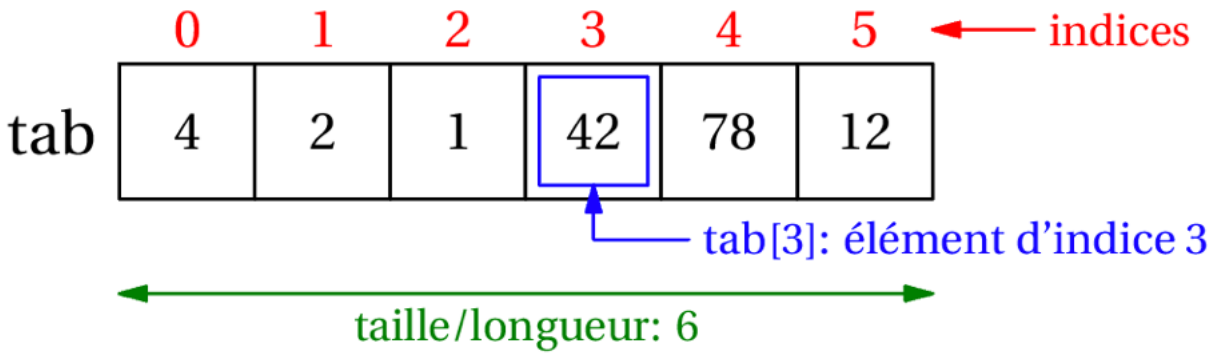
Langages 5 – Traitement de listes – cours

Tableau ou liste python

Un tableau (liste) permet de stocker de façon structurée plusieurs valeurs. En langage python, un tableau est stocké dans une liste (type list)

Une liste est une séquence ordonnée, rangés par numéro (indices). Le premier élément a l'indice zero 0, puis 1, 2, … Dans une liste python, on peut y mettre tout type d'objets.

On repère chaque élément du tableau par sa position, le numéro de la case, qu'on appelle **indice**.



(source: cgouygou.github.io)

- On utilise la notation **`tab[i]`** pour désigner l'élement d'indice i du tableau tab. Dans l'exemple ci-dessus, **`tab[3]`** désigne la valeur 42.
- Les indices commencent **toujours** à 0. Le premier élément d'une liste tab (non vide) est donc toujours tab[0].

Accéder à un élément de liste

L'élément d'une liste L à l'indice **`i`** s'écrit : `L[i]`
Il ne faut surtout pas confondre i (l'indice d'un élément) et **`tab[i]`** (l'élément de la liste tab d'indice i).

Remarques:

- on obtient la taille d'une liste avec la fonction len;
- un indice supérieur à len(tab) - 1 provoque une erreur IndexError: list index out of range (en dehors de la liste);
- on peut utiliser les indices négatifs: en particulier l'indice -1 permet d'accèder facilement au **dernier** élement.

En Python, les objets de type list sont modifiables (on emploie le mot *mutable*).

Longueur de liste: len()

La fonction `len` va retourner la longueur (le nombre d'éléments) de la liste.

```
>>> L = [5,4,3,2,1]
>>> len(L)
5
>>> L[4]
```

Noter que l'indice maximum dans la liste est egal à `len(L)-1`, car l'indice du premier élément est 0 et non 1.

Ajouter un élément à la fin: APPEND

La méthode append permet d'ajouter un élément en fin de liste (et donc d'augmenter la taille de la liste).

```
>>> famille = ["Bart", "Lisa", "Maggie"]
>>> famille.append("Homer")
>>> famille
['Bart', 'Lisa', 'Maggie', 'Homer']
>>> famille.append("Marge")
>>> famille
['Bart', 'Lisa', 'Maggie', 'Homer', 'Marge']
```

La méthode append ne prend **qu'un seul** paramètre: on ne peut donc ajouter qu'un élément à la fois. Si on a plusieurs éléments à ajouter à une liste, il faudra donc autant d'instruction append que d'éléments à ajouter.

Concaténation: +

Comme avec les chaînes de caractères, on peut concaténer deux listes avec l'opérateur + :

```
>>> ['truc', 'bidule'] + ['chose', 'machin']
['truc', 'bidule', 'chose', 'machin']
```

Parcours de listes (à connaître par coeur)⚓

Il existe principalement deux méthodes pour parcourir une liste: par ses éléments ou par les indices. Mais dans les deux cas on utilise une boucle `for`.

Parcours par élément

On l'a déjà rencontré sur la boucle for:

```
famille = ["Bart", "Lisa", "Maggie", "Homer", "Marge"]
for membre in famille:
    print(membre)
affichera:
Bart
Lisa
Maggie
Homer
Marge
```

Parcours par indice

Chaque élément étant accessible par son indice (de 0 à `len(liste) - 1`), il suffit de faire parcourir à une variable i l'ensemble des entiers de 0 à `len(liste) - 1`, par l'instruction range(len(liste)) :
Le code suivant affichera donc la même chose que le précédent:

```
famille = ["Bart", "Lisa", "Maggie",  "Homer", "Marge"]
for i in range(len(famille)):
    print(famille[i])
```

Accéder à une sous-liste : slice `L[i :j]`

La sous-liste issue de L entre les indices i et j+1 s'écrit : `L[i :j]`

Accéder à un élément dans une matrice

Une matrice est une liste de listes :

```
> M = [['A','B','C','D'], ['E','F','G','H'], ['I','J','K','L']]
```

M[0] permet d'acceder à la liste `['A','B','C','D']`
M[1] permet d'acceder à la liste `['E','F','G','H']`
M[2] permet d'acceder à la liste `['I','J','K','L']`

M[0][0] permet d'acceder au caractère `'A'`, M[0][1] -> `'B'`, M[0][2] -> `'C'`, …

Quelle expression vaut la chaîne de caractères 'H' ?
Réponses :
A- M[1][3]
B- M[3][1]

Test de la présence d'un élément dans une liste : IN

Le mot-clé in permet de tester la présence d'un élément dans une liste :

```
> L = [10, 10, 8, 6, 0, 9]
> 0 in L
True
>  11 in L
False
```

Suppression du dernier élément : POP

La méthode pop permet de supprimer le dernier élément d'une liste, celui dont l'indice est le plus élevé :

```
>>> L = ['lundi', 'mardi', 'mercredi']
>>> L.pop()
'mercredi'
>>> L
['lundi', 'mardi']
```

La mathode va supprimer l'élément de la liste, et le retourner. (Affichage dans la console)

Suppression d'un élément dans une position quelconque: REMOVE

La métode remove permet de supprimer la **première** occurrence de l'élément (et seulement la première).
À condition bien entendu que l'élément soit dans la liste...

```
>>> matieres = ["nsi", "maths", "anglais", "français", "maths"]
>>> matieres.remove("maths")
>>> matieres
["nsi", "anglais", "français", "maths"]
>>> matieres.remove("espagnol")
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Créer une liste par compréhension

simplifier le code pour le rendre plus lisible et donc plus rapide à écrire et plus simple à maintenir :

```
new_list = [function(item) for item in list if condition(item)]
```

Exemples :

```
# exemple 1
X = [I for I in range(11)]

# exemple 2
from random import randint
L = [randint(0,10) for i in range(10)]
L
[10, 10, 8, 6, 0, 9, 8, 5, 6, 1]
```