# Listes Python

DIU-NSI, 11 juin 2019

Adeline Pierrot

# Rappels sur les listes en python

- Une liste se définit par la donnée explicite de ses éléments entre crochets `[ ]`.
- Ses éléments peuvent être de n'importe quel type :
```
>>> L = [1, 0.5, [1,2,3], "toto"]
```
- La fonction `len( )` renvoie le nombre d'éléments d'une liste :
```
>>> len(L)
4
```
- On accède aux éléments par leur indice, entre crochets.
  Le premier élément a pour indice 0.
  Le dernier élément a pour indice `len(L)-1` (ou simplement -1) :
```
>>> L[0]; L[-1]; L[len(L)]
1
"toto"
IndexError:  list index out of range
```

# Opérations sur les listes

- Contrairement aux t-uplets ou aux chaînes de caractères, les listes sont des objets dont on peut modifier un élément :

```
>>> L = [1, 0.5, [1,2,3], "toto"]
>>> L[0] = "changé"; print(L)
["changé", 0.5, [1,2,3], "toto"]
```

- L'opération + concatène deux listes ; une nouvelle liste est créée

```
>>> L + [3.14, 'a']
[1, 0.5, [1,2,3], "toto", 3.14, 'a']
>>> 2 * L
[1, 0.5, [1,2,3], "toto", 1, 0.5, [1,2,3], "toto"]
```

- Pour ajouter un élément à un liste, utiliser `append`:

```
>>> L.append(42)
[1, 0.5, [1,2,3], "toto", 42]
```

Attention, `L.append(x)` est une instruction, pas une expression. Ne pas écrire ~~`L=L.append(x)`~~ !

# Création de liste

- On peut mettre directement les éléments dans la liste:

```
>>> l1 = [4,-1,10,9,7]
```

- On peut partir de la liste vide et lui ajouter des éléments un par un avec append:

```
>>> l1=[]
for i in range(n):
     l1.append(0)
```

- On peut aussi initialiser la liste à 0 ainsi:

```
>>> l1=[0]*n
```

- On peut enfin créer une liste "par compréhension" (comme un ensemble mathématique):

```
>>> listeCarres = [ i**2 for i in range(10) ]
>>> print(listeCarres)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# Sous-liste

- On peut extraire une sous-liste d'une liste par un slicing :

  `L[i:j]` est la tranche de i (inclus) à j (exclu)

  `L[i:j:k]` est la tranche de i à j par pas de k

  Une copie est effectuée.

```
>>> L = [1, 0.5, 1+2j, [1,2,3], "toto"]

>>> L[1:-1:2]
[0.5, [1,2,3]]

>>> L[-1::-1]
["toto", [1,2,3], 1+2j, 0.5, 1]
```

# Copie de listes

- Attention à l'affectation de listes :

```
>>> l2 = L
>>> l2[-1] = 0 # Modification du dernier élément de l2
>>> print(l2)
[1, 0.5, 1+2j, [1,2,3], 0]
>>> print(L)
[1, 0.5, 1+2j, [1,2,3], 0] # Modifier l2 a aussi changé L
```

  L'affectation `l2 = L` a juste créé un alias : un nouveau nom référant à la même liste, dont une seule copie figure en mémoire.

- Pour recopier effectivement la liste, utiliser plutôt le slicing `L[:]` :

```
>>> l3 = L[:]
>>> l3[-1] = 0 # Modification du dernier élément de l3
>>> print(l3)
[1, 0.5, 1+2j, [1,2,3], 0]
>>> print(L)
[1, 0.5, 1+2j, [1,2,3], "toto"] # Modifier l3 n'a pas
changé L
```

# Listes de listes

```
L = [ [8,24,-2] , [7,3] , [-5, 67, -1, 0] ]
```

`L[0]` vaut `[1,24,-2]`
`L[0][1]` vaut 24
`L[1][0]` vaut 7
`len(L[2])` vaut 4
`len(L)` vaut 3

Parcours d'une liste de listes (par exemple pour remettre tous les éléments à 0):

```
for i in range(len(L)):
    for j in range(len(L[i])):
        L[i][j] = 0
```