# Chaînes de caractères et listes



## 1. Les Chaînes de caractères

Une chaîne de caractère est un objet python, de type **str** (**string**), qui permet de gérer du texte.

**Délimitation**

```
1  >>>'Informatique'
2  Informatique
3  >>> 'Que j'aime à faire apprendre un nombre utile aux sages/cv' /cv
4  SyntaxError: invalid syntax
5  >>> "Que j'aime à faire aprendre un nombre utile aux sages"
6  /r"Que j'aime à faire aprendre un nombre utile aux sages"/r
7  >>> 'Que j\'aime à faire apprendre un nombre utile aux sages'
8  "Que j'aime à faire apprendre un nombre utile aux sages"
9  >>> 'Il m\'a dit : "oui"'
10 'Il m\'a dit : "oui"'
11 >>> """Il m'a dit : "oui" """
12 'Il m\'a dit : "oui" '
13 >>> print("""Il m'a dit : "oui" """)
14 Il m'a dit : "oui"
15 >>> n=10
16 >>> print("J'ai dépensé",n,"euros.")
17 J'ai dépensé 10 euros.
18 >>> print('')
19
20 >>>
```

1 : les chaînes de caractères sont délimitées par des apostrophes.

3, 5 et 7 : si la chaîne contient des apostrophes, elle peut être délimitée par des guillemets ou l'apostrophe remplacée par le **caractère d'échappement** `\'`.

11 : l'emploi de triples guillemets permet d'éviter ces problèmes.

13 : la fonction `print` affiche les valeurs passées en arguments.

18 : chaîne vide.

Les caractères à l'intérieur d'une chaîne sont numérotés en commençant par 0. On peut également y accéder en commençant par la fin, les indices sont alors des entiers négatifs.

| $-7$ | $-6$ | $-5$ | $-4$ | $-3$ | $-2$ | $-1$ |
|---|---|---|---|---|---|---|
| B | o | n | j | o | u | r |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

**Extraction d'une sous-chaîne**

```
1  >>> mot='Bonjour'
2  >>> print(mot[0],mot[2],mot[6])
3  B n r
4  >>> print(mot[1:6])
5  onjou
6  >>> print(mot[1:7])
7  onjour
8  >>> print(mot[0:7:2])
9  Bnor
10 >>> print(mot[::2])
11 Bnor
12 >>> print(mot[::-1])
13 ruojnoB
14 >>> print(mot[-2:7])
15 ur
16 >>> mot[2]='w'
17 Traceback (most recent call last):
18   File "<pyshell#104>", line 1, in <module>
19     mot[2]='w'
20 TypeError: 'str' object does not support item assignment
```

2 : on extrait un caractère particulier en donnant son indice entre crochets.

4 et 6 : on extrait une sous-chaîne en donnant l'indice du premier caractère et **l'indice suivant l'indice du dernier caractère** à extraire.

8 : extrait un caractère sur deux.

10 : si les premiers indices sont omis, ils prennent respectivement la valeur minimale et la valeur maximale (la pas par défaut est 1).

12 : avec un pas de $-1$, on obtient la chaîne renversée.

14 : extrait la sous-chaîne commençant par le caractère d'indice $-2$, à savoir « u », et finissant par le caractère d'indice 6, à savoir « r ».

16 : les chaînes de caractères ne sont pas modifiables.

La fonction `len` renvoie la longueur d'une chaîne de caractères (son nombre de caractères).

```
1  >>> len('Bonjour')
2  7
3  >>> len('a b c d')
4  7
```

Voici deux façons de parcourir une chaîne de caractères.

**Parcours d'une chaîne de caractères**

```
1  >>> mot='Bonjour'
2  >>> for i in range(len(mot)):
3          print(mot[i])
4
5  B
6  o
7  n
8  j
9  o
10 u
11 r
12 >>> for i in range(len(mot)):
13         print(mot[i],end='')
14
15 Bonjour
16 >>> for c in mot:
17         print(c,end='*')
18
19 B*o*n*j*o*u*r*
```

2 : le compteur représente l'indice par lequel on accède à chaque caractère. Ici, comme la chaîne a pour longueur 7, il prend les valeurs de 0 à 6.

13 : par défaut, chaque utilisation de `print` provoque un retour à la ligne (semblable au **caractère d'échappement** `\n`). En procédant ainsi, on affiche les résultats les uns à la suite des autres.

16 : avec cette méthode, on parcourt directement les caractères de la chaîne.

17 : les résultats seront séparés par des « * ».

On termine par quelques opérations sur les chaînes de caractères, dont la **concaténation** qui consiste à juxtaposer une ou plusieurs chaînes.

```
1  >>> 'Bon'+'jour'
2  'Bonjour'
3  >>> 'o' in 'Bonjour'
4  True
5  >>> 'v' in 'Bonjour'
6  False
7  >>> 'v' not in 'Bonjour'
8  True
9  >>> 'B'*4
10 'BBBB'
```

1 : concaténation de deux chaînes.

3 et 5 : teste l'appartenance des caractères « o » et « v » à la chaîne « Bonjour».

7 : teste la non appartenance du caractère « v » à la chaîne « Bonjour».

9 : répète la chaîne (correspond à la concaténation de « B », « B », « B » et « B »).

**Quelques fonctions et méthodes pour la classe string**

```
1  >>> int('18')
2  18
3  >>> a=int(input('Entrez votre âge : '))
4  Entrez votre âge : 25
5  >>> a
6  25
7  >>> str(27)
8  '27'
9  >>> 'Bonjour'.upper()
10 'BONJOUR'
11 >>> 'BONJOUR'.isupper()
12 True
13 >>> 'Bonjour'.find('o')
14 1
15 >>> 'Bonjour'.replace('o','z')
16 'Bznjzur'
17 >>> 'Bonjour'.count('o')
18 2
```

1 : convertit une chaîne de caractères en entier.

3 et 5 : en associant `int` et `input`, on peut saisir un entier (la fonction `float` permet d'obtenir un type **flottant**).

7 : convertit un entier en chaîne de caractères.

9 : retourne une chaîne dont tous les caractères sont en majuscule. La méthode `lower` agit de même, mais en minuscule.

11 : teste si la chaîne est en majuscules. La méthode `islower` fait de même avec les minuscules.

13 : retourne la première occurrence d'une sous-chaîne dans une chaîne.

15 : remplace toutes les occurrences d'une sous-chaîne par une autre.

17 : compte le nombre d'occurrences d'une sous-chaîne.

La classe string dispose de nombreuses autres méthodes que l'on trouvera à l'adresse suivante :
https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

1. Écrire une fonction `miroir(chaine)` qui renvoie l'inverse de la chaîne `chaine` (sans avoir recours au pas négatif).
2. Écrire une fonction `remplaceLettre(chaine,l1,l2)` qui renvoie la chaîne `chaine` dans laquelle toutes les lettres `l1` sont remplacées par la lettre `l2`. Les lettres en majuscule (resp. miniscule) sont remplacées par des majuscules (resp. minuscules), comme dans l'exemple ci-dessous.

```
1 >>> remplaceLettre('Caractère','c','v')
2 'Varavtère'
3 >>> remplaceLettre('Caractère','C','v')
4 'Varavtère'
5 >>> remplaceLettre('Caractère','c','V')
6 'Varavtère'
7 >>> remplaceLettre('Caractère','C','V')
8 'Varavtère'
```

3. Le but de cette question est d'écrire une fonction permettant le décryptage brutal d'un texte chiffré avec le code César : étant donné un texte ainsi chiffré, on affichera le texte déchiffré avec tous les décalages possibles (il y en a 26) et on recherchera ceux ayant du sens. On n'utilisera que les majuscules (`A, B,..., Z`) et ni espace ni apostrophe.
   Les fonctions `ord()` et `chr()` pourront être utiles.

```
1 >>> ord('A')
2 65
3 >>> ord('Z')
4 90
5 >>> chr(72)
6 'H'
```

   1 et 3 : la fonction `ord()` retourne le numéro du caractère dans la table Unicode.

   5 : `chr()` est la fonction réciproque.

   On écrira deux fonctions :
   - `dechiffrementCesar(texte,decalage)` qui retourne le texte `texte` déchiffré avec le décalage `decalage`, entier positif ;
   - `decryptageBrutal(texte)` qui affiche le texte déchiffré avec tous les décalages possibles.

   Décrypter ensuite le message suivant :
   « `HLFEKUVGRIKZTLCZVICVJTYVMVLOUVJDRKYVDRKZTZVEJZCJFEKUVJIRTZEVJTRIVVJ` ».

## 2. Les listes

Une liste peut-être vue comme un tableau constitué d'une seule ligne et d'un certain nombre de colonnes numérotées (en commençant par 0).

**Comportement d'une liste**

```
1  >>> liste=[12,'Bonjour',[1,2]]
2  >>> len(liste)
3  3
4  >>> liste[0]
5  12
6  >>> liste[1]
7  'Bonjour'
8  >>> liste[1].upper()
9  'BONJOUR'
10 >>> liste[2]
11 [1, 2]
12 >>> liste[2][1]
13 2
14 >>> liste[0]=1
15 >>> liste
16 [1, 'Bonjour', [1, 2]]
17 >>> type(liste)
18 <class 'list'>
```

1 : on définit une liste en utilisant des crochets. Les éléments sont séparés par des virgules. La liste `[]` est la liste vide.

2 : la liste `liste` comporte 3 éléments (3 colonnes).

4 : on accède un élément particulier en indiquant son rang. Ce premier élément est un entier.

6 et 8 : le deuxième élément est une chaîne de caractère, et se comporte comme telle (on peut lui appliquer toutes les méthodes des chaînes de caractères).

10 et 12 : le dernier élément est une liste de deux éléments, et se comporte comme telle. Une liste de listes permet de créer, par exemple, un tableau à deux dimensions, avec des lignes et des colonnes : la première liste de la liste principale est la première ligne, le deuxième liste est la deuxième liste, etc.

14 : on peut modifier ainsi un élément particulier.

**Ajout d'éléments à une liste**

```
1  >>> liste=[]
2  >>> liste.append(5)
3  >>> liste
4  [5]
5  >>> liste.append([4,3])
6  >>> liste
7  [5, [4, 3]]
8  >>> liste=[5]
9  >>> liste.extend([4,3])
10 >>> liste
11 [5, 4, 3]
12 >>> [5]+[4,3]
13 [5, 4, 3]
```

1-7 : la méthode `.append()` permet d'ajouter un élément en bout de liste. Finalement, `liste` contient deux éléments.

8-10 : la méthode `.extend()` ajoute en bout de liste tous les éléments de la liste passée en argument ; `liste` a alors trois éléments.

Dans les exemples qui suivent, on crée de deux façons la liste des carrés des entiers de 1 à 5.

**Création d'une liste**

```
1  >>> liste1=[]
2  >>> for i in range(1,6):
3          liste1.append(i**2)
4
5
6  >>> liste1
7  [1, 4, 9, 16, 25]
8  >>> liste2=[i**2 for i in range(1,6)]
9  >>> liste2
10 [1, 4, 9, 16, 25]
```

1-7 : première méthode : on part d'une liste vide à laquelle on ajoute un par un les éléments nécessaires en utilisant une boucle `for`.

8-10 : deuxième méthode, la compréhension de liste.

**Quelques méthodes et fonctions pour les listes**

```
1  >>> liste=[5,4,3,4]
2  >>> liste.remove(4)
3  >>> liste
4  [5, 3, 4]
5  >>> liste.insert(4,1)
6  >>> liste
7  [5, 3, 4, 1]
8  >>> liste=[5,4,3,4]
9  >>> liste.remove(4)
10 >>> liste.insert(1,4)
11 >>> liste
12 [5, 4, 3, 4]
13 >>> liste.pop(3)
14 4
15 >>> liste
16 [5, 4, 3]
17 >>> max(liste)
18 5
19 >>> liste=['B', 'o', 'n', 'j', 'o', 'u', 'r']
20 >>> '*'.join(liste)
21 'B*o*n*j*o*u*r'
22 >>> ''.join(liste)
23 'Bonjour'
24 >>> liste=list('Bonjour')
25 >>> liste
26 ['B', 'o', 'n', 'j', 'o', 'u', 'r']
```

2 : supprime la première occurrence de l'argument dans la liste. Si l'argument n'est pas présent dans la liste, une erreur se produit.

5 : insert un élément à un rang donné (rang puis élément).

13 : supprime un élément de rang donné.

17 : retourne le maximum de la liste.

20 et 22 : concatène les éléments d'une liste constituée uniquement de chaînes de caractères, en les séparant, dans ce cas, par `'*'`. Le séparateur peut-être vide.

24 : crée une liste dont les éléments sont les caractères de la chaîne de caractères passée en argument.

D'autres méthodes de la classe list à l'adresse suivante.

https://docs.python.org/3/tutorial/datastructures.html

Les listes sont des pointeurs (ou des alias) : elles pointent vers une adresse mémoire.

**Les listes sont des alias**

```
1  >>> liste=[5,4,3]
2  >>> liste1=[5,4,3]
3  >>> liste2=liste1
4  >>> liste1[0]=7
5  >>> liste1
6  [7, 4, 3]
7  >>> liste2
8  [7, 4, 3]
9  >>> liste3=liste1[:]
10 >>> liste1[0]=9
11 >>> liste1
12 [9, 4, 3]
13 >>> liste3
14 [7, 4, 3]
```

3 : on pense créer une copie de la liste 1.

4-8 : mais la modification d'un élément de la liste originale modifie aussi l'élément dans la « copie » : les deux noms pointent vers la même adresse mémoire.

9-14 : on obtient ainsi une copie indépendante.

## 3. Conversion de nombres avec bases de 2 à 16

On considère ici les nombres comme des chaînes de caractère, puisqu'à partir de la base 11 on utilise des lettres pour les chiffres supplémentaires. Chaque « chiffre » dans une base est donc un caractère
On définit la liste `Chiffre` avec laquelle on peut associer nombres et caractères de la manière suivante :

```
>>> Chiffre=['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
>>> Chiffre[12]
'C'
>>> Chiffre.index('A')
10
```

On utilisera cette liste dans la suite.

1. Définir des fonctions :
   (a) `baseDecimal(nBase,base)` : retourne l'écriture en base 10 d'un nombre (de type `str`) donné en base `base`.
   (b) `decimalBase(nDec,base)` : retourne l'écriture en base `base` d'un nombre donné en base 10.
   (c) `baseBase(n,baseInitiale,baseFinale)` : retourne l'écriture en base `baseFinale` d'un nombre `n` donné en base `baseInitiale`.

   On pourra vérifier que l'on obtient bien les résultats suivants :

   ```
   >>> baseDecimal('FF',16)
   255
   >>> decimalBase(12364,15)
   '39E4'
   >>> baseBase('15012',6,13)
   '1115'
   ```

2. Définir une fonction `Conversion()` qui :
   - Demande un nombre, une base initiale et une base finale ;
   - Calcule puis affiche la conversion demandée ;
   - Répète les points précédents tant que l'on ne répond pas `'n'` à la question « Voulez-vous continuer ? (O/n) ».
3. Améliorer la fonction précédente pour faire en sorte que le programme ne s'arrête pas en cas de réponse inattendue ou incohérente de l'utilisateur.
   En fonction des idées pour faire cela, il sera sans doute nécessaire de faire des recherches dans la documentation de python.