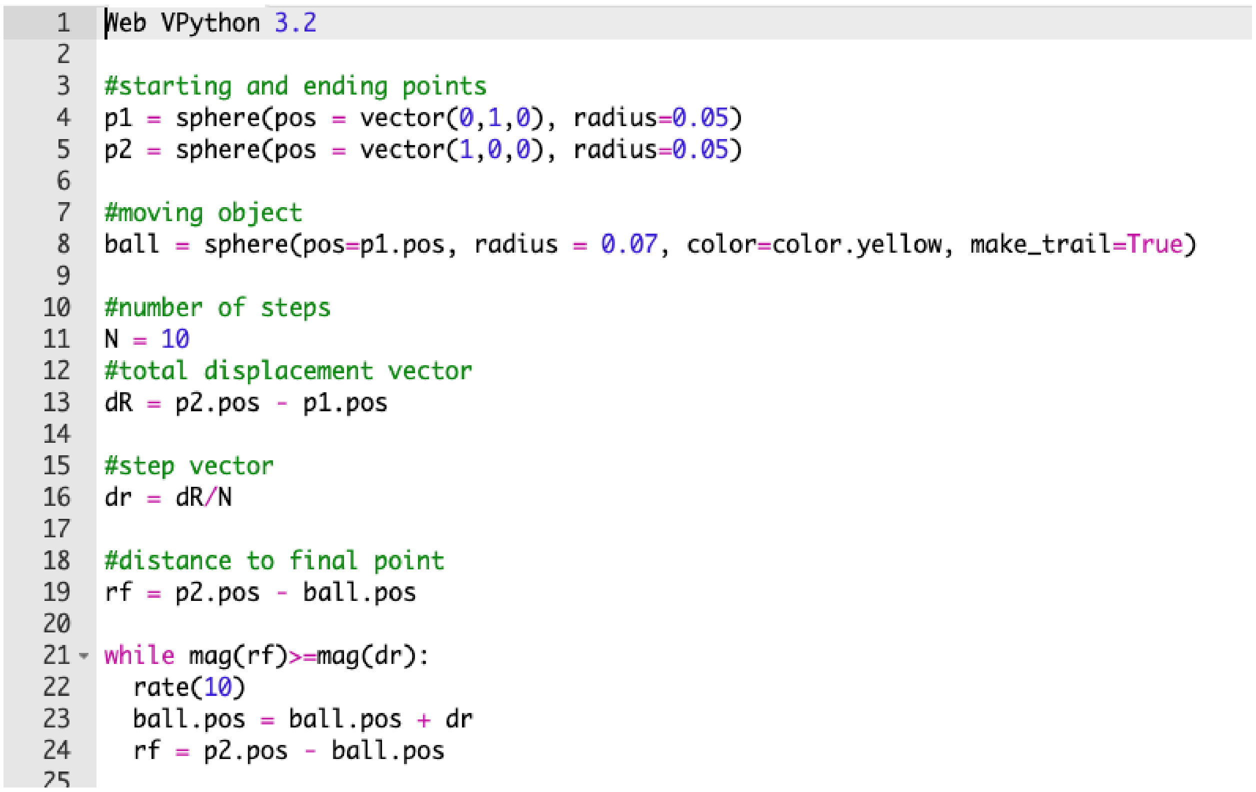
Numerical Line Integrals

We are going to calculate the work done by a force using small (but finite) displacements. During each step, we can approximate the force as constant.

1. Let's start with an object moving near the surface of the Earth. Let $\vec{F} = m\vec{g} = 0\hat{x} - 1\hat{y}$. Make sure you can calculate the work going from $\vec{r}_1 = 0\hat{x} + 1\hat{y}$ to $\vec{r}_2 = 1\hat{x} + 0\hat{y}$.

   Do this analytically. Show your work. Hint: the answer is 1.

2. Now let's do this numerically. There's going to be a bunch of stuff here - so let's build this one part at a time. Here is code that moves from point 1 to point 2 using N steps. See if you can get this to run (it should run). Here's the code followed by comments.



```
Web VPython 3.2

#starting and ending points
p1 = sphere(pos = vector(0,1,0), radius=0.05)
p2 = sphere(pos = vector(1,0,0), radius=0.05)

#moving object
ball = sphere(pos=p1.pos, radius = 0.07, color=color.yellow, make_trail=True)

#number of steps
N = 10
#total displacement vector
dR = p2.pos - p1.pos

#step vector
dr = dR/N

#distance to final point
rf = p2.pos - ball.pos

while mag(rf)>=mag(dr):
  rate(10)
  ball.pos = ball.pos + dr
  rf = p2.pos - ball.pos
```

- We are making a 3D model for this motion. With that in mind, we have the sphere object. In Web VPython, sphere() is a function that creates a 3D sphere. Some of the important properties that you can assign to this object are: pos (for the vector position) and radius: for the size of the sphere. You can also give it a color and turn on the trail (make_trail=True).
- When dealing with objects, we can access different attributes. For instance, if you want to print (or use or change) the vector position of the ball, you can say ball.pos = vector(1,2,3) or something like that.

- In lines 4-5, I create two spheres. P1 is the starting position and p2 is the ending position.
- The object that will move is called “ball” (line 8).
- Line 11 is the number of steps we are going to use to get from p1 to p2.
- dR is the vector displacement from p1 to p2. Notice that I’m using p1.pos and p2.pos for the vector locations of these objects.
- dr is the vector step size.
- Finally, rf is a vector from the ball to the ending location. As the ball moves towards p2, this value will get smaller.
- The loop (lines 21-24) move the ball. The while loop runs as long as the magnitude of rf is greater than the magnitude of the step size. We have to recalculate rf each time the ball moves.

Run the code. It should work. If it doesn’t, let me know.

3. Calculating the work. See if you can add the following to your code.
    - Outside of the loop, add W = 0. This will be the total work. In order to add the the work from each tiny step.
    - Inside the loop, calculate the vector force. In the case of constant gravitational force, it would just be something like F = vector(0,-1,0).
    - Calculate the work during each step. The dot product is built into Web VPython such that dW = dot(F,dr) will return the work.
    - Add this small work to the total work (W = W + dW)

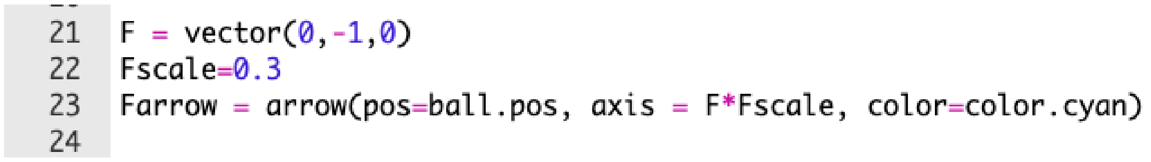
   After the loop, you can print the total work. How does the value compare to your theoretical value?

4. Work around a closed loop. See if you can also calculate the work going from p2 to the origin and then another loop going from the origin back to p1. What is the total work? What happens as you change the value of N?

5. What if we change the force from a constant gravitational force to $\vec{F} = -y\hat{x} + x\hat{y}$. Note: you can get the x-coordinate of the ball as ball.pos.x. The y-coordinate is ball.pos.y (you will need these to calculate the vector force at different locations.

Bonus

If you want something to be cool, you can add an arrow representing the force on the ball. Here’s some code for the constant gravitational force. First, we need to create an arrow object.

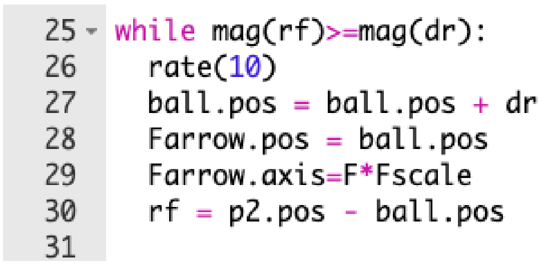


```
F = vector(0,-1,0)
Fscale=0.3
Farrow = arrow(pos=ball.pos, axis = F*Fscale, color=color.cyan)

```

The arrow is an object in Web VPython and we are giving it the name “Farrow”. The two important properties of an arrow are the position–which is the vector location of the start of the arrow. In this case, we are starting the arrow on the ball. The other property is “axis”. This is a vector from the position to the pointy part of the arrow. We want this axis to represent the force, F–however, the magnitude of the force might not be appropriate for our distance scale. To fix this, we multiply by “Fscale”--a scaling factor. I picked Fscale = 0.3, but you can just play with the value to get something that you like.

Of course, we need the arrow to move along with the ball. For that, we can also include the following in the loop.



```
25 while mag(rf)>=mag(dr):
26   rate(10)
27   ball.pos = ball.pos + dr
28   Farrow.pos = ball.pos
29   Farrow.axis=F*Fscale
30   rf = p2.pos - ball.pos
31
```

- In line 28, I update the position of the arrow so that it’s the same as the ball.
- In line 29, I update the axis of the arrow. Yes, in this case the axis doesn’t change but I’m just trying to prepare for the future.