Python 3 Tutorial 2: Writing a script

Anthony DeStefano

May 30, 2017

1 Creating Python scripts and user-defined functions

1.1 Running script using python3

Create a new file called my_script.py and open it with your favorite text editor e.g., gedit. In this file, import the numpy and matplotlib modules as we did in the previous tutorial by typing

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

Create a range of x-values from 0 to 100 in increments of 0.05 and define the y-values using a combination of predefined mathematical functions (you may need to search the numpy documentation about which functions are available). Plot y vs. x and have the script save the figure, with axes and title labeled, as an image and display it to the screen.

To run the Python script, in the terminal type

```
python3 my_script.py
```

This saves you time so you do not have to retype everything in the interactive terminal. *If there are any errors in your program, fix them before you proceed.*

1.2 Running script as an executable

We can also run the python script as a program in the terminal by including the following statement at the begging of the .py file

```
#!/usr/bin/python3
```

and by adding executable privileges to our script file by typing in the terminal

```
chmod +x my_script.py
```

To execute the python script, we type in the terminal

```
./my_script
```

1.3 Defining functions

Example 1: Create a new file called my_func.py. In this new file import the numpy module. To define a new function we type

```
def pow(x, n, A = 1):
    return A * np.power(x, n)
```

and save the file. Notice the third argument has a default value of 1, so if we only provide 2 arguments to our function, the value of 1 for A is assumed. The indentation in python is more than important, it is necessary for functioning code. The indentation defines the scope and makes it so curly brackets are unnecessary, compared to C/C++.

First, let us test our new function in the python3 interactive terminal by typing python3 in the terminal (be sure to be in the same directory as your file). To make our function available, type

```
import my_func as my
```

Be sure to import the numpy and matplotlib modules as well. Next, define a range of x-vaules from 0 to 10 with reasonable spacing and make an x-y plot of our new function. Try making a single figure with plots of different values of n and A. See Figure 1 for an example.

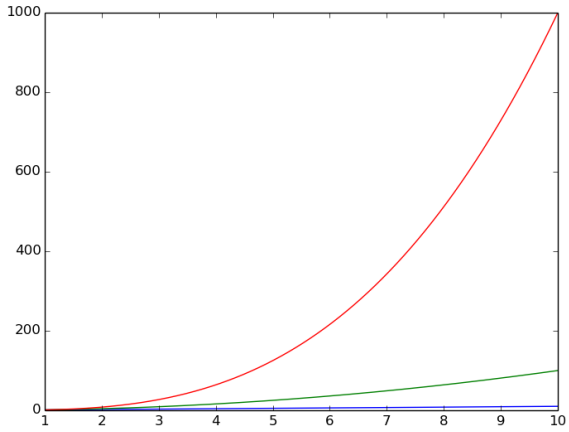

Figure 1: Here is an example of 3 calls to our pow function using $n = 1, 2, 3$ and using the default argument for A.

Example 2: Let us define another function in the my_func.py file that can plot 2D data (import the matplotlib module),

```
def plot2D(xmin, xmax, dx, ymin, ymax, dy):
    x, y = np.mgrid[slice(xmin, xmax, dx), slice(ymin, ymax, dy)]
    z = np.sin(2*np.pi*x*y)*np.cos(np.pi*y/3)
    plt.pcolor(x, y, z)
    plt.show()
```

If we make a call to the function my.plot2D(0,3,0.01,0,10,0.01), for example, we will get the following plot shown in Figure 2. One thing to note, in Python 3 if we divide by a number without a decimal (as in our definition for z) this will not produce integer division like in Python 2 or C/C++, but will produce a floating point division.

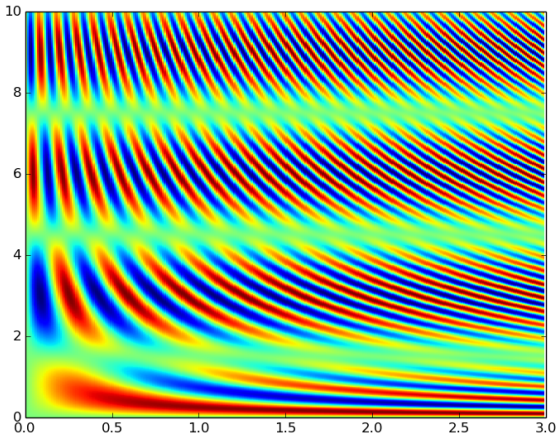

Figure 2: An example of our plot2D function using xmin $= 0$, xmax $= 3$, dx $= 0.01$, ymin $= 0$, ymax $= 10$, and dy $= 0.01$.

In order to use a user-defined function in an executable python script, just import the file name without the file extension as we did in the first example above.

2 For loop and if-else statement

Loops are useful for manipulating a list (array) of data one element at a time. In Python, it is strongly advised to avoid for-loops as much as possible (they are very slow!), but sometimes they are unavoidable. Consider the following program head_tails.py:

```
#!/usr/bin/python3

import numpy as np

n = 1000
data = np.random.rand(n)
heads = 0
tails = 0
for i in range(0,n):
  if data[i] > 0.5:
    heads += 1
  else:
    tails += 1

print('fraction of heads = ', heads/n)
print('fraction of tails = ', tails/n)
```

3 Task: Write a program to generate a histogram

A histogram is a way of sorting and describing data. For instance, if there are 10 people in the room and you sort them into ages (three 20 year olds, four 21 year olds, and three 22 year olds) you have made a histogram. In this case, the input of the program would be the ages of the people, i.e. ages = [20, 21, 22, 21, 20, 20, 21, 22, 22, 21], and the output would be the array that describes the ages (bin = [20,21,22]), and the numbers of people in each age bin (hist = [3, 4, 3]).

In your program you should have at least 1 input (e.g., ages) and at least 2 outputs (e.g., bin and hist). The other thing you have to consider is the size of the bins. For instance, what if your bin was in 10 year increments, bin = [10, 20, 30]? All the ages would be in the 20 bin, such that hist = [0, 10, 0] and your histogram would be very boring. Alternatively, the bin size could be too small, you could have increments of days or even minutes. In that case, the bin array would be huge and there would be one person in each one bin so hist would be mostly 0's with ten 1's. Therefore, binsize could also be an input to your program or you could figure our a way to make the program calculate the correct bin size. **Avoid using the built-in numpy function hist**. The goal here is to understand how to create a histogram program for yourself.

Once you have written your histogram program, try it out on some real data (from the first tutorial). Make a histogram of the observed sunspot number that was seen for each month. Write the plot into an eps file and send it to jacob.heerikhuisen@uah.edu along with your histogram program and the python script you used to call your histogram function.