Introduction to `python`

`lists` and `strings`

Mauricio Sevilla

`email= j.sevillam@uniandes.edu.co`

11.01.19

We have already started with some of the most basic ideas behind the `list` s,

Today we are going to focous on the features, advantages and disadvantages of working with `list` s.

lists

On `python` , a `list` is a set of *things*, of any kind!, and even each compound can have a different type than the others, you can have,

- `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 [1]: list1=[]
        print(list1)
        type(list1)

        []

Out[1]: list
```

An empty `list` , and

```
In [2]: list2=[10]
        print(list2,type(list2),type(list2[0]))

        [10] <class 'list'> <class 'int'>
```

Operations such as `+` , `*` can be performed, but the result is not what one would expect,

```
In [3]: list1=[1,2,3,4,5]
        list2=[3,2,4,6,9]
        print(list1+list2)
```

```
[1, 2, 3, 4, 5, 3, 2, 4, 6, 9]
```

This operation cannot be performed with an number and a list

```
In [4]: print(list1+2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-f7df2789d0eb> in <module>
----> 1 print(list1+2)

TypeError: can only concatenate list (not "int") to list
```

We have to take into account that, not all operations are allowed between certain types

```
In [5]: print(list1*list2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-2f25b862d700> in <module>
----> 1 print(list1*list2)

TypeError: can't multiply sequence by non-int of type 'list'
```

Sometimes they work with specific types,

```
In [6]: print(list1*2)
```

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

Finally

```
In [7]: print(list1**2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-a2e329edf605> in <module>
----> 1 print(list1**2)

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

So, we have to explore which operators can be used on which variables.

Lets create a list with different kind of data, an `int` , `float` , character, `str` and a `list`

```
In [8]: list_test=[1,2.0,'c',"word",['list_a','list_b']]
```

Let us explore the data.

```
In [9]: for i in list_test:
            print(type(i))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
<class 'list'>
```

look that `'c'` and `"word"` are of the same type.

```
In [10]: print(list_test)
```

```
[1, 2.0, 'c', 'word', ['list_a', 'list_b']]
```

Let us explore some of the functions we can use on `list` s

- `len()`

```
In [11]: print(len(list_test))
```

```
5
```

```
In [12]: print(len(list_test[0]))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-cc4495ea67a3> in <module>
----> 1 print(len(list_test[0]))

TypeError: object of type 'int' has no len()
```

Note : Doesn't work on numbers, but on strings?

```
In [13]: print(len(list_test[3]), list_test[3])
```

```
4 word
```

In some sense, the `str` and `list` have the same structure!

Differences on `for`

```
In [14]: for i in list_test:
             print(i)
```

```
1
2.0
c
word
['list_a', 'list_b']
```

```
In [15]: for i in range(len(list_test)):
             print(i)
```

```
0
1
2
3
4
```

`enumerate` function

```
In [16]: for i,j in enumerate(list_test):
             print(i,j)
```

```
0 1
1 2.0
2 c
3 word
4 ['list_a', 'list_b']
```

`list` s also can be accesed with negative values!

```
In [17]: print(list_test[-1])
```

```
['list_a', 'list_b']
```

Sometimes, you can have more than one index

```
In [18]: print(list_test[-1][1])
```

```
list_b
```

`append` method,

there are different ways to use *functions* on structures as `lists`, for example `len()`, or `range()`, but there are some such as `append` that are called **Methods**, they are the heart of `python` because is a language based on *Object Oriented Programming*

In [19]:
```
print(list_test)
```
```
[1, 2.0, 'c', 'word', ['list_a', 'list_b']]
```

In [20]:
```
list_test.append(1)
```

In [21]:
```
print(list_test)
```
```
[1, 2.0, 'c', 'word', ['list_a', 'list_b'], 1]
```

```
In [22]: list_test.append([1,2,3,4,5])
```

```
In [23]: print(list_test)
```

```
[1, 2.0, 'c', 'word', ['list_a', 'list_b'], 1, [1, 2, 3, 4, 5]]
```

Homework

Look for some methods to erase cells on a `list`

You can change the elements of a list, even if the new value have a different type.

```
In [24]: list_test[6]=2
```

```
In [25]: print(list_test)
```

```
[1, 2.0, 'c', 'word', ['list_a', 'list_b'], 1, 2]
```

There are other things we can do on `list` s, for example, how can we get more than one value of a `list` at a time?

```
In [26]: print(list_test[1:4])
```

```
[2.0, 'c', 'word']
```

```
In [27]: print(list_test[3][2:])
```

```
rd
```

When we use `[2:]` it means that it starts at `[2]` and goes until the end. we can also use `[:3]` and it means that goes from the begining until the 2nd compound.

Strings

A string is a set of characters,

```
In [28]: test1='test'
         test2="test"
```

```
In [29]: print(test1==test2)
```

```
True
```

There is no difference between `'` , `"` .

And we can use the same structure than we just did with the `list` s!

```
In [30]: len(test1)
```

```
Out[30]: 4
```

```
In [31]: print(test1[1],test2[3])
```

```
e t
```

```
In [32]: test1.append('a')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-32-2fffa1be48b3> in <module>
----> 1 test1.append('a')

AttributeError: 'str' object has no attribute 'append'
```

```
In [33]: test1=test1+'\t'+'test2'
```

```
In [34]: print(test1)
```

```
test    test2
```

Strings can be multiplyed

```
In [35]: print(test2*2,type(test2*2))
```

```
testtest <class 'str'>
```

How can this be useful?