**Chapitre 8 : Les Listes (Tableaux)**

1. **Introduction  :**

Il peut arriver que vous deviez lire, stocker, traiter et enfin imprimer des dizaines, voire des centaines, voire des milliers de nombres. Et alors? Avez-vous besoin de créer une variable distincte pour chaque valeur? Aurez-vous à passer de longues heures à rédiger des déclarations comme celle ci-dessous?

var1 = int(input())

var2 = int(input())

var3 = int(input())

var4 = int(input())

var5 = int(input())

var6 = int(input())

:

:

Si vous ne pensez pas que c'est une tâche compliquée, alors prenez un morceau de papier et écrivez un programme qui:

* lit cinq nombres,
* les imprime dans l'ordre du plus petit au plus grand (NB, ce type de traitement est appelé **tri** ).

Vous devriez constater que vous n'avez même pas assez de papier pour terminer la tâche.

Jusqu'à présent, vous avez appris à déclarer des variables capables de stocker exactement une valeur donnée à la fois. Ces variables sont parfois appelées **scalaires** par analogie avec les mathématiques. Toutes les variables que vous avez utilisées jusqu'à présent sont en fait des scalaires.

Pensez à la façon dont il serait commode de déclarer une variable qui pourrait **stocker plus d'une valeur** . Par exemple, cent, mille ou même dix mille. Ce serait toujours une seule et même variable, mais très large et vaste. Cela semble attrayant? Peut-être, mais comment gérerait-il un tel conteneur plein de valeurs différentes? Comment choisirait-il celui dont vous avez besoin?

Et si vous pouviez simplement les numéroter? Et puis dites: *donnez-moi la valeur numéro 2; attribuer le numéro de valeur 15; augmentez le nombre de valeur 10000* .

Nous allons vous montrer comment déclarer de telles **variables à valeurs multiples** . Nous allons le faire avec l'exemple que nous venons de suggérer. Nous allons écrire un **programme qui trie une séquence de nombres** .

1. **Les listes :**
2. **Syntaxe déclaration :**

numbers = [10, 5, 7, 2, 1]

Créons une variable appelée numbers; il est attribué avec non pas un seul numéro, mais il est rempli d'une liste composée de cinq valeurs (remarque: la **liste commence par un crochet ouvert et se termine par un crochet fermé** ; l'espace entre les crochets est rempli par cinq chiffres séparés par des virgules ).

Disons la même chose en utilisant une terminologie adéquate : numbers **est une liste composée de cinq valeurs, toutes des nombres** . Nous pouvons également dire que cette déclaration crée une liste de longueur égale à cinq (car il y a cinq éléments à l'intérieur).

Les éléments d'une liste **peuvent avoir différents types (contrairement au tableau dans l’algorithme)**. Certains d'entre eux peuvent être des nombres entiers, d'autres des flottants et d'autres encore peuvent être des listes.

Les listes aussi sont indexés ; chaque élément de la liste a son indice

Python a adopté une convention stipulant que les éléments d'une liste sont **toujours numérotés à partir de zéro**. Cela signifie que l'élément stocké au début de la liste aura le numéro zéro. Puisqu'il y a cinq éléments dans notre liste, le dernier d'entre eux se voit attribuer le numéro quatre. N'oublie pas ça.

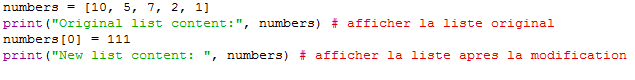
**Remarque :** **liste est une collection d'éléments, mais chaque élément est un scalaire. On peut le manipuler**

1. **Liste d’indexation  :**

Comment changez-vous la valeur d'un élément choisi dans la liste?

Nous allons **attribuer une nouvelle valeur** 111 **au premier élément** dans la liste. Nous le faisons de cette façon:

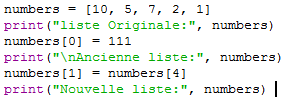
Numbers[0] = 111



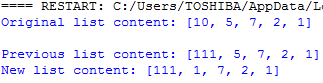
Résultat :



Et maintenant, nous voulons que **la valeur du cinquième élément soit copiée dans le deuxième élément** - pouvez-vous deviner comment le faire?



Résultat :



La valeur entre crochets qui sélectionne un élément de la liste est appelée **index** , tandis que l'opération de sélection d'un élément dans la liste est appelée **indexation** .

Nous allons utiliser la print()fonction pour imprimer le contenu de la liste chaque fois que nous apportons les modifications. Cela nous aidera à suivre chaque étape plus attentivement et à voir ce qui se passe après une modification particulière de la liste.

Remarque: tous les indices utilisés jusqu'à présent sont des littéraux. Leurs valeurs sont fixes lors de l'exécution, mais **toute expression peut également être l'index** . Cela ouvre de nombreuses possibilités.

* **Accéder au contenu de la liste**

Chacun des éléments de la liste est accessible séparément. Par exemple, il peut être imprimé:

print(numbers[0]) # accéder au premier élément

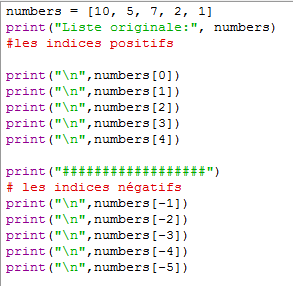
* **Les indices négatifs :**

Cela peut sembler étrange, mais les indices négatifs sont légaux et peuvent être très utiles.

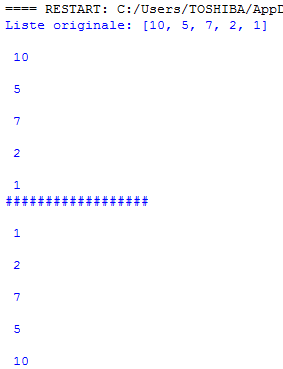
Un élément avec un indice égal à -1 **est le dernier de la liste**.

print(numbers[-1])

Exemple :



Résultat :



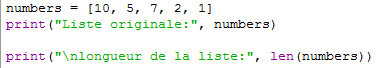
1. **La Fonction Len() :**

Les listes en python contrairement au d’autres language de programmation et l’algorithme sont dynamique et extensible donc La **longueur d'une liste** peut varier pendant l'exécution. De nouveaux éléments peuvent être ajoutés à la liste, tandis que d'autres peuvent en être supprimés. Cela signifie que la liste est une entité très dynamique.

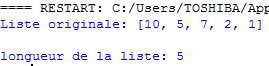
Si vous souhaitez vérifier la longueur actuelle de la liste, vous pouvez utiliser une fonction nommée **len()**(son nom vient de la *longueur* ).

La fonction prend le **nom de** la **liste comme argument et retourne le nombre d'éléments actuellement stockés** dans la liste (en d'autres termes - la longueur de la liste).

Exemple :



Résultat :

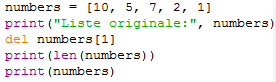


1. **Suppression des éléments de la liste  :**

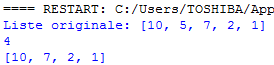
N'importe quel élément de la liste peut être **supprimé** à tout moment - cela se fait avec une instruction nommée del (delete). Remarque: c'est une **instruction**, pas une fonction.

Vous devez pointer l'élément à supprimer - il disparaîtra de la liste et la longueur de la liste sera réduite d'une unité.

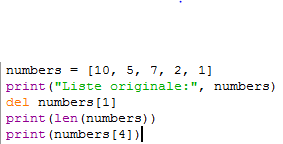
Exemple :



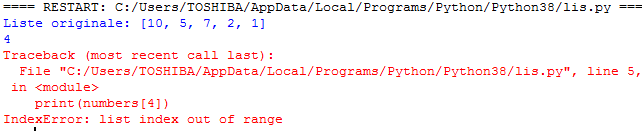
Résultat :



**Vous ne pouvez pas accéder à un élément qui n'existe pas -** vous ne pouvez ni obtenir sa valeur ni lui affecter une valeur.



Résultat :



1. **Fonctions vs Méthodes :**

Une **méthode est un type spécifique de fonction** - elle se comporte comme une fonction et ressemble à une fonction, mais diffère dans la façon dont elle agit et dans son style d'invocation.

Une **fonction n'appartient à aucune donnée** - elle obtient des données, elle peut créer de nouvelles données et elle (généralement) produit un résultat.

Une méthode fait tout cela, mais est également capable de **changer l'état d'une entité sélectionnée**

**Une méthode appartient aux données pour lesquelles elle fonctionne, tandis qu'une fonction appartient à tout le code** .

Cela signifie également que l'invocation d'une méthode nécessite une spécification des données à partir desquelles la méthode est invoquée.

Cela peut sembler déroutant ici, mais nous y reviendrons en détail lorsque nous nous plongerons dans la programmation orientée objet.

En général, un appel de fonction typique peut ressembler à ceci:

result = function(arg)

La fonction prend un argument, fait quelque chose et renvoie un résultat.

Un appel de méthode typique ressemble généralement à ceci:

result = data.method(arg)

Remarque: le nom de la méthode est précédé du nom des données propriétaires de la méthode. Ensuite, vous ajoutez un **point** , suivi du **nom** de la **méthode** et d'une paire de **parenthèses entourant les arguments** .

La méthode se comportera comme une fonction, mais peut faire quelque chose de plus - elle peut **changer l'état interne des données à** partir desquelles elle a été invoquée.

nous allons maintenant voir comment ajouter de nouveaux éléments à une liste existante. Cela peut être fait avec des méthodes appartenant à toutes les listes, pas par des fonctions

1. **Ajouter des éléments à la liste :**

On ajoute des éléments à la liste en utilisant deux méthodes : append() et insert()

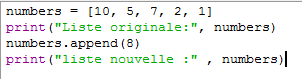
* La méthode append() :

Cette méthode permet d’ajouter le nouveau élément toujours à la fin de la liste :

Syntaxe :

Nomliste.append(valeur)

Exemple :



Résultat :



* La méthode insert() :

La insert()méthode est un peu plus intelligente - elle peut ajouter un nouvel élément **à n'importe quel endroit de la liste** , pas seulement à la fin.

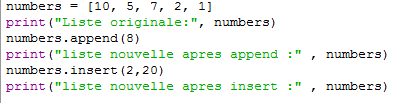
Syntaxe :

Nomliste.insert(position , valeur )

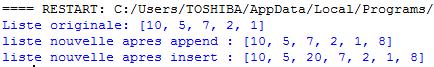
Il faut deux arguments:

* le premier indique l'emplacement requis de l'élément à insérer; note: tous les éléments existants qui occupent des emplacements à droite du nouvel élément (y compris celui à la position indiquée) sont décalés vers la droite, afin de faire de la place pour le nouvel élément;
* le second est l'élément à insérer.

Exemple :



Résultat :

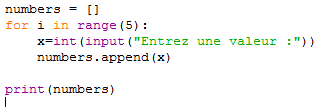


* **Création d’une liste vide :**

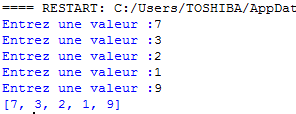
Vous pouvez **démarrer la vie d'une liste en la rendant vide** (cela se fait avec une paire de crochets vides), puis en lui ajoutant de nouveaux éléments si nécessaire.

Et on peut la remplir apres avec les données que l’utilisateur entre et cela se fait avec une boucle for

Exemple :



Résultat :



* **Autre façon pour supprimer les éléments de la liste :**

Il y a une autre façon pour supprimer les éléments de la liste en utilisant 2 méthodes : **Remove()** et **pop()**

**Remove() :** Supprime l’élément par sa valeur

Exemple : nomliste.Remove(valeur)

**Pop() :** Supprime l’élément par son indice

Exemple : nomliste.pop(indice)

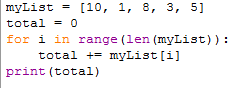
Il y a une méthode aussi pour vider carrément la liste : **clear()**

Exemple : nomliste.clear()

1. **Parcourir les listes :**

La boucle for a une variante très spéciale qui peut **traiter les listes** très efficacement.

Exemple : calculer la somme des éléments d’une liste (Méthode 1) :



Résultat :



Supposons que vous souhaitiez **calculer la somme de toutes les valeurs stockées** dans la liste myList.

Vous avez besoin d'une variable dont la somme sera stockée et attribuée initialement une valeur de 0 - son nom sera total. (Remarque: nous n'allons pas le nommer sum car Python utilise le même nom pour l'une de ses fonctions intégrées - sum (). L'utilisation du même nom serait généralement considérée comme une mauvaise pratique.) Ensuite, vous ajoutez à tout cela les éléments de la liste utilisant la boucle for. Jetez un oeil à l'extrait dans l'éditeur.

Commentons cet exemple:

• la liste se voit attribuer une séquence de cinq valeurs entières;

• la variable i prend les valeurs 0, 1, 2, 3 et 4, puis elle indexe la liste en sélectionnant les éléments suivants: le premier, le deuxième, le troisième, le quatrième et le cinquième;

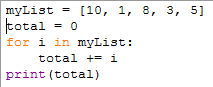
• chacun de ces éléments est additionné par l'opérateur + = à la variable total, donnant le résultat final à la fin de la boucle;

• notez la façon dont la fonction len() a été utilisée - elle rend le code indépendant de tout changement possible dans le contenu de la liste.

Exemple : calculer la somme des éléments d’une liste (Méthode 2) :

Mais la boucle for peut faire bien plus. Il peut masquer toutes les actions liées à l'indexation de la liste et fournir tous les éléments de la liste de manière pratique.

Cet extrait modifié montre comment cela fonctionne:



Que se passe t-il ici?

• l'instruction for spécifie la variable utilisée pour parcourir la liste (i ici) suivie du mot-clé in et du nom de la liste en cours de traitement (ici myList)

• la variable i se voit attribuer les valeurs de tous les éléments de la liste suivante, et le processus se produit autant de fois qu'il y a d'éléments dans la liste;

• cela signifie que vous utilisez la variable i comme copie des valeurs des éléments et que vous n'avez pas besoin d'utiliser des indices

• la fonction len() n'est pas non plus nécessaire ici.

1. **Trier ordonner une liste :**

Pour trier une liste , Python a ces propres mécanisme on ne va pas le faire manuellement comme on a déjà vu à l’algorithme , car il existe une fonction qui va trier la liste dans l’ordre croissant : c’est la méthode **sort()**

myList = [8, 10, 6, 2, 4]

myList.sort()

print(myList)

La sortie de l'extrait est la suivante:

[2, 4, 6, 8, 10]

Comme vous pouvez le voir, toutes les listes ont une méthode nommée sort(), qui les trie le plus rapidement possible.

Il existe aussi une autre méthode qui s’appelle **reverse()** qui renverse l’ordre de la liste .

myList = [8, 10, 6, 2, 4]

myList.reverse()

print(myList)

La sortie de l'extrait est la suivante:

[4,2,6,10,8]

* **Autres méthodes pratiques avec les listes :**

list.**extend**(*liste*) :

Étend la liste en y ajoutant tous les éléments de la liste en paramètre

list.**index**(*x*[, *start*[, *end*]])

Renvoie la position du premier élément de la liste dont la valeur égale *x* (en commençant à compter les positions à partir de zéro). Une exception [ValueError](https://docs.python.org/fr/3/library/exceptions.html" \l "ValueError" \o "ValueError) est levée si aucun élément n'est trouvé.

Les arguments optionnels *start* et *end* sont interprétés de la même manière que dans la notation des tranches et sont utilisés pour limiter la recherche à une sous-séquence particulière. L'indice renvoyé est calculé relativement au début de la séquence complète et non relativement à *start*.

list.**count**(*x*)

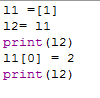
Renvoie le nombre d'éléments ayant la valeur *x* dans la liste.

list.**copy**()

Renvoie une copie superficielle de la liste.

1. **Trancher les listes**
2. **Copie des listes :**

**Reflexion :**



Le programme:

* crée une liste à un élément nommée l1;
* l'assigne à une nouvelle liste nommée l2;
* change le seul élément de l1;
* imprime l2.

La partie surprenante est le fait que le programme produira: [2]non [1], ce qui semble être la solution évidente.

Les listes (et de nombreuses autres entités Python complexes) sont stockées de différentes manières que les variables (scalaires) ordinaires.

Tu pourrais dire ça:

* le nom d'une variable ordinaire est le **nom de son contenu** ;
* le nom d'une liste est le nom d'un **emplacement mémoire où la liste est stockée** .

Lisez ces deux lignes encore une fois - la différence est essentielle pour comprendre de quoi nous allons parler ensuite.

L'affectation: l2 = l1 copie l’adresse de la liste , pas sa valeur . En effet, les deux noms ( l1 et l2) identifient le même emplacement dans la mémoire de l'ordinateur. La modification de l'un d'eux affecte l'autre, et vice versa.

1. **Les tranches d’une liste :**

En python on peut extraire une partie de la liste et la copier dans une autre liste ou la manipuler séparément.

Une tranche est un élément de la syntaxe Python qui vous permet de **faire une toute nouvelle copie d'une liste ou de parties d'une liste**.

Syntaxe :

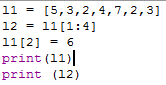
* Pour copier la liste complète : **liste[ : ]**



* Pour copier une partie de la liste : **liste[ i\_debut : i\_fin]** mais la valeur finale est exclus

**Remarque :** Contrairement à la copie d’une liste , la copie d’une tranche d’une liste copie les valeurs de la tranche et pas son adresse du coup la modification de la liste originale n’affecte pas la liste tranchée



1. **Les tranches négatives :**

En python on peut utiliser les indices négatifs pour trancher une liste

Voici comment **les indices négatifs** fonctionnent avec la tranche:

myList = [10, 8, 6, 4, 2]

newList = myList[1:-1]

toujours l’indice de fin n’est pas inclus

Si le i\_debut spécifie un élément situé plus loin que celui décrit par le i\_fin (du point de vue initial de la liste), la tranche sera **vide** :

myList = [10, 8, 6, 4, 2]

newList = myList[-1:1]

print(newList)

La sortie de l'extrait est: [].

Si vous omettez le i\_debut dans votre tranche, on suppose que vous voulez obtenir une tranche commençant à l'élément avec index 0.

En d'autres termes, la tranche de ce formulaire:

myList[:i\_fin]

est un équivalent plus compact de:

myList[0 : i\_fin]

**Exemple :**

myList = [10, 8, 6, 4, 2]

newList = myList[:3]

print(newList)

Ceci est la raison pour laquelle sa sortie est: [10, 8, 6].

De même, si vous omettez le i\_fin dans votre tranche, on suppose que vous voulez que la tranche se termine à l'élément avec l'index len(myList).

En d'autres termes, la tranche de ce formulaire:

myList[i\_debut : ]

est un équivalent plus compact de:

myList[i\_debut:len(myList)]

Regardez l'extrait de code suivant:

myList = [10, 8, 6, 4, 2]

newList = myList[3:]

print(newList)

Sa sortie est donc: [4, 2].

L'instruction del décrite précédemment est capable de **supprimer plus qu'un simple élément de liste à la fois - elle peut également supprimer des tranches**

myList = [10, 8, 6, 4, 2]

del myList[1:3]

print(myList)

**Remarque:** dans ce cas, la tranche **ne produit aucune nouvelle liste** !

La sortie de l'extrait est : [10, 4, 2].

La suppression de **tous les éléments** à la fois est également possible:

myList = [10, 8, 6, 4, 2]

del myList[:]

print(myList)

La liste est vide, et la sortie est: [].c’est équivalent de liste.clear()

1. **Les opérateurs in /not in :**

Python propose deux opérateurs très puissants, capables de **parcourir la liste afin de vérifier si une valeur spécifique est stockée dans la liste ou non** .

Ces opérateurs sont:

elem **in** myList

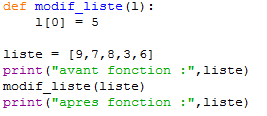
elem **not** in myList

Le premier d'entre eux ( **in** ) vérifie si un élément donné (son argument de gauche) est actuellement stocké quelque part dans la liste (l'argument de droite) - l'opérateur retourne *True* dans ce cas.

Le second ( **not in** ) vérifie si un élément donné (son argument de gauche) est absent dans une liste - l'opérateur retourne *True* dans ce cas.

1. **Les listes et les Fonctions :**

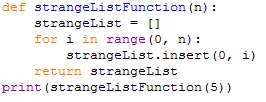
Comme n’importe quelle donnée on peut passer les listes comme paramètre dans les fonctions . le passage des liste se fait automatiquement par adresse et non pas par valeur comme les variables scalaire



**Resultat :**



la liste peut être un objet de retour d’une fonction :



**Résultat :**



**EXERCICES :**

**Exercice 1:** Écrire un algorithme qui demande successivement N nombres à l’utilisateur, et qui lui dise ensuite quel était le plus grand et le plus petit parmi ces N nombres et leurs positions :

Entrez le nombre numéro 1 : 12

Entrez le nombre numéro 2 : 14

…

Entrez le nombre numéro N : 6

Le plus grand de ces nombres est : 14, sa position : 2

Le plus petit de ces nombres est : 6, sa position 10

P.S : utilisez un tableau.

**Exercice 2:** Soit T un tableau de N réels. Écrire l’algorithme qui permet de calculer le nombre des occurrences d’un nombre X (c'est-à-dire combien de fois ce nombre X figure dans le tableau T).

**Exercice 3:** Écrire un algorithme qui augmente de 1 tous les éléments d’un tableau, le nouveau tableau sera affiché à l’écran.

Exemple : tab[5]={1,23,6,9,-1} devient tab[5]={2,24,7,10,0}.

**Exercice 4:** Écrire un programme constituant un tableau, à partir de deux tableaux de même longueur préalablement saisis. Le nouveau tableau sera la somme des éléments des deux tableaux de départ.

**Exercice 5:** Écrire un programme qui permet de lire N notes et de déterminer le nombre de celles qui sont supérieures à la moyenne. (En utilisant un tableau).

**Exercice 6:** Écrire un programme qui permet de lire N nombres et de déterminer le nombre de celles qui sont pairs et impairs (En utilisant un tableau).

**Exercice 7:** Écrire un algorithme qui lit la dimension N d'un tableau T du type **entier** (dimension maximale: 50 éléments), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Effacer ensuite toutes les occurrences de la valeur 0 dans le tableau T et tasser les éléments restants.

**Exercice 8:** Écrire un algorithme qui lit la dimension N d'un tableau T du type **entier** (dimension maximale: 20 éléments), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Ranger ensuite les éléments du tableau T dans l'ordre inverse sans utiliser de tableau d'aide.

**Exercice 9:** Écrire un algorithme qui lit la dimension N d'un tableau T du type **Entier** (dimension maximale: 50 éléments), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Copiez ensuite toutes les composantes strictement positives dans un deuxième tableau TPOS et toutes les valeurs strictement négatives dans un troisième tableau TNEG.

**Exercice 10:** Ecrire un programme qui demande à l'utilisateur de taper 10 entiers compris entre 0 et 20 qui seront stockés dans un tableau et qui affiche le nombre de fois qu'on a tapé un 0, le nombre de 1, le nombre de 2, ..., le nombre de 20.

**Exercice 11:** Vérifier que deux tableaux sont identiques (les éléments des deux tableaux seront saisis par l’utilisateur).