## Iterative Calculation of Projectile Motion

### *OBJECTIVES*

In this activity you will learn how to:

- Create 3D box objects
- Update the position of an object iteratively (repeatedly) to animate its motion
- Update the momentum and position of an object iteratively (repeatedly) to predict its motion

### *TIME*

You should plan to finish this activity in 50 minutes or less.

### *COMPUTER PROGRAM ORGANIZATION*

A typical program has four sections:

1. Setup statements
2. Definitions of constants (if needed)
3. Creation of objects and specification of initial conditions
4. Calculations to predict motion or move objects (done repetitively in a "loop")

### 1. Setup statements

Using the VIDLE editor, open the shell file, "Model2_Shell.py" and save it on your computer as "Model2.py". Notice that the first few lines are the import statements you have seen before.

### 2. Constants

Following the setup section of the program you would define physics constants. The only such constant we will need for this model is the Earth's gravitational acceleration, which you can see has been defined.

### 3a. Creating objects

The first two statements of the Objects section create a white track and a green cart to slide on that track. The next statement, "`cart.m=1`", creates a mass attribute for our cart variable and gives it a value of 1 kg.*

**Run the program** by pressing F5. Arrange your windows so the Python Shell window is always visible. You should currently see the cart sitting just above the track.
**Kill the program** by closing the graphic display window.

### 3b. Initial conditions

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

1. initial position; and
2. initial momentum

---

***Note:** In VPython there are no "built in" physical attributes like **`p`** or **`m`** the way there are for geometrical attributes like **`pos`** or **`radius`**. However, Python allows us to create new attributes for objects. We could have called the momentum and mass `pcart` and `mcart`, instead of `cart.p` and `cart.m` It can be helpful to create attributes like mass or momentum associated with objects so we can easily tell apart the masses and momenta of different objects in a complex program.

Edited by M. Haley

You've already given the cart an initial position at the left end of the track. Now you need to give it an initial momentum. If you push the cart with your hand, the initial momentum is the momentum of the cart just *after* it leaves your hand.

To make our code flexible, we will specify the initial velocity by an initial speed and angle. For now, the speed is 5 m/s in the direction of 0°, which is along the track in the +x direction. The `radians()` function converts degrees to radians. The `sin()` and `cos()` functions have been used to decompose the initial velocity into its x and y components.

Complete the next line defining the initial momentum using these components—use only symbols and no numbers. Remember that a three component vector can be created in VPython with the expression `vector( , , )`.

When you run your program, you should see the cart momentum outputted to the shell. Is this the value you expect, given the cart's mass and initial velocity? Nothing is animated yet, since we haven't yet programmed position updating.

### 3b. Time step and total elapsed time

To make the cart move we will use the position update equation $\vec{r}_f = \vec{r}_i + \vec{v}\Delta t$ repeatedly in a "loop". In a programming language, a loop is an order that executes the same operations multiple times. For our case, it will execute the position update many times and display the corresponding box location in a 3D scene each time the calculation is done.

Carrying out an iteration over time requires an initial time and a time step, both of which are defined in the shell code. For the moment we will suppose that our time step, `deltat`, is small enough for our purposes.

This completes the first part of the program, which tells the computer to:

a. Create numerical values for constants we might need
b. Create 3D objects
c. Give them initial positions and momenta

### 4. Repeated calculations: Loops

The kind of loop we will use in VPython starts with a "while" statement. Instructions inside the loop are indented; this is not just for readability! The Python interpreter uses the indent to parse your code, so you must indent the contents of a loop.

In the Calculations section, uncomment all of the lines below "`## CALCULATIONS`". The fastest way to accomplish this in the VIDLE editor is to highlight these lines and press **Alt-4** (Alt-3 would add comment markings). Note that you only want to remove the first # sign of each line.

Try running the program now. Looking at the shell, can you see what the following two active lines are doing?

```
print 'the time is now', t
t = t + deltat
```

This should give you a sense of the enormous number of times a computer can quickly do iterative calculations, which are quite tedious when done by hand. The loop will continue until the condition of the loop, `t < 10`, is

no longer true. Since we don't need to see the time for every run of the loop, comment out the print statement before going on (but leave the print statement that is after the loop.)

The statement:

```
t = t + deltat
```

may look like a mathematical error. However, in a program the "=" sign has a different meaning than in a mathematical equation. The right hand side of the statement tells Python to read the old value of `t` and add the value of `deltat` to it. The left hand side of the statement tells Python to store this new value into the variable `t`. For this reason, the "=" is better read as "gets" than "equals", as in the phrase, "t gets t plus deltat".

### *4a. Constant momentum motion*

Consider a cart moving with constant momentum. We will predict how the cart will move in the future, *after* it acquired its initial momentum using your iterative calculational "loop". Each time the program runs through this loop, it will do two things:

1. Use the cart's current momentum to calculate the cart's new position
2. Increment the cumulative time `t` by `deltat`

You know that the new position of an object after a time interval $\Delta t$ is given by

$$\vec{r}_f = \vec{r}_i + \vec{v}_{\text{avg}}\Delta t$$

where $\vec{r}_f$ is the final position of the object, and $\vec{r}_i$ is its initial position. Since the time interval $\Delta t$ is very short, the velocity doesn't have time to change very much and is therefore similar to the initial or final velocity. We will approximate the average velocity by the final velocity of each iteration.*

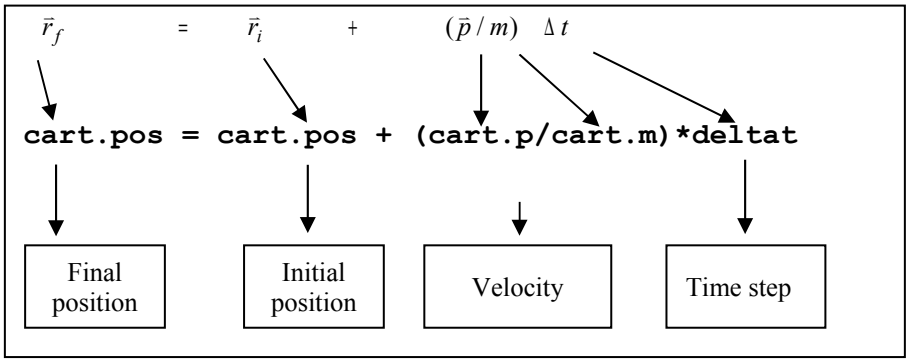
Since at low speed $\vec{p} \approx m\vec{v}$, or $\vec{v} \approx \vec{p}/m$, we can write

$$\vec{r}_f = \vec{r}_i + (\vec{p}/m)\Delta t$$

We will use this equation to increment the position of the cart in the program. First, we must translate it so VPython can understand it. Complete the position update line in your code with the following:

```
cart.pos = cart.pos + (cart.p/cart.m)*deltat
```

Notice how this statement corresponds to the algebraic equation:




* The accuracy of this approximation obviously increases as $\Delta t \to 0$.

***Slowing down the animation***

The animation speed of your program depends on many factors, such as the machine running the code. Depending on what you want to observe, it is important to know how to slow down or speed up the animation. Notice the statement in the loop,

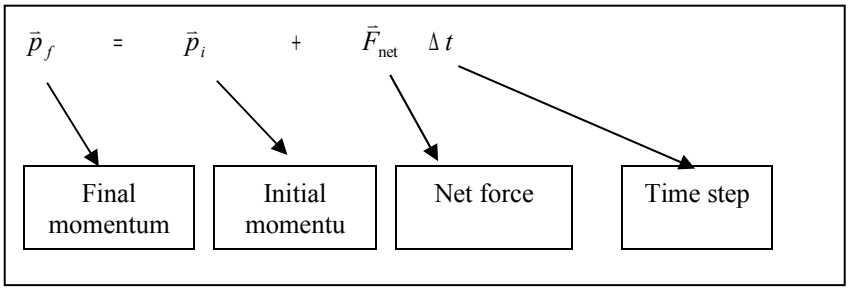
```
rate(100)
```

A larger number in the argument causes the program to run faster. Change the rate so that when you run the program, the cart takes a few seconds to reach the end of the track. You should observe a constant velocity at this point in the program.

**Note**: The cart going beyond the edge of the track isn't a good simulation of what really happens, but it's what we told the computer to do. There are no "built-in" physical behaviors, like gravitational force, in VPython. Right now, all we've done is tell the program to make the cart move in a straight line. If we wanted the cart to fall off the edge, we would have to enter statements into the program to tell the computer how to do this.

### *4b. Changing momentum*

Now we will complete the program by adding in the possibility of a changing momentum. Recall that a change in momentum is given by the Momentum Principle:




Complete the line in your code defining `cart.p`. If you are unsure how to accomplish this, look at how you programmed the position update—it is very similar. Your line should set the new value of the cart momentum equal to the old value of the cart momentum plus the net force times the time step. Note that the variable `Fnet` has already been defined in the line above the momentum update.

Since `Fnet` is currently set to the zero vector, the momentum update will still produce a constant momentum motion, but in a few minutes you will get to play with some other possibilities.

## 5. Reflection

At this point, you may begin answering the reflection questions for Model 2.

Remember that you only get one try per question on a reflection assignment, so be sure discuss with your group or an instructor before answering!