# TP 3 : Dictionnaires

On présente dans ce TP le type `dict` des dictionnaires.

## I. Dictionnaires de données

### I.1. Le type `dict`

- Dans les TP précédents, nous avons utilisé des *listes* pour regrouper des données de types élémentaires (`int`, `float`, `bool`). On dit que la liste est un type **construit** (ou séquentiel). L'accès à l'un de ses éléments de fait par son *indice*.
- En **Python**, un dictionnaire est un autre type construit. Il représente un ensemble d'association `clé: valeur`, chaque clé étant unique.
  À la différence des listes, qui sont indexées par des entiers, les dictionnaires sont indexés par des clés qui peuvent être de n'importe quel type usuel non modifiable : des chaînes de caractères (`str`), des tuples (`tuple`) ou des types élémentaires (`int`, `float`, `bool`).

### I.2. Création d'un dictionnaire

#### I.2.a) Définition directe

- La création d'un dictionnaire se réalise en suivant la syntaxe :

  `{c1 : v1, ..., cn : vn}`

  où :
  - `c1`, ..., `cn` sont des clés (nécessairement deux-à-deux distinctes),
  - `v1`, ..., `vn` sont les valeurs qui leur sont associées.

```
1  nombre_de_roues = {'voiture': 4, 'vélo': 2, 'tricycle': 3}
```

  Ici, la valeur associé à la clé `'vélo'` est l'entier 2. Comme précisé précédemment, les clés, comme les valeurs, peuvent être de différents types.

**Commentaire**

À la différence d'une liste, un dictionnaire n'est pas une collection *ordonnée*.

  Un autre exemple de dictionnaire qui associe à chaque chiffre sa valeur numérique :

```
1  valeur = { 'zéro': 0, 'un': 1, 'deux': 2, 'trois': 3, 'quatre': 4, 'cinq': 5}
```

- `{ }` crée un dictionnaire vide.

**Exercice 1**
Construire un dictionnaire `jours_par_mois` qui associe à chaque mois d'une année non bissectile, son nombre de jours. On représentera les mois par des chaînes de caractères.

#### I.2.b) Utilisation de la fonction `dict`

La fonction `dict` permet également de créer des dictionnaires.

```
dico_en_fr = dict(yes = 'oui', no = 'non', why = 'pourquoi')
```

#### I.2.c) À partir de tuples à 2 éléments

On peut également créer un dictionnaire à l'aide d'une liste de tuples à 2 éléments.

```
liste_en_nb = [('one', 1), ('two', 2), ('three', 3)]
dico_en_nb = dict(liste_en_nb)
```

#### I.2.d) Définition par compréhension

Comme pour les listes, on peut également définir un dictionnaire par compréhension.

```
suite_carres = {x: x**2 for x in range(5)}
```

#### I.2.e) À l'aide d'une structure itérative

Enfin, un dictionnaire peut également être construit à l'aide d'une structure itérative.

```
chiffres = ['zéro', 'un', 'deux', 'trois', 'quatre', 'cinq']
valeurs = {}
for i in range(6) :
    c = chiffres[i]
    valeurs[c] = i
```

#### I.2.f) En effectuant une copie

Comme pour les listes, on peut effectuer une copie d'un dictionnaire avec la méthode `.copie`.

```
suite2 = suite_carres.copie()
```

### I.3. Opérations sur les dictionnaires

#### I.3.a) Ajout de clés et de valeurs

L'instruction `d[clé] = valeur` peut avoir deux effets :

× si la clé `clé` n'est pas présente dans le dictionnaire `d`, alors cette instruction ajoute l'association `clé: valeur` dans le dictionnaire `d`.

```
dico_en_fr['because'] = 'seulement'
```

× si la clé `clé` est présente dans le dictionnaire `d`, alors cette instruction modifie la valeur associée à la clé `clé` dans le dictionnaire `d`.

```
dico_en_fr['because'] = 'parce que'
```

### I.3.b) Supprimer des entrées

- Comme pour les listes, c'est la commande `del` qui permet de supprimer des entrées d'un dictionnaire.

```
In [1] :   del dico_en_fr['no']
In [2] :   print(dico_en_fr)
Out [2] :  {'yes': 'oui', 'why': 'pourquoi', 'because': 'parce que'}
```

- La méthode `.clear` permet de vider complètement un dictionnaire.

```
In [3] :   suite_carres.clear()
In [4] :   print(suite_carres)
Out [4] :  {}
```

### I.3.c) Tester l'appartenance à un dictionnaire

L'instruction `in` permet de tester l'appartenance d'une clé à un dictionnaire.

```
In [5] :   print('yes' in dico_en_fr)
Out [5] :  True
```

L'instruction `in` `NE` permet `PAS` de tester l'appartenance d'une valeur à un dictionnaire.

```
In [6] :   print('oui' in dico_en_fr)
Out [6] :  False
```

### I.3.d) Consultation d'un dictionnaire

Comme un dictionnaire est une structure séquentielle, on retrouve les commandes spécifiques à ce type de données (décrites dans le TP2).

- La fonction `len` permet de calculer la longueur d'un dictionnaire.

```
In [7] :   len(dico_en_fr)
Out [7] :  3
```

- On accède à la valeur associé à une clé avec la syntaxe `[clé]`.

```
In [8] :   dico_en_fr['why']
Out [8] :  'pourquoi'
```

- La méthode `.keys` donne la liste de toutes les clés du dictionnaire.

```
In [9] :   dico_en_fr.keys()
Out [9] :  dict_keys(['yes', 'why', 'because'])
```

- La méthode `.values` donne la liste de toutes les valeurs du dictionnaire.

```
In [10] :  dico_en_fr.values()
Out [10] : dict_keys(['oui', 'pourquoi', 'parce que'])
```

- La méthode `.items` donne la liste de tous les couples `(clé, valeur)`.

```
In [11] :  dico_en_fr.items()
Out [11] : dict_items([('yes', 'oui'), ('why', 'pourquoi'), ('because',
           'parce que')])
```

#### I.3.e) Parcours d'un dictionnaire

On peut enfin utiliser un parcours de dictionnaire pour effectuer une énumération.

```
1 for i in dico_en_fr.keys :
2     print(i)
```

# II. Applications

## II.1. Décodage

**Exercice 2**

*1.* Récupérer le fichier `morse.txt`. Vous y trouverez un dictionnaire `dict_morse` contenant l'alphabet Morse (inventé par Samuel Morse pour la télégraphie en 1832).

*2.* Écrire une fonction `trad_to_morse` prenant en argument un message sous forme de chaîne de caractère `message` (ne contenant que des majuscules sans accent et des espaces), et renvoyant une chaîne de caractère codant `message` en morse.
*On rappelle que pour séparer les mots de* `message`*, on peut utiliser la méthode* `.split`*. Par défaut, le séparateur est un espace. Par exemple :*

```
In [12] : message = 'HELLO WORLD'
In [13] : print(message.split())
Out[13] : ['HELLO', 'WORLD']
In [14] : print(message.split('L'))
Out[14] : ['HE', '', 'O WOR', 'D']
```

Dans le message codé, chaque lettre doit être séparée d'une autre par un espace ' ' et chaque mot d'un autre par trois espaces ' '. On obtient par exemple :

```
In [15] : trad_to_morse('HELLO WORLD')
Out[15] : '.... . .-.. .-.. ---   .-- --- .-. .-.. -..'
```

*3. a)* Écrire une fonction `inverser_dict` prenant en argument un dictionnaire et renvoyant un dictionnaire dont les clés sont les valeurs du dictionnaire initial, et les valeurs sont les clés du dictionnaire initial.
*Notez que cette fonction n'est utilisable que pour un dictionnaire injectif, c'est-à-dire dans lequel une même valeur n'est jamais associée à deux clés distinctes.*

*b)* Appliquer cette fonction au dictionnaire `dict_morse` pour obtenir un dictionnaire de décodage du morse.

*4. a)* À l'aide du dictionnaire de décodage, écrire une fonction de décodage du morse.

*b)* Appliquer cette fonction à la chaîne `message_mystere`.

## II.2. Compter avec un dictionnaire

### Exercice 3

***1.*** Compléter la fonction suivante qui crée un dictionnaire comptant le nombre d'apparitions de chaque lettre dans un mot.

```
def occurences_lettre(mot) :
    dico = {}
    for lettre in mot :
        if lettre in dico :
            dico[lettre] = ...
        else :
            dico[lettre] = ...
    return(dico)
```

***2.*** La tester sur le mot `'abracadabra'`.

### Exercice 4

***1.*** Récupérer le fichier `LaFontaine.txt`.
*On rappelle que pour accéder à un fichier* `.txt` *en* ***Python*** *sans avoir à en copier le contenu dans l'éditeur, on utilise la fonction* `open`*, en prenant garde à indiquer le chemin d'accès au fichier. En fin d'utilisation, on le ferme grâce à la méthode* `.close`.
*Un fichier peut être ouvert en lecture (*`open` *avec l'option* `'r'`*) ou en écriture (*`open` *avec l'option* `'w'`*). Cette option permet également de créer un fichier. Il existe d'autres options comme* `'a'` *(append) pour l'ajout à la fin du fichier.*

× Écriture dans un fichier

```
montxt = open('chemin\fichier.txt', 'w')
montxt.write('texte...')
montxt.close()
```

× Lecture d'un fichier

```
montxt = open('chemin\fichier.txt', 'r')
montxt.read() # renvoie une chaîne de caractères
montxt.close()
```

***2.*** Écrire une fonction `occurrences(a: str, s: str)` comptant le nombre d'occurrences d'un caractère `a` dans une chaîne `s`.

***3.*** ***a)*** Ouvrir le fichier `LaFontaine.txt` à l'aide de **Python** et stocker son contenu dans une chaîne de caractères.

***b)*** Convertir toutes ses majuscules en minuscules à l'aide de la méthode `.lower`.

***4.*** Définir un dictionnaire qui associe à chaque lettre de l'alphabet le nombre de fois où elle apparaît dans ce texte.
*Pour tester si un caractère* `c` *est une lettre, on peut utiliser la méthode* `.isalpha` *qui renvoie* `True` *si* `c` *est une lettre et* `False` *sinon.*

***5.*** ***a)*** Écrire une fonction `pourcentages` qui prend en argument un dictionnaire associant à chaque clé un nombre d'occurences et renvoie un nouveau dictionnaire associant à chaque clé le pourcentage que représente les occurrences de cette clé sur les total des occurrences du dictionnaire.
*On pourra utiliser la fonction* `round(x: float, n: int)` *qui arrondit* `x` *à la* $\texttt{n}^{ème}$ *décimale.*

***b)*** Tester votre fonction sur le dictionnaire précédent.

***6.*** Bonus : Proposer une fonction `occurrences_chaine(chaine: str, message: str)` comptant le nombre d'occurences d'une `chaine` donnée dans la chaîne de caractères `message`.

## II.3. Jointures

À partir de plusieurs dictionnaires liés entre eux, il est possible d'en créer d'autres pour faire apparaître des informations souhaitées. On appelle cela « effectuer une jointure »
*N.B. : c'est une opération qui peut aussi être effectuée sur des bases de données. Cela sera étudié en* $2^{ème}$ *année.*

### Exercice 5

1. ***a)*** Récupérer le fichier `countries.csv`.

   ***b)*** L'accès à un fichier `.csv` en **Python** suit à peu près le même principe que l'accès à un fichier `.txt`. En implémentant les lignes suivantes, créer un dictionnaire des capitales.

```
import csv
fichier = open('countries.csv', 'r') # ou open('chemin\countries.csv', 'r')
    # si le fichier n'est pas dans votre répertoire courant
lecture = csv.reader(fichier)
capitale = {ligne[1]: ligne[2] for ligne in lecture}

print(capitale['France'])
```

2. Créer de la même façon, un dictionnaire `continent` associant les continents aux pays et un dictionnaire `monnaie` associant les monnaies aux pays.
3. Afficher la liste des noms des pays de l'Océanie, puis de ceux dont la monnaie est l'euro.
4. Afficher la liste des noms des pays non-européens dont la monnaie est l'euro.
5. Créer un dictionnaire associant à chaque continent la liste des monnaies qui y sont utilisées.

## II.4. Devinettes

### Exercice 6

1. Reprendre le dictionnaire `capitale` des capitales de l'exercice précédent.
2. Écrire un programme dans lequel l'ordinateur choisit aléatoirement un nom de pays et vous fait deviner sa capitale. En cas de succès, il vous répond `'Gagné ¡`, sinon `'Perdu ! La bonne réponse était...'`. On utilisera les commandes suivantes :

```
import random as rd
rd.choice(list(capitale.keys()))
```

3. Modifier le programme précédent pour qu'il compte 1 point par succès et affiche votre score final au bout de cinq parties.
4. Modifier le programme précédent pour qu'il vous autorise trois essais à chaque partie.