♦ TP6 – Dictionnaires

I. — Définition

Un dictionnaire est une structure ayant son propre type (`dict`) et qui représente un ensemble (non ordonné) de couples (clé,valeur) saisis sous la forme `clé:valeur`. Un dictionnaire est délimité par des accolades et ses éléments (qui sont des couples) sont séparés par des virgules. La longueur d'un dictionnaire est le nombre d'entrées (i.e. de couples clé/valeur) et, comme pour les listes, on peut utiliser la fonction `len` pour connaître cette longueur.

Dans un dictionnaire, les clés sont des objets de types non mutables (des nombres, des chaînes de caractères, des booléens mais pas des listes ni des dictionnaires) et les valeurs peuvent être de types quelconques (y compris des dictionnaires). Dans un même dictionnaire, les différentes clés peuvent être de types différents (et de même pour les valeurs).

Par exemple, l'instruction suivante définit un dictionnaire `dico` :

```
dico = {'abc':5, 2:True, 3.4:[1,2,3]}
```

Ce dictionnaire possède trois clés : la chaîne de caractères `'abc'`, l'entier `2` et le flottant `3.4`. La valeur associée à la clé `'abc'` est l'entier `5`, la valeur associée à la clé `2` et le booléen `True` et la valeur associée à la clé `3.4` est la liste `[1,2,3]`. La longueur de ce dictionnaire est 3 (puisqu'il y a 3 couples) donc `len(dico)` renvoie `3`.

Le dictionnaire vide, qui ne contient aucun couple `clé:valeur`, se définit par `{}`.

On utilise un dictionnaire lorsqu'on veut regrouper des éléments qui partagent une caractéristique commune (la clé) à laquelle on veut associer une donnée (la valeur). Un exemple typique d'utilisation d'un dictionnaire est le suivant. On considère la chaîne de caractères

```
ch='AGCTTACGATACTT'
```

et on aimerait avoir accès au nombre d'occurrences de chaque caractère de cette chaîne. Une façon de stocker ces données est d'utiliser le dictionnaire suivant :

```
nb_occur = {'A':4, 'C':3, 'G':2, 'T':5}.
```

Dans cet exemple, on a défini le dictionnaire « en extension » c'est-à-dire en saisissant explicitement tous les couples clé/valeur. Il est également possible de définir un dictionnaire « en compréhension » (comme on peut le faire pour les listes). Par exemple, si on considère liste de chaînes de caractères

```
L = ["CCT", "ACGTTA", "ATTCTGA", "AG"]
```

on peut définir un dictionnaire qui associe à chaque chaîne sa longueur par la syntaxe suivante

```
{ chaine:len(chaine) for chaine in L }.
```

Dans un dictionnaire, on s'intéresse, en général, à la valeur associée à une clé donnée. Contrairement aux cas des listes ou des chaînes de caractères, ceci ne peut pas se faire à l'aide d'un indice car un dictionnaire n'est pas ordonné.

Si `dic` est un dictionnaire et si `c` est une clé de ce dictionnaire alors on peut obtenir la valeur associée à cette clé grâce à l'instruction suivante :

```
dic[c]
```

Ainsi, si on veut savoir quelle est la valeur associée à la clé `'T'` dans le dictionnaire `nb_occur`, on écrit

```
nb_occur['T']
```

Remarque. Ainsi, une entrée (clé/valeur) dans un dictionnaire est identifiée par sa clé ; il est donc nécessaire, pour pouvoir déterminer chaque entrée de manière unique, que toutes les clés soient différentes (contrairement aux valeurs qui peuvent se répéter).

Question 1. On considère le dictionnaire `dic1` suivant :

`dic1 = {'a':2, 2:'b', 'c':3}`

Déterminer ce que renvoient les instructions suivantes

1) `type(dic1)` **2)** `len(dic1)` **3)** `dic1[2]` **4)** `dic1[1]` **5)** `dic1['b']` **6)** `type(dic1['a'])`

Vérifier à l'aide de l'ordinateur.

Solution.

1) `type(dic1)` renvoie `<class 'dict'>` ce qui signifie que `dic1` est de type `dict`.
2) `len(dic1)` renvoie `3` car le dictionnaire contient trois entrées.
3) `dic1[2]` renvoie `'b'` car la valeur associée à la clé `2` est `'b'`.
4) `dic1[1]` renvoie un message d'erreur car `1` n'est pas une clé de `dic1`.
4) `dic1['b']` renvoie un message d'erreur car `'b'` n'est pas une clé de `dic1`.
6) `type(dic1['a'])` renvoie `int` puisque `dic1['a']` renvoie `2` qui est de type entier.

Question 2.

1. Écrire une fonction `nb_occurences` qui prend en argument une chaînes de caractères `ch` et un caractère `car` et qui renvoie le nombre d'occurrences de `car` dans `ch`.
2. On considère la liste suivante :

   `L = ['aab', 'adea', 'de', 'aaaaa', 'teaaaza', 'zqtad'].`

   À l'aide de la fonction `nb_occurences`, définir un dictionnaire `D` dont les clés sont les chaînes de caractères de `L` et les valeurs associées sont les nombres de 'a' dans ces chaînes.
3. Est-il possible d'écrire un dictionnaire identique si `L` est une liste de listes plutôt qu'une liste de chaîne de caractères ?

Solution.

1.

```
def nb_occurrences(ch,car):
  c = 0
  for e in ch:
    if e == car:
      c += 1
  return c
```

2.

```
D = { ch:nb_occurrences(ch,'a') for ch in L}
```

3. Il n'est pas possible de définir un dictionnaire identique pour une liste de listes car les clés d'un dictionnaire ne peuvent pas être des objets mutables (et donc ne peuvent pas être des listes).

II. — Modification d'un dictionnaire

1) Modification d'une valeur

Un dictionnaire est un objet mutable (on peut modifier les valeurs ou ajouter des couples clé/valeur).

Reprenons le dictionnaire `nb_occur` qui donne le nombre d'occurrences de chaque lettre de la chaîne de caractères `ch='AGCTTACGATACTT'`. Si on modifie la chaîne `ch` par la syntaxe `ch += 'G'` alors on obtient la chaîne `ch='AGCTTACGATACTTG'` et pour actualiser le dictionnaire `nb_occur`, sans le reconstruire complètement, on peut simplement modifier la valeur de la clé `'G'` grâce à l'instruction

```
nb_occur['G'] += 1
```

qui va ajouter 1 à la valeur associée à la clé `'G'`. Ainsi, le dictionnaire `nb_occur` devient

`nb_occur = {'A':4, 'C':3, 'G':3, 'T':5}`

Remarque. C'est là tout l'intérêt d'un dictionnaire. On pourrait en effet se dire qu'on peut remplacer le dictionnaire `nb_occur` initial par la liste de listes

`L = [['A',4], ['C',3], ['G',2], ['T',5]].`

Cependant, si on veut changer le nombre associé à `'G'` et l'augmenter de 1, il faut savoir que ce nombre est stocké dans la quatrième liste et utiliser l'instruction `L[3][2] += 1` alors que, pour un dictionnaire, il suffit d'utiliser la clé `'G'` sans se préoccuper de la place de cette clé (un dictionnaire n'étant de toute façon pas ordonné).

Question 3. Quelle instruction doit-on écrire pour adapter le dictionnaire `nb_occur` si on modifie la chaîne `ch` par la syntaxe `ch += 'T'` ?

Solution.

```
nb_occur['T'] += 1
```

2) Test d'appartenance

Pour tester si une clé `c` est présente dans un dictionnaire `dic`, on peut utiliser la syntaxe suivante : `c in dic`. Celle-ci renvoie un booléen. Ainsi, `'A' in nb_occur` renvoie `True` alors que `'B' in nb_occur` renvoie `False`.

Question 4. On considère le dictionnaire `dico` défini dans le paragraphe I. Que renvoient les instructions suivantes ?

1) `'abc' in dico` 2) `True in dico` 3) `3.4 in dico` 4) `'ab' in dico`.

Solution.

1) `'abc' in dico` renvoie `True` car `'abc'` est une clé de `dico`.
2) `True in dico` renvoie `False` car `True` n'est pas une clé de `dico`.
3) `3.4 in dico` renvoie `True` car `3.4` est une clé de `dico`.
4) `'ab' in dico` renvoie `False` car `'ab'` n'est pas une clé de `dico`.

3) Ajout et suppression d'un couple clé/valeur

Il est possible d'ajouter un couple clé/valeur à un dictionnaire. Précisément, pour ajouter une clé `c` à un dictionnaire `dic` en lui attribuant la valeur `v`, on utilise la syntaxe

```
dic[c]=v
```

Remarque. Ainsi, si `dic` est un dictionnaire et si `c` est un objet alors

- soit `c` est une clé déjà existante dans `dic` et, dans ce cas, l'instruction `dic[c]=v` modifie la valeur associée à la clé `c` qui est remplacée par `v` ;
- soit `c` n'est pas une clé présente dans `dic` et, dans ce cas, l'instruction `dic[c]=v` ajoute la clé `c` au dictionnaire `dic` en lui attribuant la valeur `v`.

De même, il est possible de supprimer un couple clé/valeur d'un dictionnaire. Précisément, pour supprimer la clé `c` (et sa valeur `v`) d'un dictionnaire `dic`, on utilise la syntaxe

```
del dic[c]
```

Question 5. On considère les instructions suivantes :

```
1 dico = { 'Pierre': 15, 'Paul':12, 'Jacques':18, 'Rémi':7,
      'Jean': 11, 'Marc':5 }
2 dico['Pierre']-=1
3 dico['Michel']=8
4 del dico['Rémi']
```

Donner l'état du dictionnaire `dico` après l'exécution de la ligne 2, après l'exécution de la ligne 3 et après l'exécution de la ligne 4 puis vérifier à l'aide de l'ordinateur.

Solution.
Après l'exécution de la ligne 2, **dico** vaut

```
{ 'Pierre': 14, 'Paul':12, 'Jacques':18, 'Rémi':7, 'Jean': 11, 'Marc':5 }
```

Après l'exécution de la ligne 3, **dico** vaut

```
{ 'Pierre': 14, 'Paul':12, 'Jacques':18, 'Rémi':7, 'Jean': 11, 'Marc':5,
                                                    'Michel':8 }
```

Après l'exécution de la ligne 4, **dico** vaut

```
{ 'Pierre': 14, 'Paul':12, 'Jacques':18, 'Jean': 11, 'Marc':5, 'Michel':8 }
```

Question 6. Écrire une fonction `nb_occ` qui prend en argument une chaîne de caractères `ch` et qui renvoie le dictionnaire dont les clés sont les caractères qui composent la chaîne `ch` et les valeurs associées sont les nombres d'occurrences de ces caractères dans la chaîne `ch`.
Vérifier que l'instruction `nb_occ('ATTENTION')` renvoie le dictionnaire

```
{ 'A':1, 'T':3, 'E':1, 'N':2, 'I':1, 'O':1 }.
```

Solution

```
def nb_occ(ch):
  dic = {}
  for e in ch:
    if e in dic:
      dic[e] += 1
    else:
      dic[e] = 1
  return dic
```

III. — Itération sur un dictionnaire

Comme les listes, les dictionnaires sont itérables (c'est-à-dire, on peut les parcourir selon leurs clés et/ou leurs valeurs). Voici quelques exemples de syntaxes qu'on peut utiliser et adapter en fonction du contexte et des besoins.

Supposons qu'on dispose d'un dictionnaire `dic`.

- Si on souhaite obtenir la liste des clés de `dic`, on peut utiliser l'une des deux syntaxes suivantes :

```
[ c for c in dic ]
```

ou

```
[ c for c in dic.keys() ]
```

- Si on souhaite obtenir la liste des valeurs de `dic`, on peut utiliser l'une des deux syntaxes suivantes :

```
[ dic[c] for c in dic ]
```

ou

```
[ v for v in dic.values() ]
```

- Si on souhaite obtenir la liste des couples (clé, valeur) de `dic`, on peut utiliser l'une des deux syntaxes suivantes :

```
[ (c,dic[c]) for c in dic ]
```

ou

```
[ i for i in dic.items() ]
```

Question 7. En utilisant la fonction `nb_occ` de la question 6, écrire une fonction `caracteres` qui prend en argument une chaîne de caractères `ch` et qui renvoie la liste des caractères distincts qui apparaissent dans `ch`.
Vérifier que l'instruction `caracteres('ananas')` renvoie la liste `[ 'a', 'n', 's' ]`.

Solution.

```
def caracteres(ch):
    return [c for c in nb_occ(ch)]
```

IV. — Exercices

Exercice 1. On considère le dictionnaire suivant qui donne la masse en kg de certains animaux :

`masse = {'girafe':1100, 'tigre':250, 'singe':70}`.

1. Prédire puis vérifier à l'aide de l'ordinateur ce que renvoie les commandes suivantes.

   `type(masse)` `len(masse)` `masse[1]` `masse['tigre']` `masse['souris']`.

2. **a.** On exécute l'instruction `masse['souris'] = 0.02`. Que vaut à présent `len(masse)` ? Vérifier à l'aide de l'ordinateur.
   b. On exécute l'instruction `masse['souris'] = 0.03`. Que vaut à présent `len(masse)` ? Vérifier à l'aide de l'ordinateur.
3. En utilisant une itération sur les clés, créer un dictionnaire `masse_gramme` qui donne la masse en gramme (et non pas en kg) des animaux du dictionnaire `masse`.

Solution.

1. `type(masse)` renvoie `<class 'dict'>` ce qui signifie que `masse` est de type `dict`
   `len(masse)` renvoie 3, `masse[1]` renvoie un message d'erreur car 1 n'est pas une clé de `masse`
   `masse['tigre']` renvoie 250
   `masse['souris']` renvoie un message d'erreur puisque `'souris'` n'est pas une clé du dictionnaire `masse`.
2. **a.** On ajoute une clé au dictionnaire donc `len(masse)` renvoie 4.
   b. On modifie une valeur mais on ne change pas les clés du dictionnaire donc `len(masse)` renvoie encore 4.
3. On peut créer le dictionnaire des masses moyennes en gramme par la syntaxe suivante :

```
masse_gramme = { e:masse[e]*1000 for e in masse }
```

Exercice 2. Les syntaxes suivantes sont-elles correctes ?

1. `dic1 = { "a":[1,2,3], "b":3, "c":True }`.
2. `dic2 = { [1,2,3]:"a", 3:"b", True:"c" }`.
3. `dic3 = { 1:"a", 2:{ 1:"a", "a":[1,2] }, 4:4 }`.

Solution. Les syntaxes de `dic1` et `dic3` sont correctes mais pas celle de `dic2` car une liste ne peut pas être une clé dans un dictionnaire.

Exercice 3.

1. Écrire en extension le dictionnaire `D1 = { k:2**k for k in range(5) }`.
2. Écrire en compréhension le dictionnaire `D2 = {0:1, 1:4, 2:7, 3:10, 4:13, 5:16}`.

Solution.

1. `D1 = { 0:1, 1:2, 2:4, 3:8, 4:16 }`.
2. `D2 = { 3k+1 for k in range(6) }`.

Exercice 4. On considère une séquence d'acides aminés sous la forme d'une chaîne de caractères, comme par exemple :

`seq_ex = 'AGWPSGGASAGLAILWGASAIMPGALW'.`

1. Écrire une fonction `nb_occ_bis` qui prend en argument une chaîne de caractères `seq` représentant une séquence d'acides aminés et qui renvoie, sous forme d'un dictionnaire, le nombre d'occurrences de chaque acide aminé présent dans cette séquence. Vérifier que `nb_occ_bis(seq_ex)` renvoie

   `{ 'A':7, 'G':6, 'W':3, 'P':2, 'S':3, 'L':3, 'I':2, 'M':1 }`

2. En utilisant la fonction `nb_occ_bis`, écrire une fonction `aa_presents` qui prend en argument une chaîne de caractères `seq` représentant une séquence d'acides aminés et qui renvoie la liste des acides aminés présents dans cette séquence (chaque acide aminé ne devant apparaître qu'une seule fois dans la liste).

Solution.

1.

```
def nb_occ(seq):
  dic = {}
  for e in seq:
    if e in dic:
      dic[e]+=1
    else:
      dic[e]=1
  return dic
```

2.

```
def aa_present(seq):
  return [e for e in nb_occ(seq)]
```

Exercice 5. On représente une recette de cuisine par un dictionnaire dont les clés sont les ingrédients et les valeur sont les quantités de chaque ingrédient (l'unité est variable selon l'ingrédient : gramme, millilitre, nombre). Par exemple, la recette des crêpes est représentée par le dictionnaire

`crepes = {"farine":250, "oeufs":4, "lait":300, "beurre":50, "sucre":30}.`

1. Écrire une fonction `nombre_ingredients` prenant en argument un dictionnaire `D` représentant une recette et qui renvoie le nombre d'ingrédients différents de la recette.
2. Supposons qu'on soit allergique aux noix. Écrire une fonction `est_sans_noix` prenant en argument un dictionnaire représentant une recette et qui renvoie `True` si la recette ne contient pas de noix, et `False` sinon.
3. On souhaite faire un régime. Écrire une fonction `est_sain` prenant en argument un dictionnaire représentant une recette et qui renvoie `True` si la recette contient moins de 50 grammes de sucre, et `False` sinon. On prendra garde au fait qu'il y a deux situations possibles : ou bien `"sucre"` est une clé du dictionnaire dont la valeur doit être inférieure à 50, ou bien `"sucre"` n'est pas une clé du dictionnaire.

Solution.

1.

```
def nombre_ingredients(D):
  return len(D)
```

2.

```
def est_sans_noix(D):
  return not("noix" in D)
```

3. Version courte

```
def est_sain(D):
  return not("sucre" in D and D["sucre"] > 50)
```

Version longue

```
def est_sain(D):
  if "sucre" in D and D["sucre"] > 50:
    return False
  return True
```

Exercice 6. On représente un porte-monnaie contenant des pièces et des billets par un dictionnaire `dic`, où `d[x]` représente le nombre de billets (ou pièces) de `x` euro. Par exemple, le dictionnaire `D = { 1:4, 2:7, 10:1 }` représente un porte-monnaie avec 4 pièces de 1 euro, 7 pièces de 2 euros et 1 billet de 10 euros. Dans cet exemple, la somme totale est de 28 euros.

Écrire une fonction `total` qui prend en argument un tel dictionnaire `D` représentant un porte-monnaie et qui renvoie la somme d'argent totale que cela représente.

Solution.

```
def total(D):
  S = 0
  for c in D:
    S += c*D[c]
  return S
```

Exercice 7.

1. Écrire une fonction `conversionDL` qui prend en argument un dictionnaire `D` et renvoie la liste des listes [clé,valeur]. Par exemple, si `D={"a":1, "b":2, "c":3}` alors `conversionDL(D)` doit renvoyer `[["a",1], ["b",2], ["c",3]]`.
2. Écrire une fonction `conversionLD` qui prend en arguments deux listes de même longueur (une liste `cles` de « clés » et une liste `valeurs` de « valeurs ») et renvoie le dictionnaire correspondant. Par exemple, si `cles=["a","b","c"]` et si `valeurs=[1,2,3]` alors l'instruction `conversionDL(cles,valeurs)` doit renvoyer `{"a":1, "b":2, "c":3}`.

Solution.

1.

```
def conversionDL(D):
  L = []
  for c in D:
    L.append([c,D[c]])
  return L
```

2.

```
def conversionLD(cles,valeurs):
  n = len(cles)
  dic = {}
  for i in range(n):
    dic[cles[i]] = valeurs[i]
  return dic
```

Exercice 8. Écrire une fonction `changeCle` prenant en argument un dictionnaire `dic` et deux objets `a` et `b` et qui renvoie le dictionnaire obtenu à partir de `dic` en modifiant la clé `a` en `b` lorsque `a` est une clé de `dic` et qui renvoie `dic` sinon. Ainsi, si `dic={"a":1, "b":2, "c":3}` alors `changeCle(dic, "b", "r")` doit renvoyer le dictionnaire `{"a":1, "r":2, "c":3}` alors que `changeCle(dic, "d", "r")` doit renvoyer `{"a":1, "b":2, "c":3}`.

Solution.

```
def changeCle(dic,a,b):
  if a in dic:
    dic[b] = dic[a]
    del dic[a]
  return dic
```

Exercice 9. Écrire une fonction `recherche` prenant en argument un dictionnaire `D` et un objet `x` et qui renvoie la liste de toutes les clés du dictionnaire `D` associées à la valeur `x`. Si aucune clé n'est associée à `x`, la fonction renvoie la liste vide.

Solution.

```
def recherche(D, x):
  L = []
  for c in D:
    if D[c] == x:
      L.append(c)
  return L
```

Autre possibilité :

```
def recherche(D, x):
  return [c for c in D if D[c] == x]
```

Exercice 10. Écrire une fonction `frequenceLettres` ayant pour argument une chaîne de caractères `ch` et qui renvoie un dictionnaire qui contient la fréquence de toutes les lettres de la chaîne `ch`. Par exemple, `frequenceLettres('aabtetetaa')` doit renvoyer `{'a':0.4, 'b':0.1, 't':0.3, 'e':0.2}`.

Solution.

```
def frequenceLettres(ch):
  n = len(ch)
  D = {}
  for c in ch:
    if c in D:
      D[c] += 1/n
    else:
      D[c] = 1/n
  return D
```

Exercice 11. Écrire une fonction `renverse` qui prend en argument un dictionnaire `D` et qui qui renvoie un dictionnaire dont les clés sont les valeurs de `D` et dont les valeurs sont les clés de `D`. Dans le cas où certaines clés de `D` sont associées à une même valeur, le dictionnaire renvoyé doit regrouper ces clés dans une liste. Ainsi, si `D = {"a":1, "b":2, "c":3, "d":4}` alors `renverse(D)` doit renvoyer `{1:"a", 2:"b", 3:"c", 4:"d"}` mais si `D = {"a":1, "b":2, "c":1, "d":4}` alors `renverse(D)` doit renvoyer `{1:["a","c"], 2:"b", 4:"d"}`.

Solution.

```
def renverse(D):
  dic_renv = {}
  for c in D:
    if D[c] in dic_renv:
      if type(dic_renv[D[c]]) == list:
        dic_renv[D[c]].append(c)
      else:
        dic_renv[D[c]] = [dic_renv[D[c]], c]
    else:
      dic_renv[D[c]] = c
  return dic_renv
```

Exercice 12. Écrire une fonction `fusion` qui prend en arguments deux dictionnaires `D1` et `D2` dont les valeurs sont des nombres et qui renvoie le dictionnaire obtenu en réunissant les entrées de `D1` et celles de `D2`. Quand une même clé est présente dans les deux dictionnaires, on somme les valeurs associées. Ainsi, si `D1={'a':1, 'b':2}` et `D2={'b':7, 'c':3}` alors `fusion(D1,D2)` doit renvoyer le dictionnaire `{'a':1, 'b':9, 'c':3}`.

Solution.

```
def fusion(D1,D2):
  dic_fusion = {c:D1[c] for c in D1}
  for c in D2:
    if c in dic_fusion:
      dic_fusion[c] += D2[c]
    else:
      dic_fusion[c] = D2[c]
  return dic_fusion
```

Exercice 13.

1. Écrire une fonction `chaines_2_lettres` qui prend en argument une chaîne de caractères `ch` et qui renvoie un dictionnaire dont les clés sont les chaînes de caractères de deux lettres consécutives apparaissant dans `ch` et les valeurs sont les nombres d'occurrences de ces chaînes.
   Vérifier que `chaine_2_lettres("ACCTAGCCCTA")` renvoie le dictionnaire

   `{"AC":1, "CC":3, "CT":2, "TA":2, "AG":1, "GC":1}`.

2. En déduire une fonction `sous_chaines_2_lettres` qui prend en argument une chaîne de caractères `ch` et qui renvoie la liste des chaines de caractères distinctes formées de deux lettres consécutives apparaissant dans `ch`.
   Vérifier que `sous_chaine_2_lettres("ACCTAGCCCTA")` renvoie la liste

   `["AC", "CC", "CT", "TA", "AG", "GC"]`.

Solution.

1.

```
def chaines_2_lettres(ch):
  dic = {}
  for k in range(len(ch)-1):
    if ch[k:k+2] in dic:
      dic[ch[k:k+2]] += 1
    else:
      dic[ch[k:k+2]] = 1
  return dic
```

2.

```
def sous_chaines_2_lettres(ch):
  return [ c for c in chaines_2_lettres(ch) ]
```