# Motion in one dimension

A.C. Norman
ACN.Norman@radley.org.uk

In this activity, we shall model the kinematics of a ball moving in one dimension (vertically).

## Getting started

1. Create a new program at `http://www.glowscript.org` (maybe named 'SimpleUpThrow').
2. Make a floor using the `floor=box(pos=vec(0,0,0), size=vec(...))` command (fill in the blanks so your 'floor' is thin in the $y$ direction and extended in the $x$ and $z$ directions).
3. Make a ball using `ball=sphere(pos=vec(0,0,0), radius=...)`. You might want to make the floor and the ball different colours using e.g. `color=color.red`.

## Making the ball move

In the absence of acceleration, the ball's motion will be described by the equation

$$\mathbf{s} = \mathbf{u}t, \text{ or for a small time interval } \Delta t,$$
$$\Delta\mathbf{s} = \mathbf{u}\Delta t.$$

To use this equation to update the ball's position, we need to give the ball a velocity $\mathbf{u}$ pointing in the $y$ direction, define a small interval in our program and tell it to start the clock at $t = 0$ with the lines

```
ball.v = vec(0,2,0)
t = 0
dt = 0.01
```

Now let's get the ball to move. For every time-step $\Delta t$, we need to work out how far to move the ball using $\Delta\mathbf{s} = \mathbf{u}\Delta t$, and add this to the old position. In Python, we need a `while` loop to make this happen:

```
while t < 3 :
    rate(20) #slows the while loop down for animation
    ball.pos = ball.pos + ball.v * dt #using velocity to update position
    t = t + dt #this updates the clock time
```

The result of running your program should now be a ball which moves upwards from the floor at a constant rate.

# Adding constant acceleration

For an object moving with constant acceleration, we can use the constant acceleration (*suvat*) equations. In particular,

$$\mathbf{v} = \mathbf{u} + \mathbf{a}t.$$

For a small time interval $\Delta t$, the change in the velocity $\mathbf{u}$ is given by

$$\Delta\mathbf{u} = \mathbf{a}\Delta t.$$

We need to start by defining an acceleration vector. Let's choose to model acceleration under gravity (so our ball, 'thrown' upwards at $t = 0$ ought to eventually come back down to the floor), so this means that our acceleration vector will be

```
ball.a = vec(0,-9.81,0)
```

In our program, we will need to use the acceleration to update the velocity each time the while loop commands are executed by working out the change in velocity and adding it to the old velocity. We can do this with the line

```
    ball.v = ball.v + ball.a * dt
```

At this point you may want to change the `while` loop test so that the program executes until the ball comes back to the floor level ($y = 0$):

```
while ball.pos.y >= 0:
```

# Testing your program

Now do some simple tests of your program. A good way to do this is to use it to solve a problem that you can already work out the answer to.

If a ball is thrown upwards at 3.4 m/s,

1. How long does it take to come back down?
2. What is its maximum height?

You might want to add a `print(...)` statement to output e.g. the time after the `while` loop has finished. Another useful way to get information out of your program is by plotting a graph.

## Graph plotting

Add the line `f1 = series()` near the start of your program. This tells the computer that we are going to plot a graph. However, it doesn't tell it what to plot. We need to add a data point to the graph during each step of the calculation. You can do this by adding a line inside the loop (and thus indented) like this:

```
    f1.plot(t,ball.p.y)
```

From this plot, you could easily find the maximum height.