

# Travaux Pratique

## Activité 1 : Chaînes de caractères

- 1) Écrire un Programme Python qui :
  - Prend une phrase puis imprime le nombre de mots de cette phrase
  - Prend une chaîne en entrée et compte le nombre de lettres majuscules dans la chaîne
  - Vérifie si une chaîne donnée est un palindrome ou non.
  - Inverse l'ordre des mots dans une phrase donnée.
- 2) Écrire un Programme Python qui effectue une compression de chaîne de base en utilisant le nombre de caractères répétés. Par exemple, la chaîne « aabccccaaa » deviendrait « a2b1c5a3 »

## Activité 2 : Manipulation de chaînes et fonctions basiques

1. Créer une fonction `format_usernames()` qui prend une liste de noms d'utilisateurs

Pour chaque nom :

- Mettre en minuscules
- Remplacer les espaces par des underscores
- Ajouter un préfixe "user\_" si le nom ne commence pas déjà par "user\_"

Retourner la liste des noms formatés

*Exemple :*

```
>>> noms = ["John Doe", "user_alice", "BOB SMITH"]
>>> format_usernames(noms)
['user_john_doe', 'user_alice', 'user_bob_smith']
```

## Activité 3 : Les Fonctions

- 1) Ecrire une fonction `som()` qui calcule et retourne la somme de deux paramètres numériques.  
*Exemple : `som(1,2) => 3`*
- 2) Modifier la fonction somme en attribuant une valeur par défaut (0) au deuxième paramètre pour pouvoir appeler la fonction avec un seul paramètre.  
*Exemple : `som(4) => 4`*
- 3) Modifier la fonction somme pour rendre le nombre de paramètres variable.  
*Exemple : `som(3,5,2) => 10`*
- 4) Ajouter un paramètre qui contrôle si la fonction accepte ou non des valeurs négatives.  
*Exemple : `som(1,-2,3, negative=False) => 4` / `som(1, -2, 3, negative=True) => 2`*
- 5) Modifier la fonction pour accepter n'importe quel mot-clé et afficher les valeurs des paramètres keywords reçus. Tester avec différents paramètres keywords.
- 6) Déclarer une liste contenant les valeurs suivantes : `l1 = [13, 10, 2, 5, 1, 9]`
- 7) Déclarer une fonction `select()` qui reçoit en paramètre un entier et retourne True ou False selon si cet entier est inférieur ou supérieur à 10.
- 8) Utiliser la fonction `filter()` pour appliquer la fonction `select()` sur les éléments de `l1` et stocker le résultat dans une liste `l2`.

9) Réaliser le même comportement en utilisant une fonction lambda au lieu de la fonction `select()` et stocker le résultat dans une liste l3.

#### **Activité 4 : Fonctions à Arguments variables et fonctions lambda avec map/filter**

Objectif : Créer un système de filtrage et de formatage de données.

1) Créer une fonction `process_data()` qui accepte:

- Une liste de nombres comme premier argument
- Un nombre variable d'opérations à appliquer (\*args)
- Des paramètres de formatage optionnels (\*\*kwargs)

Paramètres kwargs acceptés :

- 'precision': nombre de décimales (défaut: 2)
- 'prefix': préfixe à ajouter (défaut: '')
- 'suffix': suffixe à ajouter (défaut: '')

2) Utiliser `map()` avec des lambdas pour appliquer les opérations

3) Utiliser `filter()` pour éliminer les valeurs négatives

4) Formater chaque résultat selon les paramètres

*Exemple:*

```
>>> ops = [lambda x: x*2, lambda x: x+3]
>>> process_data([1, 2, 3], *ops, precision=1, suffix='€')
['5.0€', '7.0€', '9.0€']
```

#### **Activité 5 : Les Décorateurs**

1) Créer un décorateur `cache` qui une fois appliqué sur une fonction :

- Mesure le temps d'exécution et l'affiche
- Compte le nombre d'appels de la fonction et l'affiche
- Permet d'implémenter un système de cache en retournant la dernière valeur calculée au lieu d'exécuter la fonction

Créer un fonction `fact()` qui calcule la factorielle d'un nombre et l'affiche dans la fonction avant de le retourner.

2) Appliquer le décorateur `cache` sur la fonction `fact` et vérifier que le comportement attendu est correctement implémenté.