Les chaînes de caractères

Résumé

Une **chaîne de caractères** (type `str`) est une suite de caractères : un mot, une phrase, un mot de passe, une ligne de fichier. Chaque caractère occupe une position numérotée à partir de 0, son **indice**. On lit un caractère avec son indice, on parcourt la chaîne avec une boucle, on en extrait un morceau avec une tranche. Une chaîne ne se modifie pas : pour la transformer, on en construit une nouvelle. Ces gestes servent ensuite tels quels pour les listes.

Prérequis : Les types de base, Les boucles.

① Mémo

Définition 1 (Indices d'une chaîne)

Dans `s = "python"`, le premier caractère a l'indice 0 et le dernier l'indice `len(s) - 1`, ici 5.

- `len(s)` renvoie le nombre de caractères : `6`.
- `s[0]` vaut `"p"`, `s[5]` vaut `"n"`.
- Les indices négatifs partent de la fin : `s[-1]` vaut `"n"`, `s[-2]` vaut `"o"`.

Parcourir une chaîne

- Par ses caractères : `for c in s:` donne `c` égal à chaque caractère, dans l'ordre.
- Par ses indices : `for i in range(len(s)):` donne `i` de 0 à `len(s) - 1`, et `s[i]` est le caractère.

On choisit le parcours par indices quand la position compte (comparer un caractère à son voisin, à son symétrique).

Tranches

`s[a:b]` est la chaîne des caractères d'indices `a` à `b - 1`.

- `"python"[1:4]` vaut `"yth"`;
- `s[:3]` : les trois premiers caractères ; `s[3:]` : tout à partir de l'indice 3 ;
- `s[::-1]` : la chaîne à l'envers.

Opérations courantes

`+` Concaténation : `"bon" + "jour"` vaut `"bonjour"`.
`*` Répétition : `"ab" * 3` vaut `"ababab"`.
`in` Appartenance : `"th" in "python"` vaut `True`.
`==` Égalité de deux chaînes, caractère par caractère, majuscules comprises.

Définition 2 (Une chaîne ne se modifie pas)

On dit qu'une chaîne est **immuable** : `s[0] = "P"` provoque une erreur. Pour obtenir une chaîne transformée, on en construit une nouvelle, par exemple `s = "P" + s[1:]`, ou caractère par caractère en partant de la chaîne vide `""`.

Pièges fréquents

- Le dernier indice est `len(s) - 1`, pas `len(s)`.
- Dans une tranche `s[a:b]`, le caractère d'indice `b` n'est pas pris.
- `"10" < "9"` vaut `True` : deux chaînes se comparent caractère par caractère, comme dans un dictionnaire, et `"1"` vient avant `"9"`.
- `"A" == "a"` vaut `False` : majuscules et minuscules sont des caractères différents.

Erreurs classiques

Code erroné	Code correct	Explication
`s[len(s)]` → `IndexError`	`s[len(s) - 1]` ou `s[-1]`	Les indices valides vont de 0 à `len(s) - 1`.
`s[0] = "P"` → `TypeError`	`s = "P" + s[1:]`	Une chaîne ne se modifie pas : on en construit une nouvelle.
`"Score : " + 12` → `TypeError`	`"Score : " + str(12)`	On ne concatène que des chaînes : convertir le nombre, ou écrire une f-string.
`for i in s: print(s[i])` → `TypeError`	`for c in s: print(c)`	Avec `for i in s`, la variable reçoit des caractères, pas des indices.

② Exemples

Programme 1 · Lire des caractères

```
s = "bonjour"
print(len(s))      # 7
print(s[0])        # b
print(s[-1])       # r
print(s[1:4])      # onj
```

Programme 2 · Compter un caractère

```
phrase = "les chaussettes de l'archiduchesse"
nb = 0
for c in phrase:
    if c == "s":
        nb = nb + 1
print(nb)    # 6
```

Programme 3 · Construire une chaîne à l'envers

```
mot = "radar"
envers = ""
for c in mot:
    envers = c + envers    # chaque caractère passe devant
print(envers)              # radar
print(envers == mot)       # True : radar est un palindrome
```

Programme 4 · Comparer un caractère à son voisin

```
mot = "pomme"
for i in range(len(mot) - 1):
    if mot[i] == mot[i + 1]:
        print(f"lettre doublée : {mot[i]}")    # lettre doublée : m
```

③ Exercices

Exercice 1 *Indices et tranches* On pose `s = "informatique"`. Sans exécuter, donner la valeur de chaque expression, puis vérifier.

1. `len(s)`
2. `s[0]` et `s[-1]`
3. `s[2:6]`
4. `s[:5] + s[-3:]`
5. `"mat" in s`

Exercice 2 *Voyelles et espaces* Écrire une fonction `compter(phrase)` qui renvoie le couple (nombre de voyelles, nombre d'espaces) de la chaîne `phrase`, écrite en minuscules et sans accents. Les voyelles sont `a`, `e`, `i`, `o`, `u`, `y`.

Exemple. `compter("le chat dort")` renvoie `(3, 2)`.

Exercice 3 *Palindrome sans retourner la chaîne* Un palindrome se lit de la même façon dans les deux sens, comme `"kayak"`. Écrire une fonction `est_palindrome(mot)` qui renvoie `True` ou `False` **sans construire la chaîne à l'envers** : comparer chaque caractère à son symétrique. Combien de comparaisons suffisent pour un mot de longueur `n` ?

Exercice 4 *Chiffrement de César* Le chiffrement de César décale chaque lettre de l'alphabet d'un nombre fixe de rangs : avec un décalage de 3, `a` devient `d`, et `z` devient `c`. On donne `alphabet = "abcdefghijklmnopqrstuvwxyz"`.

1. Écrire une fonction `chiffrer(texte, decalage)` qui renvoie le texte chiffré ; les caractères qui ne sont pas des lettres minuscules restent inchangés.
2. Comment déchiffrer avec la même fonction ?

Exemple. `chiffrer("ave cesar", 3)` renvoie `"dyh fhvdu"`.

Solutions des exercices

Corrigé de l'exercice 1.

1. `12`.
2. `"i"` et `"e"`.
3. `"form"` : indices 2, 3, 4 et 5.
4. `"infor" + "que"`, soit `"inforque"`.
5. `True` : les lettres `m`, `a`, `t` se suivent aux indices 5, 6 et 7.

Corrigé de l'exercice 2.

```
def compter(phrase):
    voyelles = 0
    espaces = 0
    for c in phrase:
        if c in "aeiouy":
            voyelles = voyelles + 1
        elif c == " ":
            espaces = espaces + 1
    return voyelles, espaces

print(compter("le chat dort"))    # (3, 2)
```

Principe : `c in "aeiouy"` teste si le caractère est l'une des voyelles ; deux compteurs avancent pendant un seul parcours.

Corrigé de l'exercice 3.

```
def est_palindrome(mot):
    n = len(mot)
    for i in range(n // 2):
        if mot[i] != mot[n - 1 - i]:
            return False
    return True

print(est_palindrome("kayak"))    # True
print(est_palindrome("kayaks"))   # False
print(est_palindrome(""))         # True
```

Le symétrique de l'indice `i` est `n - 1 - i`. Il suffit de `n // 2` comparaisons : au-delà, on comparerait les mêmes paires dans l'autre sens, et le caractère du milieu d'un mot de longueur impaire n'a pas besoin d'être comparé.

Corrigé de l'exercice 4.

1. `alphabet.index(c)` donne le rang de la lettre ; le modulo 26 fait revenir au début après `z`.

```python
alphabet = "abcdefghijklmnopqrstuvwxyz"

def chiffrer(texte, decalage):
    resultat = ""
    for c in texte:
        if c in alphabet:
            rang = alphabet.index(c)
            resultat = resultat + alphabet[(rang + decalage) % 26]
        else:
            resultat = resultat + c
    return resultat

print(chiffrer("ave cesar", 3))     # dyh fhvdu
print(chiffrer("dyh fhvdu", -3))    # ave cesar
```

2. On chiffre avec le décalage opposé : `chiffrer(texte, -3)`. En Python, `-1 % 26` vaut 25, donc le retour en arrière fonctionne aussi avant `a`.