Exercices Pratiques de Python : Fonctions

Ali ZAINOUL

Exercices Pratiques de Python : Fonctions

1. **Définition et appel d'une fonction** :
   - Créez une fonction `greet()` qui affiche "Hello, world!" et appelez-la.
2. **Retourner des valeurs** :
   - Écrivez une fonction `add(a, b)` qui prend deux arguments et retourne leur somme.
3. **Valeurs par défaut** :
   - Créez une fonction `greet(name, message="Hello")` qui affiche un message de salutation avec un nom et un message par défaut.
4. **Passage par étiquette** :
   - Écrivez une fonction `multiply(a, b=2)` qui multiplie deux nombres avec un argument par défaut.
5. **Nombre variable de paramètres dans une fonction** :
   - Créez une fonction `add_all(*numbers)` qui prend un nombre variable d'arguments et retourne leur somme.
6. **Fonctions avec des paramètres clé/valeur** :
   - Écrivez une fonction `personal_info(**kwargs)` qui prend des arguments clé/valeur et affiche les informations sous forme de dictionnaire.
7. **Les générateurs - Instruction** `yield` :

- Créez un générateur `count_to_ten()` qui génère les nombres de 1 à 10.

8. **Notion de scope (Espace de noms) - portée des variables** :
   - Écrivez un programme pour illustrer la portée des variables globales et locales dans une fonction.
9. **Scopes pré-définis (`__builtins__`)** :
   - Utilisez la fonction `dir()` pour lister les fonctions et variables prédéfinies.
10. **Fonction `dir()`** :
    - Créez une fonction `inspect(obj)` qui utilise `dir()` pour afficher les attributs et méthodes d'un objet donné.
11. **Fonctions génériques (duck typing)** :
    - Écrivez une fonction `process(element)` qui accepte différents types d'éléments (entiers, chaînes, listes) et les traite en conséquence.
12. **Nombre d'arguments arbitraire (`*args`, `**kwargs`)** :
    - Créez une fonction `calculate(operation, *args)` qui prend une opération ('add', 'multiply') et un nombre variable d'arguments pour effectuer l'opération.
13. **Fonctions anonymes (`lambda`)** :
    - Écrivez un programme qui utilise une fonction `lambda` pour trier une liste de tuples en fonction du deuxième élément.
14. **Fonction `eval()`** :
    - Créez un programme qui prend une expression mathématique sous forme de chaîne de caractères et utilise `eval()` pour calculer le résultat.
15. **Fonction `exec()`** :
    - Écrivez un script qui lit un code Python à partir d'un fichier et utilise `exec()` pour l'exécuter.

16. **Fonction `map()` :**

    - Créez une liste de nombres et utilisez `map()` avec une fonction `lambda` pour convertir chaque nombre en une chaîne de caractères indiquant si le nombre est pair ou impair. Par exemple, pour la liste `[1, 2, 3, 4]`, la nouvelle liste devrait être `["1 est impair", "2 est pair", "3 est impair", "4 est pair"]`.

17. **Fonction `filter()` :**

    - Écrivez un programme qui utilise `filter()` pour créer une liste de mots dont la longueur est supérieure à 3 caractères à partir d'une liste de mots donnés. Par exemple, pour la liste `["chat", "chien", "poisson", "rat"]`, la nouvelle liste devrait être `["chat", "chien", "poisson"]`.

18. **Combinaison de `map()` et `filter()` :**

    - Utilisez `map()` et `filter()` ensemble pour transformer une liste de phrases en leur longueur et ensuite filtrer les longueurs qui sont supérieures à 5. Par exemple, pour la liste `["Bonjour", "Salut", "Hello", "Bienvenue"]`, la nouvelle liste devrait être `[7, 9]`.

19. **Projet Final** : Créez un programme de gestion de bibliothèque qui inclut les éléments suivants :

    - **Partie 1 : Définition des fonctions de base**
      - Créez une fonction `add_book(library, book_name)` qui ajoute un livre à la bibliothèque (représentée par un dictionnaire).
      - Créez une fonction `remove_book(library, book_name)` qui supprime un livre de la bibliothèque.
      - Créez une fonction `find_book(library, book_name)` qui vérifie si un livre est présent dans la bibliothèque.
    - **Partie 2 : Générateurs et itérateurs**
      - Créez un générateur `available_books(library)` qui génère la liste des livres disponibles dans la bibliothèque.
    - **Partie 3 : Gestion avancée des livres**
      - Utilisez des fonctions anonymes `lambda` avec `map()` pour créer une liste des noms des livres en majuscules.

  - Utilisez `filter()` pour créer une liste des livres dont le nom commence par une lettre spécifique.

- **Partie 4 : Exécution dynamique**
  - Utilisez `eval()` pour permettre à l'utilisateur de saisir une expression conditionnelle pour rechercher des livres (par exemple, tous les livres contenant "Python").
  - Utilisez `exec()` pour permettre à l'utilisateur d'ajouter dynamiquement une nouvelle fonctionnalité au programme (par exemple, ajouter une nouvelle méthode de tri pour les livres).