Dictionnaires - TP (énoncé)

Objectif ✔ : connaître la structure de données "dictionnaire".
Savoir faire : savoir créer et manipuler les dictionnaires.

Signification des icônes :
- savoir ;
- exercices ;
- code à exécuter ;
- notion à comprendre ;
- exécution à observer.

Objectif concours

⚠ Respecter impérativement les notations de l'énoncé.
Une lecture attentive de l'énoncé est indispensable (grandeurs d'entrée et grandeurs de sortie pour une fonction en particulier).

⚠ Notebook séquencé

Les cellules de ce notebook s'affichent au fur et à mesure de la progression à condition de valider certaines cellules.
Pour que cette fonctionnalité soit opérante il faut charger la bibliothèque suivante.

Valider (ctrl + entrée) impérativement la cellule ci-dessous pour poursuivre.
Pour débloquer la suite du notebook, valider :
- la **dernière cellule** du notebook lorsqu'elle est atteinte ;
- la ligne contenant une instruction **test_xxx()** ;
- la ligne contenant le **symbole** .

```
# Valider cette cellule
from _validation import *
```

Python - Aide syntaxe (shell, notebook)

Pour obtenir de l'aide sur la syntaxe, utiliser la commande ***help()***.
Par exemple, pour connaître la syntaxe de la fonction round() (pour arrondir un nombre), entrer : `help(round)`

```
help(round)
```

```
# Valider systématiquement toutes les cellules contenant ce symbole
```

Dictionnaires

Rappels et compléments

Structure de dictionnaire

Une **liste** L, un **tuple** t, un **tableau numpy** a (array) sont des structures de données dont ***les éléments sont indexés par des entiers*** : on accède à l'élément d'index i par `L[i]`, `t[i]`, `a[i]`.

Un dictionnaire (ou encore table d'association) est une structure de données associant un ensemble de clefs (keys) à un ensemble de valeurs (values).
Dans la suite, C désigne l'ensemble des clefs et V l'ensemble des valeurs.
Les clés peuvent être de type str, int, float, tuple, bool
Etant donnés un ensemble C de clés possibles et un ensemble V de valeurs possibles, un dictionnaire représente un ensemble fini d d'associations (clé, valeur) (d est un sous-ensemble de C x V) tel qu'à chaque clé corresponde au plus une valeur : si (k1, v1) et (k2, v2) sont deux associations du dictionnaire d et k1 = k2, alors v1 = v2.
Si k est une clé du dictionnaire d, la seule valeur qui lui est associée sera notée `d[k]`.
On note $\{k_0 : v_0,\ k_1 : v_1, ..., k_{n-1} : v_{n-1}\}$ le dictionnaire de n objets qui associe la valeur v_i à la clé k_i avec $0 \leq i \leq n-1$.
***Un dictionnaire d est donc une structure de données dont les éléments sont indexés par des clés : on accède à la valeur v associée à la clé c par* d[c]** (i.e. d[c] renvoie la valeur v).
Un dictionnaire est une structure de données **mutable** (modifiable) et **non ordonnée** (on ne pas trier un dictionnaire).

Exemple : tableau des coefficients à CCINP-physique
`ccinp_physique = {'Maths':14, 'phys':15, 'chimie':8, 'philo':9, 'mod':8, 'lva':4, 'lvb':2}`

Exemple d'application réelle

La gestion des **noms de domaines** (DNS, Domain Name System) utilise un dictionnaire : à chaque adresse IP (Internet Protocol) de la forme "https://fr.wikipedia.org/wiki/Adresse_IP" (chaîne de caractères = clé) correspond une adresse de la forme 135.160.254.100 (valeur).

📚 Opérations sur les dictionnaires

- Création d'un dictionnaire
- Existence d'une clé c dans le dictionnaire
- Lecture de la valeur v associée à une clé existante c
- Ajout d'une nouvelle association clé/valeur
- Suppression d'une association clé/valeur

Ces opérations sont détaillées ci-dessous ainsi que les techniques de parcours des dictionnaires.

1 Ensemble vs dictionnaire

💡 Ne pas confondre ensemble et dictionnaire (accolades dans les deux cas).
💻 Exécuter les cellules suivantes :

```
# Ensemble
e = {'ab', 1, -2.36, True}
print(e)
type(e)

# Dictionnaire
d = {'a1':4, 'azerty':'fr', 'No':False}
print(d)
type(d)
```

I - Manipulation des dictionnaires

Conseil : faire un **résumé** sur papier des instructions suivantes.
⚠ Il est impératif de respecter les consignes sous peine de bloquer les tests.
Si vous désirez tester vos propres idées, prenez soin de rétablir les modifications attendues en validant à nouveau les cellules, dans l'ordre, depuis la question 1-a.

I-1. Création d'un dictionnaire

📚 Méthode 1 par extension

nom_variable = { *'clé1'*: *valeur1*, *'clé2'*: *valeur2*,... }
nom_variable = {} crée un dictionnaire vide.

✏ Exercice n°1-a) Créer le dictionnaire *ccinp_physique*
Coefficients à CCINP-physique :
'Maths':14, 'phys':15, 'chimie':8, 'philo':9, 'mod':8, 'lva':4, 'lvb':2

```
# Création du dictionnaire ccinp_physique
ccinp_physique =

# Valider systématiquement toutes les cellules contenant un tel test
test_1a()
```

✏ Exercice n°1-b) Afficher le dictionnaire ccinp_physique

```
# Affichage du dictionnaire
ccinp_physique
```

📚 Méthode 2 conversion (dict)

nom_variable = dict(iterable)
où *iterable* est une liste, un tuple,... de couples de la forme (clé, valeur).

✏ Exercice n°2 Créer le dictionnaire *elements* donnant le numéro atomique des 6 premiers éléments du tableau périodique à partir de la liste :
elt = [('H', 1), ('He', 2), ('Li', 3), ('Be', 4), ('B', 5), ('C', 6)] et afficher le dictionnaire.

```
# Création : utilisation de dict()


test_2()

# Affichage du dictionnaire
```

📚 Méthode 3 par compréhension

nom_variable = {formule for variable in range(a,b,c)}
Ou bien création d'un dictionnaire vide puis remplissage à l'aide d'une boucle :

```
d = {}
for ... in range(...):
    d[...] = ...
```

✏ Exercice n°3 Créer un dictionnaire nommé *da* de la forme 1 : 'a', 2 : 'aa', 3 : 'aaa' comportant 9 clés grâce à une formule ou une boucle.

```
# Dictionnaire de la forme {1: 'a', 2: 'aa', 3: 'aaa'} à 9 clés.
```

```
test_3()
```

I-2. Accès à un élément

📚 *nom_du_dictionnaire*`[cle]`

✏ Exercice n°4 Quel est le coefficient de l'épreuve de modélisation à CCINP-Physique (écrire l'instruction donnant la réponse ?

```
# Quel est le coefficient de l'épreuve de modélisation à CCINP-Physique ?
```

✏ Exercice n°5-a) Quel est le numéro atomique de l'oxygène ?

```
# Quel est le numéro atomique de l'oxygène ?
```

Le message d'erreur est explicite...

📚 Afin d'éviter l'interruption "KeyError", on peut **tester la présence d'une clé dans le dictionnaire** :
cle `in` *nom_du_dictionnaire*
La complexité temporelle de ce test d'appartenance est O(1).

✏ Exercice n°5-b) Tester la présence de la clé 'O' dans *elements* (utiliser le mot clé `in`)

```
# Tester la présence de la clé 'O' dans elements
```

I-3. Parcours d'un dictionnaire

📚 Méthodes associées aux dictionnaires

Un dictionnaire est un objet itérable (comme les ranges, les listes, les tuples, les chaines de caractères).
Si d est un dictionnaire, une boucle for et les méthodes d.keys(), d.values() et d.items() permettent de parcourir le dictionnaire d.
`for c in` *nom_du_dictionnaire*`.keys(): # Parcours par clé` ou encore `for c in` *nom_du_dictionnaire*`:`
`for v in` *nom_du_dictionnaire*`.values(): # Parcours par valeur`
`for c, v in` *nom_du_dictionnaire*`.items(): # Parcours par couple(clé, valeur)`
Les noms de variable (c, v) sont arbitraires.

✏ Exercice n°6 Afficher les clés puis les valeurs puis les associations (clé, valeur) pour l'un des dictionnaires précédents.

```
# Méthode keys()
```

```
# Méthode values()
```

```
# Méthode items()
```

Ces trois méthodes permettent de parcourir les dictionnaires (cf.ci-dessous).

💡 **Noter que les résultats de ces commandes ne sont pas des listes** (utiliser la fonction type pour s'en convaincre).
Si nécessaire, on peut convertir ces structures en listes en utilisant la fonction **list()**.

I-3a. Parcours par clé

✏ Exercice n°7 Afficher (print()) les épreuves de CCINP physique à l'écrit (utiliser une boucle for).

```
# Parcours par clé
```

```
# Remarque - Autre syntaxe (plus courte) :
```

I-3b. Parcours par valeurs

✏ Exercice n°8 Calculer la somme, nommée *s*, des coefficients de CCINP à l'écrit en utilisant une boucle for.

```
# Parcours par valeur
```

```
test_8()
```

I-3c. Parcours par clé et par valeurs

La syntaxe du parcours par clé/valeur nécessite deux index au lieu d'un :

```
for k,v in nom_du_dictionnaire.items():
    faire quelque chose avec k et v
```

✏ Exercice n°9 Afficher (print()) les couples épreuve/coefficient de CCINP.

```
# Parcours par clé/valeur
```

I-4. Ajout, suppression dans un dictionnaire

Ajout d'un seul élément - Méthode 1

nom_du_dictionnaire`[cle] = valeur`
⚠ Remplace (écrase) la valeur si la clé est déjà existante.

✏ Exercice n°10 Ajouter l'élement azote au dictionnaire *elements*. Vérifier.

```
# Ajout d'un élément
```

```
test_10()
```

Ajout de plusieurs éléments - Méthode 2

nom_du_dictionnaire.update({cle1: valeur1, cle2: valeur2,...})

Exercice n°11 Ajouter les éléments oxygène et fluor au dictionnaire *elements*.

```
test_11()
```

Suppression

del *nom_du_dictionnaire*[cle]
Génère une erreur si la clé n'existe pas.

Exercice n°12 Supprimer l'élément Hélium du dictionnaire *elements*.

```
test_12()
```

I-5. Copie d'un dictionnaire

d2 = *d1*.copy() # Même instruction que pour les listes

Exercice n°13-a) Exécuter les instructions suivantes.

```
# Comme pour une liste, les instructions suivantes ne copient pas le dictionnaire :
# un nouveau nom de variable est associé au même dictionnaire.
d1 = {'a': 0, 'b': 1}     # Création de d1
d2 = d1                   # d2 et d1 référencent le même dictionnaire
d1['b'] = -9              # Modification de d1
print(d1)
print(d2)
```

Que peut-on en déduire ?
L'instruction d1 = d2 ne crée donc pas une copie du dictionnaire puisque la modification de d1 a entraîné le même modification pour d2 : d2 est donc un alias (autre nom) de d1, ces deux noms de variable sont associés au même dictionnaire (même fonctionnement avec les listes).
Exercice n°13-b) Modifier le code précédent en utilisant la méthode copy() et vérifier que d2 n'est pas modifié lorsque d1 est modifié.

```
d1 = {'a': 0, 'b': 1}
d2 = d1.copy()            # Création d'une copie du dictionnaire d1
d1['b'] = -9
print(d1)
print(d2)
```

```
test_13()
```

I-6. Valeurs d'un dictionnaire

Tout objet Python (nombre, chaîne, liste, dictionnaire...) peut être la **valeur** associée à une clé.

Exemple : valeur de type dictionnaire

```
# Exemple : clé =chaîne et valeur = dictionnaire
pays = {"France": {"capitale": "Paris", "population": 68014000, "superficie": 672051.0},
        "Italie": {"capitale": "Rome", "population": 60359546, "superficie": 301336.0},
        "Espagne": {"capitale": "Madrid", "population": 46934632, "superficie": 505911.0},
        "Allemagne": {"capitale": "Berlin", "population": 84079811, "superficie": 357588.0}}
```

Exercice n°14 Ecrire le code pour afficher la capitale de la France.

I-7. Clés d'un dictionnaire

Objet mutable (i.e. modifiable) vs objet non nutable

On distingue les objets mutables i.e. **modifiables** (liste, tableau numpy, ensemble, dictionnaire) des objets non mutables (entier, flottant, booléen, chaîne, tuple).

Exemples - Observer le résultat des commandes suivantes et conclure quant au caractère mutable des objets de type int, float, bool, str, tuple, list, dict.

Instruction	Exécution
$3 = 8$	SyntaxError : cannot assign to literal
$1.0 = 3.14$	SyntaxError : cannot assign to literal
True = False	SyntaxError : cannot assign to True
`s = 'abcde' s[2]= 'z'`	TypeError : 'str' object does not support item assignment
`t = (0, 1, 2, 3) t[2]= 18`	TypeError : 'tuple' object does not support item assignment
`L = [0, 1, 2, 3] ; L[2]= 3.14`	[0, 1, 3.14, 3]
`d ={0: 'faux', 1: 'fake'}; d[1]= 'vrai'`	{0 : 'faux', 1 : 'vrai'}

Clés d'un dictionnaire

Les **clés** d'un dictionnaire peuvent être tout objet de type non mutable récursivement (récursivement signifie que si une clé est un tuple par exemple, les éléments du tuple doivent être non mutables).
Une liste, un dictionnaire (objets mutables) ne peuvent pas être des clés d'un dictionnaire.

Exemple Les clés du dictionnaire suivant sont valides.

```
dic = 'a': True, 45:'azerty', True: 3.14, 1.33: 0, (4,-5): 111
```

```
# Exemple de clés
dic = {'a': True, 45:'azerty', True: 3.14, 1.33: 0, (4,-5): 111}

dic[(4,-5)]  # Le tuple (4,-5) est une clé valide
```

Mini projet : statistiques linguistiques

Objectif ✔ : effectuer des statistiques linguistiques dans un texte (nombre d'occurrences de chaque lettre).
Il s'agit ici de construire deux dictionnaires.
Un premier dictionnaire de la forme : `lettre = {'a': 250, 'b': 67, 'c': 28, ...}` où les clés sont les lettres de l'alphabet et les valeurs représentent le nombre d'occurrences de chaque lettre dans un texte.
Et un second dictionnaire de la forme : `frequence = {'a': 0.13, 'b': 0.05, 'c': 0.01, ...}` où les clés sont les lettres de l'alphabet et les valeurs représentent la fréquence d'apparition des lettres en %.

Objectif concours

La manipulation des fichiers et le traitement des chaînes de caractères sont fréquents en concours et en TIPE.

Mini projet - Etape 1

MiP - Etape 1 - Nombre d'occurrences des lettres dans une chaîne.

1. Tester l'instruction s.**split()** après avoir défini la chaîne s = 'abca ba ccc'.

```
# 1 - Test split sans argument
```

💻👁 Tester l'instruction suivante.

```
# Test split avec argument
'abc d, jk , dz'.split(',')
```

MiP - Etape 1 - Nombre d'occurrences des lettres dans une chaîne.

2. Ecrire, **en pseudo-code** (sur feuille annexe) le corps d'une fonction *occurrences(s, lettre)* dont les paramètres sont une chaîne *s* et un dictionnaire (vide ou non) nommé *lettre*.
   La fonction ne renvoye rien car elle *modifie* le dictionnaire.
   Après l'exécution de la fonction, le dictionnaire est mis à jour à partir des lettres présentes dans la chaîne s.

2 - Pseudo-code du corps de la fonction (à compléter sur papier) :
MiP - Etape 1 - Nombre d'occurrences des lettres dans une chaîne.

3. Ecrire la fonction *occurrences* en python.

```
# 3 - Fonction occurrences
```

MiP - Etape 1 - Nombre d'occurrences des lettres dans une chaîne.

4. Tester la fonction (par exemple avec la chaîne s précédente).

```
# 4 - Test fonction occurrences
```

```
test_occurrences()
```

Mini projet - Etape 2

Python - Formatage d'une chaîne

- La méthode `s.split()` (où s est une chaîne) permet de scinder s en plusieurs chaînes en utilisant le caractère vide (blanc) comme critère, le résultat est une liste de chaînes.
  Rq : s.split(sc) découpe s en utilisant la chaîne sc comme critère de découpage.
- La méthode `s.strip()` (où s est une chaîne) permet d'enlever les ***espaces et les sauts de ligne*** (un saut de ligne est codé par
  n) ***en début et en fin de chaîne*** (mais pas à l'intérieur de la chaîne).
  Rq : s.strip('., ;ah !') effectuera le même travail avec tous les caractères présents dans la chaîne passée en argument.
- La méthode `s.lower()` (où s est une chaîne) permet de transformer toutes les majuscules en ***minuscules*** dans la chaîne s.
  Rq : `s.upper()` effectue le travail inverse.

Ces méthodes ne modifient pas la chaîne initiale (il faut sauvegarder la transformation dans une nouvelle chaîne).

MiP - Etape 2 - Nombre d'occurrences des lettres dans un texte (Extrait de Notre Dame de Paris, texte intégral disponible sur https://www.gutenberg.org/).
💻👁 Exemples d'utilisation de la méthode strip()

```
# Exemple de chaîne à épurer (avec sauts de lignes et espaces indésirables au début et en fin de chaîne)
l = "\n   Le Caractère \n Est Un Saut De Ligne     \n      "
l

# Suppression des sauts de lignes et espaces indésirables au début et en fin de chaîne
l = l.strip()  # Ici, on sauvegarde la transformation dans la chaîne initiale
l
```

Exemple d'utilisation de la méthode lower()

```
# Conversion de la totalité de la chaîne en minuscules
l.lower()    # Ici, pas de sauvegarde de la transformation
```

Python - Ouverture d'un fichier en lecture

```
with open(nom_de_fichier) as f:                    # Ouverture du fichier
     while 1:                                       # Boucle "infinie" : 1 est toujours vrai...
         ligne = f.readline()                       # Lecture d'une ligne (type chaîne)
         if ligne == "":                            # Ligne vide : fin du fichier atteinte
             break                                  # Sortie de la boucle while
         else:
             faire quelque chose avec ligne         # Traitement de la ligne
 f.closed                                           # Fermeture (faculatif avec la syntaxe with open)
```

MiP - Etape 2 - Nombre d'occurrences des lettres dans un texte

5. Ecrire un script permettant de déterminer le nombre d'apparition des lettres dans le **fichier nommé 'NdP_utf8.txt'** (associé à ce notebook donc directement disponible sous ce nom).
   Ce script créera un dictionnaire nommé *lettreNdP*, il fera appel à la fonction occurences (les chaînes seront préalablement converties en minuscules et débarrassées des sauts de ligne grâce à strip()).

```
# 5 - Occurrences des caractères dans Notre Dame de Paris : création de lettreNdP
```

```
test_occ_NdP()
```

MiP - Etape 3 - Statistiques

6. On observant le dictionnaire *lettre* ci-dessus, on constate que l'espace ' ' (non significatif) est le plus représenté et que d'autres caractères doivent être supprimés (guillemets, signes de ponctuation, chiffres, parenthèses, crochets, tirets...).
   Supprimer ces caractères du dictionnaire (les regrouper dans une liste et utiliser une boucle).
   Rq : il serait également possible d'utiliser la méthode strip() lors de la lecture du fichier.
   Vérifier le traitement en affichant les clés du dictionnaire.

Compléter le code ci-dessous.

```
# 6 - Suppression des caractères non alphabétiques du dictionnaire lettre
carac_suppr = [str(i) for i in range(10)] + \
[' ', '.', ',', ';', ':', '!', '?', "'", '«', '»', '(', ')', '[', ']', '-', '_']
for c in
    if
        del
lettreNdP.keys()
```

MiP - Etape 3 - Statistiques

7. Ecrire une fonction *total_lettres(dic)* de paramètre un dictionnaire *dic* obtenu par la fonction occurrences et renvoyant le nombre total de lettres présentes dans *dic*.
   Tester la fonction avec le dictionnaire *lettre* (étape 1 question 4).
   Appliquer la fonction au dictionnaire *lettreNdP*.

```
# 7 - Fonction total_lettres(dic)
```

```
test_total_lettres()
```

MiP - Etape 3 - Statistiques

8. Ecrire une fonction *frequence(dic, tot)* de paramètres un dictionnaire *dic* obtenu par la fonction occurrences et *tot* le nombre total de letttres (obtenu par *total_lettres(dic)*) et renvoyant le dictionnaire des fréquences de chaque lettre (en %). On pourra utiliser la fonction round().
   Tester cette fonction comme précédemment.
   L'appliquer au texte de Notre Dame de Paris.

```
# 8 - Fonction frequence(dic, tot)
```

```
test_frequence()
```

MiP - Etape 3 - Statistiques

9. 💻👁 Comparer aux valeurs des fréquences dans https://fr.wikipedia.org/wiki/Fr%%A9quence_d%27apparition_des_lettres"
   Fréquence d'apparition des lettres : pour les 7 premières lettres (cf. ci-dessous), afficher pour chacune la lettre, sa fréquence obtenue dans Wikipedia et sa fréquence dans Notre Dame de Paris (une ligne pour chaque lettre).
   Le caractère '' est une tabulation et permet d'obtenir un affichage en colonnes.

```
# 9. Etape 3 - Comparaison avec les données de Wikipedia
freqWikipedia ={'e':12.1, 'a':7.11, 'i':6.59, 's':6.51, 'n':6.39, 'r':6.07, 't':5.92}
print ('Lettre', '\t', '% Wiki', '\t', '% NdP')
for c in freqWikipedia:
    print (c, '\t', freqWikipedia[c], '\t\t', freq[c])
```

II - Implémentation d'un dictionnaire - Tables de hachage

A suivre en cours...