When called with a 2nd integer parameter (n), it rounds the float to the precision of n number of decimals points.

```
print round(10.87) # 11
print round(5.23423) # 5

print ''

print round(10.87, 1) # 10.9
print round(5.2342, 3) # 5.234
```

This will obviously be very useful for those of you working with lots of data.

Working with Python and Data

While not as good as some data analysis specific languages (e.g. R) for work with data, Python has its fair share of powerful features that allow rapid data sorting, filtering, and analyses.

If you're curious, I would Google:

- sets
- list comprehensions
- itemgetter, attrgetter

FINAL THOUGHTS AND ADVICE

Abide by the Google Python Style Guide:

I tried teaching some of the style guide in this class but left out several parts because I thought they would confuse people who are completely new to programming (besides who wants to spend all their time learning the correct way to format `docstrings`?). Please look over the guide and change your style to fit the official Google form:

http://www.corp.google.com/eng/doc/pyguide.xml

Comments, comments, comments!

Always put tons and tons of comments in your code! It might seem like a hassle now but it will really help you understand your own logic when you need to alter the code. You have no idea how much time is wasted by engineers desperately trying to read each other's code (or even their own if they've been away from it for a while). The style guide also shows you how to write module level, class level, and function/method level comments (or `docstring`s).

Simplicity and Elegance

Really try to cut the number of lines of code you write. If something can be written in 1 line rather than 2, try to use 1 line. Python was designed for simplicity and has many native functions to accomplish this (enumerate, iteritems, map...just to name a few we covered in class).

Encapsulation

Repeating code is a great way to break your own scripts and confuse yourself. Always try to use functions wherever possible because it encapsulates execution in 1 place and you will only need to change it in 1 place rather than 5! Encapsulation really aids in logic programming and design and even though python doesn't enforce this as strictly as other languages, you should always do it for yourselves with functions/methods, classes, and importing modules in to perform functions rather than adding the same function to 2 modules!

Ask for help

Google has a wealth of knowledgeable engineers, developers, designers, etc. Whatever you're doing, as long as it involves programming, please know you can **ALWAYS** come to me. I'm more than happy to help or help investigate.

There are also a ton of great online resources like the official python docs:

http://docs.python.org/

There are also a ton of online forums for programmers to ask their questions:

http://stackoverflow.com/

If that doesn't work, **the guy who created python, Guido Van Rossum also works at Google (in the SF office) and is a really nice and helpful guy.**

Alex Martelli, widely regarded as one of the top experts in python programming also works here too! (on main campus)

Alex actually did my python readability review, and he was awesome - couldn't have asked for a better experience.