Informatique Pour Tous

TD 9 – Les dictionnaires

A/ Les numéros de téléphone (d'après Mathieu Linet, lycée Berthollet)

Les numéros de téléphone géographiques (code Z) respectent le codage suivant :
(1 : IDF), (2 : NO), (3 : NE), (4 : SE), (5 : SO).

Q1 – ⌨ Initialiser le dictionnaire nommé `regions` permettant d'associer les clés 1, 2, 3, 4 et 5 avec leurs régions respectives. On remarquera que le dictionnaire `regions` est **bijectif !**

Q2 –
a) ⌨ Ecrire une fonction `inverser(d)` qui prend en argument un dictionnaire bijectif `d` et qui renvoie le dictionnaire inversé `inv_d` dont les clés sont les valeurs du dictionnaire `d` et dont les valeurs sont les clés de `d`.
b) ⌨ Générer et afficher le dictionnaire global nommé `numero` qui associe à toute région son numéro par appel de la fonction `inverser` précédente, c'est à dire `numero=inverser(regions)`.
c) ⌨ Donner un exemple de dictionnaire pour lequel la fonction inverser n'aurait pas de sens.
d) ⌨ Compléter le code suivant permettant de tester que la fonction semble correcte :

```
ind_region='SO'
#A compléter
print(num)
```

Q3 – Pour chaque numéro de téléphone, on a récupéré le nom de la personne ainsi que l'indicatif de la région sous la forme d'un dictionnaire (déclaré en globale) :

```
personnes={'theo':1,'Philippine':4,'arthur':4,'margot':1}
```

a) ⌨ Ecrire une fonction `compter(ind_region)` qui prend en argument l'indicatif (type `int`) d'une des cinq régions et renvoie le nombre de personnes appartenant à cette région pour le dictionnaire précédent `personnes`, qui est global.
b) ⌨ Compléter le code suivant permettant de tester que la fonction semble correcte :

```
ind_region='NO'
#A compléter
print(nb_region)
```

B / Dérivée d'un polynôme

Nous avons vu qu'un polynôme peut être représenté par un dictionnaire, dont les clés sont les degrés des différents termes, et les valeurs associées sont les coefficients correspondants. Par exemple, soient les deux polynômes suivants (que vous copie-collerez dans l'espace des variables globales, pour les tests de cet exercice) :

$P_1(x) = 4$ (degré 0) → sera représenté par le dictionnaire `P1 = {0:4}`

$P_2(x) = -x^5 + 2x^3 - 3$ (degré 6) → sera représenté par le dictionnaire `P2 = {0:-3, 3:2, 5:-1}`

Remarque : nous prendrons la convention que le polynôme nul est représenté par `{}` (plus concis que `{0:0}`).

Q4 – ⌨ Sans regarder le cours, écrire une fonction `eval_poly(P, a)` qui prend en argument un polynôme `P` (codé sous forme de dictionnaire) et une valeur `a` (de type `int` ou `float`) et qui renvoie la valeur de $P(a)$.

Tests :

```
print(eval_poly(P1,3))    # doit afficher 4
print(eval_poly(P2,-1))   # doit afficher -4
print(eval_poly(P2,2))    # doit afficher -19
```

Il est donc aussi possible de représenter la dérivée d'un polynôme (dictionnaire) par un autre dictionnaire !

Q5 – ⌨ Écrire une fonction `derivee(P)` qui aura pour argument le dictionnaire `P` caractérisant un polynôme $P(X)$ et qui renverra le dictionnaire associé au polynôme $P'(X)$, représentant sa dérivée.

Tests :
```
print(derivee(P1))        # doit afficher {}    (polynôme nul)
print(derivee(P2))        # doit afficher {2: 6, 4: -5} car P2'(x) = -5x^4 + 6x^2
```

Q6 – ⌨ Écrire alors une fonction `deriv_k(P,k)` qui aura pour argument le dictionnaire `P` caractérisant un polynôme $P(x)$ et qui renverra le dictionnaire associé au polynôme $P^{(k)}(x)$, représentant sa dérivée $k^{ième}$.

Remarque : cette fonction s'appuiera sur la fonction auxiliaire `derivee()` codée en `derivee(P)`.

Contrainte : attention, cette fonction ne devra pas modifier le polynôme *P* en argument, à portée globale. Vous penserez donc à faire une copie, si nécessaire ! Pour vérifier, après l'appel de votre fonction, affichez la valeur du polynôme `P1` ou `P2` sur lequel vous avez fait le test, et vérifiez qu'il n'a pas été modifié globalement par l'appel.

Tests :
```
print(deriv_k(P1,0))      # affichera {0:4} (non dérivé, donc inchangé)
print(deriv_k(P2,2))      # affichera {1:12, 3:-20} car P2''(x) = -20x^3 + 12x
print(deriv_k(P2,3))      # affichera {0:12, 2:-60} car P2^(3)(x) = -60x^2 + 12
print(P2)                 # doit afficher {0: -3, 3: 2, 5: -1}, non modifié
```

C/ Comptage des occurrences d'un élément dans une liste

La fonction `Counter(L)`, de la bibliothèque `collections` permet de générer un dictionnaire, qui reprend les éléments distincts de la liste, et compte le nombre d'occurrence de chacun de ces éléments. Les développeurs utilisent beaucoup ce genre de fonctions. Deux exemples ci-dessous :

Nous disposons de la base de données des recettes d'un vendeur de marché, en fonction de la semaine, et nous souhaitons savoir combien de fois il a touché 100€ ou plus.

```
>>> import collections
>>> ventes =[0, 100, 100, 80, 70, 80, 20, 10, 100, 100, 80, 70, 10, 30, 40]
>>> print(collections.Counter(ventes))
Counter({100: 4, 80: 3, 70: 2, 10: 2, 0: 1, 20: 1, 30: 1, 40: 1})
# La réponse est donc 4
```

Nous disposons d'un texte, et on souhaite connaître le nombre de fois où chaque lettre est utilisée dans ce texte (c'est, entre autres, ce qui permet ensuite de calculer la probabilité de rencontrer une lettre, etc).

```
>>> texte = "ceci est une phrase purement arbitraire"
>>> liste_correspondante = list(texte)
# Ca donne ["c","e","c","i"," ","e","s",…]

>>> print(collections.Counter(liste_correspondante))
Counter({'e': 7, ' ': 5, 'r': 5, 'i': 3, 't': 3, 'a': 3, 'c': 2, 's': 2, 'u': 2, 'n': 2, 'p': 2, 'h': 1, 'm': 1, 'b': 1})
# On voit sans surprise que les consonnes, particulièrement "e", sont très représentées.
```

On souhaite ici écrire une fonction `comptage(L)` qui réalise à peu près la même opération. Cette fonction renverra un dictionnaire contenant toutes les occurrences des éléments distincts de `L`. A la différence de `Counter(L)`, nous ne nous intéresserons **pas** ici au classement par ordre décroissant de ces éléments.

Pour le dictionnaire en sortie, chaque **clé** sera un élément distinct de `L`, et la valeur associée sera son nombre d'occurrences dans `L`. Pour ce faire, l'algorithme à coder est le suivant :

✓ Nous allons déclarer un dictionnaire vide Rappel : ceci s'écrit `dico = {}`

✓ Nous allons parcourir la liste `L`, et à chaque élément rencontré :

- Si cet élément correspond déjà à une clé, on ajoute 1 au nombre d'occurrences de cet élément Rappel : un dictionnaire n'est pas un objet itérable. Pour savoir si une clé existe ou non dans un dictionnaire, on écrira : **`if cle in dico`** ou **`if cle not in dico :`**
- Sinon, on crée une nouvelle clé dans le dictionnaire, initialisée avec la bonne valeur (je vous laisse réfléchir).

Rappel : pour ajouter une clé et lui associer une valeur, on écrit : `dico[clé] = valeur`

Q7 – Écrire la fonction `comptage(L)` décrite ci-dessus.

Q8 – Tester votre fonction avec les deux listes décrites précédemment :

```
ventes =[0, 100, 100, 80, 70, 80, 20, 10, 100, 100, 80, 70, 10, 30, 40]
texte = list("ceci est une phrase purement arbitraire")
```

L'affichage (`print`) du dictionnaire renvoyé par le *comptage* devrait vous donner :

```
{0: 1, 100: 4, 80: 3, 70: 2, 20: 1, 10: 2, 30: 1, 40: 1}
{'c': 2, 'e': 7,'i': 3,' ': 5,'s': 2,'t': 3,'u': 2,'n': 2,'p': 2,'h': 1,...
'r': 5, 'a': 3,'m': 1,'b': 1}
```