CS 35L

Software Construction Laboratory

Lecture 3.2

17th October, 2019

Logistics

- Assignment 3
  - Due on October 21st
- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
- Assignment 10 Signup Sheet
  - Teams of 2
  - https://docs.google.com/spreadsheets/d/1PVqVMEEsHjmmj9YLyqz5K4wU-k0Dwwm2iO9uS1Zq1wk/edit?usp=sharing
  - Names must be filled by Friday of Week 3
    - Topic can be selected later
    - Ensure that topics are not the same
- Office Hours this week:
  - Friday - 9:30 – 11:30 am – BH3256S

Review - Previous Lab

- Modifying large-scale software
- Decompressing files
- Compilation process
- Build Process

Python

What is Python?

- Not just a scripting language
- Object-Oriented language
  - Classes
  - Member functions
- Compiled and interpreted
  - Python code is compiled to bytecode
  - Bytecode interpreted by Python interpreter
- Not as fast as C but easy to learn, read and use. Why?
- Very popular at Google and other big companies

Why is it Popular?

- Uses English keywords frequently where other use different punctuation symbols
- Fewer Syntactical Constructions
- Automatic Garbage Collection
- Easy integration with other programming languages

Different Modes

- Interactive:
  - Run commands on the python shell without actually writing a script/program.
- Script Mode:
  - Type a set of commands into a script
  - Execute all the commands at once by running the script

Python Variables

- Case sensitive
- Can start with _ (underscore) or letters followed by other letters, underscores or digits
- Other special characters are not allowed as part of the variable name
- Certain reserved words may not be used as variable names on their own unless concatenated with other words

Example – Python Variables

```
#!/usr/bin/python3
counter = 100     # An integer assignment
miles = 1000.0    # A floating point
name = "John"    # A string
print(counter)
print(miles)
print(name)
```

- Output:
  - 100
  - 1000.0
  - John

Python Lines and Indentation

- No braces to indicate blocks of code for class and function definitions or flow control
- Blocks of code are denoted by line indentation, which is why it is strictly enforced
- Number of spaces for indentation may be variable but all the statements within the same block must be equally indented
- Hence, a single space has the ability to change the meaning of the code

Python Decision Making

```
#!/usr/bin/python3
var = 100
if var == 100:
    print(”Correct”)
print("Good bye!”)
```

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]

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 – Python Dictionary

```
dict = {}
dict[‘france’] = “paris”
dict[‘japan’] = “tokyo”
print(dict[‘france’])

dict[‘germany’] = “berlin”
if (dict[‘france’] == ”paris”):
    print(“Correct!”)
else:
    print(“Wrong!”)

del dict[‘france’]
del dict
```

For loops

list1 = [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]

```
for i in list1:
    print(i)
```

Result:

Mary
had
a
little
lamb

```
for i in range(len(list1)):
    print(i)
```

Result:

0
1
2
3
4

I/O Basics

- The input([prompt]) function reads one line from standard input and returns it as a string
  - str = input("Enter your input: ")
  - print("Received input is : ", str)

Functions

- A function is a block of organized, reusable code that is used to perform a single, related action. They provide better modularity for your application and a high degree of code reusing.

  - Syntax:
  - def function_name(parameters):
  - #code inside the function

Functions - examples

- Example 1:
  - def printme(new_string): #string is a parameter
  - #This prints a passed string into this function
  - print(new_string)
  - return
- Example 2: To print sum of numbers in a list
  - def find_sum(new_list):
  - sum=0 #initialize variable*
  - for element in new_list:
  - sum = sum + element
  - return sum #returns the computed sum
- answer_variable=find_sum([2,3,4,5]) #function call
- Print(answer_variable)
- * # are used for putting comments

Task 1

- Take a list a = [1,1,2,3,5,8,13,21,34,55,89] and write a program that prints out all the elements of the list that are less than 5
- Instead of printing the elements one by one, make a new list that has all the elements less than 5 from this list in it and print out this new list.
- Ask the user for a number and return a list that contains only elements from the original list a that are smaller that that number given by the user

Task 2

- Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string.
  - Sample String : 'w3resource'
  - Expected Result : 'w3ce'
  - Sample String : 'w3'
  - Expected Result : 'w3w3'

Task 3

- Create a python dictionary with the following keys and values:
- “Names” : [“Mickey”, “Minnie”]
- “Mickey” : [“UCLA”, “Bachelor Degree”]
- “Minnie” : [“UCB”, “Bachelor Degree”]
- The values in the dictionary are in the form of a list.
  - Now traverse the list whose key is ‘Names’ and for every element in this list, find the corresponding key (eg. ‘Mickey’). Append the word “Computer Science” to the value (eg. the list of ‘Mickey’) of that particular key.
  - Now create a new key-value pair for “DonaldDuck” - [“Stanford”, “PhD”, “Computer Science”]. Add the name ‘DonaldDuck’ to the ‘Names’ list as well.

Assignment 3 - Homework

- 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 (Q3, Q4)
- Implement shuf utility in python3

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
  - Refer to these links:
    - https://docs.python.org/3/howto/argparse.html
    - https://docs.python.org/3/library/optparse.html

Running randline.py

- Run it
  - ./randline.py –n 3 filename (need execute permission)
  - python3 randline.py –n 3 filename (no execute permission)
- randline.py has 3 command-line arguments:
  - filename: file to choose lines from
    - argument to script
  - n: specifies the number of lines to write
    - option
  - 3: number of lines
    - option argument to n
- Output: 3 random lines from the input file

Shuf.py

- Support the options for shuf
  - --echo (-e)
  - --head-count (-n)
  - --repeat (-r)
  - --help
- Support all type of arguments
  - File names and – for stdin
  - Any number of non-option arguments
- Error handling

Assignment 3 - Homework

- shuf.py – this should end up working almost exactly like the utility ‘shuf’
  - Check $ man shuf for extensive documentation
- Use randline.py as a starting point!
  - Modify to accomplish logical task of shuf
- shuf C source code :
  - Present in coreutils
  - This will give you an idea of the logic behind the operation that shuf executes
- Python argparse module instead of optparse:
  - How to add your own options to the parser
  - -e -n --repeat --echo etc

Assignment 3 - Homework Hints

- If you are unsure of how something should be output, run a test using existing shuf utility!
  - Create your own test inputs
- The shuf option --repeat is Boolean
  - Which action should you use?
- Q4: Python 3 vs. Python 2
  - Look up “automatic tuple unpacking”
- 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

Create a file handle

Read 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 return the line corresponding to the randomly selected number

The beginning of main function

version message

usage message

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

Questions?