Top of Form



Bottom of Form

Top of Form



**Python 3: The Python Environment  
Lesson 12, Project 1**

Handed in: 11 Jun 2015 09:36:49PM Graded: 12 Jun 2015 05:30:22PM

**Here are your instructions:**

Create a **Python3\_Homework12** project and assign it to your **Python3\_Homework** working set. In the **Python3\_Homework12/src** folder, copy **property\_address.py** and **test\_property\_address.py** from the last project. Modify **property\_address.py** to accept the following options if called directly, with the five address values used to instantiate an Address class if no parser errors are thrown.

|  |  |  |  |
| --- | --- | --- | --- |
| **option** | **default** | **address?** | **task** |
| -l/--level | INFO | yes | Sets the log level to DEBUG, INFO, WARNING, ERROR, and CRITICAL |
| -n/--name | Throws a parser error if empty | yes | Sets the name value of the Address object |
| -a/--address | Throws a parser error if empty | yes | Sets the street\_address value of the Address object |
| -c/--city | Throws a parser error if empty | yes | Sets the city value of the Address object |
| -s/--state | Throws a parser error if empty | yes | Sets the state value of the Address object |
| -z/--zip\_code | Throws a parser error if empty | yes | Sets the zip\_code value of the Address object |

If you run your code with the following command-line arguments:

**property\_address.py -n Tom -a "my street" -c "San Diego" -s "CA" -z 21045**

...you should see something like this in **property\_address.log**:

2010-10-11 14:48:59,794 - INFO - \_\_init\_\_ - Creating a new address

**Note:** When you use the ${string\_prompt} or direct input option in the Eclipse run configuration, you can't use single quotes ('). If your input requires quotes, use double quotes ("), so the full string is parsed rather than breaking at white space. If you run your code without command-line arguments, you should see:

Usage: property\_address.py [options]

property\_address.py: error: options -n, -a, -c, -s, -z are required

If you run your code with the following command-line arguments:

property\_address.py -l WARNING -n Tom -a "my street" -c "San Diego" -s "CA" -z "EZ 123"

...you should see:

usage: property\_address.py [options]

property\_address.py: error: option -z requires a valid 5-digit US zip code

...AND you should see something like this in **property\_address.log**:

2010-10-11 17:10:32,702 - ERROR - ZipCodeError - ZIPCODE exception

**Note:** Your date and time values will vary.

Now, modify your **propertyaddress.py** program to use the configparser library to load the settings from the following **property\_address.cfg** config file:

[log]

format = %(asctime)s - %(name)s - %(levelname)s - %(message)s

output = homework12.log

[validators]

zip\_code = \d{5}\-\d{4}$

state = [A-Z]{3}$

As you can see, the log formatting, log output, and Address validators are different from what you currently have.

Your project must meet the following conditions:

The tests in test\_property\_address.py must run correctly. Also, the results of your logfile should appear as below:

2010-10-11 17:34:38,968 - root - ERROR - STATE exception

2010-10-11 17:34:39,098 - root - ERROR - ZIPCODE exception

2010-10-11 17:34:39,112 - root - INFO - Creating a new address

2010-10-11 17:34:39,113 - root - ERROR - NAME exception

2010-10-11 17:34:39,113 - root - INFO - Creating a new address

2010-10-11 17:34:39,113 - root - INFO - Creating a new address

**Note:** Your date and time values will vary.

Submit **property\_address.py**, **test\_property\_address.py**, and **property\_address.cfg** when they are working to your satisfaction.

**Your Comment:**

Kirby,

This was a great assignment! It blended multiple concepts very nicely (logging, parsing command

line args, properties). It was challenging from the perspective of having to work through the

different steps involved, but not so challenging that you didn't feel like you were getting anywhere.

The end result is the makings of a very robust testing and logging framework. Very nice.

-Jason

**Items Handed In**

* [Open Project Handed In](https://students.oreillyschool.com/student/project/?/.handin/147-6661-1/com.ost.jwoloson.147.6661.1.Python3_Homework12.zip)

**Overall Comments:**

Good work.

Best to avoid absolute paths where possible e.g. I never run your code in the

same subtree you do so:

config.read('V:/workspace/Python3\_Homework12/src/property\_address.cfg')

has to be changed to:

config.read('property\_address.cfg') -- at which point everything runs perfectly.

A theme here is different ways to feed programs initial arguments / data as that's often

the user's primary role (to define starting conditions). This partially derives from an era

when end users practically never looked at or used source code. The program was

a "black box". If you have source, then you have the option to change the source

itself (a good point made by R0ml at one of his recent OSCON talks).

The ways we're looking at: settings.py (a Python config file, written in Python,

usually to define a bunch of read-only globals); a configuration file (common

in Windows as well); command line arguments. It's not either/or: many

programs will have a command line option but when the command line gets

to be 100s of characters, it's nice to have the config file or some other file to

store initial stuff. The command line and config file pair for the ray tracer POV-Ray

(povray.org) is a good one to study, as an example (plus it's a very impressive

program which I've used quite a bit in my time, mostly for geometric studies).

-Kirby

**Grade:**

Great

[Take Me Back](ostreturn:)

© 2014, O'Reilly Media, Inc. All rights reserved.

Bottom of Form