Introduction to python

lists and strings

Mauricio Sevilla

email= jsevilla@credicorpcapital.com

09.04.19

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

Sometimes they work with specific types,

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

Finally

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

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

Let us explore the data.

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

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

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

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

- `len()`

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

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

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

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

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

Differences on `for`

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

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

`enumerate` function

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

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

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

Sometimes, you can have more than one index

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
In [ ]: print(test1[1],test2[3])
In [ ]: test1.append('a')
```

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

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

Strings can be multiplyed

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

How can this be useful?