CSC/CPE 101: Fundamental of Computer Science I

Winter 2019 (LAB-4)

Lab 4, CSC 101

This lab provides additional exercises on conditional reasoning and an introduction to loops.

Download Lab4_Given_Files.zip, from PolyLearn (place it in your `CPE101` directory, and unzip the file.

Conditionals

This part of the lab is structured a bit differently from what you have been asked to do in the past. The given files, in the `patterns` directory, include a "driver" and multiple pattern files. You will not (and should not) need to modify these files. The driver provides a mostly complete program but it needs a function to work properly. This is the function that you are to write. The pattern files provide text-based "images".

letter

Each pattern file consists of an "image" made up of letters. For a given pattern, you must write a function that returns the correct letter for the row/column position specified (as the function's arguments). This function will be called multiple times; once for each position in the pattern image.

As an example, pattern00 is made up entirely of repeated 'A' characters. As such, the `letter` function could be written as follows.

```
def letter(row, col):
   return 'A'
```

In general, however, the `letter` function must determine which letter to return based on the row and column provided. In the pattern files, rows and columns are counted beginning at 0. As such, the top-left character is at `row == 0` and `col == 0`. So if the pattern consisted of all 'A' characters but with a single 'G' character in the top-left corner, then the `letter` function might look as follows.

```
def letter(row, col):
   if (row == 0 and col == 0):
      return 'G'
   else:
      return 'A'
```

Multiple Files and All That

You will write multiple versions of this `letter` function. Since the `driver.py` file will remain unchanged, you must place your `letter` function in a separate file. The name for this file is given below (for convenience, it is the name of the pattern).

Of course, for a given execution, the driver must be "told" which specific `letter` function to use. You will do this by importing the `driver` in your file and calling the `compare Patterns` function in the driver. To this function, you must pass your `letter` function (by specifying the name as you would a variable). The following code completes the "all A" pattern example discussed above.

```
import driver

def letter(row, col):
   return 'A'

if __name__ == '__main__':
   driver.comparePatterns(letter)
```

Patterns - What you need to do.

pattern00

The `pattern00.py` is complete. Just run the pattern to see the message.

```
python3 pattern00.py < pattern00
```

pattern01

In `pattern01.py`, write the `letter` function that will generate the pattern image in `pattern01`.

pattern02

In `pattern02.py`, write the `letter` function that will generate the pattern image in `pattern02`.

pattern03

In `pattern03.py`, write the `letter` function that will generate the pattern image in `pattern03`.

pattern04

In `pattern04.py`, write the `letter` function that will generate the pattern image in `pattern04`.

pattern05

In `pattern05.py`, write the `letter` function that will generate the pattern image in `pattern05`.

pattern06

In `pattern06.py`, write the `letter` function that will generate the pattern image in `pattern06`.

pattern07

In `pattern07.py`, write the `letter` function that will generate the pattern image in `pattern07`.

The file runAllPaterns runs all patterns at once. This file is executable. Make sure it has an executable access specifier.

```
chmod +x runAllPaterns
./ runAllPaterns
```

Loops

This part of the lab introduces you to loops in Python. You are asked to edit a small program that uses various loops to accomplish a task.

Code Reading

In the `loops` directory you will find a file called cubesTable.py. Study the source code to see how it works.

Predict the output from the program if the user enters: `9 3 0`
Run the code to see if your prediction is correct.

Then predict the output from the program if the user enters: `-5 5 3 0`
Run the code to see if your prediction is correct.

Finally predict the output from the program if the user enters: `-5 5 -9 3 0`
Run the code to see if your prediction is correct.

Enhancement #1

Enhance the `get_first()` function by adding a input validation loop similar to the one in the `get_table_size()` function. The loop should reject negative user inputs. Follow the format of the example output below.

```
Enter the value of the first number in the table: -3
First number must be non-negative.
Enter the value of the first number in the table: 2
```

Enhancement #2

Complete the `show_table()` function by implementing the code for a for loop that will display the desired table of cubes. For example, if the user enters 4 5 as the inputs, the table will look like this:

```
Number  Cube
5       125
6       216
7       343
8       512
```

Enhancement #3

Add a new function `get_increment()` that is similar to `get_first()`. It should display the prompt `"Enter the increment between rows: "`. The function should use an input validation loop to reject negative user inputs using the message `"Increment must be non-negative."`.
Modify the `show_table()` function so that it takes a third parameter that is the row increment in the table. Modify the counting loop so that it uses the row increment when displaying the table.
Modify the `main()` function so that it calls `get_increment()` and passes the result to `show_table()`. For inputs of `4 5 3` the table should now look like this:

```
Number  Cube
5       125
8       512
11      1331
14      2744
```

Hint: To get the numbers to line up nicely (as shown above), `print` the first number left justified in a column of six spaces. You can do that using the `%-6d` modifier, like this or use .format:

```
print ("%-6d  %d" % (first, first**3))
```

Enhancement #4 - Last one!

Enhance the `show_table()` function so that it displays the sum of all the cubes in the table on a separate line below the table, in the format `"The sum of cubes is: x"` (where x is the sum of cubes).

Testing

Run the `cubesTableInst` solution version and compare it against yours to make sure they work the same. You do not need to be make input files and compare using `diff`. Just make sure the functionality is the same.

Demonstration

Demonstrate running the code from each part of the lab to your instructor to have this lab recorded as completed. In addition, be prepared to show the source code to your instructor.

For first part run all patterns by using given file:

```
./ runAllPaterns
```

Submission

Demo and submit all your .py files in PolyLearn (7 patterns and cubesTable.py)