4.6: Using Computation to Simulate Motion in Multiple Dimensions

Let's continue where we left off with computationally simulating one-dimensional motion. We saw that we can write vectors in Visual Python using the syntax.

```
r = vec(x, y, z)
```

where `x, y`, and `z` are the components of the position vector `r`. Of course, we can write a vector of any quantity. For example, position, velocity, or acceleration. Continuing the example of a ball moving from before, try adding a velocity component of $v_y = 0.5$ m/s by editing `ball.vel`. This will create motion in the y-direction that is the same as the x-direction. You may start from the code that ended the previous computational simulation. We will make a small change first. In the definition of the ball, we are going to create a trail of dots that shows the ball's past positions, and we will remove the velocity arrow for the this example.

```
ball = sphere(pos=vec(0,0,0), color=color.green, radius=0.1, make_trail=True, trail_t
ball.vel = vec(0.5, 0, 0)
r_0 = ball.pos
t = 0
dt = 1
while t<10:
    rate(1)
    ball.pos = r_0 + ball.vel * t
    t = t + dt
```

Next, let's look at an example from an earlier section that includes acceleration. The example is shown again below.

Exercise 4.6.1

An object starts at the origin of a coordinate system at time $t = 0s$, with an initial velocity vector $v_0 = (10m/s)\hat{x} + (15m/s)\hat{y}. The acceleration in the \(x$ direction is $0m/s^2$ and the acceleration in the y direction is $-10m/s^2$.

1. Write an equation for the position vector as a function of time.
2. Determine the position of the object at $t = 10s$.
3. Plot the trajectory of the object for the first $5s$ of motion.

Answer

```
scene2 = canvas(title='Examples of 2D Accelerated Motion',
     width=400, height=400,
     center=vector(5,0,0), background=color.black,
```

```
     range=300)

ball = sphere(pos=vec(0,0,0), color=color.green, radius=10, make_trail=True, tra
ball.vel = vec(10, 15, 0)
ball.acc = vec(0, -10, 0)
r_0 = ball.pos
t = 0
dt = 1
while t<=10:
  rate(1)
  ball.pos = r_0 + ball.vel * t + 0.5 * ball.acc * t**2
  t = t + dt

print(ball.pos)
```

To solve this we need to do several things to our constant velocity code above. First, we need to edit the initial velocity, `ball.vel`, using what is given in the example. Then, we should define `ball.acc = vec(0, -10, 0)`. Third, we should make the while loop go until less than or equal to 10s by writing `while t <= 10:`. Fourth, inside the loop, we need to edit the ball.pos to include the acceleration. In Python, raising to a power is done by two asterisks, e.g., `t**2` is the square of `t`. The time-dependent position is

$$\vec{r}(t) = \vec{r}_0 + \vec{v}_0 t + \frac{1}{2}\vec{a}t^2 \tag{4.6.1}$$

will be written in Visual Python as

```
ball.pos = r_0 + ball.vel * t + 0.5 * ball.acc * t**2
```

Finally, we can print the ball's position after the loop finishes with the command `print(ball.pos)`. This print statement should be unindented after the while loop to let Python know that it occurs after the looping is complete. Of course, you are welcome to put the print statement in the loop (indented under t = t + dt) if you want to see what difference this makes. Making these changes we can show the position vs. time of the object and print the position after 10 s have passed. To summarize, the steps are

1. Edit `ball.vel`.
2. Create an acceleration of the ball, `ball.acc`.
3. Make the loop end at 10 s.
4. Edit `ball.pos` in the loop to include `ball.acc`.
5. Add a print statement to see the ball's position at $t = 10s$.

Below is a trinket that will keep the scene fixed. Copy the constant velocity code above and make these five edits. The ball radius may need to be increased if it is difficult to see. If you get stuck look at the answer to the example.

It's not necessary to simulate the motion if we only want to calculate or graph. We can use Visual Python to calculate the position of an object as we did above. To do this, we simply define the initial position, initial velocity, and acceleration. Then, we can use the kinematic equation to calculate the position at a later time. Try putting this into the trinket above.

```
r0 = vec(0,0,0)
v0 = vec(10, 15, 0)
a = vec(0, -10, 0)

t = 10

rt = r0 + v0*t + 0.5*a*t**2
print(rt)
```

Similarly, we can use Python (not Visual Python) to plot. You'll notice the trinket below says Python3. This distinguishes it from the other trinkets above. It also does not have the first line that imports Visual Python (`Web VPython 3.2`).

To make lists (arrays) of numbers, it is best to use a library called `numpy`. To make graphs also requires importing a library `matplotlib` for making graphs. Using the command `import a as b` makes library `a` available by using `b` to call it.

```
import numpy as np
import matplotlib.pyplot as plt
```

We won't use vectors, but we keep our initial values for $x_0, y_0, v_{x0}, v_{y0}, a_x$, and a_y. Then, we need to make lists of $x(t), y(t)$, and t that correspond to the points we want to graph. These are calculated using $x_0, y_0, v_{x0}, v_{y0}, a_x, a_y$, and the list of times t. Therefore, we must define t before attempting to calculate $x(t)$, and $y(t)$

```
x_0 = 0
y_0 = 0
v_x0= 10
v_y0 = 15
a_x = 0
a_y = -10
t = np.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0])

xt = x_0 + v_x0 * t + 0.5 * a_x * t**2
yt = y_0 + v_y0 * t + 0.5 * a_y * t**2
```

That will create lists of $x(t)$ and $y(t)$ because Python recognizes that t is a list and calculates for each value.

Finally, we insert the code to make a graph.

```
plt.plot(xt, yt, 'or')
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.show()
```

The first line plots `xt` on the x axis and `yt` on the y axis. The `'or'` makes circles for the points that are red. The second line labels the x axis. The third line labels the y axis. The last line puts all the plot code into an image. Try combining these three blocks of code in the trinket below to see that they plot the 2D motion of the object.