This activity aims to give you a (very) brief introduction to Python so that you will be able to focus on the physics while you are working on the solar system simulation. If you are familiar with programming, this will likely be something you can move through relatively quickly; when you finish it, work on the challenge problem at the bottom. If you have not spent much time programming, take this opportunity to ask lots of questions of your instructor and peers. Use the in-class time we spend on this to get yourself as prepared as possible.

On the last page of these instructions are a series of questions to answer as you move through the activity.

Getting Python

If you have not used Python before (and it's not expected that you have) it is recommended that you obtain Python by downloading Anaconda `https://www.anaconda.com/distribution/`. Follow the instructions on the website to download the Python 3.7 version for your operating system. Also note that you do not need to use the terminal or command line as the page suggests; you can edit the code in a browser-based front end through Jupyter.

If you already have experience with Python, feel free to use whatever incarnation of it you are most comfortable with.

Getting started with Python

Before you can jump straight into simulating the solar system, open your own notebook and experiment around with Python a bit. For now, you should just get in the habit of starting all of your Python code with the following two lines:

```
import numpy as np
import matplotlib.pyplot as plt
```

The first line allows you to use a range of mathematical tools. The second allows you to produce plots. The first line especially is very standard practice and something you will find in pretty much any Python code used for scientific computing.

Try running a few simple things. Can you get Python to tell you what $2+3$ is? How about 4×9? Or $\sqrt{4}$ (hint: try `np.sqrt`)? If you hit `Enter`, you will insert a line in your code. To run a chunk of code, hold `Shift` and press `Enter` (in a Jupyter notebook).

Vectors in Python

In physics computing you will need to use a lot of vectors. A good way to do that in Python is using numpy arrays. In the simulation code you will be working with in the simulation lab, you would implement the vector $\vec{v} = (1, 2, 3)$ using `v=np.array([1,2,3])`. Define this vector in your code. Also define the vector $\vec{u} = (4, 5, 6)$. Try the following operations on $\vec{v}$ and $\vec{u}$. What is each of them doing? Do they give you the results you expect?

```
u+1
2*v
```

```
v+u
v-u
v**2
u**0.5
np.sum(u)
#comment
v[1]
v[-1]
v[2]
v[-2]
```

Try combining all of the ideas from above to find $|\vec{v} - \vec{u}|$ (i.e., the length of the vector $\vec{v} - \vec{u}$). As a reminder, $|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$.

Functions in Python

You are probably familiar with mathematical functions, even if you are not familiar with functions in programming. We can think of addition as a function which takes two inputs (the numbers you're adding) and returns a single output (the numbers' sum). S short piece of Python code to add numbers could look like the following:

```
#bring in numerical functions and plotting utilities
import numpy as np
import matplotlib.pyplot as plt

#define a function to add two numbers
def add_two_numbers(n1,n2):
    sum = n1 + n2; #find the sum
    return sum #output the sum

add_two_numbers(5,6) #add 5 and 6
```

Make sure you understand what each of the three chunks of the code is doing:

- the first three lines have to do with importing mathematical functions and plotting utilities (see above),
- the last line adds the numbers 5 and 6 (it should return 11) by calling the function `add_two_numbers`, and
- the rest of the code gives a definition for the function `add_two_numbers`, which accepts two arguments called `n1` and `n2` and returns the `sum` of the two arguments.

  Now take the code you wrote to compute $|\vec{v} - \vec{u}|$ and make a Python function called `magnitude_of_difference` which takes two arbitrary vectors and computes the magnitude of their difference. Once you write this function, you should be able to type in the line

  ```
  magnitude_of_difference(np.array([1,0,0]),np.array([0,1,2]))
  ```

  and get `2.449489742783178`. Write your function on the answer sheet. Call your instructor over; they may give you random vectors to test your function and you will need their initials on your answer sheet.

  Keep this function; you will need to use it next week in the N-body simulation lab!

Logic in Python

In the N-body simulation, the code will need to divide by $|\vec{v}-\vec{u}|$. But if $\vec{v}$ and $\vec{u}$ are the same vector you will end up dividing by zero! You will need to use `if` statements to avoid doing this.[1] Writing a Python `if` statement looks like this:

```
if ((1==1)&(1==0))|(2==2):
    print("It's true!");
else:
    print("It's false!");
```

This tests first to see if $1 = 1$ and if $1 = 0$. If either both of those things are true, or if $2 = 2$, it prints "It's true!" Otherwise, it prints "It's false!" Take a minute before executing this code to think about what the output should be. Note: `==` is used to test whether two things are equal,[2] `&` is the logical AND function, and `|` is the logical OR function.

Use this to write a function called `reciprocal_of_magnitude_of_difference` which computes $1/|\vec{v}-\vec{u}|$ when $\vec{u} \neq \vec{v}$ and gives 0 when $\vec{u} = \vec{v}$. Write your code for this function on your answer sheet and get your instructor's initials again. (Two hints: $|\vec{v}-\vec{u}| = 0$ if and only if $\vec{u} = \vec{v}$ and `reciprocal_of_magnitude_of_difference` can call `magnitude_of_difference`.)

Doing a bit more with Python

If you make it quickly through the previous sections, try tackling the following problem by constructing a Python model of the situation:

What does the trajectory of an object launched at an angle from the edge of a cliff look like? If you know how to algebraically plot x as a function of y, that's great – but that's not what we're trying to do here. You should try to make a simulation which increments the object's position over many small time steps using the DVAT equations. Can you make your code:

- able to work for variable launch angles, cliff heights, initial speeds, and different gravities?
- detect when the object has reached the ground level at the base of the cliff?
- determine the time of flight and range of the projectile (again, without using algebraic formulas you may be able to write out)?

Don't forget to comment your code!

[1] The N-body code just returns zero instead of trying to compute $\frac{1}{|\vec{v}-\vec{u}|}$ if the vectors are the same. This strategy works in this case, but it is arguably bad coding practice (though why this is so is not necessarily important to understand in this class). If you get down to the Doing a Bit More with Python section, come back to this point and see if you can think of a better way to handle the computation of $\frac{1}{|\vec{v}-\vec{u}|}$ when $\vec{v}$ and $\vec{u}$ are equal.

[2] Whereas a single equals sign, `=`, sets them to be equal.

Questions

1. What do each of the following do? Explain using words, don't just quote the output.
   (a) `u+1`
   (b) `2*v`
   (c) `v+u`
   (d) `v-u`
   (e) `v**2`
   (f) `u**0.5`
   (g) `np.sum(u)`
   (h) `#comment`
   (i) `v[1]`
   (j) `v[-1]`
   (k) `v[2]`
   (l) `v[-2]`
2. Write down the code you used to compute $|\vec{v} - \vec{u}|$.
3. Write your code for `magnitude_of_difference`. Don't forget to get your instructor's initials!
4. Write your code for `reciprocal_of_magnitude_of_difference`. Don't forget to get your instructor's initials!