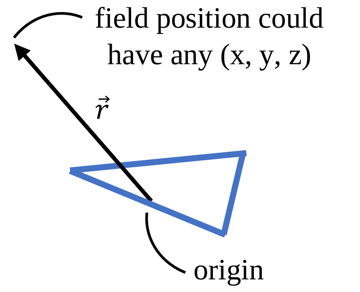
The deliverable you will submit for these assignments is a single Python file uploaded to Canvas. Use wide comments and spaces to clearly mark the start of each problem in the Python file and in your outputs (print statements). It should be possible to execute a single program containing all problems.

All outputs of your code will be assessed for accuracy, but for some of the problems I will also grade the code itself according to the "Code Review" guidelines in the syllabus. Therefore, you should be sure to include descriptive comments and print statements so that the user understands the meaning of each input and output.

**Problem 1**

Consider an equilateral triangle in the xy-plane with charge uniformly distributed along its perimeter ($\lambda$ = charge/length = constant). Each side of the triangle is 0.2 meters long. One side of the triangle is along the x-axis, and there is a vertex on the positive y-axis. The total charge is $Q = -10^{-9}$ C. Approximate the triangle's charge distribution with a sequence of point charges spaced equally along each side (minimum 3 charges, one at each corner). Your goal is to find the total electric field (at an arbitrary position) caused by the entire triangle. Write a function where the arguments are the number of point charges per side, N (so there are 3N total charges), and the x, y, and z coordinates of the (unprimed) position where the electric field is calculated. Have the function return the total electric field vector (with x, y, and z components). Use $k_e = 8.99\times10^9$ N*m$^2$/C$^2$. Print the function's output for any set of arguments.



triangle of charge in xy-plane

Hint: There are some simple cases to check, like the field at the center of the triangle has an obvious value (but finding the position of the triangle's center requires some effort). Also, the field directly above or below the triangle's center has a clear direction that follows from symmetry. Review Coding Exercise Set 3 for a reminder about how to translate Coulomb's Law (involving numerous source charges) into language easily transferrable to a computer.

**Problem 2**

Complete Exercise 3.8 from the textbook. Use the "pylab" package (maybe with: *from pylab import ** ) for all graphs. Make sure your program "shows" all appropriate plots, and print your calculated value for Planck's constant at the end.

**Problem 3**

Next we will use the "vpython" 3D graphics package to visualize interesting physics scenarios. The "vpython" package is imported like any other package (you may have to Google how to install vpython):

*from vpython import **

Use the following website as a reference for generating graphics objects with different properties: https://www.glowscript.org/docs/VPythonDocs/primitives.html

Try the following vpython command to create a red sphere in the graphics window:

*sphere(pos = vector(0,0,0), color = color.red, radius = 0.5)*

Our physics objective in problem 3 is to build a program using vpython that visualizes where the center of mass is located for a system of objects. These objects will consist of a more massive red sphere surrounded by less massive white spheres. Try out the following program template that demonstrates how to add vpython objects to the graphics window at positions where the user clicks:

```
from vpython import *

scene.range = 5   # sets size of our scene window
print("Click to place a ball")
# Start with just one ball
starterBall = sphere(pos = vector(0,0,0), color = color.red, radius =.5)
starterMass = 1
smallMass = .5   # the added balls will be lighter than the starter ball
# Allow the user to create new vpython objects
while True:
   scene.waitfor("click")   # waits for user to click in the scene window
   ball = sphere(pos = scene.mouse.pos, size = 0.4*vector(1,1,1))   # creates a new ball at mouseclick
location
   print("New ball located at: [", ball.pos.x, ", ", ball.pos.y, ", ", ball.pos.z, "] meters")
```

Modify the above template to accomplish the following goals:

a) Before anything else occurs, prompt the user to input a maximum number of additional objects. Then modify the while loop so that it terminates when the user's specification is satisfied.
b) Use the positions of the mouseclicks and masses of each object to calculate and print out the center of mass position. The print statements in the template demonstrate how to find the mouseclick position vectors.
c) Generate a green cube at the center of mass once its position has been calculated. Make sure the green cube is big enough to see, but not so big that it overlaps other objects.