CS35L Software Construction Laboratory

Lab 5: Sneha Shankar

Week 3; Lecture 2

What is Python?

- Not just a scripting language
- Object-Oriented language
  - Classes
  - Member functions
- Interpreted language
  - Python code is compiled to bytecode
  - Bytecode interpreted by an interpreter
- Not as fast as C but easy to learn, read and use
- 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
- 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
```

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

Indentation contd…

- 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!!

Python Decision Making

```
#!/usr/bin/python
var = 100
if ( var == 100 ) :
        print "Value of expression is 100”
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

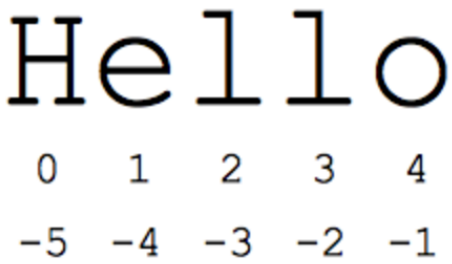
- >>> t = [123, 3.0, ‘hello!’]
- >>> print t[0]
  - 123
- >>> print t[1]
  - 3.0
- >>> print t[2]
  - hello!

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]**

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 Split

Python split using delimiter:

```
>>> x = “blue, red, green”

>>> x.split(“,”)  #”,” is a delimiter here

Output: [‘blue’, ‘red’, ‘green’]

>>> a, b, c = x.split(“,”)

>>> a

‘blue’

>>> b

‘red’

>>> c

‘green’
```

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

I/O Basics

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

- The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.
  - *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 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

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 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.