A brief Python/IPython crash course

This document will give you a brief tour of the features of the Python programming language and of the capabilities of the IPython notebook.

You can view its contents by scrolling around, or execute each cell by typing `Shift-Enter`.

The basics

You can use Python as a plain calculator:

```
In [1]: (1 + 1) * 3.14 / 2 - 1

Out[1]: 2.14
```

NOTE: exponentiation is represented by the ** operator (**not** the ^ operator):

```
In [2]: 2**4

Out[2]: 16
```

Strings are represented with single `'` or double `"` quotation marks:

```
In [3]: 'hello world'

Out[3]: 'hello world'

In [1]: "hello world"

Out[1]: 'hello world'
```

Types

Python is a **typed** language:

```
In [4]: type(3)

Out[4]: builtins.int

In [5]: type(3.)

Out[5]: builtins.float

In [6]: type('hello world... again')

Out[6]: builtins.str
```

When numerical objects of different type interact, type conversions take place:

```
In [7]: type(3 * 3.0)

Out[7]: builtins.float
```

Variables, blocks and control structures

Assignment of variables is straightforward:

```
In [8]: x = 5
        s = 'witty remark here'
        print(x)
        print(s)
```

```
5
witty remark here
```

You can assign multiple variables at the same time (even with different types):

```
In [9]: x,y,z,s = 1, 2, 3.2, 'possibly a monty python quote'
        print(z)
        print(s)
```

```
3.2
possibly a monty python quote
```

Blocks are indicated through indentation, and **only** through indentation (no BEGIN/END or braces). As a convention, from now on we will use **four spaces** as indentation marker:

```
In [10]: x = 2
         if x < 2 or x > 2:
             print('no good')
         else:
             print('OK')
```

```
OK
```

```
In [11]: x = 10
         while x >= 0:
             print('x is still not negative')
             x = x-1
```

```
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
x is still not negative
```

The common way of writing `for` loops in Python uses the `range()` function:

```
In [12]: for i in range(0,5):
             print(i)
```

```
0
1
2
3
4
```

NOTE: the assignment operator is the equal sign =, the equality operator is ==:

```
In [13]: 3 == 3

Out[13]: True
```

The inequality operator is `!=`:

```
In [14]: 3 != 3

Out[14]: False
```

Data structures

The most basic data structure in Python is the **list**:

```
In [15]: l = [1,2,3,4]
```

Lists elements are accessed via indices (starting from zero):

```
In [16]: print(l[0])
         print(l[1])
         print(l[-1])
```

```
1
2
4
```

Assignment of list elements works as expected:

```
In [17]: l[0] = 'another string'
         print(l[0])
```

```
another string
```

The length of a list is returned by the `len()` builtin function:

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

```
Out[18]: 4
```

Another basic data structure is the **dictionary**, which is used to map *keys* to *values*:

```
In [19]: d = {'Alice' : 23452532, 'Boris' : 252336, 'Clarice' : 2352525, 'Doris' : 23624643}
         d['Alice']
```

```
Out[19]: 23452532
```

```
In [20]: d['Alice'] = 12345
         d['Alice']
```

```
Out[20]: 12345
```

```
In [21]: len(d)
```

```
Out[21]: 4
```

Functions

Function are defined via the `def` keyword:

```
In [22]: def hello_universe():
             print('witty remark no. 2')
```

Now the function can be used:

```
In [23]: hello_universe()
```

```
witty remark no. 2
```

Arguments can be specified in the function's definition, return values are specified by the `return` keyword:

```
In [24]: def my_add(a,b):
             result = a + b
             return result

         my_add(1,2)
```

```
Out[24]: 3
```

Functions define their own scope, i.e., variables defined within a function are not visible outside it:

```
In [25]: result
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-a5b1e83cd027> in <module>()
----> 1 result

NameError: name 'result' is not defined
```

Modules

Apart from the most basic features, functions, types and classes are grouped in separate *modules* which can be accessed via the `import` keyword. Let's import the standard `math` module:

```
In [26]: import math
```

Now we can access the functions and contstants defined in `math`:

```
In [27]: math.cos(1.23)

Out[27]: 0.3342377271245026
```

```
In [28]: math.sqrt(2)

Out[28]: 1.4142135623730951
```

```
In [29]: math.pi

Out[29]: 3.141592653589793
```

Rational numbers can be represented with the help of the standard `fractions` module:

```
In [30]: import fractions
         q = fractions.Fraction(2,3)
         print(q)
         print(q**5)

         2/3
         32/243
```

In order to reduce the typing, it is possible to import selectively:

```
In [31]: from fractions import Fraction
         q = Fraction(8,12)
         print(q)

         2/3
```

```
In [32]: from fractions import Fraction as Frac
         q = Frac(8,12)
         print(q)

         2/3
```

In addition to the standard modules, there are many Python modules freely available for download. NumPy is a popular package for linear algebra:

```
In [33]: from numpy import dot, cross
         from numpy.linalg import det
         dot([1,2,3],[4,5,6])

Out[33]: 32
```

```
In [34]: cross([1,2,3],[4,5,6])

Out[34]: array([-3,  6, -3])
```

```
In [35]: det([[1.2,3.4],[5.6,7.8]])
```

```
Out[35]:  -9.6800000000000015
```

mpmath is a package for arbitrary precision calculations:

```
In [36]: from mpmath import mpf, mp, sqrt
         sqrt(mpf(2))

Out[36]:  mpf('1.4142135623730951')

In [37]: mp.dps = 100
         sqrt(mpf(2))

Out[37]:  mpf('1.414213562373095048801688724209698078569671875376948073176679737990732478462107038850387534327641572735')
```