Computation Class 1: Introduction to Python, VPython, and Jupyter Notebooks

1. Getting started

We will edit and run programs in the Python language from an environment called a Jupyter notebook. (There are many other ways to run Python programs, also know as *scripts*.)

- – For Bucknell Windows computers:
  `Windows Menu` → `Anaconda 3` → `Jupyter Notebook`
  – For Linux computers:
  Start from a terminal and type the following on a command line
  `jupyter notebook`
  or something like
  `jupyter notebook --browser=firefox` if you want to specify a browser.
  – For Macs:
  ??

- A browser will appear running Jupyter. Click the button on the upper right labelled `New`. From the pulldown menu, select `Vpython`.

- You now have a Jupyter notebook running in which you can either start writing a new program. Click on the text `Untitled` near the top, and give your notebook a name — something like

  ```
  ligare_lab1
  ```

- Jupyter notebooks are organized into "cells"; the first cell is the box to the right of the `In[ ]`. In our work, the cells will be of two kinds: either

  – `Code` cells, in which you enter programming commands, or
  – `Markdown` cells, in you can add formatted comments.

  The kind of cell is indicated in the Menu Bar at the top. Change the initial cell to a `Markdown` cell, and enter the following in the cell

  ```
  ## My first notebook
  ```

  Then hit `Shift+Enter` (simultaneously). You should see something like a nice title, or section heading.

- Double click on the first cell, and change the number of `#` signs, and execute the cell by hitting `Shift+Enter` again. This is the first example of some of the formatting capabilities of the `Markdown` language.

- A new empty cell should have been automatically created below the first cell, and this should be a `Code` cell. In this cell, type

  ```
  print(6 + 5)
  ```

  and execute the cell. The output below the cell should make sense.

- Another `Code` cell should appear. In this cell type

  ```
  cos(0)
  ```

  and execute the cell.

  You get an error message because Python is a pretty bare-bones language, and doesn't on its own know about math functions like cosine and sine. But there many, many *modules* that can be imported into a Python program for a wealth of applciations.

- Single click on your title cell, and select

  `Insert` → `Insert Cell Below`

  In the new Code cell that appears, type

  ```
  from math import *
  from  vpython import *
  ```

  and execute the cell, to import two modules. The purpose of the `math` module is obvious; the `vpython` module has functions that allow you to do 3D animations easily. Nothing should appear to happen when you execute this cell, but if you go back to the cell with the cosine function, re-executing it should now give you a reasonable result.

2. Python as a Calculator

Type the following commands, or sets of commands into cells, and execute the cells.

- `print(6+5)`
- `print(2*3)`
- `print(2**3)`
- `print(x-y)` (Erase this command when you are done.)

  ```
  x = 3
  y = 5
  print(x-y)
  ```

- `print("Hello World!")`
- `print("x+y")`
- `print( (y+x)*(y-x) )`
- ```
  a = vector(1,2,3)
  b = vector(4,5,6)
  c = vector(-3,0,3)
  print(a + b)
  ```
  ```

- `print(dot(a,c))`
- `print(cross(a,b))`
- `print(cross(a,cross(b,c))`
- `print(b*dot(a,c)-c*dot(a,b))`
- `print(mag(a+b))`
- `print(sin(0))`
- `print(sin(pi/2))`
- `print(sin(pi))`
- `print(4.0/3.0)`
- `print(15//4)` (The `//` operator is called *integer division*. Can you see what is happening here?)

3. Assignment Operator

The equals sign in Python (and many other programming languages) is not that same as in mathematics. The math equation $x = x + 6$ has no solution. Now try this:

```
x = 1
x = x + 6
print(x)
```

Do you see what is going on here? Now try it with vectors:

```
a = vector(1, 1, 0)
b = vector(0, 4, 4)
a = a + b
print(a)
```

4. Loops

Loops are how we tell the computer to do a particular task (or a slight variation of it) over and over. The first line in a loop is a *conditional statement*. Everything on the following indented lines will be executed repeatedly as long as the condition is true. **The indentation is essential.** Jupyter will automatically indent the first line after a conditional statement, but you must "unindent" when you get to the end of the statements that are to be repeated. Execute each of the following four-line blocks of code (one at a time):

- ```
  i = 1
  while i<11:
      print(i)
      i = i+1
  ```

- ```
  i = 1
  while True:
      print(i)
      i = i+1
  ```

- ```
  i = 1
  while i<11:
      print(i)
  i = i+1
  ```

In the second example, the argument to the `while` loop is simply the constant value `True`, so the loop is continually executed. To break an infinite loop like this you will need to stop the process by selecting `Kernel` → `Restart`. What's going on in the third example?

5. Conditionals

Your program can do things only if certain conditions are met. Try this small program. Any indented lines after the `if ...:` will be executed if the condition is met. Try:

- ```
  k = 1
  while k<11:
      if cos(pi*k) == 1:
          print(k)
      k = k+1
  ```

Do you understand what is going on here? Try to explain it to your neighbor.

6. 3-D Geometric *Objects*

Start a new program (`File` → `New Notebook` → `Vpython`), or erase your previous work. Don't forget the line: `from vpython import *`. Try the following commands. NOTE: The commands will actually fit on one line in your notebook; they are wrapped to fit on the printed page.

- ```
  ball = sphere(pos = vector(0,0,0),
                radius = 0.5, color = color.red)
  ```

- ```
  wall1 = box(pos = vector(-10,0,0),
                length = 0.1, height = 10, width = 5, color = color.blue)
  ```

Execution of the preceding commands should produce a graphics cell with a 3-D view of a red ball beside a blue wall. You can can change the view with your mouse buttons: when the cursor is over the display window the middle mouse button will zoom you in and out when you move the mouse (on some computers you will need to use the right and left mouse buttons together to get the zooming effect), and the right mouse button will change the viewpoint. You should check that the *attributes* specified in the `box()` command make sense; you can change some to check your understanding. Now add a second wall:

- ```
  wall2 = box(pos = vector(10,0,0),
                length = 0.1, height = 10, width = 5, color = color.blue)
  ```

(Copying and pasting makes this pretty easy.) You can change the *attributes* of the *objects* later from within the program. For example adding the line:
  ```

- `ball.pos.x = ball.pos.x - 5`

should make the ball appear to the left of its previous position. You can see details of all the pre-defined geometrical attributes of 3D objects at `http://vpython.org/contents/docs/primitives.html`.

7. Animation: Combining 3-D Objects and Loops

Add the following lines to your program with the ball and walls.

- ```
  v = 1
  dt = 0.1
  while True:
      rate(20)
      ball.pos.x = ball.pos.x + v*dt
  ```

And that's your first 3-D animation! The `while` loop will never end on its own, so again you will need to stop the process by selecting `Kernel` → `Restart`. The `rate(20)` line just slows down the computer to make the animation rate about right. You can adjust this parameter.

- Now modify the program above to make the ball stop when it hits the wall. Instead of `while True`, we want `while ball.pos.x < 10`.

8. More Realistic Animation: Adding `if` Statements

Now try to modify the animation to have the ball bounce between the walls. Switch back to `while True`, but in the loop we will put in an `if` statement, for example

- ```
  while True:
      rate(100)
      ball.pos.x = ball.pos.x + v*dt
      if ball.pos.x > 10:
          v = -1
  ```

You add another `if` statement to handle bounces off the second wall (there are other ways to do this too).

Comment: If you have time you can think about improving and generalizing your script by letting `v` be a vector. Also, in the above code, when we check whether the ball is to the right of position `x = 10`, we really mean to check whether the ball is to the right of the wall. Can you think of a better way to do this?

9. Assignment

Your assignment for this week is to get the program working with the ball bouncing off of both walls. Once you have this working, hand in your code.

10. How to hand it in . . .

Read and follow these instructions carefully! If you named your program as suggested above, there will be file on your computer with a name

`yourlastname_labXX.ipynb`,

where XX is the two-digit assignment number (in this case 1) and "_" is the underscore character. If the program you wish to submit doesn't have this name, please rename your program before submission.

Once you've saved a copy of your code with this filename you can submit it by copying it into my drop box (`facultystaff/m/mligare`). (You can't save it directly there, but you can copy or drag the file into it.)

11. Going Further . . .

If you finish with time to spare, there are many things you could add to your bouncing ball simulation to make it more realistic or interesting. Try one or more of the following, or come up with a goal of your own and try to implement it. (You may want to do this in a new script, so you don't break the one you've just finished!)

- Make the bounces "lossy," so that the ball only regains some fraction of its incident kinetic energy after the bounce.
- Add walls on the other sides.
- Add gravity.
- Make the walls springy, so that there's a force on the ball that gets strong when the ball gets very close.
- Add a second ball that will bounce off the first.

12. Downloading Python for your Computer

This is not required, but I recommend it.

- Download the Anaconda distribution of Python from Continuum Analytics; `https://www.anaconda.com/`).
- Download the VPython module; see instruructions at `http://vpython.org/`. An easy method, after the Anaconda distribution has been installed is to use the shell command: `conda install -c vpython vpython`.

13. Additional Documentation

We will be using a very small subset of the Python programming language, but if you're interested in learning more there are many, many books and online tutorials that are available. Here are just a few:

- Documents available on the official Python website: `https://docs.python.org`

- Reference manual for VPython (which is also available as Help once the VPython module is loaded): `http://vpython.org/contents/docs/`
- *Learning Python*, Mark Lutz (O'Reilly, Sebastopol, CA, 2013)
- *Python Programming: An Introduction to Computer Science*, John Zelle (Franklin Beedle & Associates, 2010)