Basic programming with Python

Student and Staff IT Introduction

ssiti2013@gmail.com*

25/06/2013

1 Programming?

Programming is the key to a whole new world:

- Make simple computer task automatic
- Become the master of your own analyses
- Make your analyses automatic
- Simulate data
- Answer new research questions

All this is more accessible than you think!

2 Programming languages: opening the black box

Software that you usually use are black boxes. You input data, select a few parameters, and let the magic operates. Today we are going to open the black box[1].
Your computer speaks and operates in term of 0 and 1. A programming language is a way to communicate with the computer, make it compute your instructions (calculation, algorithm, file treatment), using a language understandable by humans and that can ultimately be translated into 0 and 1 to the computer.
Programming languages fall into two different categories: interpreted languages and compiled languages. Using an interpreted language, such as Python, you speak to an Interpreter which answers immediately to you — "on the fly". You can communicate to the interpreter directly in command lines or via scripts. If you write a program in a compiled language however, you will have to write a code in a text file and compile it using another program. This will transform your text file into an executable, i.e. a program that will be ready to run.

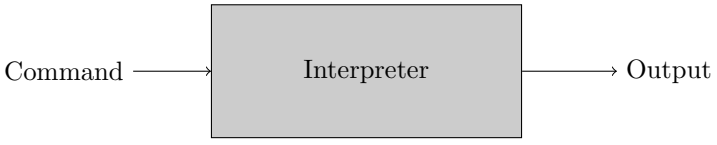


Figure 1: Programming with an interpreted language

*Document written by Elsa Guillot (e.guillot@massey.ac.nz), for IFS staff and students, Massey University.
[1]This is a simplified version of the theory of programming languages. The field is actually animated by fierce debate on the different definitions of languages, difference between interpreted and compiled, etc. None of which is meaningful for the purpose of this introduction

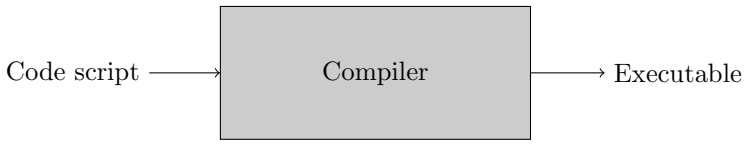
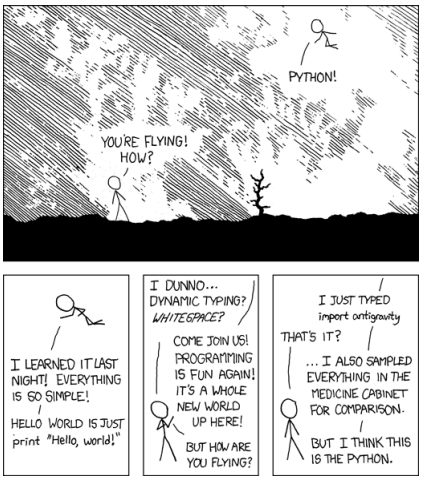


Figure 2: Programming with a compiled languages

Language	Compiled / Interpreted	Object oriented
FORTRAN	compiled	FORTRAN 2003
C	compiled	No
C++	compiled	Yes
C#	compiled	Yes
Python	Interpreted	Yes
Java	Interpreted	Yes
Ruby	Interpreted	Yes
Perl	Interpreted	from version 5
R	Interpreted	Yes
PHP	Interpreted	From PHP 3

In Bioinformatics, two languages are dominating in the programming of data analyses scripts : Python and Perl. Both are called "scripting languages" and are very similar in what they can do, but the battle is fierce between the pro-Python and the pro-Perl.

	Python	Perl
Creation year	1991	1987
Syntax	Simple	Complex and cryptic
Power	Vast amount of libraries	Rich in operator symbol



`http://xkcd.com/353/`

3 Python programming

Python is an easy language to learn and practice.
Open a shell :
`$ python`
You are now in the python interpreter, any command that you will type will be interpreted into python. As a tradition

in computer science, we will start by making the computer says “Hello World” :

```
>>> print(‘‘Hello World’’)
Hello World
>>> print(’I print what I want’)
I print what I want
>>> print 42
42
>>>#this is a comment
```

Python knows mathematics:

```
>>> 1+1
2
>>> 36-7
29
>>> 21*2
42
>>> 42/2
21
```

Call previous commands : up-arrow.
Quit python:

```
>>>exit()
```

You are back to the main shell. Type python to go back in the interpreter:

```
$python
```

All the documentation you need to program in python is on this website `http://docs.python.org/2.6/`

Variables !

A variable is an object in which you store a value.

```
>>> myvalue = 42
>>> myothervalue = ‘‘Hello world’’
>>> myvalue
42
>>> myothervalue
‘‘Hello world’’
>>> a=1 # save value 1 into a
>>> b=2
>>> a+b
3
>>> c=a+b+b
>>> c
5
>>> b=3 # modify the value of b
>>> b
3
>>> a=c # copy the value of c to a
>>> a
5
```

Each value has a type, which will define what you can do with it. Python handles the type of your variable for you.

```
>>> type(myvalue)
<type ’int’>
>>> type(myothervalue)
<type ’str’>
```

type	abbreviation	null value	example
boolean	bool	False	False
integer	int	0	42
floating point	float	0.0	1.5
complex	complex	0j	4+2j
string	str	”	’Hello World’
list	list	[]	[0,1,2,3] [’a’,’b’,’cde’,42]
tuple	tuple	()	(42,’hello world’)
dictionary	dict	{}	{’English’:’Hello world’, ’French’:’Bonjour tout le monde’, ’computer’:’0011001101’}

There exist a type called **tuple**. It is used to send a small set of values to and from functions.

We ignore two more types *set* and *frozenset* for now.

Why does it matter to know the type then ?

```
>>> 1 # this is an integer
1
>>> 1.0 # this is a float
1.0
>>> ’1’ # this is a string
’1’
>>> 1+1
2
>>> 1+1.0 # adding an integer and a float gives you a float
2.0
>>> 1+’1’ # you can not add an integer to a string
TypeError
>>> ’1’+’1’ # adding string concatenate them
’11’
>>> 43/2
21
>>> 43.0/2
21.5
>>> 43/2.0
21.5
```

The operation of adding string is called string concatenation. It is extremely useful: `>>> ’Hello’+’ ’+’fellow programmers’ # adding string concatenate them`
`’Hello fellow programmers’`

```
>>>mylist=[’a’,’b’,’c’,’d’,’e’]
>>>mylist[0] # access the first cell
```

In python, as in C and C++, for example, the first element of the cell has index 0

```
’a’
>>>mylist.index(’a’) # find the index of the first cell containing ’a’
0
>>>mylist[2]
’c’
>>>mylist[1:4] # access cell 1, 2 and 3
[’b’,’c’,’d’]
```

```
>>>mylist[-1]
'e'
>>>mylist
['a','b','c','d','e']
>>>mylist.append('f') # add a cell at the end with value f
>>>mylist
['a','b','c','d','e','f']
>>>mylist.pop() # delete last cell
>>>mylist
['a','b','c','d','e']
>>>mylist.pop(0) # delete cell 0
>>>mylist
['b','c','d','e']
>>>mylist2=['Hello','world']
>>>mylist2
['Hello','world']
>>>mylist+mylist2 # concatenate lists
['b','c','d','e','Hello','world']
```

To keep on exploring the list, try to play with the commands *mylist.reverse(), mylist.count(), mylist.index(), mylist.insert(), mylist.remove(), mylist.sort(), mylist.extend()*
Use python help :
`>>>help(list.reverse)`
When calling the help of a function, do not write the parenthesis!
The dictionary are very similar to list, except that you access the values using keys:

```
>>>mydict={'key':'value','english':'hello','french':'bonjour'} # create the dictionary
>>>mydict['english']
'hello'
>>>mydict.keys() # list the keys
['french', 'key', 'english']
>>>mydict.values() #list the values
['bonjour', 'value', 'hello']
>>>mydict.pop('key') # remove the entry 'key' and its value 'value'
['value']
>>>mydict['spanish']='hola' # add a new entry
```

Dictionaries have plenty of functions to handle them, you can see the list using the help :
`>>>help(dict)`

Strings have specific functions too. You can access each letter as if it was an array: `>>>a='hello'`

```
>>>a[1]
'e'
```

You can also find letters in the string (using **find()**), count the number of occurrence of a letter (**count()**), check if the string represent an integer (**isdigit()**), change the text into capital letters (**upper()**) or lower ones (**lower**). Look at the **help(str)** for a full description.
A useful function on the string is the function **split('x')**. It will find all the occurrence of **'x'** in the string, and make a list out of all the elements between the **'x'**. The result will be a list of n+1 elements, if n is the number of occurrences of **'x'** (x can be a single character or a string itself).

```
>>>a='hello my fellow'
>>>a.split('y')
['hello m', ' fellow']
>>>a.split('l')
['he', '', 'o my fe', '', 'ow']
>>>a.split()
['hello', 'my','fellow']
```

By default (i.e. without argument), **split** will separate the string where empty spaces are found.

Finally it is possible to change the type of a variable, using **int()**, **str()**, **float()**:

```
>>>a=1
>>>a
1
>>>b=str(a)
>>>b
'1'
>>>c=float(b)
>>>c
1.0
>>>d=float(a)
>>>d
1.0
>>>f=int(b)
>>>f
1
```

Be careful when you copy a variable ! There is great danger with copy, and you may not even know that you doing copies. A copy is the transfer of value from one variable to another. For example, here b will be a copy of a:

```
>>>a=1
>>>b=a
>>>b
1
```

With integers, float and string, there are no problem. You copy, and then each variable is independent.

```
>>>a=1
>>>b=a
>>>b=36
>>>b
36
>>>a
1
```

With list however, when you copy a variable using = there is dependency between the variable: they will refer to the same set of elements. It is problematic because if you modify one, you will also modify the other:

```
>>>a=range(10)
>>>a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>b=a
>>>b
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>a[1]=42
>>>a
[0, 42, 2, 3, 4, 5, 6, 7, 8, 9]
>>>b
[0, 42, 2, 3, 4, 5, 6, 7, 8, 9]
```

To avoid that you have to make a proper copy of the list. For that you could copy each element one by one, but there are simpler ways. A good way is to use the **list()** function with the list to copy as an argument.

```
>>>a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>b=list(a)
>>>b
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>a[1]=42
```

```
>>>a
[0, 42, 2, 3, 4, 5, 6, 7, 8, 9]
>>>b
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
Dictionaries behave in the same way. Therefore to copy a dictionary always use ***copy*=dict(*mydict*)**

Now you can save, copy, modify variables, apply simple operations to them. You can store those variables in arrays or dictionaries.
To do simple calculations and to play with variables, the shell is very convenient. But all is lost as soon as you exit the current python session. In order to do more complex things, we will need to write commands that we can save, and re-use.
For the next part we will leave the shell and use scripts.

```
>>>exit()
```

Scripts

A script is a list of python commands, saved in a file, using the exact same syntax as the one in the interpreted shell. First we will need to open a new file with the editor TextWrangler.
Start by saving this file. By convention names of python script file ends with *.py*. Save this file as *my1script.py* on your Desktop.
In your file, every command will use the same syntax as in a the shell. Therefore to print hello world on you screen enter in your file:

```
print('Hello world')
```

To execute this script in python: open a shell, move to your Desktop and use the python command line with your file name.

```
$ cd /Users/bioinformatics/Desktop/
$ python my1script.py
Hello world
```

The interpreter will execute all the command lines in the script and output the commands **print()** in your script. You can write some of the previous example in your files, and output with print to check values.

```
print('Hello world')
a=1
b=2
c=a+b
c # nothing is outputted in the shell
print(b) # the value of b is outputted the shell
mylist=[1,2,3,4,5]
mylist.reverse()
print(mylist) # output the list which is now [5,4,3,2,1]
```

Now let's get the computer to do something a bit more useful. The power of programming often comes from conditional statement and iteration tools.
Condition:**if () : / else :**

```
a=1
b=2
if (a==b): # condition
    print('a is equal to b')
else: # to do if condition is not met
    print('a is different from b')
```

Python uses indentation to mark which commands are dependant on a condition. After a ':' all lines that are indented in the same way form a block, which are all dependent on the same condition.

Conditions often come from comparison of variables :

Operation	Meaning
$<$	strictly less than
$<=$	less than or equal
$>$	strictly greater than
$>=$	greater than or equal
$==$	equal
$!=$	not equal
is	object identity
is not	negated object identity

You can also combine conditions using the operators **and** and **or**:

```
a=1
b=2
c=1

if ((a==b) or (a==c)): # condition
    print('a is equal to b or c')
else: # to do if condition is not met
    print('a is different from b and c')

if ((a==b) and (a==c)): # condition
    print('a is equal to b and c')
else: # to do if condition is not met
    print('a is different from b or c')
```

Iteration: Loop **for − in − :**

```
mylist=['Hello','world','we','come','in','peace']
for i in mylist: # this is an integration over the element of mylist
    print(i)
```

The message outputted by python takes several lines. This is because each time the function **print** has terminated its job, it sends the signal to start a new line. How can you make the message hold on one line? Using the same loop, can you print 'Hello world we come in peace !' in the shell?

Iteration: Loop **for − in range() :**

```
print('Lets print the number from 0 to 9')
for i in range(10):
    print(i)

print('Lets print the number from 2 to 10')
for i in range(2,11):
    print(i)

print('Lets print the number from 0 to 10, with a step of 2')
for i in range(0,11,2):
    print(i)
```

Can you print the sum of the first hundred number (from 1 to 100)?

Iteration: Loop **while():**

```
print('Lets do the multiplication table of 42 with while')
```

```
a=0
while(a<10):
    b=a * 42
    print('42 x %d = %d'%(a,b))
    a+=1 # equivalent to a = a+1
```

type	print()
int	print('a = %d'%a)
float	print('a = %f'%a)
complex	print('a = %s'%a)
str	print('a = %s'%a)

Using the function **pow(number,exponent)** can you print 2^i for the ten first value of i?

Functions

We have been using functions already. We have used **print()**, **help()**, **range()pow()**, etc. We are now going to create our own function.

```
def say_hello():
    print('Hello world')
```

If you send the function to the interpreter, nothing happens. You have to call the function to make it run.

```
def say_hello():
    print('Hello world')

say_hello()
```

This function is very simple, it has no parameter. Let's improve it:

```
def say_hello(name):
    print('Hello '+name)

say_hello('enter_your_name')
say_hello('enter_the_name_of_your_neighbour')
```

We can do even better. It is time to interact with the user.

```
def say_hello(name):
    print('Hello '+name)

myname=raw_input('Enter_your_name : ') # ask you in the shell to write the name
say_hello(myname)
```

A function can also have several variables.

```
def say_hello(name,language):
    if((language=='english') or (language=='English')):
        print('Hello '+name)
    elif((language=='maori') or (language=='Maori')):
        print('Kia ora '+name)
    elif((language=='spanish') or (language=='Spanish')):
        print('Hola '+name)
    elif((language=='french') or (language=='French')):
        print('Bonjour '+name)
    elif((language=='kiwi') or (language=='Kiwi')):
        print('Haye mate')
    else:
        print('Sorry I do not speak '+language)

myname=raw_input('Enter your name : ') # ask you in the shell to write the name
```

```
mylanguage=raw_input('Enter your language : ') #
say_hello(myname,mylanguage)
```

Finally your function can return a value, using the key word **return**. You can return something of any type: int, bool, dict, tuple, etc.

```
def who_am_I(name,language):
  return name
def what_is_my_language(name,language):
  return language
myname=raw_input('Enter your name : ') # ask you in the shell to write the name
mylanguage=raw_input('Enter your language : ') #
result = who_amI_(name,language)
print(result)
result2 = what_is_my_language(name,language)
print(result2)
```

Make your own program

In order to apply all what we have learned today, try to make the following program. The user will input a string sequence of DNA (i.e. containing only ACTG). The program will read the sequence and output the opposite strand. As reminder, if you have a base A on a strand, you have a T on the opposite strand. Here are the rules
A—T
C—G
T—A
G—C

Hint: you can iterate on the letters of a string using the structure **for *letter* in *string*:**

If you have time you can make your program count the number of sites containing each base, find a codon (3 bases), count the number of codon.

Programming vocabulary

Script	File containing commands written in a programming language
Source code	Set of files containing all the command lines that make a program run
Open Source	A program is open source when the source code is available (on the internet) for others to read, potentially modify and improve
Portability	Ability to use the same code on different computers, running under different O.S.
Maintained code	A code is well maintained when you have a support person or group that will continuously test and modify the code to make it run while the OS, language and interpreter change
Variable	Instance of an object, i.e. something that store data
Argument	An argument of a function is something you insert within parenthesis when you call the function, a parameter.
Indentation	Set of consecutive white spaces that report the start of a text to the right side.