CS 35L: Python Basics

Lab 6

Hengda Shi

Week 3 Lecture 1

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

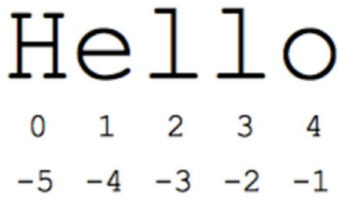
```
>>> t = [123, 3.0, ‘hello!’]

>>> 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)

[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"



- 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)
  - Why? (Hint: related to how hash table works)

Example

```
>>> dict = {}
>>> dict[‘hello’] = “world”
>>> print dict[‘hello’]
world
>>> dict[‘power’] = 9001
>>> if dict[‘power’] > 9000:
…      print(F"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:

May
had
a
little
lamb

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

Result:

0
1
2
3
4

enumerate

```
>>> list = ['Mary', 'had', 'a', 'little', 'lamb']

>>> for i, val in enumerate(list):

…       print("i is {}, val is {}".format(i, val))

i is 0, val is Mary
i is 1, val is had
i is 2, val is a
i is 3, val is little
i is 4, val is lamb
```

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
  - WHITESPACES are important!!!
  - Tabs change code’s meaning!!
  - Python 3 disallows mixing the use of tabs and spaces for indentation.

I/O Basics

- The *input([prompt])* function reads a line from input, converts it to a string (stripping a trailing newline), and returns that.

```
>>> str = input("Enter your input: ")
>>> print("Received input is: ", str)
```

Context Managers

```
with open('filename', 'r') as file:
    for line in file:
        tokens = line.split()
        ...
```

is preferred over

```
f = open('filename', 'r')
f.close()
```

Context managers automatically manages resources such as files and network connections. e.g. a file will be closed if an exception happens in the inner scope

Python Variables

- Case sensitive
- 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

Python Script:

```
#!/usr/bin/python
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
```

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): # new_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
```

Note: # are used for putting comments

Argument Passing

```
def test():
    li = [1, 2, 3]
    f1(li)
    print(li) # prints [1, 2, 3, 10]
    f2(li)
    print(li) # still prints [1, 2, 3, 10]
def f1(li):
    li.append(10)
def f2(li):
    li = [1]

test()
```

Pass by reference

In Python, objects are passed by a pointer value.

When the callee accesses the object's method, the original object will be affected. Think of the pointer->method syntax in C/C++.

However, reassigning a variable obj to something else merely changes the memory address obj points to, so it does not affect the original variable.

Python split

Python split using delimiter:

```
>>> x = "blue, red, green"
>>> x.split(",") # "," is a delimiter here
['blue', 'red', 'green']
```

```
>>> a, b, c = x.split(",")
>>> a
'blue'
>>> b
'red'
>>> c
'green'
```

Task 1

- Take a list a = [1, 1, 3, 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 than that number given by the user

Task 2

Write a program that accepts sequence of newlines from the user as input till the user gets bored.
(The user gets bored when he/she presses Enter twice)
Hint: to detect if Enter is pressed twice (in other terms - the user input is void)

Suppose your input string name is ‘s’ just type:
if s:
    #write code to capitalize (s.upper()) this input string ‘s’ and append it to a ‘new_list’

Now, print all the elements of this ‘new_list’

Suppose the following input is supplied to the program:

Hello world
Practice makes perfect

Then, the output should be:

HELLO WORLD
PRACTICE MAKES PERFECT

Task 3

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 4

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 dictionary with the key as ‘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

Task 5

1. Sort the list and store it into variable list2

```
list1 = [12, 232, 2, 4, 4, 45, 4, 5, 1, 100] # whatever elements you want to add
```

2. Sort list1 in descending order and store it into variable list3

3. Sort the following list based on second element and store it into variable list5

```
list4 = [('berkeley', 2), ('ucla', 1), ('usc', 3)]
```

Task 6

Write a factorial function which takes a number ‘n’ as input and prints/returns its factorial (don’t forget to call the function)

fact(5) = 120
Base case:
fact(1) = 1
fact(0) = 1

```
def fact(n):
    #.....
```

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 shuf utility in python
- Check python documentation when in doubt: https://docs.python.org/3/index.html

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
```

shuf.py

- Support all options for shuf
  - --input-range (-i), --head-count (-n), --repeat (-r), and --help
- 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 shuf utility!
  - Create your own test inputs
- Comm C source code :
  - shuf C source code
  - This will give you an idea of the logic behind the operation that comm executes
- Python argParse link:
  - OptParse is deprecated, use argParse module instead
  - Python argParse
  - How to add your own options to the parser