Phys105-Week03-Student

November 13, 2020

1 Introduction to Computational Physics - Week 3

1.1 Table of contents week 3

1.2 Introduction to week 3

This week we will see how we can write functions using Python.

1.3 Functions

We saw last week that Python libraries like Numpy offer a large range of mathematical and other functions. Examples include `np.sin()`, `np.log()` and `np.linspace()`. If we give these the required arguments, they will return the value of the indicated mathematical function, define an array or carry out whatever job they are designed to do.

1.3.1 Week 3 exercise 1

Print out the sine of the angle 77° and the natural logarithm of 17.6, and use `np.linspace` to create a Numpy array that starts at 3, finishes at 7 and has 5 entries. Use the optional parameter in `np.linspace` that returns the step size and check that this is one! Print the values of $\sin(77°)$ and $\log_e(17.6)$ to a precision of two decimal places.

Hint Don't forget to import the Numpy library, and remember also that `np.sin` wants its argument to be in radians. Look up how to convert degrees to radians using Numpy!

Python allows us to create our own functions. For example, supposing we need a function to calculate the area of a circle from its radius, we can do this as follows:

[1]:
```
# <!-- Student -->
import numpy as np
#
def circleArea(radius):
    '''
    Calculate the area of a circle given its radius
    '''
    area = np.pi*radius**2
    return area
#
r = 0.2
A = circleArea(r)
print("Radius of circle",r)
print("Area of circle",A)
```

```
Radius of circle 0.2
Area of circle 0.12566370614359174
```

The start of the definition of the function is indicated by the keyword `def` followed by the name we have chosen for the function, `circleArea`. The arguments (inputs) to the function are given in brackets and the first line closes with a colon (:). The subsequent lines of code that belong to the function are indented, and the end of the function is indicated by the word `return`, followed by the values the function returns (outputs) to the routine that called it (which is the code in the cell below the function definition in this case). The indentation stops at the end of the function.

An aside - Python doesn't specify how much the lines in a function have to be indented, but the amount of indentation has to be the same for all lines. Python also doesn't specify whether the indentation is done using tabs or spaces (both of which are invisible!), but the same method has to be used throughout. It is good practice to pick an amount of indentation and how you achieve it and stick to it. In these Notebooks, I have used four spaces, which is the most common convention.

The function is *called* by using its name and giving it the value of the input parameter it needs. In the case above, this is done as follows:

```
A = circleArea(r)
```

We see that we pass the function the value `r`, which is given the name `radius` in the function. After the function has finished its calculations, the area is returned and given the name `A`.

Note two things: 1. We have used the value of π given by NumPy, `np.pi`. This is more precise and less prone to error (typos) than entering `3.1415926...` every time π is needed. 2. The first few lines of the function consist of a comment describing what the function does, using three quotes to delimit the description. This kind of comment (appearing after a `def` line) is called a *docstring* and can be picked up by Python's documentation system and provided to users of the function who need an explanation of what the function does. More about docstrings can be found here. Including docstrings in your functions is a good habit to get into!

Functions can take more than one argument and can return more than one result. For example, here is how we could calculate the area of a rectangle and the length of its perimeter, given its width and length.

[2]:
```
# <!-- Student -->
def rectangleParams(width, length):
    '''
    Return area and perimeter of rectangle given its width and length
    '''
    area = width*length
    perimeter = 2*(width + length)
    return area, perimeter
#
w = 0.2
l = 0.5
print("Width of rectangle",w)
print("Length of rectangle",l)
A, peri = rectangleParams(w, l)
print("Area of rectangle",A)
print("Perimeter of rectangle",peri)
```

```
Width of rectangle 0.2
Length of rectangle 0.5
Area of rectangle 0.1
Perimeter of rectangle 1.4
```

As in the function `circleArea`, the names used for the width and length of the rectangle can be different inside and outside the function. (Inside, we have used `width` and `length`, outside, `w` and `l`.) The function decides which is width and which is length using the order in which they appear in the call to the function. If we put the arguments to a function in the wrong order, the results will (usually) be wrong. (Of course, in the case of `rectangleParams`, the order doesn't affect the calculation of the area and the perimeter!)

The values of the variables `width` and `length` can be changed inside the function without affecting the values of `w` and `l` outside it.

The outputs of the function can also be given different names inside and outside the function. Again, Python decides which is which using the order in which they are given.

1.3.2 Week 3 exercise 2

Write a function that takes the width, length and height of a rectangular prism and returns its volume, its surface area and the total length of all of its edges. Print out the results of a test case.

1.3.3 Week 3 exercise 3

We have said that one of the advantages of using Numpy is that Numpy functions can take arrays as arguments and produce arrays as answers. Show this is the case by creating an array of angles (in degrees) starting at 0 and running up to 360 with a stepsize of one. Then use this to create an array containing the sine of all these angles. Print out both arrays, showing only two decimal places.

1.3.4 Week 3 exercise 4

Now plot the the value of the sines calculated in exercise 3 as a function of the angles. Plot the cosines of the angles in the same figure as the sines.

Hint: In the first week we saw that we could plot a line connecting the points with x coordinates `x_array` *and y coordinates* `y_array` *using the method*

```
plt.plot(x_coord, y_coord, linestyle = '-', marker = '', color = 'r', label = "Line")
```

1.3.5 Week 3 exercise 5

Write a function that calculates the sum of the sine and cosine of an angle (given in degrees). Using this function, make a plot that shows the sine, the cosine, and their sum as a function of angle.

1.4 Week 3 marks

Exercise	Mark	Comments
1	2	
2	2	
3	2	
4	2	
5	2	
Total	**10**	