Lists in Python and some methods for list objects

Author: Birane Koundoul, PhD student

at University Alioune Diop of Bambey – Senegal
(birane.koundoul@uadb.edu.sn)

Reviewer: Aliou Diouf (aliou.diouf5@gmail.com)

September 24, 2023

PYTHON

Summary

1. Introduction

- ✓ Sometimes you need a single variable to store several values, values of the same type or of different types. Lists can do this.
- ✓ In this paper, we will look at lists. The latter are very interesting because it allow us to:
  - put in several variables;
  - contain values of different types;
  - etc.
- ✓ Creating a list is simple. A list is created using square brackets [ ] and the objects in the list are separated using a comma.
- ✓ But you can also create an empty list. Items can be added as you go along.
- ✓ The advantage is that there are lots of methods we can use to easily manipulate our list.
- ✓ Its methods include :
  - **append** : to add an item to the end of the list.
  - **extend** : to extend the list by adding all the elements of the iterable.
  - **remove** : deletes from the list the first element whose value is equal to "val". Note that a **ValueError** exception is thrown if "val" does not exist.
  - etc.
- ✓ We are going to make examples of the methods

2. Examples :

We will show you here two types of list:

- An empty list
- And a non empty list

2.1. Example 1: empty list

Here we create an empty list

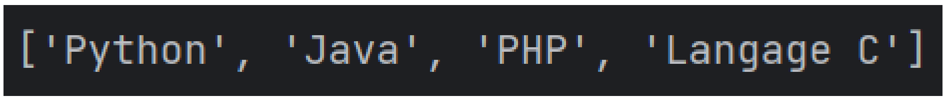
```python
# Example of empty list
subjects = []
print(subjects)
```

2.2. Example 2: non empty list

Here we put the contents of the list directly

```python
# Example of non-empty list
subjects = ["Python", "Java", "PHP", "Langage C"]
print(subjects)
```

What is printed:



```
['Python', 'Java', 'PHP', 'Langage C']
```

2.3. Methods

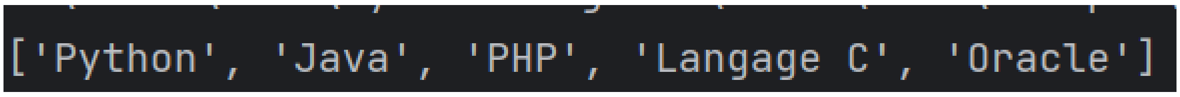
###

2.3.1. Method append()

We said in the introduction that the **append()** method lets us add an element to the end of the list.

```
"""List of subjects"""

subjects = []
subjects = ["Python", "Java", "PHP", "Langage C"]
subjects.append("Oracle") # add a new subject to the end of the list
print(subjects)
```



```
['Python', 'Java', 'PHP', 'Langage C', 'Oracle']
```

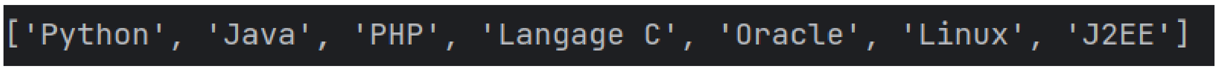
###

2.3.2. Method extend()

This method extends the list by adding all the elements of another list or iterator to the end of the first list.

We are going to create a new list containing new subjects.

```
"""extend method"""
new_subjects = ["Linux", "J2EE"]
subjects.extend(new_subjects)
print(subjects)
```



```
['Python', 'Java', 'PHP', 'Langage C', 'Oracle', 'Linux', 'J2EE']
```

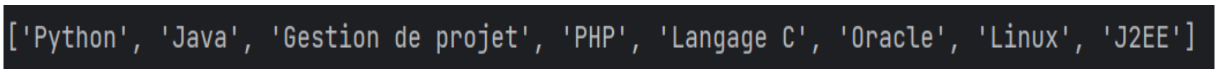
Note also that the extend() method does not just take a list as a parameter. It can also take other iterator like tuple, string, etc...

##

2.3.3. Method insert

Instead of placing an element at the end of the list with the append() method, you can specify the position to insert an element. To do this, we are going to use the insert() method.

```python
# Insert in position 3
subjects.insert(2,"Gestion de projet")
print(subjects)
```

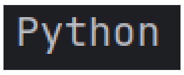


```
['Python', 'Java', 'Gestion de projet', 'PHP', 'Langage C', 'Oracle', 'Linux', 'J2EE']
```

You are going to wonder why we have put 2 instead of 3. It is like with arrays, the first element corresponds to index 0.

Let's check the first value in the list according to its index:

```python
print(subjects[0])
```



```
Python
```

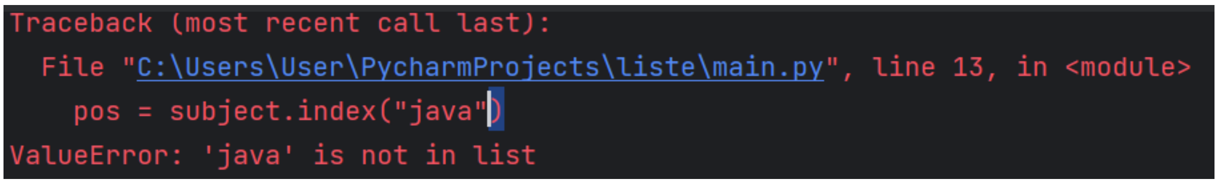
However, the index of an element in the list can be found using the index() function.

###

2.3.4. Method index()

This method returns the index of the specified element. But you need to be very careful: if the element is not present, an error is thrown.

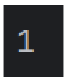
```
pos = subjects.index("java")
print(pos)
```



```
Traceback (most recent call last):
  File "C:\Users\User\PycharmProjects\liste\main.py", line 13, in <module>
    pos = subject.index("java")
ValueError: 'java' is not in list
```

You may wonder why this error while "java" is in the list. Yes, it is true, "java" is in the list, but as Python is case-sensitive, the "Java" in the list is with an uppercase "J" and we get a lowercase "j".

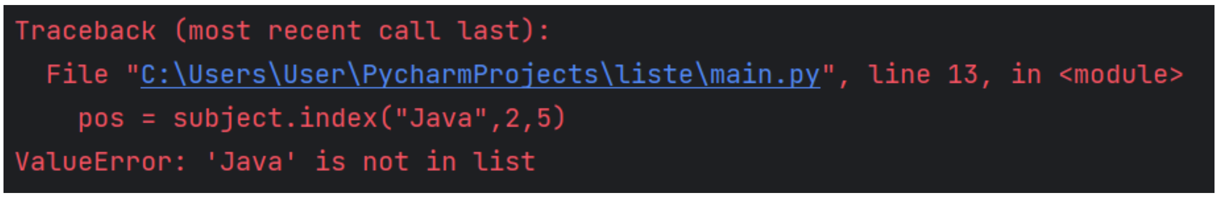
```
pos = subjects.index("Java")
print(pos)
```



```
1
```

Warning: the item may be present in the list but still has an error. With the index() method, you can also specify the search interval by giving the desired start index and the desired stop index too.

```
pos = subjects.index("Java",2,5)
print(pos)
```



```
Traceback (most recent call last):
  File "C:\Users\User\PycharmProjects\liste\main.py", line 13, in <module>
    pos = subject.index("Java",2,5)
ValueError: 'Java' is not in list
```

"Java" with a capital "J" is in the list, but for the range specified, "Java" is not in."

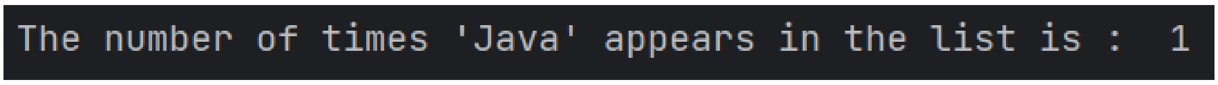
##

2.3.5. Method count()

It is useful to know how many times an element occurs in the list. It is easy to do, using the count() method, the number of times an item is present in the list.

We are looking for the number of times "Java" appears in the list.

```
print("The number of times 'Java' appears in the list is : ",
subjects.count("Java"))
```



```
The number of times 'Java' appears in the list is :  1
```

Now that we have seen a few methods, let's sort the items in the list.

##

2.3.6. Method sort()

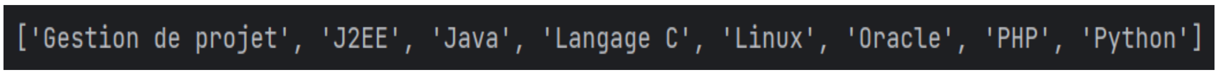
The sort() method lets us sort a list. It is different from **sorted**. sorted() sorts like sort() but is built-in function while sort() is list member. sorted() can take an iterator like string, tuple, dictionary, set, etc.

The syntax of the method is shown below:

<list>.sort(reverse = True | False, key = <func>) with :

- ✓ **<list>** is a valid Python list object.
- ✓ **reverse** is an optional that takes the value True or False.
    - The default value of reverse is False and the list is sorted in ascending order.
    - If we set reverse to True, the list will be sorted in descending order.
- ✓ key is also an optional whose parameter is set to <func>.
- ✓ <func> can be a built-in function or a user-defined function.

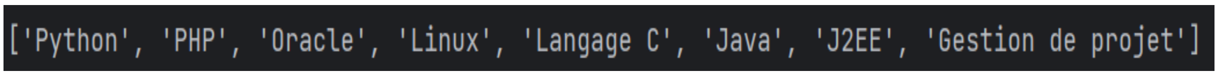
```
subjects.sort()
print(subjects)
The items are sorted in ascending order, because the "reverse"
parameter has a default value of False.
```



```
['Gestion de projet', 'J2EE', 'Java', 'Langage C', 'Linux', 'Oracle', 'PHP', 'Python']
```

To do a descending sort, set the value of "reverse" to True.

```
subjects.sort(reverse=True)
print(subjects)
```



```
['Python', 'PHP', 'Oracle', 'Linux', 'Langage C', 'Java', 'J2EE', 'Gestion de projet']
```

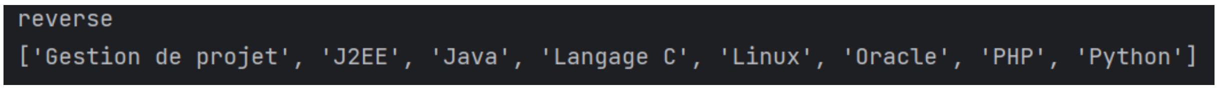
Sometimes you may want to invert the items in the list. That is, starting from the last to the first. This can be done with reverse() method.

```
########################################################################
```

2.3.7. Method reverse()

This method (reverse) allows us to invert the elements of the list. Remember that the initial list is now sorted. We work on this list.

```python
print("reverse")
subjects.reverse()
print(subjects)
```



```
reverse
['Gestion de projet', 'J2EE', 'Java', 'Langage C', 'Linux', 'Oracle', 'PHP', 'Python']
```

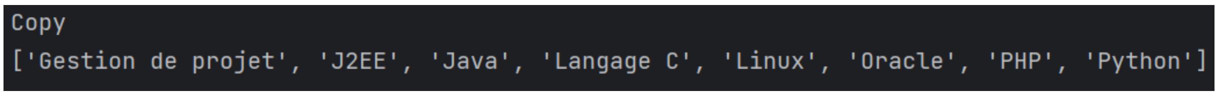
With lists, you can copy one list to another.

```
########################################################################
```

2.3.8. Method copy()

This method allows us to copy the list to another.

```python
print("Copy")
copy_subjects = subjects.copy()
print(copy_subjects)
```



```
Copy
['Gestion de projet', 'J2EE', 'Java', 'Langage C', 'Linux', 'Oracle', 'PHP', 'Python']
```

You can have fun deleting an element or all the elements in the list. It may be that you no longer need an element or even the list.

##

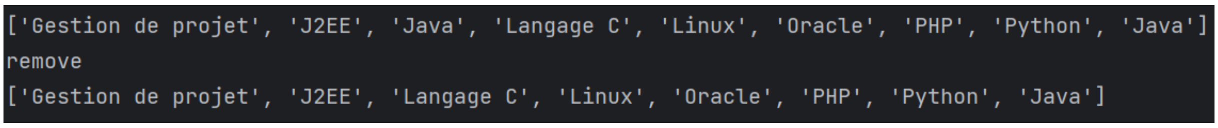
2.3.9. Method remove()

The remove() method allows us to remove an element from the list.

Warning:

- ✓ If the element is present several times in the list, only the first element passed as an argument will be removed from the list. The other elements will remain. In other words, the other elements corresponding to the specified element.
- ✓ If the element passed as an argument is not present, a **ValueError** exception is thrown.

```python
subjects = ['Gestion de projet', 'J2EE', 'Java', 'Langage C',
'Linux', 'Oracle', 'PHP', 'Python', 'Java']
print(subjects)
print("remove")
subjects.remove("Java")
print(subjects)
```



```
['Gestion de projet', 'J2EE', 'Java', 'Langage C', 'Linux', 'Oracle', 'PHP', 'Python', 'Java']
remove
['Gestion de projet', 'J2EE', 'Langage C', 'Linux', 'Oracle', 'PHP', 'Python', 'Java']
```

The "Java" passed as an argument is deleted, but the other "Java" remains.

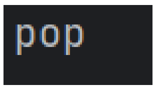
###

2.3.10. Method pop()

This method allows us to remove the element located at the specified position from the list. It returns a return value after deletion.

However, if no position is specified, **list.pop()** removes and returns the last element in the list.

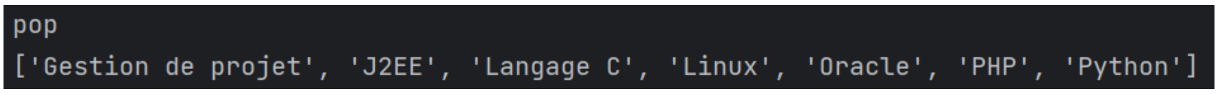
```
print("pop")
subjects.pop("Langage C")
```



```
pop
```

At this time, We are not going to put any arguments in the pop() method.

```
print("pop")
subjects.pop()
print(subjects)
```



```
pop
['Gestion de projet', 'J2EE', 'Langage C', 'Linux', 'Oracle', 'PHP', 'Python']
```

We will see that it is the last element that is deleted and that corresponds to "Java".

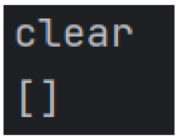
Now let's delete all the items in the list.

##

2.3.11. Method clear()

This method allows you to delete all the elements in the list.

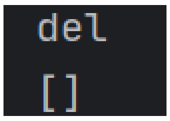
```
print("clear")
subjects.clear()
print(subjects)
```



```
clear
[]
```

The clear() method is equivalent to **del list**[index_start : index_final].

```
subjects = ['Gestion de projet', 'J2EE', 'Java', 'Langage C',
'Linux', 'Oracle', 'PHP', 'Python', 'Java']
print(subjects)
print("del")
del subjects[0:9]
print(subjects)
```



```
del
[]
```