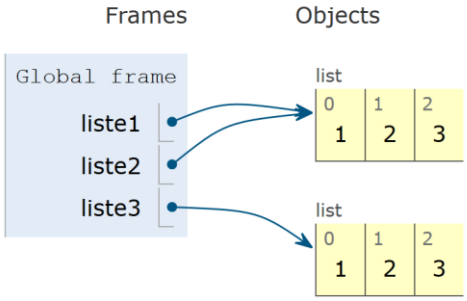
STRUCTURES DE DONNEES A 1 DIMENSION (1) : CORRIGE LES LISTES.

```
2
3  liste1 = [1 , 2 , 3 ]
4
5  liste2 = liste1
6
7  liste3 = [1 , 2 , 3 ]
8
9
10
11
```



- *Logiciels et sites internet* : Editeur et console Python (Thonny, VS Code etc.) ; pythontutor.com, franceioi.org.
- *Pré-requis pour prendre un bon départ :*

	☹	😐	☺	☺☺
Variables : initialisation, affectation, auto-affectation, incrémentation, etc.				
Boucles				
Tests conditionnels.				
Fonctions : transmission des arguments aux paramètres.				

VERIFIEZ TOUTES VOS REPONSES SUR ORDINATEUR !

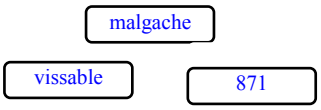
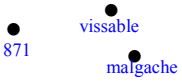
Ce cours Python fonctionne en pédagogie inversée. Ce livret est donc un post-cours complémentaire aux exos de France IOI, et doit être fait juste après les chapitres correspondants sur France IOI :

Exercices France IOI	*Chapitres de ce livret*
Niveau 2 Chapitres 2-3	Tous les chapitres de ce livret.

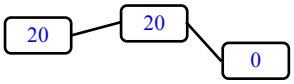
NOM et prénom : …… *Première spécialité NSI*

I. PROBLEMATIQUE DES LISTES : INFORMATIONS LIEES.

- Jusqu'à présent, nous avons vu les informations de manière atomique, isolément les unes des autres :

Type d'informations	*Exemple*	*Représentation schématique*	*Représentation informatique*	*Géométrie*	
Informations ponctuelles sans liens entre elles.	vissable malgache 871	**Cases indépendantes :** 	**Variables indépendantes :** a = 'vissable' b = 'malgache' c = 871	Points non alignés : 	*Dimension 0*

- Ces informations ponctuelles peuvent aussi être liées entre elles c-à-d avoir quelque chose en commun.
  Par exemple les notes de 3 élèves à un même contrôle d'Informatique :

Type d'informations	*Exemple*	*Représentation schématique*	*Représentation informatique*	*Géométrie*	
Informations liées.	Notes à un même contrôle : 20 ; 20 ; 0	**Cases liées :** 	?	Points alignés : 	*Dimension 1*

- Existe-t-il un objet informatique regroupant ces informations liées et permettant ainsi de les manipuler plus commodément ? La réponse est évidemment *oui !*
  En Python, cet objet regroupant des informations liées s'appelle tout simplement une *liste.*

II. DEBUTER AVEC LES LISTES.

Définition	• En Python, une liste est un **ensemble ordonné fini et modifiable d'objets indicés (c-à-d repérés par des nombres entiers)**. Autrement dit, une liste est une **séquence finie modifiable d'objets numérotés** à partir de **0**.
Vocabulaire	• Les objets contenus dans une liste s'appellent aussi les **éléments** de cette liste. Les éléments sont modifiables et pas forcément du même type (souvent oui : infos liées !). Une liste peut elle-même contenir d'autres listes ! • Le numéro attaché à chaque objet d'une liste s'appelle aussi **l'indice** de l'objet. Attention les indices d'une liste commencent toujours à ***0*** et non à *1* !
Définir une liste.	• Définir une liste, c'est écrire tous les éléments la composant selon 2 règles : **entre crochets [ ]** **séparés par des virgules « , ».** • Exemples : **[ ] : liste vide.** **Utile pour initialiser une liste à vide.** [1 > 0] : liste à un seul élément de type bool (booléen). ['Petit', 'papa', 'Noël'] : liste de 3 éléments tous de type str (string c-à-d caractère). [1 , 2 , 3 , 'soleil !'] : liste de *4* éléments, 3 de type *int,* 1 de type *str.* [True, [1 , 5], 2/3] : liste de *3* éléments, 1 de type *bool*, 1 de type *list*, 1 de type *float*.

Nommer une liste.	• C'est affecter la liste à un nom de variable bien choisi. Ex : point1 = [x1 , y1 , z1] point2 = [x2 , y2 , z2] liste_points = [point1 , point2]
Initialiser une liste.	• C'est la toute première affection d'une liste à un nom de variable. Initialiser une liste f à 3 zéros : *f = [0 , 0 , 0]* Initialiser une liste b à vide : *b = [ ]*
Type d'une liste.	• Une liste est un objet de type **list**. Exemple : type([1]) renverra *< class 'list' >*. « list » est donc un mot réservé de Python. Evitez de l'utiliser comme nom de qq chose !

III. CREER UNE LISTE.

Création simple d'une liste.	
• Revient à initialiser une liste en écrivant entre crochets tous ses éléments un par un. Ex : a = ['a' , 4]	Créer la liste liste_chiffres des chiffres usuels : *liste_chiffres = [0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9]*
Génération automatique d'une liste.	
• Liste de n éléments tous identiques : **[élément] * n ou n * [élément]** ⚠ *génération par accolement assez lente (voir p.13-IX).* ***Mieux par compréhension de liste (voir plus bas).***	• Ex : ['a'] * 2 ↔ ['a', 'a'] ; 3 * [1] ↔ [1 , 1 , 1] Générer une liste a de 20 zéros : *a = [0] * 20* Générer une liste b de 8 'ca' : *b = ['ca'] * 8*
• Suite arithmétique d'entiers : **range (début, fin, p) range(début, fin) range(fin)** Génère les entiers de **début inclus (0 si rien)** jusqu'à **fin exclue**, augmenté de p à chaque fois (+1 si rien).	• Ex : range (-1 , 5 , 2) vaut [-1 , 1 , 3]. range (1) ↔ *[0]* ; range (n) ↔ *[0, ... , n-1]* Générer [5 , 6 , 7] ↔ *range(5 , 8)* Générer [-1 , 3 , 7] ↔ *range(-1 , 8 , 4)*

Génération par compréhension (intention) de liste ou dit autrement liste image.

• En Maths, on connait déjà l'image d'un ensemble E par une fonction f. Cela se note :

{ tous les f(x) pour x ∈ E } Ex : { tous les x^2 pour x ∈ [0 ; 1] }.

Il existe la même chose en programmation : l'image d'une liste par une fonction (ou par une méthode).

Cela s'appelle la **compréhension de liste ou liste image.**

• Syntaxe : **[fonction(x) for x in liste] ou [x.méthode( ) for x in liste]**

L'image d'une liste est elle-même une liste ! → On parle donc de liste image. → Notation entre [ ].

La compréhension de liste permet donc de créer une nouvelle liste à partir d'une liste de départ :

[f(x) for x in liste] équivaut à la liste [f(élément0) , f(élément1) , etc. , f(élémentn)]

• Exemples : Soient 2 listes nb = [65 , 66] , ltr = ['A' , 'D'] et une fonction essai(k) renvoyant k – 3.

Voici l'évaluation (le « calcul ») pas à pas de 6 exemples de listes images (compréhensions de listes) :

① [x + 1 for x in nb]	② [essai(x) for x in range (2)]	③ [k.lower( ) for k in ltr]
→ [x + 1 for x in [65 , 66]]	→ [essai(x) for x in [0 , 1]]	→ [k.lower( ) for k in ['A' , 'D']]
→ [65 + 1 , 66 + 1]	→ [essai(0) , essai(1)]	→ ['A'.lower( ) , 'D'.lower( )]
→ [66 , 67]	→ [0 – 3 , 1 – 3] → [-3 , -2]	→ ['a' , 'd']

④ [chr(x) for x in nb]
→ [chr(x) for x in [65 , 66]]
→ [chr(65) , chr(66)]
→ [‘A’ , ‘B’]

⑤ [1 for lettre in ltr]
→ [1 for lettre in [‘A’ , ‘D’]]
(“1” est une fonction constante !)
→ [1 , 1]

⑥ [k + ‘C’ for k in ltr]
→ [k + ‘C’ for k in [‘A’ , ‘D’]]
→ [‘A’ + ‘C’ , ‘D’ + ‘C’]
→ [‘AC’ , ‘DC’]

De la même façon, écrire l’évaluation détaillée des intentions de liste suivantes :

[2*x – 3 for x in nb]
*→ [2*x – 3 for x in [65,66]]*
*→[2*65 – 3 , 2*66 – 3]*
→ [127 , 129]

[chr(x) for x in range(67,69)]
→ [chr(x) for x in [67 , 68]]
→ [chr(67) , chr(68)]
→ ['C' , 'D']

['bo' + j for j in ltr]
→ ['bo'+j for j in ['A','D']]
→ ['bo' + 'A' , 'bo'+ 'D']
→ ['boA' , 'boD']

[4 for x in range(2)]
→ [4 for x in [0 , 1]]
→ [4 , 4]

Evaluer directement les listes suivantes :

[1 for k in range(4)] → *[1 , 1 , 1 , 1]*

[0 for k in range (2)] → *[0 , 0]*

[triple(p) for p in [4 , -1 , 0]] → *[12 , -3 , 0]*

[2*m + 1 for m in range (4)] → *[1 , 3 , 5 , 7]*

[h.upper( ) for h in ['n','s','i']] → *['N','S','I']*

[ord(z) for z in ['a','b','c']] → *[97 , 98 , 99]*

['mot' + str(k) for k in range(1)] → *['mot0']*

Générer par compréhension de liste :

- Liste de 100 zéros :
  [0 for k in range(100]
- [1 , 4 , 9 , 16 , etc. , 100] :
  [nombre² for nombre in range(1 , 11)]
- ['a1' , 'b1'] à partir de ['a' , 'b'] :
  [lettre + '1' for lettre in ['a' , 'b']]
- Les 26 lettres majuscules de l’alphabet français :
  [chr(nombre) for nombre in range(65 + 0, 65 + 25 + 1)]

Listes en compréhension avec filtrage (condition) ou listes images restreintes.

Syntaxe : **[fonction (k) for k in liste if condition]** ou **[k.methode( ) for k in liste if condition]**

La condition restreint (filtre) la liste de départ ⇒ plus de souplesse dans la fabrication de listes image !

Ex : La liste des carrés des (entiers pairs ≤ 100) ↔ [k**2 for k in range (101) if k % 2 == 0]. Vérifions :

[k**2 for k in range (101) if k % 2 == 0]
→ [k**2 for k in [0 , 1 , etc. , 100] if k % 2 == 0]
→ [k**2 for k in [0 , 2 , 4 , etc. , 100]] car condition k % 2 == 0 ↔ k est dans la table de 2.
→ [0**2 , 2**2 , 4**2 , etc. , 100**2]
→ [0 , 4 , 16 , 36 , etc. , 10 000] c’est bien la liste des carrés des entiers pairs ≤ 100.

Evaluer les listes images filtrées suivantes :

- ['a'*nombre for nombre in [2 , -1 , 0] if nombre >= 0]
  → ['aa' , ' ']
- [2*k for k in range(14, 23) if k % 7 == 0]
  *→ [2*14 , 2*21]*
- [lettre.upper( ) for lettre in 'bonjour' if lettre < 'k']
  → ['B' , 'J']

Générer par intention de liste :

- Les cubes des (entiers ≤ 99 impairs) :
  *[nb**3 for nb in range(100) if nb%2 == 1]*
- Les carrés ≥ 245 des entiers ≤ 30 :
  *[b**2 for b in range(31) if b**2 >= 245]*
- Liste des lettres communes à 'bonjour' et 'jour' :
  [lettre for lettre in 'bonjour' if lettre in 'jour']

❶ Associer chaque liste en compréhension avec la seule expression qui lui correspond :

Aide : La fonction len(séquence) donne la longueur de la séquence (voir page suivante).

mot[k] donne le caractère d'indice k quand mot est une chaîne de caractères.

Mots d'une liste, mis en majuscules, seulement pour ceux qui commencent par '3'.	[k*2 for k in liste if len(k) < 3]
Mots d'une liste, mis en minuscules, seulement pour ceux de moins de 3 lettres.	[k**2 for k in range(3)]
Mots d'une liste, « doublés », seulement pour ceux de moins de 3 lettres.	[k.upper( ) for k in liste if k[0] == '3']
Mots d'une liste, mis en majuscules, seulement pour ceux de moins de 3 lettres.	[k*2 for k in range(3)]
La liste des doubles des (entiers de 0 inclus à 2 inclus).	[k.lower( ) for k in liste if len(k) < 3]
La liste des carrés des (entiers < 3).	[k.upper( ) for k in liste if len(k) < 3]

Remarque : Utiliser une boucle For ou une liste en compréhension ?

On peut toujours comme avant n'utiliser que des boucles For pour générer une liste à partir d'une autre.

Mais, quand on peut les utiliser, les compréhensions de liste sont plus rapides à écrire.

Exemple : Génération de la liste des 26 lettres minuscules de l'alphabet français grâce à la table Unicode :

par liste en compréhension	par boucle For (à compléter)
liste_minuscules = [chr(k) for k in range(97 + 0 , 97 + 26)]	liste_minuscules = [ ] for k in range(97 + 0 , 97 + 26) : liste_minuscules.append(chr(k))

❷ Application : Ecrire les boucles For équivalentes aux listes images suivantes puis écrire le résultat.

```
[ x**2 for x in range (3 , 6) ]
liste = [ ]
for x in range(3 , 9) :
    liste.append(x**2)
```
→ [9 , 16 , 25]

```
['point' + str(k) for k in range(3)]
liste = [ ]
for k in range (3) :
    liste.append('point' + str(k))
```
→ ['point0' , 'point1', 'point2']

```
[3*x for x in range(6) if x%2==0]
liste = [ ]
for x in range(6) :
    if x%2 == 0 :
        liste.append(3*x)
```
→ [0 , 6 , 12]

Inversement, écrire les listes images correspondant aux boucles suivantes, puis évaluer :

```
L = [ ]
for nb in range (7) :
    if nb % 3 == 0 :
        L.append( 'mot' + str(nb) )
```
L = ['mot' + str(nb) for nb in range(10) if nb % 3 == 0]

→ L vaut ['mot0' , 'mot3' , 'mot6'].

```
Liste_villes , aux = [ ] , [ 'Tana' , 'Majunga' , 'Tuléar' ]
for ville in aux :
    if ville[0] == 'T' :
        Liste_villes.append(ville)
```
Liste_villes = [ville for ville in aux if ville[0] = 'T']

→ Liste_villes vaut ['Tana' , 'Tuléar'].

IV. EVALUER UNE LISTE.

Une liste peut être composée de tout ce qui est évaluable : variables, expressions booléennes, expressions à évaluer, valeurs retournées par des fonctions, fonctions etc.

Evaluer une liste, c'est ainsi évaluer tous les éléments qui la composent en remplaçant les variables par leurs valeurs, en calculant les expressions, en remplaçant les fonctions par leurs valeurs retournées s'il y en a etc.

① Entourer la seule bonne réponse :

[1 + 2 , [3 , 3]] est évaluée (égale) à	[3 , [3 , 3]	[[3] , [3, 3]]	[3 , 3 , 3]	[3 , [3 , 3]]
[[1 , 2], print(3), 2 > 1] est évaluée (égale) à	[1 , 2 , 3 , True]	[[1 , 2] , 3 , True]	[[1 , 2],None,'2 > 1']	[[1 , 2],None,True]

Commentaire : la fonction print( ) affiche qqchose sur l'écran mais renvoie comme valeur None.

② A quoi sont évaluées (égales) les listes suivantes ?

from math import cos , pi nbjours , a = 5 , 2 [[1 , a] , nbjours + 1 , cos(pi)]	def essai1(k) : return k + 5 ['a' , print('oui') , 1 + essai1(5)]	def essai2(k) : y = k + 5 [essai2(5) , print(1) , 1 > 3]
[[1 , 2] , 6 , -1]	*['a' , None , 11]*	*[None , None , False]*

V. TIRER DE L'INFORMATION D'UNE LISTE.

A partir de maintenant, on utilisera souvent le mot « objet » à la place de « valeur » car en Python, tout (valeurs simples (int, float, str, bool, binary), listes, fonctions, etc.) est objet.

Nombre d'items d'une liste, longueur d'une liste : Fonction len( ).	
len(liste) • Renvoie le nombre d'éléments de liste, donc la longueur de la liste. • Diminutif de length en anglais qui veut dire longueur. Ex : len([1 , 3]) → 2 len(['a' , 'b' , 'c']) → 3	len([[0] , [1 , 2]]) → *2 (liste de 2 listes)* len(5 * [1]) → *5* len (**[** 5 * [1] **]**) → *1* len(range (3 , 13 , 5)) → *len([3 , 8]) → 2* len(**[**range(2)**]**) → *1* ; len(range(2,n)) → *n – 2* len([range(3) , [7] * 2 , 'a']) → *3*
Récupérer, utiliser un élément d'une liste.	
• Récupérer dans une variable a l'élément d'indice (numéro, position) k de liste : **a = liste[indice k]** • Attention, indice commence toujours à *0* et non à *1* ! • Les indices négatifs lisent la liste en sens inverse à partir du dernier élément : liste[-1] → dernier item. Attention donc aux indices pouvant devenir négatifs ! • Si l'indice demandé hors de la liste → IndexError.	Soient les 2 listes pos et point suivantes : pos = [x, y, z] et point = [pos , diam , [R , G , B]] pos[2] → z point[1] → diam pos[-1] → z point[2][1] → G point[0][2] → z pos[1] → *y* point[2][0] → *R* pos[-3] → *x* point[3] → *IndexError* *pos[0]* → x *point[0]* → pos point*[0][0]* → x *point[2][1]* → B

Parcourir une liste L à l'aide d'une boucle For.

Parcours séquentiel classique à l'aide des indices.	Parcours direct sur les éléments (sans les indices).
for indice in range(longueur de L) : **traitement sur L[indice]** Grâce au range, indice parcourt la liste des indices. A chaque tour, un traitement utilisant les éléments repérés par leur indice peut être effectué.	**for élément in L :** **traitement utilisant élément** élément parcourt tous les objets de la liste L. A chaque tour, un traitement utilisant simplement chaque élément peut être effectué.

Ex : L = ['je','tu','il'] for k in range(2) : L[k] = L[k+1]	Que vaut L ? *L[k] = L[k+1] signifie qu'un item est remplacé par son suivant →* *['tu','il','il']*	Ex : for lettre in ['a' , 'b' , 'c'] : print(lettre + 'd', end=',')	Que s'affiche-t-il ? *ad,bd,cd,*

- **Le parcours par indice est plutôt indiqué lorsqu'on a besoin des éléments et de leur indice.**
- **Le parcours par éléments est indiqué lorsqu'on a simplement besoin des éléments sans leur indice.**

① Ecrire une fonction qui en entrée reçoit une liste et une valeur, et en sortie renvoie le nombre de fois où cette valeur est présente dans la liste.

Ici pas besoin des indices donc parcours direct de la liste.

```
def compter_occurences(liste,valeur) :
    nb_occurences = 0
    for élément in liste :
        if élément == valeur :
            nb_occurences = nb_occurences+1
    return nb_occurences
```

Remarque :

Cette fonction équivaut à la fonction count() plus bas.

② Ecrire une fonction qui en entrée reçoit une liste et une valeur, et en sortie renvoie la liste de tous les indices où se trouve cette valeur dans la liste.

Ex : Pour [1,2,1,5] et 1 en entrée, [0,2] est renvoyé.

```
def trouver_indices(liste,valeur) :
    liste_indices = []
    for k in range(len(liste)) :
        if liste[k] == valeur :
            liste_indices.append(k)
    return liste_indices
```

Remarque : Cette fonction est une amélioration de la méthode .index() plus bas.

③ Ecrire une fonction qui en entrée reçoit une liste d'entiers, et en sortie renvoie le maximum de cette liste.

```
def trouver_max(itérable) :
    max = itérable[0]
    for élément in itérable :
        if élément>max :
            max=élément
    return max
```

Commentaires :

- *Ici on ne demande pas le ou les indices où se trouvent le max, donc on a une boucle For directement sur les éléments de la liste. Si on voulait en plus les indices où se trouvent ce max, il aurait fallu une boucle For sur les indices de la liste.*
- *Cette fonction équivaut à la fonction python max( ).*

Récupérer le nombre de fois où apparait un objet dans une liste : Méthode .count( ).

a = liste.count(objet)

• Récupère dans a le nombre de fois où objet est présent dans liste.

Si objet non dans la liste → renvoie la valeur 0.

• Ex : Soit la liste A = [3 , 10 , 1 + 2 , '3' , sqrt(9)].

b = A.count(3)

b recevra 3 car la valeur cherchée 3 (ou 3.0, mais pas '3') apparaît 3 fois dans A après évaluation.

• def essai (k) :

b = k + 1

prix = [5 , [5] , essai(4) , sqrt(25) , '5' , 5.0]

b = prix.count(5) Combien vaut b ? *3*

• Ecrire l'instruction qui stockera dans c le nb d'occurrences de None dans prix. Valeur de c ?

c = prix.count(None) → c vaut 1 (essai(4) !).

Juste savoir si oui ou non un objet est présent dans une liste : opérateur « in ».

Objetcherché in liste

• Expression booléenne qui renvoie True si Objetcherché est dans liste, False sinon.

• Ex : Soit la liste ages = [3 , 7 + 2 , 7 – 2].

if **5 in ages** :
print ('Oui')
else : print ('Non')

Ce programme affiche 'Oui'. En effet, l'objet cherché 5 est bien dans la liste ages.

def essai (k) :

b = k + 1

prix = ['5' , [5] , essai(4)]

Ecrire une instruction affichant « hourrah » si 5 est dans la liste prix. « hourrah » s'affiche-t-il ?

if 5 in prix :

print('hourrah') → pas d'affichage ici !

Récupérer la 1ère position d'un objet dans une liste : Méthode .index( ).

a = liste.index(objetcherché)

• Récupère dans a le 1er indice où objetcherché apparaît dans liste.

Si objetcherché non dans la liste → ValueError.

• Ex : Soit la liste ages = [3 , 10 , 1 + 2 , 3 , sqrt(9)].

b = ages.index(3)

b recevra la valeur 0 car la valeur cherchée 3 (ou 3.0) apparaît la première fois à la position 0 dans la liste ages.

def essai (k) :

b = k + 1

prix = [1 , essai(4) , 5.0 , [5] , sqrt(25)]

• b = prix.index(5)

Combien vaut b ? *2 grâce à 5.0.*

• On cherche 4 dans la liste prix.

Ecrire l'instruction qui affecte à c le 1er indice de cette valeur cherchée 4. Combien vaut c ?

c = prix.index(4) → c vaut ValueError.

VI. MODIFIER LES ITEMS OU L'ORDRE DES ITEMS D'UNE LISTE.

Modifier, remplacer un élément d'une liste.

liste[indice k] = nouvelle valeur

• Nouvelle valeur affectée à l'item d'indice k.

• Si indice k hors de la liste → IndexError.

Ex : Soit la liste age = [2 , 1 , 2.0].

age[2] = 5 → age vaut [2 , 1 , 5]

Ecrire l'instruction qui change le 1er item par 4 :

age[0] = 4

Inverser une liste : Méthode .reverse( ) ou Fonction reversed( ).	
liste.reverse( ) • Modifie sur place liste elle-même en la renversant. **Pas d'affectation !** **a = list(reversed(liste))** • **Crée une nouvelle liste renversée.** **A affecter** à elle-même ou à un autre nom (a ici).	• Ex : Soit la liste age = [1 , 2 , 3]. age.reverse( ) → age est modifiée et vaut [3 , 2 , 1]. b = list(reversed(age)) → b vaut [3 , 2 , 1] et age est conservée. • On veut inverser « sur place » une liste. Faut-il utiliser la *méthode liste.reverse( )* ou la fonction reversed(liste) ?
Trier une liste par ordre croissant : Méthode .sort( ) ou Fonction sorted( ).	
Tri par ordre croissant. Si liste ne peut être triée → TypeError. **liste.sort( )** • Modifie sur place la liste elle-même en la triant. **Pas d'affectation !** **a = sorted(liste)** • **Crée une nouvelle liste triée**. **A affecter** à elle-même ou à un autre nom (a ici).	• Soit la liste age = [1 , 5 , 3]. age.sort( ) → age est modifiée et vaut [1 , 3 , 5]. b = sorted(age) → b vaut [1 , 3 , 5] et age est conservée. • On veut trier une liste tout en la conservant. Faut-il utiliser la méthode liste.sort( ) ou la *fonction sorted(liste)* ?

VII. RACCOURCIR OU ALLONGER UNE LISTE.

Vider une liste : Méthode .clear( ).	
liste.clear( ) Efface tous les éléments de la liste elle-même qui devient alors vide ([ ]). **Pas d'affectation !**	Soit la liste a = [1 , 2 , 3]. a.clear( ) → a vaut maintenant *[ ].*
Supprimer un élément connaissant soit sa position (Mot clé del), soit sa valeur (Méthode .remove( )).	
• **del liste[indice k]** Supprime de la présente liste l'élément d'indice k. **Pas d'affectation !** Si indice k hors de la liste → IndexError. • **liste.remove(objet)** Supprime de la présente liste **seulement la 1ère fois** où objet apparait dans liste. **Pas d'affectation !** Si objet non dans la liste → ValueError.	Ex : Soit la liste age = [2 , 1 , 2.0]. • En repartant de la liste age à chaque fois : del age[2] → age vaut [2 , 1]. age.remove(2.0) → age vaut [2 , 1]. del age[3] → *IndexError* age.remove(2) → *[1 , 2.0]* • Soit une liste b. Ecrire l'instruction qui enlève de b : - le 1er item qui vaudra toto : *b.remove('toto')* - l'item d'indice final : *del b[(len(b)–1)] ou del b[-1].*

Ajouter un seul élément en toute fin de liste (à droite) : Méthode .append( ).

liste.append(objet)

• Ajoute tel quel objet à la fin de la liste elle-même.

Pas d'affectation !

• Ex : Soient les listes a = [1 , 2 , 3] et b = [4 , 5]

a.append(4) → a vaut [1 , 2 , 3 , 4]

a.append(b) → a vaut [1 , 2 , 3 , [4 , 5]]

• Attention : .append( ) n'ajoute qu'1 seul objet !

• Soit la liste a = [1 , 2 , 3].

```
for k in range (3) :
    a.append( 3 – k )
```

Que vaut maintenant a ?

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

• Soit b = [ ]. Compléter ci-dessous afin que b soit la liste des 10 premiers entiers en ordre inverse.

```
for k in range (10) :
    b.append(10 – 1 – k)
```

Insérer un élément dans une liste : Méthode .insert( ).

liste.insert(indice , objet)

• Insère tel quel objet à la position indice de la liste elle-même. **Pas d'affectation !**

• Ex : Soient les listes a = [1 , 2 , 3] et b = [4 , 5]

a.insert(1 , 4) → a vaut [1 , 4 , 2 , 3]

a.insert(0 , b) → a vaut [[4 , 5] , 1 , 2 , 3]

• Attention : ne pas inverser indice et objet !

Ajouter au début revient à liste.insert(*0* , objet).

• Soit la liste a = [1 , 2 , 3]. Dans chq cas, que vaut a ?

```
for k in range (2) :
    a.insert( 1 , k )
```

a → [1 , 1 , 0 , 2 , 3]

```
for p in range (2) :
    a.insert( p , 5 – p )
```

a → [5 , 4 , 1 , 2 , 3]

• Soit la liste b = [1 , 10]. Compléter ci-dessous afin que b soit la liste des entiers ⟦ 1 , 10 ⟧ de 1 jusqu'à 10.

```
for k in range (2 , 10) :
    b.insert(k – 1 , k)
```

VIII. COPIER UNE LISTE : SOURCE DE NOMBREUSES ERREURS !

La recopie d'objets et de listes en particulier est très importante car c'est ce qui s…sse par exemple lorsque des arguments sont affectés aux paramètres d'une fonction lors de l'appel de cell… ci.

Comparons sur pythontutor la copie entre 2 variables simples (V1 vers V2) et entre 2 listes (L1 vers L2).

Script 1		Que contiennent V2 et V1 ?	Script 2		Que contiennent L2 et L1 ?
	V1 = 5 V2 = V1 V2 = 'a'	V2 → *'a'* V1 → *5*		L1 = [5] L2 = L1 L2.append('a')	L2 → *[5 , a]* L1 → *[5 , a]*
V1 et V2 indépendantes ? *Oui* Logique ? *Oui*			L1 et L2 indépendantes ? *Non* Logique ? *Non !*		
☺ Ok tout baigne !			☹ Mais comment est-ce possible ?!		

A. Véritable représentation en mémoire des variables et des valeurs :

En fait, il faut revoir la représentation qu'on avait faite des variables comme cases mémoire nommées.

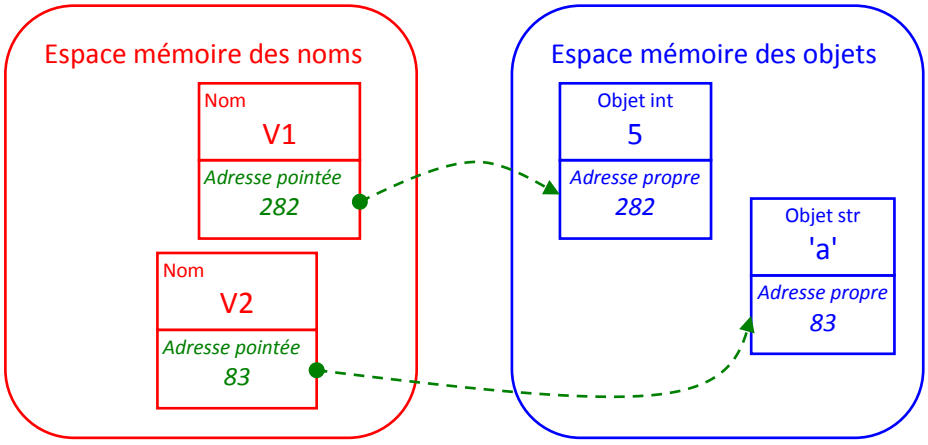
Une variable est en fait l'association d'un nom avec un objet (une valeur).

- Le nom est en fait une case mémoire située dans un espace spécial de la mémoire : l'espace mémoire des noms qui regroupe tous les noms de variables, les noms des listes, les noms de fonctions etc. Chaque case mémoire de nom contient l'adresse de la case mémoire de l'objet auquel le nom est affecté. On dit que le nom pointe vers l'objet.
- La valeur est une case mémoire située dans un autre espace mémoire : l'espace mémoire des objets. Chaque case mémoire contenant un objet est repérée par une adresse propre.

Schémas de l'état mémoire après l'exécution de chaque script plus haut (voir aussi sur pythontutor.com) :

Script 1

On voit bien que les variables V1 et V2 sont indépendantes car leurs noms ne pointent pas vers le même objet.

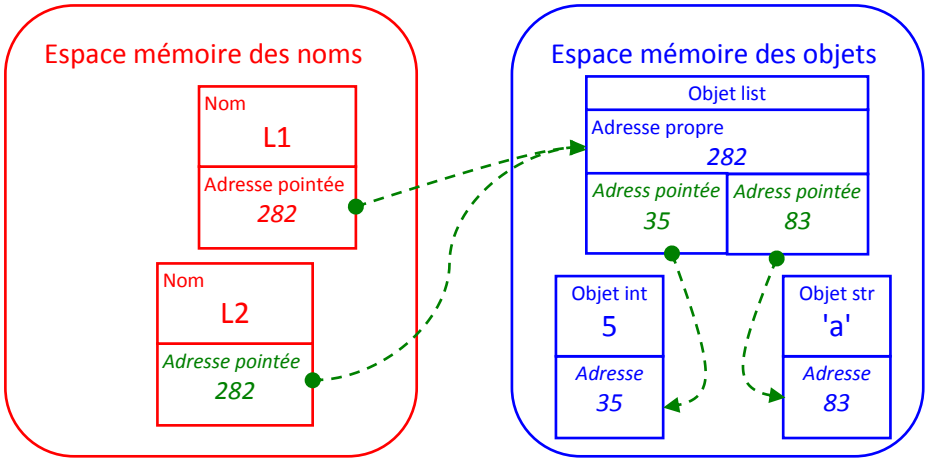


Script 2

L'affectation L2 = L1 fait que les 2 noms pointent vers le même objet liste !

L2 et L1 sont maintenant 2 noms du même objet. On parle d'alias.

Donc modifier l'objet sous un nom revient au même que le modifier sous l'autre nom !



B. Copie par le nom entre 2 listes : danger !

Cette simple copie par le nom entre 2 listes (L2 = L1) est source de nombreuses erreurs pour les élèves qui pensent créer une copie de travail indépendante et se retrouvent en fait avec 2 listes liées !

• **Ce problème est particulièrement vicieux lorsqu'on transmet une liste en argument lors de l'appel d'une fonction :** cette copie par le nom se fait silencieusement, sans instruction écrite, lors de l'affectation des arguments aux paramètres de la fonction (voir cours Fonctions p.4.). • Exemple frappant ci-contre : Lors de l'appel de la fonction essai1 à la ligne 4, le nom a de la liste est affecté au nom listelocale. Donc les 2 noms vont pointer vers la même liste [1] qui a maintenant 2 noms (alias). Et donc l'ajout en ligne 2 sera visible dans listelocale évidemment mais aussi, plus déroutant, dans a ! Malgré l'absence de return ou de global !!	Exemple frappant : 1. def essai1 (listelocale) : 2. listelocale.append(3) 3. a = [1] 4. essai1(a) listelocale ? → *[1 , 3]* a ? → *[1 , 3] !*

Fonction avec transmission de liste en argument → attention !

C. Copies non liées : méthode sécurisée .copy( ) :

Créer une copie non liée (un clone) d'une liste : Méthode .copy.

a = liste.copy( ) • **Crée une nouvelle liste** ayant les mêmes items que liste : un clone indépendant. **A affecter à un autre nom (a ici).** • Ex : h = [1 , 2] j = h.copy( ) Un nouvel objet [1 , 2] non lié à h est créé, puis affecté à j. En particulier, si je modifie la liste j, la liste h reste inchangée.☺ C'est comme si on avait fait j = [1 , 2] mais surtout pas j = h !	def essai2 (param) : listelocale = param.copy( ) listelocale.append(3) a = [1] essai2(a) listelocale ? → *[1 , 3]* liste a ? → *[1]*

Fonction recevant une liste en argument : quand et comment travailler en toute sécurité.

Quand ? Lorsqu'on veut garder intacte une liste transmise en argument à une fonction.

Comment ? Avec la méthode .copy( ), il faut copier la liste en paramètre (qui réceptionnera la vraie liste transmise en argument) vers une listelocale.(voir ligne 2 fonction essai2 ci-dessus)

Puis on travaille tranquillou avec listelocale : pas d'effets de bord garantis sur la liste d'origine !

Ne pas oublier de retourner la listelocale pour profiter du traitement de la fonction !

Application : Ecrire une fonction qui reçoit une liste en paramètre et renvoie une liste non liée avec les mêmes éléments + 'début' en début de liste + 'fin' en fin de liste. Ex : Pour a = [1, 2], la fonction renverra : → ['début' , 1 , 2 , 'fin'].	*def ajout_debut_fin(liste) :* *liste_locale = liste.copy( )* *liste_locale.insert(0 , 'début')* *liste_locale.append('fin')* *return liste_locale*

IX. FUSIONNER 2 LISTES.

Etendre une liste existante avec les éléments d'une autre liste : Méthode .extend( ).

liste1.extend(liste2)

• Etend liste1 en rajoutant à la fin de liste1 elle-même (à droite) tous les éléments de liste2.

Pas d'affectation !

• Ex : soit la liste a = [1 , 2].

a.extend([3 , 4]) → a vaut [1 , 2 , 3 , 4].

a.extend([[3 , 4]]) → a vaut [1 , 2 , [3 , 4]].

⚠ a.extend(5) → TypeError : 5 n'est pas une liste !

• Soient a = [k*2 for k in range(-1,3,2)] et b = [1 , 0].

a vaut *[-2 , 2].*

b.extend(a) → b vaut *[1 , 0 , -2 , 2].*

a.extend(b) → *a vaut [-2 , 2 , 1 , 0] donc a[2] vaut 1.*

• Ecrire un script qui étend la liste b avec son inverse :

c = b.copy( )	*c = reversed(b)*
c.reverse( )	*b.extend(c)*
b.extend(c)	

Créer une nouvelle liste à partir de la concaténation (accolement) de 2 listes : liste1 + liste2.

liste3 = liste1 + liste2

• **Crée un nouvel objet liste3** composé d'abord des éléments de liste1 puis de ceux de liste2.

• Ex : liste1 = [1 , 2] liste2 = [[3 , 4]] liste3 = [[1 , 2]]

c = liste1 + liste2 → c vaut [1 , 2 , [3 , 4]].

Ecrire l'instruction qui concatène liste2 et liste3 et assigne le résultat à la variable res. Que vaut res ?

res = liste2 + liste3 → [[3 , 4] , [1 , 2]] res est une liste constituée de 2 éléments listes.

• **Grosse différence avec la méthode .extend( ) : la concaténation crée un nouvel objet liste !**

Conséquences : - la concaténation est moins rapide que la méthode .extend( ).

- par contre, gros avantage : liste1 et liste2 sont conservées donc réutilisables.

X. TRANCHES DE LISTES OU SLICES.

Créer une tranche (une slice) à partir d'une liste : liste[d : f].

a = liste[d : f]

• **Crée une nouvelle liste** allant de l'élément d'**indice d inclus (si rien : 0)** jusqu'à l'élément d'**indice f exclu (si rien : jusqu'à fin incluse)**.

Et l'affecte à un nom (a ici)

• Ex : Soit la liste a = [5 , 6 , 7 , 8 , 9]

a[1 : 4] → [6 , 7 , 8] a[3 : 5] → [8 , 9]

a[: 4] → [5 , 6 , 7 , 8] a[2 :] → [7 , 8 , 9]

a[: -1] → tout jusqu'au dernier exclu.

a[:] → clone de la liste a. a[: 14] → clone.

Les tranches servent à beaucoup de choses !

• Soit a = list(range (-3 , 7 , 2)) → *[-3 , -1 , 1 , 3 , 5]*

b = a[1 : 3] → b vaut *[-1 , 1]* a[1 : 2] → *[-1]*

c = a[: 2] → c vaut *[-3 , -1]* a[: 1] → *[-3]*

d = a[2 :] → d vaut *[1 , 3 , 5]* a[6 :] → *[ ]*

• Soit une liste h. Ecrire les tranches suivantes :

indice n exclu à p inclus : *h[n+1 : p+1]*

indice du début jusqu'à m inclus : *h[: m+1]*

indice k jusqu'à la fin : *h[k :]*

du début jusqu'à l'avant dernier inclus : *h[: -1]*

un clone de la liste Noms : *Noms[:]*

➢ Exercice : Soient 2 listes : nombres = [0 , 1 , 2] et lettres = ['a' , 'b'].

A l'aide de slices et de la méthode .extend( ), écrire une fonction insere( ) qui insèrera dans une liste1 une liste2 à l'indice k.

Ex : insere(nombres, lettres, 2) → [0 , 1 , 'a' , 'b' , 2].

Remarque : Cette fonction est une variation de la méthode .insert( ) vue p.9.

```
def insere(liste1 , liste2 , k) :
    liste_loc = liste1[ : k]
    liste_loc.extend(liste2)
    liste_loc.extend(liste1[k :  ])
    return liste_loc
```

Ici pas besoin de .copy() car listes 1 et 2 en lecture

XI. LISTES ET CONVERSIONS.

Créer une nouvelle liste en convertissant un objet itérable : Fonction list( ).

a = list(objet_itérable)

• La fonction list( ) agit en entrée sur objet_itérable (chaîne de caractères, range( ), tuple, dictionnaire, etc.) et **renvoie en sortie une nouvelle liste** composée des éléments de l'objet. **A affecter à un nom (a ici) !**

• Ex : list(range(n)) renvoie la liste d'entiers ⟦ 0 , n-1 ⟧. a = list (range(2)) → a vaut [0 , 1]

a = list('lol') → a vaut ['l' , 'o' , 'l'] **Permet d'obtenir la liste des caractères d'une chaîne.**

• La fonction list( ) donne une autre syntaxe pour les compréhensions de liste :

list(fonction(k) for k in liste) ou list(k.method( ) for k in liste)

Créer une seule chaîne de caractères à partir d'une liste de plusieurs caractères : Méthode .join( ).

a = "caractères de jointure".join(liste)

• Attention : que pour des éléments de type str !

• **Crée une nouvelle chaîne** avec tous les caractères de liste joints par les caractères de jointure. **A affecter !**

• Ex : Soit la liste a = ['b' , 'ao' , 'bab'].

mot = '-'.join(a) → mot vaut 'b-ao-bab'

'i'.join(a) → 'biaoibab' 'ut'.join(a) → 'butaoutbab'

• b = ['A' , 'DC'] et c = ['méli' , 'mélo']

groupe = 'C'.join(b) → groupe vaut *'ACDC'.*

a = ''.join(c) → a vaut *'mélimélo'.*

• mots = ['Super' , 'la' , 'spécialité' , 'NSI']

Transformer cette liste en phrase avec '!' final.

```
phrase=''.join(mots) + ' !'
```

Créer une liste de caractères à partir d'une chaîne de caractères : Méthode .split( ).

a = "chaîne".split("caractères de coupure")

• chaîne est coupée à chaque fois qu'apparaît le caractère de coupure dans chaîne.

Puis **création d'une nouvelle liste** à partir des morceaux issus de ce découpage. **A affecter à un nom (a ici) !**

• Ex : a = 'b-ao-bab'.split('-') → a vaut ['b', 'ao' , 'bab']

b = 'faiscifaisça'.split.('fais') → b vaut ['' , 'ci', 'ça']

• Très utile pour découper une phrase en mots : il faut découper selon les *espaces.*

• c = 'clicclac'.split('cl') → c vaut *['' , 'ic' , 'ac'].*

• a = 'Moeve,Mahe,Jules'. Découper a en liste :

```
liste = a.split(',')
```

• b = 'Je veux ton amour.' Ecrire un script qui donne la liste des mots mais sans le point final !

```
liste_mots = b.split('.')
del liste_mots[-1]
liste_mots = liste_mots[0].split(' ')
```

XII. EXERCICES.

① Sans regarder les pages précédentes ! Réécrire la fonction, la méthode en face de l'action qui convient :

Création simple d'une liste	***a = [elt1, elt2, elt3]***	~~a = 'caractère'.join(liste)~~
Génération automatique d'une liste d'éléments tous identiques.	***a = n*[élément]***	~~len(liste)~~
Génération d'une liste d'entiers	***a = list(range(k))***	~~liste.insert(indice, élément)~~
Intention de liste sans filtrage	***a = [expression for k in liste]***	~~liste.append(élément)~~
Compréhension de liste avec filtrage	***a = [expression for k in liste if condition]***	~~liste1.extend(liste2)~~
Longueur d'une liste	***len(liste)***	~~nomliste2 = nomliste1~~
Accéder à un élément d'indice k	***a = liste[k]***	~~fonction list( )~~
Compter le nb d'occurrences d'une valeur.	*a = liste.count(valeur)*	~~liste.reverse( )~~ ~~a = reversed(liste)~~
Savoir si une valeur est présente dans une liste.	***valeur in liste***	~~del liste[k]~~
Connaître la 1ère position d'une valeur.	*a = liste.index(valeur)*	~~liste.sort( )~~ ~~a = sorted(liste)~~
Modifier un élément d'une liste.	***liste[k] = valeur***	~~liste2 = liste1.copy( )~~
Inverser une liste.	*liste.reverse( )* *a = reversed(liste)*	~~a = liste[d : f]~~
Trier une liste.	*liste.sort( )* *a = sorted(liste)*	~~a = 'chaîne'.split('caractère')~~
Vider une liste.	*liste.clear( )*	~~a = n*[élément]~~
Supprimer un élément connaissant sa position.	***del liste[k]***	~~a = [expression for k in liste if condition]~~
Supprimer un élément connaissant sa valeur.	***liste.remove(valeur)***	~~valeur in liste~~
Ajouter un seul élément en fin de liste.	***liste.append(élément)***	~~a = [elt1, elt2, elt3]~~
Insérer un élément.	*liste.insert(indice, élément)*	~~a = list(range(k))~~
Listes liées par leurs noms.	*nomliste2 = nomliste1*	~~a = liste.index(valeur)~~
Clonage indépendant d'une liste.	***liste2 = liste1.copy( )***	~~liste[k] = valeur~~
Etendre une liste par la fin avec les élément d'un autre liste.	***liste1.extend(liste2)***	~~a = list('chaîne-caractères')~~
Tranche de liste (slices)	***a = liste[d : f]***	~~liste.remove(valeur)~~
Convertir un itérable en liste.	***fonction list( )***	~~liste.clear( )~~
Convertir une chaîne en liste.	***a = list('chaîne-caractères')***	~~a = liste.count(valeur)~~
Convertir une liste en chaîne.	***a = 'caractère'.join(liste)***	~~a = [expression for k in liste]~~
Découper une chaîne de caractères en liste de caractères.	*a = 'chaîne'.split('caractère')*	~~a = liste[k]~~

② Sujets 0 et 0bis sur mon site, thèmes Types construits et Langage Programmation.

③ Créer les fonctions classiques suivantes en utilisant uniquement des boucles et affectations (sans utiliser les méthodes ou fonctions spéciales vues auparavant) :

1. Fonction qui renvoie la somme des éléments d'une liste de nombres (sans utiliser la fonction sum( )).

```
def calcul_somme(liste) :
    somme=0
    for nombre in liste :
        somme = somme + nombre
    return somme
```

2. Fonction qui renvoie une liste en ordre inverse (sans utiliser la méthode .reverse( )).

```
def renverse_liste(liste) :
    liste_inversée = [ ]
    for k in range(len(liste)) :
        liste_inversée.append(liste[-1 – k])
    return liste_inversée
```

3. (Fonction qui trie les éléments d'une liste (sans utiliser la méthode .sort( ).)

Voir cours d'algorithmique.

XIII. QUELQUES REMARQUES FINALES SUR LES LISTES.

- Les listes en Python n'ont pas le même nom dans d'autres langages où elles sont appelées tableaux (array en Javascript par exemple).
  A cette adresse, une explication détaillée de la représentation en mémoire des listes qui, en Python, sont à mi-chemin entre la vraie liste (séquence finie d'objets) et le tableau (séquence de données indexées par des entiers) : http://mpechaud.fr/scripts/donnees/listestableaux.html.
- Rassurez-vous, contrairement aux exemples purement pédagogiques qui précèdent, **une liste informatique est souvent constituée d'éléments homogènes ayant le même type entre eux !**
  En effet, dans la réalité, une liste est une collection d'objets liés ayant quelque chose en commun.
- **La maîtrise des listes est incontournable en Algorithmique-Programmation : elles constituent l'un des fondements des Bases de Données.**

Ai-je tout compris ? Listes.	☹	😐	☺	☺☺
Définition, vocabulaire.				
Créer une liste : création simple ; création automatique.				
Créer une liste en compréhension avec ou sans filtrage.				
Tirer de l'information d'une liste : Longueur d'une liste ; accéder à un élément ; compter une valeur ; mot clé in ; 1ère position d'une valeur.				
Modifier une liste : Modifier un élément ; inverser une liste ; trier une liste.				
Transformer une liste : Vider une liste ; supprimer un élément ; ajouter un élément ; insérer un élément.				
Danger de la recopie d'une liste par le nom ; clonage d'une liste.				
Fusionner des listes : étendre une liste par une autre.				
Convertir en liste ; listes ↔ chaînes de caractères.				
Tranches de liste : slices.				
Ecrire une fonction de gestion de base sur les listes sans utiliser les méthodes ou fonctions toutes prêtes.				

Quel est le sujet du prochain cours Python ? *Les listes « constantes » : les tuples.*