Physics 514 – Basic Python Intro

Emanuel Gull

September 6, 2016

1 Python Introduction

Download and install python. On Linux this will be done with `apt-get`, `evince`, `portage`, `yast`, or any other package manager. On OSX the preferred method is `macports`. If you take Prof. Leonard Sander's class please consider installing `vispython`. On Windows you will need to download and install the windows package e.g. from `www.python.org`

CATS will help you with the installation if you get stuck: a homepage with a windows installation bundle is linked here, and you can visit them (near the mailboxes). The lecture homepage has additional information.

1.1 Intro – Getting started

Please start by running 'python' on the command line. You should see something like this:

```
egull$ python
Python 2.7.3 (default, Apr 19 2012, 00:55:09)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The prompt tells you which version of python you have. Typical versions are `Python 2.5.6`, `Python 2.6.8`, or one of the 2.7 branch. If you have a python version ≤ 2.4 or ≥ 3 please install one of the 2.4to 2.6 branch.

Exit Python again:

```
>>> quit()
egull$
```

You are only allowed to use a programming language after saying politely 'hello' to the world:

```
>>> print "hello, world!"
hello, world!
```

Advanced: compare this program to the hello world program in `C`, `C++`, the `FOR`mula `TRAN`slator, `perl`, and your other favorite programming languages.

1.2 First Steps

We just used python as an interactive interpreter: there is a prompt (the `>>>`) waiting for you to input data. This is sometimes useful, e.g. to use it as a calculator. Let's get started:

```
>>> 1+1
2
>>> 2*8
16
>>> 2**4
16
>>> a=5; b=3; a+b
8
>>>
```

Usually we want to create scripts: instead of typing commands at the prompt (the `>>>`), we write them into a file and then 'execute' them. Here is an example: write the file `hello.py` and execute it:

```
egull$ cat hello.py
print "hello, world"
print 1+1
egull$ python hello.py
hello, world
2
egull$
```

Traditionally there were two main classes of languages: *interpreted* languages (examples are `sh` and variants, `perl`) and *compiled languages* (`C`, `C++`, `java`). Interpreted languages 'interpret' the code at runtime (i.e. translate the code to machine language), whereas 'compiled' languages have an intermediate step: a 'compiler' takes the 'source code', 'compiles' it to machine language, and a 'program' is created. Here is an example:

```
egull$ cat hello.cc
#include<iostream>
int main(){ std::cout<<"hello, world"<<std::endl;}
g++ -o hello hello.cc
egull$ ./hello
hello, world
egull$
```

`./hello` executes the program `hello`. In python, no compilation is required, but even though python looks like an interpreted language it is actually compiled using a just-in-time compiler. This makes python substantially faster than interpreted languages, though not quite as fast as `C++` or `C`.

2 Python Object Types

Python knows several data types. The built-in types are `Numbers`, `Strings`, `Lists`, `Dictionaries`, `Tuples`, `Files`, along with some less frequently used ones.

2.1 Numbers

Python knows integers, floats, and complex numbers. Depending on the problem they are stored with different precision:

```
>>> 111+222
333
>>> 1.5*2
3.0
>>> 3%2
1
>>> 2*100
200
>>> 2**100
1267650600228229401496703205376L
>>> 2.**100
1.2676506002282294e+30
>>> 3+5j
(3+5j)
>>>
```

Note that the sign for the imaginary number is `j` (commonly used in engineering), not `i` as used in science.

We will often need mathematical constants. For this we will need to 'import' the `math` library (more about importing and libraries next time):

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

Please do not compute mathematical constants using arithmetic operations (`pi=4*math.atan(1)` and the like). Apart from it begin slow and bad style, you also don't get the full precision:

```
>>> format(math.pi, ".32f")
'3.14159265358979311599796346854419'
```

The math library we just imported has a range of really useful functions. For complex numbers also import `cmath`.

```
>>> cmath.sqrt(-1)
1j
>>> math.sqrt(2)
1.4142135623730951
>>> cmath.log(math.e)
(1+0j)
```

To the contents of a module like `math` you can use `help`:

```
>>> help(math)
>>> help(cmath)
```

’q’ will get you out of the help screen; ’space’ will show the next page, etc. Note that for linear algebra and vector/matrix operations we will introduce the `numpy` and `scipy` modules next week.

2.2 Strings

Python provides a powerful string class. Here are some elementary operations:

- strings are defined with either single or double quotes:

  ```
  >>> S=’Spam’
  >>> S
  ’Spam’
  >>> S="Spam"
  >>> S
  ’Spam’
  ```

- There are a range of string operations which look similar to what you may be used to from matlab. Important: array addressing (or matrix addressing) is very similar! First, second, last element and the string length:

  ```
  >>> S
  ’Spam’
  >>> S[0]
  ’S’
  >>> S[1]
  ’p’
  >>> S[-1]
  ’m’
  >>> S[-2]
  ’a’
  >>> len(S)
  4
  >>> S[len(S)-1]
  ’m’
  ```

Note that similar to `C` and `Java` but different form `Pascal` the string addressing is done from element `0` to element `len-1`.

- Slicing of a string (i.e. taking part of it) can be done in a few ways, all of them involving a colon (:) to separate the beginning from the end. If no beginning/end is specified the entire range is taken:

```
>>> S[:]
'Spam'
>>> S[:2]
'Sp'
>>> S[2:]
'am'
>>> S[1:3]
'pa'
```

- The string type also has a couple of 'arithmetic' operations for concatenation defined on it:

```
>>> S*4
'SpamSpamSpamSpam'
>>> S+"xyz"
'Spamxyz'
```

- ...as well as some other useful methods:

```
>>> S.find('p')
1
>>> S.replace('pa', 'gugu')
'Sgugum'
>>> S.split('a')
['Sp', 'm']
```

2.3 Lists

Python 'Lists' are, as the name says, lists of objects. Lists are positionally ordered, so similar to e.g. the `C++` vectors. However, lists can contain different objects. Note that while lists are useful, they should not be used for vectors of numerical data. We will use `numpy arrays` for this. There are a number of reasons for this, which we will get into next week.

- Here is an elementary list with an integer, a string, and a float:

```
>>> L=[1, 'abc', 1.23]
>>> len(L)
3
>>> L[0]
```

```
1
>>> L[1:3]
[’abc’, 1.23]
>>> len(L)
3
>>> L+[4,5,"6"]
[1, ’abc’, 1.23, 4, 5, ’6’]
>>>
```

- Whereas previous objects were immutable, lists are mutable. Here is the result of a sorting operation:

  ```
  >>> M=["aa","bb","dd","cc"]
  >>> M.sort()
  >>> M
  [’aa’, ’bb’, ’cc’, ’dd’]
  >>>
  ```

  Note that the content of M was changed by the `sort` operation. Similarly, we can append, replace, reverse, pop, etc (have a look at `help(list)` and `dir(list)`

- We can construct nested lists (‘lists of lists’). You could imagine creating matrices like this:

  ```
  >>> A=[[1,2,3],[4,5,6],[7,8,9]]
  >>> A
  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  >>>
  ```

  Note that this is not how matrices of numbers should be stored (again, see `numpy`’s `array` class next week). *Advanced: discuss why not.*

2.4 Dictionaries

Dictionaries provide a map between a `key` and a `value` pair, they are what in other languages is called a map. Here is an example:

```
>>> D={’food’:’Spam’, ’quantity’:4, ’color’:’pink’}
>>> D[’food’]
’Spam’
>>> D[’location’]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: ’location’
>>>
```

Like the lists, dictionaries are mutable:

```
>>> D['quantity']+=1
>>> D['quantity']
5
```

And they have a range of functions defined on them. Again, see `help(dict)` and `dir(dict)`.

3 Exercises

1. Familiarize yourself with python! Have a look at the e-book and read through chapter 4. Solve the quizzes at the end of the chapter.