# Red Hat Python Lab - Lesson 3: Modules, Sockets, Regular Expressions, Creating Simple Server

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

## Modules

- import module
- from module import foo, bar
- from module import *
- Use __import__ or imp module for advanced work with imports

```
# correct:
import os.path
os.path.isdir("/usr/bin")
os.path.isfile("/usr/bin/python")

# also correct:
from os.path import isdir, isfile
isdir("/usr/bin")
isfile("/usr/bin/python")

incorrect:
import os, os.path, sys
```

## Modules - writing a new module

- A module can be either a python file or a directory with __init__.py inside
- Modules have to be located under current directory or in sys.path
- PYTHONPATH environment variable

You should have the following structure after you split your code into modules:

```
.
├── plugins
│   ├── __init__.py
│   ├── calculator.py
│   ├── karma.py
│   ├── shutdown.py
│   ├── word_count.py
│   └── state.py
├── interface.py
├── ircbot.py
└── test_interface.py
```

## Task - move commands in hooks to modules

plugins/calculator.py

```
import state
def command_calculator(msg):
```

plugins/karma.py

```
import state
KARMA = {}
def command_karma(msg):
def filter_karma(msg):
```

plugins/shutdown.py

```
import state
def command_shutdown(msg):
```

plugins/word_count.py

```
import state
WORD_COUNT = 0
def filter_word_count(msg):
```

plugins/__init__.py

```
from calculator import command_calculator
from karma import command_karma, filter_karma
from shutdown import command_shutdown
from word_count import filter_word_count

COMMANDS = {
    "=": command_calculator,
    "karma": command_karma,
    "SHUTDOWN": command_shutdown,
}

HOOKS = [
    filter_karma,
    filter_word_count,
]
```

## Regular Expressions

- module: re
- for performance reasons always use re.compile in **global scope** or in **class constructor**
- use raw strings (r"text") if you want to avoid escaping
- compile(), match(), search(), sub()
- Match object: group(), groups(), groupdict()

```
import re
EMAIL_RE = re.compile(r"(.*)@(.*)")

def split_emails(email_list):
    result = []
    for email in email_list:
        result.append(EMAIL_RE.match(email).groups())
    return result

split_emails(["spam@example.com", "eggs@example.com"])
# [('spam', 'example.com'), ('eggs', 'example.com')]
```

## Network connections

- **socket** module
- connect, listen, receive, send, getaddrinfo - as in C
- bind, listen, accept - as in C
- socket.makefile to create Python file object - read, write, close, flush

```
# import the module
import socket

# resolve address
for (family, socktype, proto, canonname, sockaddr) in socket.getaddrinfo("www.fit.vutbr.cz", 80):

s = socket.socket(family, socktype, proto)

# create CLIENT a socket and connect it
s.connect(sockaddr)

# or create a SERVER socket and wait for connection
s.bind(sockaddr)
s.listen()
client = s.accept() # the rest is the same, only use client instead of s

# set timeout to 5 minutes
s.settimeout(300)

# get info about peer
s.getpeername()

# raw read and write
s.recv(size)
s.send(data)

# get file object and read one line
f = s.makefile()
f.readline() # reads including line terminators
```

## Testing and debugging

### Unit tests

Please see the slide from the first lesson and the provided file for some examples.

### Debugger

Python contains an integrated debugger **pdb** to help you with testing and debugging. If you want to start the program from the beginning in debug mode, launch it as:

```
python -m pdb script.py
```

You can also add the debugger call to the place you want to debug, just import pdb and then at the proper place write:

```
pdb.set_trace()
```

The last nice thing for us at the moment will be post mortem debug. Just call **pdb.post_mortem()** in the exception handler.

```
try:
    raise Exception()
except:
    import pdb
    pdb.post_mortem()
```

## Task: Connect to the IRCBot over network

The main excersise for today is to implement server interface to your bot so you can control it over network (using telnet or netcat).

Create the emhasised methods' body in the IRCBotTelnetInterface class.

```
class IRCBotTelnetInterface:
    def __init__(self, host, port):
        ... the code from provided template ...
```

It is also necessary to write different read and write methods for this interface.

```
    def read(self):
        """Gets valid _sfile for current client and then reads one line from network."""
      # use self.client() to get a valid file object
      # read a line and return it
      # in the case the line was empty or exception was thrown
      # -- call self.disconnect() and try again

    def write(self, message):
        """Send one line to the network and call flush."""
```

The last part is to write network specific helper functions below.

```
    def client(self):
        """Returns a file object which can be used to communicate with currently connected client.
           If there is no client, waits for someone to connect and creates the self._client and self._sfile values."""
        # check if the client variables are filled
        # return the file if they were, else continue
        # get new client using server socket's accept()
        # create new file object using client's makefile

    def disconnect(self):
        """Deletes the values from _client and _sfile values to signalize the client has disconnected."""
    def open(self):
        """Prepare the server and start listening."""
        # get all possible values for the address using getaddrinfo
        # if not proper family, check the next one (continue)
        # if not proper socktype, ditto
        # now you have all the info you need to setup a server
        # create new socket
        # call bind and listen, watch for errors
        # break! you only want to set the thing up for one address

    def close(self):
        """Close the server."""
```

As the last step towards the network you have to update your __main__ section to create instance of this new interface, initialize the server and start the IRCBot.