Introduction to Python: Main Concepts and Some Exercises

Summer School “Design and Optimization Under Uncertainty of Large Scale Numerical Models”

Anne Auger, anne.auger@inria.fr
Asma Atamna, asma.atamna@inria.fr
Dimo Brockhoff, dimo.brockhoff@inria.fr

1. To start Python, type `ipython` in your command line. To run the code, two possibilities:
   - Type the code directly in the IPython shell, or
   - Write the code in a Python file, e.g. `my_script.py`, then run the script using

```
run my_script.py
```

2. First Python program: The classical “Hello World”

```
# first Python program
print “Hello World”
```

3. Variables

A variable is a reserved memory space to store some value. No explicit declaration is needed, i.e. the declaration happens when the variable is assigned.

```
a = 10    # integer assignment
b = 10.   # float assignment
```

There are five standard data types:

- number: `a = 10, b = 10., c = -5e3`
- string: `language = “Python”`
- **list**: `l = [2.5, “Python”, 19]`
- tuple: `t = (“hello”, 127), read-only lists`
- dictionary: `d = {“day”: 4, “month”: “July”, “year”: 2017}`

4. Mathematical operators

+	addition
–	subtraction
*	multiplication
/	division
%	modulo (remainder)
**	exponent

Exercise 1: Run the following:

i. `3/2`

ii. `3./2`

iii. `from __future__ import division`

iv. `3/2` again

5. Loops

`while` `condition:`
`    action`
	`a = 0`
	`while a < 5:`
	`    a += 1`
	`    print a`

`for` `var in sequence:`
`    action`
	`l = [2.5, “Python”, 19]`
	`for x in l:`
	`    print x`

6. Conditional statements

```
if condition:  # condition evaluates to either True or False
    action 1
else:     # if alternative
    action 2
```

Comparisons and boolean operators

`<, <=`	less than, less than or equal to
`>, >=`	greater than, greater than or equal to
`==`	equal to
`!=`	not equal to
`is`	object identity
`is not`	negated object identity
`x or y`	if x is false, then y, else x
`x and y`	if x is false, then x, else y
`not x`	if x is false, then `True`, else `False`

7. Back to lists

A list contains items (possibly of different data types) separated by commas and enclosed within square brackets ([]). Items are numbered starting from 0.

Exercise 2:

We have

```
    l1 = [“Python”, 185 , 5.43, “Tuesday”, 90.3]
```

```
l2 = [123, “July”]
```

What does the code below produce?

```
print l1
print l1[0]
print l1[1:3]
print l1[2:]
print l1[-1]
l1.append(“hello”)
print len(l1)
print 2 * l2
print l1 + l2
l2 = l1
print l2
l1[-1] = 5
print l1
print l2
```

List comprehensions: A powerful and simple way to construct lists.

```
l = [x**2 for x in range(5)]
```

instead of

```
l = []
for i in range(5):
     l.append(i**2)
```

8. Functions

i. Definition

```
def function_name(parameters):
     treatment
     return result
```

ii. Call

```
function_name(parameters)
```

Exercise 3: Define in Python the following mathematical function:

f(x) = x^2, x is a real number.

To get help for predefined Python functions, you can type

```
function_name?
```

in IPython shell.

9. Modules and packages

i. A module is a Python file (`module_name.py`) that contains definitions of functions, classes, and variables.

ii. A package is a collection of modules.

Exercise 4: Using modules and packages

1. Create a Python module, `my_module.py`, that contains the definition of the function f from Exercise 3.
2. Run the following code and explain the results (you can use the provided documentation as previously explained above). We use here the predefined `numpy` package and `pyplot` module.

```
import numpy as np
import matplotlib.pyplot as plt
import my_module
my_list = np.arange(0, 10, 0.1)
my_array = np.array(my_list)
result = my_module.f(my_array)
plt.semilogy(my_array, result)
plt.show()
```