Chapitre 11 : Les chaines des caractères

I. Introduction :

Une chaîne de caractères est une suite ordonnée de caractères.

En algorithmique, une chaîne de caractères est considérée comme un tableau de caractères.

En Python, les chaînes de caractères sont classées dans les séquences tout comme les listes et les tuples **(les tuples et les chaînes sont des séquences immuables, c.à.d. qu'elles ne peuvent pas être modifiées)**.

Et on a vu que pour déclarer une chaine on doit utiliser les guillemets ou les simples quotes. On a vu aussi comment on peut les manipuler en les concaténant ensemble ou les parcourir avec la boucle for, mais en fait avec les chaines (string) on n'arrête pas ici mais il y a plein de méthodes avec lesquelles on peut manipuler les chaines

II. Code ASCII :

Dans un système informatique, à chaque caractère est associé une valeur numérique : son code ASCII (American Standard Code for Information Interchange). L'ensemble des codes est recensé dans une table nommée "table des codes ASCII". Quand on stocke un caractère en mémoire (dans une variable), on mémorise en réalité son code ASCII. Un code ASCII est codé sur un octet (huit bits).

- **Comparaison des caractères :**

Voici un exemple de comparaison de caractères :

"B" > "A" car le code ASCII du caractère "B" (66 en base 10) est supérieur au code ASCII du caractère "A" (65 en base 10).

- Pour comparer deux chaînes de caractères, on compare les caractères de même rang dans les deux chaînes en commençant par le premier caractère de chaque chaîne (le premier caractère de la première chaîne

est comparé au premier caractère de la seconde chaîne, le deuxième caractère de la première chaîne est comparé au deuxième caractère de la seconde chaîne, et ainsi de suite…).

Exemples : Comparaison de deux chaînes

- **"baobab" < "sapin"** car le code ASCII de "b" est inférieur au code ASCII de "s".
- **"baobab" > "banania"** car le code ASCII de "o" est supérieur au code ASCII de "n" (la comparaison ne peut pas se faire sur les deux premiers caractères car ils sont identiques).
- **"1999" > "1998"** car le code ASCII de "9" est supérieur au code ASCII de "8" (la comparaison ne peut pas se faire sur les trois premiers caractères car ils sont identiques). Attention, ici ce ne sont pas des valeurs numériques qui sont comparées, mais bien des caractères.
- **"333" > "1230"** car le code ASCII de "3" est supérieur au code ASCII de "1".
- **"333" < "3330"** car la seconde chaîne a une longueur supérieure à celle de la première (la comparaison ne peut pas se faire sur les trois premiers caractères car ils sont identiques).
- **"Baobab" < "baobab"** car le code ASCII de "b" est supérieur au code ASCII de "B".

III. La manipulation des chaines de caractère en algorithme :

- La fonction **SSCHAINE (chaine , position , nombre) : chaine**

Le rôle de la fonction SSCHAINE() est de retourner une "sous-chaîne" (copie d'un extrait de la chaîne passée en premier paramètre) de nombre caractères de la chaîne chaîne à partir du caractère qui se trouve en position **position**

- La longueur de la chaine : la fonction **LONGUEUR(chaine) :entier**

La fonction LONGUEUR() a pour rôle de calculer et de retourner le nombre de caractères présents dans la chaîne de caractères passée en paramètre.

- La fonction **RANG(chaine , sous-chaine , position) : entier**

La fonction RANG() recherche et retourne la première occurrence de la sous-chaîne dans la chaîne. La recherche commence à partir de position. Cette fonction retourne la position du premier caractère de la sous-chaîne, ou la valeur -1 si la "sous-chaîne" n'existe pas dans la chaîne (à partir de position).

- La fonction **CODE(caractere) : entier**

La fonction CODE() retourne le code ASCII du caractère passé en paramètre.

- La fonction **CAR(code_ascii) :caractere**

La fonction CAR() retourne le caractère dont le code ASCII est passé en paramètre.

- La fonction **MINUSCULE(chaine) : chaine**

Retourne la même chaine passée en paramètre en minuscule

- La fonction **MAJUSCULE(chaine) : chaine**

Retourne la même chaine passée en paramètre en majuscule

IV. La manipulation des chaines de caractère en python :

Les chaînes de caractères peuvent être considérées comme des listes (de caractères) un peu particulières :

```
>>> animaux = "girafe tigre"
>>> animaux
'girafe tigre'
>>> len(animaux)
12
>>> animaux[3]
'a'
```

Nous pouvons donc utiliser certaines propriétés des listes comme les tranches :

```
>>> animaux = "girafe tigre"
>>> animaux[0:4]
'gira'
>>> animaux[9:]
'gre'
>>> animaux[:-2]
'girafe tig'
>>> animaux[1:-2:2]
'iaetg'
```

Mais contrairement aux listes, les chaînes de caractères présentent toutefois une différence notable, ce sont **des listes non modifiables**. Une fois une chaîne de caractères définie, vous ne pouvez plus modifier un de ses éléments. Le cas échéant, Python renvoie un message d'erreur :

```
>>> animaux = "girafe tigre"
>>> animaux[4]
'f'
>>> animaux[4] = "F"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Par conséquent, si vous voulez modifier une chaîne de caractères, vous devez en construire une nouvelle. Pour cela, n'oubliez pas que les opérateurs de concaténation (+) et de duplication (*) (introduits dans le chapitre 2 *Variables*) peuvent vous aider. Vous pouvez également générer une liste, qui elle est modifiable, puis revenir à une chaîne de caractères (voir plus bas).

V. Caractères spéciaux :

Il existe certains caractères spéciaux comme \n que nous avons déjà vu (pour le retour à la ligne). Le caractère \t produit une tabulation. Si vous voulez écrire des guillemets simples ou doubles et que ceux-ci ne soient pas confondus avec les guillemets de déclaration de la chaîne de caractères, vous pouvez utiliser \' ou \".

```
>>> print("Un retour à la ligne\npuis une tabulation\t puis un guillemet\"")
Un retour à la ligne
puis une tabulation     puis un guillemet"
>>> print('J\'affiche un guillemet simple')
J'affiche un guillemet simple
```

Vous pouvez aussi utiliser astucieusement des guillemets doubles ou simples pour déclarer votre chaîne de caractères :

```
>>> print("Un brin d'ADN")
Un brin d'ADN
>>> print('Python est un "super" langage de programmation')
Python est un "super" langage de programmation
```

Quand on souhaite écrire un texte sur plusieurs lignes, il est très commode d'utiliser les guillemets triples qui conservent le formatage (notamment les retours à la ligne) :

```
>>> x = """souris
... chat
... abeille"""
>>> x
'souris\nchat\nabeille'
>>> print(x)
souris
chat
abeille
```

VI. Méthodes associées aux chaînes de caractères

Voici quelques méthodes spécifiques aux objets de type str :

```
>>> x = "girafe"
>>> x.upper()
'GIRAFE'
>>> x
'girafe'
>>> 'TIGRE'.lower()
'tigre'
```

- Les méthodes **.lower()** et **.upper()** renvoient un texte en minuscule et en majuscule respectivement. On remarque que l'utilisation de ces méthodes n'altère pas la chaîne de caractères de départ mais renvoie une chaîne de caractères transformée.

Pour mettre en majuscule la première lettre seulement, vous pouvez faire :

```
>>> x[0].upper() + x[1:]
'Girafe'
```

- Les méthodes **.isupper()** et **.islower()** retournent True si la chaine est toute en majuscule ou minuscule
- La méthode **.capitalize()** plus simplement utiliser la méthode adéquate :

```
>>> x.capitalize()
'Girafe'
```

- la méthode **.split()** Il existe une méthode associée aux chaînes de caractères qui est particulièrement pratique :

```
>>> animaux = "girafe tigre singe souris"
>>> animaux.split()
['girafe', 'tigre', 'singe', 'souris']
>>> for animal in animaux.split():
...     print(animal)
...
girafe
tigre
singe
souris
```

La méthode **.split()** découpe une chaîne de caractères en plusieurs éléments appelés *champs*, en utilisant comme séparateur n'importe quelle combinaison « d'espace(s) blanc(s) ».

Un espace blanc (*whitespace* en anglais) correspond aux caractères qui sont invisibles à l'œil, mais qui occupent de l'espace dans un texte. Les espaces blancs les plus classiques sont l'espace, la tabulation et le retour à la ligne.

Il est possible de modifier le séparateur de champs, par exemple :

```
>>> animaux = "girafe:tigre:singe::souris"
>>> animaux.split(":")
['girafe', 'tigre', 'singe', '', 'souris']
```

Attention, dans cet exemple, le séparateur est un seul caractère « : » (et non pas une combinaison de un ou plusieurs :) conduisant ainsi à une chaîne vide entre singe et souris.

Il est également intéressant d'indiquer à .split() le nombre de fois qu'on souhaite découper la chaîne de caractères avec l'argument maxsplit :

```
>>> animaux = "girafe tigre singe souris"
>>> animaux.split(maxsplit=1)
['girafe', 'tigre singe souris']
>>> animaux.split(maxsplit=2)
['girafe', 'tigre', 'singe souris']
```

- La méthode **.find(sous-chaine , position)**, quant à elle, recherche une chaîne de caractères passée en argument :

```
>>> animal = "girafe"
>>> animal.find("i")
1
>>> animal.find("afe")
3
>>> animal.find("z")
-1
>>> animal.find("tig")
-1
```

Si l'élément recherché est trouvé, alors l'indice du début de l'élément dans la chaîne de caractères est renvoyé. Si l'élément n'est pas trouvé, alors la valeur -1 est renvoyée.

Si l'élément recherché est trouvé plusieurs fois, seul l'indice de la première occurrence est renvoyé :

```
>>> animaux = "girafe tigre"
>>> animaux.find("i")
1
```

- La méthode **.index(sous-chaine , position) :** recherche une sous chaine et renvoie sa position si la sous chaine est trouvé sinon génère une exception (ValueError)

- La fonction **ord(caractere) :** renvoie le code ascii d'un caractère passé en paramètre .
- La fonction **chr(code_ascii)** : renvoie le caractère correspondant au code ascii passé en paramètre
- la méthode **.replace(ancienne_chaine , nouvelle_chaine)** qui substitue une chaîne de caractères par une autre :

```
>>> animaux = "girafe tigre"
>>> animaux.replace("tigre", "singe")
'girafe singe'
>>> animaux.replace("i", "o")
'gorafe togre'
```

- La méthode **.count(chaine)** compte le nombre d'occurrences d'une chaîne de caractères passée en argument :

```
>>> animaux = "girafe tigre"
>>> animaux.count("i")
2
>>> animaux.count("z")
0
>>> animaux.count("tigre")
1
```

- La méthode **.startswith(), .endswith()** vérifie si une chaîne de caractères commence par une autre chaîne de caractères :

```
>>> chaine = "Bonjour monsieur le capitaine !"
>>> chaine.startswith("Bonjour")
True
>>> chaine.startswith("Au revoir")
False
```

Cette méthode est particulièrement utile lorsqu'on lit un fichier et que l'on veut récupérer certaines lignes commençant par un mot-clé. Par exemple dans un fichier PDB, les lignes contenant les coordonnées des atomes commencent par le mot-clé ATOM.

- la méthode **.strip()** permet de « nettoyer les bords » d'une chaîne de caractères :

```
>>> chaine = "  Comment enlever les espaces au début et à la fin ?
>>> chaine.strip()
'Comment enlever les espaces au début et à la fin ?'
```

La méthode **.strip()** enlève les espaces situés sur les bords de la chaîne de caractère mais pas ceux situés entre des caractères visibles. En réalité, cette méthode enlève n'importe quel combinaison « d'espace(s) blanc(s) » sur les bords, par exemple :

```
>>> chaine = "  \tfonctionne avec les tabulations et les retours à la ligne\n"
>>> chaine.strip()
'fonctionne avec les tabulations et les retours à la ligne'
```

La méthode .strip() est très pratique quand on lit un fichier et qu'on veut se débarrasser des retours à la ligne.

VII. Extraction de valeurs numériques d'une chaîne de caractères :

Une tâche courante en Python est de lire une chaîne de caractères (provenant par exemple d'un fichier), d'extraire des valeurs de cette chaîne de caractères puis ensuite de les manipuler.

On considère par exemple la chaîne de caractères val :

```
>>> val = "3.4 17.2 atom"
```

On souhaite extraire les valeurs 3.4 et 17.2 pour ensuite les additionner.

Dans un premier temps, on découpe la chaîne de caractères avec la méthode .split() :

```
>>> val2 = val.split()
>>> val2
['3.4', '17.2', 'atom']
```

On obtient alors une liste de chaînes de caractères. On transforme ensuite les deux premiers éléments de cette liste en *floats* (avec la fonction float()) pour pouvoir les additionner :

```
>>> float(val2[0]) + float(val2[1])
20.599999999999998
```

Remarque : Retenez bien l'utilisation des instructions précédentes pour extraire des valeurs numériques d'une chaîne de caractères. Elles sont régulièrement employées pour analyser des données depuis un fichier.

VIII. Conversion d'une liste de chaînes de caractères en une chaîne de caractères :

On a vu dans le chapitre 2 *Variables* la conversion d'un type simple (entier, *float* et chaîne de caractères) en un autre avec les fonctions int(), float() et str(). La conversion d'une liste de chaînes de caractères en une chaîne de caractères est un peu particulière puisqu'elle fait appelle à la méthode **.join().**

```
>>> seq = ["A", "T", "G", "A", "T"]
>>> seq
['A', 'T', 'G', 'A', 'T']
>>> "-".join(seq)
'A-T-G-A-T'
>>> " ".join(seq)
'A T G A T'
>>> "".join(seq)
'ATGAT'
```

Les éléments de la liste initiale sont concaténés les uns à la suite des autres et intercalés par un séparateur qui peut être n'importe quelle chaîne de caractères. Ici, on a utilisé un tiret, un espace et rien (une chaîne de caractères vide).

Attention, la méthode .join() ne s'applique qu'à une liste de chaînes de caractères.

```
>>> maliste = ["A", 5, "G"]
>>> " ".join(maliste)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 1: expected string, int found
```

On espère qu'après ce petit tour d'horizon vous serez convaincu de la richesse des méthodes associées aux chaînes de caractères. Pour avoir une liste exhaustive de l'ensemble des méthodes associées à une variable particulière, vous pouvez utiliser la fonction dir().

```
>>> animaux = "girafe tigre"
>>> dir(animaux)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '_
_getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '_
_init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mo
d__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__'
, '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'for
mat_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'i
sidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'is
title', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Pour l'instant, vous pouvez ignorer les méthodes qui commencent et qui se terminent par deux tirets bas (*underscores*) __.

Vous pouvez également accéder à l'aide et à la documentation d'une méthode particulière avec help(), par exemple pour la méthode .split() :

```
>>> help(animaux.split)
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator.
(END)
```

Attention à ne pas mettre les parenthèses à la suite du nom de la méthode. L'instruction correcte est help(animaux.split) et non pas help(animaux.split())

IX. Exercices :

Exercice 1 : Ecrivez un algorithme puis un programme en python qui permet de convertir une chaine saisis par l'utilisateur en majuscule

Exercice 2 : Ecrivez un algorithme puis un programme en python qui compte le nombre occurrences d'une lettre donnée dans cette chaîne.

Exercice 3 : Ecrivez un algorithme puis un programme en python qui compte le nombre occurrences des lettres et nombres.

Exercice 4 : Ecrivez un algorithme puis un programme en python qui permet de calculer le nombre de répétition de chaque caractère de la chaine.

Exercice 5 : Ecrivez un algorithme puis un programme en python qui permet de calculer le nombre de voyelles dans une chaine de caractères.

Exercice 6 : Ecrivez un algorithme puis un programme en python qui supprime toutes les lettres « e » (minuscules) d'un texte de moins d'une ligne (supposée ne pas dépasser 132 caractères) fourni au clavier.

Exercice 7 : Ecrivez un algorithme puis un programme en python qui supprime toutes les voyelles d'un texte de moins d'une ligne fourni au clavier.

Exercice 8 : Ecrivez un algorithme puis un programme en python qui permet de supprimer les doublons dans une chaine de caractère saisis par l'utilisateur.

Exercice 9 : Soit la liste ['girafe', 'tigre', 'singe', 'souris'].

1. Affichez chaque élément ainsi que sa taille (nombre de caractères).
2. Affichez pour chaque élément le nombre des voyelles et le nombre des consonnes.