Red Hat Python Lab - Lesson 5: Command line arguments, Configuration, Serialization

Daniel Mach <dmach@redhat.com>
Martin Sivák <msivak@redhat.com>

Useful links

- Python Quick Reference: http://rgruet.free.fr/#QuickRef
- Python docs: http://www.python.org/doc/
- PEP 8: http://www.python.org/dev/peps/pep-0008/
- pep8.py: http://pypi.python.org/pypi/pep8/
- pylint: http://www.logilab.org/project/pylint
- Epydoc: http://epydoc.sourceforge.net/

Interesting links:

- Letajici cirkus on root.cz: http://www.root.cz/serialy/letajici-cirkus/
- py.cz: http://www.py.cz/
- Unladen swallow: http://code.google.com/p/unladen-swallow/

Don't forget

- Batteries included -> read docs, find existing function, use it
- Follow PEP 8
- Pylint is your friend
- Write unit tests
- Objects are not everything, don't make your code over-complicated
- Return from functions as soon as possible -> more readable code, better indentation

Command line arguments

To avoid hardcoding runtime arguments directly into the source code, we can use couple of approaches. One of them is passing the arguments through command line interface.

Basic access to cmdline arguments is provided by **sys.argv** list. It is standard python array in the form of:

```
import sys
[program_name, argument1, argument2, ...] = sys.argv
```

This is the same structure like you may be used to in C language.

Option parser

Python provides higher level helper to make parsing options in differrent formats easier. You just construct a list of accepted options, their arguments, description... and create an object.

```
import optparse

parser = optparse.OptionParser("%prog [options]") # 1st argument is usage, %prog is replaced with sys.argv[0]
parser.add_option(
    "-s", "--server",    # short and long option
    dest="server",       # not needed in this case, because default dest name is derived from long option
    type="string",       # "string" is default, other types: "int", "long", "choice", "float" and "complex"
    action="store",      # "store" is default, other actions: "store_true", "store_false" and "append"
    default="localhost", # set default value here, None is used otherwise
    help="IRC server address",
)
options, args = parser.parse_args()
# options.key = value
# args = [arg1, ... argN]

print options.server

# use parser.error to report missing options or args:
parser.error("Option X is not set")
```

Task: Make the bot configurable using command line.

We would like you to enhance our ircbot so it accepts (and uses) the following arguments:

- --help, -h prints the help message and ends
- --server, -s
- --port, -p
- --delay, -d sets delay for the side-thread

Create new file **conf_parse.py**, and write function **parse_argv**, which will parse the command line arguments, and returns tuple containing parsed options and args.

Configuration files

Using the command line gets quite mundane after some time, so we prefer if all "constant" options can be set using some kind of configuration file.

There are also multiple ways to accomplish this:

- import or eval - Python source file acting as config file
- ConfigParser - built in module dealing with .ini files
- Serialization modules - we will deal with these later

ConfigParser

- .ini file parser
- key-value pairs in sections, both keys and values are strings

ini file example:

```
[section]
key = value
```

Basic usage:

```
import ConfigParser

config = ConfigParser.SafeConfigParser()
config.read("ircbot.ini")

# get values from config parser
config.get(section, option)
# config.get{boolean, float, int}

# set custom values
config.add_section(section)
config.set(section, key, value)
```

A bit of advanced string parsing using shlex

Standard module shlex contains split function which (among other possible configurations) can split a string into parts while taking care not to split apostrophed strings.

```
>>> import shlex
>>> shlex.split("""./test arg1 arg2 "arg3" "arg4 arg4b" """)
['./test', 'arg1', 'arg2', 'arg3', 'arg4 arg4b']
```

Advanced: Pythonizing ConfigParser

To make the class behave more in the Python way (eg. access attributes using the *dot* notation, just derive your own object and add some "advanced magic" (modification to the class internal __ methods).

For an example of how to do it, take a look at this source file.

The change requires you to understand internal methods of classes. You can read more about them in the documentation. Take a good look at __getattr__ method.

Task: Make command line arguments also configurable using config file

We would like to configure server name, port and delay using configuration file, which will be passed as the last argument on the command line. Command line options (when used) should override the configuration file.

```
./ircbot.py --port 6667 settings.ini
```

Add functions **load_config** and **load** to the previously created file **conf_parse.py**. Function **load_config** parses the configuration file, and returns dictionary with parsed options. The **load** function then combines the command line arguments and config file options, and returns dictionary with correct key-value pair for each option. Don't forget that command line arguments should override the configuration file!

Serialization

pickle and cPickle

In case you need to store python structures to a file (or database), you have couple of couices:

- use builtin *repr* function to get a representation of the python data structure - this representation can be then evaluated in python to get the value back

```
def fce():
  pass

a = ["a", 1, 2, ["a", (5, 6), fce]]

repr(a)
"['a', 1, 2, ['a', (5, 6), <function fce at 0x7f54513377d0>]]"
```

- use builtin module pickle and let python do the serialization and loading for you (with more efficient representation)

```
import cPickle as pickle

file_obj = open("spam", "wb")
pickle.dump(obj, file_obj)

file_obj = open("spam", "rb")
obj = pickle.load(file_obj)

string = pickle.dumps(obj)
obj = pickle.loads(string)
```

Gzip compression

Read a gzipped file:

```
import gzip

f = gzip.open("ircbot.log.gz", "rb")
data = f.read()
f.close()
```

Write to a gzipped file:

```
import gzip

data = "..."
f = gzip.open("ircbot.log.gz", "wb")
f.write(data)
f.close()
```

FI only: kolokvium task

Using the knowledge from this and previous lessons write a standalone line based (testable by telnet) network server using SocketServer class that will write everything sent to it into a gzipped log file. It should be configurable (host, port, destination file) using command line and configuration file.

A second part of the task is new filter for your ircbot, that will send everything to the logging server you just implemented. You can hardcode the address and port of the server into the source code, but we will like it better if you enhanced your configuration file methods.