TRAVAUX PRATIQUES n° 6
Introduction aux listes

I. Définition de listes

Une liste est un type de donnée de Python (type `list`) permettant de stocker plusieurs données en une seule variable.

Il existe plusieurs manières de définir une liste :

⋆ **Définition en extension** :

```
nom_liste = [e_1, e_2, ..., e_n]
```

⋆ **Définition d'une liste contenant plusieurs fois la même valeur** :

```
nom_liste = nombre * [valeur]
```

⋆ **Définition en compréhension** :

- `nom_liste = [f(k) for k in range(a, b)]`
- `nom_liste = [f(k) for k in autre_liste]`

Voici quelques exemples que vous pouvez tester dans le shell pour mieux comprendre chaque syntaxe. Pour chaque liste, on doit recopier la définition ci-dessous **et** retaper le nom de la liste dans le shell. **N'hésitez pas à modifier ces exemples pour voir les effets des modifications !**

```
>>> liste_1 = [1, 3, 5]
>>> liste_2 = [5, "Bonjour !", 4.56, True]
>>> liste_3 = [k**2 + 2*k for k in range(4, 10)]
>>> liste_4 = [i**3 for i in range(100)]
>>> liste_5 = [2*e-4*e**2 for e in liste_3]
>>> liste_6 = 33 * [2.3]
```

Exercice 1 : Pour chacune des questions suivantes, écrire la définition de la liste dans l'éditeur, exécuter le code, puis vérifier le contenu de la liste en tapant son nom dans le shell.

1) Définir une liste vide nommé `vide`.
2) Définir la liste `premiers` des 10 premiers nombres premiers (les écrire un par un).
3) Définir la liste `zeros` contenant 100 fois le nombre 0.
4) Définir la liste `gauss` des nombres de la forme $\frac{n(n+1)}{2}$ où $n \in [\![-10, 10]\!]$ (il doit donc y avoir 21 éléments). Faire en sorte que les éléments de cette liste soient bien des entiers (`int`) et non des flottants (`float`).
5) Définir la liste `carres` des carrés des nombres de la liste `gauss`.

II. Accéder aux éléments et modifier un élément

Une fois qu'une liste est définie, on doit pouvoir récupérer la valeur d'un élément en particulier. Pour cela, il faut donner à Python deux informations :

- ⋆ le nom de la liste dans laquelle se trouve l'élément
- ⋆ la position de l'élément dans cette liste : on appelle cette position **l'indice de l'élément dans la liste**

Ainsi la syntaxe pour accéder à un élément particulier d'une liste est la suivante :

```
nom_liste[indice]
```

Pour une liste ayant n éléments, l'indice varie de 0 (pour le premier élément) à $n-1$ (pour le dernier élément).

Par exemple, pour la liste `L = [5, 3.14, "abc", -3, 0]`, on a

Elément	`5`	`3.14`	`"abc"`	`-3`	`0`
Indice	`0`	`1`	`2`	`3`	`4`

Exercice 2 : Deviner ce qui devrait apparaitre après chacune des ces instruction puis vérifier en exécutant l'instruction dans le shell.

```
>>> L = [5, 3.14, "abc", -3, 0]
>>> L[0]
>>> L[1]
>>> L[2]
>>> L[3]
>>> L[4]
>>> L[5]
```

Que signifie l'erreur `IndexError` et pourquoi apparait-elle lors de l'exécution de la dernière instruction ?

Variante

Il est aussi possible d'accéder à un élément d'une liste par un indice négatif : dans ce cas, on compte en partant de la fin. Pour une liste à n éléments, le dernier élément est l'élément d'indice -1, l'avant-dernier est l'élément d'indice -2 et le premier élément est l'élément d'indice $-n$.
Par exemple, pour la liste `L = [5, 3.14, "abc", -3, 0]`, on a

Elément	`5`	`3.14`	`"abc"`	`-3`	`0`
Indice	`0`	`1`	`2`	`3`	`4`
Indice négatif	`-5`	`-4`	`-3`	`-2`	`-1`

Exercice 3 : Deviner ce qui devrait apparaitre après chacune des ces instruction puis vérifier en exécutant l'instruction dans le shell.

```
>>> L[-1]
>>> L[-2]
>>> L[-3]
>>> L[-4]
>>> L[-5]
>>> L[-6]
```

Pourquoi observe-t-on encore une erreur `IndexError` ?

Pour modifier un élément d'une liste, on utilise la syntaxe :

```
nom_liste[indice] = nouvelle_valeur
```

Exercice 4 : Deviner ce qui devrait apparaitre après chacune des ces instruction puis vérifier en exécutant l'instruction dans le shell.

```
>>> L[2] = "def"
>>> L
>>> L[-1] = L[-1] + 2
>>> L
>>> L[1] = L[0] + L[-1]
>>> L[-2] = L[2]
>>> L
>>> L[1] = L[0] + L[-1]
>>> L
```

III. Fonctions sur les listes

Il y a 4 fonctions à connaître sur les listes et la première est la plus importante :

- ⋆ Fonction `len` : calcule la longueur (la taille) d'une liste, c'est-à-dire le nombre d'éléments d'une liste.
- ⋆ Fonction `sum` : calcule la somme des éléments d'une liste numérique.
- ⋆ Fonction `max` : calcule le maximum des éléments d'une liste numérique.
- ⋆ Fonction `min` : calcule le minimum des éléments d'une liste numérique.

Exercice 5 : Deviner ce qui devrait apparaitre après chacune des ces instruction puis vérifier en exécutant l'instruction dans le shell.

```
>>> L = [4, 6, -3, 13, 2, 4, 6, 1, -3]
>>> len(L)
>>> sum(L)
>>> max(L)
>>> min(L)
```

Exercice 6 :

1) Calculer, en utilisant la définition en compréhension des listes, la somme $\sum_{k=0}^{100} k^4$.
2) Ecrire une fonction `somme(n)` qui calcule et renvoie la valeur de la somme $\sum_{k=2}^{n} \frac{1}{k-1}$.

IV. Parcourir une liste

Parcourir une liste consiste, à l'aide d'une boucle, à examiner chaque élément d'une liste. Il existe deux types de parcours différents :

- ⋆ Le parcours des éléments :

```
for e in nom_liste:
    Instructions
```

Dans cette boucle, la variable `e` prend les valeurs successives des **éléments** de la liste.

- ⋆ Le parcours des indices :

```
for i in range(len(nom_liste)):
    Instructions
```

Dans cette boucle, la variable `i` prend les valeurs successives des **indices** des éléments de la liste. Si l'on souhaite utiliser la valeur de l'élément correspondant, on utilisera alors la syntaxe `nom_liste[i]`.

Exercice 7 : Deviner ce qui devrait apparaitre après chacune des ces instruction puis vérifier en exécutant l'instruction dans l'éditeur.

```
L = [4, 6, -3, 13, 2, 4, 6, 1, -3]

for e in L:
    print(e)

for i in range(len(L)):
    print(i)

for i in range(len(L)):
    print(i, L[i])
```

V. Réécriture de fonctions

Dans cette partie, on va s'entrainer à travailler avec les listes. Le meilleur entrainement consiste à recréer par nous-mêmes des fonctions qui existent déjà car ce sont en général les plus simples. **Une seule condition** : s'interdire d'utiliser ces fonctions dans cette partie sinon cela n'a aucun intérêt !

Il est donc interdit d'utiliser les syntaxes suivantes dans cette partie :

- ⋆ `sum(liste)`
- ⋆ `max(liste)`
- ⋆ `min(liste)`
- ⋆ `element in liste`
- ⋆ `liste.index(element)`
- ⋆ `liste.count(element)`

Seule la fonction `len` est autorisée.

Exercice 8 (*Sommes et produits*) :

1) Écrire une fonction `somme(liste)` renvoyant la somme d'une liste de nombres.
2) Écrire une fonction `moyenne(liste)` renvoyant la moyenne d'une liste de nombres.
3) Écrire une fonction `produit(liste)` renvoyant le produit d'une liste de nombres.
4) À l'aide de la fonction `produit`, écrire une fonction `factorielle(n)` renvoyant la factorielle d'un entier `n` en **une seule ligne** (en utilisant la syntaxe de définition en compréhension).

Exercice 9 (*Présence*) : Écrire une fonction `est_present(element, liste)` renvoyant `True` si `element` est présent dans la liste `liste` et `False` sinon.

Exercice 10 (*Indice*) : Écrire une fonction `indice(element, liste)` renvoyant l'indice du premier élément de `liste` égal à `element`.

Exercice 11 (*Maximum et minimum*) :

1) Écrire une fonction `minimum(liste)` renvoyant le minimum d'une liste de nombres.
2) Écrire une fonction `maximum(liste)` renvoyant le maximum d'une liste de nombres.
3) Écrire une fonction `indice_minimum(liste)` renvoyant l'indice du premier élément dont la valeur est minimale dans la liste.
4) Écrire une fonction `indice_maximum(liste)` renvoyant l'indice du premier élément dont la valeur est maximale dans la liste.

Exercice 12 (*Compter*) : Écrire une fonction `compte(liste, e)` qui calcule et renvoie le nombre de fois où l'élément `e` est présent dans la `liste`.

Exercice 13 (*Écart-type*) :

1) Écrire une fonction `ecart_type(liste)` renvoyant l'écart-type d'une liste de nombres en utilisant la formule suivante :

$$\sigma = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(x_i - \mu)^2}$$

où μ est la moyenne de la liste et $x_1, x_2, \ldots, x_n$ sont les éléments de la liste.

2) Réécrire la fonction précédente en utilisant l'algorithme suivante :
   - ⋆ Calculer la moyenne `m` de la liste `liste`.
   - ⋆ Construire la liste `L` dont les éléments sont les nombres $(x_i - m)^2$ en utilisant la syntaxe de définition d'une liste en compréhension.
   - ⋆ Calculer la moyenne `sigma2` de la liste `L`.
   - ⋆ Calculer et renvoyer la racine carrée de `sigma2`.

VI. Résumé des syntaxes

- ⋆ **Définition d'une liste :**
  - `nom_liste = [e_1, e_2, ..., e_n]`
  - `nom_liste = nombre * [valeur]`
  - `nom_liste = [f(k) for k in range(a, b)]`
  - `nom_liste = [f(k) for k in autre_liste]`
- ⋆ **Accès à un élément :** `nom_liste[indice]`
- ⋆ **Modification d'un élément :** `nom_liste[indice] = nouvelle_valeur`
- ⋆ **Fonctions sur les listes :**
  - **Taille / longueur d'une liste :** `len(nom_liste)`
  - **Somme d'une liste numérique :** `sum(nom_liste)`
  - **Maximum d'une liste numérique :** `max(nom_liste)`
  - **Minimum d'une liste numérique :** `min(nom_liste)`
- ⋆ **Parcours d'une liste :**
  - **Parcours des éléments :**

```
for e in nom_liste:
    Instructions # e = élément de nom_liste
```

  - **Parcours des indices :**

```
for i in range(len(nom_liste)):
    Instructions # i = indice de l'élément nom_liste[i] de nom_liste
```