Les listes python

Les **listes**(ou **list**/ **array**) en **python**sont une variable dans laquelle on peut mettre plusieurs variables.

Créer une liste en python

Pour créer une **liste**, rien de plus simple:

>>> liste = []

Vous pouvez voir le contenu de la **liste**en l'appelant comme ceci:

>>> liste

<type 'list'>

Ajouter une valeur à une liste python

Vous pouvez ajouter les valeurs que vous voulez lors de la création de la **liste python**:

>>> liste = [1,2,3]

>>> liste

[1, 2, 3]

Ou les ajouter après la création de la liste avec la méthode append (qui signifie "ajouter" en anglais):

>>> liste = []

>>> liste

[]

>>> liste.append(1)

>>> liste

[1]

>>> liste.append("ok")

>>> liste

[1, 'ok']

On voit qu'il est possible de mélanger dans une même liste des variables de type différent. On peut d'ailleurs mettre une liste dans une liste.

Afficher un item d'une liste

Pour lire une liste, on peut demander à voir l'index de la valeur qui nous intéresse:

>>> liste = ["a","d","m"]

>>> liste[0]

'a'

>>> liste[2]

'm'

Le premier item commence toujours avec l'index 0. Pour lire la premier item on utilise la valeur 0, le deuxième on utilise la valeur 1, etc.

Il est d'ailleurs possible de modifier une valeur avec son index

>>> liste = ["a","d","m"]

>>> liste[0]

'a'

>>> liste[2]

'm'

>>> liste[2] = "z"

>>> liste

['a', 'd', 'z']

Supprimer une entrée avec un index

Il est parfois nécessaire de supprimer une entrée de la liste. Pour cela vous pouvez utiliser la fonction del .

>>> liste = ["a", "b", "c"]

>>> **del** liste[1]

>>> liste

['a', 'c']

Supprimer une entrée avec sa valeur

Il est possible de supprimer une entrée d'une liste avec sa valeur avec la méthode remove .

>>> liste = ["a", "b", "c"]

>>> liste.remove("a")

>>> liste

['b', 'c']

Inverser les valeurs d'une liste

Vous pouvez inverser les items d'une liste avec la méthode reverse .

>>> liste = ["a", "b", "c"]

>>> liste.reverse()

>>> liste

['c', 'b', 'a']

Compter le nombre d'items d'une liste

Il est possible de compter le nombre d'items d'une liste avec la fonction len .

>>> liste = [1,2,3,5,10]

>>> len(liste)

5

Compter le nombre d'occurrences d'une valeur

Pour connaitre le nombre d'occurences d'une valeur dans une liste, vous pouvez utiliser la méthode count .

>>> liste = ["a","a","a","b","c","c"]

>>> liste.count("a")

3

>>> liste.count("c")

2

Trouver l'index d'une valeur

La méthode index vous permet de connaitre la position de l'item cherché.

>>> liste = ["a","a","a","b","c","c"]

>>> liste.index("b")

3

Manipuler une liste

Voici quelques astuces pour manipuler des listes:

>>> liste = [1, 10, 100, 250, 500]

>>> liste[0]

1

>>> liste[-1] *# Cherche la dernière occurence*

500

>>> liste[-4:] *# Affiche les 4 dernières occurrences*

[500, 250, 100, 10]

>>> liste[:] *# Affiche toutes les occurences*

[1, 10, 100, 250, 500]

>>> liste[2:4] = [69, 70]

[1, 10, 69, 70, 500]

>>> liste[:] = [] *# vide la liste*

[]

Boucler sur une liste

Pour afficher les valeurs d'une liste, on peut utiliser une boucle:

>>> liste = ["a","d","m"]

>>> **for** lettre **in** liste:

... print lettre

...

a

d

m

Si vous voulez en plus récupérer l'index, vous pouvez utiliser la fonction enumerate .

>>> **for** lettre **in** enumerate(liste):

... print lettre

...

(0, 'a')

(1, 'd')

(2, 'm')

Les valeurs retournées par la boucle sont des **tuples.**

Copier une liste

Beaucoup de débutants font l'erreur de copier une liste de cette manière

>>> x = [1,2,3]

>>> y = x

Or si vous changez une valeur de la liste **y**, la liste **x**sera elle aussi affectée par cette modification:

>>> x = [1,2,3]

>>> y = x

>>> y[0] = 4

>>> x

[4, 2, 3]

En fait cette syntaxe permet de travailler sur un même élément nommé différemment

Alors comment copier une liste qui sera indépendante?

>>> x = [1,2,3]

>>> y = x[:]

>>> y[0] = 9

>>> x

[1, 2, 3]

>>> y

[9, 2, 3]

Pour des données plus complexes, vous pouvez utiliser la fonction deepcopy du module copy

>>> **import** **copy**

>>> x = [[1,2], 2]

>>> y = copy.deepcopy(x)

>>> y[1] = [1,2,3]

>>> x

[[1, 2], 2]

>>> y

[[1, 2], [1, 2, 3]]

Transformer une string en liste

Parfois il peut être utile de transformer une chaine de caractère en liste. Cela est possible avec la méthode split .

>>> ma\_chaine = "Olivier:ENGEL:Strasbourg"

>>> ma\_chaine.split(":")

['Olivier', 'ENGEL', 'Strasbourg']

Transformer une liste en string

L'inverse est possible avec la méthode " **join**".

>>> liste = ["Olivier","ENGEL","Strasbourg"]

>>> ":".join(liste)

'Olivier:ENGEL:Strasbourg'

Trouver un item dans une liste

Pour savoir si un élément est dans une liste, vous pouvez utiliser le mot clé **in**de cette manière:

>>> liste = [1,2,3,5,10]

>>> 3 **in** liste

**True**

>>> 11 **in** liste

**False**

La fonction range

La fonction range génère une liste composée d'une simple suite arithmétique.

>>> range(10)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Agrandir une liste par une liste

Pour mettre bout à bout deux listes, vous pouvez utiliser la méthode extend

>>> x = [1, 2, 3, 4]

>>> y = [4, 5, 1, 0]

>>> x.extend(y)

>>> print x

[1, 2, 3, 4, 4, 5, 1, 0]

Permutations

La permutation d'un ensemble d'éléments est une liste de tous les cas possibles. Si vous avez besoin de cette fonctionnalité, inutile de réinventer la roue, **itertools**s'en occupe pour vous.

>>> **from** **itertools** **import** permutations

>>> list(permutations(['a', 'b', 'c']))

[('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')]

Permutation d'une liste de liste

Comment afficher tous les cas possibles d'une liste elle-même composée de liste? Avec l'outil **product**de **itertools**:

>>> **from** **itertools** **import** product

>>> list(product(['a', 'b'], ['c', 'd']))

[('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd')]

Astuces

Afficher les 2 premiers éléments d'une liste

>>> liste = [1,2,3,4,5]

>>> liste[:2]

[1, 2]

Afficher le dernier item d'une liste:

>>> liste = [1, 2, 3, 4, 5, 6]

>>> liste[-1]

6

Afficher le 3ème élément en partant de la fin:

>>> liste = [1, 2, 3, 4, 5, 6]

>>> liste[-3]

4

Afficher les 3 derniers éléments d'une liste:

>>> liste = [1, 2, 3, 4, 5, 6]

>>> liste[-3:]

[4, 5, 6]

Vous pouvez additionner deux listes pour les combiner ensemble en utilisant l'opérateur + :

>>> x = [1, 2, 3]

>>> y = [4, 5, 6]

>>> x + y

[1, 2, 3, 4, 5, 6]

Vous pouvez même mutiplier une liste:

>>> x = [1, 2]

>>> x\*5

[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]

Ce qui peut être utile pour initialiser une liste:

>>> [0] \* 5

[0, 0, 0, 0, 0]

# Différence entre tuple et liste en Python

**L**es listes et les tuples stockent un ou plusieurs objets dans un ordre spécifique. Les objets stockés dans une liste ou un tuple peuvent être de n’importe quel type, y compris le type None.  
   
   
Les listes et les tuples sont pareils dans la plupart des contextes, mais il y a quelques différences que nous allons découvrir dans cet article.

###### **Différences de syntaxe**

La syntaxe de liste et tuple est légèrement différente. Les listes sont entourées par des crochets [] et tuples sont entouré par des parenthèses ().  
   
**Exemple :**

list\_color = **[**'blue', 'red', 'green'**]**

tuple\_color = **(**'blue', 'red', 'green'**)**

print**(**list\_color**)**

print**(**tuple\_color**)**

Cela produit le résultat suivant:

['blue', 'red', 'green']

('blue', 'red', 'green')

Nous avons défini un tuple nommé « tuple\_color » et une liste nommée « list\_color ». Dans la définition du tuple, nous avons utilisé les parenthèses () alors que dans la définition de liste, nous avons utilisé les crochets [].  
   
Python possède la fonction type() qui nous aide à connaître le type d’un objet créé. Nous pouvons l’utiliser comme suit:

print**(**type**(**list\_color**))**

print**(**type**(**tuple\_color**))**

Cela produit le résultat suivant:

**<class** 'tuple'**>**

**<class** 'list'**>**

###### **Mutable vs Immutable**

Les listes sont modifiables tandis que les tuples ne sont pas modifiables, c’est à dire que nous pouvons changer ou modifier les valeurs d’une liste mais nous ne pouvons pas changer ou modifier les valeurs d’un tuple.  
   
**Exemple :**

list\_color = **[**'blue', 'red', 'green'**]**

list\_color**[**1**]** = 'orange'

print**(**list\_color**)** #['blue', 'orange', 'green']

tuple\_color**[**1**]** = 'orange' #Erreur

Dans le code ci-dessus, nous avons affecté ‘orange’ à list\_color à l’index 1 et nous en avons trouvé ‘orange’ à l’index 1 en sortie. De plus, nous avons assigné ‘orange’ à tuple\_color à l’index 1 et nous avons eu une erreur genre, nous ne pouvons pas modifier le tuple en raison de sa nature immuable.

###### **Différence de taille**

Tuple a une taille plus petite que celle d’une liste, ce qui le rend un peu plus rapide, mais pas grand chose à mentionner jusqu’à ce que vous ayez un grand nombre d’éléments.  
   
**Exemple :**

list\_color = **[**'blue', 'red', 'green'**]**

tuple\_color = **(**'blue', 'red', 'green'**)**

print**(**tuple\_color.\_\_sizeof\_\_**())**

print**(**list\_color.\_\_sizeof\_\_**())**

Cela produit le résultat suivant:

48

64

###### **Autres différences:**

* Les listes ont une taille variable, les tuples ont une taille fixe.
* Au niveau des performances, les tuples ont une complexité de O(N) pour l’ajout, insertion et suppression alors que les listes ont une complexité de O(1) pour l’ajout, insertion et suppression.
* Les listes ont plus de fonctionnalités que les tuples.