# Week 4
# Python and OptParse

30 January 2019

CS 35L Lab 4

Jeremy Rotman

# Announcements

- Assignment #3 is due Saturday by 11:55pm
- For Assignment #10
  - You should begin to choose stories
  - Email me to tell me what you are choosing at least a week before you have to present
  - Here is the link to see what stories people have signed up for already
  - Here is the link to sign up to present
    - **Sign up to present by Friday, February 1st**

# Questions?

# Outline

- ➔ Python optparse library
- ➔ Homework #3

# Optparse Library

- Library to help with parsing command-line options
- Argument
  - String entered on the command line and passed into the script
  - Arguments are elements of sys.argv[1:]
    - sys.argv[0] is the name of the program being executed
- Option
  - An argument used to supply information to guide or customize the execution of a program
  - Usually, one dash followed by a single letter (-x or -F)
  - OR, two dashes followed by word(s) (--filename or --dry-run)

# Optparse Library

- Option Argument
  - An argument that follows an option and is closely associated to that option
  - It is consumed from the argument list when the option is
  - It can be a separate argument, or part of the option:
    - E.g. --file foo.txt
    - OR --file=foo.txt

# Homework 3

- randline.py
  - You can run it with
    - `./randline.py -n N filename`
  - This takes *N* random lines from *filename*
- Options and Arguments:
  - -n specifies number of lines to write
    - This is an option
  - *N* is the number of lines we want
    - This is an option argument
  - *filename* is the file we are taking lines from
    - This is an argument

```
#!/usr/bin/python

import random, sys
from optparse import OptionParser

class randline:
        def __init__(self, filename):
                f = open (filename, 'r')
                self.lines = f.readlines()
                f.close ()

        def chooseline(self):
                return random.choice(self.lines)


def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION]...
FILE Output randomly selected lines from
FILE."""
```

Tells the shell which interpreter to use

Import statements, similar to include statements
Import OptionParser class from optparse module

The beginning of the class statement: randline
The constructor
Creates a file handle
Reads the file into a list of strings called lines
Close the file

The beginning of a function belonging to randline
Randomly select a number between 0 and the size of lines and returns the line corresponding to the randomly selected number
The beginning of main function
version message
usage message

```
parser = OptionParser(version=version_msg,
                      usage=usage_msg) parser.add_option("-n",
"--numlines",              action="store", dest="numlines",
        default=1, help="output NUMLINES  lines (default
1)")

options, args = parser.parse_args(sys.argv[1:])

try:
    numlines = int(options.numlines)
except:
    parser.error("invalid NUMLINES: {0}".
          format(options.numlines))
if numlines < 0:
    parser.error("negative count: {0}".
format(numlines))
if len(args) != 1:
    parser.error("wrong number of operands")
input_file = args[0]
try:
    generator = randline(input_file)
    for index in range(numlines):
        sys.stdout.write(generator.chooseline())
except IOError as (errno, strerror):
    parser.error("I/O error({0}): {1}". format(errno, strerror))

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

Creates OptionParser instance

Start defining options, action “store” tells optparse to take next  argument and store to the right destination which is “numlines”.  Set the default value of “numlines” to 1 and help message.

options: an object containing all option args
args: list of positional args leftover after parsing options
Try block
  get numline from options and convert to integer
Exception handling
  error message if numlines is not integer type, replace {0} w/ input

If numlines is negative
  error message

If length of args is not 1 (no file name or more than one file name)
  error message
Assign the first and only argument to variable input_file
Try block
   instantiate randline object with parameter input_file
   for loop, iterate from 0 to numlines – 1
      print the randomly chosen line
Exception handling
   error message in the format of “I/O error (errno):strerror

In order to make the Python file a standalone program

# Optparse Actions

- There are a fixed set of actions already in optparse
  - You should not need to make new ones
- Most of these are related to storing an argument
  - The type of variable you are storing may make a difference though

# Optparse Actions: Store

- The most basic and probably most useful action
- It is also the default action
  - Meaning you don’t technically need to specify it
- This includes 3 arguments in the add_option function
  - action=“store”
    - Declares the action as store
  - type=“string”
    - Declares the type of argument being stored as a string
    - String is default
  - dest=“var_name”
    - Indicates the variable you want to store the argument as

# Optparse Actions: Store

So an example may look like:

```
parser.add_option(“-f”, “--file”,
   action=“store”, type=“string”, dest=“filename”)
```

# Optparse Actions: Booleans

- If you want an option that is simply a flag to turn things on or off, there is a different set of actions similar to store
  - store_true
  - store_false
- These will still need to be stored in a variable, but the type is not important

# Optparse Actions: Booleans

So for example:

```
parser.add_option(“-v”, action=“store_true”,
   dest=“verbose”)

parser.add_option(“-q”, action=“store_false”,
   dest=“verbose”)
```

# More Optparse Actions

- store_const
  - Store a constant value
- append
  - Append this option’s argument to a list
- count
  - Increment a counter by one
- callback
  - Call a specified function

# Default Values for Options

- ➔ Everytime you add an option, you can include a default value
- ➔ Say for example, we want the option for the script to be verbose, but want it to be quiet by default

```
parser.add_option(“-v”, action=“store_true”,
   dest=“verbose”, default=False)
```

# Generating Help Messages

- By default, Optparse uses the -h or --help options to display help messages
- The help messages include multiple useful things

# Program Usage

- When creating your OptionParser, you can specify a program's usage message
- What is a usage message?

# Program Usage

- ➔ When creating your OptionParser, you can specify a program's usage message
- ➔ What is a usage message?
  - ◆ The line at the top of the help file describing how to run the program
  - ◆ Additionally, it might include a description of what the program does

```
Usage: randline.py [OPTION]... FILE

Output randomly selected lines from FILE.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -n NUMLINES, --numlines=NUMLINES
                        output NUMLINES lines (default 1)
```

# Program Usage

For example this creates the usage message in randline.py

```
usage_msg = """%prog [OPTION]... FILE

Output randomly selected lines from FILE"""
parser = OptionParser(usage=usage_msg)
```

# Option Help Messages

- In addition to the usage at the top, the help display also showed help messages for all of the options that you could use
- This message must be defined in the add_option function

# Option Help Messages

Going back to our verbose example before:

```
parser.add_option(“-v”, action=“store_true”,
   dest=“verbose”, default=False,
   help=“Print out extra information about
   the program while running”)
```

# Printing Version Number

- ➔ Similar to usage, when creating the OptionParser object, you can give it a version message
- ➔ This will naturally be tied to the --version option

```
version_msg = “%prog 2.0”
parser = OptionParser(version=version_msg)
```

# Error messages

- ➔ OptionParser objects also have a built in error function
- ➔ This won't actively look for errors
  - ◆ Instead you must anticipate potential errors and call the function when you encounter these errors
    - E.g. being passed two boolean options that contradict each other
    - optparse does catch some things, like option argument type
- ➔ Does a few things
  - ◆ Print the usage message to stderr
  - ◆ Print the error message (the function parameter) to stderr
  - ◆ Exit with error status 2

# Homework 3

- You will be creating shuf.py
  - This should function essentially the same way as GNU shuf
  - Including the options:
    - --input-range (-i), --head-count (-n), --repeat (-r), and --help
  - Support any number (including zero) of non-option arguments, as well as the argument "-" meaning standard input
  - You will have to port your shuf.py to Python 3

# Homework 3 Hints

- For Q4:
  - Lookup “automatic tuple unpacking”
- If you’re unsure how shuf.py should output something
  - Try it on GNU shuf
- Use randline.py as a starting point
  - There are still plenty more to look up about arguments, options, and option arguments
- If you have troubles with optparse under python 3, you can use argparse instead.

# Questions?