

# TP13 Dictionnaires

<b>TD n°13 : Structures de données - Les Dictionnaires</b>	<b>Thème 1 : Structures de données</b>
	<b>COURS et EXERCICES</b>



## I. Introduction : nécessité d'un dictionnaire

Prenons l'exemple d'un répertoire téléphonique. Nous pouvons le mémoriser simplement comme un tableau (ou liste) de tableaux [nom, numéro]

```
liste_tel = [["Paul", '0650523454'],
              ["Emile", '0684515345'],
              ["Victor", '0651355186'],
              ["Rose", '0611245678'],
              ["Hélène", '0774845432']]
```

Si nous voulons appeler *Rose*, nous avons deux possibilités avec un tel tableau : \* soit il faut savoir que les informations la concernant sont dans le quatrième élément de la liste (ce qui ne semble pas très pratique et réaliste)

```
print(liste_tel[3][1]) # il faut savoir que l'index de R
```

- soit nous cherchons dans le tableau en partant du premier élément de la liste jusqu'à ce que nous trouvions *Rose* (ce qui revient à feuilleter son répertoire) : cela nécessite d'utiliser une boucle pour parcourir le tableau.

```
for element in liste_tel:
    if element[0] == 'Rose':
        print(element[1])
```

Vous conviendrez que **ce n'est pas pratique** pour accéder à son numéro de téléphone.

De même, la modification ou l'ajout d'un information nécessiterait de devoir feuilleter tout le répertoire. Il semblerait plus pratique d'associer un nom à un numéro, autrement dit d'associer à une **information à une clé**.

C'est ce que les dictionnaires permettent !

## II. Les dictionnaires en Python

Un dictionnaire, de **type dict** en Python, est un ensemble **non ordonné** de paires (clé, valeur) avec un accès très rapide à la valeur à partir de la clé.

C'est un type de conteneur comme les list et les tuple mais ce n'est pas une séquence. Au sens où les valeurs des tableaux ne sont pas indexés par des entiers.

Dans un dictionnaire, **chaque élément est accessible par une clé** qui n'est plus forcément un nombre : une chaîne de caractère, un nombre, ou autre chose, peut être une clé.

Imaginons que je fasse l'inventaire de mon dressing :

habits	quantité
pantalons	3
pulls	4
tee-shirts	8

### ✍ Exemple fondateur n°1 :

- La création du dictionnaire représentant mon dressing se fera par :

```
>>> dressing = {"pantalons":3, "pulls":4, "tee-shirts":8}
```

- L'accès à une valeur se fera par :

```
>>> dressing["pulls"]
4
```

- On dit que "pulls" est la clé et que 4 est la valeur associée à la clé.

- Un dictionnaire est un ensemble clés / valeurs.

```
dressing = {"pantalons":3, "pulls":4, "tee-shirts":8}
dressing["pulls"]
```

4

## 2. Définitions et propriétés d'un dictionnaire

### 2.1 Définitions



#### Définition :

Un dictionnaire est une donnée composite qui \*\*n'est pas ordonnée\*\* (à la différence des listes !) Il fonctionne par un système de `clé:valeur`. Les clés, comme les valeurs, peuvent être de types différents.  
Un dictionnaire est délimité par des accolades.

\_Rappel :\_ - crochets `[ ]` -> listes - parenthèses `()` -> tuples - \*\*accolades `{ }` -> dictionnaires\*\*

### 2.2 Méthodes `.keys()` et `.values()`

#### Exemples fondateurs n°2 :

- Pour lister les clés d'un dictionnaire :

```
>>> dressing.keys()
dict_keys(['pantalons', 'pulls', 'tee-shirts'])
```

- Pour lister les valeurs d'un dictionnaire :

```
>>> dressing.values()  
dict_values([3, 4, 8])
```

```
dressing.keys()
```

```
dict_keys(['pantalons', 'pulls', 'tee-shirts'])
```

```
dressing.values()
```

```
dict_values([3, 4, 8])
```

## 2.3 Parcours d'un dictionnaire :

Il est possible de parcourir un dictionnaire de trois manières :

- parcourir l'ensemble des **clés** avec la méthode `keys()` ;
- parcourir l'ensemble des **valeurs** avec la méthode `values()` ;
- parcourir l'ensemble des **paires clés-valeurs** avec la méthode `items()`.

On peut itérer sur un dictionnaire grâce à l'une de ces méthodes.

### Exemple fondateur n°3 : par les clés (idem tableau)

```
>>> for habit in dressing:  
    print(dressing[habit])  
3  
4  
8
```

Par cette méthode, on obtient les valeurs associées aux clés.

```
for habit in dressing:  
    print(dressing[habit])
```

```
3  
4  
8
```

✍ Exemple fondateur n°4 : par les clés avec la méthode **keys()**.

```
>>> for habit in dressing.keys():  
        print(dressing[habit])  
3  
4  
8
```

Par cette méthodes, on extrait les clés.

```
for habit in dressing.keys():  
    print(habit)
```

```
pantalons  
pulls  
tee-shirts
```

✍ Exemple fondateur n°5 : par les valeurs avec la méthode **values()**.

```
>>> for habit in dressing.values():  
        print(dressing[habit])  
3  
4  
8
```

Par cette méthodes, on extrait les valeurs associées aux clés.

```
for valeur in dressing.values():
    print(valeur)
```

```
3
4
8
```

 **Exemple fondateur n°6 : par les clés et les valeurs avec la méthode items().**

```
>>> for cle,valeur in dressing.items():
            print(cle,valeur)
pantalons 3
pulls 4
tee-shirts 8
```

Par cette méthodes, on extrait les valeurs associées aux clés.

```
for cle,valeur in dressing.items():
    print(cle,valeur)
```

```
pantalons 3
pulls 4
tee-shirts 8
```

## 2.4 Crédit d'un dictionnaire vide

 **Exemple fondateur n°7 :**

Deux méthodes existent pour créer un dictionnaire : `dict()` et  
`{}`

```
>>> mondico = dict()
>>> mondico
{}
>>> mondico['john'] = 12
>>> mondico
{'john': 12}
```

```
>>> contacts = {}
>>> contacts['bob'] = '06 12 17 21 32'
```

## 2.5 Ajout / Modification d'un élément dans un dictionnaire

### Exemple fondateur n°8 :

Pas besoin d'une méthode `append()`, il suffit de rajouter une paire `clé : valeur`

```
>>> dressing["chaussettes"] = 12
```

On peut aussi modifier un dictionnaire existant.

```
dressing["chaussettes"] = 11
```

```
dressing["chaussettes"] = 12
dressing
```

```
{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8, 'chaussett
```

```
dressing["chaussettes"] = 11
dressing
```

```
{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8, 'chaussett
```

## 2.6 Suppression d'une valeur

### Exemple fondateur n°9 :

On utilise l'instruction `del` (déjà rencontrée pour les listes)

```
del dressing["chaussettes"]
```

### Exercice :

Reprendons notre dictionnaire `dressing` :

```
dressing = {"pantalons":3, "pulls":4, "tee-shirts":8}
```

Créer une fonction `achat(vetement, quantite)` qui augmente de `quantite` le nombre de vêtement (pantalon, pull ou tee-shirt) de mon `dressing`.

### Remarque :

Petit problème si on essaie d'acheter un vêtement pour la 1ère fois

```
>>> achat("chemises", 2)
-----
```

KeyError	Traceback
	<pre>&lt;ipython-input-28-fd9d1ac5f62d&gt; in &lt;module&gt; ----&gt; 1 achat("chemises", 2)  &lt;ipython-input-27-feb173444189&gt; in achat(habit)       1 def achat(vetement, quantite): ----&gt; 2     dressing[vetement] = dressing[vetement]  KeyError: 'chemises'</pre>

Nous allons résoudre ce problème grâce à :

## 2.7 Test d'appartenance à un dictionnaire

### **Exemple fondateur n°7 :**

Le mot `in` permet de tester l'appartenance d'une clé à un dictionnaire. Un booléen est renvoyé.

```
>>> "cravates" in dressing
False
```

```
>>> "cravates" in dressing.keys()
False
```

```
>>> 3 in dressing.values()
True
```

### **Exercice :**

Améliorer la fonction `achat(vetement, quantite)` en y incluant un test pour prendre en compte les nouveaux habits.

## II.1. Création d'un dictionnaire

Plusieurs méthodes permettent de créer soit un dictionnaire vide, soit de le noter en extension, soit par compréhension.

```
d1 = {}      # Création d'un dictionnaire vide
d2 = dict() # Création d'un dictionnaire vide (autre mét
d3 = {'poires': 5, 'bananes': 7, 'abricots' : 12} # créa
d4 = {k: k**2 for k in range(1, 10)} # création d'un dic

print(d4)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9:
```

Il est même possible de **créer un dictionnaire à partir d'une liste de couples.**

```
liste = [('cle1','valeur1'),('cle2','valeur2')]
d5 = dict(liste)

liste_tel = [["Paul", 5234], ["Emile", 5345], ["Victor",
d6 = dict(liste_tel)

print("d5 =>", d5)
print("d6 =>", d6)
```

---

A vous 1

Créez un dictionnaire appelé `notes` qui contient les paires (matières, moyenne) de vos trois spécialités. Affichez ensuite ce dictionnaire.

```
# à vous de jouer !
```

## II.3. Taille d'un dictionnaire

La fonction `len` renvoie la taille d'un dictionnaire.

```
len(dressing)
```

```
3
```

### A vous 2

On reprend le dictionnaire `notes` de A vous 1.

```
notes={'NSI':18, 'Maths':15, 'PC':14}
```

1. Affichez la moyenne de NSI.

```
notes={'NSI':18, 'Maths':15, 'PC':14}
```

1. Modifiez votre moyenne de NSI qui a gagné 2 points. Affichez le dictionnaire.

1. Ajoutez la matière `Anglais` avec sa moyenne. Affichez le dictionnaire.

1. Affichez la taille du dictionnaire.

1. Supprimez une des trois spécialités et affichez le dictionnaire.

### A vous 3

On considère le dictionnaire `dressing`.

1. Affichez tous les vêtements du dressing .

1. Ecrivez un programme permettant d'obtenir l'affichage suivant.

Le `dressing` comporte :  
3 pantalons  
4 pulls  
8 tee-shirts

## IV. Les dictionnaires : EXERCICES

### Exercice 1 :

On considère le dictionnaire suivant qui contient différents fruits ainsi que leurs quantités.

```
fruits = {"pommes": 8, "melons": 3, "poires": 6}
```

1. Quelle instruction permet d'accéder au nombre de melons ?

1. On a acheté 16 clémentines et utilisé 4 pommes pour faire une tarte. Quelles instructions permettent de mettre à jour le dictionnaire ?

### Exercice 2 :

On dispose d'un dictionnaire associant à des noms de commerciaux d'une société le nombre de ventes qu'ils ont réalisées. Par exemple : ventes={"Dupont":14, "Hervy":19, "Geoffroy":15, "Layec":21} 1. Écrivez une fonction qui prend en entrée un tel dictionnaire et renvoie le nombre total de ventes dans la société. 2. Écrivez une fonction qui prend en entrée un tel dictionnaire et renvoie le nom du vendeur ayant réalisé le plus de

ventes. Si plusieurs vendeurs sont ex-aequo sur ce critère, la fonction devra retourner le nom de l'un d'entre eux.

### Exercice 3 :

Voici deux dictionnaires :

```
athletes = {"Mike": (1.75, 68), "John": (1.89, 93), "Kat"
sportifs = {"Mike": {"taille": 1.75, "poids": 68}, "John"
```

1. De quel type sont les clés des deux dictionnaires `athletes` et `sportifs` ? De quels types sont les valeurs de ces deux dictionnaires ?

### Réponse :

1. Quelle instruction permet d'accéder à la taille de Kate dans le dictionnaire `athletes` ?

1. Quelle instruction permet d'accéder à la taille de Kate dans le dictionnaire `sportifs` ?

### Exercice 4 :

Le Scrabble est un jeu de société où l'on doit former des mots avec tirage aléatoire de lettres, chaque lettre valant un certain nombre de points. Le dictionnaire `scrabble` contient cette association entre une lettre et son nombre de points.

```
scrabble = {'A': 1, 'B': 3, 'C': 3, 'D': 2, 'E': 1, 'F':
```

Ecrivez une fonction `points(mot)` qui renvoie le nombre de points au scrabble de `mot`, qui est une chaîne de caractères majuscules.

*Par exemple, le mot "ARBRE" doit rapporter 7 points, le mot "XYLOPHONE" doit rapporter 32 points.*

## Exercice 5 :

On considère la variable `personnages` suivante qui réunit quelques informations sur des personnalités (les âges sont fictifs, vous l'aurez compris).

```
personnages = [{"nom": "Einstein", "prénom": "Albert", "âge": 76}, {"nom": "Hamilton", "prénom": "Margaret", "âge": 55}, {"nom": "Nelson", "prénom": "Ted", "âge": 64}, {"nom": "Curie", "prénom": "Marie", "âge": 48}
```

1. Quel est le type de la variable `personnages` ? Quel est le type des éléments de `personnages` ?

## Réponse :

1. Quelle instruction permet d'accéder au dictionnaire de Ted Nelson ?

1. Quelle instruction permet d'accéder à l'âge de Ted Nelson ?

1. Dans le programme suivant, quel est le type de la variable `p` à chaque tour de boucle ? Quel est le rôle de ce programme ?

```
for p in personnages:  
    if int(p['âge']) <= 40:  
        print(p['nom'], p['prénom'])
```

### Réponse :

1. Proposez un programme qui affiche uniquement les noms et prénoms des femmes du tableau `personnages`.

1. Ecrivez une fonction `age_moyen(personnages)` qui renvoie l'âge moyen des personnalités du tableau `personnages` entré en paramètre. *On doit trouver 40,75 ans.*

### Exercice 6 :

On considère le dictionnaire suivant :

```
res={'nsi' :18, 'maths':17, 'svt':14, 'français':14, 'lv1':8}
```

1. Ajouter la moyenne de 12 en lv2.

1. Calculer la moyenne des notes.

Réaliser un affichage des notes qui ressemble à cela :

```
la moyenne en nsi est 18
la moyenne en maths est 17
etc
la moyenne générale est ...
```

## Exercice 7 :

1. Ecrire une fonction `const_dico(cle,valeur)` qui renvoie le dictionnaire définie par les clés et les valeurs entrées en argument.

```
def const_dico(cle:list,valeur:list):
    pass

    return dico
```

- On donne des listes de certains joueurs de League Of Legend ainsi que leur classement et leur nombre de points :

```
pseudo=['Major Alexander', 'KBM Wiz', 'FNC MagiFelix', 'Av
classement=[(12, 1406), (1, 1613), (4, 1507), (9, 1429), (16, 134
```

Appliquer votre fonction `const_dico(cle,valeur)` sur les joueurs de LOL.

### Exercice 8 :

On donne le dictionnaire suivant :

```
turing={'nom': 'Turing', 'prenom': ('Alan', 'Mathison'), 'nat
```

- Afficher les prénoms de Turing.
- Afficher sa nationalité
- Déterminer l'âge qu'avait Alan Turing à sa mort.

### Exercice 9 :

On considère le dictionnaire suivant qui contient les noms des élèves et leurs âges :

```
dico_eleves={'Pierre': 17, 'Chloé': 18, 'Simon': 16, 'Titouan'
```

Créer à partir du dictionnaire `dico_eleves` un dictionnaire `majeur` qui va contenir les élèves majeurs avec leurs âges et un dictionnaire `mineurs` qui va contenir les personnes mineurs avec leurs âges

## Exercice 10 :

Voici une citation célèbre de Gandhi :

```
La vie est un mystère qu'il faut vivre, et non un
problème à résoudre.
```

Créer un dictionnaire qui associe à chaque lettre (clé) son occurrence (valeur)

\* Par exemple la lettre 'a' apparait deux fois.

Par exemple `dico= {'a':2, .....`

## Exercice 11 : Gestion d'une bibliothèque

On considère la base de données suivante :

```
global Livres_BD
Livres_BD ={'Les misérables':{'Auteur':'Victor Hugo', 'St
```

Ecrire un scrcipt permettant d'afficher le stock d'un livre ainsi que son auteur.

Ecrire une fonction `titres_base(LivresBD)` qui renvoie la liste des titres des livres de la base.

Ecrire une fonction `auteurs_base(LivresBD)` qui retourne l'ensemble des noms d'auteurs de cette base dans un tableau.

Ecrire une fonction `auteur(nom)` qui renvoie un dictionnaire qui a pour clé le nom de l'auteur et pour clé les titres des livres de cet auteur.

```
auteur('Victor Hugo')
{'Les misérables':5,'Notre-dame de Paris':4,'Les comtemp
```

Ecrire une fonction `est_present(livre)` qui renvoie True si le livre est présent ou False s'il n'est pas présent.

Ecrire une fonction `ajoute_base(titre,auteur,stock)` qui rajoute à la base LivreBD les données nécessaires pour un nouveau livre après avoir vérifié qu'il ne soit pas présent dans la base.

Ecrire une fonction `ajoute_stock(livre)` qui ajoute 1 au stock du livre concerné et qui affiche la nouvelle base de données.

Ecrire une fonction `enleve_stock(livre)` qui soustrait 1 au stock du livre concerné et qui affiche la nouvelle base de données.

Ecrire une fonction `titres_empruntables(LivresBD)` qui retourne l'ensemble des livres empruntables.

## Exercice 12 : QCM de NSI

Les réponses correctes d'un QCM de NSI sont stockées dans un dictionnaire nommé `reponses_valides`. Les clés sont des chaînes de caractères de la forme "Q1". Les valeurs possibles sont des chaînes de caractères correspondant aux quatre réponses "a","b","c","d".

Exemple : `reponses_valides =`  
`{"Q1": "c", "Q2": "a", "Q3": "d", "Q4": "c", "Q5": "b"}`

Les réponses données par Alice sont stockées dans le dictionnaire `reponses_Alice` dont voici un exemple possible :

```
reponses_Alice = {"Q1": "b", "Q2": "a", "Q3": "d", "Q5": "a"}
```

Lorsqu'Alice n'a pas répondu à une question, il n'y a pas de clef correspondant au nom de l'exercice.

La notation d'un QCM de NSI est la suivante : 3 points par réponse correcte, -1 point par réponse incorrecte et 0 si l'on n'a pas répondu

Compléter la fonction

`correction_QCM_Alice(reponses_Alice,reponses_valides)` qui, à partir des dictionnaires `reponses_Alice` et `reponses_valides` passées en paramètres renvoie le nombre de points obtenus au QCM par Alice.

```
def correction_QCM_Alice(reponses_Alice, reponses_valides
    pass
```

## Exercice 13 : L'application "Contacts" de vos smartphones

L'objectif de cette activité est de programmer deux des fonctionnalités importantes des smartphones actuels :

- Ajouter un contact au répertoire ;
- Rechercher un contact dans le répertoire.

On suppose pour simplifier que le répertoire téléphonique est mémorisé dans le smartphone sous la forme d'un dictionnaire et que chaque élément du dictionnaire est une paire (`prenom`, `numero`) où `prenom` est la clé et `numero` la valeur associée.

### **Etape 1 : Ajouter un contact**

On considère que le répertoire téléphonique est mémorisé dans le dictionnaire `repertoire`. Quelques contacts sont déjà enregistrés dans ce répertoire.

```
repertoire = {'David': 1010, 'Mélanie': 1111, 'Alain': 1}
```

**Question 1 :** Ecrivez une fonction `ajout_contact(repertoire)` qui demande à l'utilisateur de saisir les données (prénom et numéro de téléphone) d'un contact et qui ajoute ce contact à `repertoire`.

**Question 2 :** On veut maintenant créer une fonction `remplissage` qui permet d'ajouter des contacts au répertoire autant de fois que l'on souhaite. Plus précisément, une fois qu'un contact a été saisi on demande à l'utilisateur s'il souhaite ajouter un autre contact.

Complétez la fonction `remplissage` en conséquence. Vous utiliserez la fonction `ajout_contact` écrite à la question précédente.

```
def remplissage(repertoire):
    encore = True
    # à compléter
```

## Etape 2 : Rechercher un contact

On souhaite maintenant écrire une fonction `numero_de(prenom, repertoire)` qui renvoie le numéro de `prenom` si `prenom` est bien dans `repertoire` et qui renvoie un message sinon.

**Question 3 :** Si `prenom` est présent dans `repertoire`, quelle instruction permet d'afficher le numéro associé à `prenom` ?

**Question 4 :** Complétez la fonction `numero_de(prenom, repertoire)` qui renvoie le numéro de téléphone associé dans l'affirmative et un message d'erreur sinon.

```
def numero_de(prenom, repertoire):
    '''prenom est une chaîne de caractères et repertoire
    # à compléter
```

**Exercice 13 :** Quel est le mot de 6 lettres le plus présent dans *Le tour du monde en 80 jours* de Jules Verne ?

Le fichier texte de l'oeuvre de Jules Verne, intitulé `1tdme80j.txt`, a été placé dans le dossier `data` du répertoire de ce notebook. Par souci de simplification, le texte ne contient aucun signe de ponctuation.

*De manière générale, le site du Projet Gutenberg permet de récupérer librement le texte de plusieurs milliers d'oeuvres du domaine public :  
<https://www.gutenberg.org>.*

### **Etape 1 : Lecture du contenu du fichier**

On peut ouvrir et mémoriser dans une variable `texte` le contenu du fichier texte. Pour cela, il suffit d'ouvrir le fichier puis en lire le

contenu sous la forme d'une unique chaîne de caractères avec la méthode `read()`. On ferme ensuite le flux de lecture du fichier.

```
# Ouverture du fichier ('r' pour read = lecture, 'utf-8'
fichier = open("ltdme80j.txt", mode = "r", encoding = "u
# Mémorisation du texte de l'oeuvre dans une chaîne de c
texte = fichier.read()
# Fermeture du flux de lecture
fichier.close()

print(texte)
```

## **Etape 2 : Conversion en un tableau de mots**

On peut ensuite convertir la chaîne `texte` en un tableau contenant les différents mots de l'oeuvre. Pour cela, on peut utiliser la méthode `split()` des chaînes de caractères.

```
tab = texte.split()
print(tab)
```

## **Etape 3 : Compter le nombre d'occurrences de chaque mot**

Un cas d'utilisation typique des dictionnaires consiste à compter les occurrences des éléments d'un tableau.

Considérons par exemple le tableau suivant :

```
[ 'b', 'c', 'e', 'b', 'c', 'j', 'd', 'b', 'j', 'a', 'b']
```

Dans cette liste le caractère 'b' est par exemple répété quatre fois, et le 'j' deux fois, etc. L'objectif est de définir une fonction `occurrences(t)` qui renvoie un dictionnaire avec le nombre

d'occurrences de chaque élément du tableau `t` entrée en paramètre.

Par exemple, la fonction `occurrences` appliquée au tableau précédent

```
occurrences(['b', 'c', 'e', 'b', 'c', 'j', 'd', 'b', 'j']
```

doit renvoyer le dictionnaire :

```
{'b': 4, 'a': 1, 'c': 2, 'e': 1, 'j': 2, 'd': 1}
```

**Question 1 :** Ecrivez la fonction `occurrences(t)` et testez-la sur un tableau de caractères.

**Question 2 :** Appliquez la fonction `occurrences` à ce tableau pour récupérer un dictionnaire `d` du nombre d'occurrences de chaque mot.

#### Etape 4 : Trouver le mot de 6 lettres le plus présent

**Question 3 :** Ecrivez une fonction

`mot_6_lettres_plus_frequent(d)` qui renvoie le mot de 6 lettres les plus fréquent dans l'oeuvre de Jules Verne ainsi que son nombre d'occurrence. (*réponse : 'heures' avec 243 occurrences*)

**Réponse :** cela revient à effectuer une recherche de maximum sur les occurrences des mots de 6 lettres. On parcourt donc toutes les clés du dictionnaire `d` (les clés sont les mots) et parmi les mots de 6 lettres on regarde si son nombre d'occurrence est le nouveau maximum. Dans l'affirmative, ce mot devient le mot le plus fréquent (provisoire) et sa valeur dans le dictionnaire le nombre d'occurrences maximum (provisoire).

**Question BONUS :** Ecrire une fonction `mot_plus_frequent(d, k)` qui renvoie le mot de `k` lettres le plus présent dans le dictionnaire `d`. Affichez ensuite le mot le plus fréquent d'une lettre, de deux lettres, etc.

## Exercice 14 : gestion de commandes

Compléter les fonction pour répondre aux docstring

```
global commandes
commandes={'0': {'numero': 'EMA70495', 'nom': 'Ada Lovel
    '1': {'numero': 'VWD74550', 'nom': 'Dorothy V
    '2': {'numero': 'SWK65993', 'nom': 'Gilles Ka
    '3': {'numero': 'NKR34542', 'nom': 'Ada Lovel
    '4': {'numero': 'GEG58414', 'nom': 'Jacques-L
    '5': {'numero': 'FZA36963', 'nom': 'Al-Khwari
    '6': {'numero': 'QWE58690', 'nom': 'Alonzo Ch
    '7': {'numero': 'NLY90647', 'nom': 'Hypatie d
    '8': {'numero': 'VVL26047', 'nom': 'Alonzo Ch
    '9': {'numero': 'CX007384', 'nom': 'Jacques-L
}
```

```
def afficher_commande_numero(numero_commande):
    """
    affiche la commande correspondant au numero
    : numero : str
    : return : print
    """
```

```
>>>afficher_commande_numero('NLY90647')
commande :NLY90647
Nom :Hypatie d'Alexandrie
Adresse :51 rue Whitfield Diffie
Ville :Bordeaux
Etat :En cours
>>>afficher_commande_numero('NLY90687')
NLY90687: numero commande non enregistré
```

```
def recherche_par_nom(nom):
    """
    recherche les commandes correspondantes au nom
    : nom : str
    : return : un tuple contenant les commandes
    >>> print(recherche_par_nom("Ada Lovelace"))
    ({'numero': 'EMA70495', 'nom': 'Ada Lovelace', 'adre
    {'numero': 'NKR34542', 'nom': 'Ada Lovelace', 'adres
    >>>print(recherche_par_nom("Alan Turing"))
    ()
    """
    global commandes

def ajouter_commande(numero,nom,adresse,ville,etat):
    """
    ajoute une commande
    : numero,nom,adresse,ville,etat : str
    : return : le dict commande modifié
    global commandes """
```

```
>>>commandes={'0': {'numero': 'EMA70495', 'nom': 'Ada Lovelace'}}

>>>ajouter_commande("AZE1029", "Alan Turin", "314 rue d'En
>>>print(commandes)
{'0': {'numero': 'EMA70495', 'nom': 'Ada Lovelace', 'adr
    }
```

```
def supprimer_commande(numero):
    """
        supprime la commande correspondant au n°
    : numero : str
    : return : le dict commandes
    """

    
```

```
>>>supprimer_commande('EMA70495')
>>>print(commandes)
{'1': {'numero': 'VWD74550', 'nom': 'Dorothy Vaughan', 'adr
'2': {'numero': 'SWK65993', 'nom': 'Gilles Kahn', 'adres
'3': {'numero': 'NKR34542', 'nom': 'Ada Lovelace', 'adre
'4': {'numero': 'GEG58414', 'nom': 'Jacques-Louis Lions'
'5': {'numero': 'FZA36963', 'nom': 'Al-Khwarizmi', 'adre
'6': {'numero': 'QWE58690', 'nom': 'Alonzo Church', 'adr
'7': {'numero': 'NLY90647', 'nom': 'Hypatie d'Alexandrie
'8': {'numero': 'VVL26047', 'nom': 'Alonzo Church', 'adr
'9': {'numero': 'CX007384', 'nom': 'Jacques-Louis Lions'
>>> supprimer_commande('EMA70895')
numero de commande non existant
```

## Exercice 14 : reconnaissance de la langue utilisée

Cette étude ne prendra en compte que les lettres de notre alphabet pour simplifier le problème mais peut-être généralisée.

```
extrait1="Si je vous ai raconté ces détails sur l'astéro
```

```
extrait2='If I have told you these details about the ast
```

Ecrire une fonction `analyse_lettre(extrait)` qui renvoie un dictionnaire avec le nombre d'apparition de chacune des lettres dans le texte.

Exemple :

<b>Lettre</b>	<b>Extrait1</b>	<b>Extrait2</b>
A	40	55
B	7	11
C	22	10
D	18	28
e	22	86
...	...	...

Ecrire une fonction `analyse(extrait)` qui renvoie un tableau avec le nombre d'apparitions des lettres (mais unique les nombres)

Exemple :

- Pour l'extrait1 : la fonction doit renvoyer le tableau

```
[40, 7, 22, 18, 105, 8, 8, 1, 44, 8, 0, 49, 19, 56, 40, 11, 8, 34, 76, 33, 3]
```

Pour pouvoir faire l'analyse du texte, on va utiliser une analyse fréquentielle.

De plus on va utiliser une notion de distance, pour cela on va utiliser la librairie maths de python.

```
from math import *
```

On va également utiliser les apparition des divers lettres dans les divers pays étudiés.

```
LANGUES=['FR', 'ANG', 'ALL', 'ESP', 'POR', 'ITA']
#Effectif des lettres dans les langues respectives du ta
EFF=[[763, 90, 326, 366, 1471, 106, 86, 73, 752, 54, 4, 545, 296, 709
      [816, 149, 278, 425, 1270, 222, 201, 609, 696, 15, 77, 402, 240
      [651, 189, 306, 508, 1740, 166, 301, 476, 755, 27, 121, 344, 25
      [1253, 142, 468, 586, 1368, 69, 101, 70, 625, 44, 1, 497, 315, 6
      [1463, 104, 388, 499, 1257, 102, 130, 128, 618, 40, 2, 278, 474
      [1174, 92, 450, 373, 1179, 95, 164, 154, 1128, 0, 0, 651, 251, 6
      ]]
```

On utilisera la fonction suivante :

```
def distance(liste1, liste2):
    s1=sum(liste1)    #somme des éléments de la première li
    s2=sum(liste2)    #somme des éléments de la seconde lis
    dist=0
    #on va calculer la somme des carré des différences des
    for i in range(26):
```

```

    dist+=(liste1[i]/s1-liste2[i]/s2)**2    #liste1[i]/s1
    return sqrt(dist)      #pour avoir la distance, il faut

```

Un exemple : comparaison de deux tableaux

```

Liste1=[40, 7, 22, 18, 105, 8, 8, 1, 44, 8, 0, 49, 19, 56, 40, 11, 8, 34,
Liste2=[763, 90, 326, 366, 1471, 106, 86, 73, 752, 54, 4, 545, 296, 7

distance(Liste1,Liste2)

```

0.06197049246150164

Maintenant calcuer la distance entre le tableau extrait d'un texte (ici `extrait1` et `extrait2`) avec le tableau correspondant à chacune des langues présentes dans le tableau LANGUES.

Pour cela créer une fonction `langue(extrait)` qui calcule la distance entre chaque langue et l'extrait testé.

```

def langue(extrait):
    e=analyse(extrait)
    numero_langue=0  #au initialise à 0 ce qui correspond
    for tab in EFF:  # cette méthode permet de parcourir l
        pass

```

Compléter la fonction afin d'obtenir le résultat suivant avec l'extrait 1

```

print(langue(extrait1)
FR 0.061940492..... #distance la plus faible donc la lan
ANG 0.119750.....
ALL 0.10130.....
ESP 0.0999...
POR 0.13131.....
ITA 0.12394...

```

## Faire de même avec l'extrait 2

```
#Test avec de l'espagnol  
extrait3='Si les he contado estos detalles sobre el aste
```

```
#Texte en italien  
extrait4='''Be , loro alzeranno le spalle, e vi tratterra  
Ma certo, noi che comprendiamo la vita, noi ce ne infisc  
"C'era una volta un piccolo principe che viveva su di un
```

```
#Test avec un texte en allemand  
extrait5="Wenn ich euch dieses nebensächliche Drum und D
```