OADnotes

Unknown Author

September 02, 2013

1 Day 1

1.1 What is Programming?

Programming is when you give a computer a set of instructions you want it to carry out. In essence computers are stupid (at least for now)! It is us human that need to guide them.

1.2 Why programming is helpful then!?

To simplify difficult problems or automate processes.

E.g of programs - Word - to write letters, thesis etc

- apps - weather apps takes weather information from the meteorological

station and display them live on your smartphone

- calculator - makes our lives easier! we all know how to add, subtract, multiply, divide, etc but its much easier to use a calculator and you are sure the answer given is the right one too!

Lots of the time it is problems we can solve, if we are given enough time, but most of those task could be cumbersome.

1.3 Why programming is important in the scientific world?

Help us solve tedious problems that would otherwise take days to calculate. In the scientific world today, the amount of data we collect is huge, and without computers it would be impossible to deal with the data. In case of the Square Kilometer Array, it will produce 1 petabyte of data in 1 day: How could we possibly deal with that amount of data without using super computers. Similarly in DNA research, the sequencing occupies hundreds of gigabytes of data and only with the help of computers can we hope to handle and find pattern in the data.

1.4 Why Python?

There are so many programming languages out there and each one has its own advantages and offers something different to the users. Among the most commonly used languages are: C/ C++, Fortran, Python, Java, Matlab, IDL, Scilab Perl, etc… So why choose to do a course on python? What advantages does Python offer over the others : Python is an open source software and therefore already offers a big advantage over packages such as Matlab or IDL. It is one of the fastest growing languages and enjoys tremendous support from the open source community and when Github (website for python developers) came online in 2010 the amount of people involved in python developing just sky-rocketed. Now why is python so convenient to do science and especially why is it use so much in astronomy?

1. Python is much easier to use than C or Fortran and it has a more direct approach (high level). Therefore you will spend less time coding and more time will be dedicated to research.
2. It is an object oriented language
3. Python provides a great amount of flexibility and remains as powerful as C or Fortran.
4. Python is very modular and has capabilities to run C, Fortran, Java, IDL codes inside itself.
5. Python is widely popular among the open source community. Your problems will not remain unsolved for long. Search engines will surely return a favourable answer to your query. Bear in mind that any problems you encounter most probably has already been encountered from someone in the community. Just Google it!

The main issue concerning Python is that it is slower than C or Fortran. Running huge simulations is not feasible. But if your main interests lie in solving statistical problems or fitting best fit lines to some data and plotting them, then Python is the place to be!

This set of notes is not an original piece of work. But rather a tweaked version of everything I have as literature on the subject. My sources vary from search engines, YouTube and from Luis Balona's NASSP lectures. It will be a steep learning curve but that is the case for all crash courses. But hopefully after the course, it will enable you to work in Python, plot and fit data and more importantly be able to read any online Python guide to teach yourself other techniques that you might be interested in.

So now is the time to start your journey in the python world . . .

"Hold on to your Hippogriff" would Hagrid have said and "If in doubt, Meriadoc, always follow your nose" would Gandalf the Grey tell you!

1.5 Introduction

You are now in ubuntu 12.04, on your left hand side you will find a sidebar which contains some quick launch icons to some important programs. Click the little black box will give you a terminal. In the terminal type 'python' and you should get something like this:

```
rajin@rajin:~$ python

Python 2.7.3 (default, Aug 1 2012, 05:16:07)

[GCC 4.6.3] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>
```

This will start python and from then on you will be inside the python prompt - The python version will be written up - in case of ubuntu 12.04 the default python is 2.7.3

If you want to exit python just type 'exit()' or press ctrl+D

```
In [1]: 3+5
```

```
Out [1]: 8
```

```
In [2]: (2.7 + 3.1416)/22.73
```

```
Out [2]: 0.2569995600527936
```

So you have used Python as a simple calculator. How about this:

In [3]:
```
xx = 4
yy = 3.6
xx+yy
```

Out [3]:
```
7.6
```

In [4]:
```
(2.1*xx + 6*yy)/7.2
```

Out [4]:
```
4.166666666666667
```

Some Explanations:

Data types

Computer programming language can understand 3 types of data:

- Numbers
- Letters and numbers
- Non letter or numbers

Numbers :

Integer - integers e.g 1,4,6,100, -10

or Floats e.g: 1.5,2.8,4.78

letters and number stuffs

Strings - whole words and word/ number mixtures. usually you need to put strings in between single quotes ' or double quotes " e.g 'New' , 'I like sunny days' , '6eggs' , "Hi5"

Non letter or numbers

Boolean - either True or False

Variables

A variable is a word/identifier that hangs onto a single value. For example, let's say you needed the number 6 for your program, but you're not going to use it immediately. You can set a variable, say Constant, to grab the value 6 and hang onto it for later use, like this:

In [5]:
```
Constant = 6
```

Also do NOTE that in python the variable 'Constant' is not the same as 'constant' or 'CONStant' or any other combination of upper case and lower case. In some other programming language this is not the case.

Statements

You can think of a Python statement as being similar to a sentence in English: it's the smallest unit of the language that makes sense by itself. Just like "I," "like," and "Spam" aren't statements by themselves, but "I like Spam" is, variables and data types aren't statements in Python, but they are the building blocks that form them. To continue the sentence analogy, it's clear that we also need a kind of punctuation to make it obvious where one statement ends and another begins.

Getting back to our work how about this:

```
In [6]: 19/5
```

```
Out [6]: 3
```

What's happening here? The answer should be 3.8. Actually, the answer is correct for integer arithmetic. Since there are no decimal points present, Python assumes that both 19 and 5 are to be treated as pure integers. The answer must therefore also be an integer and 3.8 is truncated to 3. If you wanted the numbers to be treated as real, then make sure at least one of the numbers has a decimal point :

```
In [7]: 19.0/5
```

```
Out [7]: 3.8
```

NOTE: It is recommended that you take a habit to always use decimal points in everything you are coding unless you really need integer manipulation

Here is how to obtain the power of a number (**) and the remainder after division (%):

```
In [8]: 2**8
```

```
Out [8]: 256
```

But following the advice given, you should be doing the following instead:

```
In [9]: 2.**8.
```

```
Out [9]: 256.0
```

```
In [10]: 19 % 5
```

```
Out [10]: 4
```

Here is how you get the absolute value of a number, the integer part, and the decimal part:

```
In [11]: abs(-8.34)
```

```
Out [11]: 8.34
```

```
In [12]: int(8.34)
```

```
Out [12]: 8
```

```
In [13]: 28.34 % 1.0
```

```
Out [13]: 0.33999999999999986
```

It is important to notice that above, we have a few ‘in-built’ functions of python i.e abs() and int(), when you provide those functions with some input inside the brackets, they perform some calculations that they have been programmed to do and then output the value desired.

Now time to go onto some string manipulations

```
In [14]: 'alpha'+'beta'
```

```
Out [14]: 'alphabeta'
```

Some more manipulations:

Now say that you stored the value 10 in the variable *new*. If you want to convert *new* to a string then try:

```
In [15]: new = 10
         str(new)
```

```
Out [15]: '10'
```

Now if you want to add *new* as a string to another string you can do it.

```
In [16]: 'alpha'+str(new)+'beta'
```

```
Out [16]: 'alpha10beta'
```

As you guessed str() is also another in built function. Another extremely useful in built function is len() which tells you the lengths of strings, list and arrays (more on arrays and lists tomorrow). So lets try to see how to use it:

```
In [17]: random_string = 'alpha'+str(new)+'beta'
         random_string
```

```
Out [17]: 'alpha10beta'
```

```
In [18]: len(random_string)
```

```
Out [18]: 11
```

If you want to convert from integer to float:

```
In [19]: a = float(14)
         a
```

```
Out [19]: 14.0
```

This works even with strings i.e in the case you have the string ‘15.4’ and wants to convert it back to a floating variable then:

```
In [20]: float('15.4')
```

```
Out [20]: 15.4
```

Code Commenting

One of the most important part in coding is to comment what you are doing. Just imagine that sometime you write programs which could be hundreds or thousands of lines long. It wont be easy if someone else needed to use your code, or most of the times, you will forget what you did when you get back to the code you wrote a couple of weeks after writing it. Therefore it is essential to add complementary comments to make it as clear as possible what is happening in the program. In python we use the hash i.e '#' to start commenting. Usually, on any line, whatever is AFTER the '#' is commented and therefore not processed when the computer is running the code. e.g

```
In [21]: Constant = 1.9  # Defining a variable and setting a float value with it
```

Sometime you need to write long explanations of what is happening in your code and therefore it is not practical to put a hash on each line. In those case you need to put 3 quotation marks at the beginning of the comments and end it again with 3 quotation marks:

```
In [22]: ''' We want to write a simple program to perform simple
         mathematical calculations like addition, subtraction,
         multiplication and division below is the code '''
         alpha = 1.2
         beta = 4.5
         delta = alpha + beta
         phi = alpha - beta
         theta = alpha * beta
         mu = beta / alpha
```

It is very good practice also to always define variable with as explicit names as possible. In this way you dont need to comment a lot to explain the meaning of each variable.

Python scripts

There are 2 ways to run programs usually (especially in python), either you do it interactively as we have been doing all along until now or you write scripts that contain several lines and can then be run entirely in one go. Usually when you are doing some trial and error you use the interactive part to make sure you are using the right syntax. Scripts are more useful for longer work and you can save scripts so as to go back to them later for modifications and improvements. Usually scripts are just python instructions that are read one line after the other. To begin with, lets start by opening a text editor in Ubuntu -> try using 'gedit' text editor and then type in:

age = raw_input("What is your age?")

print "Your age is",age

Save this script as test.py in the folder 'day1' . Then in a terminal type :

cd day1/

python test.py

The result will be something like:

What is your age? 22

Your age is 22

Let us examine this script. First you have a raw input(message) which prints a message and waits for you to type in a character string. The character string is assigned to the variable *age*. Then there is the print string which prints the character string. In this case the character string is the message in quotes onto which is appended the value of *age*. Note the comma between the two parts of the string. The main point to note here is that raw input always returns a character string. A character string is any combination of characters such as Abc, ghtH%$, 1234, etc. Although the string 1234 looks like a number to you, Python will simply regard it is just another character string unless you convert it to a number with the method already described i.e to convert to integer use int() and to floating point use float(). Below is a small script to convert degrees Fahrenheit to degrees Celsius:

# Converts Fahrenheit to Celsius.

Fahrenheit = float(raw_input("Temperature in Degrees Fahrenheit?"))

Celsius = (Fahrenheit - 32.0)/1.8

print "Degrees Celsius =",Celsius

As you can see, the raw_input is converted to a float so that we can then do some basic calculation on top of it. So save this script into a text file (into a file name temperature.py) and then run it a couple of time to see what results you get.

As a practice example, try checking online for the exchange rate of Rand to US Dollar and write a short script to convert between Rand and Dollar

The output you get out of by using simple print statements are unformatted and if you wish to play around to get different types of display when using the print statement, then check the script below

In [23]:
```
x = "Hello dear!"
i = 123
j = 99
y = 123.45678912345
print "d - an integer: %d" % (i)
print "f - a floating point number: %f" % (y)
print "6.2f - a floating point number: %6.2f" % (y)
print "6.1f - a floating point number: %6.1f" % (y)
print "e - a float in scientific notation: %e" % (y)
print "8.4e - more scientific notation: %8.4e" % (y)
print "s - The value of my string is %s" % (x)
print "int with 4 characters space: %4d %4d" % (i,j)
```

```
d - an integer: 123
f - a floating point number: 123.456789
6.2f - a floating point number: 123.46
6.1f - a floating point number:  123.5
e - a float in scientific notation: 1.234568e+02
8.4e - more scientific notation: 1.2346e+02
s - The value of my string is Hello dear!
int with 4 characters space:  123   99
```

Importing Packages

If you need to do something quickly but can't find any in-built function to do the task there are only 2 solutions available to you: write you own function or call for an external function which is available inside other packages (that do not come in with the standard python installation). This is one of the main feature of python; the ability to call for packages which do not come in the default installation. Once you have imported the package, then you can call for the functions inside that package almost like the default inbuilt function. For e.g, if you want to do trigonometric calculations like sine or cosine, there are many packages can accomplish that for you. One of the most extensive package of python is SciPy – scientific python. Lets try to use it

To import scipy we do the following:

```
In [24]: import scipy
```

Now scipy has been loaded in the python running inside the terminal. We can now try some calculation with it

```
In [25]: scipy.cos(50)

Out [25]: 0.9649660284921133
```

As you can see, to make python aware we are using a package outside its own in-built package we need to specify the package everytime we use it like here we just had to write scipy.cos(). In some cases some external packages name are quite long and it is not convenient to write it everytime you are calling a function from that package. A shorter way to do this is to import it in the following way

```
In [26]: import scipy as sc
```

And then you can use the package in the following way

```
In [27]: sc.sin(10)

Out [27]: -0.5440211108893698
```

It is much easier now to call scipy and much more convenient.

NOTICE that by default python (and scipy) will be in radians mode and will compute anything in radians. Therefore when giving the arguments '50' and '10', the computer believes it is 50 and 10 radians respectively To work in degrees you can do the following:

```
In [28]: deg = sc.radians(30)
         sc.cos(deg)

Out [28]: 0.8660254037844387
```

where the first command 'deg = sc.radians(30)' converts 30 degrees into radians

A quicker way would be:

```
In [29]:  sc.cos(sc.radians(30))

Out [29]: 0.8660254037844387
```

Below is listed some of the mathematical functions available with scipy

- exp(x): Returns e x.
- log(x): Natural logarithm loge (x).
- log10(x): Base-10 logarithm log10 (x).
- sqrt(x): Return square-root of x.
- cos(x), sin(x), tan(x): Trig functions (x in radians).
- arccos(x), arcsin(x), arctan(x): Inverse trig functions (the output will be in radians)
- radians(x): Returns radians if x in degrees.
- degrees(x): Returns degrees if x in radians.

A more extensive list of functions is available at (http://docs.scipy.org/doc/numpy/reference/routines.math.html). Do check it!

A seemingly very convenient approach of importing would be:

```
In [30]: from scipy import *
         #this import all modules and function from scipy in one go
```

With this type of import, the advantage then, is that you can call any function without putting any prefix, i.e :

```
In [31]: cos(10)

Out [31]: -0.83907152907645244
```

```
In [32]: degrees(0.5)

Out [32]: 28.64788975654116
```

You will most probably encounter this method online, in tutorials and a lot of people use this approach. This method is fine when you are writing small codes (below 20 lines) . But the big problem with this method is when you start importing different packages which contains similar functions. For e.g there is a package called MATH which does all the trigonometric functions that SCIPY does. Say in a code you do:

```
In [33]: from math import *
         from scipy import *
```

then when computing a cos or sine of an angle, the computer can get confused with which package to use in the calculation. To avoid that, the best approach is to just do the usual

```
In [34]: import math as mth
         import scipy as sc
```

This method also offer another advantage in that everytime you type ‘sc.’ and press ‘tab’ on your keyboard, ipython will display a list of all the functions in scipy.

Exercise Cape Town has longitude λ_1 = 18.4, latitude $\phi_1 = -33.9$ degrees. The corresponding position of Johannesburg is λ_2 = 28.1, $\phi_2 = -26.2$ degrees. The angular separation of the two cities relative to the Earth’s centre, θ, is given by

$$cos\theta = sin\phi_1 sin\phi_2 + cos\phi_1 cos\phi_2 cos(\lambda_2 - \lambda_1). \quad (1)$$

The Earth’s radius is 6357 km. Write a script to calculate the distance between the two cities. Your output should look like this:

Distance between CT and JHB is 1262.7 km

```
In [ ]:
```