VPython: Learning about Vectors

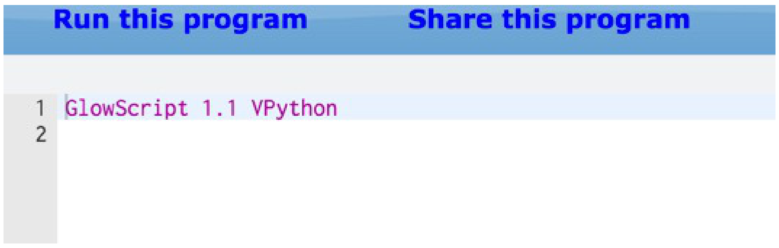
A.C. Norman
ACN.Norman@radley.org.uk

Adapted by ACN from a Glowscript tutorial by Rhett Allain of Southeastern Louisiana University

Introduction

VPython is a programming language that is easy to learn and is well suited to creating 3D interactive models of physical systems.

Go to `http://glowscript.org`. At the upper right of the screen, click **Sign in**. You should be able to use a google account, and By signing in, you can save your programs. Next go to your program space (click on your **username** in top right, or 'your programs are **here**' and click **Create New Program**. Choose a name for your program (maybe 'VectorExercise'). Note that any spaces and underscores will be deleted from the name. You will see a blank edit window:



```
GlowScript 1.1 VPython

```

Getting started

Now we are going to make three balls (ball 1, ball 2, and ball 3). All three balls have a radius of 0.1 meters and the following positions:

	name	position <x,y,z> vector	colour
Ball 1	`ball1`	`<0,1,0>`	red
Ball 2	`ball2`	`<-2,-1,0>`	green
Ball 3	`ball3`	`<1,-1.5,0>`	yellow

Here is the code to make ball 1:

```
ball1=sphere(pos=vec(0,1,0), radius=0.1, color=color.red)
```

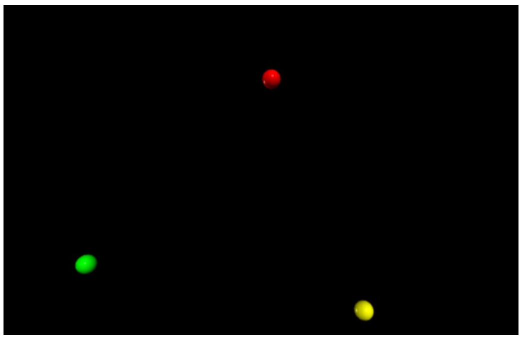
In this line of code you make a sphere and assign it to the name `ball1`. The `sphere()` is a function built into glowscript. You can assign it a position, a radius and colour.

NB In common with most programming languages, colour is spelt color in the program.

Here is the partial code for balls 2 and 3 (you need to fill in some of the blanks):

```
ball2=sphere(pos=vec( , , ), radius=0.1, color=color.green)
ball3=sphere(
```

Click **Run this program**. It should look like this:



This is a 3d display. In the VPython window, hold down the left+right buttons on a two-button mouse (or the Alt key) and move the mouse (or use the scroll wheel). You should see that you are able to zoom into and out of the scene. Now try holding down the right mouse button (or the Ctrl key). You should find that you are able to rotate around the scene (you can also tell that you are moving because the lighting changes). Keep in mind that you are moving a camera around the object. When you zoom and rotate you are moving the camera, not the object.

If your program doesn't look like that, you have made an error somewhere.

Properties of the balls

Once you make a sphere object, you can call back some of the properties of the object. Look at this code.

```
print(ball1.pos)
```

`ball1.pos` is the vector position of the ball1. Using the `print` statement, the value of this will appear in the output window. This is what it looks like when you run it (the text output window will probably be below the 3d display window).

```
< 0, 1, 0 >
```

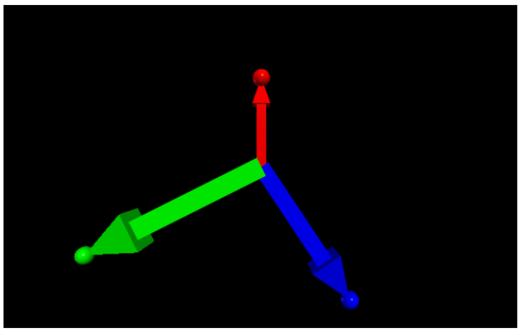
Displaying position vectors

How do we display a vector in the 3d environment? Here is the arrow object in Glowscript.

```
r1=arrow(pos=vector(0,0,0),axis=ball1.pos,color=color.red)
```

For the arrow object, there are two important attributes: `pos` is the location of the start of the arrow and `axis` is a vector from the starting position to the end of the arrow. The code above creates an arrow from the origin `(0,0,0)` to the position of `ball1`, i.e. the *position vector* of ball 1.

Now create two more arrows which represent the position vectors of balls 2 and 3. Your program's output should look like this:



Displacement vectors

Let's get rid of ball 1 and its position vector. We can do this by putting a # symbol at the start of the lines

```
#ball1=sphere(pos=vec(0,1,0), radius=0.1, color=color.red)

#r1=arrow(pos=vector(0,0,0),axis=ball1.pos,color=color.red)
```

The # indicates that the rest of the line is a *comment*. This means they are skipped by the computer when it is interpreting the program. Comments are inserted by programmers to help people (often themselves at a later date) read and understand the program. They can be placed anywhere in a program.

Now let's make a new vector `r` which is a *displacement vector*. It will displace us from ball 2's position to ball 3's.

```
r=ball3.pos-ball2.pos
rarrow=arrow(pos=ball2.pos, axis=r, color=color.orange)
```

Now go back to the three balls. Make 3 arrows pointing from each ball to the other balls.

Operations with vectors

VPython knows how to do sums with vectors, so vectors can be added. After getting rid of the vector arrows you have made so far (possibly by commenting out their lines of code), try this:

```
r = ball3.pos - ball2.pos
s = ball1.pos - ball3.pos
t = r + s
tarrow = arrow(pos=ball2.pos, axis=t, color=color.magenta)
```

Can you explain what is going on? Try tinkering around with this snippet to check that you understand vector addition.

Now try making the vector `t` longer or shorter, by multiplying it by a length factor `a` (try making `a` equal to 0.2, 1.0 and 3.4 in the snippet below:

```
a = 0.5
t = a * t # note that this means 'make t be equal to a times t' (not like everyday maths!)
tarrow = arrow(pos=ball2.pos, axis=t, color=color.magenta)
```

New variables can be created that are just vectors. `A = vec(2,-3,-1)` creates a vector. There are two other important vector operations. The `mag` function finds the magnitude of a vector and `norm` finds the unit vector.

You can actually calculate these two quantities manually. First, the x-component of vector `A` can be accessed via `A.x`:

```
A = vec(2,-3,-1)
print(A.x)
```

Now the following code will calculate the magnitude of `A` and the unit vector in the direction of `A`:

```
Amag = sqrt(A.x**2 + A.y**2 + A.z**2)
Anorm = A / Amag
```

Notice that in python, `sqrt` is the square root function and `**2` means 'to the power 2'.

- You should be able to work out on paper the magnitude and unit vector for vector `A`.
- Add two print statements to check your work.
- Display the unit vector `Anorm` with an arrow. Compare with with an arrow scaled via a constant multiplier to be the right size and direction.
- Repeat this using the `mag` and `norm` functions.
- Try out with other vectors!