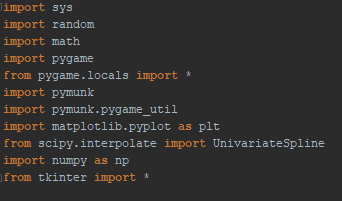
**Explanation**

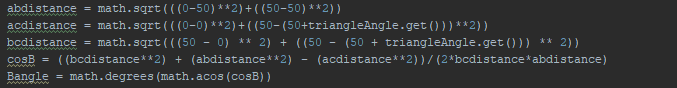


We imported several different libraries in order to perform calculations as well as to be able to help the simulation run smoothly. The random library is to implement the following lines of code:



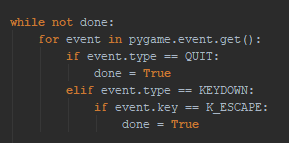
In the x and y variables we added a random range for the x and y variables to become so that when the simulation is run and the particles will be placed over a range of an area rather than placed in one area that would cause the simulation to run slowly.

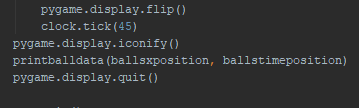
Next, the math library is used to implement the distance formula, and use the cosine function so that when the program is run and we are at the screen to set up the simulation we are able to decide what angles we want the triangles in the hourglass to become as seen in the following code:



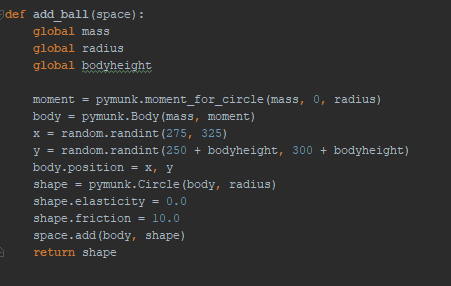
We first find the distance between the known points of the angle and then we use the user’s value to calculate the distance between the remaining points, thus allowing us to find the cosine of the angle using the formula and transposing it to become and finally finding the inverse Cosine of C to find the angle.

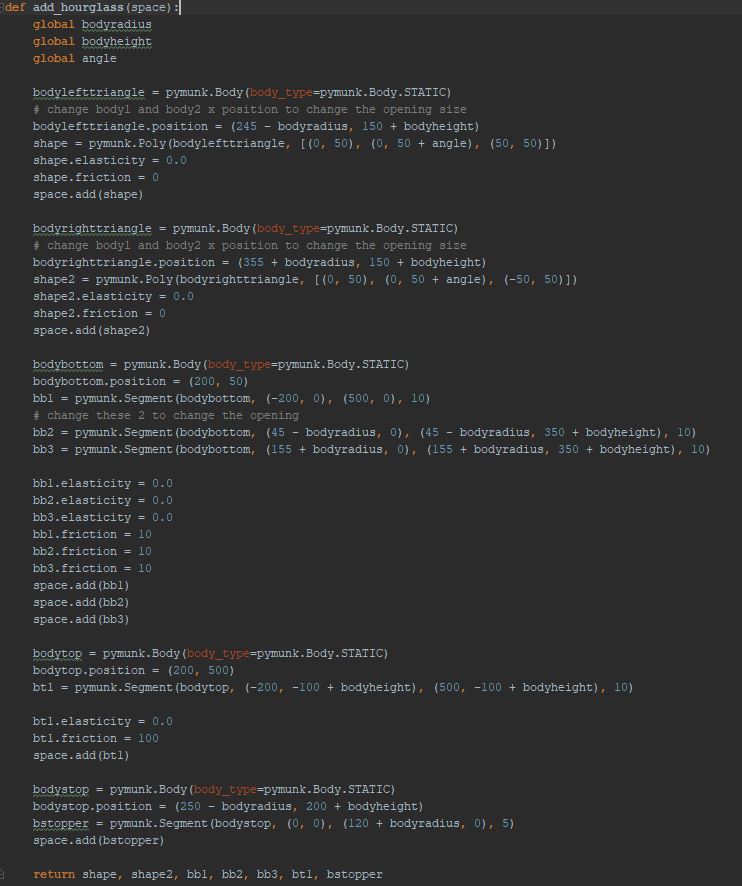
For the Pygame library, it is a necessary import when using the Pymunk library since Pymunk is built on top of Pymunk, meaning that it can use its functions as well as its setup to be able to run a window that the simulation can then be displayed in. We run the main program loop with Pygame, but the physics specific library Pymunk is what we use to implement the simulation and its resources, such as the particles and the shape of the hourglass as well as add gravity and so forth.





As seen in the above example we use the Pygame library to read the keystrokes done by the user to signal the end of the simulation as well as to update the window the simulation is in and to do other controls for the window such as turn it into an icon or quit all together.





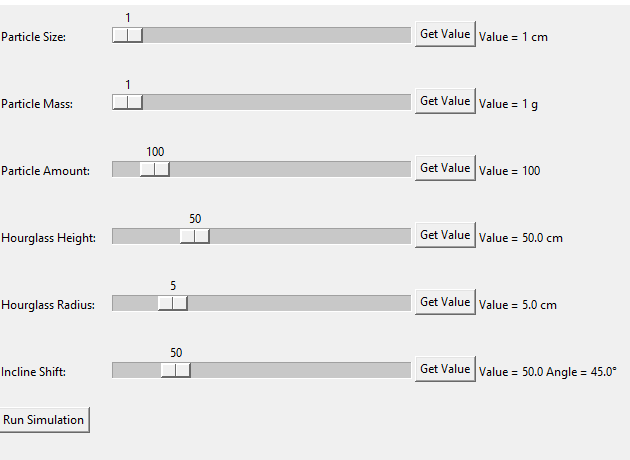
Pymunk, however, is what does most of the heavy lifting since with this unique library we are able to set the elasticity of the particles (which has been set to 0 meaning non-elastic) as well as the center of mass of the particle (which is the center of the render of the particle) and the shape (which is a circle since we can most represent the likeness of a sand particle to a circle). In the rendering of the hourglass itself, we set the position of the hourglass relative the screen and we make the shapes static, meaning that it will not move, and is only there to guide the particles to the bottom of the hourglass, just like a real hourglass.

The next three libraries, matplotlib.pyplot, scipy.interpolate, and numpy are used in conjunction in the following code to analyze the data given by the simulation and graph it accordingly and then finding the derivatives of the data in relation to time twice, before finally showing all of the graphs together.



As we can see we first input the necessary constants based on the user’s request of the simulation such as the total mass of the balls and gravity. Then we convert the units of cm to meters and add the arrays ballx and balltime to a numpy array to that it can be easily traversed by the program. Then we use the UnivariateSpline function to connect the points together so that we can then find the first, and the second derivative of the data. We then use the y-axis data which would now be and then use the following equation, to find the force at each point in the graph. Next we plot all the graphs and set the axis labels to their appropriate items.

The final library, Tkinter, is used to create a GUI interface that the user can access in order set all the variables to specific values so that they can see difference situations of the hourglass as the particles inside are falling.



The variables that can be changed are the, particle size, mass, and amount, as well as the Hourglass’s height, radius, and incline of the funnel in the middle of the hourglass. We can use the slider to change to any of the desired variables and then press the *Get Value* button to confirm our selection. The *Get Value* button can be changed as many times as possible before the simulation is run. Next the *Run Simulation* button begins the simulation with the user’s desired values and shows the user how the particles would fall and how the graphs would look based on the current variables. The user can then open the main window above and change the variables in order to run the simulation again and see another set of graphs unique to that situation. If the user wishes they can end the simulation at any time in order to see the graphs that would be displayed up to that point such as right before the first particle lands on the bottom of the hourglass, or while it is in mid-fall or a variety of other situations.

