Lycée Saint-Louis — Octobre 2023
1Bio3 — Informatique

TP 3 : Chaînes et listes

1 Chaînes de caractères

1.1 Définition

Une **chaîne de caractères** est une suite quelconque de caractères (lettres, chiffres, symboles, *etc.*). Elle est délimitée par des apostrophes ou des guillemets. C'est un objet Python de type `str`.
La **chaîne vide** se note `""` (deux guillemets avec rien entre) ou `''` (deux apostrophes avec rien entre).

La **longueur** d'une chaîne de caractères est le nombre de caractères qu'elle contient. La longueur d'une chaîne `d` s'obtient par l'instruction `len(d)`.

Exemple 1

`"1BCPST 3"` est une chaîne de caractères de longueur 8 (l'espace est un caractère particulier).

1.2 Indice d'un caractère

Soit `d` une chaîne non vide de longueur $n \in \mathbb{N}^*$. Les caractères de `d` sont numérotés de deux façons :

1. de gauche à droite (de 0 à $n-1$) ;
2. de droite à gauche (de -1 à $-n$).

On peut accéder au caractère d'indice $i \in \mathbb{N}$ de `d` par l'instruction `d[i]`.

Exemple 2

Considérons la chaîne de caractères `d = "Python 3"`, de longueur 8. Le graphique ci-dessous donne les différents indices (aussi appelées rangs ou numéros) de chaque caractère.

d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
P	y	t	h	o	n		3
d[-8]	d[-7]	d[-6]	d[-5]	d[-4]	d[-3]	d[-2]	d[-1]

1.3 Opérations sur les chaînes de caractères

1.3.1 Test d'appartenance

On peut facilement tester si un caractère donné est présent dans une chaîne de caractères de la manière suivante : L'instruction `'a' in ch` est un booléen qui vaut `True` si le caractère `'a'` figure dans la chaîne `ch`, et qui vaut `False` sinon.

1.3.2 Concaténation

Pour concaténer deux chaînes, on utilise le symbole `+`.

Exemple 3

Si `ch1 = 'tagada'` et `ch2 = 'tsointsoin'`, alors `ch1+ch2` donne la chaîne `'tagadatsointsoin'`.

Pour concaténer plusieurs fois la même chaîne, on peut utiliser le symbole `*`.

Exemple 4

Supposons que `ch = 'tin'`. Les instructions `3*ch` ou `ch*3` donnent la chaîne `'tintintin'`. Ces instructions sont une forme condensée de l'instruction `ch+ch+ch`.

1.3.3 Extraction d'une sous-chaîne (ou tranche)

Soient a et b des entiers positifs et `ch` une chaîne de caractères.

- Si $a < b$, l'instruction `ch[a:b]` désigne la sous-chaîne de `ch` du rang a inclus au rang b exclu.
  De plus, si $c \in \mathbb{N}^*$, alors `ch[a:b:c]` désigne la sous-chaîne de `ch` du rang a inclus au rang b exclu, avec un pas de c.
- Si $a \geqslant b$, `ch[a:b]` désigne la chaîne vide.

Exemple 5

On suppose que `ch = "opérations"`.

- L'instruction `ch[0:5]` donne `"opéra"`.
- L'instruction `ch[3:8]` donne `"ratio"`.
- L'instruction `ch[3:8:2]` donne `"rto"`.

1.3.4 Modification ou suppression d'un caractère

On ne peut pas modifier ou supprimer un caractère dans une chaîne déjà créée. On dit que les chaînes de caractères sont **non-mutables**.

1.4 Premiers exercices

Exercice 1. Répondre sans utiliser l'ordinateur, puis vérifier dans la console.
On considère la chaîne de caractère `a = "Bonjour"`. Donner le résultat de chacune des instructions ci-dessous (si la syntaxe est incorrecte, répondre « erreur »).

1. `a[0]`, `a[3]`, `a[-3]`, `len(a)`, `a[7]`, `a[0:4]`, `a[1:5:2]`
2. `a[0]+a[-1]`, `a[0]+4*a[-1]`
3. `'B' in a`, `'b' in a`, `a[1]=='o'`, `a[1]='B'`

Exercice 2. Écrire une fonction d'en-tête `appartient(car, ch)` qui prend en entrée un caractère `car` et une chaîne `ch` et qui simule le test d'appartenance `car in ch`, c'est-à-dire qui renvoie `True` si `car` figure dans `ch` et `False` sinon. Bien sûr on n'utilisera pas la commande `in`, le but étant de la reproduire. On sera conduit à :

- affecter à une variable `n` le nombre de caractères de `ch` ;
- utiliser une boucle *for* et une instruction conditionnelle.

2 Listes

2.1 Définition

Une **liste** est une suite finie d'éléments de types quelconques (entiers, flottants, chaînes, autres listes, *etc.*). Elle est délimitée par des crochets et ses éléments sont séparés par des virgules. L'ordre a une importance. C'est un objet Python de type `list`.
La **liste vide** se note `[]`.

La **longueur** d'une liste est son nombre d'éléments. La longueur d'une liste `L` s'obtient par l'instruction `len(L)`.

Exemple 6

`[3.14, 'pi', [0,0]]` est une liste de longueur 3. Ses trois éléments sont : le flottant `3.14`, la chaîne `'pi'` et la liste `[0,0]`.

2.2 Indice d'un élément

Même syntaxe que pour les chaînes.

2.3 Opérations sur les listes

Le test d'appartenance, la concaténation, la répétition et l'extraction fonctionnent exactement comme avec les chaînes de caractères.

2.3.1 Ajout d'un élément à droite

Pour ajouter un objet Python `o` en queue d'une liste `L`, on peut soit concaténer `L` avec la liste `[o]` à un élément, soit utiliser une méthode spécifique aux listes : `L.append(o)`.

Exemple 7

On veut modifier la liste `L = [1, 2, 3]` en lui ajoutant l'élément `4` à droite.

- On peut utiliser l'instruction `L = L+[4]`, qui peut aussi s'écrire `L+=[4]`.
- On peut également utiliser l'instruction `L.append(4)`.

Après chacune de ces instructions `L` vaudra `[1, 2, 3, 4]`.

Attention, on n'écrira jamais `L = L.append(4)`, qui serait redondant : le mot-clé `append` indique déjà que `L` doit être mise à jour.

2.3.2 Suppression ou modification d'un élément

À la différence des chaînes, on peut supprimer ou modifier un élément d'une liste déjà créée (les listes sont mutables) :

- Si l'on souhaite supprimer l'élément d'indice `i` de la liste `L`, on peut utiliser l'instruction `del(L[i])` ; après quoi la liste est raccourcie de cet élément (et le nouvel élément d'indice `i`, s'il existe, est celui qui se trouvait auparavant à l'indice `i+1`).
- Si l'on souhaite remplacer l'élément d'indice `i` de la liste `L` par l'objet `o`, on procède comme pour une affectation de variable : `L[i] = o` ; après quoi la liste est modifiée.

Exemple 8

Soit `L = [1, 4, 9]`.

- On veut supprimer l'élément 4 pour obtenir la liste `L = [1,9]`. On utilise l'instruction `del L[1]`.
- L'instruction `L[0] = 10` change `L` en `[10,4,9]`.

Pour supprimer le **dernier** élément de la liste, on peut aussi utiliser la méthode `pop()`, qui a également pour effet de renvoyer cet élément.

Exemple 9

Soit encore `L = [1,4,9]`. Après l'instruction `a = L.pop()`, la liste `L` sera `[1,4]` et on aura affecté à la variable `a` la valeur 9.

Attention : La duplication des listes oblige à la prudence. Par exemple, en écrivant `x = [5,2,9]` suivi de `y = x`, on dispose en fait d'une seule et même liste qui possède deux noms. Si l'on décide alors de muter le dernier élément de la liste `x` en écrivant `x[2]=10`, la liste devient `[5,2,10]` mais son adresse ne change pas en mémoire : les variables `x` et `y` pointent toujours vers la même liste. Si l'on demande les valeurs de `x` et `y`, on obtient `[5,2,10]` dans les deux cas [1]. Tout se passe donc comme si la mutation de `x` s'était répercutée sur `y`.
La bonne syntaxe pour dupliquer la liste `x` en une nouvelle variable `y` sans qu'une mutation de `x` n'affecte ensuite `y` est `y = x.copy()`. Nous reviendrons sur ce point plus tard dans l'année. Pour l'instant, il suffit de se souvenir que la copie des listes est quelque chose de sournois.

2.4 Premiers exercices

Exercice 3. Répondre sans utiliser l'ordinateur, puis vérifier.
On considère les listes `L = [2, -4, 1, 5, 1.5, 3]` et `M = [[1,-2],[0.5, 6], [-1, 3.45]]`. Donner le résultat de chacune des instructions ci-dessous (si la syntaxe est incorrecte, répondre « erreur »).

1. `L[1]`, `L[-2]`, `len(L)`, `L[1.5]`, `L[1:5]`, `L[0:4:2]`
2. `L[0]+L[3]`, `[L[0]]+[L[3]]`, `3*L[0]`, `3*[L[0]]`
3. `L[2] == 1.5`, `L[4] == 1.5`, `2 in L`, `0 in L`, `[2,-4] in L`
4. `M[0]`, `M[0][1]`, `M[0][1][2]`, `len(M)`, `len(M[0])`, `M[0] + M[2]`

N.B. Une liste de listes comme `M` ci-dessus s'appelle un tableau (ici de dimensions 3×2).

Exercice 4.

1. Compléter les programmes suivants pour qu'ils écrivent la liste `L` de tous les entiers de 1 à 100.

Programme 1

```
L = []
for k in range(.....):
    L += .....
print(L)
```

Programme 2

```
L = 100*[0]
for k in range(.....):
    L[k] = .....
print(L)
```

2. Écrire une fonction `double : list -> list` qui multiplie par 2 tous les éléments d'une liste `L` passée en entrée et renvoie la nouvelle liste obtenue.
3. Écrire une fonction d'en-tête `carre(n)` prenant en argument un entier $n \in \mathbb{N}^*$ et renvoyant la liste des carrés des entiers de 1 à n. Par exemple, `carre(5)` doit renvoyer `[1, 4, 9, 16, 25]`.

2.5 Précisions sur la commande range

Soient a et b des entiers tels que $a < b$. La commande `range(a,b)`, dont le type est `range` et non `list`, commande à Python de parcourir l'intervalle d'entiers entre a et $b-1$. Donc, pour être précis, la commande `range` ne fabrique pas une liste, elle la parcourt. En exécutant `range(5,9)` dans la console Python, rien ne s'affiche.

La commande `list(range(a,b))` permet de transformer `range(a,b)` en objet de type `list`. En exécutant `list(range(5,9))` dans la console Python, il s'affiche : `[5,6,7,8]`. Dans la pratique, cette subtilité ne pose pas de problème car l'affichage d'un range est rarement utile ; seul compte le fait que les entiers soient parcourus.

Soient a, $b \in \mathbb{N}$ et $r \in \mathbb{Z}^*$ des entiers. La commande `range(a,b,r)` parcourt, avec un pas r, les entiers à partir de a jusqu'à b, b étant exclu. Le pas r peut être négatif quand $a > b$.

Soit $n \in \mathbb{N}^*$. La commande `range(n)` a le même effet que la commande `range(0,n)`, c'est-à-dire parcourt les entiers $0, 1, ..., n-1$.

1. C'est ainsi très différent de ce qui se passe avec les autres types de variables, où seul `x` serait modifié.

2.6 Définition en compréhension

La description d'une liste en extension, c'est-à-dire comme ci-dessus par la donnée exhaustive de ses éléments, n'est pas la seule façon de définir une liste. On peut aussi procéder en compréhension, c'est-à-dire en donnant une « formule » décrivant les composantes de la liste. Ainsi, la commande

```
L = [expression(k) for k in M]
```

permet de construire la liste `L` dont les composantes sont les objets `expression(k)`, où `k` est une variable décrivant une autre liste `M` (ou un objet de type `range`).

Exemple 10

Soit `L = [1, 2, 4, 5]`. L'instruction `[3*x for x in L]` renvoie la liste `[3, 6, 12, 15]`.

Exercice 5. Répondre sans utiliser l'ordinateur, puis vérifier.
On reprend la liste `L = [1, 2, 4, 5]`. Que renvoient les instructions suivantes ?

```
[[x,x**3] for x in L],    [3*x for x in L if x**2>5],    [i*j for i in L for j in range(3)]
```

Exercice 6. Dans la console, construire en compréhension les listes dont les composantes successives sont :

1. les inverses des entiers de 1 à 20 ;
2. les cubes des dix premiers entiers naturels impairs, rangés dans l'ordre décroissant.

Cette syntaxe `for k in L` (où `L` est une liste) permet également de créer des boucles `for` comme celles que nous avons étudiées au TP précédent : au lieu de parcourir un `range(a,b)`, le « compteur » peut parcourir n'importe quelle liste. Cela fonctionne également avec les chaînes de caractères.

Exemple 11

Soit `ch` la chaîne `"Bonjour"`. Le programme :

```
1 ch = "Bonjour"
2 for a in ch:
3     print(a)
```

aura le même effet que :

```
1 ch = "Bonjour"
2 n = len(ch)
3 for k in range(n):
4     print(ch[k])
```

(Deviner ce que fait ce dernier programme puis vérifier à l'ordinateur.)

3 Programmes types

Tous les exercices de cette section constituent la base de ce qu'on peut vous demander de savoir faire avec des boucles, des tests et des listes. Ils sont donc **à maîtriser parfaitement**.
Efforcez-vous d'ajouter des commentaires dans vos programmes (à l'aide d'un #) : cela vous aidera quand vous les reprendrez, et c'est une compétence attendue au concours.

Exercice 7. *Présence d'un élément dans une chaîne*
Voir exercice 2 ci-dessus.

Exercice 8. *Nombre d'occurrences d'un élément*
Écrire une fonction d'en-tête `compte(a, L)` prenant en argument un entier `a` et une liste d'entiers `L` et renvoyant le nombre d'occurrences de `a` dans `L`. Par exemple, pour `L = [1, 3, 4, 2, 3, 2, 1, 1]`, `compte(1, L)` doit

renvoyer 3 et `compte(9, L)` doit renvoyer 0.

Remarque : Ce dernier exercice fait intervenir une boucle, qui peut s'écrire aussi bien avec un *range* qu'avec la syntaxe décrite à la fin du paragraphe 2.6 ci-dessus. Assurez-vous de savoir faire les deux.

Variante : On peut de la même manière compter le nombre d'occurrences d'un caractère dans une chaîne.

Exercice 9. *Somme des éléments d'une liste*
Écrire une fonction d'en-tête `somme(L)` prenant en argument une liste d'entiers et renvoyant la somme de ses éléments.

Remarque : Il existe en fait déjà une fonction Python qui fait cela : la fonction `sum`. Reproduire les fonctions déjà implémentées dans Python est un excellent exercice.

Exercice 10. *Rangs d'un élément*
Écrire une fonction d'en-tête `rangsOccurrences(a, L)` prenant en argument un entier `a` et une liste d'entiers `L` et renvoyant une liste contenant les indices des occurrences de `a` dans `L`. Par exemple, pour `L = [1, 3, 4, 2, 3, 2, 1, 1]`, `rangsOccurrences(1,L)` devra renvoyer `[0, 6, 7]`, et `rangsOccurrences(5,L)` devra renvoyer la liste vide.

Exercice 11. *Recherche du maximum*

1. Écrire une fonction d'en-tête `maximum(L)` prenant en argument une liste d'entiers et renvoyant son maximum.
2. Écrire une fonction d'en-tête `rangsMaximum(L)` prenant en argument une liste d'entiers et renvoyant la liste des rangs auxquels apparaît son maximum. Cette fonction appellera directement les fonctions `maximum` et `rangsOccurrences` ci-dessus.
3. À l'aide des deux questions précédentes, écrire une fonction `deuxiemeMax(L)` prenant en argument une liste d'entiers et renvoyant son deuxième maximum, c'est-à-dire le deuxième nombre dans l'ordre décroissant.

Exercice 12. *Test d'une condition sur une liste*
Écrire une fonction d'en-tête `positifs(L)` prenant en argument une liste d'entiers et renvoyant `True` si tous ces entiers sont positifs ou nuls, et `False` sinon.

Remarque : On peut abréger le code de cette fonction en se souvenant qu'un `return` interrompt la fonction quoi qu'il arrive.

Exercice 13. *Recherche d'un motif*
Écrire une fonction d'en-tête `rechercheMotif(mot, ch)` prenant en argument deux chaînes de caractères `mot` et `ch`, et indiquant si `mot` se trouve dans `ch`. Par exemple si `ch` est la chaîne `"tagada tsoin tsoin"`, alors `rechercheMotif("oin", ch)` devra renvoyer `True`, mais `rechercheMotif("tagda", ch)` devra renvoyer `False`.

Remarque : Il existe en fait déjà une instruction Python qui fait cela : l'instruction `mot in ch` (similaire au test d'appartenance décrit plus haut). En revanche, il n'en existe pas pour les listes, et on doit alors adapter l'algorithme de l'exercice au cas des listes.

4 Exercices complémentaires

Exercice 14. Donner les résultats des deux scripts suivants (répondre sans utiliser l'ordinateur, puis vérifier).
Rappel : L'instruction `%` *a été définie au TP1.*

Script 1

```
ch = ''
for k in range(1, 5):
    if k % 2 == 1:
        ch += 'a'
    else:
        ch += 'b'
print(ch)
```

Script 2

```
ch = ''
for k in range(0, 4):
    if k % 2 == 1:
        ch += 'a'
    else:
        ch += 'b'
    print(ch)
```

Exercice 15.

1. Écrire une fonction d'en-tête `croissante(L)` prenant en argument une liste d'entiers et renvoyant `True` si ces entiers sont rangés dans l'ordre croissant, et `False` sinon.
2. Utiliser la fonction précédente pour écrire une fonction d'en-tête `monotone(L)` renvoyant `True` si la liste passée en argument est monotone (c'est-à-dire croissante ou décroissante) et `False` sinon. Cette fonction appellera la fonction `croissante`, et de préférence on ne fera *pas* appel à une fonction analogue pour la décroissance.

Exercice 16. *Palindromes*
On rappelle qu'un palindrome est un mot qui reste identique si on le lit à l'envers, comme « kayak ».

1. Écrire une fonction d'en-tête `palindrome(ch)` prenant en argument une chaîne de caractères et renvoyant `True` si cette chaîne est un palindrome, et `False` sinon. Cette fonction pourra appeler une fonction auxiliaire renvoyant le « miroir » d'un mot pris en argument, et que l'on aura codée préalablement.
2. On veut modifier la fonction précédente pour qu'elle ne tienne pas compte des espaces et considère que la chaîne `"elu par cette crapule"` est aussi un palindrome. Écrire une telle fonction d'en-tête `palindrome2(ch)`. On pourra pour cela s'appuyer sur une fonction auxiliaire permettant de supprimer tous les espaces d'une chaîne prise en argument.