Exercices sur les boucles for

Programmation en Python

Une seule boucle `for`

Exercice 1 Somme des nombres d'une liste

Écrire un programme en Python qui calcule la somme des éléments d'une liste de nombres, avec deux méthodes :

1. En parcourant les éléments par leur indice,
2. Avec la syntaxe `for element in liste`.

Solution :

```
# On définit la liste de nombres à traiter
liste_nombres = [2, 5, 7, 10, 13, 18]

# On initialise la variable qui va stocker la somme des nombres
somme_nombres = 0

# On parcourt la liste par les indices
for i in range(len(liste_nombres)):
    # On ajoute l'élément courant à la somme des nombres
    somme_nombres += liste_nombres[i]

# On affiche la somme des nombres
print("La somme des nombres de la liste est :", somme_nombres)
```

Solution :

Dans cette première version, on utilise la fonction `range()` pour générer une séquence d'indices allant de 0 à la longueur de la liste - 1. On parcourt ensuite cette séquence d'indices avec la boucle `for`, et pour chaque indice, on ajoute la valeur correspondante à la variable `somme_nombres`.

Solution :

```
# On définit la liste de nombres à traiter
liste_nombres = [2, 5, 7, 10, 13, 18]

# On initialise la variable qui va stocker la somme des nombres
somme_nombres = 0

# On parcourt la liste par les éléments
for nombre in liste_nombres:
    # On ajoute l'élément courant à la somme des nombres
    somme_nombres += nombre

# On affiche la somme des nombres
print("La somme des nombres de la liste est :", somme_nombres)
```

Solution :

Dans cette deuxième version, on utilise directement la syntaxe `for element in liste` pour parcourir les éléments de la liste. À chaque itération de la boucle, on ajoute la valeur de l'élément courant à la variable `somme_nombres`.

Exercice 2 Somme des nombres pairs d'une liste

1. Vous disposez d'une liste de nombres [2, 5, 7, 10, 13, 18].
2. Écrivez un programme en Python qui calcule la somme de tous les nombres pairs de la liste à l'aide d'une boucle for.
3. Affichez ensuite cette somme à l'écran.

Solution :

```
liste_nombres = [2, 5, 7, 10, 13, 18]
somme_nombres_pairs = 0

for nombre in liste_nombres:
    if nombre % 2 == 0:  # On vérifie si le nombre est pair
        somme_nombres_pairs += nombre  # On ajoute le nombre à la somme

print("La somme des nombres pairs de la liste est :", somme_nombres_pairs)
```

Explications :

- On dispose d'une liste de nombres : [2, 5, 7, 10, 13, 18].
- On va parcourir tous les nombres de cette liste pour trouver ceux qui sont pairs.
- Pour chaque nombre de la liste, on va tester s'il est pair. Pour cela, on va utiliser la propriété suivante : un nombre est pair s'il est divisible par 2 sans reste. On utilise donc l'opération modulo % pour calculer le reste de la division par 2, et on teste si ce reste est égal à 0.
- Si le nombre courant est pair, on l'ajoute à une variable somme_nombres_pairs qui va stocker la somme des nombres pairs.
- Après avoir parcouru tous les nombres de la liste, on affiche la valeur de la variable somme_nombres_pairs.

Exercice 3 Nombre d'occurences d'une lettre dans une chaîne de caractères

Écrire un programme en Python qui doit compter le nombre d'occurrences d'une lettre particulière dans une chaîne de caractères donnée. Par exemple, pour la chaîne de caractères `"Bonjour tout le monde"`, il doit être retourné la valeur `3` pour la chaîne de caractères `"o"`.

Solution :

```
# Définition de la chaîne de caractères à traiter
chaine = "Bonjour tout le monde"

# Définition de la lettre que l'on cherche
lettre = 'o'

# Initialisation du compteur d'occurrences de la lettre
nb_occurrences = 0

# Parcours de chaque caractère de la chaîne
for caractere in chaine:
    # Vérification si le caractère est égal à la lettre que l'on cherche
    if caractere == lettre:
        # Si oui, on incrémente le compteur d'occurrences
        nb_occurrences += 1

# Affichage du nombre d'occurrences de la lettre
print(f"La lettre '{lettre}' apparaît {nb_occurrences} fois dans la chaîne '{chaine}'.")
```

Explications :

L'objectif de cet exercice est de compter le nombre d'occurrences de chaque lettre dans une chaîne de caractères.

Pour ce faire, on peut utiliser un dictionnaire en Python. Un dictionnaire est une structure de données qui permet de stocker des éléments sous la forme de paires clé-valeur. Dans notre cas, la clé sera la lettre et la valeur sera le nombre d'occurrences de cette lettre dans la chaîne.

Voici les étapes du programme :

- On définit la chaîne de caractères que l'on veut analyser avec une variable `chaine`.
- On définit la lettre que l'on cherche avec une variable `lettre`.
- On initialise un compteur `nb_occurrences` qui va stocker le nombre d'occurrences de la lettre dans la chaîne.

- On parcourt chaque caractère de la chaîne à l'aide d'une boucle for.
- Pour chaque caractère de la chaîne, on vérifie s'il est égal à la lettre que l'on cherche. Si c'est le cas, on incrémente le compteur `nb_occurrences` de 1.
- Enfin, on affiche le nombre d'occurrences de la lettre en question à l'aide de la commande `print()`.

Exercice 4 Nombre d'occurences de chaque lettre d'une chaîne de caractères

Écrire un programme en Python qui permet de compter le nombre d'occurrences de chaque lettre dans une chaîne de caractères donnée.

Dans ce programme, on utilise un dictionnaire pour stocker les occurrences de chaque lettre dans la chaîne. Les clés sont les lettres rencontrées dans la chaîne, et les valeurs sont les occurences de chaque lettre.

Solution :

```
# On définit la chaîne de caractères à traiter
chaine = "Bonjour tout le monde"

# On initialise un dictionnaire pour stocker les occurrences de chaque lettre
occurrences = {}

# On parcourt chaque caractère de la chaîne
for lettre in chaine:
    # Si la lettre est déjà dans le dictionnaire, on incrémente son nombre d'occurrences
    if lettre in occurrences:
        occurrences[lettre] += 1
    # Sinon, on ajoute la lettre au dictionnaire avec une occurrence de 1
    else:
        occurrences[lettre] = 1

# On affiche les résultats
print("Occurrences de chaque lettre dans la chaîne :")
for lettre, nb_occurrences in occurrences.items():
    print(lettre, ":", nb_occurrences)
```

Explications :

L'objectif de cet exercice est de compter le nombre d'occurrences de chaque lettre dans une chaîne de caractères.

Pour ce faire, on peut utiliser un dictionnaire en Python. Un dictionnaire est une structure de données qui permet de stocker des éléments sous la forme de paires clé-valeur. Dans notre cas, la clé sera la lettre et la valeur sera le nombre d'occurrences de cette lettre dans la chaîne.

Voici les étapes du programme :

- On définit la chaîne de caractères à traiter avec une variable `chaine`.
- On initialise un dictionnaire vide avec une variable `occurrences` qui nous permettra de stocker les occurrences de chaque lettre.
- On parcourt chaque caractère de la chaîne avec une boucle `for`.
- Pour chaque lettre, on vérifie si elle est déjà présente dans le dictionnaire `occurrences`. Si c'est le cas, on incrémente son nombre d'occurrences. Sinon, on ajoute la lettre au dictionnaire avec une occurrence de 1.
- Enfin, on affiche les résultats obtenus en parcourant le dictionnaire avec la méthode `items()`. On obtient ainsi une liste de tuples (lettre, nombre d'occurrences), que l'on peut afficher à l'aide de la commande `print()`.

Exercice 5 Paires de nombres premiers jumeaux

Écrire un programme qui génère toutes les paires de nombres premiers jumeaux inférieures à 1000.

Pour rappel, deux nombres premiers sont dits jumeaux si leur différence est égale à 2. Par exemple, (3, 5) est une paire de nombres premiers jumeaux car 5-3 = 2.

Dans le code, on utilisera une fonction `est_premier` qui prend un entier `n` en argument, et qui renseigne si l'entier est premier ou pas :

```
def est_premier(n):
    """Fonction qui renvoie True si le nombre n est premier, False sinon"""
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
```

Il faut compléter ce code pour écrire le script demandé.

Solution :

```
def est_premier(n):
    """Fonction qui renvoie True si le nombre n est premier, False sinon"""
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

# Recherche des paires de nombres premiers jumeaux
for i in range(2, 1000):
    if est_premier(i) and est_premier(i+2):
        print(i, i+2)
```

Explications :

Voici les étapes du programme :

- On a défini une fonction `est_premier(n)` qui permet de vérifier si un nombre `n` est premier ou non.
- On initialise une liste vide avec une variable `paires` qui nous permettra de stocker toutes les paires de nombres premiers jumeaux trouvées.
- On parcourt chaque nombre entre 3 et 999 (inclus) avec une boucle `for`.
- Pour chaque nombre, on vérifie s'il est premier et s'il existe un nombre premier jumeau. Si c'est le cas, on ajoute la paire de nombres à la liste `paires`.
- Enfin, on affiche les résultats obtenus en parcourant la liste `paires` avec une boucle `for` et en affichant chaque paire à l'aide de la commande `print()`.

Deux boucles `for`

Exercice 6 Les tables de multiplication

Écrire un programme Python qui crée une table de multiplication à partir des nombres de 1 à 10. On doit créer une liste (`table`) de listes(`ligne`). L'écriture de chaque liste de `table` doit donner ce résultat :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
[4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
[6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
[8, 16, 24, 32, 40, 48, 56, 64, 72, 80]
[9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Solution :

```
table = []
for i in range(1, 11):
    ligne = []
    for j in range(1, 11):
        resultat = i * j
        ligne.append(resultat)
    table.append(ligne)

for ligne in table:
    print(ligne)
```

Explications :

Voici les étapes du programme :

- On crée une liste vide appelée `table` pour stocker la table de multiplication.
- On utilise une boucle `for` pour parcourir les nombres de 1 à 10. À chaque tour de boucle, la variable `i` prend la valeur du nombre en cours d'analyse.
- À l'intérieur de cette boucle `for`, on crée une nouvelle liste vide appelée `ligne` pour stocker les résultats de la multiplication en cours.
- On utilise une deuxième boucle `for` pour parcourir les nombres de 1 à 10 à nouveau. Cette boucle permet de multiplier le nombre de la première boucle (`i`) par chaque nombre de la deuxième boucle (`j`) pour obtenir le résultat de chaque multiplication.
- À chaque multiplication, le résultat est stocké dans la variable `resultat`, puis ajouté à la liste `ligne` à l'aide de la méthode `append()`.
- Une fois que la boucle intérieure est terminée, la liste complète `ligne` est ajoutée à la liste `table` à l'aide de la méthode `append()`. Cela permet de stocker toutes les lignes de la table de multiplication.
- Une fois que la boucle extérieure est terminée, on affiche la table de multiplication en parcourant chaque ligne de la liste `table` à l'aide d'une troisième boucle `for`. Pour chaque ligne, on utilise la commande `print()` pour afficher les résultats de chaque multiplication.