ECE 122: Introduction to Programming for ECE- Summer 2019

Project 3: Drawing and Animations with Tkinter (OOP, Encapsulation and Graphics examples)

Due Date: see website, class policy and Moodle for submission. This is an individual project- to do it alone (discussions are encouraged but no sharing of code)

Description:

The goal of this project is to practice some OOP using class, instance variables and methods. You will also operate Graphics.

The project must include four files:

`Mapping_for_Tkinter.py`

> A module containing the class Mapping_for_Tkinter (that will be used by all other application files) and a main function for simple testing and demo.

`plotter.py`

> An application file that plots mathematical functions.

`bouncing ball.py`

> An application file that simulates a bouncing ball.

`cannon ball.py`

> An application file that simulates a cannonball.

The project is designed to be developed incrementally' one task at a time. You can then debug, test and run your code after each new task/option is implemented. After Task 1 has been done, all the other Tasks/options can be completed in any order. Do not forget to comment your code. Make sure you obtain the exact same output for the exact same input for the application examples (this includes syntax, blank spaces, and skipping blank lines). Your program will also be tested with different inputs.

Submission/Grading:

Read this document completely and look over the code provided. Then begin to develop task 1. Once you have done that, and you have verified that your code works correctly, move on to the next task.

Use your preferred IDE to write and save your files- preferably as a project.

Your *project submission must be a .zip file (not a 7-zip, rar, or tarball) and must contain only the four files mentioned above. The zip file be named in this manner:*

Your first initial_your last name_ECE122project3.zip For example, my name is Gordon Anderson, so my file would be named:

g_anderson_ECE122project3.zip

I will only accept any files that follow this naming convention. I will only accept files with a .zip extension.

Remember: *any function definitions should go at the top of the file*. Do not forget to comment your code when necessary. Make sure you obtain the exact same output for the exact same input for the examples below (this includes syntax, blank spaces, and skipping blank lines). Your program will also be graded on coding style as this is a very important aspect of writing code.

Important: Use the style detailed in the *"Python Coding Style"* link posted on Moodle. Remember that you are writing code for another human to read and understand. Someone else will need to work with your code at some point. Good coding style makes that much easier, and your code looks professional, which inspires confidence in your work.

Grading:

1. Your program should implement all basic functionality/Tasks and run correctly (100 points).

2. This time the overall programming style grade is included in the 100 grading. It means that up to 10pts will be withdrawn if programs do have proper identification, and comments.

Tasks to Complete (4 tasks, total of 100pts):

Task-1- [30pts]

First develop the module Mapping for Tkinter as all other modules rely on it. Once you create a simple root window with its associated canvas in Tkinter, the origin is located at the top left corner and the y-axis points down. From now on, we will talk about (i,j) coordinates for the Tkinter canvas such that (i=0,j=0) for the top left corner, (i=0,j=height) for the bottom left corner, (i=width,j=0) for the top right corner, and (i=width,j=height) for the bottom right corner. One can then quickly realize that this coordinate system is not that convenient if you want to represent mathematical expressions. Math equations are better represented using a traditional coordinate system where the x-axis extends from x_{min} to x_{max} (left to right) and the y-axis extends from y_{min} to y_{max} (bottom to up). We propose to develop a mapping that will allow us to convert any (x,y) points in the math system to the corresponding points (i,j) in the Tkinter canvas, and vice-versa. The two coordinate systems are represented in Figure 1.

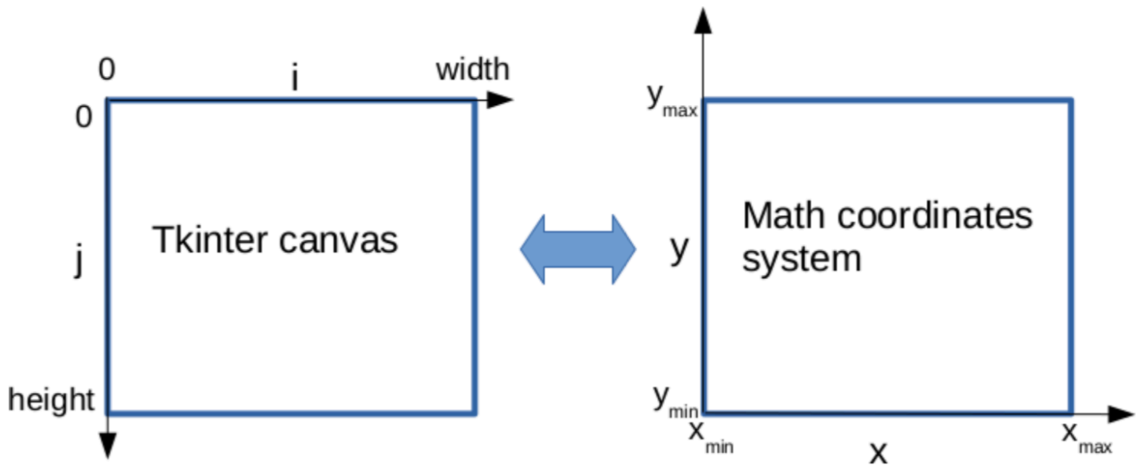


Figure 1: Tkinter canvas (left), Math coordinate systems (right).

To achieve this mapping, we propose to implement a new class Mapping for Tkinter. Here is an example of a statement that that instantiates a new mapping object:

```
mapping = Mapping_for_Tkinter(-5.0,5.0,-5.0,5.0,500)
```

Consider the following arguments for the Mapping_for_Tkinter constructor:

```
Mapping_for_Tkinter(xmin, xmax, ymin, ymax, width)
```

where `xmin, xmax, ymin, ymax` (float numbers) will be used to set the math coordinate system, and `width` (int number) is the width of the Tkinter canvas that we want to create (width is in number of pixels). The class will use those arguments to initialize the instance attributes (of the same name for example). In addition, you will declare the class attribute `height` that must be calculated proportionally to the math coordinates using this formula:

$$height = width * \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

The variable `height` will need to be converted to an int number (since it represents a discrete number of pixels). All attributes should be private and you will be using encapsulation. You will use the traditional OOP way to implement encapsulation using getter and setter methods. You will then need to implement these set and get methods:

`set_xmin, set_xmax, set_ymin, set_ymax, set_width,`

and the private method `__set_height`

`get_xmin, get_xmax, get_ymin, get_ymax, get_width, get_height.`

Once this is done, you need to define the following instance methods that will allow other modules to make use of the mapping object. You will need to implement:

- `get_x` that accepts a parameter i (x coordinate in the Tkinter canvas) as argument and returns the x coordinate in the math format.
- `get_y` that accepts a parameter y (y coordinate in the Tkinter canvas) as argument and returns the y coordinate in the math format.
- `get_i` that accepts a parameter x in math format and returns the i (int) coordinate in the Tkinter format.
- `get_j` that accepts a parameter y in math format and returns the j (int) coordinate in the Tkinter format.
- `__str__` that returns a string representation of the object.

Note: You can use the following mappings:
for (i,x):
  if i = 0 then x = x_{min}
  if i = width – 1 then x = x_{max}
for(j,y):
  if j = 0 then y = y_{max}
  if j = height–1 then y = y_{min}

Complete the code for the Mapping_for_Tkinter which already contains the following main method:

```python
def main():
    m = Mapping_for_Tkinter(-5.0, 5.0, -5.0, 5.0, 500)  # instantiate mapping
    print(m)  # print info about object

    window = Tk()  # instantiate a tkinter window
    canvas = Canvas(window, width=m.get_width(), height=m.get_height(), bg="white")  # create a canvas width*height
    canvas.pack()
    # create rectangle the Tkinter way
    print("Draw rectangle using tkinter coordinates at (100,400) (400,100)")
    canvas.create_rectangle(100, 400, 400, 100, outline="black")

    # create circle using the mapping
    print("Draw circle using math coordinates at center (0,0) with radius 3")
    canvas.create_oval(m.get_i(-3.0), m.get_j(-3.0), m.get_i(3.0), m.get_j(3.0), outline="blue")

    # create y=x line pixel by pixel using the mapping
    print("Draw line math equation y=x pixel by pixel using the mapping")
    for i in range(m.get_width()):
        x = m.get_x(i)  # obtain the x coordinate
        y = x
        canvas.create_rectangle((m.get_i(x), m.get_j(y)) * 2, outline="green")

    window.mainloop()  # wait until the window is closed
```

It should execute this way in the console:

```
Mapping created between x=[-5.0,5.0] y=[-5.0,5.0] math => (500,500) tkinter
Draw rectangle using tkinter coordinates at (100,400) (400,100)
Draw circle using math coordinates at center (0,0) with radius 3
Draw line math equation y=x pixel by pixel using the mapping
```

and create this graphical output:

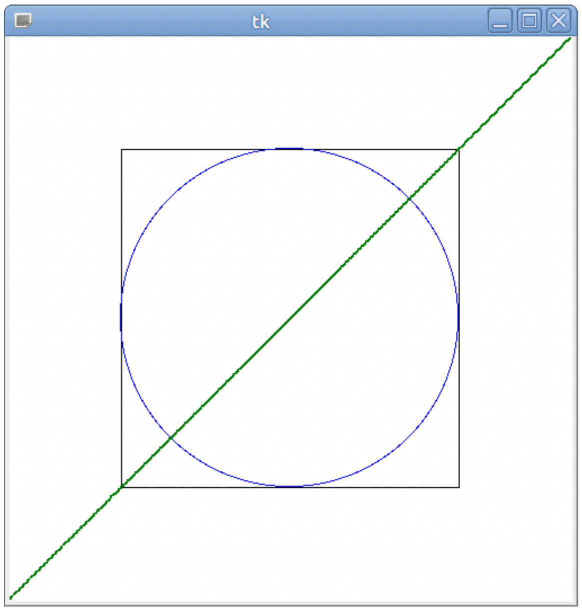


Figure 2: Output of running Mapping_for_Tkinter.

Task2: plotter.py- [30pts]

Here is an example of the console interaction output after execution of plotter:

```
Enter math formula (using x variable): 0.4*x*sqrt(abs(x))
Enter xmin,xmax,ymin,ymax (return for default -5,5,-5,5):
```

In this example, we chose to enter our own formula but use default values for the boundary of the coordinate systems. We also obtain the graphical plot in Figure 3.

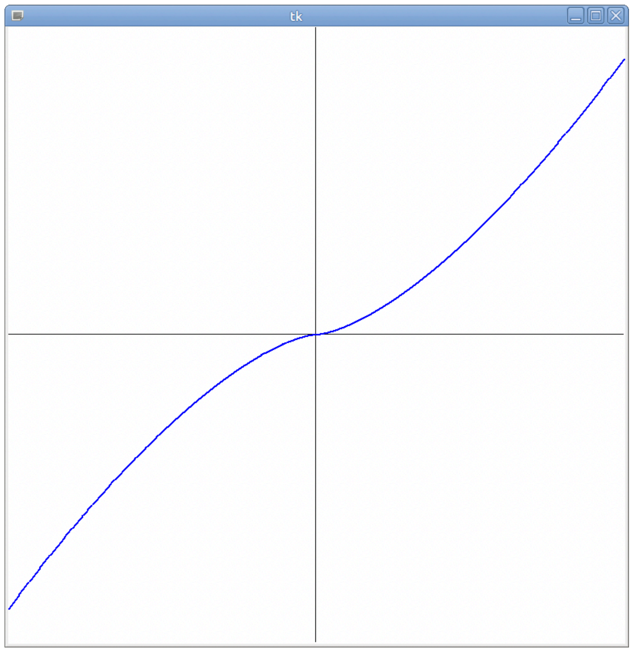


Figure 3: Example output of Task2- 0.4 * x * |x|

Another example where we enter a formula and some bounds:

```
Enter math formula (using x variable): sin(x)**2
Enter xmin,xmax,ymin,ymax (return for default -5,5,-5,5): -6.28 6.28 -0.01 1.1
```

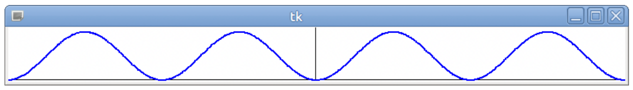


Figure 4: Example output of Task2- $\sin(x)^2$

What you need to implement:

1. Complete the plotter.py file. You will be using the mapping class defined in task1 to translate from the user coordinates to Tkinter coordinates.

2. Get the formula and coordinate strings from the user with the input function. If the user did enter coordinates (not using the default), you can use the split method to tokenize the coordinates as a list, then use the map function to convert the elements in the list to floating point numbers. This uses the built-in functions of python in one statement!

3. The mathematical formula will be stored as a string. It is notoriously difficult to parse a mathematical formula to identify and extract the math operations. Luckily, python provides a very powerful function called eval that can be used to evaluate a string as if it was a piece of code. (Note that the function eval could be dangerous to use for security reasons because you may not know what code you are evaluating. This code could include malware, a virus, etc.). The eval function will be fine for our purpose here.

4. Use a width of 800 for the canvas for all situations in this file.

5. If 0 is included between [xmin,xmax] and/or [ymin,ymax], you will also plot the x and y axis (in black).

6. To plot the math function (in blue) you can take inspiration from the last part of the main method in Task1.

7. Finally, we also would like to have some safeguards in case of typos while entering the coordinates. You will need to modify the set_xmax and set_ymax methods in the Mapping_for_Tkinter class. For example: if you call the method set_xmax, it should test if the argument xmax is <= the instance variable __xmin. If not, it should print this prompt:

   "Your xmax is invalid (xmax<=xmin), Re-Enter correct [xmin,xmax]: "

   until the user has entered an appropriate xmax. Do the same for the set_ymax method.

Here is an example of the console interaction:

```
Enter math formula (using x variable): sin(x)
Enter xmin,xmax,ymin,ymax (return for default -5,5,-5,5): 3.14 -3.14 3.14 -3.14
Your xmax is invalid (xmax<=xmin), Re-Enter correct [xmin,xmax]: -3.14 3.14
Your ymax is invalid (ymax<=ymin), Re-Enter correct [ymin,ymax]: 1 -1
Your ymax is invalid (ymax<=ymin), Re-Enter correct [ymin,ymax]: -1 1
```

Task3- Bouncing Ball- [20pts]

It is time to do a bit of (simple) physics with a bouncing ball. The following is the console interaction if we execute the application `bouncing_ball.py` (with default values- just clicking enter):

```
Enter xmin,xmax,ymin,ymax (return for default -300,300,-300,300):
Enter x0,y0,v,theta (return for default 0,0,70,30):
Total number of rebounds is: 24
```

The last statement is printed after 15s of simulation time once the animation of the bouncing ball is done (using a while loop that breaks when t_total = 150).

Two examples video of the bouncing balls are included in this project: bb1.mp4 that uses the default values (see below for a screen shot),

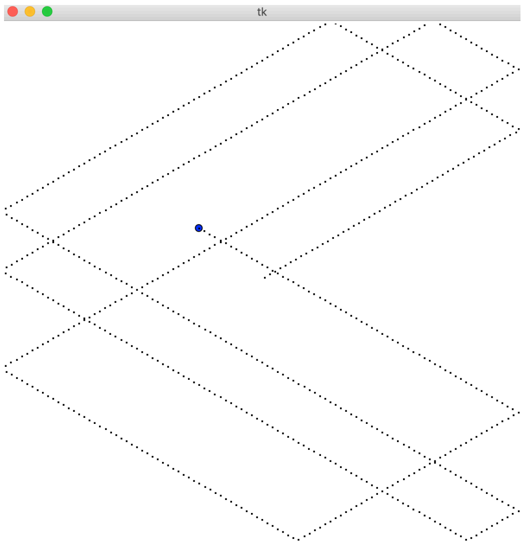


Figure 5: bouncing_ball.py with defaults.

and bb2.mp4 which is associated with this console interaction:

```
Enter xmin,xmax,ymin,ymax (return for default -300,300,-300,300): -400 400 -40 40
Enter x0,y0,v,theta (return for default 0,0,70,30): 0 0 80 60
Total number of rebounds is: 133
```

What you need to implement:

1. You will complete the `bouncing_ball.py` file. You will be using an instance of the Mapping_for_Tkinter class defined in task1.

2. The ball itself is a filled blue circle of radius 4. The center of the ball is taken as

reference for the position of the ball (to ease the problem). It means that a rebound will happen when the center of the ball reaches one side (technically half of the ball will be momentarily absorbed/hidden by the wall).

3. The variables x0 and y0 specified the original position of the ball. The variable v represents its velocity in m/s and theta is the original angle (30 degree is the default value). Starting from x0,y0 at time t=0, the new position of the ball can be obtained using the following simple formula (θ is in radian):

   $x = x_0 + v \cos(\theta)t$

   $y = y_0 + v \sin(\theta)t$

   Let us call a single event, the ball trajectory/animation until it reaches one side (one boundary).

4. If the ball reaches either the left or right sides, its angle will change to $\pi - \theta$

5. If the ball reaches either the bottom or top sides, its angle will change to $-\theta$

6. As soon as a rebound occurred you could initialize x0,y0 at the positions x and y and then start a new event by setting the time to 0.

7. You should still keep track of the total time, at each iteration the time is incremented by 0.1s and you would exit the loop as soon as the total time reaches 150s (1500 loops). This is the real physical time. For the animation you will be waiting 0.01s (using the sleep method of the module time) before updating the tkinter window, so the simulation time should be 15s.

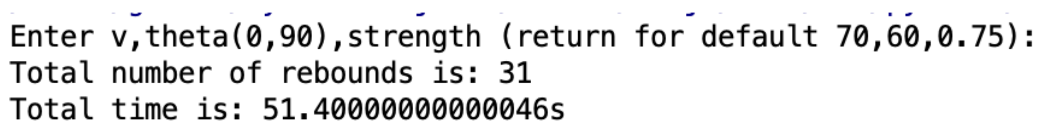
8. In addition to the ball animation, you also keep track of the trajectory by plotting it.

9. At the end, the ball turns red.

10. You will also keep track on the number of rebounds.

Task4- Cannonball [20pts]

A little more physics. This is the console interaction when the code is run with default values:



```
Enter v,theta(0,90),strength (return for default 70,60,0.75):
Total number of rebounds is: 31
Total time is: 51.40000000000046s
```

The video file cb1.mp4 shows the graphical output using the default values. A screen shot is also seen below.

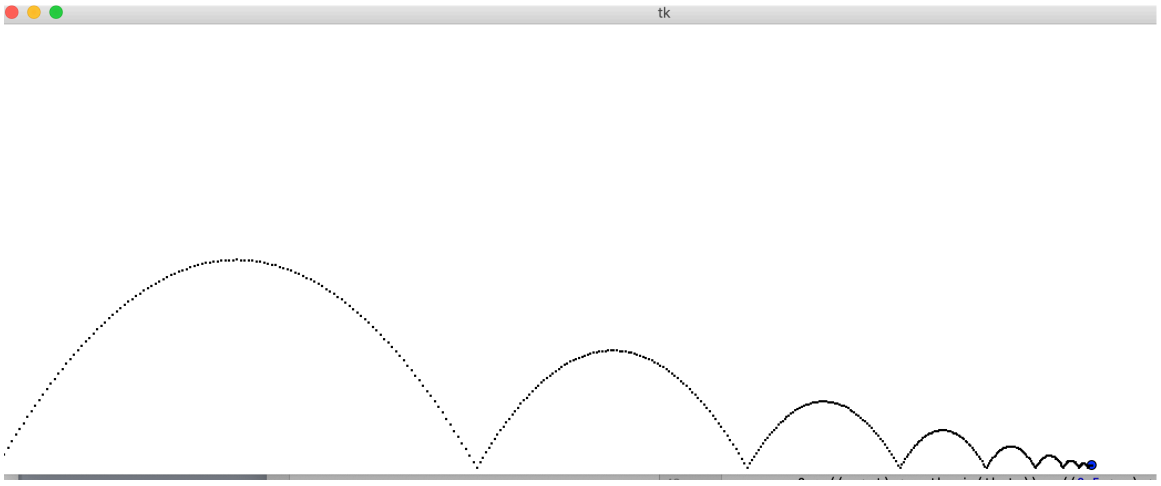


Figure 6: cannon_ball.py with defaults.

As you can see the ball is now subject to a gravitational force (along the y-axis), it can still rebound when it reaches the surface floor but it is gradually loosing strength until reaching a complete stop.

What you need to implement:

1. The mapping object should be instantiated with this statement:

```
mapping=Mapping_for_Tkinter(0.0,1200.0,0.0,400.0,1200)
```

2. The x0,y0 origin of the ball is at the left bottom corner at (0,0). Similarly to the previous application, the ball is 8 pixel diameter and the center of ball is taken as reference for locating the ball.

3. The input variables are : v- the velocity of the ball in m/s, theta- the incidence angle, and strength- which represents the coefficient of restitution. The coefficient of restitution is a parameter of a ball/surface, and reflects the fraction of velocity just after the bounce divided by the velocity just before. For a golf ball, this can be as high as 0.8; for a tennis ball it is around 0.75. The velocity is then multiplied by the value of strength, after each rebound. After each rebound, a new event can take place where you could reinitialized the position x0,y0 and set the time to 0 (the incidence angle stays the same) The equation for the x position is similar to the previous application, but the equation for y must now include the gravitational

force g:

$$y = y_0 + v\sin(\theta)t - \frac{g}{2}t^2$$

we will be using the earth gravity g = 9.8m/s^2.

4. As in task 3, use a time step of 0.1s (update to t_total). You need to keep track of the total time as well. The simulation will stop if the velocity decreases below 0.01m/s or if the ball reaches the right side. You also keep track of the trajectory by plotting it.

5. For the animation we will be waiting 0.02s (using time.sleep(0.02)) before updating the tkinter window. At the end of the simulation the ball turns red, the number of rebounds are displayed as well as the total real time.