Introduction to `python`

Basics

Mauricio Sevilla

email= j.sevillam@uniandes.edu.co

11.01.19

We have been working on the `shell` so far, but now, we are going to start talking about programming on `python`.

First of all, we are going to talk about the way `python` works compared with different programming languages such as `C/C++`

Programming languages families

One can divide the programming languages in two big families,

- Compiled
- Interpreted

and the two of them have huge differences!.

Generally compiled languages are way more dificult to write than interpreted laguages.

That is because the computer must *translate* the codes into binary operations so that they can be performed.

As we cannot write binary, we use a different **Language**!

There are two different strategies to perform this *translation*

>

- *Translate the complete file (code).*
- *Translate the code line by line.*

This task can is the difference between the compiled and interpreted languages.

Which is which?

Python as an interpreter

`Python` can be used as an interpreter, in the sense that makes a translation line by line of our code, so we can run one intruction at a time in the same way we did with the `shell` , to do so open python

> ***Note*** *: We are going to use* `python3` *instead of just* `python`
> *Today, the latest version of python is* `Python 3.7.2` *, so we are not going to use* `python` *(which means* `Python 2.7` *).*

The main reason for that, is because `Python 2.7.8 was released on July 1, 2014.` is too old!!, meanwhile Release `Python 3.7.2 was released on Dicember 24, 2018.`

Invoke `python 3` by typing

```
python3
```

on the terminal

```
In [1]: 2+3
Out[1]: 5

In [2]: 2.0+3.0
Out[2]: 5.0

In [3]: 2.+3.
Out[3]: 5.0
```

It is also possible to save values in variables to operate them

In [4]:
```
a=3
```

In [5]:
```
b=2
```

In [6]:
```
print(a+b,a-b)
```

```
5 1
```

Operators

There are two different classes of operators on any programming languages,

>

- *Arithmetic*
- *Comparison*
- *Logical*

What do you think they are?

Variables

To understand how the operators work, we have to study a bit deeper what a variable mean.

In [7]:
```
a=2
print(a)
```

```
2
```

The value `2` is saved in `a`.

`a` has a specific physical place on the memory of the computer, so that every time we type `a` the computer goes to that particular place and reads the value.

As the computer doesn't *understand* the number `2` but binary instead, there is a huge difference if we use

In [8]:
```
a=2
```

In [9]:
```
b=2.0
```

What do you think is the difference?

Arithmetic Operators

To test them, let us explore some operators such as `+`, `-`, `*`, `/`, `**`

In [10]:
```
a+b
```

Out[10]:
```
4.0
```

In [11]:
```
a+a
```

Out[11]:
```
4
```

NOTE: On different versions the `python` , you get different results.

In [12]:
```
1/a
```

Out[12]:
```
0.5
```

In [13]:
```
1/b
```

Out[13]:
```
0.5
```

Let us use the method `type`

In [14]:
```
type(a)
```

Out[14]:
```
int
```

In [15]:
```
type(b)
```

Out[15]:
```
float
```

which means that, `a` is an integer, while `b` can have decimals

This have a huge impact but depends on the `python` version we are using, on different languages such as `C++` , you have to say which kind of variable you want, for instance

```
int a=2;
float b=2.0;
double c=2.0;
```

where `float` and `double` are data types that allow decimal points.

But, there are some other data types, one of great interest, they are called `boolean` .

`boolean` varialbes only have two different possible values `True` or `False` .

Depending on the language, they can be written differently, for example, in languages such as `c++` and `julia` , the possible values are written in lower case.

C++

```
bool test=true;
```

julia

```
test=true
```

While in python the first letter must be upper case,

In [16]:
```
test=True
```

Where do you think we can find bools?

Comparison Operators

In [17]:
```
a>b
```

Out[17]: False

In [18]:
```
a<b
```

Out[18]: False

Logic operators

Sometimes we will need to have a combination of conditions to satisfy on a particular problem,

For example, the set of people on our course is different if we ask for

- *Female and older than 20.*
- *Female or older than 20.*

Think about the difference.

On other languages is common to use `!`, `||`, `&&` for *negation*, *or* and *and*. `python` is way simpler, it uses

- `!` for negation.
- `or` for or.
- `and` for and.

In [20]:
```
a=True
b=False
```

In [21]:
```
a and b
```

Out[21]:
```
False
```

In [22]:
```
a or b
```

Out[22]:
```
True
```

In [23]:
```
a=2
b=3
```

how can I ask if `a` and `b` are the same?

Comparison equality

In [27]:
```
a==b
```

Out[27]:
```
False
```

In [28]:
```
a==2.0
```

Out[28]:
```
True
```

Composed operators

In [31]:
```
a>=b
```

Out[31]:
```
False
```

You guys should take a look at **Truth tables**.

Now, we are ready to start working on a little bit more complex structures, such as **control statements**

`if` statement

This is used for running some part of the code, only if a condition is satisfied.

`python` is based on identation rather than characters to the control statements such as other languages can.

A `python` structure of an `if` structure goes as

```
if condition:
    inside
outside
```

if the condition results to be `True` the inside part is executed, let see some examples.

In [37]:
```
if True:
    print('Inside of If')
print('Outside of If')
```

```
Inside of If
Outside of If
```

In [38]:
```
if False:
    print('Inside of If')
print('Outside of If')
```

```
Outside of If
```

In [43]:

```
a=1
b=2
if a>b:
    print('a is greater than b')
else:
    print('b is greater that a')
```

```
b is greater that a
```

but, what if a=b?

In [46]:
```
a=1
b=1
if a>b:
    print('a is greater than b')
elif a<b:
    print('b is greater that a')
else:
    print('b is equal to a')
```

```
b is equal to a
```

`elif` holds for `else if`

`for` statement

The `for` is one of the loop structures that can be used on `python` , when a procedure must be repeated

Again, we have to be careful with the indentation, so that

In [2]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

lists

The `for` structure, have the huge advantage compared with languages such as `C++` and `fortran` , because of it allows to iterate on the compounds of a data type called `list` s.

A `list` is a set of things on python, the most important thing here is that one can use any kind of thing inside a list, such that

- `lists`
- `strings`
- Numbers: `int` or `float`
- objects
- pointers
- ...

The only thing we have to consider is to make it inside of `[]`, let see some examples

In [12]:
```
list1=[]
print(list1)
```

```
[]
```

In [13]:
```
list2=[10]
print(list2)
```

```
[10]
```

In [16]:
```
list2=[1,2,3,4,5]
print(list2)
```

```
[1, 2, 3, 4, 5]
```

List on a `for`

In [17]:
```
for i in ['value1', 'value2']:
    print(i)
```

```
value1
value2
```

In [18]:
```
list3=[1,2,3,4,5,6,7,8,9,10]
for i in list3:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

Then one can have more than one structure inside another, let see some examples

In [21]:
```
for i in range(5):
    if i == 2:
        print('Inside first if')
    elif i ==3:
        print('Inside elif')
    print('Inside for on loop: ',i)
```

```
Inside for on loop:  0
Inside for on loop:  1
Inside first if
Inside for on loop:  2
Inside elif
Inside for on loop:  3
Inside for on loop:  4
```

In [19]:
```
for i in range(10):
    if i%2==0:
        print(i)
```

```
0
2
4
6
8
```