Recitation 13 - Python (Loops, Conditions, Functions)

CSCI1300

Instructor: Dr.David Knox

Python

From this point on in the semester we will be working with Python. A great resource for learning Python is, **"Learning Python the hardway":** https://learnpythonthehardway.org/book You will have to save your files as .py instead of .cpp when working with python.

You can use Geany as your IDE for writing python programs or just use a text editor and a terminal window to compile your program.

Note: During the recitation your TA will be demonstrating the example code shown below in Geany and a Text Editor.

Indentation:

In Python the delimiter is a indentation of the code itself. No curly braces will be used to mark where the code starts and stops and for any loops, if-else statements, or functions there is not an explicit begin or end without the indentation. There are no explicit braces, brackets, or keywords. This means that whitespace is significant, and must be consistent.

"Code blocks" or sections of code are defined by their indentation. Code blocks means sections of code for functions, if statements, for loops, while loops, and so forth. *Indenting* starts a block and is ended when the indentation moves back a tab.

For example:

```
Block 1  #statements in Block 1 is Executed
        Block 2  #statements in Block 2 is Executed
                Block 3  #statements in Block 3 is Executed
        Block 2  #Remaining statements in Block 2 is Executed
Block 1 #Remaining statements in Block 1 is Executed
```

Functions in Python

A function has a few key components: the name, the parameters, the body and the return value. Let's take a look at a simple example.

```
    Name Parameters
        ↓      ↓
def adder(x, y):
        z = x+y       ← Body
        return z
```

↑
Return

Function Example:
This example defines a function called *adderTwo* that takes two parameters and returns their sum. The *adderTwo* function is called from another function called *main*.

```
def adderTwo(numOne, numTwo):
        print(numOne, numTwo)
        numOne = numOne + numTwo
        print(numOne)
        return numOne

def main():
        X = 3
        Y = 4
        print(X,Y)
        Z = adderTwo(X,Y)
        print(X, Y, Z)
```

Loops in Python: For and While

Loops allow us to run a chunk of code multiple times. If we know how many times we'll need to run the code, we typically use a for loop. When we don't know how many times we need to run the code, we usually use a while loop.

For Loops

The basic structure of a for loop in Python looks like this:

```
for i in range(startNum, endNum):
    # do something
```

Here, *i* is the dummy variable and *range(startNum, endNum)* is the iterable. The dummy variable takes on the values of the iterable, beginning with *startNum* and up to, but not including, *endNum*. Each loop, *i* will increment by 1. For example, if we enter the following code:

```
for i in range(0, 5):
    print(i)
```

We should see the following output:

```
0
1
2
3
4
```

Alternatively, we can define range with a single parameter: *range(endNum)*
By default, the dummy variable will begin at 0 and iterate up to, but not including, *endNum*.

If we want to have *i* increase by some number other than 1, we can specify this number as the 3rd parameter in *range*:

```python
for i in range(0, 5, 2):
    print(i)
```

The output:

```
0
2
4
```

Of course we can do things far more complex than just printing a single variable. Try writing a loop that outputs the even numbers between 0 and 10.

While loops

A while loop in python uses the following syntax:

```python
while (condition):
    # do something
```

As long as *condition* is true, the code within the while loop will run. Consider the following code:

```python
userChoice = 1
while (userChoice):
    userChoice = int(input("Do you want to see this question again? "
        "Press 0 for no, any other number for yes. "))
```

Entering ‘0’ will terminate the loop, but any other number will cause the loop to run again.

Note how we have to initialize the condition before the loop starts. Setting *userChoice* equal to 1 ensures that the while loop will run at least once.

Also note that we must “cast” the input to type int. By default, input() returns a string value. A string value will always evaluate as true. Consequently, we must turn it into an integer to have its “truth” evaluated within the while condition.

Converting For into While

It's possible to use a while loop instead of a for loop. Let's rewrite the second for loop example from above as a while loop:

```
i = 0
while i < 5:
    print(i)
    i = i + 2
```

This accomplishes the same as output as above, but with more lines of code. Notice how you must manually initialize *i* to equal 0 and then manually increment i by 2.

Debugging Tip: Inserting print statements into your loops is a quick way to debug your code if something isn't working.

Lists:

A list in python is an important data type. To store elements as a list, place all the elements enclosed within square brackets and separated by comma. Unlike in c++, we can have different types of elements in the same python list. Below are some of the examples of lists

```
myList = []   # empty list
myList = [1, 2, 3] # list of integers
myList = [1, "Hello", 3.4] # list with mixed datatypes
myList = ["csci1300", [2, 4, 6]] # nested list
```

Access elements in a list

To access the elements in a list we need to use the index operator [ ]. Similar to arrays in c++, python lists index starts from 0. For example, a list having 10 elements will have index from 0 to 9.

Example:

```
myList = [1, "Hello", 3.4]

print(myList[0])  #Output is 1
print(myList[2]) #Output is 3.4


myList_1 = ["csci1300", [2, 4, 6]]

print(myList_1[0][1])    #Output is s
print(myList_1[1][2])       #Output is 6
```

Slice Lists in Python:

```
myList = ['p','y','t','h','o','n']
# elements from 2nd to 5th
print(myList[1:4])   #Output is ['y', 't', 'h']

# elements from 3rd to end
print(myList[2:])  #Output is  ['t', 'h', 'o', 'n']
```

Add/Change Elements in a list:

```
myEven = [2, 5, 6, 8]

# change the 2nd item
myEven[1]  = 4
print(myEven) #Output is [2, 4, 6, 8]

# change 2nd to 4th items
myEven[1:4] = [10, 12, 14]
print(myEven)  #Output is [2, 10, 12, 14]
```

For examples of extra commands used with lists see the following:

https://docs.python.org/3/tutorial/datastructures.html

File I/O in Python:

The file I/O operations are similar to C++. The steps followed to open and close a file are as shown below

1) Open a file in the correct mode and store it in a file object.
2) Use this file object to modify or read from the file as required.
3) Once done with all the operations in a file , please make sure to close the file.

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Table Reference: https://www.tutorialspoint.com/python/python_files_io.htm

To Write lines to a files:

```
fruits = ['Apple\n', 'Pineapple\n']
fw = open('fruits_file.txt', 'w')
   fw.writelines(fruits)
   fw.close()
```

To Read lines from a file:

```
fr = open('fruits_file.txt')
fr.readline()
#Output is 'Apple\n'
fr.readline()
#Output is 'Pineapple\n'
fr.close()
```

To Read lines in a loop:

```
fr = open('fruits_file.txt')
for line in fr:
     print(line)

#Output is:
#     Apple
#     Pineapple

fr.close()
```

Recitation Activity (Due by Sunday 5PM):

1. Write a function ReadEvens that takes a filename, reads all the values (one per line, integers) from a file and store the even numbers in a list. The function should return the list.

2. Write a function ReadDivBy that takes a filename and a multiplier, reads all the values (one per line, integers) from file and store stores all the numbers that are divisible by that multiplier in a list. Return the list.

Name your file Lastname_Firstname.py and submit it to the moodle.