CC BY NC SA

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
```

II. Opération de base sur les chaînes de caractères

II. 1. Longueur d'une chaîne

On peut déterminer la longueur (c'est-à-dire 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 *chaine* en argument, et renvoyant le pluriel 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 *chaine*1 et *chaine*2, deux chaines de caractères en argument, et renvoyant `"chaines égales"` si *chaine*1 et *chaine*2 sont égales, et le classement lexicographique si les chaînes sont différentes.

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. 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

.

Propriété 1

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

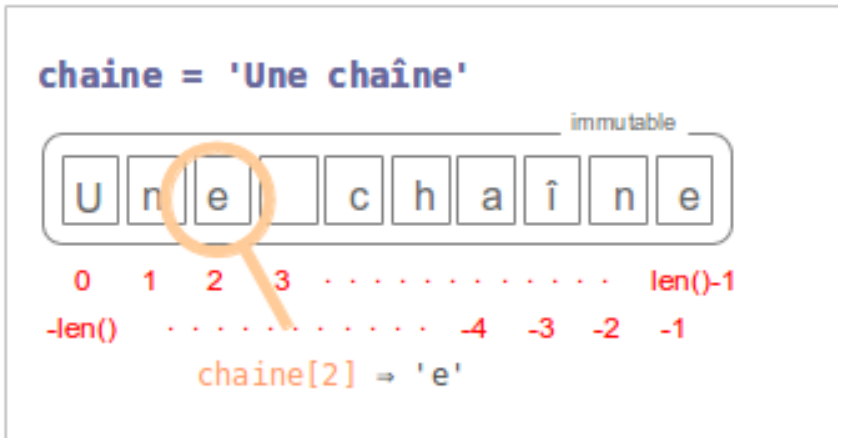

FIGURE 13.1 – Indexation dans une chaîne de caractères

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]` ?

IV. Non-mutabilité d'une chaîne de caractères

Propriété 2

Une cahîne de caractères est non-mutable (on dit aussi immuable), c'est à dire que l'on ne peut ni modifier l'un de ses éléments, ni supprimer l'un de ses éléments, ni ajouter un nouvel élément.

Exemple 10 (Erreur générée par la tentative d'affectation à une chaîne de caractères)

```
>>> ma_chaine = 'truc'
>>> ma_chaine[2]
'u'
>>> ma_chaine[2] = 'a'
TypeError                                 Traceback (most recent call last)
<ipython-input-2-4d26f6437d87> in <module>
----> 1 ma_chaine[2] = 'a'

TypeError: 'str' object does not support item assignment
```

V. Parcours d'une chaîne de caractères

V. 1. 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 le nombre de caractères de la chaîne (longueur de la chaine) ;
- ▷ on utilise une boucle bornée (boucle `for`) pour l'index 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

```
début
    longueur = nombre de caractères
    pour i compris entre 0 et longueur par pas de 1 faire
        action
    fin pour
fin
```

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

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

Exemple 12 (copie d'une chaîne de caractères)

Algorithme : double(chaine)

```
/* algorithme qui renvoie une chaîne de caractères, en doublant tous les
   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 i compris entre 0 et longueur par pas de 2 faire
5         nouvelle_chaine ← nouvelle_chaine + chaine[i] + chaine[i]
6     fin pour
7     renvoyer nouvelle_chaine
8 fin
```

```
1 def double(chaine):
2     longueur = len(chaine)
3     nouvelle_chaine = ""
4     for i in range(0, longueur, 1):
5         nouvelle_chaine = nouvelle_chaine + 2*chaine[i]
6     return nouvelle_chaine
```

V. 2. Parcours par élément

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

Exemple 13 (affichage d'une chaîne caractère par caractère)

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

Exemple 14 (copie d'une chaîne de caractères)

Algorithme : double(chaine)

```
/* algorithme qui renvoie une chaîne de caractères, en doublant tous les
   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 i compris entre 0 et longueur par pas de 2 faire
5         nouvelle_chaine ← nouvelle_chaine + chaine[i] + chaine[i]
6     fin pour
7     renvoyer nouvelle_chaine
8 fin
```

```python
def double(chaine):
    longueur = len(chaine)
    nouvelle_chaine = ""
    for car in chaine:
        nouvelle_chaine = nouvelle_chaine + 2*car
    return nouvelle_chaine
```

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.

VI. Exercices

Exercice 1 (conjugaison).

1. Écrivez 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'`.
2. Écrivez une 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 chante ... . On pourra utiliser la fonction *radical()*.

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. Écrivez une 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.
2. Deux mots sont paranymes s'ils diffèrent par une seule lettre. Écrivez une fonction *est_paronyme()* prenant *chaine1* et *chaine2* en argument, et renvoyant `True` si *chaine1* et *chaine2* sont paronymes, et `False` sinon. On pourra utiliser la fonction *distance_hamming()*.

Exercice 3 (paronymes).
Les paronymes sont des mots qui se ressemblent beaucoup du fait de leur orthographe. Cependant, même si deux mots ont une seule lettre qui diffère, ils ont un sens complètement différent.

1. Écrivez une fonction *modification()* prenant *chaine, car* et *n* en argument, et renvoyant une chaîne de caractères contenant les mêmes caractères que *chaine*, à l'exception du n^e caractère qui est remplace par *car*.
   Par exemple, l'exécution de `modification('trac', 2, 'u')` renverra 'truc' (le 2^e caractère est remplacé par 'a').
2. Dans la console Python, créez une variable *mon_mot* prenant la valeur `estamper`. Quelle instruction Python à saisir en console permet de modifier *mon_mot* pour qu'elle prenne la valeur `estomper` ?

Exercice 4 (Verlan).
.

1. Écrivez une fonction *verlan*() prenant *chaine* une chaîne de caractères en argument, et renvoyant un mot à l'envers : SALUT devient TULAS.
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 5 (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 une chaîne de caractères et un entier, et qui renvoie une chaîne de caractères modifiée suivant le code de César.
   Par exemple, `cesar("HELLO", 3)` renverra "KHOOR".
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.