Lab 8

1. Write some code to plot the function $y(t) = e^{-t^2/5}$ from $t = 0$ to $t = 20$.[1] Do this twice:
   (a) Once the "old fashioned way" using only built in Python lists
   (b) Again using numpy **without using lists**
      i. Write the values of t and $f(t)$ to a text file

2. This exercise will serve as an example to familiarize you with numpy and with reading data files. It is organized as a step-by-step tutorial.

   To start, download the file named "radioactive_decay.txt" from Google Drive. This file contains (artificial) data which show the radioactive decay of a certain element as a function of time. In particular, each line of the file specifies the elapsed time (in years) and the amount of material remaining (in kg). From this information, you will estimate the half-life of the element (the time it takes for 50% of the material to decay.) You don't need to submit separate answers for each part, just submit the final product.

   (a) Write code to read the lines of the file (do not use numpy for file reading, do it yourself!). The end result should be two lists of floating point numbers (one containing time, another mass)
   (b) To get an idea of what the data looks like, plot mass vs time
   (c) We now wish to find the time value for which the mass is most nearly half of its starting value (the half life). This is very easy using numpy.
   (d) First, convert both lists to numpy arrays
   (e) We want to find the array index of the mass array for which the mass is closest to mass[0]/2. To do this: create a new array which is the absolute value of mass - mass[0]/2 (use np.abs).
   (f) We now want to find the array index for which this new array is at its minimum. Numpy arrays have a method which does this for you: argmin(). This index also corresponds to the value in the time array for which the absolute value of mass - mass[0]/2 is minimal. So you can use this index to print the corresponding time value, and you have your answer.

[1] I didn't specify Δt or the number n of points. You get to decide. There is a tradeoff here: you want Δt to be small enough that the curve is smooth and individual points are hard to notice. However, note that the number of loops needed is proportional to $1/\Delta t$, so you don't want Δt to be so small that your program wastes time on an unnecessary number of computations. If you are unsure, play around with different values until you find a good compromise.