PTN-102 - Python 2.x-3.x in practice

Activity Guide

Release 1 rev20

Component Soft

July 25, 2014

Contents

Lab 1: Simple scripts

Task 1: Celsius - Fahrenheit conversion

Write a program, that converts between Celsius and Fahrenheit. The script has two inputs

- the type of temperature (c,C,f,F)
- the temperature value (floating point value)

```
$> ./lab1_task1.py
Input type: c
Input temperature: 100
Output: 212 F
$> ./lab1_task1.py
Input type: d
Invalid type
$>
```

Hint: C= 5 * (F-32) / 9

Task 2: Celsius - Fahrenheit conversion - infinite loop

Rewrite the previous program to do the task in infinite loop. The program should quit on `EOF` signal (`ctrl+d` on UNIX, `ctrl+z` on Windows) Use:

- built-in function raw_input
- try/except for EOFError in `raw_input`

Hint:

See the manual for `raw_input` at python_func

```
>>> s = raw_input('Input type: ')
    Input type: <ctrl+d>
    Traceback (most recent call last):
    File "<stdin>", line 1, in ?
    EOFError
```

Task 3: Name - email conversion

Read a name from input (order surname, first name), and convert it to a ‘corporate’ email.

```
$> ./lab1_task3.py
Name: Kovács Béla
bela.kovacs@foobar.com
$> ./lab1_task3.py
Name: James Joe O'Neal
oneal.joe.james@foobar.com
```

Assumptions:

- Input has **utf-8** encoding, and might contains accent letters.

Hint:

- Use `str.decode`, `str.encode` for coversion between `str` and `unicode`
- `unicodedata.normalize`, `unicodedata.category` to wipe accent characters

```
>>> import unicodedata
>>> s=u'áé'
>>> s
u'\xe1\xe9'
>>> s2=unicodedata.normalize('NFD',s)
>>> s2
u'a\u0301e\u0301'
>>> for c in s2: unicodedata.category(c)
...
'Ll'
'Mn'
'Ll'
'Mn'
```

Lab 2: Scripts with sequence types

Task 1: Name - email conversion using input file

Rewrite the previous script to accept input files also.

```
$> ./emailgen.py input.txt
jakab.gipsz@foobar.com
fu.geza@foobar.com
$> ./emailgen.py
Name: Kovács Béla
bela.kovacs@foobar.com
$>
```

Hint:

- Use `sys.argv`

Task 2: UID-GID match

From file `/etc/passwd` print all user names, where UID is smaller than GID

- `/etc/passwd` use ":" as delimiter
- UID is field 3, GID is field 4
- try to solve the problem using list comprehensions

```
$> ./uidgidcmp.py
aaa (uid: 100, gid: 101)
bbb (uid: 14, gid: 100)
```

Warning: Compare values as int not as strings!

```
>>> a="100"
>>> b="99"
>>> a<b
True
```

Task 3: Word frequency count

Count the number of occurrences of a word in a file.

- Take words as case insensitive
- Print a histogram as follows

```
$> cat file1
alma KortE ALMa szilva
kORte korte korte
banan
$> ./wordcount.py file1
alma   | @@     | 2
szilva | @      | 1
korte  | @@@@   | 4
banan  | @      | 1
```

Task 4: Top-N word with command line option, and ordering

Make two extensions of the previous script

- Script must handle multiple files. If files are missing, or when `-` is specified as file, the standard input is used
- Result are printed in the order of values
- Script accept option `-t` or `--top`, with a value, that specifies the top N values to be printed
- If `-t` option is missing, all entries are printed

```
$> cat file1
alma KortE ALMa szilva
kORte korte korte
banan
$> ./wordcount.py --top 2 file1
korte  | @@@@ | 4
alma   | @@   | 2
```

Hint: Use module getopt for option parsing, and module fileinput to handle input files.

Task 5: Process output from outer programs

Write a script, that determines whether free space left on local filesystems is critically low

- The level of critical usage is provided as input parameter
- Use outer program `df` with `-Ph` option
  - Column 5 (`Use%`) will tell the usage info
- Print all filesystem, that are filled up over the critical level

Implement this task with two functions **df()** and **alert_used(list,int)**.

- `df()` processes the output from command `df -P`, and creates a list of dictionaries as follows:

```
[
  { 'filesystem': '/dev/sda1', 'size': '20',
      'used': 12, 'avail': 6.4, 'use': 64, 'mounted_on': '/'},
  { 'filesystem': '/dev/sda3', 'size': '20',
      'used': 11, 'avail': 7.5, 'use': 60, 'mounted_on': '/vmware'},
]
```

- `alert_used()` processes the above list, with the desired threshold (second parameter), and produces the output below

```
$> df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1              20G   12G  6.4G  64% /
/dev/sda3              20G   11G  7.5G  60% /vmware
tmpfs                 506M   12K  506M   1% /dev/shm
$> ./dfalert.py 62
Overused filesystems:
/ (/dev/sda1)
```

Hint: Use subprocess.Popen

```
>>> import subprocess
>>> p=subprocess.Popen(['ls','-l'],stdout=subprocess.PIPE)
>>> p.stdout.readlines()
['total 0\n',
'drwxrwxr-x 2 atis atis 60 May 15 10:08 dir2\n',
'-rw-rw-r-- 1 atis atis  0 May 15 10:08 file1\n']
```

Task 6: Copy files with extras

Implement a script that copies files with .log suffix. Options:

- Optional switch `--to` specifies a directory to copy the log files to.
- Optional switch `--tar` specifies a 'tar' file, so that the script tar all log files into that tape archive.

Without options, the script just lists the log files found.

```
$> ./mycopy.py --to /archive /var/log
/var/log/mail.log copied
/var/log/last.log copied
$> ./mycopy.py --tar /root/logs.tar /var/log
/var/log/maillog
/var/log/lastlog
$> ./mycopy.py /var/log
/var/log/maillog
/var/log/lastlog
$> ./mycopy.py foo.txt
foo.txt is not a valid directory
```

Hint:

- Use glob to find files
- Check if `*.log` is really a file
- Use tar as an external command to create the tar archive (subprocess.call)

Note: If time permits, create an alternate version as follows

- use os.listdir and regexp to find logfiles
- use tarfile for archive creation.

Task 7: Top-N file in a directory

Write a script that list the top large files in a directory. (Files in sub-directories are ignored) The number of files listed is limited by option `-n`. If `n` is not specified, the default is to list the top 5 files.

```
$> ./topfiles.py -n 2 /root
/root/alma.txt  20000
/root/korte.jpg 4000
```

Hint:

- Use os.path to determine the size of a file

Lab 3: Using regural expressions

Task 1: Temperature conversion with regexp

Rewrite script cf_conv as follows

```
$> ./fcre.py
Input temperature: -32.5 c
-26.5 F
```

- The input format starts with a float number (with or without sign), followed by temperature format letter (c,C,f,F).
- Leading zero can be omitted , such as `.5 F`
- Use `re.match` to check whether the input format is correct.

Hint: See Regular expressions

Task 2: Backreferencing

Write a script that reads `/usr/share/dict/words` and finds all words that matches the following criteria:

- The word contains a sequence of repeating letters, and this sequence is repeats once again at the end of line.
- The length of the sequence is configurable as command line parameter of the script

```
$> ./reg.py 3
AAAAAA
$> ./reg.py 2
...
yoo-hoo
youthlessness
zeallessness
zoozoo
```

Note: `/usr/share/dict/words` contains one word per line

Task 3: Substitution

Write a script that switches the first and the last word in specified lines of a file.

```
$> ./reg_sub.py
Usage: ./reg_sub.py -l start,end filename
$> tail -n +10 file2|head -3
10  line10  xxxx
11  line11  xxxx
12  line12  xxxx
$> ./reg_sub.py -l 10,12  file2
$> tail -n +10 file2|head -3
xxxx        line10  10
xxxx        line11  11
xxxx        line12  12
```

- Please **do not** buffer the file into the memory. Operate directly on the file line by line
- Provide a usage text on parameter errors

Hint:

- Use re.sub
- Use flag `r+` for io.open
- Use file.seek and file.tell to navigate inside the file

Task 4: Process multiline expression

See the following database format.

```
METACLUSTER=SUNWCmreq
NAME=Minimal Core System Support
DESC=Internal private metacluster not installable by end users.
VENDOR=Sun Microsystems, Inc.
HIDDEN=y
REQUIRED=y
VERSION=2.4
SUNW_CSRMEMBER=SUNWCfca
SUNW_CSRMEMBER=SUNWCfct
SUNW_CSRMEMBER=SUNWCfmd
..
END
```

Each block starts with the keyword `CLUSTER` or `METACLUSTER` and ends with `END`. The file represents a hierarchical database, so members in the above example also has a section to describe their sub elements

```
CLUSTER=SUNWCfca
NAME=Sun ISP Fibre Channel Device Drivers
DESC=Sun Fibre Channel FCA framework, port drivers, and device drivers
VENDOR=Sun Microsystems, Inc.
VERSION=1.0
SUNW_CSRMEMBER=SUNWqlc
```

```
SUNW_CSRMEMBER=SUNWemlxs
SUNW_CSRMEMBER=SUNWjfca
END
```

Write a python script to that first parses the database file, then lists names of all leaf elements for a cluster name, that is provided on the standard input.

```
$> ./reg2.py /labfiles/clustertoc
Enter a group name: SUNWCssh
SUNWCssh:   SUNWsshcu,SUNWsshdr,SUNWsshdu,SUNWsshr,SUNWsshu,
```

The file is found under `/labfiles/cluster.toc`

Lab 4: OOP in Python

Task 1: Simple object

Please create a object named **Person**, with the following attributes:

1. `Person` is a new style class
2. Attributes: `name`, `age`, `gender`
3. Class Person should have a constructor, that makes `name` and `gender` obligatory, but `age` can be 0 if undefined

Task 2: Overload, static variables, built-ins

Extend you previous object to implement the following

1. Rewrite constructor

- Use the `__slots__` to limit the number of args to the previously mentioned ones. Your class should behave like that:

```
>>> p=Person('Bela','Male',99)
>>> p.alma=1
    Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    AttributeError: 'Person' object has no attribute 'alma'
```

- The constructor should register the new instance into a class static list called `persons`.

2. It should implement a destructor.

- Destructor should remove the current instance from list persons

3. Use the property decorator to intercept getter/setter calls for property 'name'

- Setter should deny the change of the name attribute, except when changing upper/lower case letters in the name

```
>>> p=Person('Bela','Male',99)
>>> print p
Bela/Male/99
>>> p.name='bela'
>>> print p
bela/Male/99
>>> p.name='jozsi'
    Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "use_person2.py", line 37, in name
    raise AttributeError("Changing name is not allowed!")
    AttributeError: Changing name is not allowed!
```

Hint:

- Use `weakref` when mantaining the list of Persons.

```
>>> import weakref
>>> def callback(wr): print "Del called on ", wr
>>> class C(object): pass
>>> c=C()
>>> r=weakref.ref(c,callback)
>>> r
<weakref at 0xa277aa4; to 'C' at 0xa23e8ec>
>>> del(c)
Del called on  <weakref at 0xa277aa4; dead>
```

---

4. Overload the following methods, as in the example below

```
>>> p=Person('Bela','Male',99)
>>> p2=Person('Jozsi','Male',88)
>>> print p
Bela/Male/99
>>> p+=2
>>> print p
Bela/Male/101
>>> p<p2
True
```

- print on a `Person` reference should return `Name/Age/Gender` string
- function `cmp` compares by names

5. Class `Person` has a method `printAll`, that

- prints all created objects sorted by name
- uses the overridden methods

Task 3: Gender as an internal class

1. Extend object `Person` with a new internal descriptor class for parameter `gender`.

- Gender accepts `Male` and `Female` as values, and raises an `AttributeError` on other values

```
>>> p=Person('Bela','Male')
>>> p.gender='ABC'
    Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "Person.py", line 17, in __set__
    raise AttributeError("Invalid value for gender")
    AttributeError: Invalid value for gender
```

Task 4: Make class Person iterable

- Make class Person be iterable like a list (reflecting the internal list persons).
-

Example:

```
>>> p=Person('Bela','Male',99)
>>> p2=Person('Jozsi','Male',88)
>>> print p
Bela/MALE/99
>>> for x in p: print x
...
Bela
Jozsi
```

Hint:

- Use ABC class Sequence from collections
- The value __weakref__ in __slots__ is not needed, if using ABCMeta as metaclass (so when you extend from Sequence).

Task 5: Automatic variables

- Make Person class instances to return the string ‘N/A’ for all undefined attributes

```
>>> p=Person('Bela','Male',99)
>>> p.job
N/A
```

Hint: Use function __getattr__ to intercept calls to unknown attributes

Optional Task 6: Arbitrary function

1. Modify Person to act like that:

```
>>> p=Person('Bela')
>>> p.run()
I am Bela, and I'm runing!
>>> p.smile()
I am Bela, and I'm smileing!
```

- The actions (such as ‘run’,’smile’, etc) are defined in a class static tuple called __actions__
- For something not in __actions__, you should raise an AttributeError

Hint:

- use function `__getattr__` to intercept calls to unknown methods/attributes
- Please note, that in this case, __getattr__ should return a callable object

Task 7: Inheritance

Extend the previous example with new classes

Employee

1. Extends `Person`
2. New attributes: `job`, `salary`, `boss`, `colleague`
3. Attribute boss should be a reference to an `Manager` object
4. Implements the same overloads as for `Person`, and ensure that the call is dispatched to the appropriate `Person` function
5. It should implement an abstract class `Driver`, with functions `getMaxDriveSpeed`, and `getLicenseNumber`
    - `maxDriveSpeed` value should be derived from the age parameter with the following formula:
        - `speed=200-(age-20)*3`

Hint: For abstract class `Driver`, see @abstractmethod

Extra scripts (if you got bored)

Task 1: Processing log files

Create a script, that analyses a squid access log file (see sample, and creates a pie chart report of top level domains visited (such as .hu, .eu , etc)

- Use regexp to extract the required info.
- Take care of FQDN's only. (Don't count requests sent to unresolvable addresses)
- Use reportlab or pyplot to draw the chart
- The resulting document can be any format of `jpg`, `gif`, `ps`

Hint: Use `easy_install` or `pip-python` to install the required libraries

Task 2: Simple chat application

Implement a simple chat program, that communicates over UDP mulitcast. It should work like that:

```
[bash$] chat.py
Enter you nick: **Jozsi**
[Bela] Hello World!





------------------------
Your message: Hi Bela!
------------------------
```

```
[Bela] Hello World!
[Jozsi] Hi Bela!

------------------------
Your message:
------------------------
```

- Messages are sent as plain text in form of `User###Message`
- Use multicast address `240.1.2.3`, port `10000`

Hint:

- Use py-multicast for udp communication.
- Use curses for display
- Use threading to read from stdin and socket in parallel

Solutions

Lab 1: Simple scripts

Task 1: Celsius - Fahrenheit conversion

```python
from __future__ import print_function

try:
    intype = raw_input("Input type: ").upper().strip()
    if (not intype in 'CF'):
        print("Invalid type")
        exit(-1)
    degree = raw_input("Input temperature:").strip()
    ndegree = float(degree)
except ValueError:
    print("Wrong number:",degree)
    exit(-1)
except (KeyboardInterrupt,EOFError):
    print()
    exit(0)

if (intype == "C"):
    outtype = "F"
    outdegree=9.0 * ndegree / 5.0 + 32
else:
    outtype = "C"
    outdegree = 5.0 * (ndegree - 32) / 9.0


print("Output: %g %s" %(outdegree,outtype))
```

Task 2: Celsius - Fahrenheit conversion - infinite loop

```python
from __future__ import print_function

while True:
  try:
    intype=raw_input("Input type: ").strip().upper()
    if not intype in 'CF':
      print("Invalid type")
      continue
    degree=raw_input("Input temperature: ").strip()
    ndegree=float(degree)
  except ValueError:
    print("Wrong number",degree)
    continue

  except (EOFError,KeyboardInterrupt):
    print()
    break



  if (intype=="C"):
    outtype="F"
    outdegree=9.0*ndegree/5.0+32
  else:
    outtype="C"
    outdegree=5.0*(ndegree-32)/9.0

  print("Output: %g %s" %(outdegree,outtype))
```

Task 3: Name - email conversion

```
from __future__ import print_function
from  unicodedata import category,normalize
import sys
from StringIO import StringIO

def strip_accents(s):
    out=StringIO()
    for c in normalize('NFD',s):
        if (category(c) in ('Ll','Zs')):
            out.write(c)
    return(out.getvalue())

try:
  name=raw_input("Name: ").strip()
except (EOFError,KeyboardInterrupt):
  print();exit(0)


newname=strip_accents(name.decode(sys.stdin.encoding).lower())
names=newname.split()
email=u'.'.join(reversed(names))
email+="@foobar.com"
print(email.encode('ascii'))
```

A more elegant way to filter out accented characters is to use `filter`

```
#Using filter
def strip_accents(s):
  return filter(lambda x:category(x)!='Mn',normalize('NFD',s))
```

Lab 2: Scripts with sequence types

Task 1: Name - email conversion using input file

```
from __future__ import print_function
from unicodedata import category,normalize
import sys
import codecs


def strip_accents(s):
    return filter(lambda x:category(x)!='Mn',normalize('NFD',s))

def ask_for_name():
    try:
        name=raw_input("Name: ").strip()
    except (EOFError,KeyboardInterrupt):
        print();exit(0)
    return name.decode('utf8',"ignore")

def create_email(name,domain):
    newname=strip_accents(name).lower()
    names=newname.split(" ")
    email=u'.'.join(reversed(names))
    email+=domain
    return email.encode('ascii')

if (len(sys.argv)>=2):
    errors=0
    for file in sys.argv[1:]:
        try:
            fh=codecs.open(file,encoding='utf-8')
            for line in fh:
                name=line.strip()
                if (len(name)>0):
                    print(create_email(name,"@foobar.com"))
        except IOError:
            print("No such file:",file)
            exit(1)
        except UnicodeDecodeError:
            errors+=1
    if (errors>0):
        sys.stderr.write("File processed with %d errors!\n" %(errors))
else:
  name=ask_for_name()
  print(create_email(name,"foobar.com"))
```

Task 2: UID-GID match

```
import sys

file='/etc/passwd'
try:
  passwd=(line.strip().split(':') for line in open(file))
except Exception as e:
  print >>sys.stderr, ("Error opening file %s! (%s)" % (file,e.args[1]) )
  exit(-1)

for line in passwd:
  if int(line[2])>int(line[3]):
    print "%s (uid:%s gid:%s)" % (line[0],line[2],line[3])
```

Task 3: Word frequency count

```
import sys
from itertools import chain

if len(sys.argv)<2:
  print "Missing filename"
  exit(-1)

words={}
for line in open(sys.argv[1]):
  for word in line.strip().split():
    word=word.lower().decode('utf-8')
    if(words.has_key(word)):
      words[word]+=1
    else:
      words[word]=1

for word in words:
  print "%10s | %-20s | %d" % (word,"@"*words[word],words[word])
```

Nicer solution using collections.defaultdict. If using Python 2.7, you can also use collection.Counter. This solution also takes care of the max word length along with the maximum number of occurences.

```
import sys
from collections import defaultdict

if len(sys.argv)<2:
  print "Missing filename"
  exit(-1)

words=defaultdict(int)
for line in open(sys.argv[1]):
  for word in line.strip().split():
    words[word.lower().decode('utf-8')]+=1

maxlen=len( max( words, key=len))+1
maxval=max(words.values())
formatstr="%"+str(maxlen)+"s | %-20s | %d"



for word in words:
  dots=int(float(words[word])/maxval*20.0)
  print formatstr % (word,"@"*dots,words[word])
```

Task 4: Top-N word with command line option, and ordering

```
import sys
from getopt import getopt,GetoptError
import fileinput
from collections import defaultdict

try:
  opt,args=getopt(sys.argv[1:],"t:",['top='])
except GetoptError as error:
  print "Wrong option:",error
  exit(-1)

opts=dict(opt)

words=defaultdict(int)

for line in fileinput.input(args):
  for word in line.strip().split():
    words[word.lower()]+=1

top=int(opts['--top']) if opts.has_key('--top') else len(words)
topn=sorted(words,key=words.__getitem__,reverse=True)[:top]
for word in topn:
  print "%10s | %-20s | %d" % (word,"@"*words[word],words[word])
```

Task 5: Process output from outer programs

```
from subprocess import Popen,PIPE
import sys
from getopt import getopt,GetoptError

try:
    opt,args=getopt(sys.argv[1:],"",[])
except GetoptError as error:
    print "Wrong option:",error
    exit(-1)

tres=int(args[0]) if len(args)==1 else 0

header="filesystem size used avail use mounted_on".split()

p=Popen('df -hP'.split(),stdout=PIPE)
p.stdout.readline()                             #Drop header

#Create a list of dicts from header as keys and df output columns as values
df=(dict(zip(header,l.strip().split())) for l in p.stdout)
#Filter out lines where 'use' larger than threshold
used=( (d['filesystem'],d['mounted_on']) for d in df  if int(d['use'][:-1])>tres )

print "Overused filesystems"
for t in used:
  print "%50s (%s)" % t
```

Task 6: Copy files with extras

```python
import sys
from getopt import getopt,GetoptError
import glob
import shutil
import os.path
import subprocess

try:
    opt,args=getopt(sys.argv[1:],":t:a:",['to=','tar='])
except GetoptError as error:
    print "Wrong option:",error
    exit(-1)

opts=dict(opt)

if ( (len(args)<1) or (not (os.path.isdir(args[0]) ) ) ) :
    print "Missing directory or not a directory"
    exit(-1)


logfiles=(f for f in glob.iglob( os.path.join(args[0],"*.log") ) if os.path.isfile(f) )
if (opts.has_key('--to')):
    todir=opts['--to']
    if ( os.path.isdir(todir) ):
        for f in logfiles:
            shutil.copy2(f,todir)
            print "%s is copied" % (f)
    else:
        print 'Output directory %s does not exists' % (todir,)
        exit(-1)
elif (opts.has_key('--tar')):
    tarfile=opts['--tar']
    options=['/bin/tar','-cvf',tarfile]
    options.extend(logfiles)
    if (subprocess.call(options)!=0):
        print "Creating tar file %s was unsuccessful!" % (tarfile,)
        exit(-1)
else:
  print '\n'.join(logfiles)
```

Task 7: Top-N file in a directory

```python
import sys
from getopt import getopt,GetoptError
import os
from operator import itemgetter

try:
    opt,args=getopt(sys.argv[1:],"n:",[])
except GetoptError as error:
    print "Wrong option:",error
    exit(-1)

opts=dict(opt)
```

```
topn=int(opts['-n']) if opts.has_key('-n') else 5

if ( (len(args)<1) or (not os.path.isdir(args[0]) ) ):
    print "Missing directory"
    exit(-1)

dir=args[0]
oldcwd=os.getcwd()
os.chdir(dir)

ls=( (f,os.path.getsize(f))
     for f in os.listdir("./")
     if os.path.isfile(f)
   )
for tup in sorted( ls,key=itemgetter(1),reverse=True )[:topn] :
  print "%-50s %s" % tup

os.chdir(oldcwd)
```

Lab 3: Using regural expressions

Task 1: Temperature conversion with regexp

```
import re

temp=raw_input("Temperature?: ").strip()
sre=re.match(
r'''
^\s*
([+-]?
  \d*\.?\d+
)
\s*
([cf])
$''',
temp,re.I+re.X)

if (sre):
  degree=float(sre.group(1))
  intype=sre.group(2).upper()

  if (intype=="C"):
    outtype="F"
    outdegree=9.0*degree/5.0+32
  else:
    outtype="C"
    outdegree=5.0*(degree-32)/9.0
  print "%f %s" % (outdegree,outtype)
else:
    print "Invalid format"
    exit(-1)
```

Task 2: Backreferencing

```
from __future__ import print_function
import re
import sys

try:
    num=int(sys.argv[1])
except (ValueError,IndexError):
    print("An int is required as argument!",file=sys.stderr)
    exit(-1)

#reg_text=r"((\w)"+r"\2"*(num-1)+r").*\1$"   #like r'((\w)\2).*?\1$'
reg_text=r"(\w)\1{%d}.*\1{%d}$" % (num-1,num)   #like r'(\w)\1{2}.*\1{3}$'
reg=re.compile(reg_text,re.I+re.M)

for line in open("/usr/share/dict/words"):
    if reg.search(line):
        print(line,end="")
```

Task 3: Substitution

```
from __future__ import print_function

import re
import sys
import io
from getopt import getopt,GetoptError

def usage():
    print("Usage: %s -l start,end filename" % sys.argv[0])
    exit(-1)


try:
    opt,args=getopt(sys.argv[1:],"l:",[])
    filename=args[0]
    opts=dict(opt)
    file=io.open(filename,'r+')
    if (opts.has_key('-l')):
        start,end=(int(x) for x in opts['-l'].split(','))
except Exception:
    usage()

reg=re.compile(r'^(\S+)(.*?)(\S+)$')


for i in xrange(start-1):
    line=file.readline()

for i in xrange(start,end+1):
    line=file.readline()
    new_line=reg.sub(r'\3\2\1',line)
    file.seek(file.tell()-len(line))
    file.write(new_line)
```

Task 4: Process multiline expression

```
from __future__ import print_function

import re
import sys

try:
    db_str=open(sys.argv[1]).read()
except Exception:
    print("Usage: %s cluster_toc_file" % (sys.argv[0],))
    exit(-1)

re_grp=re.compile(r'(?:META)?CLUSTER=(\w+)(.+?)\bEND\b',re.DOTALL)
re_member=re.compile(r'SUNW_CSRMEMBER=(\w+)',re.DOTALL)

mydb=dict()

def rprint(name):
    for m in mydb[name]:
        member=m.group(1)
        if mydb.has_key(member):
            rprint(member)
        else:
            print(member,end=',')


for i in re_grp.finditer(db_str):
    groupname=i.group(1)
    mydb[groupname]=re_member.finditer(i.group(2))

try:
    group=raw_input("Enter a group name: ")
    print("%s:\t" % (group),end="")
    rprint(group)
except (EOFError, KeyboardInterrupt):
        exit(0)
except KeyError:
    print("No such group")
finally:
    print("")
```

Lab 4: OOP in Python

Task 1: Simple object

Please create a object named **Person**, with the following attributes:

Task 2: Overload, static variables, builtins

Extend you previous object to implement the following

1. Rewrite constructor

```
import weakref

class Person(object):

  __slots__=('__weakref__','_name','_gender','age')
  persons=[]

  def __init__(self,name,gender,age=0):
    self._name=name
    self.gender=gender
    self.age=age
    Person.persons.append(weakref.ref(self))
```

2. It should implement a destructor.

```
  def __del__(self):
    '''Simply remove None elements'''
    for r in Person.persons:
      if (r() is None):
        Person.persons.remove(r)
    print("Destructor is called for %s" % self)
```

And alternate (and also nicer) solution of the above two, it to specify a callback, when the weakref is created

1. Rewrite constructor

```
  def __init__(self,name,gender,age=0):
    self._name=name
    self.gender=gender
    self.age=age
    Person.persons.append(weakref.ref(self,Person.__on_delete))
```

2. It should implement a destructor.

```
  def __del__(self):
    '''Do nothing'''
    print("Destructor is called for %s" % self)

  @staticmethod
  def __on_delete(weakrf):
    '''This one is called by weakref when the referred object is about to finalize'''
    if (Person): Person.persons.remove(weakrf)
```

3. Use the property decorator to intercept getter/setter calls for property **name**

```
@property
def name(self):
  return self._name

@name.setter
def name(self,name):
  if (str.upper(name) == str.upper(self._name)):
    self._name=name
  else:
    raise AttributeError("Changing name is not allowed!")
```

4. Overload methods

```
def __str__(self):
  attrs=[ str(self.__getattribute__(attr)) for attr in ('name','gender','age') ]
  return ('/'.join(attrs))
def __cmp__(self,other):
  return cmp(self.name,other.name)

def __add__(self,other):
  self.age+=other
  return self
```

5. Class Person has a printAll method, that

```
@staticmethod
def printAll():
  for p in Person.persons:
    print p()
```

Task 3: Gender as an internal class

```
class Gender(object):

  def __get__(self,instance,owner):
    return instance._gender
  def __set__(self,instance,value):
    if (value.upper() in ('MALE','FEMALE')):
      instance._gender=value.upper()
    else:
      raise AttributeError("Invalid value from gender")

gender=Gender()
```

Task 4: Make class Person iterable

```
class Person(Sequence):
```

```
__len__=persons.__len__
def __getitem__(self,index):
  p=Person.persons[index]
  if (p()): return p().name
  return None
```

Task 5: Automatic variables

```
def __getattr__(self,what):
  return 'N/A'
```

Optional Task 6: Arbitrary function

```
__actions__=('run','smile','snor')

def __getattr__(self,what):
  def _do_print(*args):
    print "Hi, I am",self.name,"and",what+"ing!"
  if (what in Person.__actions__):
    return _do_print
  else:
    raise AttributeError(what+" is not implemented")
```