Séquences

Objectifs

— Comprendre et savoir utiliser les séquences
— Écrire et tester des sous-programmes (toujours !)
— Raffiner (toujours !)

1 Comprendre les séquences

Exercice 1 : Opérations sur les séquences
Compléter le programme `comprendre-sequence-a-trous.py` ci-après pour écrire les instructions Python qui réalisent les objectifs exprimés en commentaires. On utilisera **`assert`** pour indiquer le résultat des expressions écrites ou l'effet des instructions écrites.

```
'''Opérations sur les séquences (avec une liste).
Utiliser assert pour indiquer les résultats attendus.'''

# Initialiser un nom s avec une liste contenant dans l'ordre 9, 1, 5, 2, 1 et 3
...

# Obtenir la taille de la liste s (le nombre d'éléments qu'elle contient)
assert 6 == ...

# Savoir si l'élément 2 est présent dans s.  Idem pour l'élément 4
...

# Obtenir le premier élément de s
assert 9 == ...

# Obtenir le dernier élément de s
assert 3 == ...

# Obtenir la fréquence (nombre d'occurences) de 1 dans s
assert 2 == ...

# Supprimer l'élément à l'indice 2 de s
...
assert s == [9, 1, 2, 1, 3]

# Supprimer l'élément 2 de s
...
```

```python
assert s == [9, 1, 1, 3]

# Ajouter 7 à la fin de s
...
assert s == [9, 1, 1, 3, 7]

# Ajouter 6 en position 2 dans s
...
assert s == [9, 1, 6, 1, 3, 7]

# Ajouter 0 avant la première occurrence de x dans s (x pourrait être 1, 3...)
for x in (1, 3):
    ...
assert s == [9, 0, 1, 6, 1, 0, 3, 7]

# Obtenir la somme des carrés des nombres de s ?
...
assert somme_carres == 177
```

Exercice 2 : Le type list

Indiquer l'effet des instructions du programme suivant :

```python
s = []
s.append(2)             # s ?
s.insert(0, 4)          # s ?
s.insert(20, 7)         # s ?
s[1] = 'd'              # s ?
s[2] /= s[2]            # s ?
s.count(1)              # ?
s[0], s[1] = s[1], s[0] # s ?

p, _, d = s             # p ? _ ? d ?
premier, *suite = s     # premier ? suite ?

b = [False, True]
s.extend(b)             # s ?

x = s.pop(1)            # x ? s ?

s2 = [2, 3, 5]
i, s2[i], x = s2        # i ? s2 ? x ?

s.append(s2)            # s ?
s2.append(s)            # s2 ? s ?

t = tuple(b)            # t ?
s = list('Fin.')        # s ? s2 ?
```

Exercice 3 Répondre de manière concise et précise aux questions suivantes.

3.1. Expliquer ce qu'est un objet immuable. On donnera des exemples.
3.2. Quelles sont les séquences proposées par Python ? Donner un exemple de chaque.
3.3. Indiquer ce que signifient `is` et `==` en Python. Donner un exemple.

Exercice 4 : Éléments d'un n-uplet
Considérons le n-uplet (tuple) `t = (5, 'x', [])`.
4.1. Indiquer comment accéder aux composants du n-uplet : `5`, `'x'` et `[]`.
4.2. Indiquer si les instructions suivantes sont autorisées et, dans l'affirmative, ce que vaut `t`.

```
t[0] = 6
del t[1]
t[2].append(-1)
```

2 Algorithme de tri

Exercice 5 : Tri par sélection
Soit A un vecteur (une liste en Python) de N entiers relatifs quelconques, l'objectif est de trier le vecteur A en utilisant le tri par sélection. Le vecteur A est trié si $A[i] \leq A[i+1]$.

Le tri par sélection est un tri en $(N-1)$ étapes. L'étape i consiste à ranger à sa place le i^{e} plus petit élément du vecteur.
Exemple : Voici les différentes valeurs du vecteur 8 2 9 5 1 7 après chaque étape (la partie encadrée correspond à la partie du vecteur déjà traitée et donc triée) :

vecteur initial	:	8 2 9 5 1 7
après l'étape 1	:	[1] 2 9 5 8 7
après l'étape 2	:	[1 2] 9 5 8 7
après l'étape 3	:	[1 2 5] 9 8 7
après l'étape 4	:	[1 2 5 7] 8 9
après l'étape 5	:	[1 2 5 7 8 9]

5.1. Écrire un sous-programme qui trie un vecteur en utilisant le tri par sélection. On complètera le fichier `tri_selection.py` et on utilisera le fichier `test_tri_selection.py`. On affichera la liste après chaque étape du tri pour vérifier que c'est bien l'algorithme demandé qui a été implanté. On testera avec :

```
pytest --doctest-modules test_tri_selection.py
```

3 Pour aller plus loin...

Exercice 6 : Le jeu du pendu
Écrire un programme qui permet de jouer au pendu, la machine étant le bourreau.

Le jeu du pendu se joue à deux. Un premier joueur, le *bourreau*, choisit un nom commun qu'il garde secret. Il indique seulement à l'autre joueur, le *joueur*, le nombre de lettres du mot. Le joueur propose une lettre. Si cette lettre correspond à une lettre du mot, alors le bourreau indique le nombre d'occurrences de la lettre dans le mot et la position de chacune des occurrences. Si la lettre ne fait pas partie du mot ou si la lettre a déjà été proposée, le bourreau ajoute une pièce à la potence. Si la potence est complète (11 pièces), le joueur perd : il est pendu ! S'il trouve toutes les lettres du mot, il gagne.

Dessiner la potence et le pendu représente 11 tracés qui sont dans l'ordre :

— la *potence* : le socle, le montant vertical, la barre horizontale, le renfort ;
— le *pendu* : le corps, la jambe gauche, la jambe droite, le bras gauche, le bras droit, la tête ;
— la *corde* : lorsque la code est placée, la partie est finie : le joueur est pendu !

Avant chaque demande de proposition au joueur, l'ordinateur affichera les lettres déjà identifiées du mot à trouver. En fait, il affiche le mot à trouver en mettant un tiret « – » à la place des lettres non encore trouvées. C'est ce qu'on appelle le « mot du joueur ».

Par exemple, si le mot à trouver est « élève » et que le joueur propose successivement « A », « E », « T », « R », « C », « L », « V », le programme affichera les lignes suivantes (l'affichage de la potence n'est pas reproduit).

```
Mot :  - - - - -

Une lettre ? A
J'ajoute une pièce à la potence !

Mot :  - - - - -

Une lettre ? E
La lettre "E" est dans le mot.

Mot :  E - E - E

Une lettre ? T
J'ajoute une pièce à la potence !

Mot :  E - E - E

Une lettre ? R
J'ajoute une pièce à la potence !

Mot :  E - E - E

Une lettre ? C
J'ajoute une pièce à la potence !

Mot :  E - E - E

Une lettre ? L
La lettre "L" est dans le mot.

Mot :  E L E - E

Une lettre ? V
La lettre "V" est dans le mot.

Bravo, vous avez trouvé "ELEVE".
```

Attention : Pour faciliter la lecture du mot, les lettres (ou les tirets) seront affichées en étant séparées par un espace.

Simplifications :

1. On suppose que les mots à trouver seront *toujours* orthographiés sans accents et en majuscules. Ainsi pour le mot « Élève », le mot à trouver est « ELEVE ».
2. On suppose également que les mots à trouver ne comportent que des lettres et aucun autre signe. On ne peut donc pas avoir de mots tels que « aujourd'hui », « chasse-neige »...