Lab 3: Graphics Library Exploration

Course: CS65 - Introduction to Computer Science (Fall 2022)
Instructor: Md Alimoor Reza, Assistant Professor of Computer Science, Drake University
Due: Tuesday, September 27, 11:50 PM

You will be creating a python file for each task separately (with a **.py* extension), where you will save your python instructions. By default, Thonny opens an unnamed file in the a *text editor* (top text pane). It is an excellent practice to write a formal header and suppress it as comments. A line of code with a preceding # is considered as a comment in Python. You should type in the necessary parts as shown below, e.g., *your name, your contact email, description, etc.*. Save the file using the format as follows *firstname_lastname_lab3_task*.py* (all in lowercase letters). For example, I saved my file as *md_reza_lab3_task3.py*.

Driver and navigator: Now, you will be doing collaborative work for this lab. These exercises should be completed with your assigned partner. Both of you will take turns playing the role of i) the driver and ii) the navigator. The driver's responsibility is to type into Thonny when completing each exercise. The navigator's responsibility is to support the driver by reading the exercises, looking up relevant readings, and identifying errors.

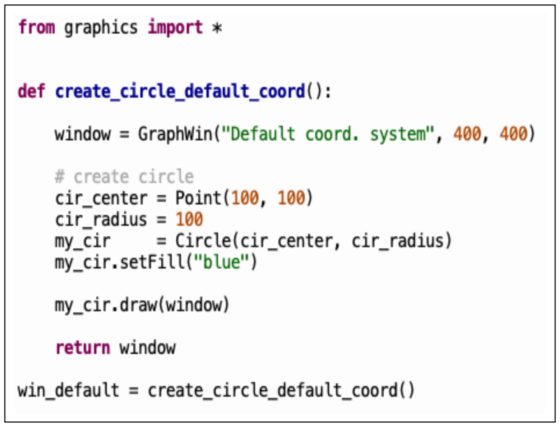
Tasks and file submission: This lab consists of several small tasks. We expect that you submit separate file for each task (a python file with a *.py* extension). In your python file, add comments to briefly explain what you are doing. You should submit your individual copy by acknowledging your partner's name at the beginning of your python file (inside a comment).

Drawing circles, triangles, and other shapes

The package **graphics** needs to be installed in Thonny. You might have already installed it. In case you have not, please follow the instructions in Lecture 5. You will implement a Python program that will draw basic graphical objects such as *circle, rectangle, triangle* and *line*. It is recommended that you write a separate function for each of the above-mentioned shapes.

Task 1: Drawing circles of different sizes, at different locations

Here is a function that draws a circle. Your goal is to write another function – *possibly with a different name* – with the following criteria. Instead of hard-coding the values inside the function body, you should ask the user to provide the values for you. *Hint: you should use built-in function input("")* to prompt the user to provide inputs. 'Hard-coding' refers to the fact that variables such as cir_radius = 400 and cir_center = Point(100, 100) have been initialized with fixed and predetermined values. It is not a very flexible way of writing your code. Your new function should have a different set of functions *parameters*. Determine how many *parameters* do you need for your new function and finish making the necessary changes inside the function. Once you **define** a function, you need to **call** it in order to see its execution.

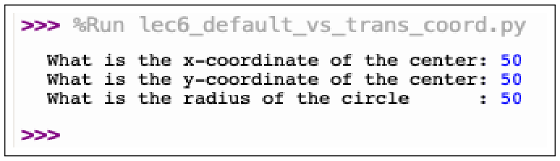


```
from graphics import *

def create_circle_default_coord():

    window = GraphWin("Default coord. system", 400, 400)

    # create circle
    cir_center = Point(100, 100)
    cir_radius = 100
    my_cir     = Circle(cir_center, cir_radius)
    my_cir.setFill("blue")

    my_cir.draw(window)

    return window

win_default = create_circle_default_coord()
```

For example, the function above can be called with the following python line:

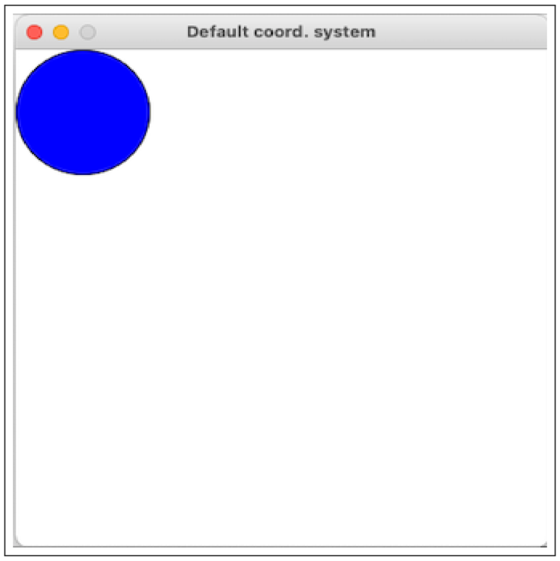
win_default = create_circle_default_coord()

Notice the function returns a variable *window*, which is defined inside it. When you **call** the function, you need to save it in a separate variable. In the example above, it is saved in the variable *win_default*. Can you tell the difference between the two variables? More precisely – Is *window* a local-variable or global variable? How about *win_default*? Write the difference inside your python file as a comment (with a preceding #).
Define the function and call them as instructed. When you run your program, it should prompt you to do the following:



```
>>> %Run lec6_default_vs_trans_coord.py

  What is the x-coordinate of the center: 50
  What is the y-coordinate of the center: 50
  What is the radius of the circle      : 50

>>>
```

Here is the output you should expect to see:



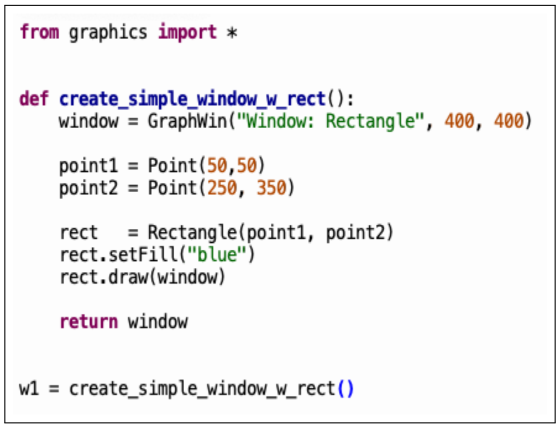
You should try run your program with different user inputs and see the differences.

Task 2: Drawing rectangles of different sizes, at different locations

Following the previous task, can you write another function which draws **rectangles** of different sizes. As discussed in class, a *rectangle* (according to **graphics library** specification) is defined by the two corners:

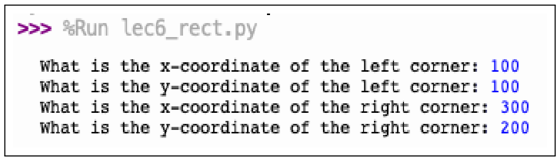
- left-most corner (2D point)
- right-most corner (2D point)

You can define a corner (2D point) the same you have defined it for the center of the circle. Here is a sample function that draws a rectangle.



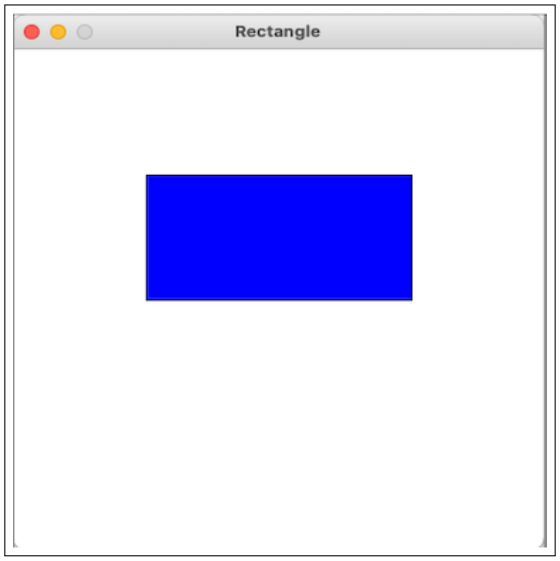
```
from graphics import *


def create_simple_window_w_rect():
    window = GraphWin("Window: Rectangle", 400, 400)

    point1 = Point(50,50)
    point2 = Point(250, 350)

    rect   = Rectangle(point1, point2)
    rect.setFill("blue")
    rect.draw(window)

    return window


w1 = create_simple_window_w_rect()
```

Instead of hard-coding the values inside the function body, you should ask the user to provide the values for you. When you define your new function, it should have a different number of *parameters*. Determine how many *parameters* do you need for your new function and finish making the necessary changes inside the function. Once you **define** a function, you need to **call** it in order to see its execution. Define the function and call them as instructed. When you run your program, it should prompt you to do the following:



```
>>> %Run lec6_rect.py

  What is the x-coordinate of the left corner: 100
  What is the y-coordinate of the left corner: 100
  What is the x-coordinate of the right corner: 300
  What is the y-coordinate of the right corner: 200
```

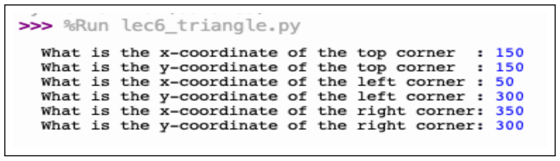
Here is the output you should expect to see:



You should try run your program with different user inputs and see the differences.

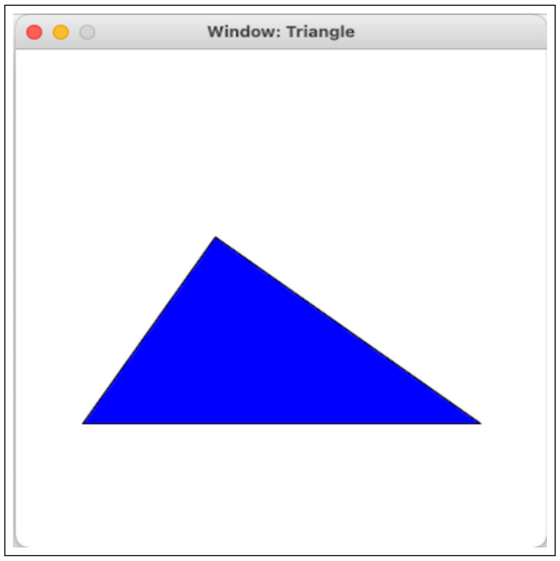
Task 3: Drawing triangle of different sizes, at different locations

Now can you write another function that draws **triangles** of different sizes? A *triangle* (according to **graphics library** specification) is defined by the three corners. You can define a corner (2D point) the same as you defined it for a rectangle. Read the specification and find out the appropriate graphical object for this task: https://mcsp.wartburg.edu/zelle/python/graphics/graphics/graphref.html.
Instead of hard-coding the values inside the function body, you should ask the user to provide the values for you. When you define your new function, it should have a different number of *parameters*. Determine how many *parameters* do you need for your new function and finish making the necessary changes inside the function. Once you **define** a function, you need to **call** it in order to see its execution. Define the function and call them as instructed. When you run your program, it should prompt you to do the following:



```
>>> %Run lec6_triangle.py
  What is the x-coordinate of the top corner   : 150
  What is the y-coordinate of the top corner   : 150
  What is the x-coordinate of the left corner  : 50
  What is the y-coordinate of the left corner  : 300
  What is the x-coordinate of the right corner: 350
  What is the y-coordinate of the right corner: 300
```

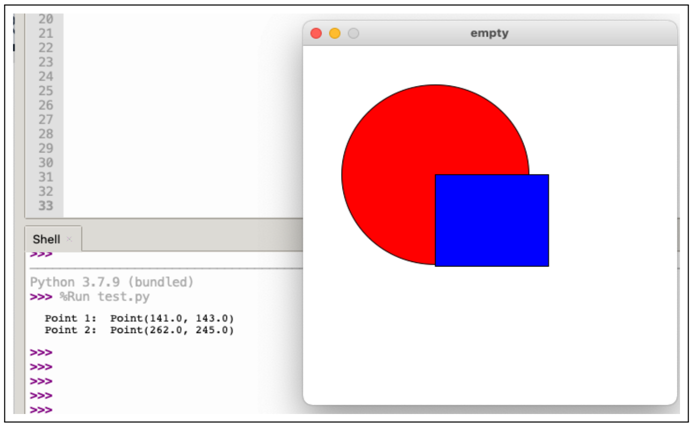
Here is the output you should expect to see:



You should try run your program with different user inputs and see the differences.

Task 4: Drawing rectangle with mouse clicks

Write a new function for this task. Instead of receiving *x*- and *y*-coordinates from the user using *input()*, you should click the mouse in two different locations in the graphics window. Those two locations should serve as corners of the rectangle. Recall that two corners define a rectangle: i) left-most corner (2D point) and ii) right-most corner (2D point). Additionally, the first corner should also serve as the center of a circle of radius **100** pixels. Once you **define** a function, you need to **call** it in order to see its execution. Define the function and call them as instructed. Here is the output you should expect to see:

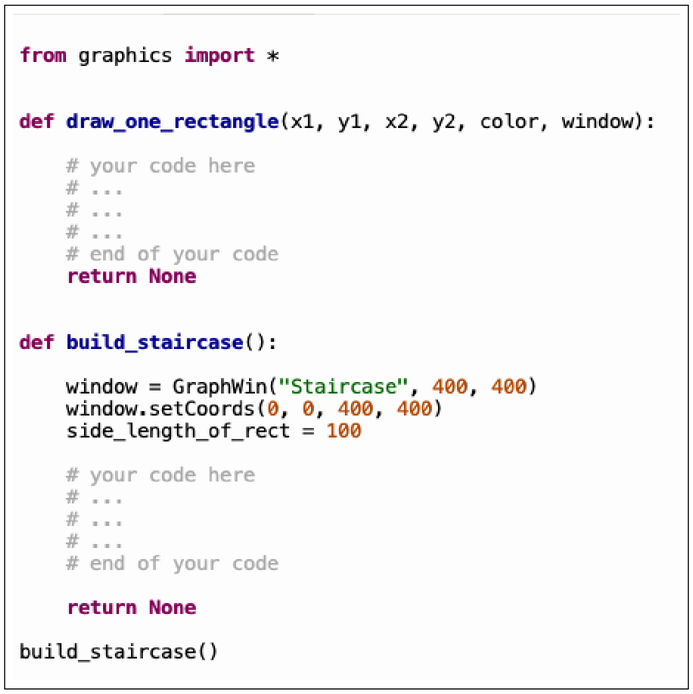


```
>>>
Python 3.7.9 (bundled)
>>> %Run test.py

  Point 1:  Point(141.0, 143.0)
  Point 2:  Point(262.0, 245.0)

>>>
>>>
>>>
>>>
>>>
```

You should try run your program with different user inputs and see the differences.

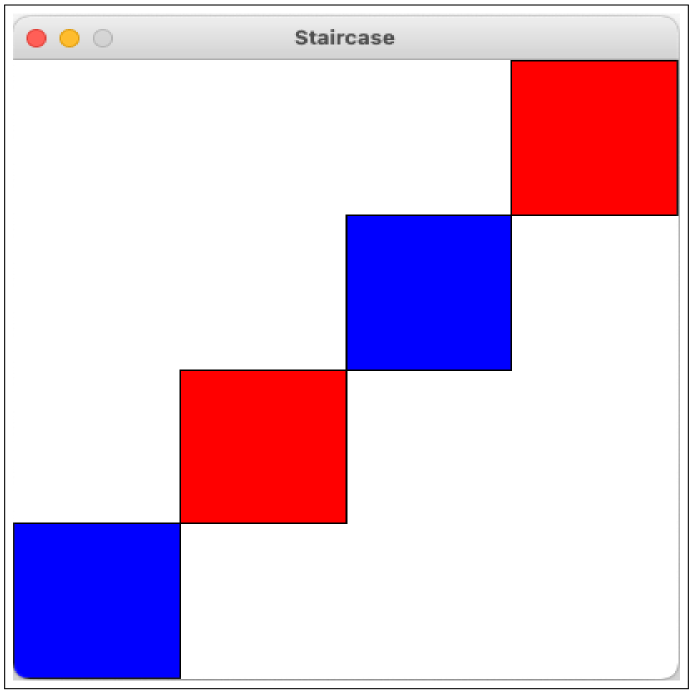
Task 5: Building a staircase

Your task is to build a staircase. The building blocks of the staircase are rectangles with alternating colors (as shown below). You have to create **4** rectangles. The length of each side of a rectangle should be **100** pixels. The coordinate system should be transformed so that the origin (0,0) is located at the bottom-left corner. The size of the window remains 400 pixels by 400 pixels. There are various ways you can attempt to solve this task. I recommend that you follow the basic structure of the code given below. In this code structure, you should not be using any global variables. The function *build_staircase()* should call another function *draw_one_rectangle(...)* each time it needs to build a rectangle. In each function call, you have to provide necessary *arguments* so that they match the *parameters* of function *draw_one_rectangle(...)*.



```
from graphics import *

def draw_one_rectangle(x1, y1, x2, y2, color, window):
    # your code here
    # ...
    # ...
    # ...
    # end of your code
    return None

def build_staircase():

    window = GraphWin("Staircase", 400, 400)
    window.setCoords(0, 0, 400, 400)
    side_length_of_rect = 100

    # your code here
    # ...
    # ...
    # ...
    # end of your code

    return None

build_staircase()
```

Define the function and call them as instructed. Here is the output you should expect to see:



What to turn in:

Please submit a zip file containing your source code (five python files with *.py* extension with proper naming convention described above) for this lab. Please make sure that your code runs prior to submission. Log in to Blackboard and find Lab 3 then submit your zip file.