# 04-cours_python_listes_boucles

January 17, 2020

## 1 Les listes, les boucles

Une liste est une suite **ordonnée** d'objets (entiers, chaînes, décimaux…) sur lesquels on va pouvoir effectuer toute une série d'opérations (ajout, modification, suppression, extraction…).

### 1.1 Initialisation / construction des listes

[ ]:
```
liste_nombres = [1, 2, 3]
liste_chaines_1 = ["1", "2", "3"]
liste_chaines_2 = ["bonjour", "hello", "python", "ISEN"]
liste_melangee = [29, "a", 22, 3.14, 35, "b", "ISEN Bretagne", 56, 44]
liste_de_listes = [[1, 2], [3, 4]]  # Liste contenant des listes
liste_vide = []  # Liste vide qu'on va pouvoir ensuite construire dynamiquement
```

### 1.2 Extraction d'éléments d'une liste

#### 1.2.1 Indiçage

Pour l'extraction d'éléments contenus dans une liste, on peut utiliser l'indiçage, qui consite à donner la position de l'élément dans la liste. Le premier élément d'une liste commence à **0**.

#### 1.2.2 Exercice

Etant données les listes suivantes :

```
liste_melangee = [29, "a", 22, 3.14, 35, "b", "ISEN Bretagne", 56, 44]
liste_de_listes = [[1, 2], [3, 4]]
```

Indiquer le contenu des éléments suivants :

```
liste_melangee[0]
liste_melangee[3]
liste_melangee[8]
liste_de_listes[0]
liste_de_listes[1]
liste_de_listes[0][0]
liste_de_listes[1][0]
liste_melangee[-1]
liste_melangee[-2]
liste_melangee[9]
```

[1]:
```
liste_melangee = [29, "a", 22, 3.14, 35, "b", "ISEN Bretagne", 56, 44]
liste_de_listes = [[1, 2], [3, 4]]

print(liste_melangee[0])
print(liste_melangee[3])
print(liste_melangee[8])
print(liste_de_listes[0])
print(liste_de_listes[1])
print(liste_de_listes[0][0])
print(liste_de_listes[1][0])
print(liste_melangee[-1])
print(liste_melangee[-2])
print(liste_melangee[9])
```

```
29
3.14
44
[1, 2]
[3, 4]
1
3
44
56
```

```
␣
↪---------------------------------------------------------------------------

IndexError                                Traceback (most recent call␣
↪last)

<ipython-input-1-327f8fd76904> in <module>
 11 print(liste_melangee[-1])
 12 print(liste_melangee[-2])
---> 13 print(liste_melangee[9])


IndexError: list index out of range
```

### 1.3 Extraction d'une sous-liste

Pour extraire des sous-listes, on procède de la façon suivante :

`sous_liste = liste[m:n]`

**Attention !** : la sous-liste contiendra les éléments de la liste originale depuis l'indice $m$ **inclus** jusqu'à l'indice $n$ **exclu** (en d'autres termes, de l'indice $m$ à l'indice *n-1* tous deux inclus).

- Si l'indice $n$ n'est pas précisé, on commence à l'indice $m$.

- Si l'indice $m$ n'est pas précisé, on va jusqu'à l'indice $n$ exclu.

Quel est le contenu des éléments suivants ?

```
liste_melangee = [29, 'a', 22, 3.14, 35, 'b', "ISEN Bretagne", 56, 44]
liste_melangee[0:2]
liste_melangee[6:9]
liste_melangee[3:4]
liste_melangee[3]
liste_melangee[2:]
liste_melangee[:3]
liste_melangee[:-1]
```

[2]:
```
liste_melangee = [29, 'a', 22, 3.14, 35, 'b', "ISEN Bretagne", 56, 44]
print(liste_melangee[0:2])
print(liste_melangee[6:9])
print(liste_melangee[3:4])
print(liste_melangee[3])
print(liste_melangee[2:])
print(liste_melangee[:3])
print(liste_melangee[:-1])
```

```
[29, 'a']
['ISEN Bretagne', 56, 44]
[3.14]
3.14
[22, 3.14, 35, 'b', 'ISEN Bretagne', 56, 44]
[29, 'a', 22]
[29, 'a', 22, 3.14, 35, 'b', 'ISEN Bretagne', 56]
```

## 1.4 La fonction len

La fonction ***len*** permet de récupérer la longueur d'une liste.

[5]:
```
liste = [1, 4, 12, "hello"]
longueur = len(liste)
print(longueur)
```

```
4
```

### 1.4.1 Remarque : quel est résultat affiché par le code suivant ?

```
liste = ["a", 1, [1, 2, 3]]
print(len(liste))
```

### 1.4.2 Exercice : somme avec une boucle while

Soit la liste : `[404, 8, 22, 310, 5]`. A l'aide d'une boucle *while* effectuer la somme des éléments de cette liste et l'afficher à la fin.

[4]:
```
liste = [404, 8, 22, 310, 5]
somme = 0
i = 0
while i < len(liste):
    somme = somme + liste[i]
    i = i + 1
print(somme)
```

749

### 1.4.3 Exercice : calcul de la valeur minimale d'une liste

Soit la liste: `liste = [1, 5, 12, 2, 27]`. En utilisant une boucle *while*, écrire un programme qui trouve la valeur minimale de cette liste et l'affiche à la fin. Avant d'écrire le programme Python, il est recommandé de réfléchir sur papier à un algorithme permettant la résolution du problème.

[6]:
```
liste = [31, 5, 12, 2, 27]
val_min = liste[0]
i = 1
while i < len(liste):
    if liste[i] < val_min:
        val_min = liste[i]
    i += 1
print(val_min)
```

2

### 1.4.4 Exercice : calcul du montant de la liste des courses

On dispose d'une liste de produits et de prix :

```
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]
```

- L'utilisateur peut choisir un produit
- On ajoute le prix correspondant dans un montant total
- On repète cela tant que l'utilisateur ne tape pas "quitter"
- A la fin on lui affiche le montant total

[7]:
```
# Version 1
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]

reponse = ""
montant_courses = 0
while reponse != "quitter":
    reponse = input("No. de votre produit (ou quitter) : ")
    if reponse != "quitter":
        num_produit = int(reponse)
        print(f"Vous avez choisi :{liste_produits[num_produit]}")
```

```
        montant_courses += liste_prix[num_produit]
print(f"Fin des courses, total : {montant_courses} Euros")
```

```
No. de votre produit (ou quitter) : 1
Vous avez choisi :gâteaux apéro
No. de votre produit (ou quitter) : 2
Vous avez choisi :bière
No. de votre produit (ou quitter) : quitter
Fin des courses, total : 7.1 Euros
```

[7]:
```
# Version 2
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]

montant_courses = 0
while True:
    reponse = input("No. de votre produit (ou quitter) : ")
    if reponse != "quitter":
        num_produit = int(reponse)
        print(f"Vous avez choisi :{liste_produits[num_produit]}")
        montant_courses += liste_prix[num_produit]
    else:
        break
print(f"Fin des courses, total : {montant_courses} Euros")
```

```
No. de votre produit (ou quitter) : 1
Vous avez choisi :gâteaux apéro
No. de votre produit (ou quitter) : 2
Vous avez choisi :bière
No. de votre produit (ou quitter) : quitter
Fin des courses, total : 7.1 Euros
```

[6]:
```
# Version 3
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]

montant_courses = 0
reponse = input("No. de votre produit (ou quitter) : ")
while reponse != "quitter":
    num_produit = int(reponse)
    print(f"Vous avez choisi :{liste_produits[num_produit]}")
    montant_courses += liste_prix[num_produit]
    reponse = input("No. de votre produit (ou quitter) : ")

print(f"Fin des courses, total : {montant_courses} Euros")
```

```
No. de votre produit (ou quitter) : 1
Vous avez choisi :gâteaux apéro
```

```
No. de votre produit (ou quitter) : 2
Vous avez choisi :bière
No. de votre produit (ou quitter) : quitter
Fin des courses, total :7.1 Euros
```

### 1.4.5 Exercice : calcul du montant de la liste des courses (version sécurisée)

Comme il n'est pas raisonnable de vendre de l'alcool aux personnes ayant moins de 18 ans, on commencera par demander l'âge au client. Ainsi au moment du choix d'un produit, s'il est alcoolisé et si le client est mineur, on refusera l'achat.

Pour indiquer si un produit contient de l'alcool, on pourra utiliser une 3ème liste :

```
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]
liste_alcools = [False, False, True]
```

**Cas d'usage** : pour vérifier si le programme fonctionne, penser à tester les cas suivants :

- Age < 18 et choix alcool, le choix est refusé et le produit ne doit pas être ajouté au total
- Age >= 18 et choix alcool, le choix est accepté et le produit doit être ajouté au total
- Age < 18 et choix non alcool, le choix est accepté et le produit doit être ajouté au total
- Age >= 18 et choix non alcool, le choix est accepté et le produit doit être ajouté au total

[1]:
```
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]
liste_alcools = [False, False, True]

reponse = ""
montant_courses = 0
age = int(input("Quel est votre âge ? "))
while reponse != "quitter":
    reponse = input("No. de votre produit (ou quitter) : ")
    if reponse != "quitter":
        num_produit = int(reponse)
        print(f"Vous avez choisi :{liste_produits[num_produit]}")
        if age < 18 and liste_alcools[num_produit]:  # == True implicite
            print("Vous ne pouvez acheter de l'alcool !")
        else:
            montant_courses += liste_prix[num_produit]
print(f"Fin des courses, total : {montant_courses} Euros")
```

```
Quel est votre âge ?12
No. de votre produit (ou quitter) : 2
Vous avez choisi :bière
Vous ne pouvez acheter de l'alcool !
No. de votre produit (ou quitter) : 1
Vous avez choisi :gâteaux apéro
No. de votre produit (ou quitter) : quitter
Fin des courses, total : 2.5 Euros
```

### 1.4.6 Exercice : calcul du montant de la liste des courses (avec promo)

On décide d'appliquer des promotions sur certains produits (en fait ici pour simplifier des coefficients). Modifier le programme de telle sorte que les promotions soient appliquées sur le prix. Pour ce faire, on utilisera une 4ème liste :

```
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]
liste_alcools = [False, False, True]
liste_promos = [1, 0.8, 0.7]
```

[2]:
```
liste_produits = ['tomates', 'gâteaux apéro', 'bière']
liste_prix = [3, 2.5, 4.6]
liste_alcools = [False, False, True]
liste_promos = [1, 0.8, 0.7]

reponse = ""
montant_courses = 0
age = int(input("Quel est votre âge ? "))
while reponse != "quitter":
    reponse = input("No. de votre produit (ou quitter) : ")
    if reponse != "quitter":
        num_produit = int(reponse)
        print(f"Vous avez choisi :{liste_produits[num_produit]}")
        if age < num_produit and liste_alcools[num_produit]:  # == True␣
 ↪implicite
            print("Vous ne pouvez acheter de l'alcool !")
        else:
            montant_courses += liste_prix[num_produit] *␣
 ↪liste_promos[num_produit]
print(f"Fin des courses, total : {round(montant_courses, 2)} Euros")
```

```
Quel est votre âge ? 19
No. de votre produit (ou quitter) : 2
Vous avez choisi :bière
No. de votre produit (ou quitter) : quitter
Fin des courses, total : 3.22 Euros
```

### 1.4.7 Exercice : inversion de l'ordre des éléments d'une liste

- Soit la liste : `liste_mots = ["Python", "de", "cours", "le", "J'aime"]`.
- Inverser l'ordre des mots de façon à obtenir : `liste_mots = ["J'aime", "le", "cours", "de", "Python"]`.

[1]:
```
# 1ère solution
liste_mots = ["Python", "de", "cours", "le", "J'aime"]
i = 0
j = len(liste_mots) - 1
while i != j:
```

```
    val_tmp = liste_mots[i]
    liste_mots[i] = liste_mots[j]
    liste_mots[j] = val_tmp
    i += 1
    j -= 1
print(liste_mots)
```

```
["J'aime", 'le', 'cours', 'de', 'Python']
```

[2]:
```
# 2ème solution (sans la variable val_tmp)
liste_mots = ["Python", "de", "cours", "le", "J'aime"]
i = 0
j = len(liste_mots) - 1
while i != j:
    liste_mots[i], liste_mots[j] = liste_mots[j], liste_mots[i]
    i += 1
    j -= 1
print(liste_mots)
```

```
["J'aime", 'le', 'cours', 'de', 'Python']
```

### 1.5 Parcours des éléments d'une liste à l'aide de l'instruction *for*

La boucle ***for*** permet de parcourir facilement une suite d'éléments, en s'affranchissant de la gestion de l'index (la variable *i* dans les exemples précédents), nécessaire dans une boucle *while*.

Voici une version du calcul de la somme des éléments d'une liste avec une boucle *for* :

[2]:
```
liste = [7, 8, 10, 3]
somme = 0
for elem in liste:
    print(f"Elément courant: {elem}")
    somme = somme + elem
print(f"Somme: {somme}")
```

```
Elément courant: 7
Elément courant: 8
Elément courant: 10
Elément courant: 3
Somme: 28
```

- *elem* est le nom d'une variable qui pointe sur l'élément courant de la liste parcourue. Son type est donc fonction de l'élément courant de la liste.
- On retrouve la même struture que pour la boucle *while*, à savoir l'indentation après la ligne `for... :` qui indique que **toutes** les instructions **identées** sous la ligne *for* seront exécutées à **chaque tour de boucle**.

### 1.5.1 Exercice : calcul de la valeur minimale d'une liste (v2)

Soit la liste: `liste = [1, 5, 12, 2, 27]`. Nous avions précédement calculé la valeur minimale en utilisant une boucle `while`. Faire de même mais cette fois en utilisant une boucle `for`..

[4]:
```
liste = [31, 5, 12, 2, 27]
val_min = liste[0]
for elem in liste:
    if elem < val_min:
        val_min = elem
print(f"Valeur minimale: {val_min}")
```

```
Valeur minimale: 2
```

### 1.5.2 Exercice : somme conditionnelle

Ecrire un programme (et un seul) qui effectue la somme des nombres d'une liste, mais en excluant ceux compris entre les élements 's' et 'e' inclus.

Exemples :

- `[1, 2, 2, 's', 99, 99, 'e']` : afficher 5
- `[1, 1, 's', 'e', 2]` : afficher 4
- `[1, 1, 's', 3, 'e', 2]` : afficher 4
- `[1, 1, 's', 'e', 2]` : afficher 4
- `[1, 2, 2]` : afficher 5
- On part de l'hypothèse que s'il y a un 's' il y a forcément un 'e' plus loin.
- Vérifier que votre programme fonctionne pour chacun des éléments ci-dessus (au moins).

[17]:
```
liste = [1, 2, 's', 99, 99, 'e', 8]
somme = 0
faire_somme = True
for elem in liste:
    if elem == 's':
        faire_somme = False
    elif elem == 'e':
        faire_somme = True
    elif faire_somme:
        somme = somme + elem
print(somme)

# Remarque : on pourrait réaliser la même chose avec 3 boucles while à la suite
```

```
11
```

## 1.6 Le mot clé *in*

Outre son utilisation dans une boucle *for* comme vu précédemment, on peut utiliser l'instruction ***in*** pour savoir si un élément appartient à une liste.

Effectuer les tests suivants :

```
1 in [3, 1, 4]
reponse = "oui"
reponse in ["oui", "non"]
```

Quelle est le type de la valeur renvoyée par l'expression *in* ?

### 1.6.1 Exercice : recherche d'une valeur dans une liste

Soit la liste suivante : `[12, 4, 5, 7]`. Demander à l'utilisateur de saisir un nombre. En fonction de sa réponse, lui indiquer si ce nombre appartient à la liste.

[8]:
```
liste = [12, 4, 5, 7]
nombre = int(input("Votre nombre ? "))
if nombre in liste:
    print("Il est dans la liste")
else:
    print("Il ne se trouve pas dans la liste")
```

```
Votre nombre ? 4
Il est dans la liste
```

## 1.7 La fonction `range()` et la boucle `for`

Pour boucler $n$ fois avec la boucle `for`, on peut utiliser la fonction `range()`.

[2]:
```
print("Exemple 1")
for i in range(0, 5):  # Boucle 5 fois à partir de 0 et i commence à 0
    print(i)
# Remarque 1 : ceci est équivalent : for i in in range(5):
# Remarque 2 : on peut effectuer la même chose avec une boucle while
#              mais c'est plus compliqué...

print("Exemple 2")
for i in range(2, 5):  # Boucle 5 fois à partir de 0 et i commence à 2
    print(i)

print("Exemple 3")
for i in range(2, 5, 2):  # Boucle 5 fois à partir de 0 et i commence à 2, par␣
 ↪pas de 2
    print(i)
```

```
Exemple 1
0
1
```

```
2
3
4
Exemple 2
2
3
4
Exemple 3
2
4
```

### 1.7.1 Exercice : série mathématique

Ecrire un programme permettant le calcul de la série suivante :

$$S_n = \sum_{i=0}^{n} \left(\frac{1}{2}\right)^i$$

On fixera arbitrairement $n$ en début de programme.

[1]:
```
n = 25
somme = 0
for i in range(n+1):
    somme = somme + 0.5**i
print(somme)
```

```
1.9999999701976776
```

## 1.8 Exercice : la suite de Fibonacci

Son inventeur est Léonard de Pise (1175 − v.1250), aussi connu sous le nom de Leonardo Fibonacci, qui a rapporté d'Orient la notation numérique indo-arabe et a écrit et traduit des livres influents de mathématiques.

Introduite comme problème récréatif dans son fameux ouvrage Liber Abaci, la suite de Fibonacci peut être considérée comme le tout premier modèle mathématique en dynamique des populations ! En effet, elle y décrit la croissance d'une population de lapins sous des hypothèses très simplifiées, à savoir : chaque couple de lapins, dès son troisième mois d'existence, engendre chaque mois un nouveau couple de lapins, et ce indéfiniment (source).

$$U_n = U_{n-1} + U_{n-2} \text{ avec } n >= 2$$

Ecrire un programme qui demande à l'utilisateur la valeur n et qui affiche le résultat de la suite.

[4]:
```
# Une première version...
n = int(input("Nombre n : "))
if n < 2:
    print("Merci de saisir un nombre supérieur ou égal à deux...")
```

```
else:
    u_n_1 = 1
    u_n_2 = 0
    u_n = u_n_1 + u_n_2
    for i in range(2, n):
        u_n_2 = u_n_1
        u_n_1 = u_n
        u_n = u_n_1 + u_n_2

print(u_n)
```

```
Nombre n : 4
3
```

[ ]:
```
# Une deuxième version
n = int(input("Nombre n : "))
if n < 2:
    print("Merci de saisir un nombre supérieur ou égal à deux...")
else:
    u1,u2 = 1, 1
    for i in range(1, n):
        u1,u2 = u2, u1+u2
    print(u1)
```

### 1.8.1 Exercice : affichage des nombres impairs de 1 à 10

1) Avec une boucle `for` et la fonction `range()`

2) Faire la même chose avec une boucle `while`

[8]:
```
for i in range(1, 11, 2):
    print(i)
```

```
1
3
5
7
9
```

[7]:
```
i = 1
while i < 11:
    if i%2 != 0:
        print(i)
    i += 1
```

```
1
3
5
```

```
7
9
```

### 1.8.2 Exercice : affichage d'un triangle de '#'

Avec une boucle `for` et la fonction `range()`, afficher le "graphique" suivant :

```
#
##
###
####
#####
######
```

Indication : pour répéter /n/ fois le même caractère on peut utiliser l'opérateur `*` sur un caractère.

Exemple : "s" * 4 -> "ssss"

[10]:
```
for i in range(1, 7):
    print("#" * i)
```

```
#
##
###
####
#####
######
```

## 1.9 Ajout d'éléments à une liste

L'ajout d'un élément s'effectue avec la méthode (~fonction) ***append***.

[10]:
```
liste = [29, "a", 22, 3.14, 35]
print(liste)
liste.append(123456789)
print(liste)
```

```
[29, 'a', 22, 3.14, 35]
[29, 'a', 22, 3.14, 35, 123456789]
```

- Notons que l'utilisation de la méthode *append* est différente de *len*.
  - `len(liste)`
  - `liste.append(...)`
- Après appel à la méthode *append*, il y a directement **modification de la liste originale**.
- L'intérêt principal de la méthode *append* est la construction dynamique de liste. Ecrire le programme suivant et regarder ce qui se passe :

```
liste = []
i = 0
while i < 5:
  liste.append(i)
```

```
    print(liste)
    i += 1
```

### 1.9.1 Exercice : concaténation de listes

Soient les listes suivantes :

```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6, 7]
```

En utilisant l'instruction *for* et la méthode *append*, ajouter dans une troisième liste les éléments de *liste1* et *liste2*.

[18]:
```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6, 7]
liste3 = []
for elem in liste1:
    liste3.append(elem)

for elem in liste2:
    liste3.append(elem)

print(liste3)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

### 1.9.2 Exercice : liste construite par l'utilisateur

- Demander à l'utilisateur de saisir une valeur (peu importe le type) et l'ajouter dans une liste. Continuer tant qu'il n'a pas saisi "-1".
- Lorsqu'il a saisi "-1" afficher la liste ainsi que le nombre d'éléments.

[1]:
```
valeur = ""
liste = []
while valeur != "-1":  # Pourquoi y a t-il des guillemets ici ?
    valeur = input("Valeur ? (-1 pour terminer) : ")
    if valeur != "-1":
        liste.append(valeur)
print(liste)
print(len(liste))
```

```
Valeur ? (-1 pour terminer) : 12
Valeur ? (-1 pour terminer) : 3
Valeur ? (-1 pour terminer) : -1
['12', '3']
2
```

[ ]:
```
valeur = ""
liste = []
while valeur != "-1":  # Pourquoi y a t-il des guillemets ici ?
```

```
    valeur = input("Valeur ? (-1 pour terminer) : ")
    liste.append(valeur)

liste = liste[:-1]  # On supprime le dernier élément de la liste
print(liste)
print(len(liste))
```

### 1.9.3 Exercice : extraction des nombres impairs

Pour une liste donnée de nombres, extraire ceux qui sont impairs en les stockant dans une liste.

- Exemple: `liste = [1, 3, 4, 6, 7, 11, 12]` -> `[1, 3, 7, 11]`
- Indications : on utilisera une seconde liste qui contiendra uniquement les nombres impairs

[24]:
```
liste = [1, 3, 4, 6, 7, 11, 12]
liste_impairs = []
for nombre in liste:
    if nombre % 2 != 0:
        liste_impairs.append(nombre)

print(liste_impairs)
```

```
[1, 3, 7, 11]
```

## 1.10 Exercice : supprestion des doublons

- Soit la liste : `[1, 2, 2, 3, 4, 2, 7, 1, 4, 5, 7, 6]`
- Créer une nouvelle liste en supprimant les éléments en double

[1]:
```
liste_init = [1, 2, 2, 3, 4, 2, 7, 1, 4, 5, 7, 6]
liste_res = []
for elem in liste_init:
    if elem not in liste_res:
        liste_res.append(elem)
print(liste_res)
```

```
[1, 2, 3, 4, 7, 5, 6]
```

## 1.11 Concaténation / extension de listes

La concaténation peut s'effectuer avec l'opérateur $+$.

[16]:
```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6, 7]
liste3 = liste1 + liste2
print(liste3)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

Si on souhaite étendre une liste existante, on peut utiliser la méthode ***extend***.

[20]:
```
liste1 = [1, 2, 3]
liste2 = [4, 5, 6, 7]
liste1.extend(liste2)
print(liste2)
print(liste1)
```

```
[4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
```

Que se passe t-il si on utilise la à la place d'*extend* on effectue : `liste1.append(liste2)` ?

## 1.12 Modification d'éléments d'une liste

Les éléments individuels d'une liste peuvent être modifiés (le type *list* est dit *mutable*). Tester l'exécution des lignes suivantes :

```
liste = [29, "a", 22, 3.14, 35, "b", "ISEN Bretagne", 56, 44]
liste[2] += 5
liste[6] = "Yncrea"
print(liste)
```

Que se passe t-il si on exécute : `liste[1] += 2` ?

### 1.12.1 Exercice : remplacement d'éléments (remplacement de mots)

Soit la liste : `["il", "fait", "beau", "ce", "soir"]`.

En utilisant une boucle *while*, remplacer les mots "beau" par "chaud" et "ce" par "le".

[24]:
```
liste = ["il", "fait", "beau", "ce", "soir"]
i = 0
lg_liste = len(liste)
while i < lg_liste:
    if liste[i] == "beau":
        liste[i] = "chaud"
    elif liste[i] == "ce":
        liste[i] = "le"
    i += 1
print(liste)
```

```
['il', 'fait', 'chaud', 'le', 'soir']
```

### 1.12.2 Exercice : remplacement d'éléments (remplacement de mots - v2)

Faire la même chose que l'exercice précédent, sachant que cette fois on a deux listes :

- `liste_originale = ["il", "fait", "beau", "ce", "soir"]`
- `liste_remplacements = [None, None, "chaud", "le", None]`

`liste_originale` contiendra à la fin du programme : `['il', 'fait', 'chaud', 'le', 'soir']`

[2]:
```
liste_originale = ["il", "fait", "beau", "ce", "soir"]
liste_remplacements = [None, None, "chaud", "le", None]
i = 0
lg_liste = len(liste_originale)
while i < lg_liste:
    if liste_remplacements[i] != None:
        liste_originale[i] = liste_remplacements[i]
    i+=1

print(liste_originale)
```

```
['il', 'fait', 'chaud', 'le', 'soir']
```

Python "avancé" : plutôt que d'utiliser *while* et un indice (i), on peut utiliser la fonction ***enumerate***. Ainsi le programme suivant est rigoureusement équivalent :

[ ]:
```
liste = ["il", "fait", "beau", "ce", "soir"]
for i, element in enumerate(liste):
    if element == "beau":  # ou: liste[i] == "beau"
        liste[i] = "chaud"
    elif element == "ce":
        liste[i] = "le"
print(liste)
```

A chaque tour de boucle, la variable *i* contient l'indice courant (en commençant par 0) et la variable *element* l'élément courant de la liste.

### 1.12.3 Exercice : modification des éléments d'une liste (valeurs au carré)

- Soit la liste suivante : `liste = [4, 5, 6]`
- Modifier cette liste de telle façon à remplacer chaque nombre par sa valeur au carré
- On devra donc obtenir : `[16, 25, 36]`

[28]:
```
liste = [4, 5, 6]
i = 0
while i < len(liste):
    liste[i] = liste[i]**2
    i += 1
print(liste)
```

```
[16, 25, 36]
```

## 1.13 Suppression d'éléments dans une liste

On peut supprimer un élément de la liste en utilisant la méthode ***remove(nom_element)*** ou ***pop(indice_element)***. Notons que de façon similaire à la méthode *append* vu précédemment, après appel à `remove` / `pop`, **la liste originale est modifiée**.

[26]:
```
liste = [29, 'a', 22, 3.14, 35, 'b', "ISEN Bretagne", 56, 44]
print(liste)
liste.remove("ISEN Bretagne")
print(liste)
```

```
[29, 'a', 22, 3.14, 35, 'b', 'ISEN Bretagne', 56, 44]
[29, 'a', 22, 3.14, 35, 'b', 56, 44]
```

[1]:
```
liste = [29, 'a', 22, 3.14, 35, 'b', "ISEN Bretagne", 56, 44]
print(liste)
liste.pop(6)
print(liste)
```

```
[29, 'a', 22, 3.14, 35, 'b', 'ISEN Bretagne', 56, 44]
[29, 'a', 22, 3.14, 35, 'b', 56, 44]
```

### 1.14 La méthode *split()*

La méthode ***split*** est très pratique, elle permet (par exemple) de créer une liste de mots à partir d'une chaîne de caractères. La transformation en liste permettra ensuite de remplacer / ajouter / supprimer facilement des éléments. Voici un exemple :

[27]:
```
phrase = "toto est beau"
# Par défaut le séparateur est l'espace,
# mais on peut préciser un autre caractère
# à la méthode split
mots_liste = phrase.split()
print(mots_liste)
print(type(mots_liste))

chaine = "fait,beau,fait,pas,beau"
mots_liste = chaine.split(",")
print(mots_liste)
```

```
['toto', 'est', 'beau']
<class 'list'>
['fait', 'beau', 'fait', 'pas', 'beau']
```

### 1.15 La méthode *join()*

C'est la "réciproque" de la méthode *split*. Elle permet, à partir d'une liste de chaînes de reconstituer une chaîne de caractères.

Exemple :

[2]:
```
mots_liste = ['toto', 'est', 'beau']
phrase = " ".join(mots_liste)  # Noter l'espace entre les guillemets
print(phrase)
```

```
toto est beau
```

### 1.16 Mini-TP : vérifier la validité d'une date

- Objectif : demander à l'utilisateur de saisir un jour, un mois et une année. Lui indiquer si la date est valide ou non. Tant que la date est invalide recommencer.
- Indications :
  - Il sera utile de savoir si une année est bissextile ou non : dans ce cas il y a un 29 février. Une année est bissextile si l'année est divisible par 4 et que cette année est divisible par 400 mais pas par 100.
  - Avant d'écrire le programme en Python, écrire la structure du programme avec des pseudo-instructions.

[14]:
```
while True:
    jour = int(input("Jour : "))
    mois = int(input("Mois : "))
    annee = int(input("Année : "))
    if jour > 31 or jour < 1 or annee < 0 or mois < 1 or mois > 12:
        print("Date invalide")
    elif mois in [1, 3, 5, 7, 8, 10, 12]:
        break
    else:  # Mois à 30 jours ou février
        if mois == 2:
            if annee%4 == 0 and (annee%400 == 0 or not annee%100 == 0):  #␣
↪Année bissextile
                if jour > 29:
                    print("Le mois de février d'une année bissextile ne peut␣
↪avoir plus de 29 jours")
                else:
                    break
            else:  # Année non bissextile
                if jour > 28:
                    print("Le mois de février d'une année non bissextile ne␣
↪peut avoir plus de 28 jours")
                else:
                    break
        else:
            if jour > 30:
                print("Erreur, pour ce mois il ne peut y avoir plus de 30␣
↪jours")
            else:
                break

print("Date valide !")
```

```
Jour : 12
Mois : 4
Année : -1
```

```
Date invalide
Jour : 31
Mois : 4
Année : 2018
Erreur, pour ce mois il ne peut y avoir plus de 30 jours
Jour : 29
Mois : 2
Année : 2017
Le mois de février d'une année non bissextile ne peut avoir plus de 28 jours
Jour : 29
Mois : 2
Année : 2016
Date valide !
```

### 1.17 Mini-TP : manipulation de matrices

Soit la matrice :

```
1 4 2
5 4 4
8 2 1
```

1) Ajouter +1 pour chaque élément
2) Mettre 1 sur les éléments de la diagonale

- Indications :
  - Pour représenter une matrice en Python, on peut utiliser des listes de listes : `matrice = [[1, 4, 2], [5, 4, 4], [8, 2, 1]]`
  - Pour vérifier que la matrice a bien le résultat attendu, on peut utiliser l'instruction `assert` : `assert matrice == [[2, 5, 3], [6, 5, 5], [9, 3, 2]]` (pour la question 1)

[16]:
```python
# Question 1
matrice = [[1, 4, 2], [5, 4, 4], [8, 2, 1]]
i = 0
j = 0
for ligne in matrice:
    for elem in ligne:
        matrice[i][j] += 1
        j += 1
    i += 1
    j = 0

assert matrice == [[2, 5, 3], [6, 5, 5], [9, 3, 2]]

# Question 2
i = 0
j = 0
for ligne in matrice:
```

```
    for elem in ligne:
        if i == j:
            matrice[i][j] = 1
        j += 1
    i += 1
    j = 0

assert matrice == [[1, 5, 3], [6, 1, 5], [9, 3, 1]]
```

### 1.18 Mini-TP : le tri à bulles

Le principe du tri bulle est de comparer deux à deux les éléments consécutifs d'une liste (*e1* et *e2* et d'effecteur une permutation si *e1* > *e2*. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.

Soit la liste = `[4, 2, 8, 1, 7, 5, 3, 6]`

```
[2, 4, 8, 1, 7, 5, 3, 6]
[2, 4, 1, 8, 7, 5, 3, 6]
[2, 4, 1, 7, 8, 5, 3, 6]
[2, 4, 1, 7, 5, 8, 3, 6]
[2, 4, 1, 7, 5, 3, 8, 6]
[2, 4, 1, 7, 5, 3, 6, 8]
Fin traitement liste
[2, 1, 4, 7, 5, 3, 6, 8]
[2, 1, 4, 5, 7, 3, 6, 8]
[2, 1, 4, 5, 3, 7, 6, 8]
[2, 1, 4, 5, 3, 6, 7, 8]
Fin traitement liste
[1, 2, 4, 5, 3, 6, 7, 8]
[1, 2, 4, 3, 5, 6, 7, 8]
Fin traitement liste
[1, 2, 3, 4, 5, 6, 7, 8]
Fin traitement liste
```

On voit qu'à l'étape 13, la liste est triée, c'est donc terminé !

1) Réfléchir à un algorithme permettant de réaliser ces étapes, à savoir la permutation des deux éléments *e1* et *e2*, lorsque *e1* est supérieur à *e2*. On veillera à ce qu'il soit générique, c'est à dire qu'il puisse s'appliquer à n'importe quelle liste.
2) Ecrire en Python cet algorithme.
3) Pourquoi cet algorithme (un classique de l'apprentissage de la programmation) s'appelle le tri à bulles ?

[1]:
```
liste = [4, 2, 8, 1, 7, 5, 3, 6]
permutation = True
i = 0
while permutation:
    permutation = False
```

```
    for elem in liste:
        if i < len(liste)-1:
            if liste[i] > liste[i+1]:
                liste[i+1], liste[i] = liste[i], liste[i+1]  # Echange de␣
↪valeurs
                permutation = True
                print(liste)  # Juste pour contrôler ce qui se passe
        i += 1
    print("Fin traitement liste")
    i = 0
```

```
[2, 4, 8, 1, 7, 5, 3, 6]
[2, 4, 1, 8, 7, 5, 3, 6]
[2, 4, 1, 7, 8, 5, 3, 6]
[2, 4, 1, 7, 5, 8, 3, 6]
[2, 4, 1, 7, 5, 3, 8, 6]
[2, 4, 1, 7, 5, 3, 6, 8]
Fin traitement liste
[2, 1, 4, 7, 5, 3, 6, 8]
[2, 1, 4, 5, 7, 3, 6, 8]
[2, 1, 4, 5, 3, 7, 6, 8]
[2, 1, 4, 5, 3, 6, 7, 8]
Fin traitement liste
[1, 2, 4, 5, 3, 6, 7, 8]
[1, 2, 4, 3, 5, 6, 7, 8]
Fin traitement liste
[1, 2, 3, 4, 5, 6, 7, 8]
Fin traitement liste
Fin traitement liste
```

[1]:
```
liste = [4, 2, 8, 1, 7, 5, 3, 6]
permutation = True
i = 0
while permutation:
    permutation = False
    for elem in liste:
        if i < len(liste)-1:
            if liste[i] > liste[i+1]:
                liste[i+1], liste[i] = liste[i], liste[i+1]  # Echange de␣
↪valeurs
                permutation = True
                print(liste)  # Juste pour contrôler ce qui se passe
        i += 1
    print("Fin traitement liste")
    i = 0
```

```
[2, 4, 8, 1, 7, 5, 3, 6]
```

```
[2, 4, 1, 8, 7, 5, 3, 6]
[2, 4, 1, 7, 8, 5, 3, 6]
[2, 4, 1, 7, 5, 8, 3, 6]
[2, 4, 1, 7, 5, 3, 8, 6]
[2, 4, 1, 7, 5, 3, 6, 8]
Fin traitement liste
[2, 1, 4, 7, 5, 3, 6, 8]
[2, 1, 4, 5, 7, 3, 6, 8]
[2, 1, 4, 5, 3, 7, 6, 8]
[2, 1, 4, 5, 3, 6, 7, 8]
Fin traitement liste
[1, 2, 4, 5, 3, 6, 7, 8]
[1, 2, 4, 3, 5, 6, 7, 8]
Fin traitement liste
[1, 2, 3, 4, 5, 6, 7, 8]
Fin traitement liste
Fin traitement liste
```

[3]:
```python
# Une autre version...

liste = [4, 2, 8, 1, 7, 5, 3, 6]
lg_liste = len(liste)
# On parcours toute la liste
for i in range(lg_liste):
    for j in range(0, lg_liste-i-1):
        # échanger si l'élément trouvé est plus grand que le suivant
        if liste[j] > liste[j+1] :
            liste[j], liste[j+1] = liste[j+1], liste[j]
            print(liste)
    print("fin boucle 2")
```

```
[2, 4, 8, 1, 7, 5, 3, 6]
[2, 4, 1, 8, 7, 5, 3, 6]
[2, 4, 1, 7, 8, 5, 3, 6]
[2, 4, 1, 7, 5, 8, 3, 6]
[2, 4, 1, 7, 5, 3, 8, 6]
[2, 4, 1, 7, 5, 3, 6, 8]
fin boucle 2
[2, 1, 4, 7, 5, 3, 6, 8]
[2, 1, 4, 5, 7, 3, 6, 8]
[2, 1, 4, 5, 3, 7, 6, 8]
[2, 1, 4, 5, 3, 6, 7, 8]
fin boucle 2
[1, 2, 4, 5, 3, 6, 7, 8]
[1, 2, 4, 3, 5, 6, 7, 8]
fin boucle 2
[1, 2, 3, 4, 5, 6, 7, 8]
fin boucle 2
```

```
fin boucle 2
fin boucle 2
fin boucle 2
fin boucle 2
```

## 1.19 Mini-TP le jeu du pendu

### 1.19.1 Indications générales

- On définit une liste de mots à trouver.
- On fixe un nombre de tentatives maximum.
- On choisit un mot au hasard dans la liste de mots (voir la fonction `random.choice`).
- On demande à l'utilisateur de saisir une lettre.
- Si la lettre a déjà été saisie, on lui indique.
- Le nombre de tentatives est incrémenté si la lettre n'appartient pas au mot (et si la lettre n'a pas été déjà proposée).
- A chaque tour on affiche sa réponse en cours (lettres devinées), le nombre de tentatives et les lettres proposées.
- A la fin on affiche à l'utilisateur s'il a gagné ou s'il a perdu (quand il a dépassé le nombre de tentatives).

### 1.19.2 Indications avancées

- Algorithme général.
- Stockage de la réponse en cours : dans une liste.
- Utiliser la fonction `join` pour comparer la réponse courante et le mot à trouver.

[14]:
```python
import random

MOTS = ("python", "java", "isen", "brest")
NB_TENTATIVES_MAX = 5

# Initialisations
nb_tentatives = 0
gagne = False
lettres_proposees = []
mot_a_trouver = random.choice(MOTS)
print(mot_a_trouver)
# Initialisation de la réponse
reponse_crte = ['-'] * len(mot_a_trouver)

while not gagne and nb_tentatives < NB_TENTATIVES_MAX:
    print(f"{reponse_crte} / Tentatives = {nb_tentatives} / Lettres proposées :␣
 ↪{lettres_proposees}")
    lettre = input("Lettre ? ")
    lettre = lettre.lower()
    if len(lettre) > 1:
        print("Merci d'entrer une lettre")
    elif lettre in lettres_proposees:
```

```
        print("Vous avez déjà proposé cette lettre")
    else:
        lettres_proposees.append(lettre)
        if lettre in mot_a_trouver:
            for i, car in enumerate(mot_a_trouver):
                if car == lettre:
                    reponse_crte[i] = lettre
            gagne = "".join(reponse_crte) == mot_a_trouver
        else:
            nb_tentatives += 1

if gagne:
    print("Bravo, vous avez gagné !")
else:
    print ("Vous avez perdu, la réponse était : ", mot_a_trouver)
```

```
python
['-', '-', '-', '-', '-', '-'] / Tentatives = 0 / Lettres proposées : []
Lettre ? p
['p', '-', '-', '-', '-', '-'] / Tentatives = 0 / Lettres proposées : ['p']
Lettre ? k
['p', '-', '-', '-', '-', '-'] / Tentatives = 1 / Lettres proposées : ['p', 'k']
Lettre ? t
['p', '-', 't', '-', '-', '-'] / Tentatives = 1 / Lettres proposées : ['p', 'k',
't']
Lettre ? o
['p', '-', 't', '-', 'o', '-'] / Tentatives = 1 / Lettres proposées : ['p', 'k',
't', 'o']
Lettre ? h
['p', '-', 't', 'h', 'o', '-'] / Tentatives = 1 / Lettres proposées : ['p', 'k',
't', 'o', 'h']
Lettre ? b
['p', '-', 't', 'h', 'o', '-'] / Tentatives = 2 / Lettres proposées : ['p', 'k',
't', 'o', 'h', 'b']
Lettre ? q
['p', '-', 't', 'h', 'o', '-'] / Tentatives = 3 / Lettres proposées : ['p', 'k',
't', 'o', 'h', 'b', 'q']
Lettre ? z
['p', '-', 't', 'h', 'o', '-'] / Tentatives = 4 / Lettres proposées : ['p', 'k',
't', 'o', 'h', 'b', 'q', 'z']
Lettre ? y
['p', 'y', 't', 'h', 'o', '-'] / Tentatives = 4 / Lettres proposées : ['p', 'k',
't', 'o', 'h', 'b', 'q', 'z', 'y']
Lettre ? y
Vous avez déjà proposé cette lettre
['p', 'y', 't', 'h', 'o', '-'] / Tentatives = 4 / Lettres proposées : ['p', 'k',
't', 'o', 'h', 'b', 'q', 'z', 'y']
```

```
Lettre ? v
Vous avez perdu, la réponse était :  python
```

### 1.20 Mini-TP matplotlib

La bibliothèque `matplotlib` permet de tracer des graphique (et bien plus !). Voici un exemple :

```
import matplotlib.pyplot as plt

fig = plt.figure ()

# Initialisation des listes
liste_x = [0 , 1.5 , 2 , 3]
liste_y = [ -5 , 18 , 5.4 , 6]

# Affichage du graphique
plt.plot (liste_x, liste_y)
```

[19]:
```
%matplotlib inline
import matplotlib . pyplot as plt

fig = plt.figure ()

# Initialisation des listes
liste_x = [0 , 1.5 , 2 , 3]
liste_y = [ -5 , 18 , 5.4 , 6]

# Affichage du graphique
plt.plot (liste_x, liste_y)
```

[19]:
```
[<matplotlib.lines.Line2D at 0x7f2a67c74a30>]
```

### 1.20.1 Exercice : graphique des températures

- Pour chaque mois de l'année, demander à l'utilisateur de saisir une température.
- Afficher ensuite le graphique correspondant.

[ ]:
```
%matplotlib inline
import matplotlib . pyplot as plt

fig = plt.figure ()

liste_mois = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
# ou : liste_mois = [str(mois) for mois in range(1, 13)]

liste_temperatures = []

for mois in liste_mois:
    #print("Mois : ", mois, end=' ')
    temperature = float(input(f"Mois {mois} / Température : "))
    liste_temperatures.append(temperature)

# Graphique
plt.plot (liste_mois, liste_temperatures)
```