How to make decent simulations of motion:
The Euler-Cromer method is a variation on the Euler method of numerical integration used in calc classes.
Do the following:

- Define values of constants like g or k (spring constant)
- Specify initial conditions (initial mass, position, velocity, etc)
- Specify a tiny time interval Δt. This time interval is supposed to be small enough such that acceleration, force, and velocity are approximately constant for that tiny amount of time. In practice you make an initial guess for Δt and refine your guess based on how the program runs. More on Δt after this bulleted list.
- Create an iterative loop that does the following:
    - Calculate net force vector based on *current* position
    - Update the momentum of the system using $\vec{p}_f = \vec{p}_i + \vec{F}_{net}\Delta t$
    - Compute current avg velocity using $\vec{v}_{avg} \approx \frac{\vec{p}_f}{m}$
    - Update the position using $\vec{r}_f = \vec{r}_i + \vec{v}_{avg}\Delta t$

More on Δt: No simulation is ever perfect. Often, but not always, you can use a smaller Δt to get higher quality *approximate* answers. Unfortunately, if you make Δt *too* small, the computer is unable to perform computations fast enough to display the simulation at real speed. Generally, we want Δt as large as possible while still making the simulation smooth and accurate. For beginners: run your code then run it again using $\Delta t' = \frac{\Delta t}{2}$. If the simulation seems unchanged (or the outputs are still the same to three sig figs) your step size is *probably* good.

Ready to code? Might as well dive in…
Go to http://vpython.org/ and read the page where it says "How to Get Started".

Following the instructions, I went to glowscript.org. I clicked sign in and it let me sign in with google id. I don't particularly like having to use a google Id but I figured I wanted this done so I did it. I understand if you are morally opposed. Once I got signed in I first wrote the cheesiest program ever:

```
box()
```

I then ran the code. Once you hit run, try swiveling the point of view by holding down the right mouse button and moving the mouse. Try zooming by holding both mouse buttons and moving the mouse. Try changing the size of the black window by dragging on the bottom right corner.

Once you feel good, go on to the next page. Note: you might also click on the HELP link in the upper right corner of the glowscript page. This should link you to a page with all kinds of documentation on different types of objects. There are also some links to training videos and a tutorial if you scroll down a bit. Sometimes the documentation is not quite perfect but it is free…

From a book called Matter and Interaction more video tutorials can be found here:
www.Vpython.org/video01.html
www.Vpython.org/video02.html
www.Vpython.org/video03.html
www.vpython.org/video04.html
I started watching the first vid and it seemed pretty instructive and easy to follow.

Now try writing the following code. **Pay close attention: capitalization, punctuation, and indentation mistakes can cause the code to fail.** *Sometimes* capitalization or indentation doesn't matter, but for now, copy stuff *exactly* as shown.

```
from visual import *
ball=sphere(pos=vector(1, 2, 3),
            color=color.orange,
            radius=0.5)
velocity=vector(1, 3, 1)
arrow(pos=ball.pos,
        axis=velocity,
        color=color.green)
deltat=0.1
t=0
while t<80:
    rate(10)

    ball.pos=ball.pos+velocity*deltat
    t=t+deltat
```

Run it and see what happens. Rather than cut and paste, actually type in the code line by line. You might make a few typos and finding those typos will ultimately be valuable practice. What out for indentation! When I first typed it up, I had a heck of time before realizing that indentation seems to matter…a lot!

Once your code runs, try playing with the numbers one at a time to see what happens. When you've got this down, go to the next page.

Please read before plunging into code on next page…

I personally believe you have a bit more fun coding if your codes actually look good. To that end, I typed up a code which uses a ton of tricks. Most of the code is heavily commented in the hopes you read my comments while typing to learn something. Feel free make your comments much more brief. Do include enough detail in your comments such that you can actually understand them after being away from the code for a few months.

The list below shows the stuff I hope you will learn. If you look carefully, these things are for making your sims look good (and not physics). I thought that you could type them in to learn the pieces. For all subsequent codes, I'm hoping you could cut and paste whatever parts you want then fix up to match whatever new code you are making. Fingers crossed this is not a waste of your time.

If you type it all in I hope you will learn how to do the following:

- Make the computer to wait for you to click before running the code
- Change the background to white
- Make the computer to draw the window the size you want
- Choose the location of the camera, the direction it looks, and the range of stuff it sees
- Put a title on the window
- Put a caption below the window
- Use a print statement to output data from the code
- Create slo-mo option for your code
- Define a new attribute for an object on the fly
- Create a plot
- Label a plot (and correctly use italics!)
- Learn the shorthand trick "+=" for incrementing values

When you finish the code (on the next two pages) and are ready for more, ask me about doing a projectile simulation. I can show you a code I wrote up as a goal and give you some tips to get started.

TranslationalKinematicsWithPlot

```
GlowScript 2.7 VPython
from visual import *
scene = canvas(title = 'Translational Kinematics',
               width = 800,
               height = 400,
               center = vec(0, 0, 0),       #camera LOOKS AT this point
               background = vec(1, 1, 1))   #makes background white

scene.camera.pos = vec(0, 0, 10)            #camera LOCATED AT this point
scene.range = 10    #views points 10 units to the right or left of scene.center

#Create a caption below the main screen
s = "Click to begin the simulation.\n"
s += "Oh yeah...adding a second line to the caption.\n"
scene.caption = s

#define a ball's initial position, initial velocity & acceleration
init_posit = vec(-10, 0, 0)
init_veloc = vec(5, 0, 0)
acceleration = vec(-1, 0, 0)

#print the simulation parameters on the screen
print('Initial position is '+init_posit+' m.')
print('Initial velocity is '+init_veloc+' m/s.')
print('Simulation uses constant acceleration of '+acceleration+' m/s^2.')

#create a simulation speed scalar
#a value of 1 will play the sim at real time
#less than 1 gives slo mo
sim_speed = 0.5

#initialize the clock to time t=0
t=0
#create a time step size
#smaller step size means more computations per second and thus a smoother sim
delta_t=0.02

#draw & label a set of coordinate axes
line_x=cylinder(pos=vec(-10, -2, 0),
                axis=vec(20, 0, 0),
                radius=0.05,
                color = vec(0,0,0))
x_axis_label=text(text='x',
                pos=line_x.pos+line_x.axis,
                color=vec(0.8,0,0.7),
                billboard=True,
                emmissive=True)
line_y=cylinder(pos=vec(0, -4, 0),
                axis=vec(0, 8, 0),
                radius=0.05,
                color = vec(0,0,0))
y_axis_label=text(text='y',
                pos=line_y.pos+line_y.axis,
                color=vec(0.8,0,0.7),
                billboard=True,
                emmissive=True)
```

Code continues on next page as well…

```
#draw the ball at the initial position
ball=sphere(pos=init_posit,
            radius=0.5,
            color=color.red,
            make_trail = True,
            trail_type = "points",
            interval = 10,
            retain = 100)
#create a new attribute of the ball called velocity
#this is not necessary, but is good practice for later...trust me on this
ball.veloc=init_veloc

#create a blank graph and label it with a title and axis labels
g_1 = graph(title = '<i>x</i> vs. <i>t</i>',
            xtitle = '<i>t </i>(s)',
            ytitle = '<i>x </i>(m)')

#initialize the color for a curve to be used in the plot
f_1 = gcurve(color=ball.color)

#make program wait until click (mouse down & up at same position) before running
ev = scene.waitfor('click')

#create a loop that runs anytime the ball is in the specified scene.range
#in my code, this means the loop keeps running as long as the ball's
#x-coordiante is between -10 and +10
while abs(ball.pos.x)<=scene.range:
    rate(sim_speed/delta_t)
    #the above line of code is what allows for slo mo
    #rate tells you how many frames per second the sim will display
    #notice, when sim_speed=1 and delta_t=0.02 we get 50 frames per second
    #displaying about 30 frames or more per second looks smooth to the eye

    ball.veloc+=acceleration*delta_t
    #the above line of code uses some shorthand notation equivalent to
    #ball.veloc = ball.veloc + ACCELERATION*delta_t
    #this is the easiest way to update the ball's VELOCITY

    ball.pos+=ball.veloc*delta_t
    #the above line of code uses some shorthand notation equivalent to
    #ball.pos = ball.pos + VELOCITY*delta_t
    #this is the easiest way to update the ball's POSITION

    f_1.plot(t, ball.pos.x)
    #add a point to the xt-plot

    t+=delta_t
    #the above line of code uses some shorthand notation equivalent to
    #t = t + delta_t
    #this updates the clock time for the next frame of the sim
```