CS 1110 SPRING 2016: LAB 3: PRACTICE FOR A2 (Feb 23-24)

http://www.cs.cornell.edu/courses/cs1110/2016sp/labs/lab03/lab03.pdf

First Name: _____ **Last Name**: _____ **NetID**: _____

The lab assignments are very important. Remember this: *The lab problems feed into the assignments and the assignments define what the exams are all about.*

Start *before* your lab meets. We recommend spending an hour or two on the lab *before* coming to your section, so you can use your in-person time to ask questions most efficiently.

Also, this strategy of starting beforehand increases your chances of checking in at your lab section, which will probably take less time than waiting in line at consulting hours!

Getting credit. Complete all required blank boxes and lines on this handout. When you are finished, show your written answers to one of the CS 1110 lab staff in your section on Feb 23-24 or in any consulting hours up to and including Monday Feb 29(earlier days will probably have shorter lines). The staff member will ask you a few questions to make sure you understand the material, and then swipe your Cornell ID card or directly make a notation in CMS to record your success. This physical piece of paper is yours to keep.

Getting set up. From the Lab webpage, download `DemoGraphics.py`, `Turkey.py`, and `DemoMath.py`. From the Lecture webpage download `simpleGraphics.py`. Put them all in the same folder, say `Lab3`. In the command shell, navigate the file system so that this folder is THE CURRENT WORKING DIRECTORY.

1. DemoMath

This part of lab is about the module `DemoMath.py`:

```
# DemoMath.py
# CS 1110 (cs-1110profs-L@cornell.edu)
# February, 2016
""" Examines a function that computes approximate square roots."""

import math


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,
```

Course authors: D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White

```
    # which will thus have area x.
    length = float(x)

    # 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

    # If the "rectangle" with area x were now a square, then the length
    # of its side would be the sqrt.
    return length


def fourth_root(x):
    """Returns an approximate fourth root of x as float.

    Precondition: The value of x is a positive number."""
    return x # change this for question 6!

# Application Script
if __name__ == '__main__':
    """ A keyboard input framework for checking out sqrt.
    """

    x = input('Enter a number whose square root you want: ')
    y1 = math.sqrt(x)
    y2 = sqrt(x) # question 2: change this to "y2 = DemoMath.sqrt(x)"

    print '\n\n           x = %5.2f' % x # question 7: unindent this
    print 'math.sqrt(x) = %15.12f' %y1
    print '     sqrt(x) = %15.12f' %y2
    z1 = math.sqrt(math.sqrt(x))
    z2 = fourth_root(x)
    print '\n  True 4th Root = %16.12f' %z1
    print '  fourth_root(x) = %15.12f' %z2
```

(1) What is displayed if you ask for the square root of 9? Is this program's `sqrt(9)` correct according to `math.sqrt(9)`? What is displayed if you ask for the square root of 900? Is this program's `sqrt(900)` correct according to `math.sqrt(900)`?

9 should look good, 900's square root should be off, when the output of math.sqrt is compared against sqrt. So math.sqrt and sqrt are DIFFERENT FUNCTIONS.

(2) 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`".

NOTE: students might find this confusing, so do check in with them on this. Python is looking in the file it is "in" to find some definition of the name "DemoMath". But that name isn't defined anywhere.

From the above, you should understand: 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?

In both cases, the name "math" isn't available. (And we need the name "math" to be able to talk about Python's math function rather than the sqrt function we defined.)

(3) 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.

When a function has no return value specified, its return value is None.

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

(5) How could `sqrt` be modified so that it could handle the input `0`?

(6) Redo the body of `fourth_root` so that it makes effective use of `sqrt` and returns an approximate fourth root of `x`.

return `sqrt(sqrt(x))`

(7) Unindent 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?

Python thinks your de-dented line is the next command after the "if" statement. That's valid. It's the fact that the next line is indented that surprises it, because there's no ending colon in the line you de-dented.

(8) 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?)

the lines that were in the application script are now considered to be a continuation of the fourth_root function. So there's no error; those lines are valid parts of a function definition, but a module doesn't need to have an application script in it.

2. Designs

Here is a quick synopsis of the 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)
DrawStar(x,y,r,FillColor=c1,EdgeWidth=s,EdgeColor=c2,theta=d)
```

In each case `x` and `y` define the center. Disks and stars have a radius (that's `r`). For fill color, `c1` must be a built-in color like `YELLOW` or an rgb array like `[.2,.3,.4]`. If no color is specified, then there is no fill and the object is "transparent." The parameter `EdgeWidth` controls how much the edge is highlighted. Typical values for `s` are 0, 1 (the default), 5, 10. To color the edge, `c2` must be a built-in color like `YELLOW` or an rgb array like `[.2,.3,.4]`. If no color is specified, then there is no highlighted edge. To rotate a star or rectangle counterclockwise set `theta` to be the degree measure of the counterclockwise rotation (default is 0.)

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`.

Using the help facility, explain in English what `MakeWindow (10,labels=False,bgcolor=PURPLE)` does.

2.1. **DemoGraphics.** The module `DemoGraphics.py` looks like this:

```
# DemoGraphics.py
# CS 1110 (cs-1110profs-L@cornell.edu)
# 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
```

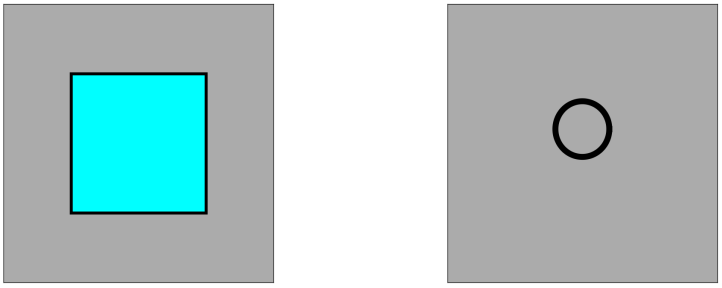
```
DrawDisk(0,1,2,EdgeWidth=10)
# Add more rings...

ShowWindow()
```

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

```
python DemoGraphics.py
```

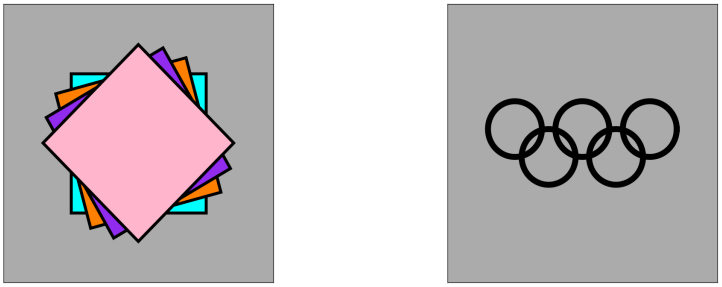
Observe that it displays a square in one figure and a ring in a second figure (the second window might be on top of the first window, so you may need to move it to see the first window as well):



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 Komodo Edit.
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.

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

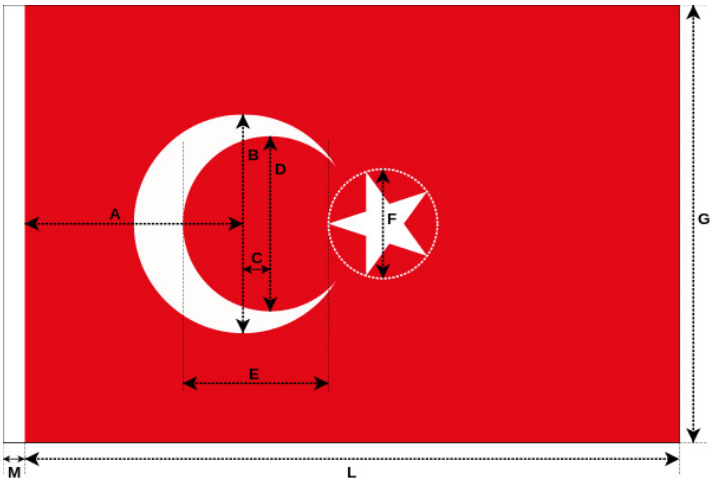


For the squares, you will need three more calls to `DrawRect`. These should produce squares with the same size, the same center, and the same perimeter highlighting. But they are to be

colored differently and rotated. The overall design has four squares. The second square should be ORANGE with rotation 15 degrees. The third square drawn should be PURPLE with rotation 30 degrees. And the fourth square drawn should be PINK with rotation 45 degrees. How many squares would you see if the PINK square had a rotation of 90 degrees? Why?

For the Olympic rings you must add four more rings. (Note that a "ring" is just a disk with no fill color and a thick perimeter.) Obtain the top row of rings by adding in one ring to the left of the given ring (shift left by 5 units) and one ring to the right of the given ring (shift right by 5 units). After that is working, add in the bottom two rings. Their centers are on the horizontal line $y = -1$ and their centers are 5 units apart from each other.

2.2. **Turkey (NOT REQUIRED FOR LAB CHECK-OFF).** Here are the specs for the flag of Turkey courtesy of Wikipedia:

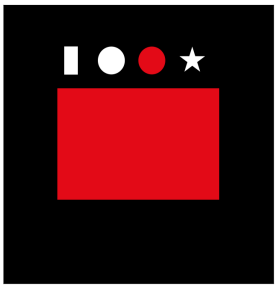


Letter	Measure	Length
G	Width	G
A	Distance between the centre of the outer crescent and the seam of the white band	G/2
B	Diameter of the outer circle of the crescent	G/2
C	Distance between the centres of the inner and outer circles of the crescent	G/16
D	Diameter of the inner circle of the crescent	2G/5
E	Distance between the inner circle of the crescent and the circle around the star	G/3
F	diameter of the circle around the star	G/4
L	Length	3G/2
M	Width of the white hem at the hoist	L/30

Take a look at the module `Turkey.py`. It houses a function for drawing the Turkish flag an an application script that can be used to check it out. By entering

```
python Turkey.py
```

we get this:



Obviously, `DrawTurkey` is not properly locating the white rectangle, the white disk, the red disk and the star. And they are all improperly sized. And the star is not tilted (try 18 degrees counterclockwise). You job is to correct these flaws using the specs given above as your guide.