Introduction to Python

U24568 - Computational Physics
Dr. Karen Masters
University of Portsmouth
(Dated: November 21, 2016)

We gain familiarity with python through working on physics problems.

I. INTRODUCTION TO PYTHON

In this section we gain familiarity with python (specifically python 3), including different methods to run python by working on Physics problems. Remember you have not enrolled on a programming unit – rather this is "Computational **Physics**", so our emphasis here will be on using computers to solve physical problems. However a good basis in programming will provide both excellent transferable skills, and give you the skills to ensure you physics computing will go smoother (and faster) in the future. We chose to teach in python rather than the Matlab you learned in "Introduction to Computational Physics" because:

1. it gives you another language to list on you CV
2. it will give you confidence that you can pick up any language you need
3. it is a sought after language by the employers of physicists
4. it's free and open source (you don't need a license like for Matlab).

We will also have a short period of Maple to give you an alternative method to do computer algebra which is commonly employed by theoretical physicists, and during discussions on accuracy and speed of particularly high performance computing you will also have some exposure to the Fortran language.

II. INSTALLING PYTHON

By far the simplest method to run python for scientific use is via the Anaconda distribution, which uses all the commonly known scientific packages such as Matplotlib, Scipy, Numpy, and takes care of all the relevent dependencies for you. It also comes with different ways to interact with python (see below).

A. University Networked Computer: Windows

The Virtual Machine, has Anaconda ready installed. To access this:

- Open Chrome web browser
- Navigate to: `https://netgate.ee.port.ac.uk/sunstone/`
- Log in with your usual Portsmouth username/password
- Open a terminal and type `python` and check that it's a version of python3 which loads (if not see a Lecturer).

B. Your own computer/laptop

Download Anaconda. It is available for Windows, OSX and Linux installation. Or follow the instructions above to access the Virtual Machine. Make sure you make a python3 environment.

III. RUNNING PYTHON

We will experiment with three different methods to run python programmes - from the command line, using the Interactive Data Environment "Spyder" (which is supposed to look at lot like Matlab), and using Jupyter Notebooks.

A. Spyder

Spyder is the Scientific PYthon Development EnviRonment. It's one of many Integrated Development Environments for the python language, and it's distributed with the Anaconda python package. I have been told it's quite similar to the Matlab Development Environment, so this may be the most familiar way for you to run python.

1. Start up Spyder (type `spyder` in the terminal).

2. Work through the beginning of the Python Programming for Physicists tutorial (Chapter 2 of Newman [1]) up to and including Exercise 2.2: "Calculating the Altitude of a Satellite" using Spyder as your IDE (instead of IDLE as suggested in the Chapter).

B. Juptyer Notebooks

Jupyter Notebooks are web based application which provide a way to create and share documents, which contain live, updateable code. Juptyer can be used with multiple languages, but we'll focus on using it with python. Jupyter is included in the Anaconda distribution.

1. Start up a Jupyter Notebook (type `jupyter notebook` in the terminal).

2. Please redo Exercise 2.2 in a Jupyter Notebook.

C. Command Line

This method of using python will be easiest if done under linux. Open the code you wrote for Exercise 2.2 using a linux text editor (e.g. `nano`. What do you notice. Now run this code from the command line.

To do this you simply type:

```
> python code.py
```

(where `code.py` is the file name of your python code).

Write a simple "I love Physics" code (ie. a code which runs and prints "I love Physics" to the terminal) in a text editor to run from the command line.

Tip: there are more advanced text editors available in Linux which colour code the coding syntax. I often use a version of Emacs to write code for this reason.

IV. PYTHON FOR PHYSICS TUTORIAL

Now please work through the remainder of Chapter 2 of Newman [1] using the python interface of your choice.

Tips:

- Save your programmes to run later with names like `example.py`
- Give your variables sensible names (so you can remember what they are)
- Your code will be more efficient (and less bug prone) if you define variables to have the correct types (integer/floating etc).
- Mistakes and error messages are normal in programming, and working out what they mean can be tricky (but is not impossible). Google (or your search engine of preference) will be your friend here, as will asking your peers (and your lecturers) for help.

V. PHYSICS EXAMPLES FOR YOUR PORTFOLIO

Please work through the following exercises writing commented code to hand in for your portfolio. Please use all three methods we have been practicing to interact with Python (i.e. write one as a Python script in a text editor to run, do one in the Spyder IDE and one as a Jupyter Notebook. You may choose which method to run the fourth, exercise but please explain your choice. There is no correct answer, it's simply a function of programming style and personal preferences).

1. Calculating the Altitude of a Satellite - Exercise 2.2 (Newman [1], pg 30)

2. Quantum potential step - Exercise 2.5 (Newman [1], pg 36) -

3. The semi-empirical mass formula - Exercise 2.10 (Newman [1], pg 75)

4. Make a user defined function to calculate binomial coefficients - Exercise 2.11 (Newman [1], pg 82)

[1] Newman, M., Computational Physics - Revised and Expanded, [2013]
[2] Why Python: http://lorenabarba.com/blog/why-i-push-for-python, [2014]
[3] Jupyter Notebooks: http://jupyter.org/, [2016]

Exercise 2.2: Altitude of a satellite

A satellite is to be launched into a circular orbit around the Earth so that it orbits the planet once every T seconds.

a) Show that the altitude h above the Earth's surface that the satellite must have is

$$h = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3} - R,$$

where $G = 6.67 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ is Newton's gravitational constant, $M = 5.97 \times 10^{24}\,\mathrm{kg}$ is the mass of the Earth, and $R = 6371\,\mathrm{km}$ is its radius.

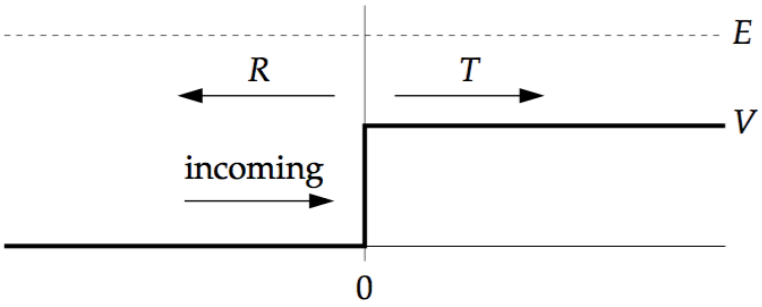
b) Write a program that asks the user to enter the desired value of T and then calculates and prints out the correct altitude in meters.

c) Use your program to calculate the altitudes of satellites that orbit the Earth once a day (so-called "geosynchronous" orbit), once every 90 minutes, and once every 45 minutes. What do you conclude from the last of these calculations?

d) Technically a geosynchronous satellite is one that orbits the Earth once per *sidereal day*, which is 23.93 hours, not 24 hours. Why is this? And how much difference will it make to the altitude of the satellite?

Exercise 2.5: Quantum potential step

A well-known quantum mechanics problem involves a particle of mass m that encounters a one-dimensional potential step, like this:



The particle with initial kinetic energy E and wavevector $k_1 = \sqrt{2mE}/\hbar$ enters from the left and encounters a sudden jump in potential energy of height V at position $x = 0$. By solving the Schrödinger equation, one can show that when $E > V$ the particle may either (a) pass the step, in which case it has a lower kinetic energy of $E - V$ on the other side and a correspondingly smaller wavevector of $k_2 = \sqrt{2m(E-V)}/\hbar$, or (b) it may be reflected, keeping all of its kinetic energy and an unchanged wavevector but moving in the opposite direction. The probabilities T and R for transmission and reflection are given by

$$T = \frac{4k_1k_2}{(k_1+k_2)^2}, \qquad R = \left(\frac{k_1-k_2}{k_1+k_2}\right)^2.$$

Suppose we have a particle with mass equal to the electron mass $m = 9.11 \times 10^{-31}$ kg and energy 10 eV encountering a potential step of height 9 eV. Write a Python program to compute and print out the transmission and reflection probabilities using the formulas above.

Exercise 2.10: The semi-empirical mass formula

In nuclear physics, the semi-empirical mass formula is a formula for calculating the approximate nuclear binding energy B of an atomic nucleus with atomic number Z and mass number A:

$$B = a_1 A - a_2 A^{2/3} - a_3 \frac{Z^2}{A^{1/3}} - a_4 \frac{(A-2Z)^2}{A} + \frac{a_5}{A^{1/2}},$$

where, in units of millions of electron volts, the constants are $a_1 = 15.67$, $a_2 = 17.23$, $a_3 = 0.75$, $a_4 = 93.2$, and

$$a_5 = \begin{cases} 0 & \text{if } A \text{ is odd,} \\ 12.0 & \text{if } A \text{ and } Z \text{ are both even,} \\ -12.0 & \text{if } A \text{ is even and } Z \text{ is odd.} \end{cases}$$

a) Write a program that takes as its input the values of A and Z, and prints out the binding energy for the corresponding atom. Use your program to find the binding energy of an atom with $A = 58$ and $Z = 28$. (Hint: The correct answer is around 490 MeV.)

b) Modify your program to print out not the total binding energy B, but the binding energy per nucleon, which is B/A.

c) Now modify your program so that it takes as input just a single value of the atomic number Z and then goes through all values of A from $A = Z$ to $A = 3Z$, to find the one that has the largest binding energy per nucleon. This is the most stable nucleus with the given atomic number. Have your program print out the value of A for this most stable nucleus and the value of the binding energy per nucleon.

d) Modify your program again so that, instead of taking Z as input, it runs through all values of Z from 1 to 100 and prints out the most stable value of A for each one. At what value of Z does the maximum binding energy per nucleon occur? (The true answer, in real life, is $Z = 28$, which is nickel. You should find that the semi-empirical mass formula gets the answer roughly right, but not exactly.)

Exercise 2.11: Binomial coefficients

The binomial coefficient $\binom{n}{k}$ is an integer equal to

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \times (n-1) \times (n-2) \times \ldots \times (n-k+1)}{1 \times 2 \times \ldots \times k}$$

when $k \geq 1$, or $\binom{n}{0} = 1$ when $k = 0$.

a) Using this form for the binomial coefficient, write a Python user-defined function `binomial(n,k)` that calculates the binomial coefficient for given n and k. Make sure your function returns the answer in the form of an integer (not a float) and gives the correct value of 1 for the case where $k = 0$.

b) Using your function write a program to print out the first 20 lines of "Pascal's triangle." The nth line of Pascal's triangle contains $n + 1$ numbers, which are the coefficients $\binom{n}{0}$, $\binom{n}{1}$, and so on up to $\binom{n}{n}$. Thus the first few lines are

1 1
1 2 1
1 3 3 1
1 4 6 4 1

c) The probability that an unbiased coin, tossed n times, will come up heads k times is $\binom{n}{k}/2^n$. Write a program to calculate (a) the total probability that a coin tossed 100 times comes up heads exactly 60 times, and (b) the probability that it comes up heads 60 or more times.