Chaînes de caractères

Activité – Première NSI

① Construire un message personnalisé

Exercice 1 *Trois manières d'afficher un message*

On souhaite afficher la phrase « `Alice a 17 ans et habite à Paris.` », en utilisant les variables `prenom = "Alice"`, `age = 17` et `ville = "Paris"`.

Version A (concaténation avec `+`) :

```
print(prenom + " a " + str(age) + " ans et habite à " + ville + ".")
```

Version B (`print` avec virgules) :

```
print(prenom, "a", age, "ans et habite à", ville + ".")
```

Version C (f-string) :

```
print(f"{prenom} a {age} ans et habite à {ville}.")
```

1. Quelle version est la plus *lisible* ? La plus courte ?
2. Dans la version A, à quoi sert l'appel `str(age)` ? Que se passerait-il si on l'oubliait ?
3. La version B affiche-t-elle exactement la même chose que la A et la C ? *Indication : regarder de près les espaces.*
4. Quel est le caractère qui distingue une f-string d'une chaîne ordinaire ? À quoi servent les accolades ?

Remarque 1. Une **f-string** (« *formatted string* ») est la manière moderne et recommandée d'insérer des valeurs dans une chaîne en Python. Elle commence par `f` avant le premier guillemet, et utilise des accolades `{...}` pour insérer des expressions.

② Accéder aux caractères

Définition 1 **(Indices et tranches)** Une chaîne est une séquence indexée à partir de **0**. On accède à un caractère par son indice : `mot[i]`. Les indices négatifs partent de la fin : `mot[-1]` est le dernier caractère.

Une **tranche** (*slice*) extrait une sous-chaîne : `mot[debut:fin]` renvoie les caractères d'indice `debut` à `fin - 1` (la borne de droite est exclue).

Exercice 2 *Premiers accès*

On considère la chaîne `mot = "INFORMATIQUE"` (12 caractères, indices 0 à 11).

1. Que valent `mot[0]`, `mot[5]`, `mot[11]` ?
2. Que valent `mot[-1]`, `mot[-3]` ?
3. Que vaut `len(mot)` ?
4. Que valent les tranches `mot[0:4]`, `mot[4:8]`, `mot[8:]`, `mot[:4]`, `mot[-3:]` ?
5. Comment obtenir la chaîne `mot` à l'envers en une seule expression ? *Indication : tester `mot[::-1]`.*

③ Les f-strings en pratique

Propriété 1 **(Ce qu'on peut mettre entre les accolades)** Tout ce qui s'évalue en Python : une variable, une opération arithmétique, un appel de fonction, voire une condition. Quelques options de formatage utiles :

— `{nombre:.2f}` : nombre flottant avec 2 décimales ;

— `{nombre:5d}` : entier sur 5 caractères (aligné à droite) ;

— `{texte:<10}` : chaîne sur 10 caractères (alignée à gauche).

Exercice 3 *Construire une facture*

1. Compléter le code pour qu'il affiche « `Total : 23.45 euros` » :

```
montant = 23.45
print(f"Total : ..... euros")
```

2. On dispose de `nb = 7` et `prix = 3.5`. Compléter pour afficher « `7 articles à 3.50 euros = 24.50 euros` » (**deux décimales** pour les prix) :

```
nb = 7
prix = 3.5
print(f"{.....} articles à {.....:.2f} euros = {.....:.2f}
    euros")
```

3. Que produit la ligne suivante ? Justifier en regardant ce qui est entre les accolades.

```
a, b = 5, 12
print(f"Le maximum de {a} et {b} est {max(a, b)}.")
```

4 Parcourir une chaîne

Propriété 2 **(Parcourir, tester l'appartenance)** On parcourt une chaîne caractère par caractère avec une boucle `for` :

```
for c in "NSI":
    print(c)
```

On teste l'appartenance d'un caractère ou d'une sous-chaîne avec l'opérateur `in` : `"S" in "NSI"` renvoie `True`, `"NS" in "ENSEMBLE"` renvoie `True`.

Exercice 4 *Compter les voyelles*

Compléter la fonction `nb_voyelles(mot)` qui renvoie le nombre de voyelles (parmi `aeiouy`) du `mot`, sans tenir compte des majuscules.

```
1  def nb_voyelles(mot):
2      voyelles = "aeiouy"
3      n = ............
4      for c in mot.lower():        # parcours en minuscules
5          if c in ............:
6              n = ............
7      return n
```

Tester avec `"NSI"` (résultat : 1), `"Anticonstitutionnellement"` (résultat : 9), `"BCDFG"` (résultat : 0).

5 Quelques méthodes utiles

Propriété 3 **(Méthodes courantes (non exhaustif))**

- `mot.upper()` et `mot.lower()` : passage en majuscules / minuscules.
- `mot.replace(a, b)` : remplace toutes les occurrences de `a` par `b`.
- `mot.split(s)` : découpe en liste selon le séparateur `s` (par défaut, les espaces).
- `s.join(liste)` : concatène les éléments d'une liste de chaînes en les séparant par `s`.
- `mot.strip()` : enlève les espaces (et autres caractères blancs) en début et fin.

Important : ces méthodes *renvoient une nouvelle chaîne*, elles ne modifient pas l'originale (les chaînes sont *immuables*).

Exercice 5 *Manipulations*

On considère la chaîne `phrase = " Bonjour le Monde "`.

1. Donner le résultat de chaque expression :
   a) `phrase.upper()`
   b) `phrase.strip()`

c) `phrase.strip().split()`

d) `phrase.replace("Monde", "Soleil")`

e) `"-".join(["NSI", "Première", "2026"])`

2. Après les appels précédents, que vaut `phrase` ? Pourquoi ?

6 Application : vérifier un palindrome

Exercice 6 *Un mot, un palindrome*

Un **palindrome** est un mot qui se lit de la même manière dans les deux sens : `"radar"`, `"kayak"`, `"laval"`.

1. Écrire en une seule ligne, à l'aide de `[::-1]`, l'expression qui inverse la chaîne `mot`.
2. Compléter la fonction `est_palindrome(mot)` qui renvoie `True` si `mot` est un palindrome, `False` sinon :

```
def est_palindrome(mot):
    return mot == ............
```

3. Tester avec `"radar"`, `"NSI"`, `""`, `"a"`.
4. **Pour aller plus loin.** Modifier la fonction pour qu'elle ne tienne pas compte de la casse. Par exemple, `"Bob"` doit être considéré comme un palindrome.

7 Ce qu'il faut retenir

Propriété 4 **(Indices et tranches)** `mot[i]` accède au caractère d'indice i (à partir de 0 ; les indices négatifs partent de la fin). `mot[a:b]` renvoie les caractères d'indice a à $b-1$. `mot[::-1]` renvoie la chaîne à l'envers.

Propriété 5 **(f-strings)** `f"texte {expression} texte"` insère le résultat de l'*expression* dans la chaîne. C'est la méthode moderne, lisible et concise pour formater des chaînes. Options utiles : `:.2f` (deux décimales), `:5d` (entier sur 5 caractères).

Propriété 6 **(Immuabilité)** Les chaînes sont **immuables** : les méthodes (`upper`, `replace`, `strip`...) renvoient une *nouvelle* chaîne sans modifier l'originale. Pour conserver le résultat, il faut affecter à une variable.

Remarque 2. On parcourt une chaîne caractère par caractère avec `for c in mot:`, et on teste l'appartenance d'un caractère ou d'une sous-chaîne avec `in`. Ces deux outils suffisent à résoudre la plupart des problèmes courants sur les chaînes.