Manipulez des objets en Python – Exercices

Les questions identifiées par (*) sont difficiles et par conséquent réservées uniquement aux étudiants ayant terminés toutes les autres questions.

1 Compétences ciblées

Savoir :

1. écrire et utiliser une fonction
2. créer et éditer des objets

2 Écrire et utiliser une fonction

1. Écrivez une procédure `table` avec quatre paramètres : `base`, `debut`, `fin`, `inc`. Cette procédure doit afficher la table des multiples de `base`, allant de `debut` à `fin`, et construits de `inc` en `inc`. Testez la procédure par un appel dans le programme principal.
2. Écrivez une fonction `cube` qui retourne le cube de son argument. Écrivez une fonction `volumeSphere` qui calcule le volume d'une sphère de rayon `r` fourni en argument et qui utilise la fonction `cube`. Testez la fonction `volumeSphere` par un appel dans le programme principal.
3. Écrire une fonction `volMasseEllipsoide` qui retrouve le volume et la masse d'une ellispsoïde grâce à un tuple. Les paramètres sont les trois demi-axes et la masse volumique. On donnera à ces quatre paramètres des valeurs par défaut. On donne : $\nu = \frac{3}{4}\pi abc$. Testez cette fonction par des appels avec différents nombres d'arguments.
4. Écrire une fonction `somme` avec un argument *tuple de longueur variable* qui calcule la somme des nombres contenus dans le tuple. Testez cette fonction par des appels avec différents tuples d'entiers ou de flottants.
5. Écrivez une autre fonction `somme` avec trois arguments, et qui renvoie leur somme. Dans le programme principal, définir un tuple de trois nombres, puis utilisez la syntaxe d'appel à la fonction qui *décompresse* le tuple. Affichez le résultat.
6. Écrivez une fonction `unDictionnaire` avec un argument *dictionnaire de longueur d'appel variable*, et qui affiche son argument. Dans le programme

principal, définir un dictionnaire, puis utilisez la syntaxe d'appel à la fonction qui *décompresse* le dictionnaire. Affichez le résultat.

3 Créer et éditer des objets

1. Définissez la liste : `liste = [17,38,10,25,72]`, puis effectuez les actions suivantes :
   — triez et affichez la liste ;
   — ajoutez l'élément 12 à la liste et affichez la liste ;
   — renversez et affichez la liste ;
   — affichez l'indice de l'élément 17 ;
   — enlevez lélément 38 et affichez la liste ;
   — affichez la sous-liste du $2e$ au $3e$ élément ;
   — affichez la sous-liste du début au $2e$ élément ;
   — affichez la sous-liste du $3e$ élément à la fin de la liste ;
   — affichez la sous-liste complète de la liste ;
   — affichiez le dernier élément en utilisant l'indiçage négatif.
   Bien remarquer que certaines méthodes de listes ne retourne rien.
2. Initialisez `truc` comme une liste vide, et `machin` comme une liste de cinq flottants nuls. Affichez ces listes. Utilisez la fonction `range()` pour afficher :
   — les entiers de 0 à 3
   — les entiers de 4 à 7
   — les entiers de 2 à 8 par pas de 2.
   Définir `chose` comme une liste des entiers de 0 à 5 et testez l'appartenance des éléments 3 à 6 à `chose`.
3. Utilisez une liste en compréhension pour ajouter 3 à chaque élément d'une liste d'entiers de 0 à 5.
4. Utilisez une liste en compréhension pour ajouter 3 à chaque élément d'une liste d'entiers de 0 à 5, mais seulement si l'élément est supérieur ou égal à 2.
5. Utilisez une liste en compréhension pour obtenir la liste `['ad', 'ae', 'bd', 'be', 'cd', 'ce']` à partir des chaînes `'abc'` et `'de'`. *Indication :* utilisez deux boucles `for` imbriquées.
6. Utilisez une liste en compréhension pour calculer la somme d'une liste d'entiers de 0 à 9.
7. Définissez deux ensembles : $X = \{a, b, c, d\}$ et $Y = \{s, b, d\}$, puis affichez les résultats suivants :
   — les ensembles initiaux ;
   — le test d'appartenance de l'élément `'c'` à X ;
   — le test d'appartenance de l'élément `'a'` à Y ;
   — les ensembles $X - Y$ et $Y - X$;
   — l'ensemble $X \cup Y$ (union) ;

— l'ensemble $X \cap Y$ (intersection).

8. Écrivez une fonction `compterMots` ayant un argument (une chaîne de caractères) et qui renvoie une *dictionnaire* qui contient la fréquence de tous les mots de la chaîne entrée.
9. Le type dictionnaire (ou tableau associatif) permet de représenter des tableaux structurés. En effet, à chaque *clé* un dictionnaire associe une *valeur*, et cette valeur peut elle-même être une structure de données (liste, tuple ou un dictionnaire ...).

   Soit le tableau suivant représentant des informations physio-chimiques sur des éléments simples (température d'ébulition (T_e) et de fusion (T_f), numéro (Z) et masse (M) atomique :

Au	T_e/T_f	2970	1063
Au	Z/A	79	196.967
Ga	T_e/T_f	2237	29.8
Ga	Z/A	31	69.72

   Affectez les données de ce tableau à un dictionnaire `dico` python de façon à pouvoir écrire par exemple :

```
>>> print dico["Au"]["Z/A"][0] # affiche : 79
```

10. (*) Implémentez une pile LIFO avec une liste. Pour cela, définir trois fonctions :
    — `pile` : qui retourne une pile à partir d'une liste variable d'éléments passés en paramètres ;
    — `empile` : empile un élément en *haut* de la pile ;
    — `depile` : dépile un élément du *haut* de la pile.
11. (*) De la même manière, implémentez une queue FIFO avec une liste. Essayez d'ajouter un menu de manipulation de la queue.