Stage 3 Introductory Python Exercises

Physics Advanced Laboratories

September 21, 2020

Instructions: Please make one Jupyter Notebook per exercise with the different sections of each exercise in separate cells. Use Markdown cells for labelling, explanations etc. Submit the Jupyter Notebooks as PDF with all PDFs combined into a single document. The exercises will be assessed for correctness (does it do what it is supposed to?) as well as structure, readability (including use of variables to improve readability and meaningful naming of variables), axes labels and titles for plots (where relevant) and comments/descriptions where appropriate.

1. The equation to convert a temperature from Celsius (T_C) to Fahrenheit (T_F) is

$$T_F = \frac{9}{5} \times T_C + 32 \tag{1}$$

   (a) Write a program which asks the user to enter a temperature in Celsius and prints to the screen the corresponding temperature in Fahrenheit, with two decimal places. (you are strongly encourages to use Python f-strings for formatting)

   (b) Write a function to do the Celsius to Fahrenheit conversion and repeat the previous exercise using this function.

   (c) Write a function which takes a string containing the temperature and its scale (as a single character, which may be any of "F" or "f" for Fahrenheit, "C" or "c" for Celsius, "K" or "k" for Kelvin), e.g. `"24 C"`, and returns a dictionary with the temperature in each of the three scales, with the keys being `"C", "F", "K"`. If an invalid character is entered then the function should return `None` instead of a dictionary (note: ideally we would use an exception here but that is a more advanced Python topic!).

   (d) Write a program to test the above function by prompting the user to enter the temperature and scale, access the values in the dictionary returned by the function, and print all three in an appropriate format, including an error message if `None` was returned.

   (e) Write a program which asks the user to enter an initial value, final value, and step size, in degrees Celcius and which loops over the desired range and prints the numbers and their corresponding values in Fahrenheit to the screen in a nicely formatted and aligned (to the decimal point) table (i.e. in columns), with one decimal place (use the function you developed earlier to do the conversion). Test with range -20 C to 50 C in steps of 5 C. (Note: this is not looking for anything

fancy like a Pandas table - just plain formatted text as illustrated at the end of this question!)

(f) Make a modified version of the above so that the table is saved to a text file (prompt the user to enter the filename and check it is not empty - if it is empty then do not save). See "Python I.pdf" (slide 36) for more information on writing to a file.

Useful Python commands:

`input()`	ask user to enter a string
`float()`	convert to a floating point number
string `split()`	split a string in to a list, e.g. `"a b c d".split()` → `["a","b","c","d"]`
string `upper()`/`lower()`	convert string to upper case or lower case

Table example output:

```
Enter start, end, and step TC, separated by spaces: -8 40 3
  -8.0   17.6
  -5.0   23.0
  -2.0   28.4
   1.0   33.8
   4.0   39.2
   7.0   44.6
  10.0   50.0
  13.0   55.4
  16.0   60.8
  19.0   66.2
  22.0   71.6
  25.0   77.0
  28.0   82.4
  31.0   87.8
  34.0   93.2
  37.0   98.6
  40.0  104.0
```

2. Note: please use Numpy arrays and vectorised operations for this exercise rather than Python lists and `for` loops.

   The equation of a cosine wave of amplitude A and frequency ν, and initial phase angle ϕ as a function of time is:

   $$y(t) = A\cos(2\pi\nu t + \phi) \tag{2}$$

   (a) Plot a cosine function with frequency 10 Hz, amplitude 1.0 (arbitrary unit) and phase angle 30° in the range $t = 0$ to 2 seconds (use 500 points). Note: π=`np.pi`

   (b) On a single graph plot in different colours two cosine functions, one of frequency 10 Hz, amplitude 1.0 and phase angle 30° and the other of frequency 8 Hz, amplitude 0.8 and phase angle 60°.

   (c) Determine and plot the resulting wave that occurs when the two waves from (2b) interfere.

   (d) Plot the frequency spectrum (amplitude -vs- frequency) determined by a fast Fourier transform (FFT) of the interference pattern from exercise (2c):

      i. Use `np.fft.fft()` which takes a numpy array as argument and returns the fast Fourier transform (FFT) of it.

      ii. The FFT algorithm returns an array of complex amplitudes with the second half of the array being a mirror copy of the first half - hence you will want to create a new array that contains the `np.abs()` of the first half of the array returned by the FFT function.

      iii. The frequencies of the elements of the array from (2(d)ii) are: 0, f_{min}, $2f_{min}$, $3f_{min}$ … $\frac{N}{2}f_{min}$, where $f_{min} = 1/T$ where T is the maximum time and N is the number of points in the input array to the FFT function.

   Use subplots to create a second plot to 'zoom' in on the region of interest. Is the resulting graph as expected?

Conventions:

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

To embed the plots in a Jupyter notebook:

```
%matplotlib inline
```

or

```
%matplotlib notebook
```

Useful Numpy commands:

`np.linspace(start, stop, numpts)`: makes a numpy array with `npts` from `start` to `stop`, both inclusive.

Useful Matplotlib commands (assuming `import matplotlib.pyplot as plt` and don't forget the `%matplotlib inline` in the notebook):

- `plt.plot(x,y)` plot data `y` vs `x`
- `plt.axis([xmin, xmas, ymin, ymax])` set the axis limits on a plot or `plot.xlim(min,max)` to set just the x-axis limits (similar for y-axis).
- `plt.subplot(abc)` creates `a` subplots in the y direction, `b` in the x direction an selects subplot `c` to draw in. Example: to create two subplots, one above the other, and select the first use `plt.subplot(211)` (to select the second use `plt.subplot(212)`).
- if when using subplots the axes of one plot and the title of the second overlap use `plt.tight_layout()` to separate them.

3. Monte Carlo simulation to estimate the value of π.
   The Greek philosophers are said to have estimated the value of π by drawing a quarter of a circle on a large flagstone. The ratio of the number of raindrops that fell in the quarter circle to the total number that fell in the square gave an estimation of the relative areas of the two shapes. The area of the whole flagstone can easily be calculated so the area of the quarter circle can then be estimated and hence a value for π determined.

   To obtain a value for π:

   (a) Use the `numpy.random.rand()` function to generate two arrays containing the x and y coordinates of 100 virtual raindrops. Note: `numpy.random.rand(N)` generates `N` random numbers from a uniform distribution between 0 and 1, or just generates one random number if `N` is not specified.

   (b) For visualisation plot the raindrops and draw the quarter circle (take care with the aspect ratio!). Note: keep the plotting separate from the calculation below.

   (c) Determine a logical test to decide whether each raindrop falls within the quarter circle and use it to count up how many raindrops fall within the quarter circle.

   (d) Use the ratio of the number of raindrops in the circle to the total number in the square to estimate π.

   (e) Run your program a few times and investigate how the estimate of π changes. Hint: you could make a function which takes N and returns π to make running it multiple times and for different values of N easier.

   (f) Does increasing N help with the accuracy?

   (g) Python is slow for loops. Instead of looping over every raindrop the logical test can be applied to all the raindrops at once using Numpy arrays and the `True` values counted with `numpy.count_nonzero()` function. When you have your program working using loops try it using Numpy vectorised operations and see if it is faster (you can time how long it takes to run any cell in a Python Jupyter notebook by putting `%%timeit` as the first line in the cell.)