**Lab-4, CSE 101**

**NAME:**

**ROLL NO.:**

**Getting credit:**

Complete all required blank boxes and lines on this handout. When you are finished, show your written answers and output to the codes to one of the TAs in your section. The TA may ask you a few questions to make sure you understand the material.

**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 file. **Make sure you mention your name and roll number 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. 5.000000000000 is the actual square root. The DemoMath sqrt function gives 5.000023178254. The actual square root of 2500 is 50.000000000000. The DemoMath function gives us 88.502206389628.**

**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. When we say DemoMath.sqrt(), we explicitly refer to the python module DemoMath’s function sqrt. As DemoMath has not been imported , the program gives an error on execution.**

**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. It would give an error because the corresponding Python module wasn’t imported.**

**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. “TypeError: float argument required, not NoneType”**

**This error is given because we are assinging a function which returns None to a float variable.**

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

**Ans. With ref. To ‘length = (length + x/length)/2’ x/length will be floating point and adding it to an int will give an error. Therefor**

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

**Ans. Add an if statement -> if(x == 0): {return 0}**

**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."""**

**root = math.sqrt(x);**

**temp = float(root)**

**temp = (temp + root/temp)/2**

**temp = (temp + root/temp)/2**

**temp = (temp + root/temp)/2**

**temp = (temp + root/temp)/2**

**temp = (temp + root/temp)/2**

**return 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. Because the indent looks for a colon above it to check whether it belongs there or not. Therefore, there is an unexpected indent.**

**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. If \_\_name\_\_ == ‘\_\_main\_\_’ is used to run the program as a standalone one. When it is not present, it runs as a module by default.**

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.
3. Top most rectangle has saffron color [0.738, 0.0860, 0.102]
4. Middle rectangle has White color [0.9, 0.9, 0.9]
5. And the bottom most rectangle has Green color [0.288, 0.395, 0.317]
6. 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. Note that this module contains a procedure, DrawChakra, defined to draw the AshokChakra. Make use of it and other procedures in the SimpleGraphics module to complete *DrawIndianFlag*.