II - TP 2

Créer et modifier des listes

1 Objet mutable

On connaît jusqu'à présent trois types itérables : le type `list`, le type `tuple`, et le type `str`.

Code python

```
iter1 = 1, 2, 3
iter2 = (1, 2, 3)
iter3 = [1, 2, 3]
iter4 = "123"
```

Remarque. On observe dans la console :

Résultat

```
>>> 1, 2, 3
(1, 2, 3)
```

Ainsi `iter1` et `iter2` sont **rigoureusement** identiques.

`iter1` (et `iter2`) ont pour type `tuple`, `iter3` a pour type `list`, et `iter4` a pour type `str`.
On peut donc en parcourir les éléments à l'aide d'une boucle `for` (par indice ou par valeur), et accéder à l'élément d'indice `i` d'`iterable` avec la syntaxe `iterable[i]`.
Il est possible de **modifier en place** la valeur d'un élément dans une liste. Cela se fait à l'aide d'une affectation, comme dans l'exemple ci-dessous.

Code python

```
notes = [9, 11, 15]
notes[0] # s'évalue en 9
notes[0] = 10
print(notes) # l'élément d'indice 0 a été modifié
```

Résultat

```
[10, 11, 15]
```

Cependant, il n'est pas possible de réaliser cette opération sur des objets de type `tuple` ou `str`.

Code python

```
notes = (9, 11, 15)
notes[0] # s'évalue en 9
notes[0] = 10
```

Résultat

```
TypeError Traceback (most recent
 ↪  call last)
Cell In[386], line 3
      1 notes = (9, 11, 15)
      2 notes[0] # s'évalue en 9
----> 3 notes[0] = 10
      4 print(notes)

TypeError: 'tuple' object does
 ↪  not support item assignment
```

Code python

```
prenom = "drédéric"
prenom[0] # s'évalue en "d"
prenom[0] = "f"
```

Résultat

```
TypeError Traceback (most recent
 ↪  call last)
Cell In[401], line 3
      1 prenom = "drédéric"
      2 prenom[0] # s'évalue en
       ↪  "d"
----> 3 prenom[0] = "f"

TypeError: 'str' object does not
 ↪  support item assignment
```

On dit que les objets de type `list` sont des objets **mutables**, tandis que les objets de type `tuple` et `str` sont des objets **non mutables**.

2 Modification en place d'une liste

2.1 Notes arrondies

Un professeur de NSI enseignant au lycée Fustel utilise un programme python pour gérer les notes de ses élèves. Il dispose d'une liste `notes` de flottants :

Code python
```
notes = [16.5, 9.9, 6.4, 7.8, 11.7, 15.4]
```

Question 1. **1.** À la fin d'un cours, inquiet de ce qu'il voit sur pronote, un élève vient voir le professeur et lui dit qu'il n'a pas obtenu `6.4` au contrôle de NSI mais bien `16.4` !

Écrire l'instruction python qui permet de modifier la variable `notes` afin de corriger le problème.

2. Dans sa grande magnanimité, ce professeur de NSI décide d'arrondir les notes **au demi-point supérieur**. Ainsi, la note de `16.5` sera arrondie à `17`, mais la note de `15.4` sera arrondie à `15.5`.

Écrire une fonction python `arrondir_notes` qui prend en argument une liste de `notes` et qui **modifie en place** les notes de la liste en les arrondissant au demi-point supérieur.

Code python
```
def arrondir_notes(notes):
    """ [float] -> None
    Arrondi les notes au demi-point supérieur """
    pass
```

Résultat
```
>>> notes
[16.5, 9.9, 16.4, 7.8, 11.7, 15.4]
>>> arrondir_notes(notes)
>>> notes
[17.0, 10.0, 16.5, 8.0, 12.0, 15.5]
```

Indication. Si `a` est un flottant, alors `a%1` représente la partie fractionnaire de `a`, et `a//1` est sa partie entière. Par exemple :

Résultat
```
>>> a = 14.25
>>> a % 1
0.25
>>> a // 1
14.0
```

3. `notes` est la liste `[10, 11, 12]`.

a. Comment est modifié la liste `notes` après l'exécution de `arrondir_notes(notes)` ? Ce comportement est-il souhaitable ?

b. Modifier la fonction `arrondir_notes` afin de corriger ce problème.

4. `notes` est le tuple `(10, 11, 12)`.

Que se passe-t-il lorsque l'on exécute l'instruction `arrondir_notes(notes)` ? À quoi ce problème est-il dû ?

2.2 Appliquer une réduction aux éléments d'un panier

Au supermarché Dôvv, la gérante est très satisfaite par le programme permettant de calculer le prix à payer par le client. Elle souhaiterait voir implémentée une nouvelle fonctionnalité afin de pouvoir gérer les réductions sur certains articles.

Par exemple on peut avoir :

Code python
```
objets = ["pomme", "ananas", "prune"]
reduction = [10, 5, 50]
prix = [100, 500, 50]
```

Cela signifie que les pommes, initialement à 100 francs l'unité, voient leur prix diminuer de 10%. Le prix des ananas est lui diminué de 5%, le prix des prunes est diminué de 50%.

Écrire une fonction `applique_reduction` qui prend en argument une liste `prix` et une liste `reduction` et qui **modifie en place** les éléments de la liste `prix` pour leur appliquer le pourcentage de réduction correspondant. Les nouveaux prix seront arrondis à l'entier inférieur.

Rappel. On rappelle que pour diminuer un prix p de t% on utilise la formule $p \times \left(1 - \frac{t}{100}\right)$.

Code python
```
def applique_reduction(prix, reductions):
    """ [int], [int] -> None
    prix et reductions sont deux listes de même taille
    Modifie la liste prix en place pour leur appliquer la réduction
    ↪ correspondante """
    pass
```

Code python
```
reductions, prix = [10, 5, 50], [100, 500, 50]
applique_reduction(prix, reductions)
assert prix == [90, 475, 25]
```

Question 2. **1.** Les réductions sont maintenant maintenant terminées.

Si p est le prix avant réduction et p' le prix après réduction de t%, quel doit être le pourcentage d'augmentation à appliquer à p' pour obtenir l'ancien prix p ?

2. Écrire une fonction `inverse_reduction` qui prend en argument une liste `reductions` et qui la **modifie en place** afin que les nouveaux éléments de la liste soient les pourcentages d'évolutions reciproques de réductions initiales.

Code python
```
def inverse_reduction(reductions):
    """ [int] -> float
    Remplace chaque élément de la liste reductions par son pourcentage
    ↪ d'augmentation reciproque correspondant """
    pass
```

Code python
```
reductions = [10, 5, 50]
inverse_reduction(reductions)
print(reductions)
```

3. En déduire une fonction `fin_soldes` qui prend en argument une liste `prix` et une liste `reduction`, la liste `prix` correspondant aux prix des articles **après** réduction. La fonction `fin_soldes` doit **modifier en place** les éléments de la liste `prix` pour que ceux-ci soient égaux à ce qu'ils valaient avant la période de soldes à un franc près.

Les prix seront arrondis à l'unité. La liste `reductions` peut-être modifiée.

Code python

```python
def fin_soldes(prix, reductions):
    """ [int], float -> None
    Annule les effets de la reduction sur les prix """
    pass
```

Code python

```python
reductions, prix = [10, 5, 50], [100, 500, 50]
print(prix, reductions)
applique_reduction(prix, reductions)
print(prix, reductions)
fin_soldes(prix, reductions)
print(prix, reductions)
```

Résultat

```
[100, 500, 50] [10, 5, 50]
[90, 475, 25] [10, 5, 50]
[100, 500, 50] [11.111111111111116, 5.263157894736836, 100.0]
```

3 Création de listes en compréhension

Jusqu'à présent on a créé des listes en **extension**. C'est à dire que l'on a listé de manière exhaustive, "à la main", tous les éléments de la liste les uns après les autres. Par exemple `[1, 2, 3]`.
En python comme en mathématiques, il est possible de créer des listes en **compréhension**, c'est à dire, non pas de donner les éléments, mais de donner une manière de calculer les éléments de la liste.

- Entiers inférieurs ou égaux à 5.

Résultat

```
>>> [i for i in range(6)]
[0, 1, 2, 3, 4, 5]
```

- Table de multiplication (partielle) de 3.

Résultat

```
>>> [3*i for i in range(7)]
[0, 3, 6, 9, 12, 15, 18]
```

- Une liste de 8 zéros.

Résultat

```
>>> [0 for i in range(8)]
[0, 0, 0, 0, 0, 0, 0, 0]
```

- La liste des 5 premiers nombres au carré.

Résultat

```
>>> [i**2 for i in range(5)]
[0, 1, 4, 9, 16]
```

Question 3. Dans cet exercice, on a déjà exécuté l'instruction : `t = [42, 23, 2, 3, 50]`.
Donner les listes créées par ces listes en compréhension :

1. `[0 for v in t]`
2. `[v for v in t if v >= 10]`
3. `[i for i in range(len(t)) if (t[i] % 2) == 1]`
4. `[ch[0] for ch in ["Ada", "Alice", "Bob", "Eve"]]`
5. `[i % 10 for i in range(9, 12)]`
6. `[i % 10 for i in range(55) if (i // 50) == 1]`
7. `[i for i in range(len(t)) if t[i] == i]`

3.1 Copier une liste : phénomène d'aliasing

Question 4. Considérons les instructions suivantes :

Résultat

```
>>> liste1 = [1, 2, 3]
>>> liste2 = liste1
>>> liste1[0] = 10
```

1. Recopier ces instructions dans la console.

   Qu'affiche l'instruction `print(liste2)` ? Commenter.

2. Écrire une fonction `copie_independante` qui prend en argument une `liste` et qui recopie dans une nouvelle liste tous les éléments de `liste` dans le même ordre avant de la renvoyer.

Code python

```
1 def copie_independante(liste):
2     """ [int] -> [int]
3     Créé une copie indépendante de
      ↪ liste """
4     pass
```

Code python

```
liste1 = [1, 2, 3]
liste2 = copie_independante(liste1)
liste1[0] = 10
assert liste2 == [1, 2, 3]
```

3.2 Ajouter un élément à une liste

Écrire une fonction `ajoute_fin` qui prend en argument une `liste` et un élément `e` et qui renvoie une copie indépendante de `liste` dans laquelle l'élément `e` a été ajouté à la fin.

Code python

```
1 def ajoute_fin(liste, e):
2     """ [int], int -> [int]
3     Renvoie une liste constituée des éléments de l suivis de e """
4     pass
```

Code python

```
1 l = [1, 2, 3]
2 print(ajoute_fin(l, 4))
3 print(l)
4 l = ajoute_fin(l, 8)
5 print(l)
```

Résultat

```
[1, 2, 3, 4]
[1, 2, 3]
[1, 2, 3, 8]
```

Question 5. 1. Justifier les affichages produits par les lignes 2, 3, et 5 du code ci-dessus.

2. **Challenge.** Écrire une fonction `ajoute_pos` qui prend en argument une liste `liste`, un élément `e` et un indice `i` et qui renvoie la liste constituée des éléments de `liste` dans laquelle on a inséré l'élément `e` à l'indice `i` (les éléments de `liste` d'indice supérieur ou égal à `i` sont donc décalés vers la droite).

Code python

```
1 def ajoute_pos(liste, e, i):
2     """ [int], int, int -> [int]
3     0 <= i <= len(liste)
4     Renvoie une liste indépendante de l dans laquelle l'élément e a
      ↪ été inséré à l'indice i """
5     pass
```

3.3 Supprimer un élément d'une liste

Écrire une fonction `supprime_pos` qui prend en argument une `liste` et un indice `i` compatible avec la taille de la liste et qui renvoie une liste indépendante constituée des éléments de `liste` dans laquelle l'élément d'indice `i` de `liste` n'apparaît pas.

Code python

```python
def supprime_pos(liste, i):
    """ [int], int -> [int]
    0 <= i < len(liste)
    Supprime l'élément d'indice i de liste ""
    pass
```

Code python

```python
liste = [1, 2, 3]
print(supprime_pos(liste, 0))
print(supprime_pos(liste, 1))
print(supprime_pos(liste, 2))
```

Résultat

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

3.4 Renverser une liste

Écrire une fonction `renverse` qui prend en argument une `liste` et qui renvoie une liste dont les éléments sont ceux de `liste` lus de droite à gauche.

Code python

```python
def renverse(liste):
    """ [int] -> [int]
    Renvoie une liste dont les éléments sont ceux de liste lus de droite à
    ↪ gauche """
    pass
```

Indication. La documentation officielle indique via `help(range)` :

Résultat

```
class range(object)
range(stop) -> range object
range(start, stop[, step]) -> range object

Return an object that produces a sequence of integers from start
(inclusive) : to stop (exclusive) by step.  range(i, j) produces
i, i+1, i+2, ..., j-1. start defaults to 0, and stop is omitted!
range(4) produces 0, 1, 2, 3. These are exactly the valid indices
for a list of 4 elements.  When step is given, it specifies
the increment (or decrement).
```

Code python

```python
l = [1, 2, 3]
print(renverse(l))
print(l)
```

Résultat

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

3.5 Liste des indices des occurrences d'un élément dans un tableau

Dans cette partie, on souhaite écrire une fonction `indices_occurrences` qui renvoie la liste des indices des occurrences de `e` dans la `liste`. Si `e` n'est pas présent dans `liste` alors la fonction renvoie la liste vide `[]`.

Question 6. **1.** Rappeler le code de la fonction `nombre_occurrences` qui prend en argument une `liste` et un élément `e` et qui renvoie le nombre de fois que l'élément `e` apparaît dans la `liste` (son nombre d'**occurrences**).

2. `liste` est la liste `[1, 1, 3, 1, 3, 2]`. Que doit renvoyer :

a. `indices_occurrences(liste, 1)`

b. `indices_occurrences(liste, 2)`

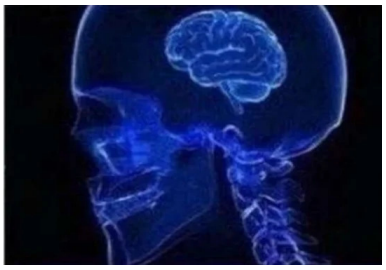
c. `indices_occurrences(liste, 3)`

d. `indices_occurrences(liste, 4)`

Question 7. On propose trois implémentations différentes pour la fonction `indices_occurrences`. Compléter les codes correspondant à la description de l'énoncé. Tester chacune des fonctions.

Implémentation 1.

Pour déterminer la liste des occurrences de `e` dans `liste`, on initialise une liste `l` de bonne taille (elle contiendra les indices de chaque occurrence de `e`), et un entier `i` qui servira à mémoriser où on doit stocker dans `l` l'indice de la prochaine occurrence de `e`. On parcourt `liste` par indice : si l'élément d'indice `k` de `liste` est une occurrence de `e`, alors on stocke `k` à l'indice `i` de `l` et on incrémente `i`.

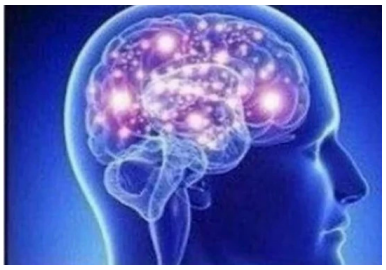


Code python

```
def indices_occurrences(liste, e):
    n = nombre_occurrences(liste, e)
    l = [0 for i in ...]
    i = 0
    for k in range(liste):
        if ...:
            ...
            ...
    return ...
```

Implémentation 2.

On fait appel à la fonction `ajoute_fin` pour ajouter successivement les indices des occurrences de `e` à la liste `l`, sans chercher à connaître au préalable le nombre d'occurrences de `e`.



Code python

```
def indices_occurrences(liste, e):
    l = []
    for k in range(liste):
        if ...:
            ...
    return ...
```

Implémentation 3.

On détermine directement les indices des occurrences de `e` dans `l` à l'aide d'une liste en compréhension et d'un filtrage.

Code python

```
def indices_occurrences(liste, e):
    return [...]
```