CHAPITRE 13

Les chaînes de caractères

I. Les chaînes de caractères en Python

Les chaînes de caractères permettent de manipuler des mots, des phrases, des messages.

En Python, une chaîne de caractères se note entre doubles quotes ou entre simples quotes.

Exemple 1

```
>>> ma_chaine = "je suis un oeuf"
>>> ma_chaine
'je suis un oeuf'
```

Le type d'une chaîne de caractère est `str` (pour string).

Exemple 2

```
>>> ma_chaine = "je suis un oeuf"
>>> type(ma_chaine)
str
```

Remarque

Une chaîne de caractères est immuable (ou immutable). Elle ne peut pas être modifiée.

II. Opération de base sur les chaînes de caractères

II. 1. Nombre de caratères dans une chaîne

On peut déterminer le nombre de caractères d'une chaîne, en faisant appel à la fonction intégrée `len()` :

Exemple 3

```
>>> ma_chaine = "je suis un oeuf"
>>> len(ma_chaine)
15
```

II. 2. Concaténation de deux chaînes

L'opération d'assemblage de plusieurs chaînes de caractères pour en construire une plus grande s'appelle la concaténation. Elle se réalise avec l'opérateur +

Exemple 4

```
>>> ma_chaine = "je suis un oeuf"
>>> ma_suite = "mais pas un boeuf"
>>> ma_chaine + ma_suite
'je suis un oeuf mais pas un boeuf'
```

Activité 1.
Écrivez une fonction `pluriel()`, prenant *chaîne* en argument, et renvoyant le pluriel de *chaîne* en ajoutant un 's' à la fin.

II. 3. Égalité de deux chaînes de caractères

Deux chaînes de caractères sont égales si elles ont la même longueur et les mêmes caractères exactement dans le même ordre.

Exemple 5

```
>>> "abc" == "abc"
True
>>> "abc" == "bca"
False
```

II. 4. Ordre de deux chaînes de caractères

Il existe une relation d'ordre pour les chaînes de caractères : l'ordre lexicographique. C'est une extension de l'ordre alphabétique. Par exemple, l'expression booléenne "bonjour" $<$ "bonsoir" est vraie.

Exemple 6

```
>>> "bonjour" < "bonsoir"
False
>>> "coucou" < "poupou"
True
```

Activité 2.
Écrivez une fonction `comparaison()`, prenant *chaine1* et *chaine2*, deux chaines de caractères en argument, et renvoyant `"chaînes égales"` si *chaine1* et *chaine2* sont égales, et la chaîne la plus "petite" dans l'ordre lexicographique si les chaînes sont différentes.
Par exemple : `comparaison("toto", "tata")` renverra "tata".

II. 5. Test d'appartenance

On peut tester l'appartenance d'un caractère dans une chaîne de caractères avec le mot clé `in`. Le test renvoie un booléen.

Exemple 7

```
>>> 'e' in 'bebe'
True
>>> 'a' in 'bebe'
False

>>> mot = 'truc'
>>> lettre = 'u'
>>> lettre in mot
True
```

De la même façon, on peut tester l'appartenance d'une chaîne de caractères à une autre chaîne de caractères.

Exemple 8

```
>>> 'be' in 'bebe'
True
>>> 'ba' in 'bebe'
False

>>> mot = 'truc'
>>> lettre = 'uc'
>>> lettre in mot
True
```

III. Parcours d'une chaîne de caractères

III. 1. Indexation d'une chaîne de caractères

Une chaîne de caractères est une séquence **ordonnée** de caractères. On peut s'y référer par les indices de ces caractères. Ainsi, chaque caractère est accessible directement par son index (ou indice). L'index est mis entre crochets

.

Remarque

Le premier caractère (celui qui est le plus à gauche) a pour index 0.

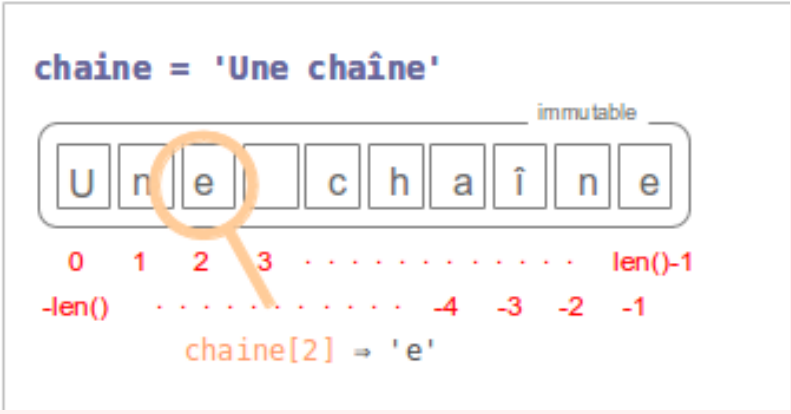

FIGURE 13.1 – Indexation dans une chaîne de caractères

L'indice peut être négatif. Dans ce cas il désigne un caractère repéré depuis la fin de la chaîne. Ainsi, `chaine[-1]` renvoie le dernier caractère de la chaîne.

Exemple 9

```
>>> ma_chaine = "je suis un oeuf"
>>> ma_chaine[14]
'f'
>>> ma_chaine[-1]
'f'
```

Activité 3.

On considère la chaîne suivante : `chaine = "couroucoucouroploplo"`.

1. Quel est l'index du premier 'r' ?
2. Que renvoie l'instruction `chaine[5]` ?
3. Que renvoie l'instruction `chaine[-2]` ?

III. 2. Parcours par index

De nombreux problèmes sur les chaînes de caractères nécessitent de parcourir l'intégralité de la chaîne afin d'obtenir la réponse à un problème donné.

Pour parcourir une chaîne de caractères, on procède en deux étapes :

- ▷ on définit une nouvelle variable définie par la longueur de la chaîne ;
- ▷ on utilise une boucle bornée (boucle `for`) pour un index i variant entre 0 et la longueur de la chaîne (exclue).

La structure algorithmique est la suivante :

Algorithme : parcours(chaine)

Entrées : *chaine* : une chaîne de caractères

```
1 début
2 |  longueur ← nombre de caractères
3 |  pour 0 ⩽ i ⩽ longueur par pas de 1 faire
4 |  |  action
5 |  fin pour
6 fin
```

Exemple 10 (affichage d'une chaîne caractère par caractère)

```
1 ma_chaine = "je suis un oeuf"
2 longueur = len(ma_chaine)
3 for i in range(0, longueur):
4     print(ma_chaine[i])
```

Activité 4.

On considère la fonction `double()` définie par l'algorithme ci-dessous :

Fonction double(chaine)

```
/* fonction qui renvoie une chaîne de caractères, en doublant chacun des
   caractères                                                          */
/* double("sieste") renverra 'ssiieessttee'                            */
```

Entrées : *chaine* : une chaîne de caractères

```
1 début
2 |  longueur ← nombre de caractères
3 |  nouvelle_chaine ← chaîne vide
4 |  pour 0 ⩽ i ⩽ longueur par pas de 1 faire
5 |  |  nouvelle_chaine ← nouvelle_chaine + chaine[i] + chaine[i]
6 |  fin pour
7 |  renvoyer nouvelle_chaine
8 fin
```

Codez cette fonction en Python.

III. 3. Parcours par élément

Pour parcourir une chaîne de caractères, on peut aussi la parcourir élément par élément.

- ▷ on définit une nouvelle variable définie par la longueur de la chaîne ;
- ▷ on utilise une boucle bornée (boucle `for`) pour parcourir chaque caractère dans la chaîne.

La structure algorithmique est la suivante :

Algorithme : parcours(chaine)

Entrées : *chaine* : une chaîne de caractères

```
début
    pour chaque caractère dans chaine faire
        action
    fin pour
fin
```

Exemple 11 (affichage d'une chaîne caractère par caractère)

```
ma_chaine = "je suis un oeuf"
for car in ma_chaine
    print(car)
```

Activité 5.

On considère la fonction `double()` définie par l'algorithme ci-dessous :

Fonction double(chaine)

```
/* fonction qui renvoie une chaîne de caractères, en doublant chacun des
   caractères                                                          */
/* double("sieste") renverra 'ssiieessttee'                            */
```

Entrées : *chaine* : une chaîne de caractères

```
début
    longueur ← nombre de caractères
    nouvelle_chaine ← chaîne vide
    pour chaque caractère dans chaine faire
        nouvelle_chaine ← nouvelle_chaine + chaine[i] + chaine[i]
    fin pour
    renvoyer nouvelle_chaine
fin
```

Codez cette fonction en Python.

Je retiens

- ▷ Une chaîne de caractères est un type de données immuable (qui ne peut pas être modifié). En Python, le type d'une chaîne de caractères est `str`.
- ▷ La concaténation d'une chaîne de caractères permet d'assembler plusieurs chaînes ensemble.
- ▷ Une chaîne de caractères est indexée. Le caractère le plus à gauche possède l'index 0.
- ▷ Le parcours d'une chaîne de caractères se fait par une boucle bornée (boucle `Pour`). On peut utiliser un parcours par index ou un parcours par élément.

IV. Exercices

Remarque

Récupérez le fichier **exercices.py** et copiez le dans l'espace **Mes Documents** de votre ordinateur. Exécutez le fichier pour vous assurer de l'absence d'erreur.

Pour chaque fonction, vous vérifierez votre travail **en autonomie**. Pour cela :

1. exécutez en console l'instruction

   ```
   run_docstring_examples(ma_fonction, globals(), verbose = True)
   ```

2. En cas d'erreur, utilisez le site `http://pythontutor.com` pour débugger votre fonction.
3. Exécutez en console l'instruction `test_ma_fonction()` pour exécuter des tests supplémentaires (test cachés).

Exercice 1 (conjugaison).

1. On souhaite écrire une fonction `radical()` prenant *chaine* une chaîne de caractères en argument, qui renvoie le radical d'un verbe du premier groupe. Par exemple, l'exécution de `radical('chanter')` renvoie `'chant'`.

a. Prenez connaissance de la documentation de la fonction `radical()`. Quel est la nom et le type de l'argument ?

b. Élaborez deux préconditions (sur le type et sur la valeur). On rappelle que les verbes du premier groupe finissemnt par **er**.

c. Quel est le nom et le type de la variable renvoyée ?

d. Élaborez deux postconditions (sur le type et sur le nombre de caractères).

e. Complétez la fonction `radical()` en utilisant l'algorithme ci-dessous :

Fonction radical(chaine)

```
/* fonction qui renvoie une chaîne de caractères, en doublant chacun des
   caractères                                                          */
/* double("sieste") renverra 'ssiieessttee'                            */
```

Entrées : *chaine* : une chaîne de caractères
Sorties : *nouvelle_chaine* : une chaîne de caractères

```
1 début
2 |  longueur ← nombre de caractères
3 |  nouvelle_chaine ← chaîne vide
4 |  pour 0 ⩽ i ⩽ (longueur − 2) par pas de 1 faire
5 |  |  nouvelle_chaine ← nouvelle_chaine + chaine[i]
6 |  fin pour
7 |  renvoyer nouvelle_chaine
8 fin
```

Exécutez en console l'instruction `run_docstring_examples(radical, globals(), verbose = True)` pour vous assurer de son bon fonctionnement.
Exécutez en console l'instruction `test_radical()` pour exécuter des tests supplémentaires.

La suite page suivante

Suite de l'exercice 1.

2. Complétez la fonction `affiche_conjugaison()` prenant *chaine* une chaîne de caractères en argument, qui conjugue un verbe du premier groupe au présent. Par exemple, `affiche_conjugaison('chanter')` renvoie

```
je chante
tu chantes
il/elle chante
nous chantons
vous chantez
ils/elles chantent
```

On pourra utiliser la fonction `radical()`.
Testez votre fonction en conjuguant quelques verbes du premier groupe.

Exercice 2 (distance entre deux mots).
La distance de Hamming entre deux mots de même longueur est le nombre d'endroits où les lettres sont différentes.

Par exemple : JAPON SAVON
La première lettre de JAPON est différente de la première lettre de SAVON, les troisièmes aussi sont différentes. La distance de Hamming entre JAPON et SAVON vaut donc 2.

1. Complétez la fonction `distance_hamming()` prenant *chaine*1 et *chaine*2 deux chaînes de caractères de même longueur en argument, qui calcule la distance de Hamming entre deux chaînes.

Exécutez en console l'instruction `run_docstring_examples(distance_hamming, globals(), verbose =` pour vous assurer de son bon fonctionnement.
Exécutez en console l'instruction `test_distance_hamming()` pour exécuter des tests supplémentaires.

2. Deux chaînes de caractères sont paronymes si elles diffèrent par une seule lettre. Par exemple, JAPON et LAPON sont paronymes, alors que JAPON et SAVON ne le sont pas.

Complétez la fonction `est_paronymes()` prenant *chaine*1 et *chaine*2 deux chaînes de caractères de même longueur en argument, qui renvoie `True` si *chaine*1 et *chaine*2 sont des paronymes, et `False` sinon.

Testez votre fonction de la même façon que précédemment.

3. Pour les plus forts !.
Complétez la fonction `modification`, prenant *chaine*, une chaîne de caractères, *car*, un caractère unique, et *n*, un entier, en arguments, et renvoyant une nouvelle chaîne de caractères dont le caractère d'indice *i* de *chaine* a été remplacé par *car*.
Par exemple, `modification('trac', 2, 'u')` renverra `'truc'` (le caractère 'a' - celui d'indice 2 - a été remplacé par 'u'). On pourra par exemple :

- initialiser une variable *nouvelle_ chaine*,
- effectuer un parcours par index entre 0 et (n-1) pour "déposer" les caractères de *chaine* dans *nouvelle_ chaine*,
- ajouter le nouveau caractère,
- effectuer un nouveau parcours par index à partir de (n+1) pour "déposer" les autres caractères de *chaine*

Exercice 3 (Verlan).

1. Complétez la fonction `verlan()` prenant *chaine* une chaîne de caractères en argument, et renvoyant un mot à l'envers : SALUT devient TULAS.

   Testez votre fonction de la même façon que précédemment.

   Astuce : on pourra parcourir la chaîne de caractères à l'envers, c'est à dire depuis `len(chaine)-1` jusqu'à 0 inclus, par pas de -1.
2. Un palindrome est un mot qui s'écrit indifféremment de gauche à droite ou de droite à gauche ; par exemple RADAR est un palindrome.
   Écrivez une fonction `est_palindrome()` prenant *chaine* en argument, et renvoyant `True` si la chaîne de caractères est un palindrome. On pourra utiliser la fonction `verlan()`.

Exercice 4 (le chiffre de César).

En cryptographie, le chiffrement par décalage, aussi connu comme le chiffre de César est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes (ce qui explique le nom « chiffre de César »). Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite), on reprend au début.

Dans le cas ci-dessous, on considère que le texte à coder ne contient que des majuscules.

Ainsi, le mot « COUCOU » s'écrira « FRXFRX » si le décalage est de 3.

1. On cherche à obtenir un code du caractère 'A' qui prenne la valeur 0, celui du caractère 'B' qui prenne la valeur 1, ..., celui du caractère 'Z' qui prenne la valeur 25.
   Indiquez le code Python permettant d'obtenir cela.
2. Combien de caractères comporte ce nouvel alphabet ?
3. Le modulo (% en Python) correspond au reste de la division euclidienne de deux nombres. Quelles valeurs prennent les calculs suivants :
   a. `1 % 3`
   b. `2 % 3`
   c. `3 % 3`
   d. `4 % 3`
   e. `5 % 3`
4. Quel devrait être le modulo à appliquer pour qu'un caractère dont la valeur est 27 dans le nouvel alphabet prenne la valeur 1 ?
   Vérifiez que votre méthode fonctionne pour le caractère de valeur 33 (la fonction doit renvoyer 7).
5. Écrivez la fonction `cesar()` prenant en argument *texte*, une chaîne de caractères et *decalage*, un entier, et qui renvoie une chaîne de caractères modifiée suivant le code de César.
6. On désire utiliser la même fonction pour décoder un texte. Comment doit-on modifier la valeur de l'argument *décalage* ?
7. Cette méthode fonctionne-t-elle pour la chaîne de caractères suivante : "HELLO LE MONDE" ?
   Proposez une modification de la fonction.