P395: Guide 0:

What is an Activity Guide?

Each week you will complete an activity guide that covers a topic in computational physics. These guides are designed to engage you in the process of active learning. Studies have shown that students learn much more when they are participating in the topic, doing exercises and applying what they have learned. The goal of these guides is to get you to

- ***Explore*** the topic, then
- ***Synthesize*** the information into a model or method, then
- ***Implement*** the method and then
- ***Analyze*** the results of the calculation.

The hope is that by engaging in such a process that you will develop the research skills, insight and creativity that result from engaging in independent work.

Each guide is organized into Tasks (highlighted in red), each providing a list of problems that you will have to work through. **Answer the problems in a Jupyter notebook**, that will be submitted for grading.

How are the Activity Guides assessed?

Review the NotebookMarkingRubric.pdf located in Canvas -> Course Information.

Can I consult outside resources and my classmates to complete the Activity Guides?

I expect that you will use Google to find information to help you complete the activities in the Activity Guides. I also expect that you will use it to search for how others have approached a given problem and solved it in Python. I encourage you to work together and share ideas with your classmates (see Discussion board on course Canvas page); however, you are required to write your own code and to cite any classmates or references that you used for a particular idea (unless it is textbook material). See the more detailed statement in the Syllabus. If you are ever uncertain about whether something should be cited, please contact me.

The Jupyter Environment: Set-up

To access Python, we will be using **syzygy**. To start a new session, go to: https://sfu.syzygy.ca/ and login with your SFU computing ID (*e.g.*, dsivak).

(You can use your own Python installation (e.g., Anaconda). However, I can only provide minimal help with installation/execution issues. NOTE: All activities have only been tested on the SFU syzygy platform and you may have installation specific issues if you use your own version of python.)

Sometimes it may appear that syzygy has crashed or has stopped. If you know that you have not asked it to run some crazy-long or memory-hogging calculation, then it is recommended to just wait for 10 or so minutes. Usually the kernel comes back and you can pick up where you left off. So wait before killing your kernel and restarting your notebook.

Task 1: Getting Started

The following task will get you familiarized with the computing environment that we will be using in this course. Open the syzygy page and:

1. Organize your home directory on syzygy. If you have never used syzygy before then there should be nothing in your home directory. Investigate how to create a new folder and rename it 'Phys395'. HINT: I recommend creating a new directory each week for that week's work. In the Phys395 folder create a new folder and rename it to be something like 'Guide 0'. Go into this folder and carry out the remaining actions.
2. Launch a new Jupyter notebook that runs Python
3. Rename your notebook to be 'Guide0_your_name' (*e.g.*, Guide0_David_Sivak). You will complete all this guide's activities in this notebook and will submit it for grading by the due date.

In the new notebook just created, we will need to import some Python packages that allow for plotting (`matplotlib`) and fast numerical calculations (`numpy`). These imported packages are each their own 'namespace' and we have to call their functions by referencing their namespace. It means a little extra typing, but it helps keep the code clear. (Some of you may have used Pylab in notebooks which loads these plotting and numerical packages, but omits their namespaces. The problem with Pylab is that it can easily lead to functions being accidentally overwritten or redefined. So the use of Pylab is discouraged these days.)

4. In the first cell type

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

   and execute it. (NOTE: Most of our notebooks will always start with these two lines of code)

Python: A (Re)Introduction

The goal of this week is to (re)familiarize yourself with Python. The activities below will focus on key skills and emphasize data types and functions that will be used frequently when doing numerics or data analysis.

Task 2: Cell types and formatting output

The notebook environment provides a convenient way to organize code and text. Let's input some headings into the notebook and learn how to comment some simple code.

1. Learn how to make text cells in your Jupyter notebook (it's known as Markdown).
2. Learn how to make a text cell into a heading.
3. Make a title for your notebook and include your name.
4. Make another text cell and put in a meaningful heading for this section.

5. Get a cell to print ‘hello world.’
6. Look up how to add comments in an input cell.
7. Print π with 3, 8, and 16 significant digits after the decimal. HINT: In python, π is represented by `np.pi` .
8. Put in a meaningful comment in the cell about the action.

Task 3: Data types

Python has a variety of data types, including integers, floats, strings, lists, and dictionaries. Variable types never have to be explicitly defined. However, it is important that you understand how to manipulate the various types.

Integers and floats

Carry out the following actions to learn about integer and floats in Python:

1. Set two variables to be integers (*e.g.*, `k=3` and `n=4`) and make a new variable that is the multiple of the two (*e.g.*, `m = n*k`). What value does it have? Use the command `type(m)` to confirm.
2. Now multiply your previous result by 2.0. What type is the result?
3. Cast this result back into an integer using the `int()` command.

NOTE: You can always find out what a command does by typing ‘?<command>’. Try `?int` to see what the command does and how to call it.

A floating-point number has a decimal. You can also define floats using scientific notation. So 1000.0 can be represented by 1e3.

4. Define a variable to have the value of 1.60218×10^{-19} using scientific notation. Print its value to see that you defined it correctly.

Strings

Strings are variables defined between ‘ ‘ or “ “.

5. Make up a string `prfx = ‘PHYS’` and `sfx = ’395’`. Concatenate the two strings together using `prfx + sfx`. What is the result?
6. Make a string that has a blank space between `prfx` and `sfx` and store it in a variable called `coursename`.

When parsing text/data it is often necessary to break strings up.

7. Use the `split()` function of string to split up `coursename` (*e.g.*, `coursename.split()`). What is the result?
8. Now try `coursename.split(‘\t’)` . What is the result?
9. Use a text box to write a short explanation of the difference between the two function calls.

You can always find a datatype's functions by typing the variable followed by '.' and then hitting the tab key.

10. Try it with `coursename.<tab key>`. Look up the help of one of the other string functions and briefly summarize what it does in a text box.

Task 4: Lists

Lists are a powerful data type in Python and function like arrays. Their elements can be mixed data types. You can do a variety of operations on lists, but note they are not vectors or matrices (we will see in a couple weeks the big differences between lists and numpy arrays, which are used to do linear algebra).

A list is created by using `[]`. You can make an empty list using `l = []`.

1. Make an empty list.
2. Make a list containing 3 elements, `l1 = [1, 2, 3]`.
3. One of the most useful list functions is `append`. Append `4` to the end of your list.
4. Make another list, `l2 = [5, 6]`. See what happens when you do `l3 = l1 + l2`.
5. What happens if you do `3*l2`?
6. Make another list, `mixed = ['dog', 2, 'rabbit', 5, 'lemur', 2]`. Use the list `count()` function to count how many instances of 'dog' and instances of 2 there are.
7. Use the list `index()` function to find the index of 'rabbit' and the index of 2.
8. What did `index` do in the case where there was more than one instance of the item?

Elements in lists are indexed starting from 0 (not 1 as in, *e.g.*, matlab). You can access individual elements using the `l[index]` where `index` is some integer, starting from 0 (*e.g.*, `l[0]` is the 1st element of the list, `l[1]` is the 2nd element, and so on).

9. Print out the 1st and 3rd element of the list `l3`.
10. You can get the length of a list using the `len()` function. What is the length of `l3`?
11. What happens if you try to use an index that is as long or longer than the list?
12. You can also index from the end of a list using negative indexes. The last element of a list is indexed with -1. Print out the 2nd last element of l3.
13. What happens if you try to use a negative index that is longer in magnitude than the list?

You can also pull out multiple sequential elements of a list using slicing. To slice a list use the format `l[start:end]` which returns values starting from index `start` up to `end-1`. You can use alternate stepping with the format `l[start:end::stepsize]` (*e.g.,* `l[2:8::2]` would step every 2 elements).

14. Slice out the 3rd to 5th elements of `l3`.
15. Try `l3[4:10]`. Does this produce an error?
16. Use a different stepsize to get every 3rd element of `l3`.
17. Now try `l3[::-1]`. What happened?

Task 5: Dictionaries

Dictionaries are useful datatypes for organizing data and information. They are particularly useful when reading in data that has multiple labels that are potentially unknown. A dictionary is organized by key-value pairs. Instead of indexing elements using integers as was the case for lists, elements are now referenced using their key value. A dictionary values can be any python type – integer, float, string, list, another dictionary, etc.

1. To define an empty dictionary enter `d = {}` in a cell.
2. You can now assign key-value pairs to your dictionary. For example, to add the key 'dog' with the value 2, enter `d['dog'] = 2`.
3. Add key-value pairs of 'rabbit' and 5 and 'lemur' and 2 to the dictionary.

To get the keys of a dictionary use the `keys()` function of the dictionary (e.g., `d.keys()`). To get the values use the `values()` function.

4. What do these two functions return for your dictionary?

NOTE: The list of keys of the dictionary do not need to be in the same order that you defined them.

Task 6: NumPy Arrays

Here is a brief review of NumPy arrays. It is useful to look here at defining NumPy arrays and some of the simple operations such as indexing and slicing.

1. Define a 1D array with `a = np.array([1., 3., 2.5])`.
2. Carry out `2.*a`. What do you get?
3. Try `a*a`? What do you get?

You can do all sorts of mathematical operations on arrays. For example, you can pass arrays to functions.

4. Try `np.sin(a)`. What do you get?

1D arrays are indexed just like lists. Things get more interesting when we work with multidimensional arrays where slicing leads to some very useful techniques.

5. Define a 2D array, `a = np.array([[1.1, 2.1, -1.2], [0.2, -1.5, 0.3], [0.2, 3.4, -1.3]])`.
6. Use the `np.shape` command to find the shape of your array.

To get a single element you write `a[i,j]`, where `i` and `j` are integers that index the row and column (always starting at index 0 for both row and column).

7. Print the 1st element of the 2nd row.

You can access a whole row with `a[i]` where `i` indexes the row. You can get a whole column with `a[:, j]` where `j` indexes the column.

8. Print out the 1st column of the array.

Slicing NumPy arrays is done similarly to lists where you can use `<start>:<end>` on any of the dimensions of the array.

9. Print out the sub-array containing just the elements of the 2nd and 3rd rows and 2nd and 3rd columns.

Task 7: Loops

Many numerical calculations require iterations. Python provides a variety of looping structures.

For loops:

The simplest looping structure is the `for` loop. The syntax of the for loop is `for <variable> in <iterable variable>:`, where the iterable variable can be as simple as a list or a more complex data type that returns multiple values. The items in the list do not have to be integer indexes.

To create an iterable list of integer indexes use the `range` function (e.g., `range(0,10)` returns an iterable object going from 0 to 9).

1. Write a for loop that prints out each index and corresponding element of `l3` for all elements in `l3`. (You may find the `len` function useful here.)

As mentioned, the iterable can be any datatype that contains multiple elements. So, the keys of a dictionary make a perfectly fine iterable. For example, to iterate over the keys of your previous dictionary you could write `for animal in d.keys():` (or more simply `for animal in d`). Now the variable animal will loop over all key values in the dictionary.

2. Write a for loop that loops over the keys in `d`, adds one to all their dictionary values, and prints out the corresponding key and the updated value.

While loops:

We may also want to iterate until a certain condition is met. This can be done using while loops. The syntax of a while loop is `while <Boolean condition>:` where if the condition is `True` the commands in the while loop get executed and it will terminate if it is `False`. Python offers a variety of ways of generating Boolean conditions with the most common being the operators `<, >, <=, >=, ==` for equal to, and `!=` for not equal to. Any operation or function that yields `True` or `False` can be used as the Boolean condition.

3. Compose a loop that, starting at 0, sums up consecutive numbers while the sum is less than some value. At each pass through the while loop, print out the number that was added and the current value of the sum.

(NOTE: a short hand for doing the operation `x = x + <value>` is `x += <value>`. Similar shorthand is available for subtraction, multiplication, and division.)

4. Try out your while loop on values such as 23 and 42.

(NOTE: take care when naming variables. In the notebook, if your variable is green that means that it's a python command and you should name it something else.)

Task 8: Flow Control

Another necessary coding operation is to test a condition and take different actions based on the result. This is done with if statements in python. The syntax is `if <Boolean condition>: <code block> (elif <Boolean condition>: ) <code block> else: <code block>`. A further point about Boolean conditions is that various condition tests can be grouped together with Boolean logic operations such as `or`, `and`, or `not` (e.g., `(<test 1>) or (<test 2>)`, or `(not <test1>) and (<test2>) )`.

1. Compose a loop that uses an if statement to create a new list containing only the elements that are > 2 and < 5 from `l3`.

List Comprehension

Python, like C and C++, has a variety of ways of shortening code. One valuable piece of shorthand is list comprehension, which gets rid of the nested loop(s) in building a list and puts everything onto a single line. The simplest syntax of a list comprehension is `[<calculation on x> for x in <iterable>]`.

2. Use list comprehension to generate a list of integers of all powers of 2 from 2^0 to 2^5 (the `**` operator does the power operation in python).

You can also stick a Boolean condition in a list comprehension to use only certain elements in the iterable. This is done with the syntax `[<calculation on x> for x in <iterable> if <Boolean test on x>]`.

3. Use list comprehension to generate a list from `l3` above in Task 4, containing only the elements that are > 2 and < 5.

NOTE: List comprehension is a very convenient tool and can be nested to build higher-dimensional lists; however, if you use them in your code, please comment your code so that we know exactly what you are trying to generate. Uncommented code can be very hard to understand.

The syntax and methods that you just worked through will form the basis for most of your code.

Writing Python Functions and Plotting

Task 9: Exploring the Lennard-Jones Potential

You are interested in modelling a system of non-bonded neutral particles that interact only over short distances via the Van der Waals potential. You have heard that the 12-6 Lennard-Jones potential is a good approximation to this. In the following, you will explore this potential and learn how to write python functions.

1. Make a heading for this part of the notebook, such as 'Lennard-Jones Potential'.

2. Read about the Lennard-Jones (LJ) potential on wikipedia.

Writing up mathematics in Jupyter notebooks is relatively straightforward, as equations can be formatted using LaTeX. For instance, you could use the latex commands `\begin{equation}` and `\end{equation}` to enter a mathematical equation. You can also use '$ … $' within a text cell to enter a mathematical expression.

3. In a text cell(s), write a short summary on the 12-6 Lennard-Jones potential and type up its equation. Define its key parameters/variables and give the physical units that they have.

More on units: LJ potentials represent interactions at molecular length scales. Instead of using meters for the lengths in the LJ potential, it is better to use nanometers (or Angstroms) as the unit. So, all length quantities would be given in nm. However, we can go one step further and get rid of the parameter σ if we define all lengths in terms of it. For example say σ=0.5 nm, then a distance r=2.3 nm corresponds to $r = 4.6\ \sigma$. So if we take the distance r to be in units of σ, then we don't need to have σ as a parameter in the LJ potential (this process of redefining variables in terms of unitful parameters is called non-dimensionalization).

Similarly for energy, since the LJ potential represents molecular interactions, writing the energy in terms of Joules is not on the correct scale. The typical energies of molecular interactions are 0.02 – 10's of eV (recall an electron volt is 1eV = 1.6×10^{-19} J). For molecular systems at room temperature, a better energy scale to use is thermal energy, $k_B T$ = 0.025 eV, where k_B is Boltzmann's constant and T=293 K. So instead of expressing energies in terms of eV, one typically writes energetic parameter in units of thermal energy for molecular systems at or near room temperature.

4. Convert ε=0.12 eV into units of thermal energy.

Defining Python functions:

5. Learn how to use the `def` command to define a function.
6. Define a Python function that computes the LJ potential at a distance r. You should pass the other parameter(s) as arguments to the function.
7. Define a variable that takes on a range of r values using a function like `np.arange` or `np.linspace`. You will likely need to experiment with this range to make a nice plot of your potential.
8. Call your function with your set of r values above (e.g., `LJpot(rs, epsilon)`). Does it return an array of values? it should.
9. Look up help on `plt.plot` and plot your function. Experiment with the range of r so that you can see the full shape of the potential with a well defined minimum of energy.
10. Plot several LJ potentials on the same graph but with different values of ε.
11. Learn how to add a legend to the plot using the function `plt.legend,` and use your values of ε as labels (e.g., 'ε = 1', 'ε=2' etc.). HINT: To use LaTeX inline formatting in your text you can use python strings that begin with r'$ latex math stuff goes in here $'. That will let you easily put in Greek symbols and use subscripts and superscripts etc.
12. Label your axes using the functions `plt.xlabel` and `plt.ylabel`, and include units.
13. Figure out how to use matplotlib to save your plot as a file in your working directory.
14. Download the plot file from syzygy to your local computer.

Task 10: Reading in data – Analyzing some GPS tracking data

You've been given some GPS tracking data of an errant droid. You are asked to visualize its trajectory and calculate various kinematic quantities from it.

1. Start a new section of cells in your python notebook and put in a text cell with a heading such as 'Trajectory Analysis'.
2. Download trajectory data '`droid_traj.csv`' from Canvas under the 'Data' module. Save it to the local computer. NOTE: The data gives *x* and *y* measurements (in meters) of the droid at uniformly spaced intervals of time at a rate of 60 snapshots per minute.
3. Learn how to upload the file into your syzygy working directory.
4. In syzygy, click on the file to look at its contents and how it is formatted.
5. A good python command for loading in array data is `np.genfromtxt`. Look up the help on `np.genfromtxt` and load the data in the file into a 2D array.
6. What is the shape of the array?
7. Look up `plt.scatter` and plot the trajectory data as a scatter plot. HINT: use array slicing to select out the x and y coordinates separately).
8. Calculate from the data the pathlength *s(t)* as a function of time. What total distance did the droid travel? HINT: You may find the commands `np.diff` and `np.sum` useful for doing this and the following calculations. Using these commands, you can do these calculations without any explicit loops.
9. Calculate a numerical estimate for the speed $v(t) = ds(t)/dt$ of the droid along the path as a function of time.
10. Calculate an estimate for its tangential acceleration as a function of time.

Let's plot the pathlength, speed, and acceleration. In matplotlib, subplots are created using `plt.subplot(Nrow, Ncol, index)`, where `Nrow` specifies the number of rows , `Ncol` specifies the number of columns and `index` specifies the specific subplot (starting with index=1!). So for example for a plot containing 6 subplots arranged in 3 rows and 2 columns, the 3rd subplot is referenced with `plt.subplot(3, 2, 3)`. All plotting commands will only address that subplot until a different subplot is referenced using `plt.subplot`.

11. Learn how to use `plt.subplot()` and make a plot having 3 rows of subplots where s(t) is plotted in the first, v(t) in the 2nd and a(t) in the 3rd. Label the axes of each of your subplots (with units). TIP: When making multiple subplots, a nice function that tidies up the appearance and keeps the tick labels from clashing is to use `plt.tight_layout()` after all your plotting commands.
12. Now make another plot showing a scatter plot of the trajectory again, but this time color the points using the tangential acceleration that you found above.

Task 11 – Wrap up

You have finished this week's activity guide. At the end of each activity I would appreciate any feedback you have on the Activity guide. Using text cells, please provide answers to the following questions:

1. Were there any questions that were confusing or worded in a way that made them difficult to understand? If so, please reference the Task and question number, so that I can improve it.
2. Please provide any other feedback that you think might have helped to improve your learning experience with this Activity Guide.
3. Download your notebook to the local computer and email it to yourself for safe keeping (i.e., make a backup). Don't assume that your files will be there the next time you log in to syzygy (but let's hope so).