Introduction and Tutorial for Python

Arpad Miskolczi

Astronomisches Institut
Ruhr-Universität Bochum

RUB

13.02.2012

Python is a general-purpose high-level programming language. It started in the late 1980's.

It has a very clear syntax and is easy to learn, especially if you have experience in other languages.

The usage of python steadily increases in science, in entertainment and even in the gaming industry.

CASA is based on python and uses the same syntax

Python source code is saved in .py file, it is a simple text file. A python script is executed by typing „python yourfile.py"

Using ipython, one can play around in an interactive python shell.

Ipython can be installed from your linux distributions repository, or downloaded from http://ipython.org/ (also for windows available)

Python has its own interactive shell, but it is a bit uncomfortable. Ipython provides more functions, for example code completion on pressing „tab".

Ipython is started by typing „ipython" into your unix terminal.

First Python Steps: „Hello World"

```
print 'hello world'
```

Important note: Python is case sensitive!

We see „print", the command to print something to the terminal, and „hello world", the string to be printed.

String is a data type, Python knows several data types:

String – A string, a collection of seperate characters. For example „Hello"
Float – A floating point numerical type, for example: 0.123456789
Int – An integer numerical type, for example: 1
Boolean – Is a variable that is either True or False

Python supports dynamic typing:

a=5 (a is an int)
a='b' (a is the character b, or a character sequence only containing 'b')
a=True (a is a boolean) and so on.

print type(a) will print the datatype of a variable
A variable with an assigned data type is also an object, which means it has several built-in functions.

Built-in functions can be accessed with a „." (dot) after the variable, we'll see some examples later.

Data structures:

A sequence is a structure to store multiple values of the same type.
A String variable is actually a sequence of single characters.

Sequnces (in python called „list") are defined by []. For example:

```
A=['a','b','c','d','e']
```

One can access every index of the sequence:

```
print A[2]
```

← **note: first index is 0, not 1**

Two lists can be concatinated:

```
a=[1,2,3]
b=[4,5,6]
c=a+b
```

To add another element to the list one can use the append function of a list:

```
a.append(5)
```

You can also append data of another type:

```
a.append('something')
```

Hint:

When typing a. and then pressing „tab", ipython shows a list of possible functions! Great to play around and experimenting.

Dictionaries:

A dictionary is a data structure in which you can assign a certain value to a certain entry, and can lookup each element seperately.
Defined by:

```
constants = {}
```

New elements are not appended, just added:

```
constants['speedLight']=299792458
constants['Hubble']=70.5
constants['parsec']=30.856776e15
```

Dictionaries consist of keys and items.

```
Constants['lightyear']=9.461e15
```

Data is accessed like in a real dictionary, by looking it up!

```
print constants['Hubble']
```

One can see the items or keys:

```
print constants.keys()
print constants.items()
```

Simple numerical operations:

```
a = 5
b = 10
c = a+b
c = a-b
c = a*b
c = a/b     ← if a and b are integer types, c will be also an integer! If you want
              c to be a floating point number, at least a or b have to be a float
              number, for example: a = 5.0
c = a**2
c = ((3.14159*a)-b)/7
```

„if" - statements:

„If variable has a value of x, do this, if not, do something else" in python language means:

```
a = 5
if a==5:
    print 'a is five'
else:
    print 'a is not five'
```

There are three things here to notice:

1. An If statement (and any other statement) is always ended with „:" (so is the else part of it)
2. Everything in the executed block is indented , one empty character („space") is enough, I prefer a „tab" for better readability.
   In other languages, like in JAVA or C, each block is surrounded by {...}

In java, the statement above would be:

```
if (a==5){
        Sytem.out.println(“a is five“);
 }else{
        System.out.println(“a is not five“);
 }
```

3. An if statement evaluates an expression. Comparisons are always done with two equal signs.

```
if(a==5)    compares a to 5
if(a=5)     sets a to 5!
```

But fortunately, python will throw an error if you use just one „=" in a statement

Boolean operators:

Using boolean operators like „and", „or", „not" or some combinations of them, one also construct more complex expressions:

```
if ((a==5) and (b>10)) or (a==b>=c):
      print 'some task'

a = True
if not a=False:
      print 'some task'
```

Loops:

Python knows two loop structures, the „for" loop, and the „while" loop.

A for-loop, in principle, tells python to do some task, for every item in a certain range or sequence.

For example: printing the square of every number from 1 to 100:

```
for a in range(1,101):
    print a**2

for a in [1,2,3,4,5]:
    print a**2
```

← note: range(x,y) includes x, but excludes y
so that it creates the sequence
x, x+1, x+2, …, y-1

A 'while'-loop executes a task until some condition is met. If you are not careful, this can lead to an infinte loop.

```
a=0
while a < 5:
    print a
```

This leads to an infinite loop, because a always remains 0, there is no command within the loop to increase a.

```
while a < 5:
    print a
    a+=1
```

Will execute the task 5 times until a is 5

To exit a loop, there is the *break* command

```
for a in range(0,10):
    if a==5:
        break
    print a
```

this will print the value of a until it reaches 5

There is also the *continue* command, which skips the current iteration of a for loop

```
for a in range(0,10):
    if a==5:
        continue
    print a
```

This can be helpful to avoid infinite loops, for example, if one wants to measure how long the loop is running and exit it after a certain amount of time.

```
import time

tstart = time.time()
while 1==1:
    print 'infinite loop'
    tend = time.time()
    if tend-tstart>60:        ← needs to be larger than 60, because it will never
        break                    be exactly 60
```

Modules:

On the last slide the time module was imported

A module is collection of functions

to get a list of functions of a module, type the module's name in ipython, followed by dot, and then press tab, or look into the module's API

Modules are imported using the *import* command

One can also just import one function from a module using

from module import function

It is also possible to give the module, or the imported function, another name using „as", for example:

```
from pyfits import getdata as gd
```

Some modules that you will most likely use:
os, sys, time, numpy, matplotlib, pyfits....

Some standard modules like os, sys, time (and a lot more) come with python, others have to be installed by the user.
On Linux, this can be done using the local package manager, if the package is available..

Writing your own functions:

One can also write a function to be used. This is very useful if you have to do the same task over and over again and do not want to write the lines every time.

This is done using „def“:

```
def name():
```

A function always needs brackets and a colon.

Within the brackets, you can give the function an argument. For example a variable which needs to be processed.

Then follows the task of the function.

If a function is designed to process a variable and you want to store the result, you can end the function with „return" followed by a variable.

```
def square(x):
        value = x**2
        return value
```

A function can be assigned to a variable or directly printed.

```
a = square(5)
print square(5)
```

This would also work if you ommit the „return", but it would print „None", instead of 25, since no value is actually returned.

A function has to be defined within the source code, BEFORE it is used for the first time!

```
print square(5)
def square(x)
    return x**2
```

Will not work!

Reading from, and writing to text files:

At one point you most certainly will have to read in values from a text file, or will want to write values to a text file.
This can easily be done

```
inputfile = open('file.txt','r')
```

sets the file name and the 'r' tag which indicates that the file is only beeing read from

to read the file line by line:

```
for line in inputfile:
    print line
```

Data read from a file in this manor always yields a string

if you need actual numbers, you need to process the string:

line = „599;14.5;1.2;-0.7;0“

use „split“ to break this string at certain points:

numbers = line.split(';') Will create a list where each element is a substring

to convert a string to a number, use „casting“

```
StokesI =   float(numbers[1])
StokesQ = float(numbers[3])
StokesU=  float(numbers[2])
StokesV = float(numbers[4])
```

Try casting to an „int“

Reading from, and writing to text files:

to write to a file, you need to open the file using 'w' tag

```
outputfile = open('test.txt','w')
```

Beware: file will be completely overwritten!

```
outputfile.write('Somestring'+'\n')
outputfile.flush()
```

'\n' tells python to write a new line

Writing to a file is not done immediately, everything is written to a buffer. The buffer is then written, if you close the file using outputfile.close() or if you flush the buffer manually. If you dont use this, and your script crashes, it can happen that nothing is written to the file because the buffer was not flushed.

Numpy

Numpy is a very useful package for doing calculations on matrices

Matrices, here, are called arrays, which, in basic, are a sequence of sequences.

You can create a numpy array by creating an array filled with 0 or 1:

```
import numpy
array = numpy.ones(shape=(50,50))
randomarray = numpy.random.randint(0,100,(50,50))
```

Elements in an array are accessed like in sequences, by specifying the index for each axis

Once you have an array, numpy provides a lot of functions to apply to the array.

```
print randomarray.max()
print randomarray.min()
print randomarray.mean()
print randomarray.std()
print numpy.median(randomarray)
```

Numpy also provides the means to do calculations with the matrix, like adding two matrices.

```
new_array = randomarray+array
new_array = randomarray-array
```

Will add/subtract each element from one array, to/from each element of the other array

Numpy can also work with higher dimensional arrays

```
Narray = numpy.ones(shape=(10,10,10,10,10))
```
creates a 5 dimensional array

Hint:
If you need do process a lot of array elements, try to avoid for (or even nested) for loops, they are much slower than numpy functions.

```
array = numpy.arange(2000**2).reshape(2000,2000)
for x in range(0,2000):
    for y in range(0,2000):
        array[x][y] = array[x][y]**2
```
takes about 7 seconds

```
array = array**2
```
takes about 0.02 seconds

Useful links:

Numpy
http://numpy.scipy.org/

Python API
http://docs.python.org/c-api/

AstroPython
http://www.astropython.org/

Matplotlib Documentation and examples
http://matplotlib.sourceforge.net/