الكلية متعددة التخصصات تارودانت

FACULTE POLYDISCIPLINAIRE TAROUDANT

TP1 : Maîtrise des fonctions en Python

Enseignants	Pr. TAHA
Tags	TP, Tri et recherche
Année universitaire	2024-2025
Section	MIP_GI_S2

Objectifs :

1. Concevoir des fonctions robustes en gérant divers types de paramètres.
2. Comprendre et manipuler les valeurs par défaut et la liaison dynamique de fonctions.
3. Explorer la notion d'arguments variables (`args` , `*kwargs`) et les implications sur la flexibilité d'appel.
4. Illustrer les différences de comportement de passage d'arguments mutables vs immutables.
5. Maîtriser la portée des variables (locale vs globale) et ses pièges.
6. Utiliser des fonctions anonymes (`lambda`) dans des contextes avancés (fonctions d'ordre supérieur, tri, filtrage).

Exercice 1 : Fonctions avec paramètres et valeurs par défaut

1. Écrire une fonction `def power(base, exponent=2):`
    - Calcule la puissance `base**exponent` .
    - Gérer les cas où `exponent` est négatif (retourner un float).
2. Tester la fonction dans les situations suivantes :
    - Appel sans second argument (`power(5)`).
    - Appel explicite (`power(base=3, exponent=4)`).
    - Appel avec nommage partiel (`power(2, exponent=5)`).

Exercice 2 : Affecter une instance de fonction à une variable

1. Assignez la fonction `power` à une variable `f1` puis appelez-la (`f1(7,3)`).
2. Définissez une fonction `scale(x, factor=1.2)` et affectez-la à `f2` .

> La fonction `scale` réalise une simple mise à l'échelle linéaire de sa valeur d'entrée. (Ex: Si on appelle `scale(10)` (sans préciser `factor`), la fonction reverra $10 \times 1.2 = 12.0$

3. Implémentez une fonction `apply_all(funcs, value)` qui :
   - Prend en entrée une liste de fonctions `funcs` et une valeur `value`.
   - Retourne la liste des résultats de `func(value)` pour chaque `func` dans `funcs`.
4. Testez `apply_all([f1, f2], 10)` et commentez le comportement.

Exercice 3 : Fonctions à nombre variable d'arguments

1. Implémentez `def summarize(*args, **kwargs)` , qui retourne la somme et la moyenne de la liste `*args` selon les paramètres de `**kwargs`
   - `args` contient des nombres ; calculez leur somme et moyenne.
   - `*kwargs` contient des options :
     - `precision` (int) pour le nombre de décimales. (utiliser `round`)
     - `verbose` (bool) pour afficher un rapport détaillé.

       > L'étudiant doit chercher que veut dire **verbose** et à quoi sert-il.

2. Testez plusieurs appels :
   - `summarize(1,2,3, precision=3, verbose = False)`
   - `summarize(5,10,15, verbose=True)`
   - `summarize()` (aucun argument)

> Question : comment gérer l'absence d'arguments pour éviter une division par zéro lors du calcul de la moyenne ?

Exercice 4 : Passage d'arguments – Immutable vs Mutable

1. Écrivez deux fonctions :

```
def append_item(lst, item):
    lst.append(item)
def increment(n):
    n += 1
    return n
```

2. Dans un script, définissez `a = [1,2]` et `b = 5`, puis appelez successivement `append_item(a, 3)` et `increment(b)`.
3. Affichez `a` et `b` avant et après appels.
4. Expliquez pourquoi `a` est modifié tandis que `b` reste inchangé.
5. Proposez une version de `append_item` qui **ne modifie pas** la liste d'origine mais en renvoie une nouvelle.

Exercice 5 : Portée des variables – Locale et Globale

1. Déclarez une variable globale `counter = 0`.
2. Créez une fonction `def increase(n):` qui
   - Utilise `global counter` pour incrémenter `counter` de `n`.
   - Retourne la nouvelle valeur de `counter`.
3. Ajoutez une variable locale `counter` dans une autre fonction `def local_increase(n):` sans déclaration `global`.
4. Testez les deux fonctions et affichez la valeur de `counter` dans le code principal.
5. Analyse :
   - Quel est l'impact de `global` ?
   - Pourquoi évite-t-on en général les variables globales ?