dictionnaires0

September 6, 2026

1 Les dictionnaires

Un dictionnaire est une collection de paires *clé : valeur*. Les clés sont des chaînes caractères ou des entiers. Contrairement aux listes, il n'y a aucune notion d'ordre dans un dictionnaire.

[1]:
```
dico1 = {'France':'Paris','Allemagne':'Berlin','Espagne':'Madrid'} # crée un␣
↪dictionnaire avec les clés 'France', 'Allemagne' et 'Espagne'
```

[2]:
```
dico1['France'] # affiche la valeur de la clé 'France'
```

[2]:
```
'Paris'
```

[3]:
```
dico1['Allemagne']
```

[3]:
```
'Berlin'
```

[4]:
```
len(dico1) # affiche la longueur du dictionnaire, c'est-à-dire le nombre de clés
```

[4]:
```
3
```

On crée un dictionnaire vide D par la commande:

[5]:
```
dico2=dict() # ou bien D={}
```

Pour ajouter un élément à un dictionnaire, on donne la clé et la valeur par une simple affectation.

[6]:
```
dico2['un']='one'
dico2['deux']='two'
print(dico2)
```

```
{'un': 'one', 'deux': 'two'}
```

On supprime un élément avec l'instruction *del*

[10]:
```
del(dico2['deux'])
print(dico2)
```

```
{'un': 'one'}
```

Quelques méthodes pour travailler avec des dictionnaires~:

— *clear()* : supprime tous les éléments.

— *values()* : renvoie une séquence des valeurs.

— *keys()* : renvoie une séquence des clés.

— *items()* : renvoie une séquence de tuples sour la forme *(clé,valeur)*

— *pop(clé)* : supprime l'élément de la clé en argument et renvoie sa valeur.

[7]:
```
dico={'a':1,'b':2,'c':3}
dico.values()
```

[7]:
```
dict_values([1, 2, 3])
```

Cette séquence peut être converties en liste avec l'instruction *list*:

[8]:
```
list(dico.values())
```

[8]:
```
[1, 2, 3]
```

1.0.1 Exercice 1

Ecrire une fonction qui prend en argument un dictionnaire et qui renvoie une liste qui contient les clés du dictionnaire et une liste qui contient ses valeurs.

[9]:
```
def ex1(dico):
    ...
```

[ ]:
```
ex1(dico1)
```

Les clés sont considérés comme les éléments du dictionnaire.

[25]:
```
dico={'a':1,'b':2,'c':3}
print('c' in dico) # print('c' in dico.keys())
print('d' in dico) # print('d' in dico.keys())
```

```
True
False
```

La méthode *keys()* sera donc assez peu utilisée.

1.0.2 Exerice 2

Ecrire une fonction qui prend en argument un dictionnaire et qui renvoie la liste de ses valeurs sans utiliser *values()* .

[11]:
```
def ex2(dico):
   ...
```

[12]:
```
ex2(dico1)
```

[12]:
```
['Paris', 'Berlin', 'Madrid']
```

1.0.3 Exercice 3

Soit *dico* un dictionnaire contenant un stock d'articles. La clé est le nom de l'article, la valeur est une liste contenant le prix et la quantité de l'article.

Ecrire une fonction qui renvoie la valeur du stock et le nombre d'articles sous forme de tuple.

[1]:
```
def ex3(dico):
    ...
```

[2]:
```
dico3={'stylo billes (lot de 10)':[6,50],
   'carnet de notes (A5, 100 pages)':[4.50,25],
   'classeur':[3.25,20],
   'agrafeuse':[8.95,30],
   'surligneur':[0.90,100],
   'post-it (bloc de 100)':[4.95,60]
  }
print(ex3(dico3))
```

```
(1133.0, 285)
```

1.0.4 Exercice 4

Ecrire une fonction qui prend en argument un dictionnaire dont toutes les valeurs sont des chaînes de caractères distinctes et qui renvoie le dictionnaire inversé (les clés de l'un sont les valeurs de l'autre).

[18]:
```
def ex4(dico):
    ...
```

[20]:
```
print(dico1)
ex4(dico1)
```

```
{'France': 'Paris', 'Allemagne': 'Berlin', 'Espagne': 'Madrid'}
```

[20]:
```
{'Paris': 'France', 'Berlin': 'Allemagne', 'Madrid': 'Espagne'}
```

1.0.5 Exercice 5

Ecrire une fonction *frequence(ch)* qui prend en paramètre une chaîne de caractères *ch* et qui renvoie un dictionnaire dont les clés sont les caractères de la chaîne *ch* et les valeurs sont les fréquences de la clé dans la chaîne *ch*.

[21]:
```
def frequence(ch):
    ...
```

[22]:
```
frequence('dictionnaire')
```

[22]:
```
{'d': 1, 'i': 3, 'c': 1, 't': 1, 'o': 1, 'n': 2, 'a': 1, 'r': 1, 'e': 1}
```

1.0.6 Exercice 6

1) Ecrire une fonction *mots(texte)* qui recherche tous les mots d'un texte et qui construit un dictionnaire. Les clés de ce dictionnaire sont les mots du texte. A chaque mot, le dictionnaire associe le nombre de fois qu'il apparaît.
2) Tester la fonction avec le teste contenu dans le fichier *recherche_mots.txt*. Afficher le nombre de mots distincts, ainsi que les mots qui apparaissent plus de 50 fois.

On pourra utiliser la méthode *split()* qui découpe une chaîne en liste. On peut préciser un séparateur, par défaut le séparateur est l'espace. Par exemple :

[30]:
```
A='Longtemps, je me suis couché de bonne heure'
A.split()
```

[30]:
```
['Longtemps,', 'je', 'me', 'suis', 'couché', 'de', 'bonne', 'heure']
```

[30]:
```
def mots(texte):
    ...
```

[39]:
```
fichier = open("recherche_mots.txt", "r")
texte=fichier.readline()
dico=mots(texte)
...
```

```
nombre de mots différents:  940
```

[39]:
```
[('de', 108), ('à', 56), ('le', 58), ('la', 75), ('et', 55)]
```

1.0.7 Exercice 7

On considère Ω un ensemble dont les éléments sont stockés dans la liste Ω et A une partie de Ω.

On dit qu'un dictionnaire représente une partie de Ω si l'ensemble de ses clés correspond à Ω et ses valeurs sont des booléens. Par exemple si $\Omega = \{1,2,3,4,5,6,7,8,9\}$ et $A = \{2,3,5,7\}$, le dictionnaire représentant A est:

[ ]:
```
{'1':False, '2':True, '3':True, '4':False, '5':True, '6':False, '7':False, '8':
↪False, '9':False}
```

1) Ecrire une fonction *cree_dico(A,omega)* prenant en argument la liste des éléments de A et qui renvoie un dictionnaire dont les clés sont les éléments de Ω et les valeurs *True* ou *False* suivant que la clé appartient ou non à A.

[40]:
```
def cree_dico(A,omega):
    ...
```

[42]:
```
cree_dico([2,3,5,7],[1,2,3,4,5,6,7,8,9])
```

[42]:
```
{1: False,
 2: True,
 3: True,
```

```
4: False,
5: True,
6: False,
7: True,
8: False,
9: False}
```

2) Ecrire une fonction *liste* prenant en argument un dictionnaire représentant une partie de Ω et qui renvoie la liste des éléments de Ω et la liste des éléments de la partie de Ω correspondant au dictionnaire.

[48]:
```
def liste(dico):
    ...
```

[43]:
```
liste({1: False, 2: True, 3: True, 4: False, 5: True, 6: False, 7: True, 8:␣
↪False, 9: False})
```

[43]:
```
([2, 3, 5, 7], [1, 2, 3, 4, 5, 6, 7, 8, 9])
```

3) Ecrire des fonctions *union(dicA,dicB)* et *intersection(dicA,dicB)* dont les arguments sont des dictionnaires représentant des parties A et B de Ω et qui renvoient respectivement les dictionnaires correspondant à $A \cup B$ et $A \cap B$.

[53]:
```
def union(dicA,dicB):
    ...
```

[52]:
```
def intersection(dicA,dicB):
    ...
```

[54]:
```
omega=[1,2,3,4,5,6,7,8,9]
A=[2,3,5,7]
B=[2,4,6,8]
dicA=cree_dico(A,omega)
dicB=cree_dico(B,omega)
dicU=union(dicA,dicB)
dicI=intersection(dicA,dicB)
print(liste(dicU))
print(liste(dicI))
```

```
([2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 9])
([2], [1, 2, 3, 4, 5, 6, 7, 8, 9])
```