OPERATIONS AVANCEES SUR LES CHAINES DE CARACTERES :

TABLEAU DE CHAINES DE CARACTERES :

Quand on commence à avoir beaucoup de chaînes de caractères à stocker dans un programme, utiliser une variable pour chaque chaîne de caractères n'est pas très efficace, il vaut mieux créer un tableau de chaînes de caractères.

```
lignes = [""] * 3
for idLigne in range(3):
   lignes[idLigne] = input()
```

Initialiser un tableau

Pour initialiser un tableau de chaînes de caractères, il faut utiliser la syntaxe suivante :

```
lignes = ["Premier texte", "Second texte", "Troisieme texte"]
```

TRIER UN TABLEAU DE CHAINES DE CARACTERES :

Vous avez déjà vu dans le chapitre sur les tableaux comment trier des tableaux d'entiers, nous allons voir comment trier des tableaux de chaînes de caractères.

```
lignes = ["Texte C", "Texte A", "Texte B"]
lignes.sort()
print(lignes[0])
```

```
↳ Texte A
```

La syntaxe est donc très similaire à ce que nous avions déjà vu sur les tableaux d'entiers.

FAIRE UNE COPIE D'UNE CHAINE DE CARACTERES :

Lorsqu'on a besoin de faire une copie d'une chaîne de caractères, il faut utiliser le code suivant :

```
texte = "Exemple de texte"
texteCopie = texte
print(texteCopie)
```

```
↳ Exemple de texte
```

Le code est donc très intuitif, il n'y a pas de remarques particulières à faire.

CONCATENER DEUX CHAINES DE CARACTERES :

Lorsqu'on a besoin de former une grande chaîne de caractères à partir de chaînes plus petites, on dit qu'on *concatène* les chaînes de caractères.

```
texteDebut = "Ceci est "
texteFin = "une phrase complete"
texteComplet = texteDebut + texteFin
print(texteComplet)
```

```
↳ Ceci est une phrase complete
```

Le code est donc très intuitif, il n'y a pas de remarques particulières à faire.

Remarque sur de possible lenteurs :

Il faut **ABSOLUMENT** éviter de concaténer des chaînes au sein d'une boucle, avec un code de la forme suivante :

```
monTexte = ""
Répéter 1000 fois
   monTexte = Concaténer monTexte et "X"
```

Un tel code fera des copies multiples de `monTexte` et son temps d'exécution sera proportionnel à 1000*1000 et non pas à 1000. Sa complexité est donc très mauvaise et peut rendre votre programme trop lent pour être accepté. N'hésitez pas à lire le chapitre sur la complexité pour réviser ce concept et à relire le cours sur la modification d'une chaîne de caractère.

ITERATIONS :

LIRE UN NOMBRE INCONNU DE CHOSES :

Dans certains exercices un peu difficiles, on peut avoir besoin de lire un nombre inconnu de choses, par exemple de calculer la somme de tous les entiers donnés (un par ligne), sans qu'on sache à l'avance combien il y a d'entiers ! Voici comment résoudre ce problème :

Il faut utiliser le code suivant :

```
try:
    entier = int(input())
except:
    # On a pas réussi à lire l'entier
    print("Une erreur est arrivée")
```

Sans rentrer dans le méchanisme des exceptions en Python, sachez que la ligne `entier = int(input())` "envoie" une exception en cas d'erreur, exception qu'on "rattrape" à l'aide du `except`, et on peut donc réagir en conséquence.

On est donc capable de détecter qu'une chose (ici un entier) a été lue ou pas. Il faudra bien sûr adapter le morceau de code ci-dessus selon la situation rencontrée.

ITERATIONS SUR CARACTERES :

Imaginons qu'on souhaite afficher les lettres de A à F. On pourrait bien sûr utiliser 6 commandes d'affichage mais on sent que cela n'est pas des plus efficace. Voici la solution la plus simple :

```
for ascii in range(ord('A'), ord('F') + 1):
    print(chr(ascii))
```

```
A
B
C
D
E
F
```

On est donc obligé de faire la boucle en itérant sur les codes ASCII des caractères.