**CSE 101**

**Lab 5**

**NAME: Bhavay Aggarwal**

**ROLL NO.: 2018384**

**SECTION: B**

**GROUP: 1**

**Getting credit:**

Complete all required blank boxes and lines on this handout.

**Getting set up:**

From the Lab webpage, download and unzip Lab 4.zip into a folder (say) Lab 4. It will include DemoGraphics.py, IndianFlag.py, DemoMath.py and SimpleGraphics.py. In the command shell, navigate the file system so that this folder is THE CURRENT WORKING DIRECTORY.

**Submission:**

Upload the following on the Backpack deadline, zipped into a folder named Lab4\_2018xxx.zip . **Make sure you mention your name, roll number, section and group in each of these files(as comments in your codes)**

1. This document, with answers to questions in the first section
2. Edited version of DemoGraphics.py, as asked in second section
3. Edited version of IndianFlag.py, as asked in second section
4. **DemoMath**

This part of lab is about the module DemoMath.py: Open the DemoMath.py in an editor and answer the questions below

**Q1:** What is displayed if you ask for the square root of 25? Is this program’s sqrt(25) correct according to math.sqrt(25)? What is displayed if you ask for the square root of 2500? Is this program’s sqrt(2500) correct according to math.sqrt(2500)?

**Ans. Output - x = 25.00**

**math.sqrt(x) = 5.000000000000**

**sqrt(x) = 5.000023178254 #incorrect because of incorrect approximation**

**True 4th Root = 2.236067977500**

**fourth\_root(x) = 25.000000000000 #incorrect**

**Enter a number whose square root you want: 2500**

**x = 2500.00**

**math.sqrt(x) = 50.000000000000**

**sqrt(x) = 88.502206389628 #incorrect because of incorrect approximation and the difference in answer is larger because our approximation fails for larger numbers**

**True 4th Root = 7.071067811865**

**fourth\_root(x) = 2500.000000000000 #incorrect**

**Q2: a)** We used “import math” instead of “from math import \*” or “from math import sqrt”. This question justifies our choice, and explores the implications of the various ways to use import statements.

What happens if you say y2 = DemoMath.sqrt(x) instead of y2 = sqrt(x) in the Application Script? Explain in a way that demonstrates that you understand “what Python thinks you are referring to” (and why it is unsuccessful in finding it) when encountering the word “DemoMath”.

**Ans. Output - NameError: name 'DemoMath' is not defined**

**Python thinks we are calling math function present inside DemoMath module which we might have imported but as it runs it shows an error because no module with the name of DemoMath has been imported.**

**Q2: b)** From the above, you should explain why would we get a “name ’math’ is not defined” error for the line y1 = math.sqrt(x) if we had used either of the two alternate import statements mentioned above?

**Ans. We get such an error because instead of importing the whole math module we only imported sqrt() function and which is why math name remained undefined in the program hence raised an error.**

**Q3:** What happens if you comment out the return statement in sqrt? Explain — specifically, write down (and remember for future programming experience) the error message you (should) get, and explain why variable y2 has the value None instead of some float value, and so the print statement that is expecting a float causes an error.

**Ans. If we comment out the return statement the function returns no output while in the print statement we have specified that we will be printing a float object(print (' sqrt(x) = %15.12f' %y2)) and so it raises an error**

**Q4:** Why is it necessary to have the statement length = float(x) in the function body of sqrt?

**Ans. Length variable allows us to store the value of the integer which we are required to square root and without it we would have to use global variable which would also change the value of the number which we input making it not usable in the other functions. So it is better to store it in a variable like Length and then return it.**

**Q5:** How could sqrt be modified so that it could handle the input 0.

def sqrt(x):

"""Returns an approximate square root of x as float.

Performs five steps of rectangle averaging.

Precondition: The value of x is a positive number."""

# As explained in lecture, imagine you have an x-by-1 rectangle,

# which will thus have area x.

length = float(x)

if length != 0:

# As explained in lecture, we change the length of our rectangle

# (and then, implicitly, the width to keep the area the same), to make a

# "more square" rectangle with the same area.

length = (length + x/length)/2

length = (length + x/length)/2

length = (length + x/length)/2

length = (length + x/length)/2

length = (length + x/length)/2

else:

length = 0

# If the "rectangle" with area x were now a square, then the length

# of its side would be the sqrt.

return length

**Ans.**

**Q6:** Redo the body of fourth root so that it makes effective use of sqrt and returns an approximate fourth root of x. Edit the fourth root function in the module

**Ans.** **def fourth\_root(x):**

**"""Returns an approximate fourth root of x as float.**

**Precondition: The value of x is a positive number."""**

**if x!= 0:**

**temp = sqrt(x)**

**temp = sqrt(temp)**

**else:**

**temp =0**

**return temp # changed temp**

**Q7:** Un-indent the line that has ”question 7” in a comment and run the program. Look at the line number of the error message you get. Why is Python reporting a problem with a line after the one you un-indented?

**Ans. We get an indentation error because when we unindented the line we meant that the command is now out of the if loop and because the rest of the lines were still indented now being out of the while loop showed an error of incorrect indentation.**

**Q8:** Why does the program no longer output anything if you comment out the line if \_\_name\_\_ == ‘\_\_main\_\_’ but doesn’t give an error, either? (Hint: the program isn’t actually executing the remaining lines. Why?)

**Ans. The program doesn’t give any error because the rest of the code is indented the same as in the function fourth\_root and hence becomes a part of the function and since we have no more function calls and nothing is executed in the program.**

1. **Designs**

Here is a quick synopsis of three graphics procedures in SimpleGraphics:

DrawRect(x,y,L,W,FillColor=c1,EdgeWidth=s,EdgeColor=c2,theta=d) DrawDisk(x,y,r,FillColor=c1,EdgeWidth=s,EdgeColor=c2)

DrawLine(x1,y1,x2,y2, LineWidth=s,LineColor=c2)

Lecture 5 has lots of “how to” examples that illustrate the functions and built-in colors that are part of SimpleGraphics. But you can also get details via the Python help facility. To illustrate, get in the Python interactive mode and enter

>>> from SimpleGraphics import \*

Now you can use the “help” facility like this

>>> help(DrawRect)

to learn about the various functions in SimpleGraphics.

**DemoGraphics**.

The module DemoGraphics.py looks like this:

# DemoGraphics.py

# CSE 101

# February, 2016

""" Draws a design with squares and a design with rings."""

from SimpleGraphics import \*

# First Figure

MakeWindow(6,bgcolor=DARKGRAY,labels=False) DrawRect(0,0,6,6,FillColor=CYAN,EdgeWidth=5,theta=0)

# Add more squares... #

Second Figure MakeWindow(10,bgcolor=DARKGRAY,labels=False)

# Rings

DrawDisk(0,1,2,EdgeWidth=10)

# Add more rings

...

ShowWindow()

From the command shell, run DemoGraphics.py: python DemoGraphics.py

Observe that it displays a square in one figure and a ring in a second figure (the second window might be on top of the first window, so you may need to move it to see the first window as well).

If you want to run this script again you must close both figure windows. To close a figure window, click on the little “x box” (Windows) or red circle (Mac) that you see in its upper right corner. Thus, the standard work pattern when you are developing graphics python code is:

Step 1. Edit the module in some editor.

Step 2. Save the module.

Step 3. Run the module from the command shell.

Step 4. Look at the displayed figures.

Step 5. Close the figures and go back to Step 1

**Q9:**With that in mind, modify DemoGraphics.py so that it produces these two figures:

|  |  |
| --- | --- |
|  | A description... |

For the squares you will need more calls to DrawRect but with lesser side lengths. These should produce squares with different sizes at the same centre, and the same perimeter highlighting. But they are to be rotated. The overall design has many squares but you need to draw only 6 square. The second square should be at rotation -5 degrees. The third square drawn should with rotation -10 degrees. And the fourth square drawn should be with rotation -15 degrees and so on.

For the packing of circles in a triangle shape you must add in total 6 disks. All the rings are of same radius, i.e., r=4. Assume the 2nd disk in the bottom most row is at x=0, and y=0.

**Indian Flag:**

Here are specs of Indian Flag courtesy of Wikipedia:

1. Length L of the flag is 1.5 times of its width W.
2. Each rectangle is of same dimension.

* Top most rectangle has saffron color [0.738, 0.0860, 0.102]

1. Middle rectangle has White color [0.9, 0.9, 0.9]
2. And the bottom most rectangle has Green color [0.288, 0.395, 0.317]
3. Middle rectangle has Ashok Chakra which has 24 spokes starting from the centre in a disk of radius = W/6.

**Q10:** Your job here is to draw the Indian Flag. Make edits to IndianFlag.py for this task. Make use of procedures in the SimpleGraphics module to complete *DrawIndianFlag*.