Data Structures

Introduction

- So far, all the data we have worked with was simple : integers, floats, strings and booleans
- Last week we discussed loops
- It would be nice to have data structures that we could do loops on!
- Examples:
  - Contact list on your smartphone
  - List of cities in France, with zip codes etc.
  - List of blog posts
  - etc.

Python Data Structure

- Lists: ordered collection of data, size can vary, values can be repeated
  - l = [65, 32, 12, 4]
  - l = [“foo”, “bar”]
  - l = [45, True, “foo”, 23.5]
- Tuples: ordered collection of data, size is fixed when initialized, values can be repeated
  - t = (1, 2, 3)
  - t = (“foo”, “bar”)
- Dictionaries: pairs of key and value, size can vary
  - d = {”city”: “Paris”, “zip”: “75000”}
- Sets: unordered collection, values are unique
  - s = {1, 4, 5}

List operation

- Initializing a list :
  - l = []
  - l = [“Mary”]
- Concatenating lists:
  - my_list = []
  - my_list += [“John”]
- Repeating the same element:
  - l = [“Paul”] * 3
  - [“Paul”, “Paul”, “Paul”]

Accessing an element of a list

- **WARNING:** the index of the first element in a list (or set or tuple) is **0!**
- my_list = [3, 5, 10, 1]
- my_list[0]   (value is 3)
- my_list[2]   (value is 10)

Iterating on lists

Length of a list

- If we want to loop on a list, we (may) need to know its length
- l = [1, 4, 3, 9]
- len(l) returns 4
- l has indexes from 0 to 3

Range is a hidden list!

- What is the result of:
    - print(list(range(2, 9, 2)))

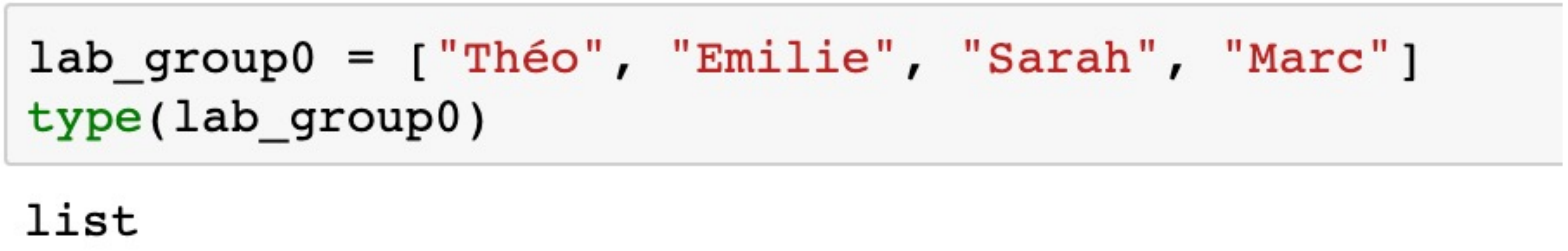
Range is a hidden list!

- What is the result of:
    - print(list(range(2, 9, 2)))
    - [2, 4, 6, 8]

Range and len

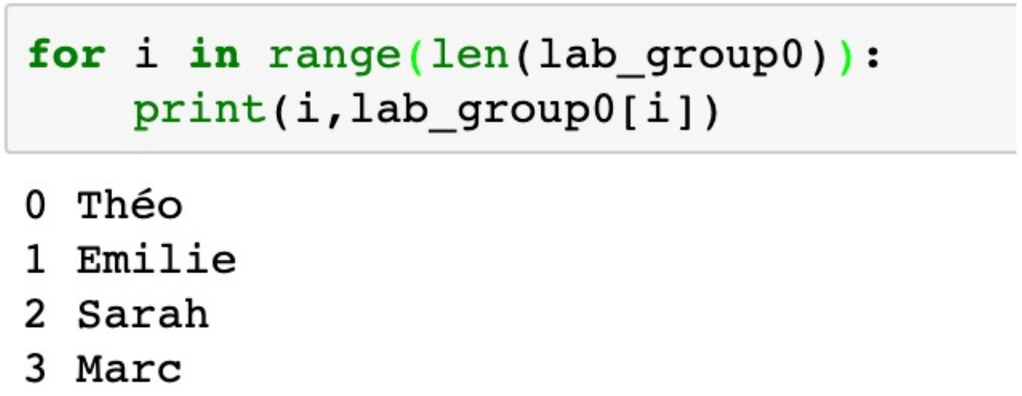
- range(a, b) goes from a **included** to b **excluded**
- range(b) goes from 0 included to b excluded
- let l be a list of length b. The indexes for the elements in l go from 0 **included** to b **excluded**
- We can use range(len(l)) to iterate on list l
- This is why the last element of a range is excluded: it helps with lists!

Example



```python
lab_group0 = ["Théo", "Emilie", "Sarah", "Marc"]
type(lab_group0)
```

```
list
```

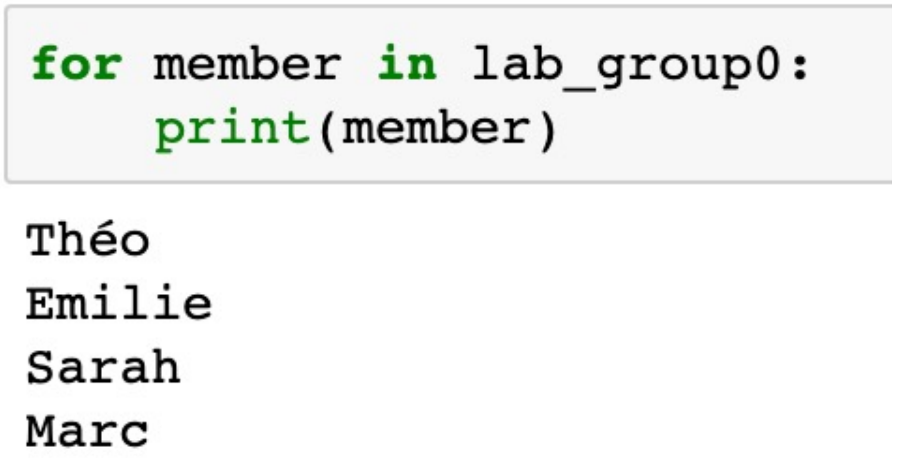


```python
for i in range(len(lab_group0)):
    print(i,lab_group0[i])
```

```
0 Théo
1 Emilie
2 Sarah
3 Marc
```

Iterating without range

- We actually don’t need range to iterate
- In this case, we do **not** access the index of each element in the list
- What if we need both the index and the element in the list?
- We can use range and len, or…

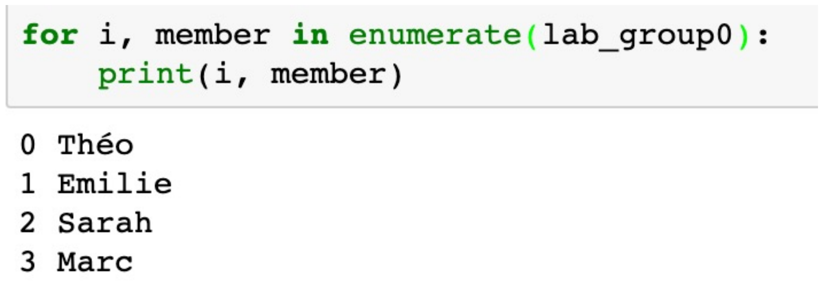


```python
for member in lab_group0:
    print(member)
```

```
Théo
Emilie
Sarah
Marc
```

Enumerate

- enumerate is a function that allows us to enumerate the elements of a list
- enumerate returns a couple (tuple of size 2) for each iteration
- The returned couple is (index, element)
- It is more concise and elegant than range(len…



```python
for i, member in enumerate(lab_group0):
    print(i, member)
```

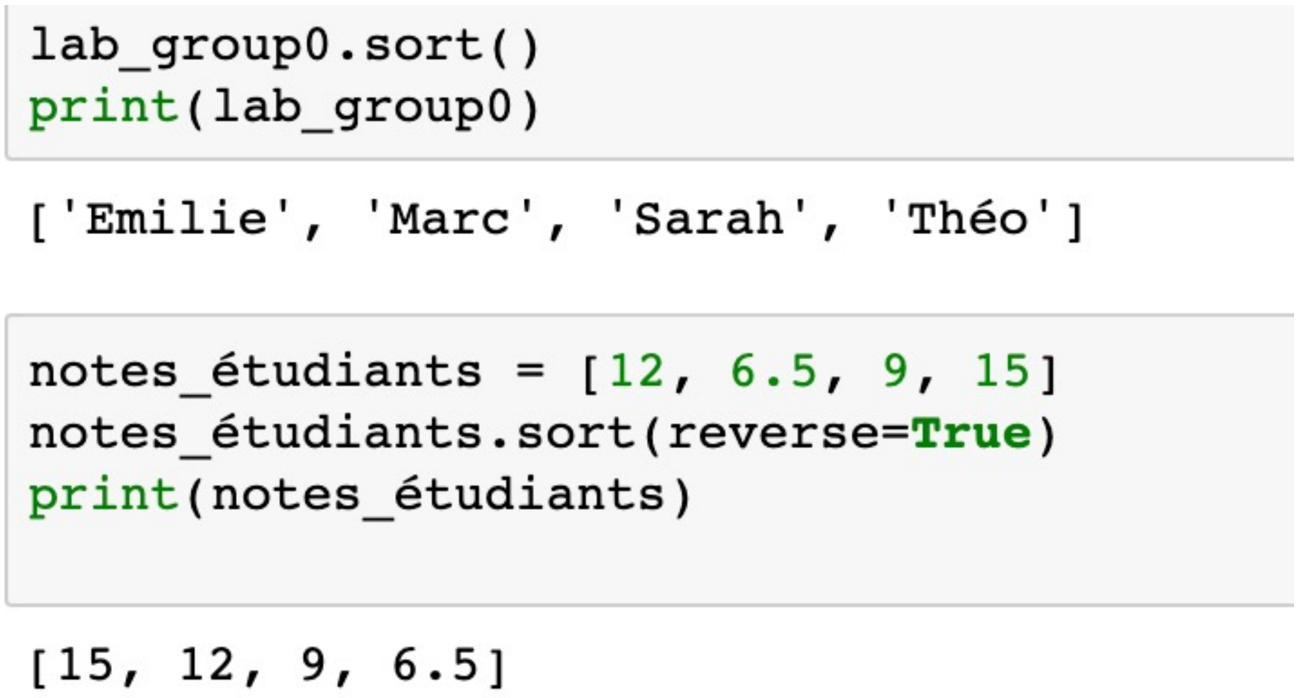
```
0 Théo
1 Emilie
2 Sarah
3 Marc
```

Exercise

- tab=[3, 7, 9, 34, 23, 18]
- How do we get the sum of all elements in tab?

Operations on lists

Sorting a list



```
lab_group0.sort()
print(lab_group0)
```

```
['Emilie', 'Marc', 'Sarah', 'Théo']
```

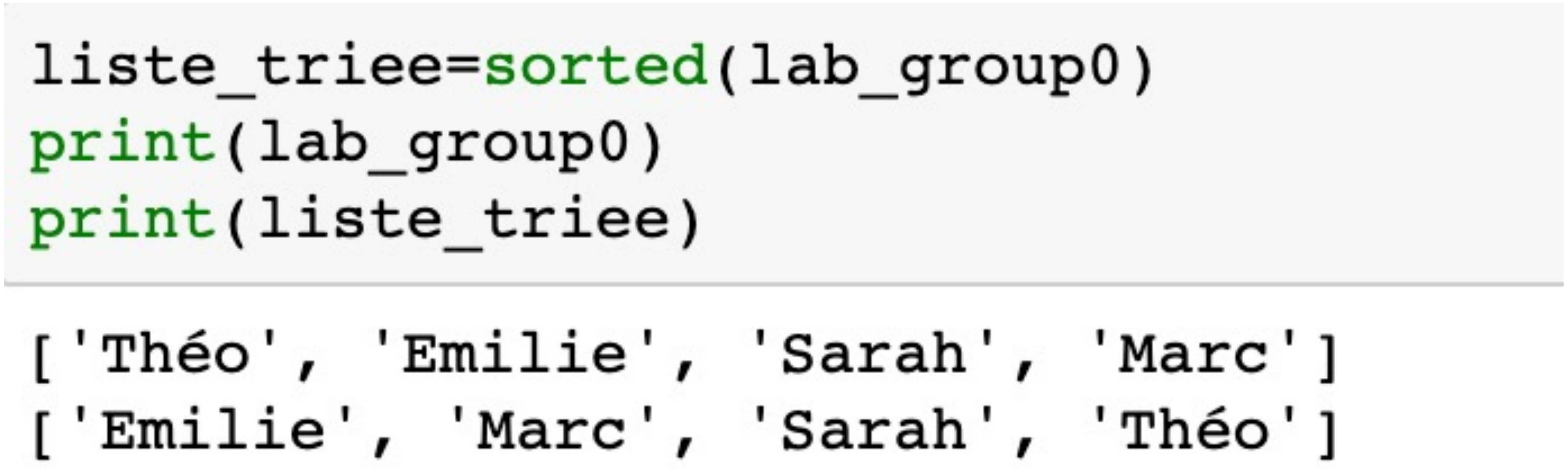
```
notes_étudiants = [12, 6.5, 9, 15]
notes_étudiants.sort(reverse=True)
print(notes_étudiants)
```

```
[15, 12, 9, 6.5]
```

Warnings

- list.sort() is a special way of writing things
- sort is a function. It is applied to list
- The content of the variable list is modified by the operation
- What if I want a function that keeps my list unsorted, but returns a new sorted list with the same elements?

Sorted

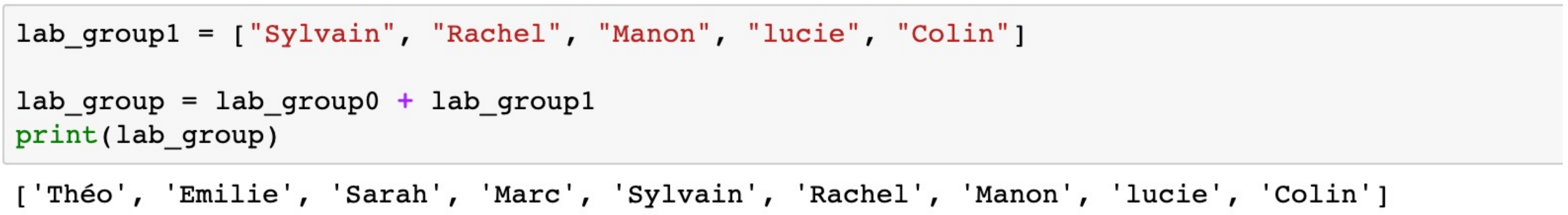


```
liste_triee=sorted(lab_group0)
print(lab_group0)
print(liste_triee)
```

```
['Théo', 'Emilie', 'Sarah', 'Marc']
['Emilie', 'Marc', 'Sarah', 'Théo']
```

Adding and removing elements

- my_list.append(34) will add 34 at the end. Like .sort(), .append(…) modifies my_list
- my_list.pop(i) will remove the element at index I
- The + operator works on lists, it concatenates them



```python
lab_group1 = ["Sylvain", "Rachel", "Manon", "lucie", "Colin"]

lab_group = lab_group0 + lab_group1
print(lab_group)
```

```
['Théo', 'Emilie', 'Sarah', 'Marc', 'Sylvain', 'Rachel', 'Manon', 'lucie', 'Colin']
```