CS 35L Software Construction Lab
Week 3 - Python

What is Python?

- Object-Oriented language
  - Classes
  - Member functions
- Interpreted language
  - Python code is compiled to bytecode
  - Bytecode interpreted by an interpreter
- Interactive

Python List

- Common data structure in Python
- A python list is like a C array but much more:
  - **Dynamic (mutable)**: expands as new items are added
  - **Heterogeneous:** can hold objects of different types
- How to access elements?
  - List_name[index]

Example

- >>> t = [123, 3.0, 'hello!']
- >>> print t[0]
  - 123
- >>> print t[1]
  - 3.0
- >>> print t[2]
  - hello!

Example – Merging Lists

- >>> list1 = [1, 2, 3, 4]
- >>> list2 = [5, 6, 7, 8]
- >>> merged_list = list1 + list2
- >>> print merged_list
  - **Output: [1, 2, 3, 4, 5, 6, 7, 8]**

String Slices

The "slice" syntax is a handy way to refer to sub-parts of sequences -- typically strings and lists. The slice s[start:end] is the elements beginning at start and extending up to but not including end. Suppose we have s = "Hello"

```
Hello
 0  1  2  3  4
-5 -4 -3 -2 -1
```

- s[1:4] is 'ell' -- chars starting at index 1 and extending up to but not including index 4
- s[1:] is 'ello' -- omitting either index defaults to the start or end of the string
- s[:] is 'Hello' -- omitting both always gives us a copy of the whole thing (this is the pythonic way to copy a sequence like a string or list)
- s[1:100] is 'ello' -- an index that is too big is truncated down to the string length

The standard zero-based index numbers give easy access to chars near the start of the string. As an alternative, Python uses negative numbers to give easy access to the chars at the end of the string: s[-1] is the last char 'o', s[-2] is 'l' the next-to-last char, and so on. Negative index numbers count back from the end of the string:

- s[-1] is 'o' -- last char (1st from the end)
- s[-4] is 'e' -- 4th from the end
- s[:-3] is 'He' -- going up to but not including the last 3 chars.
- s[-3:] is 'llo' -- starting with the 3rd char from the end and extending to the end of the string.

Python Dictionary

- Essentially a hash table
  - Provides key-value (pair) storage capability
- Instantiation:
  - dict = {}
  - This creates an EMPTY dictionary
- Keys are unique, values are not!
  - Keys must be immutable (strings, numbers, tuples)

Example

- dict = {}
- dict[‘hello’] = “world”
- print dict[‘hello’]
  - world

- dict[‘power’] = 9001
- if (dict[‘power’] > 9000):
- print “It is over “, dict[‘power’]
  - It is over 9001

- del dict[‘hello’]
- del dict

for loops

list = [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]

```
for i in list:
    print i
```

Result:
Mary
had
a
little
lamb

```
for i in range(len(list)):
    print i
```

Result:
0
1
2
3
4

Indentation

- Python has no braces or keywords for code blocks
  - C delimiter: {}
  - bash delimiter:
    - then…else…fi (if statements)
    - do…done (while, for loops)
- Indentation makes all the difference
  - Tabs change code’s meaning!!

Optparse Library

- Powerful library for parsing command-line options
  - **Argument:**
    - String entered on the command line and passed in to the script
    - Elements of sys.argv[1:] (sys.argv[0] is the name of the program being executed)
  - **Option:**
    - An argument that supplies extra information to customize the execution of a program
  - **Option Argument:**
    - An argument that follows an option and is closely associated with it. It is consumed from the argument list when the option is

Homework 3

- randline.py script
  - Input: a file and a number *n*
  - Output: *n* random lines from *file*
  - Get familiar with language + understand what code does
  - Answer some questions about script
- Implement comm utility in python

Running randline.py

- Run it
  - ./randline.py –n 3 filename (need execute permission)
  - python randline.py –n 3 filename (no execute permission)
- randline.py has 3 command-line arguments:
  - n: specifies the number of lines to write
    - **option**
  - 3: number of lines
    - **option argument** to n
  - filename: file to choose lines from
    - **argument** to script
- Output: 3 random lines from the input file
- Python 3 is installed in /usr/local/cs/bin
  - export PATH=/usr/local/cs/bin:$PATH

Python Walk-Through

```
#!/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

Python Walk-Through

```
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

Comm.py

- Support all options for comm
  - -1, -2, -3 and combinations
  - Extra option –u for comparing unsorted files
- Support all type of arguments
  - File names and – for stdin
- If you are unsure of how something should be output, run a test using existing comm utility!
  - Create your own test inputs
- Comm C source code :
  - comm C source code
  - This will give you an idea of the logic behind the operation that comm executes
- Python OptionParser link :
  - Python OptionParser
  - How to add your own options to the parser