Assignment 5

(Spoiler alert, in case you haven't looked at it: my solution to the previous assignment is below.)

For this assignment, let's take our configuration class from our last assignment solution and alter it so that it writes to a pickle object instead of a text file.

Here is my solution to the previous assignment. And I'd like to add another enhancement: instead of passing a filename to the constructor, we'll just pass an identifying name. We'll start by hard-coding a configuration directory inside the class itself. And I want to do this because we may as well put all of our configuration files in one directory, and in so doing we can enhance the feeling of a built-in configuration system by just using a name to choose the config we need.

import os

**config\_directory = '/home/dblaikie/configs/' # added: see below**

class ConfigKeyError(Exception):

def \_\_init\_\_(self, this, key):

self.key = key

self.keys = this.keys()

def \_\_str\_\_(self):

return ('key "{0}" not found. Available keys: '

'({1})'.format(self.key, ', '.join(self.keys)))

class ConfigDict(dict):

def \_\_init\_\_(self, filename):

self.\_filename = filename

if not os.path.isfile(self.\_filename):

try:

open(self.\_filename, 'w').close()

except IOError:

raise IOError('arg to ConfigDict must be a valid filename')

with open(self.\_filename) as fh:

for line in fh:

line = line.rstrip()

key, value = line.split('=', 1)

dict.\_\_setitem\_\_(self, key, value)

def \_\_getitem\_\_(self, key):

if not key in self:

raise ConfigKeyError(self, key)

return dict.\_\_getitem\_\_(self, key)

def \_\_setitem\_\_(self, key, value):

dict.\_\_setitem\_\_(self, key, value)

with open(self.\_filename, 'w') as fh:

for key, val in self.items():

fh.write('{0}={1}\n'.format(key, val))

OK, we've established a home for our configs. And our purpose here is to change the underlying data storage to a pickle file. So, what do we need to change?

* Because we decided to have a hard-coded directory where all of our configs will live, we'll need to join the name passed to the constructor to the config directory before checking to see if the file exists.
* The files should be pickle files. You can name them whatever you want -- the convention we will use is to have our pickle files end with the extension **.pickle**.
* Opening, reading and parsing the file can be replaced with **pickle.load(filehandle)**, where **filehandle** is the filehandle of the (freshly opened) pickle file. (Note: since the pickle file is pickling the object (which is essentially a **dict** with additional features) you can use **self.update(pickled\_dict)**, where **picked\_dict** is the previously pickled object, to load the instance with the retrieved key/value pairs.
* All the steps in preparing the config text (**a=b**, etc.) and writing it to file can be replaced with **pickle.dump(self, filehandle)**.

And this is pretty cool, because although we're changing something funamental about the inner workings of the configuration class, we haven't changed the interface at all. So if this class were in production use in our organization or had been distributed publicly, none of the code that uses this class would have to change. There would be 0 changes required in the calling code, because the interface stays the same. And this is part of the benefit of encapsulation. Remember encapsulation? The first pillar of object-oriented programming.

Good luck!