Modeling motion of a baseball in flight with VPython

We are going to model a baseball in flight. We are also going to discuss some better programming style.

As usual, the program will have two parts:

1. **Before the loop**: The first part of the program tells the computer to:
   a. Create numerical values for constants we might need
   b. Create 3D objects
   c. Give them initial positions and momenta

2. **The "while" loop**: The second part of the program, the loop, contains the statements that the computer reads to tell it how to increment the position of the objects over and over again, making them move on screen. These statements tell the computer:
   a. How to calculate the net force on the objects
   b. How to calculate the new momentum of each object, using the net force and the momentum principle
   c. How to find the new positions of the objects, using the momenta

It is important to write programs that are easy to read. You should therefore put spaces and blank lines in your program to make it visually easier to read. You should also add comments to specify what you are doing. Variables and objects should have sensible names. Finally, numbers that might change (like masses, etc) should be defined in the beginning of the program so you can change them easily. So if you change from a baseball on the earth to a golf ball on the moon, you will not have to go dig through the program to find all the things that must be changed.

1. Before the loop

Creating the objects

• Open IDLE for Python, and type the usual beginning statement:

```
from visual import *
```

Most programs have constants that do not change throughout the program. It is useful to group these at the beginning of the program.

• Add a comment statement:

```
#Constants
```

This part of the program is where you should put any constants that are necessary. Think about what constants we will need to describe the flight of the baseball.

Next we need to create the objects for the program (in this case, a white sphere representing the baseball)

• A few lines below, add the comment:

```
#Objects
```

Below this line, create a white sphere. The radius of a baseball is about 3.67e-3 m. This will be hard to see, so make the radius 50 times bigger.

```
ball = sphere(pos=vector(0,0, 0), radius=50*3.67e-3, color=color.white)
```

The mass of the baseball is 143e-3 kg.

```
ball.m = 143e-3
```

- Add a trail, so we can see the flight of the ball:

```
ball.trail = curve(color=ball.color)
ball.trail.append(pos=ball.pos)
```

Run the program to see the make sure it is working. At this point, you should just see a white sphere.

Initial conditions

Any object that moves needs two vector quantities declared before the loop begins:

1. initial position; and
2. initial momentum.

We've already given the ball an initial position of <0, 0, 0> m. Now we need to give it an initial momentum. Since the definition of momentum at speeds much less than the speed of light, is $\vec{p} \approx m\vec{v}$, we need to tell the computer the ball's initial velocity.

- Two lines down, add a comment:

```
#Initial values
```

- On a new line type the following:

```
ball.v = vector(30,30,0)
```

We gave the ball an initial velocity of <30, 30, 0> m/s. (That is very fast. How fast is this in mph? You can play with changing the initial velocity later.) Then, the initial momentum would be the mass times this initial velocity vector.

- On a new line type the following:

```
ball.p = ball.m*ball.v
```

The symbol `ball.p` will stand for the momentum of the ball (a vector) throughout the program.

Remember: There are no "built in" physics attributes `p` or `m` for objects like there are built-in geometrical attributes `pos` or `radius`. We can make any attributes we want. We could have called the momentum just `p`, or the mass just `m`, instead of `ball.p` or `ball.m`. It's useful to create attributes like mass or momentum associated with objects so we can easily tell apart the masses and momenta of different objects

For this program, we can make the time step 1/10 of a second.

- On the next two lines type the following:

```
deltat = 0.1
t = 0
```

2. The "while" loop

- On a new line add the comment:

```
#Loop through repetitive calculations
```

We will continue the while loop while the ball is up in the air.

- On the next line type:

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

- Make sure the statement ends with ":", then press Enter. Note that the cursor is now indented on the next line. If it is not, place the cursor just after the ":" and press Enter.

To slow down the motion, add a rate statement.

- On the indented line after the "while" statement, type the following:

```
rate(20)
```

Next we want to update the position. To do that, we need the force so we can update the momentum. For now the only force on the ball is gravity.

- Add a statement:

```
Fgrav = vector(0, ..., 0)
```

where the . . . you need to fill in. You should not put any numbers in here. (Hint, you may need to put something in the constant section.)

We will add another force soon, so in the mean time, on the next line type

```
Fnet = Fgrav
```

- Now update the momentum, velocity, and position:

```
ball.p = ball.p + . . .
ball.v = ball.p/ball.m
ball.pos = ball.pos + . . .
```

- We need to add on to the trail of the ball:

```
ball.trail.append(pos=ball.pos)
```

Finally we need to increment the cumulative time **t**.
- Add the statement:

```
    t = t + deltat
```

- Now, run the program.

CHECKPOINT 1: Ask an instructor to check your work for credit.
You can read ahead while you're waiting to be checked off.

3. Adding physics

Now we will modify our program to make it more physical. To do this, we will add a force due to air drag. For a ball, traveling at typical baseball speeds, the force due to air drag has the form:

$F_{\text{drag}} = C\, v^2$,

were $C = 0.008$ kg/m, and the direction is opposite the direction of the velocity.

- In the constants section, make a new variable:

```
C = 0.008
```

- In the loop, right after the Fgrav line add:
  ```
  Fdrag = -(C*mag(ball.v)**2)*(ball.v/mag(ball.v))
  ```

- Include the drag force in the net force

- Run the program. You should now see the ball move a much shorter distance, and the path no longer looks like a parabola.

CHECKPOINT 2: Ask an instructor to check your work for credit.

4. Playing around

How much shorter does the ball go? Program the computer to print out the final position x position of the ball after exiting the loop, both with and without the drag force. (How can you easily change between including or not including the drag?) You could also add to the program so that a ball with and without the drag force is shown. Try changing the initial velocity (speed and/or direction) to see how the ranges change.

With drag, the ball does not go very far. What other physical effects are we not modeling which could account for the ball traveling further?