Follow-up to Computer Exercise on Thursday September 4 & Hand-In Assignment

I. COMMENTS ON PRELIMINARY EXERCISE

Here is my completed version of the Table from today's exercise:

t	$y(t)$	$v_y(t)$	$(F_{\text{net}})_y/m$
0.0	4.0	0.0	-9.8
0.1	4.0	-0.98	-9.8
0.2	3.902	-1.96	-9.8
0.3	3.882	-2.94	-9.8

As I walked around the room I noticed that some of you had $y(0.1) = 3.902$ compared to my value of $y(0.1) = 0$. The formula that I suggested that you use for updating the position was

$$y_2 \simeq y_1 + v_1 \Delta t. \tag{1}$$

In this case, $y_2 = y(0.1)$, $y_1 = y(0)$, and $v_1 = v(0)$, so

$$\begin{aligned} y(0.1) &= y(0) + v(0)\Delta t \\ &= 4.0 + 0 \times 0.1 \\ &= 4.0 \end{aligned} \tag{2}$$

Some of you updated the velocity first, and then used the slight variant of Eq. (1):

$$y_2 \simeq y_1 + v_2 \Delta t, \tag{3}$$

giving

$$\begin{aligned} y(0.1) &= y(0) + v(0.1)\Delta t \\ &= 4.0 - 0.98 \times 0.1 \\ &= 3.902 \end{aligned} \tag{4}$$

Which formula is "right" for updating position, Eq. (1) or Eq. (3)? Eq. (1) gives you an approximation for the change in position of the ball that is too small, and Eq. (3) gives you an approximation for the change in position that is too large. But both of them give you better and better approximations as Δt is made smaller and smaller, and in one sense they have the same accuracy. For this course let's all agree to use Eq. (1); this will ensure that all of our programs use the same technique, and it will make comparison of programs easier. **Terminology Note:** The update algorithm given by Eq. (1) is known as the Euler step.

II. PROGRAMMING NOTES

- **The size of the time step (`dt`) matters!** Remember, the equations we are using to update the position and velocity are approximations, and the approximations are only good in the limit of a small time step. In the Preliminary Exercise I had you use a time step of $\Delta t = 0.1\,\mathrm{s}$; this is actually too big to give realistic results; you should be using time steps that are *much* smaller. (I chose it so that it would be easy to check the values in the table.)

- Remember to save your programs with a `.py` extension. This accomplishes 3 things: 1) IDLE can then find your program. (When you Open a file from IDLE, it only looks for files with a `.py` extension. You can force it to look for other files, but it's not the default behavior), 2) files with a `.py` extension that are opened with IDLE have helpful color coding. If the file doesn't have a `.py` extension, IDLE doesn't know it's a Python program, so it doesn't automatically color code things. 3) there are some ways to run Python programs (other than IDLE) that absolutely require the programs to have this extension. Using the standard extension guarantees that they will work in these environments.

- The Python/VPython combination knows about vectors. That means that you have to use the same care inside your programs that you would in any other calculations with vectors. If you define simple numbers like `a = 2` and `b = -2`, it makes perfect sense to have an statement like

  ```
  if a>b:
  ```

  but if you define two vectors like `c = vector(1,-1,0)` and `c = vector(-1,1,0)`, it makes absolutely no sense to have a statement like

  ```
  if c>d:
  ```

You can compare components of vectors in an `if` statement, or you can compare magnitudes of two vectors, comparing vectors themselves in this way just doesn't work.

- You should take advantage of the fact that Python/VPython knows about vectors, especially when you start dealing in motion in more than one dimension. Here's the long way to update the three dimensional motion of a ball:

  ```
  ball.position.x = ball.position.x + ball.velocity.x*dt
  ball.position.y = ball.position.y + ball.velocity.y*dt
  ball.position.z = ball.position.z + ball.velocity.z*dt
  ```

  and here's the short way:

  ```
  ball.position = ball.position + ball.velocity*dt
  ```

  The "short way" is equivalent to the expression (with vector signs)

  $$\vec{r} = \vec{r} + \vec{a}\Delta t, \tag{5}$$

  and the "long way" is equivalent to writing out one equation for each of the three components of the vector equation.

III. HAND-IN ASSIGNMENT

Each of you are to hand in a Python program simulating a bouncing ball. This assignment is due next Tuesday (September 9) at 5:00 pm. You may hand it in by putting in my Netspace Drop Box or by sending it me as an attachment to an email. Please include your name and the word "ball" in the name of the program, e.g., something like `ligare_ball.py`. I should be able to run your program in the form in which I receive it.

Some of you have working programs already, while some of you are not there yet. I have a minimum expectation for this assignment outlined below that everybody should be able to complete without too much trouble, but I hope that those of you for whom this has been relatively straightforward will take this a little bit beyond what we were able to accomplish in class today. (I would be happy to consult on extensions to the program.) For those of you who haven't made it quite so far yet, I will be scheduling some computer help sessions; we'll pick times for these sessions in class on Friday.

The *minimum* expectation is that your program will show a ball that is dropped from rest from an chosen initial height above the floor and bounces up from the floor. Specific requirements include the following:

- the acceleration and deceleration should be obvious in your animation,
- the ball should bounce repeatedly without significant change in height (unless you are specifically putting in a loss mechanism),
- a `print` statement that enables you to see the values of time, position, and velocity each time through the loop (this line can be "commented out" with a # symbol for normal operation)
- Comments that identify the author of the program, explain the operation of the program, and explain the meaning of any defined variables.

If you have a simple program and a more complex program you may want to submit both.