Cours d'Informatique - 2M

TP02: Les listes en Python

Lionel Chatelain - lionel.chatelain1@eduvaud.ch
Kévin Faustini - kevin.faustini@eduvaud.ch

Gymnase
de Chamblandes

Année 2024-25

1 Informations

Tous les exercices sont à rendre sur Teams afin que vous puissiez recevoir des commentaires et/ou une correction. Les astérisques (*) signifient que l'exercice sera noté. Les dièses (#) signifient que l'exercice contient du contenu pour aller plus loin.

Petite astuce, si vous souhaitez tester du code rapidement, n'hésitez pas à créer d'autres fichier ou à utiliser l'interpréteur Python !

2 Exercices

Exercice 00 - Moyenne

Nous avons vu pendant le cours comment calculer la moyenne des notes d'un·e élève. Ouvrez le fichier `ex0 – moyenne.py` et remplissez le fichier à l'endroit indiqué pour calculer la moyenne des notes de cette personne. Assignez le résultat à la variable `moyenne`. L'exécution du fichier une fois rempli devrait afficher `La moyenne de l'élève est de :  4.40625`

* Exercice 01 - Moyenne pondérée

Comme vous avez choisi une spécialisation, il semble logique que certaines matières soient plus importantes que d'autres pour votre cursus et aient donc un impact plus grand sur votre moyenne générale. Ouvrez le fichier `ex1 – moyenne_et_coefficients.py` et remplissez le à l'endroit indiqué afin d'afficher la moyenne pondérée de la personne. De nouveau, assignez le résultat à la variable `moyenne`.

Note: Si vous souhaitez connaître votre moyenne à n'importe quel moment de l'année, n'hésitez pas à garder une copie de ce fichier dans vos documents. Tout ce qu'il vous restera à faire sera de remplir la liste des notes avec vos propres notes et exécuter pour obtenir votre moyenne !

* Exercice 02 - Opérations de base

Nous avons vu en cours certaines opérations de base qu'on peut effectuer sur des listes, comme par exemple ajouter un ou plusieurs éléments à une liste. Ouvrez `ex2 – op_base.py` et en utilisant les méthodes vues en cours, remplissez le fichier à l'endroit indiqué afin

(a) d'ajouter l'élément manquant à la liste `une_liste_incomplete`,

(b) de transformer la liste `une_liste_heterogene` en remplaçant les formes littérales des nombres par leur forme numérique,

(c) de supprimer les éléments de `une_liste_avec_trop_de_trucs` afin qu'elle ne contienne plus que des éléments de type `str`.

Une fois fait, l'exécution du fichier devrait afficher :

```
["Oh", "Djadja", "Y a", "pas", "moyen", "Djadja"]
[3, 17, 9, 30]
```

```
["un petit chocolat", "on peut faire plein de trucs chouettes avec la prog", "une string vide"]
```

Note: Il ne suffit pas de réécrire à la main le résultat attendu, il faut utiliser les fonctions et méthodes vues en cours. ;) Cela dit, il y a plusieurs façons de réaliser ces trois exercices !

Exercice 04 - Les fonctions built-in

Une fonction "built-in" (ou native, ou intégrée) est une fonction qui est directement disponible avec le langage sans avoir besoin d'être définie ou importée à partir d'une bibliothèque externe. Celles- ci sont souvent très pratiques pour effectuer des tâches de base. Nous avons par exemple vu en cours qu'il est possible d'utiliser `sorted(ma_liste)` afin de trier une liste, `ma_liste.index(mon_element)` afin d'obtenir la position d'un certain élément dans une liste ou encore `min(ma_liste)` afin d'obtenir le plus petit nombre présent dans la liste (d'ailleurs petite question, comment feriez-vous pour obtenir le plus petit élément d'une liste sans utilser la fonction `min()` ?)

Ouvrez le fichier `ex4 – built–in_functions.py` et remplissez le fichier à l'endroit indiqué.

(a) Trouvez le plus grand élément de la liste `des_nombres` en utilisant une fonction "built-in" et assignez-le à la variable `max_avec_une_seule_fonction` ,

(b) Trouvez le plus grand élément de `des_nombres_2_le_retour` **sans** utiliser la fonction built-in de la question **(a)** et assignez-le à la variable `max_sans_max` ,

(c) La string *"Charlie"* se cache à un endroit de la liste `ou_est_charlie` . Trouvez son index et assignez le à la variable `index_de_charlie` ,

(d) Triez les listes `une_liste` , `une_autre_liste` , `encore_une_liste` et `encore_et_toujours_une_liste` et assignez les respectivement aux variables `une_liste_triee` , `une_autre_liste_triee` , `une_liste_triee_mmmh` et `encore_et_toujours_une_liste_triee` . Est-ce que vous arrivez à trier la 3e liste ? Pourquoi ? Et qu'observez vous pour la 4e liste ?

(e) Inversez la liste `le_pas_de_MJ` .

Indice: La seule fonction non vue en cours dont vous aurez quand même besoin est `reversed()` .

Note: En programmation, et surtout au début, Google sera souvent votre meilleur ami, vous trouverez sur ce lien une liste de fonctions natives dont vous pouvez vous inspirer et sur ce lien vous trouverez des fonctions spécifiques aux listes Python dont vous pouvez aussi vous servir.

Exercice 05 - Le slicing

Le slicing est un concept très important et très puissant en Python ! Il vous permettra d'écrire du code clair et concis sans avoir à passer par des boucles interminables. Ouvrez le fichier `ex5 – slicing.py` et utilisez le slicing pour:

(a) rassurer vos invité·e·s en leur servant une `bonne_raclette` plutôt qu'une `mauvaise_raclette` en extrayant seulement les ingrédients nécessaires pour une raclette réussie,

(b) éviter d'empoisonner vos invité·e·s en leur servant des crêpes au ciment et au plastique,
Indice: que se passe-t-il quand on écrit `ma_liste[1:3] = []` *et que* `ma_liste` *contient au moins 3 éléments ?*

(c) séparer les deux groupes d'artistes `artistes_1` et `artistes_2` en fonction de leur genre musical vers `artistes_pop` et `artistes_rap`
Indice: que se passe-t-il lorsqu'on écrit `ma_liste_1 + ma_liste_2` *?*

Note: Dans certains fichiers, vous verrez que la fonction `liste_2 = liste_1.copy()` *est utilisée. Celle-ci permet de créer une nouvelle liste identique à* `liste_1` *mais qui est indépendante de celle-ci, c'est-à-dire que toute modification ultérieure de* `liste_2` *ne sera pas répercutée sur* `liste_1`*, et vice-versa. Les raisons et le fonctionnement exact fait appel à du contenu avancé de programmation et il n'est pas nécessaire d'en savoir plus pour le moment, mais si ceci vous intéresse, n'hésitez pas à venir me voir !*

Exercice 06 - Des listes de listes ?!

En Python, une liste peut contenir des types de base, comme `int` ou `str`, mais peut aussi contenir d'autres listes !

```
une_liste_de_liste = [[1, 2], [3, 4]]
print(une_liste_de_liste[0]) # affiche [1, 2]
print(une_liste_de_liste[1][0]) # affiche 3
```

Il se trouve que dans la classe de 2M42, une certaine élève bavarde un peu trop. Il va falloir la déplacer vers l'avant de la classe, voire même l'envoyer en retenue en cas de récidive... Ouvrez et travaillez avec le fichier `ex6 – la_classe_2M42.py`.

(a) Commencez par assigner à la variable `nb_eleves` le nombre d'élèves dans cette classe. Pensez aux fonctions que vous avez vues en cours.

(b) Écrivez du code de façon à ce que Sara soit tout devant dans la liste `la_classe_de_2M42_bis` (c'est à dire à l'index 0).

(c) Comme Sara continue de bavarder en même temps que l'enseignant, celui-ci décide de l'envoyer en retenue. La salle de retenue contient des tables avec 2 places par table. Envoyez Sara en retenue à côté de Dan. Vérifiez que Sara ne soit plus présente dans la classe ! (`la_classe_de_2M42_sans_Sara`).

(d) Maintenant que le calme est revenu, il est l'heure de présenter deux nouveaux élèves dans la classe : ajoutez les élèves présents dans la liste `les_ptits_nouveaux` à la classe (`la_classe_de_2M42_avec_les_nouveaux`).
N'oubliez pas que `la_classe_de_2M42_avec_les_nouveaux` doit aussi contenir tous les élèves de départ sauf Sara !

Exercice 07 - Les compréhensions de liste

Bon, nous avons pu voir déjà plusieurs fonctionnalités très utiles des listes Python, mais il en reste encore une (au moins...) qui n'est pas forcément toujours nécessaire mais qu'il est bon de connaître pour écrire du code concis: les compréhensions de liste.

Une compréhension de liste, qu'est-ce que c'est ? Une compréhension de liste est une syntaxe concise pour créer une nouvelle liste en appliquant une expression à chaque élément d'une séquence (comme une liste ou un range) et, optionnellement, en filtrant les éléments en fonction d'une condition. C'est une manière plus compacte et lisible d'écrire des boucles for combinées à des opérations de transformation ou de filtrage.

```python
nombres = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

nombres_carres = [n**2 for n in nombres]
print(nombres_carres) # affiche [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

nombres_carres_pairs = [n**2 for n in nombres if n % 2 == 0]
print(nombres_carres_pairs) # affiche [4, 16, 36, 64, 100]

nombres_pairs_carres_impairs_plus1000 = [n**2 if n % 2 == 0 else n+1000 for n in
    nombres]
print(nombres_pairs_carres_impairs_plus1000) # affiche [101, 4, 103, 16, 105, 36, 107,
    64, 109, 100]
```

Voyons voir un peu ce qu'il se passe. Dans `[n**2 for n in nombres]` :

- `nombres` est la liste définie juste au-dessus,
- `n` est une variable qui parcours la liste (de la même façon que `i` dans `for i in ma_liste`),
- et `n**2` est la fonction qu'on applique à chacun des `n`, c'est à dire à chacun des éléments de la liste `nombres`

Donc si on récapitule, la compréhension de liste permet de créer une nouvelle liste à partir d'une autre en appliquant une fonction à chacun des éléments de la liste initiale, ce qui nous permet d'obtenir `[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`.

Maintenant, pour la compréhension `[n**2 for n in nombres if n % 2 == 0]`, le début fonctionne de façon similaire **mais** on rajoute en plus une condition : on ne veut plus appliquer la fonction x^2 à *tous* les éléments, mais seulement aux éléments de `nombres` qui sont pairs (`if n % 2 == 0`)

Êtes-vous capable de deviner ce qu'il se passe dans `[n**2 if n % 2 == 0 else n+1000 for n in nombres]` ? Notez comme ici le `for n in nombres` est situé *après* le `if-else` et non pas *avant* comme lorsqu'il n'y avait que le `if`.
Notez aussi que tout ce qui est décrit ici peut être écrit de façon différente avec ce que vous avez déjà vu jusqu'à maintenant !

Bien, maintenant que le concept des compréhensions de liste a été présenté, ouvrez le fichier

`ex7 — for_comprehension.py` et réalisez les taches suivantes.

(a) En utilisant une compréhension de liste, assignez à `top_players_solo` une liste avec les scores de `scores_solo` qui sont supérieurs à 1500. Combien de joueur·euse·s sont parmi les meilleur·e·s ?

(b) En utilisant une compréhension de liste, assignez à `scores_razzia_combines_par_equipe` une liste contenant les scores des deux équipes présentes dans `scores_razzia_3v3`. A quelle position est l'équipe gagnante ?

(c) En utilisant une compréhension de liste, assignez à `scores_solo_newbie` une liste qui reprend les scores de `scores_solo` avec la nuance suivante: les scores inférieurs ou égaux à 1500 obtiennent un bonus *newbie* de 300 points tandis que les autres sont pénalisés par un multiplicateur de 0.8 sur leur score. A quelles positions sont les 3 premier·ère·s joueur·euse·s ?