

# Chesick Scholars - Intro to computer programming in STEM



Prof. Dan Grin



Deep Patel

*Where*  
H204 - SCIENCE BUILDING

*When*  
Tuesday 8/23 1-2:30 PM  
Wednesday 8/24 11-11:30AM

# Mask please



Extra masks available at front of room

---

# Today's agenda

---

- ❖ Example(s) — falling ball — *your job, add acceleration and motion along table*
- ❖ Example- moving planet — *your job, add the moon or extra-solar planets*
- ❖ Example — simple harmonic motion
- ❖ Mini-project — projectile motion

# Loops, example

[https://matter-interactions.trinket.io/00\\_welcome\\_to\\_vpython#/welcome-to-vpython/loops](https://matter-interactions.trinket.io/00_welcome_to_vpython#/welcome-to-vpython/loops)

```
t = 0  
print("t =", t)
```

```
t = t + 1  
print("t =", t)
```

```
t = t + 1  
print("t =", t)
```



```
t = 0  
while t < 11:  
    print("t=", t)  
    t = t + 2
```

```
print("The loop is finished")
```

dg loop examples

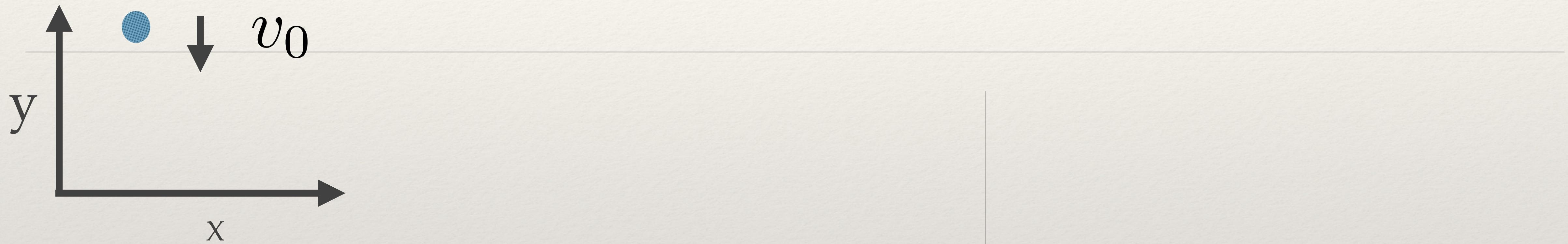
---

# Debugging

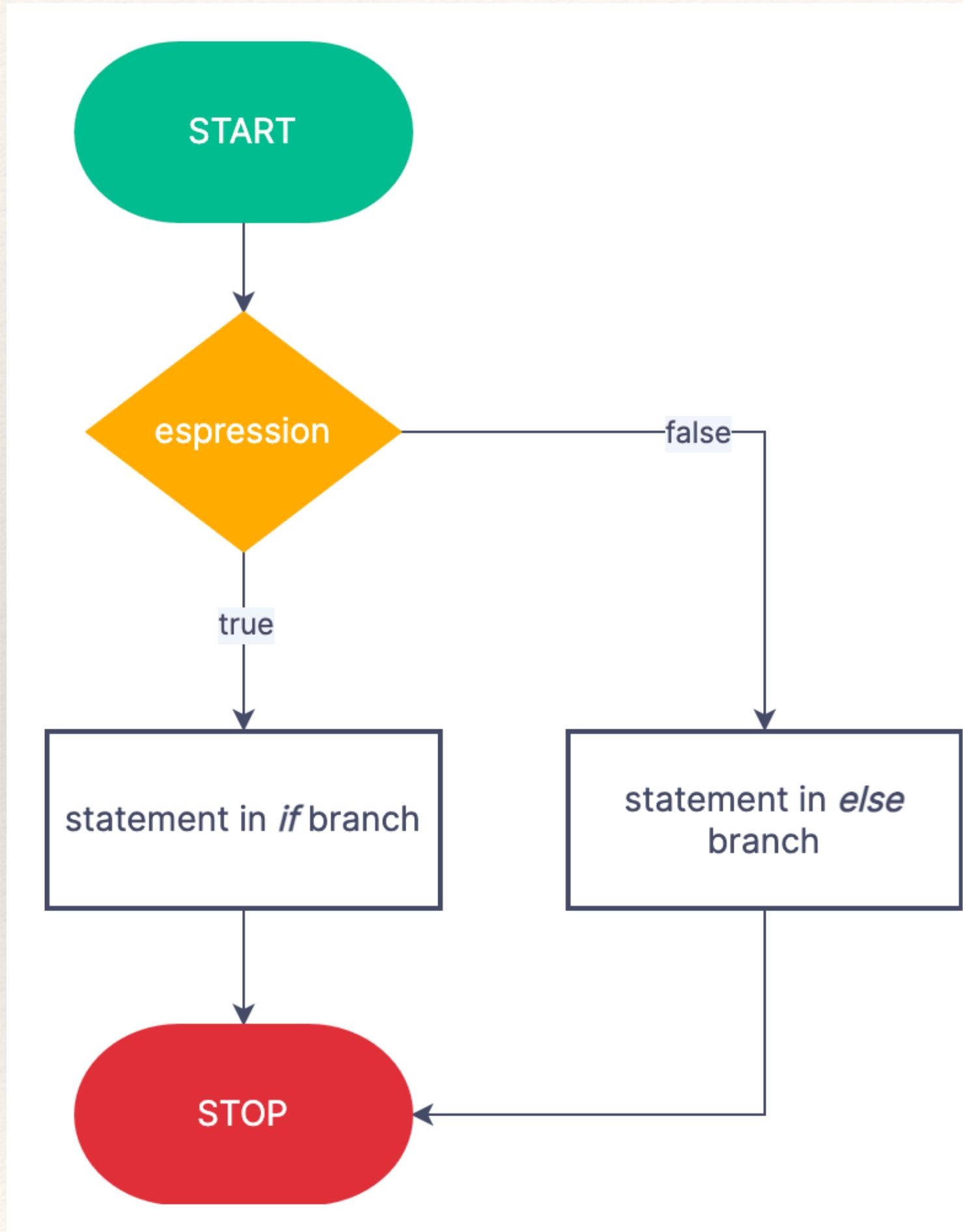
---

# Moving ball example

$$y = y_0 + v_0 t$$



# if/then program flow control



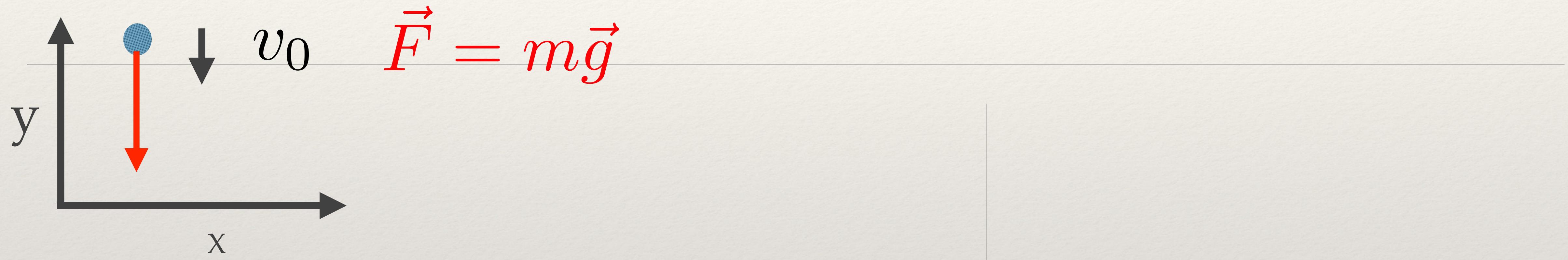
Use to impose table boundary

*Your turn – make the ball roll to the right once it hits table*

# *Falling ball with gravity – your turn! (Work in pairs)*

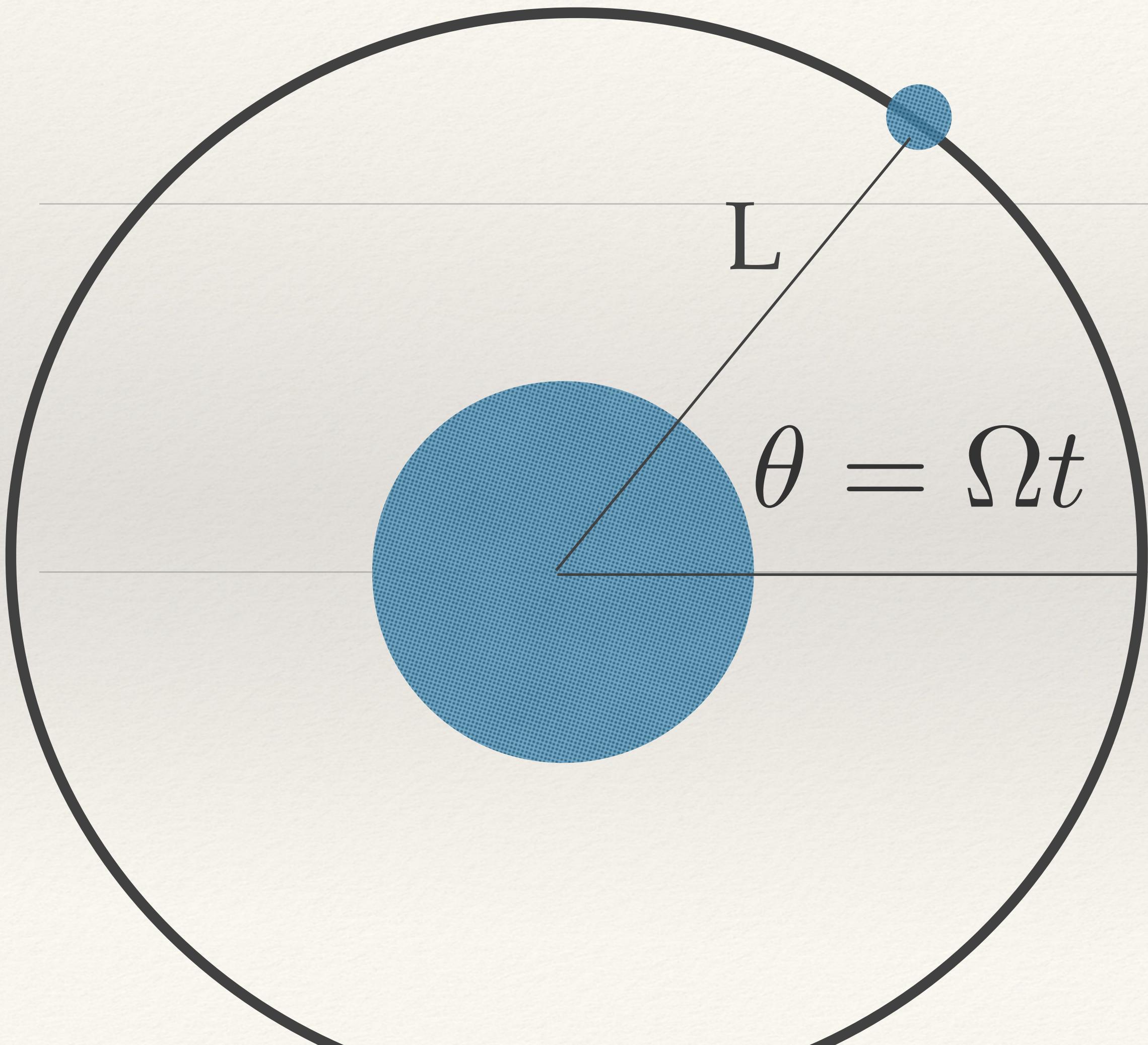
$$y = y_0 + v_0 t - gt^2/2$$

$$v = v_0 - gt$$



*Challenge – add a plot of  $v(t)$*

# Example – orbits

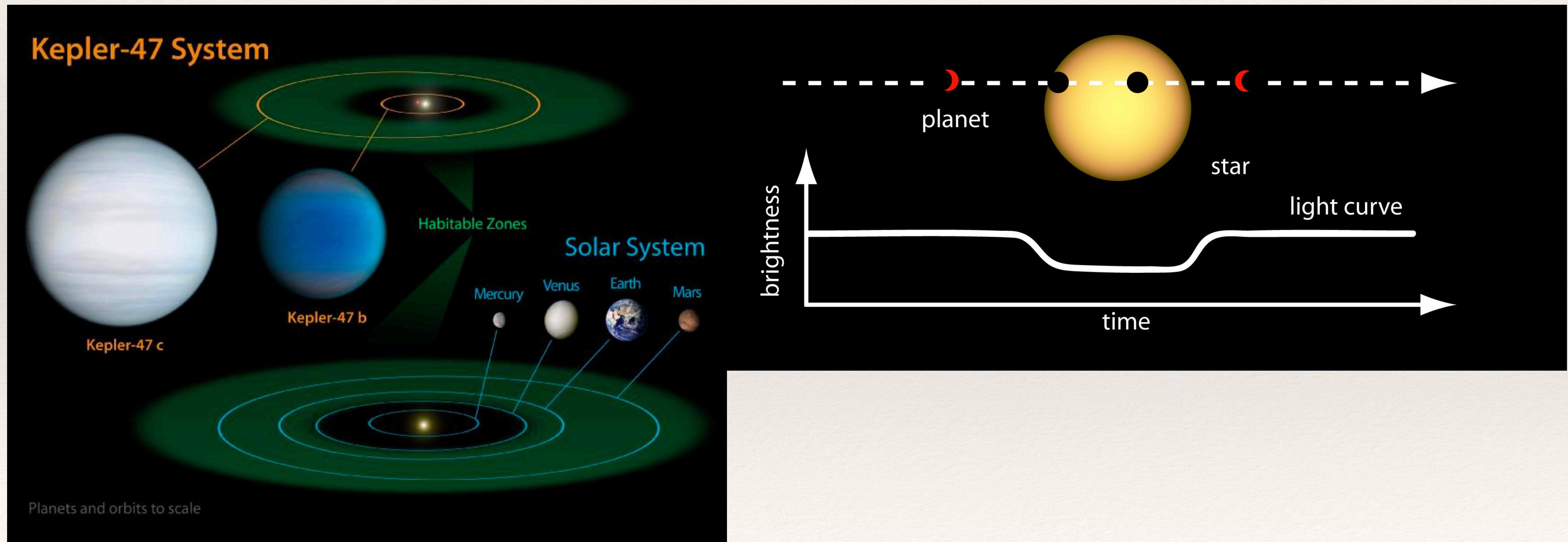


$$x = L \cos (\Omega \times t)$$
$$y = L \sin (\Omega \times t)$$

*Your turn – add the moon*

# *Your turn – extrasolar planets*

<https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=PS>



# Velocity

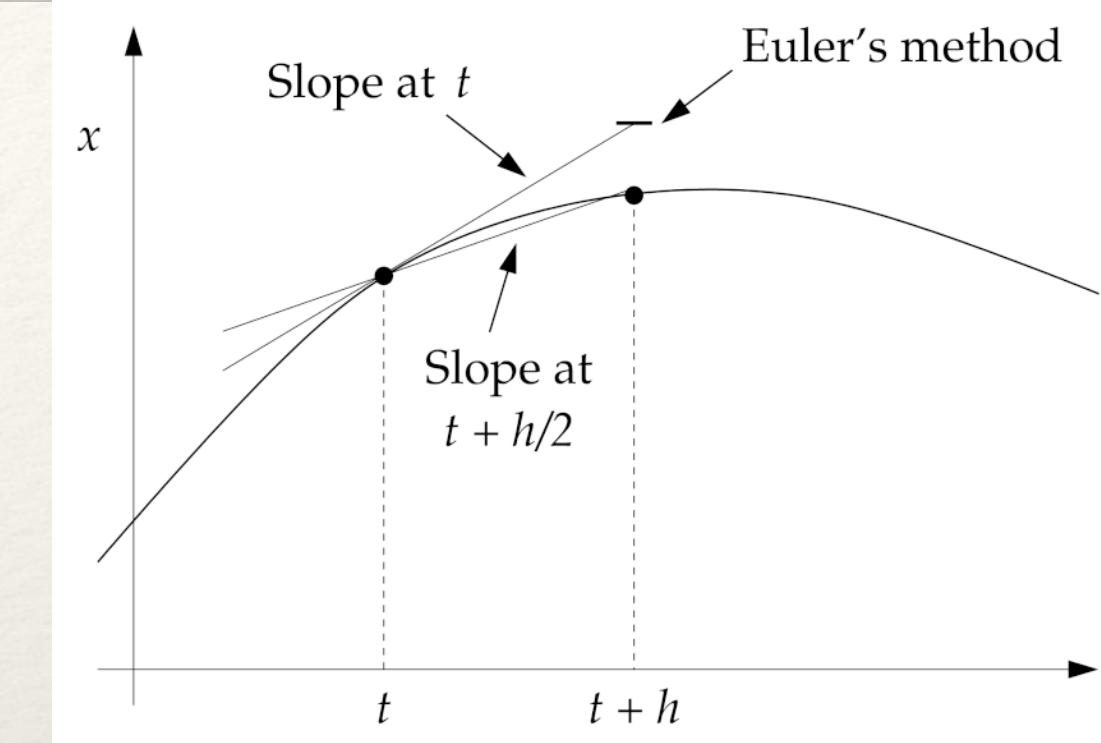
$$v_{avg} = \frac{\Delta x}{\Delta t} = \frac{x_2 - x_1}{\Delta t}$$

$$x = x + v * dt$$

Chop motion into segments/assign new position

Euler's method

$$x_2 = x_1 + v\Delta t,$$

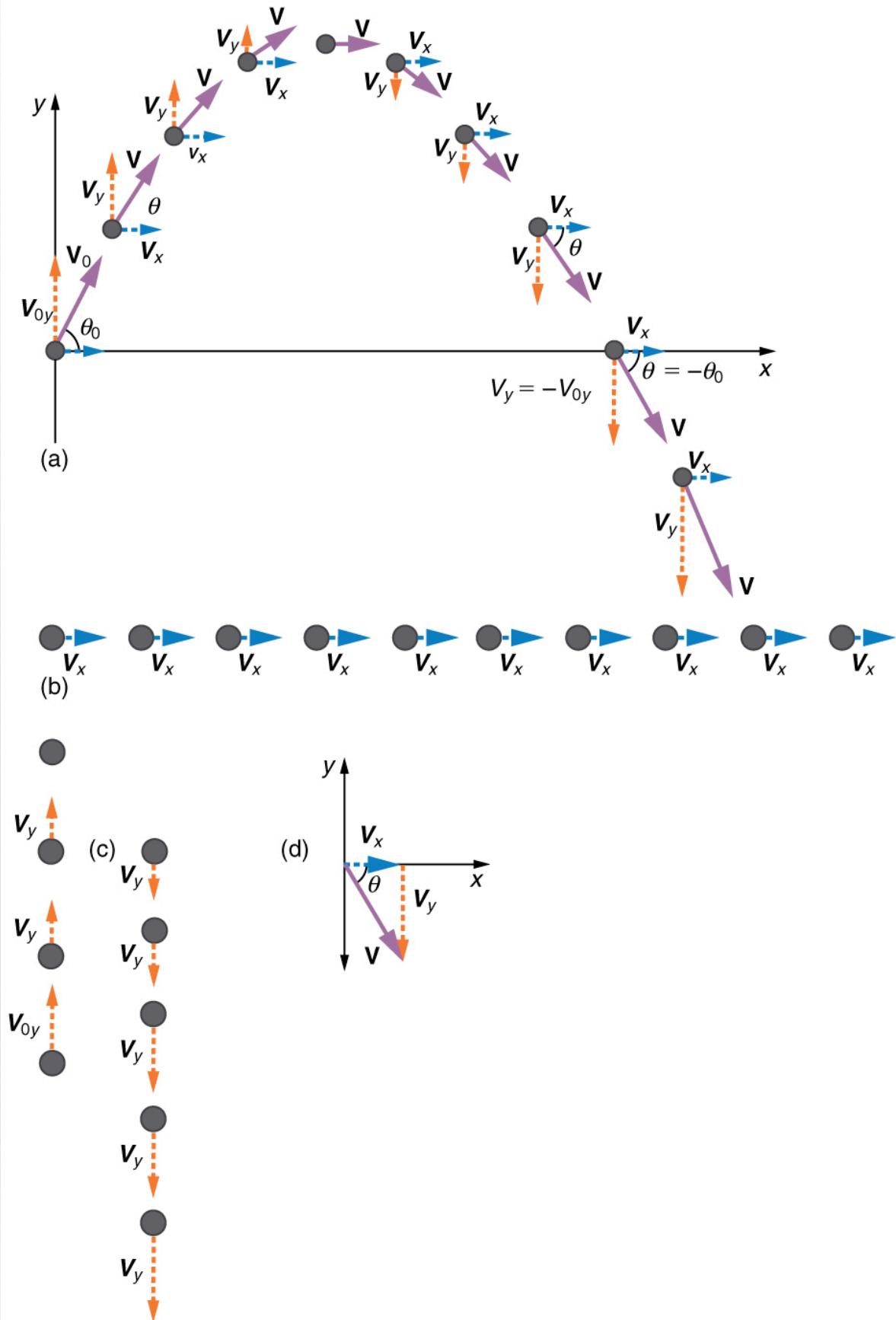


# Animation/integration

```
1 GlowScript 2.7 VPython|
2
3 # Defining the "graph"
4 g1=graph(width=400, height=250)
5 xDots=gdots(color=color.green, graph=g1)
6
7 # Defining the Object
8 obj=sphere(pos=vector(-1,0,0), radius=0.1, color=color.red)
9
10 # Setting initial conditions and step size, dt
11 t=0
12 dt=0.05
13 x=-3.0
14 v=2.0
15
16 # This is main part - the loop
17 while t<3:
18     rate(10)
19     obj.pos=vector(x,0,0)
20     xDots.plot(t,x)
21     # Updating the position
22     x=x+v*dt
23     t=t+dt
```

*Integration rather than exact formula*

# Projectile motion



$$x = x_0 + v_{x0}t + \frac{1}{2}a_xt^2;$$

$$v_x = v_{x0} + a_xt;$$

and

$$v_x^2 = v_{x0}^2 + 2a_x(x - x_0).$$

Your turn (work in pairs, make sure to write pseudo code + lots of comments):

Write a program to simulate this- make the projectile stop when it hits ground - is the range consistent with

$$R = \frac{v_0^2 \sin 2\theta}{g} \quad ?$$

# Last example - a more realistic method

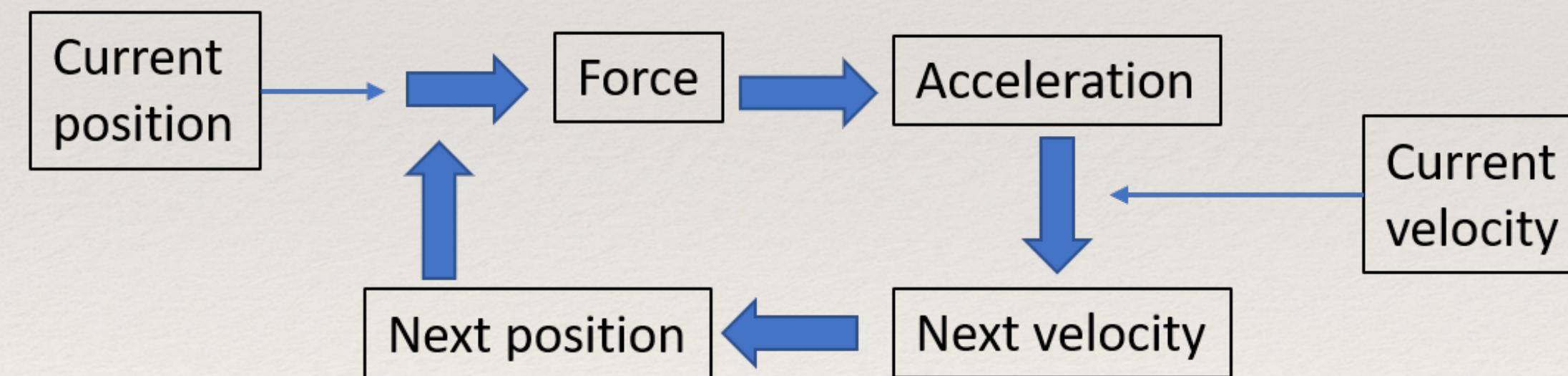
$$x = x + v * dt$$

Assumed constant velocity

$$\vec{F} = m\vec{a} = \frac{\Delta\vec{v}}{\Delta t}$$

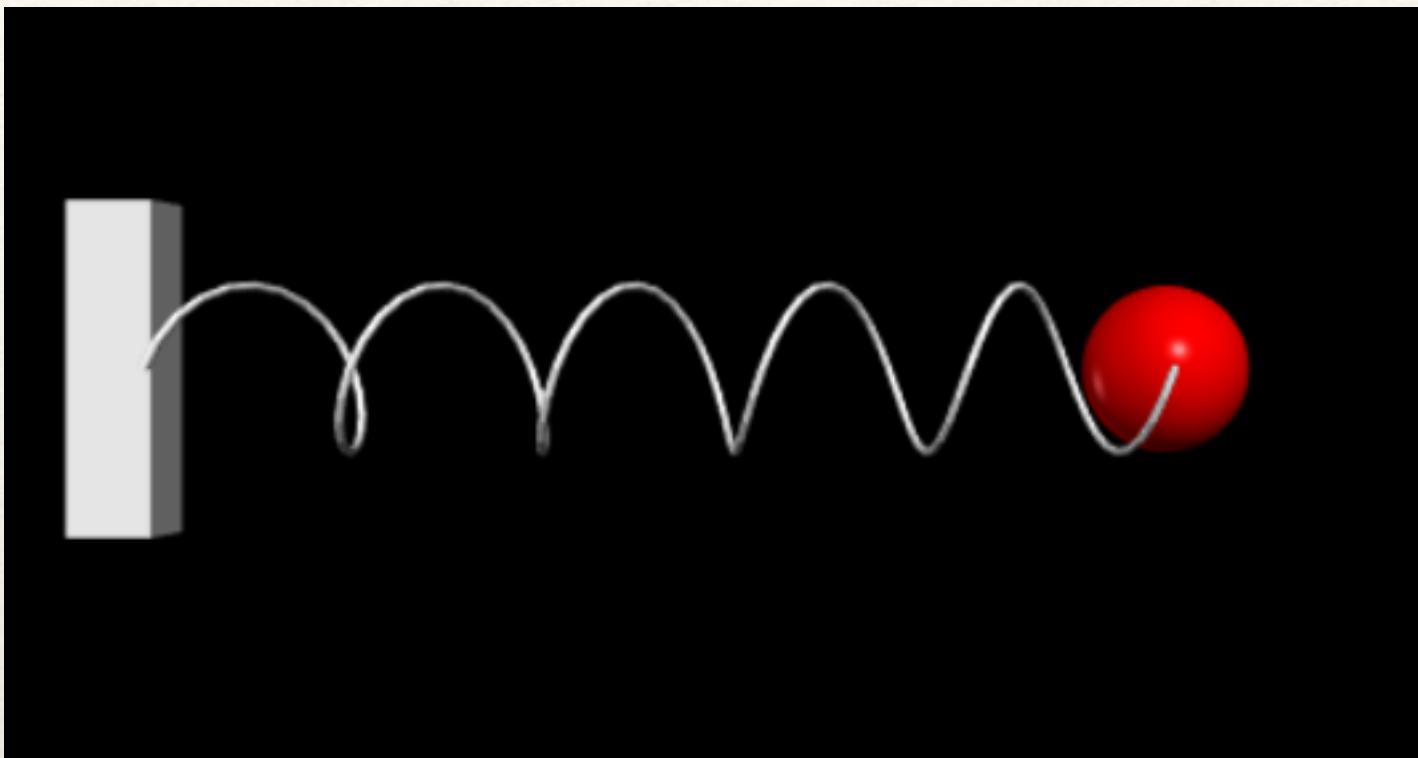
More realistic

$F = -k*x$   
 $a = F/m$   
 $v = v + a*dt$   
 $x = x + v*dt$   
 $t = t + dt$



Euler-Cromer method

# Simple harmonic oscillator



To illustrate the method, let's get our feet wet with a relatively simple problem - a mass on a horizontal spring. All we have to remember is Hooke's law for ideal springs,

$$F(x) = -kx, \quad (6.5)$$

- 1) Try DG example code
- 2) Change the initial conditions of the mass, how does that change the graphs?
- 3) *Change k and m? How does quadrupling k or m change the period of oscillation?*
- 4) *Compare your results with the formula*

$$T = 2\pi \sqrt{\frac{m}{k}}$$

# Your turn - projectile motion with drag.

1) Redo projectile motion with Euler-Cromer method (check agreement with exact formulae)  
work in pairs, make sure to write pseudo code + lots of comments

$$v_x = v_x$$

---


$$v_y = v_y + a_y t$$

$$x = x + v_x t$$

$$y = y + v_y t$$

$$a_y = -g = -9.8 \text{ m s}^{-2}$$

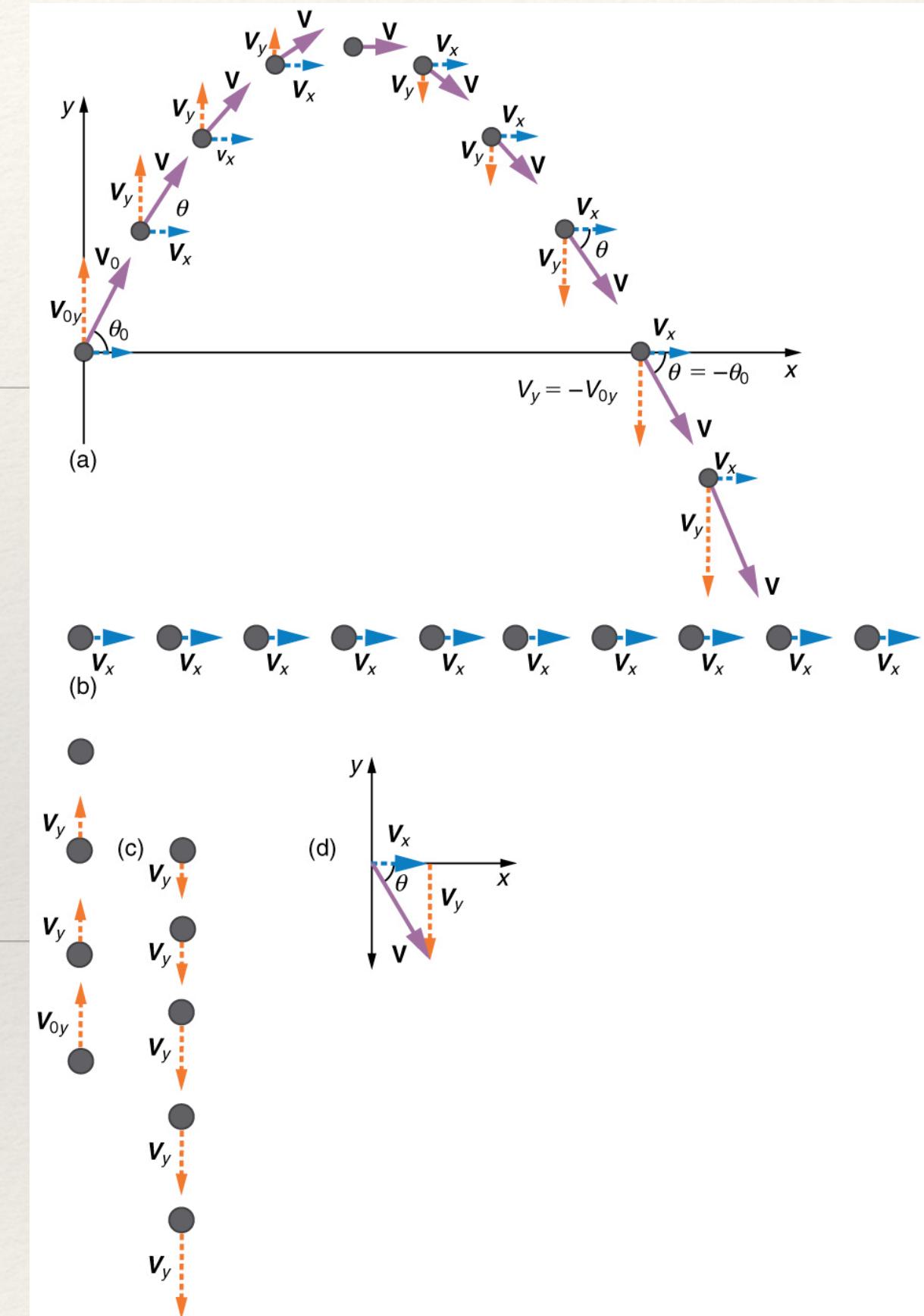

---

2) Add drag force (air resistance)

$$\vec{F}_D = \frac{1}{2} \rho A C_D v^2 (-\hat{v}),$$

$$\hat{v} = \vec{v} / |\vec{v}|.$$

- **mag(A)** and **mag2(A)**, where the output yields the length and length-squared of that vector, respectively.
- **A.hat**, which produces from the vector **A** its unit vector (by dividing it by its length). This syntax treats the directionality (given by the unit vector) as a *property* of the vector. It is just like any other property of a vector, such as **A.x**, which yields the x-component of the vector **A**.



# Projectile motion

---

- ❖ *How does the range quantitatively compare with before?*

---

# Tomorrow – command line ipython, jupyter

---