

*- Représentation des données : types construits -*

LES TUPLES ET LES LISTES AVEC PYTHON

Plan du chapitre

[I. INTRODUCTION](#_Toc16090200)

[II. LES TUPLES](#_Toc16090201)

[Définition](#_Toc16090202)

[Afficher les éléments d'un tuple](#_Toc16090203)

[Immuabilité d'un tuple](#_Toc16090204)

[Quel peut être l'intérêt d'un tuple ?](#_Toc16090205)

[III. LES LISTES](#_Toc16090206)

[Définition](#_Toc16090207)

[Affichage des divers éléments de la liste](#_Toc16090208)

[Ajout d'éléments dans une liste](#_Toc16090209)

[Suppression d'éléments dans une liste](#_Toc16090210)

[Construction d'une liste par compréhension](#_Toc16090211)

[Des listes de listes pour représenter des matrices](#_Toc16090212)

[Pour aller plus loin (non exigible)](#_Toc16090213)

[Trancher des listes (Slices)](#_Toc16090214)

[Des méthodes pour inverser et trier des listes](#_Toc16090215)

[Documentation en ligne python](#_Toc16090216)

[EXEMPLE DE TRACE D'UNE COURBE EN UTILISANT DES LISTES ET LA BIBLIOTHÈQUE MATPLOTLIB.PYPLOT](#_Toc16090217)

[IV. EXERCICES](#_Toc16090218)

# I. INTRODUCTION

Il est possible de "stocker" plusieurs grandeurs dans une même structure, ce type de structure est appelé une **séquence :** ensemble fini et ordonné d'éléments indicés de 0 à n-1 (si la séquence comporte n éléments).

Rappel : Nous avons vu dans les chapitres précédents qu'il est possible d'accéder à n'importe quel caractère d'une chaîne de caractères à l'aide de son indice de position (le premier caractère étant indexé 0).



Nous avons vu également qu'il est possible de balayer l'ensemble des caractères de la chaîne de caractères :

****

Une chaîne de caractère est donc bien un type de structure fini et ordonné d'éléments indexés de *0* à *len(chaine)-1*, c'est à dire une séquence.

Il existe d'autres types de séquences que nous allons découvrir ici, notamment les n-uplets (encore appelés tuples) et les listes.

[Retour au plan](#_top)

# II. LES TUPLES

## Définition

Un tuple est une collection d’éléments séparés par des virgules et généralement entouré de parenthèses même si ce n’est pas une obligation. Cela permet toutefois d’améliorer la lisibilité du code. Le tuple est une séquence **immuable** (une fois déclaré, le tuple ne peut plus être modifié) d’objets indicés qui peuvent être de tout type : entiers, flottants, chaînes de caractères…



[Retour au plan](#_top)

## Afficher les éléments d'un tuple

Comme pour une chaine de caractères, il est possible de parcourir directement les éléments du tuple à l'aide de l'instruction *for i in tup*

Une image contenant intérieur

Description générée automatiquement

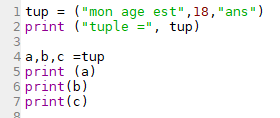
Et également par un parcours des indices :

Une image contenant intérieur, personne, mur, tenant

Description générée automatiquement

Comme avec une chaine de caractères, la fonction *len()* prenant un tuple comme argument renvoie le nombre d'éléments (longueur) du tuple.

Autre possibilité :



[Retour au plan](#_top)

## Immuabilité d'un tuple

Essayons de modifier un tuple prédéfini :

Une image contenant intérieur, personne

Description générée automatiquement

On constate que python refuse effectivement la modification d'un élément du tuple :

Une image contenant capture d’écran

Description générée automatiquement

[Retour au plan](#_top)

## Quel peut être l'intérêt d'un tuple ?

Nous souhaitons écrire une fonction qui prend en argument deux nombres entiers et nous renvoie le quotient et le reste de la division euclidienne du premier nombre par le second. Or nus avons vu qu'une fonction ne retourne qu'un seul élément ! Il est possible de solutionner ce problème en demandant à la fonction de renvoyer le résultat sous forme d'un tuple (quotient, reste).

Une image contenant capture d’écran

Description générée automatiquement

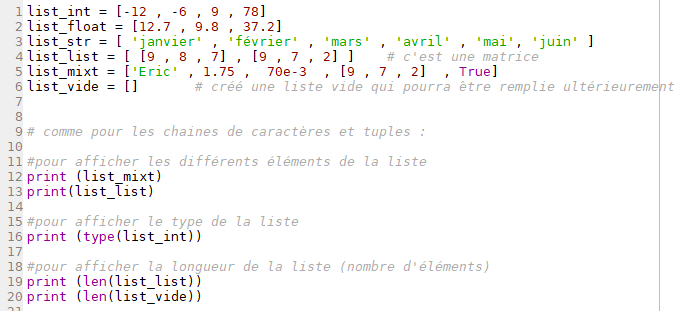
[Retour au plan](#_top)

# III. LES LISTES

## Définition

Tout comme *un tuple, une liste* est une collection d’informations qui peuvent être de même type ou de type différent. Les éléments de la liste sont séparés par des virgules et placés entre des crochets.

**Création de listes par affectation :**



La grande différence entre une liste et un tuple, c'est que la liste est **modifiable.**

[Retour au plan](#_top)

## Affichage des divers éléments de la liste

Comme pour les tuples, chaque élément est repéré dans la liste par un indice. Là aussi, l'indexation commence à 0 et non pas à 1.

Une image contenant capture d’écran

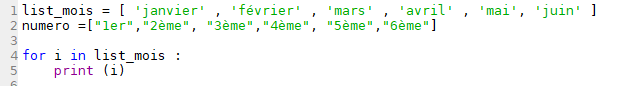
Description générée automatiquement

Il est donc possible d'afficher un par un l'ensemble des éléments d'une liste à l'aide d'une boucle for :

Une image contenant intérieur, mur

Description générée automatiquement

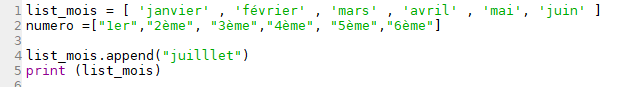
**Il est possible d'itérer directement les éléments de la liste** (comme pour les chaines de caractères):



[Retour au plan](#_top)

## Ajout d'éléments dans une liste

Pour rajouter un élément dans une liste, par exemple *'juillet'* à notre liste *liste\_mois*, il est possible d’utiliser la méthode *append()* (*to append* signifie *« ajouter »*). L'élément est ajouté en fin de liste. La méthode *append()* ne permet d'ajouter qu'un seul élément à la fois



Une autre méthode consiste à concaténer notre liste existante avec une autre liste. L'intérêt est que plusieurs éléments peuvent être rajoutés en même temps :



Attention, dans l'exemple qui suit le mois d'octobre sera rajouté en début de liste et non en fin de liste :

Une image contenant objet

Description générée automatiquement

Il est également possible d'utiliser la méthode extend() qui permet de concaténer non pas un seul élément mais une seconde liste.

Une image contenant objet

Description générée automatiquement

[Retour au plan](#_top)

## Suppression d'éléments dans une liste

Pour supprimer un élément dans une liste, par exemple l'élément de rang i, il est possible d’utiliser la commande *del* (*del* signifie « delete »).

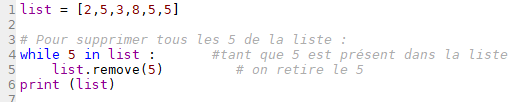


Il est possible de supprimer un élément dans un tableau à partir non pas de son indice, mais de sa valeur. La méthode *remove()* supprime uniquement la première occurrence trouvée.



Si l'élément n'est pas trouvé, un message d'erreur est retourné et le programme s'arrête, donc bien s'assurer auparavant que l'élément à supprimer est présent dans la liste.





[Retour au plan](#_top)

## Construction d'une liste par compréhension

Les compréhensions de listes fournissent un moyen de construire des listes de manière très concise. Une compréhension de liste consiste à placer entre crochets une expression suivie par une boucle *for* (ou plusieurs boucles imbriquées) et éventuellement un test conditionnel pour filtrer.

Supposons que l'on souhaite réaliser une liste contenant le carré des nombres de 1 à10 :

|  |  |
| --- | --- |
| Méthode classique :  Une image contenant intérieur  Description générée automatiquement | Méthode par compréhension : |

Il est possible de filtrer la liste précédente en ne prenant que le carré des nombres paires :

|  |  |
| --- | --- |
| Méthode classique :  Une image contenant intérieur  Description générée automatiquement | Méthode par compréhension : |

Création de la liste des caractères qui se trouvent dans une chaîne de caractère (itérable), dans l'ordre où ils sont rencontrés.

|  |  |
| --- | --- |
| Méthode classique : | Méthode par compréhension : |

Création de la liste des voyelles qui se trouvent dans une chaîne de caractère (itérable)

|  |  |
| --- | --- |
| Méthode classique :  Une image contenant intérieur  Description générée automatiquement  [Retour au plan](#_top) | Méthode par compréhension :  Une image contenant mur  Description générée automatiquement |

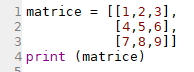
## Des listes de listes pour représenter des matrices

Voici un exemple de liste de listes :

Une image contenant objet

Description générée automatiquement

Pour des raisons de lisibilité, il est cependant plus commode de la représenter de la manière suivante, sous forme matricielle :



Pour cibler un élément de la matrice, on peut utiliser la notation avec des doubles crochets : matrice[ligne][colonne]

Attention : les numéros de ligne et colonne démarrent à 0.

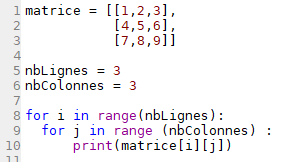


Pour affecter à cet élément la valeur 10 :

Une image contenant objet

Description générée automatiquement

Pour parcourir l'ensemble des éléments de la matrice, il faut utiliser 2 boucles *for* imbriquées :



[Retour au plan](#_top)

## Pour aller plus loin (non exigible)

### Trancher des listes (Slices)

On peut extraire d’un seul coup toute une partie de la liste :

liste[a:b] renvoie la sous-liste des éléments de rang *a* à *b* – 1 (l'élément de rang b est exclu).

List[:a] renvoie la sous-liste des éléments de rang *0* à *a* – 1 (l'élément de rang a est exclu).

List [a:] renvoie la sous-liste des éléments de rang *a inclus* jusqu'au dernier.

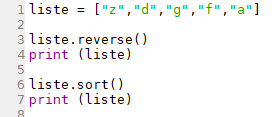
Une image contenant intérieur

Description générée automatiquement

[Retour au plan](#_top)

### Des méthodes pour inverser et trier des listes

La méthode reverse() permet d'inverser une liste et la méthode sort() permet de trier la liste



[Retour au plan](#_top)

### Documentation en ligne python

<https://docs.python.org/fr/3/tutorial/datastructures.html>

[Retour au plan](#_top)

# IV. EXEMPLE DE TRACE D'UNE COURBE EN UTILISANT DES LISTES ET LA BIBLIOTHÈQUE MATPLOTLIB.PYPLOT

Le programme suivant trace la courbe y=f(x²) en utilisant la bibliothèque matplolib.pyplot.

Pour cela il est nécessaire de remplir une liste des abscisses des points de la courbe nommée x et une liste des ordonnées nommée y :

Une image contenant capture d’écran

Description générée automatiquement

[Retour au plan](#_top)

# RÃ©sultat de recherche d'images pour "clipart python"V. EXERCICES

Exercice n°1 :

**Activité 1** (Manipulation de listes)**.**

*Objectifs : programmer des petites routines qui manipulent des listes.*

1. Programme une fonction rotation(liste) qui décale d’un rang tous les éléments d’une liste (le

dernier élément devenant le premier). La fonction renvoie une nouvelle liste.

Par exemple rotation([1,2,3,4]) renvoie la liste [4,1,2,3].

2. Programme une fonction inverser(liste) qui inverse l’ordre des éléments d’une liste.

Par exemple inverser([1,2,3,4]) renvoie la liste [4,3,2,1].

3. Programme une fonction supprimer\_element(liste,element) renvoyant une liste qui contient

tous les éléments sauf ceux égaux à l’élément spécifié.

Par exemple supprimer\_element([8,7,4,6,5,4],4) renvoie la liste [8,7,6,5] (tous les éléments

égaux à 4 ont été supprimés).

python : sorted()

**Activité 2** (Arithmétique)**.**

1. **Facteurs premiers.** Programmer une fonction facteurs\_premiers(n) qui renvoie la liste de tous

les facteurs premiers d’un entier *n* > 2. Par exemple, pour *n* = 12936, dont la décomposition en

facteurs premiers est *n* = 23 × 3 × 72 × 11, la fonction renvoie [2, 2, 2, 3, 7, 7, 11].

Le corps de l’algorithme est le suivant :

Tant que *d* < *n* :

Si *d* est un diviseur de *n*, alors :

ajouter *d* à la liste,

*n* devient *n/d*.

Sinon incrémenter *d* de 1.

2. **Liste de nombres premiers.** Écris une fonction liste\_premiers(n) qui renvoie la liste de tous

les nombres premiers strictement inférieurs à *n*. Par exemple liste\_premiers(100) renvoie la liste :

[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]

Pour cela, tu vas programmer un algorithme qui est une version simple du crible d’Ératosthène :

**Algorithme.**

• — Entrée : un entier *n* > 2.

— Sortie : la liste des nombres premiers *< n*.

• Initialiser liste qui contient tous les entiers de 2 à *n* − 1.

• Pour *d* allant de 2 à *n* − 1 :

Pour *k* parcourant liste :

Si *d* divise *k* et *d* 6= *k*, alors retirer l’élément *k* de liste

• Renvoyer liste.

*Indications.*

• Pars de liste = list(range(2,n)).

• Utilise liste.remove(k).

*Explications.* Voyons comment fonctionne l’algorithme avec *n* = 30.

• Au départ la liste est

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]

• On part avec *d* = 2, on élimine tous les nombres divisibles par 2, sauf si c’est le nombre 2 : on

élimine donc 4, 6, 8,. . . , la liste est maintenant : [2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29].

• On continue avec *d* = 3, on élimine les multiples de 3 (sauf 3), après ces opérations la liste est :

[2, 3, 5, 7, 11, 13, 17, 19, 23, 25,29].

• Avec *d* = 4, on élimine les multiples de 4 (mais il n’y en a plus).

• Avec *d* = 5 on élimine les multiples de 5 (ici on élimine juste 25), la liste devient

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29].

• On continue (ici il ne se passe plus rien).

[Retour au plan](#_top)