# TD 3 & TD 4 : Fonctions récursives, liste en compréhension, dictionnaires

---

## Exo1 : Les fonctions récursives

1. En mathématiques, la suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est

la somme des deux termes qui le précèdent : $U\ 0 = 0\ U\ 1 = 1\ U\ n = U\ (n–1) + U\ (n–2)$ pour tout $n \geq 2$

Dans la suite, penser à utiliser des « assert » pour tester les pré-conditions de vos fonctions.

1-1. En utilisant la boucle « for », écrire une fonction affichant le terme d'ordre « n » de la suite de Fibonacci.

In [164…

```
def fibonacci(n):

    if (n == 0):
        return 0
    if (n == 1):
        return 1

    val_fibo = [0] * n
    val_fibo[1] = 1
    for i in range(2,n):
        val_fibo[i] = val_fibo[i-1] + val_fibo[i-2]

    return val_fibo[n-1]
```

1-2. En utilisant la boucle « while », écrire une fonction retournant la liste des termes de la suite de Fibonacci jusqu'au terme d'ordre« n ».

In [164…
```python
def termes_fibonacci(n):

    val_fibo = [0] * n
    if(n>1):
        val_fibo[1] = 1
    i = 2

    while i < n:
        val_fibo[i] = val_fibo[i-1] + val_fibo[i-2]
        i+=1

    return val_fibo

print(termes_fibonacci(13))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

1-3. Écrire une fonction récursive pour le calcul du terme d'ordre « n » de la suite de Fibonacci.

In [164…
```python
def recursive_fibonacci(n):
    if n < 0:
        return "n cannot be negative"
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return recursive_fibonacci(n-1) + recursive_fibonacci(n-2)

print(recursive_fibonacci(3))
```

```
2
```

2. Écrire une fonction récursive pour le calcul de la factorielle d'un nombre, rappel : 4! =

$4x3x2x1 = 24$.

In [164…
```python
def recursive_factorielle(n):
    if n == 1:
        return 1
    else:
        return n * recursive_factorielle(n-1)

print (recursive_factorielle(4))
```

```
24
```

3. Écrire une fonction récursive pour le calcul du plus grand commun diviseur entre deux entiers a

et b.

In [164…
```
def recursive_PGCD(a, b):
    if b==0:
        return a
    else:
        r=a%b
        return recursive_PGCD(b,r)

print(recursive_PGCD(2,10))
```

```
2
```

4. Écrire une méthode récursive pyramide_inversee(n, s) qui écrit sur la première ligne, n fois la

chaîne « s », sur la deuxième, n-1 fois la chaîne « s », et ainsi de suite jusqu'à la dernière ligne, où il y aura 1 fois la chaîne « s ». Ainsi pyramide(3,"tot") donnera: tottottot tottot tot

In [164…
```
def pyramide_inversee(n,s):
    if n < 1:
        return None

    for i in range(n):
        print(s,end="")

    print()

    return pyramide_inversee(n-1,s)

pyramide_inversee(3,"tot")
```

```
tottottot
tottot
tot
```

5. Écrire une méthode récursive qui calcule x^n où x et n sont des entiers :

5-1. En utilisant la propriété x^n = x ∗ x^(n-1)

In [164…
```python
def puissance(x,n):
    if n < 2:
        return x
    return x * puissance(x,n-1)

print(puissance(2,4))
```

16

5-2. En utilisant l'exponentiation rapide qui repose sur les égalités : x^n = x^(n/2) * x^(n/2) si n est pair et x^n = x^(n-1/2) * x^(n-1/2) si n est impair.

In [164…
```python
def puissance_rapide(x,n):
    if n < 2:
        return x

    if n%2 == 0:
        return puissance_rapide(x,n/2) * puissance_rapide(x,n/2)
    else:
        return puissance_rapide(x,(n-1)/2) * puissance_rapide(x,(n-1)/2)

print(puissance_rapide(2,4))
```

16

# Exo2 : Manipulation d'un dictionnaire et liste en compréhension

**Rappel** : Les dictionnaires sont parfois présents dans d'autres langages sous le nom de « mémoires associatives » ou « tableaux associatifs ». À la différence des séquences (listes, tuples), qui sont indexées par des nombres, les dictionnaires sont indexés par des « clés », qui peuvent être de n'importe quel type immuable ; les chaînes de caractères et les nombres peuvent toujours être des clés. Un dictionnaire est constitué d'un ensemble d'item « clé : valeur », la clé doit être unique.

1. Créer un dictionnaire vide. Ajoutez-y les items «'nom' :'toto' » , « 'prénom ' :'titi' », « 'age':20»,

« 'adresse':404 ».

In [164…
```python
dico = dict()

dico["nom"] = "toto"
dico["prénom"] = "titi"
dico["age"] = 20
dico["adresse"] = 404

print(dico)
```

```
{'nom': 'toto', 'prénom': 'titi', 'age': 20, 'adresse': 404}
```

2. Modifier puis supprimer l'âge.

In [165…
```
dico["age"] = 25
print(dico)
dico.pop("age")
print(dico)
```

```
{'nom': 'toto', 'prénom': 'titi', 'age': 25, 'adresse': 404}
{'nom': 'toto', 'prénom': 'titi', 'adresse': 404}
```

3. Écrire une instruction pour récupérer la valeur 'toto' à partir de la clé correspondante.

In [165…
```
val = dico["nom"]
print(val)
```

```
toto
```

4. Écrire une instruction qui vérifie l'existence de la clé 'prénom ' et 'age' dans le dictionnaire.

In [165…
```
print("prénom" in dico)
print("age" in dico)
```

```
True
False
```

5. Écrire une instruction pour récupérer toutes les clés. Puis une deuxième instruction pour

récupérer toutes les valeurs. Puis une troisième pour récupérer tous les items.

In [165…
```
print(dico.keys())
print(dico.values())
print(dico.items())
```

```
dict_keys(['nom', 'prénom', 'adresse'])
dict_values(['toto', 'titi', 404])
dict_items([('nom', 'toto'), ('prénom', 'titi'), ('adresse', 404)])
```

6. En utilisant les listes en compréhension, créez une liste de nombre entiers aléatoire de 0 à 10.

Puis une deuxième liste d'entiers de -10 à -1.

In [165…
```
from random import randint

new_list_rand = [randint(0,10) for x in range(10)]
new_list_negatif = [-10+x for x in range(10)]
print(new_list_rand)
print(new_list_negatif)
```

```
[1, 6, 4, 9, 1, 10, 5, 5, 8, 7]
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
```

7. Créer un nouveau dictionnaire, dans lequel vous associez la clé « positif » à liste des entiers

positifs. Même traitement pour les entiers négatifs.

In [165…
```
dico = {}
dico["positif"] = new_list_rand
dico["negatif"] = new_list_negatif

print(dico)
```

```
{'positif': [1, 6, 4, 9, 1, 10, 5, 5, 8, 7], 'negatif': [-10, -9, -8, -7
, -6, -5, -4, -3, -2, -1]}
```

8. Créer une fonction qui prend en paramètre le dictionnaire de la question 7). La fonction tire un

entier aléatoire entre -5 et 5, puis ajoute la valeur aléatoire dans le dictionnaire selon le signe de celle-ci. Exemple -2 sera ajouté à la liste des entiers négatifs dans le dictionnaire.

In [165…
```
def ajoute_val(dico):

    val = randint(-5,5)
    if val < 0:
        dico["negatif"].append(val)
    else:
        dico["positif"].append(val)

ajoute_val(dico)
ajoute_val(dico)
ajoute_val(dico)

print(dico)
```

```
{'positif': [1, 6, 4, 9, 1, 10, 5, 5, 8, 7, 5], 'negatif': [-10, -9, -8,
-7, -6, -5, -4, -3, -2, -1, -2, -2]}
```

9. Écrire une fonction qui élève au carré les éléments positifs dans le dictionnaire précédant, puis

qui élève au cube les éléments négatifs. Utilisez les listes en compréhension.

In [165…
```
def carre_ou_cube(dico):

    dico["positif"] = [x ** 2 for x in dico["positif"]]
    dico["negatif"] = [x ** 3 for x in dico["negatif"]]

carre_ou_cube(dico)
print(dico)
```

```
{'positif': [1, 36, 16, 81, 1, 100, 25, 25, 64, 49, 25], 'negatif': [-10
00, -729, -512, -343, -216, -125, -64, -27, -8, -1, -8, -8]}
```

# Exo3 : Utilisation d'un dictionnaire

1. Écrire une fonction « occurrence(chaîne) » qui construit un dictionnaire dans lequel les clés sont

les caractères du mot « chaîne » et les valeurs sont les occurrences de chaque caractère dans le « mot ».

In [165…
```
def occurrence(chaine):

    dico = dict()

    for char in chaine:
        dico[char] = chaine.count(char)

    return dico

print(occurrence("Antananarivo"))
```

```
{'A': 1, 'n': 3, 't': 1, 'a': 3, 'r': 1, 'i': 1, 'v': 1, 'o': 1}
```

2-1. Saisir la liste de mots suivante : mots = ['eddard', 'catelyn', 'robb', 'sansa', 'arya', 'brandon', 'rickon', 'theon', 'rorbert', 'cersei', 'tywin', 'jaime', 'tyrion', 'shae', 'bronn', 'lancel', 'joffrey', 'sandor', 'varys', 'renly']

In [165…
```
mots = ['eddard', 'catelyn', 'robb', 'sansa', 'arya', 'brandon', 'rickon'
        'jaime', 'tyrion', 'shae', 'bronn', 'lancel', 'joffrey', 'sandor'
```

2-2. Écrire une fonction « mots_position1(liste, let, pos) » qui prend en paramètre une liste de mots, une lettre et une position. La fonction retourne la liste de mots dans « liste » possédant la lettre « let » à la position « pos ».

In [166…
```
def mots_position1(liste, let, pos):

    new_liste = []

    for mot in liste:
        if mot[pos] == let:
            new_liste.append(mot)

    return new_liste

print(mots_position1(mots, 'r', 0))
```

```
['robb', 'rickon', 'rorbert', 'renly']
```

2-3. Écrire une fonction « mots_position2(liste, let, pos) » qui fait le même traitement que « mot_position1 » mais cette fois-ci en compréhension de liste.

In [166…
```
def mots_position2(liste, let, pos):

    new_liste = [x for x in liste if x[pos] == let]

    return new_liste

print(mots_position2(mots, 'a', 1))
```

```
['catelyn', 'sansa', 'jaime', 'lancel', 'sandor', 'varys']
```

3-1. Écrire une fonction « dictionnaire_new(liste) » qui construit un dictionnaire dont les clés sont de la forme (pos,let) et la valeur est la liste des mots ayant la lettre « let » à la position « pos ».

In [166…
```
def dictionnaire_new(liste):

    dico = dict()
    for mot in liste:
        i = 0
        for char in mot:
            if (i,char) not in dico:
                dico[(i,char)] = []
            dico[(i,char)].append(mot)
            i += 1

    return dico

dico_mots = dictionnaire_new(mots)
print(dico_mots)
```

```
{(0, 'e'): ['eddard'], (1, 'd'): ['eddard'], (2, 'd'): ['eddard'], (3, '
a'): ['eddard', 'arya'], (4, 'r'): ['eddard', 'joffrey'], (5, 'd'): ['ed
dard'], (0, 'c'): ['catelyn', 'cersei'], (1, 'a'): ['catelyn', 'sansa',
'jaime', 'lancel', 'sandor', 'varys'], (2, 't'): ['catelyn'], (3, 'e'):
['catelyn', 'shae'], (4, 'l'): ['catelyn'], (5, 'y'): ['catelyn'], (6, '
n'): ['catelyn', 'brandon'], (0, 'r'): ['robb', 'rickon', 'rorbert', 're
nly'], (1, 'o'): ['robb', 'rorbert', 'joffrey'], (2, 'b'): ['robb'], (3,
'b'): ['robb', 'rorbert'], (0, 's'): ['sansa', 'shae', 'sandor'], (2, 'n
'): ['sansa', 'lancel', 'sandor', 'renly'], (3, 's'): ['sansa', 'cersei'
], (4, 'a'): ['sansa'], (0, 'a'): ['arya'], (1, 'r'): ['arya', 'brandon'
, 'bronn'], (2, 'y'): ['arya'], (0, 'b'): ['brandon', 'bronn'], (2, 'a')
: ['brandon', 'shae'], (3, 'n'): ['brandon', 'bronn'], (4, 'd'): ['brand
on'], (5, 'o'): ['brandon'], (1, 'i'): ['rickon'], (2, 'c'): ['rickon'],
(3, 'k'): ['rickon'], (4, 'o'): ['rickon', 'tyrion', 'sandor'], (5, 'n')
: ['rickon', 'tyrion'], (0, 't'): ['theon', 'tywin', 'tyrion'], (1, 'h')
: ['theon', 'shae'], (2, 'e'): ['theon'], (3, 'o'): ['theon'], (4, 'n'):
['theon', 'tywin', 'bronn'], (2, 'r'): ['rorbert', 'cersei', 'tyrion', '
varys'], (4, 'e'): ['rorbert', 'cersei', 'jaime', 'lancel'], (5, 'r'): [
'rorbert', 'sandor'], (6, 't'): ['rorbert'], (1, 'e'): ['cersei', 'renly
'], (5, 'i'): ['cersei'], (1, 'y'): ['tywin', 'tyrion'], (2, 'w'): ['tyw
in'], (3, 'i'): ['tywin', 'tyrion'], (0, 'j'): ['jaime', 'joffrey'], (2,
'i'): ['jaime'], (3, 'm'): ['jaime'], (2, 'o'): ['bronn'], (0, 'l'): ['l
ancel'], (3, 'c'): ['lancel'], (5, 'l'): ['lancel'], (2, 'f'): ['joffrey
'], (3, 'f'): ['joffrey'], (5, 'e'): ['joffrey'], (6, 'y'): ['joffrey'],
(3, 'd'): ['sandor'], (0, 'v'): ['varys'], (3, 'y'): ['varys'], (4, 's')
: ['varys'], (3, 'l'): ['renly'], (4, 'y'): ['renly']}
```

3-2. Écrire une fonction « mots_position3(dico, let, pos) » qui prend en paramètre le dictionnaire de la question 3-1), une lettre « let » et une position « pos ». La fonction retourne la liste des mots dans « dico» possédant la lettre « let » à la position « pos ».

In [166…

```
def mots_position3(dico, let, pos):

    return dico[(pos,let)]

print(mots_position3(dico_mots,'a',1))
```

```
['catelyn', 'sansa', 'jaime', 'lancel', 'sandor', 'varys']
```

# Exo4 : Slicing, listes et dictionnaire en compréhension

1. L'administration cherche à lister un groupe de 15 élèves dans un TD, quel est la meilleure

structure de donnée pour les regrouper ?

La meilleur structure de donnée pour les regrouper est une liste.

> Générer une liste de 15 élèves ayant chacun une lettre distincte.

In [166…
```
noms = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

liste_eleves = [char for char in noms if noms.index(char) < 15]

print(liste_eleves)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', '
O']
```

2. Importer la librairie « string » et utiliser la variable ascii_lowercase qui liste l'ensemble des

lettres minuscules. Comment obtenir les 15 premiers éléments à partir d'une notation type slice ?

In [166…
```
import string

elements = string.ascii_lowercase[:15]

print(elements)
```

```
abcdefghijklmno
```

3. On les divise en 5 sous-groupe pour un projet, comment peut-on adapter la structure de

données pour organiser les élèves en sous-groupes ?

> On peut organiser les sous-groupes dans une liste à deux dimensions (une liste de liste).

Commencer par diviser les étudiants en 5 groupes consécutifs.

In [166…
```
size_groupe = len(liste_eleves) // 5
liste_groupes_eleves = []
i = 0
while i+size_groupe <= len(liste_eleves):
    liste_groupes_eleves.append(liste_eleves[i:i+size_groupe])
    i += size_groupe

print(liste_groupes_eleves)
```

```
[['A', 'B', 'C'], ['D', 'E', 'F'], ['G', 'H', 'I'], ['J', 'K', 'L'], ['M
', 'N', 'O']]
```

Importer la librairie « random ».

In [166…
```
import random
```

Utiliser la fonction « choice » pour développer une fonction qui prend la liste originale et qui renvoie 5 groupes aléatoires.

In [166…
```
def create_groupes(liste):
    size_groupe = len(liste) // 5
    liste_groupes_eleves = []
    copie_liste = liste.copy()
    i = 0
    while i+size_groupe <= len(liste):
        sous_groupe = []
        for j in range(size_groupe):
            index_elem = copie_liste.index(random.choice(copie_liste))
            sous_groupe.append(copie_liste.pop(index_elem))
        liste_groupes_eleves.append(sous_groupe)
        i += size_groupe

    return liste_groupes_eleves

print(create_groupes(liste_eleves))
print(liste_eleves)
```

```
[['I', 'O', 'E'], ['L', 'N', 'B'], ['K', 'G', 'F'], ['C', 'M', 'H'], ['D
', 'J', 'A']]
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', '
O']
```

Utiliser la fonction « shuffle » pour organiser les groupes plus simplement.

In [166…
```
liste_groupes_eleves = []
i = 0
random.shuffle(liste_eleves)
while i+size_groupe <= len(liste_eleves):
    liste_groupes_eleves.append(liste_eleves[i:i+size_groupe])
    i += size_groupe

print(liste_groupes_eleves)
```

```
[['A', 'B', 'J'], ['I', 'C', 'E'], ['H', 'G', 'F'], ['M', 'L', 'N'], ['D
', 'O', 'K']]
```

4. Réorganiser la liste d'étudiants en 5 sous-groupes dans un dictionnaire en réutilisant la variable

originale.

In [167…
```
dico_groupes_eleves = {}

i = 0
j = 1
random.shuffle(liste_eleves)
while i+size_groupe <= len(liste_eleves):
    dico_groupes_eleves["Groupe_"+str(j)] = liste_eleves[i:i+size_groupe]
    i += size_groupe
    j += 1

print(dico_groupes_eleves)
```

```
{'Groupe_1': ['E', 'A', 'B'], 'Groupe_2': ['M', 'H', 'L'], 'Groupe_3': [
'F', 'G', 'D'], 'Groupe_4': ['N', 'I', 'C'], 'Groupe_5': ['J', 'O', 'K']
}
```

5. On veut maintenant réorganiser le dictionnaire pour assigner une note à chacun des

étudiants du sous-groupe. Le nouveau dictionnaire contient le numéro de groupe comme clé et la valeur correspond à un dictionnaire {liste_groupe : liste_notes}.

```
Comme les listes sont des objets mutables, on ne peut pas
se servir d'elles comme d'une clée.
Par conséquent, on transforme d'abord notre liste en objet
immutable (Tuple ou String).
```

In [167…
```
size_groupe = len(liste_eleves) // 5
liste_notes = []
for index in range(len(liste_eleves)):
    liste_notes.append(randint(0,20))

dico_notes_eleves = {}
i = 0
j = 1
while i+size_groupe <= len(liste_eleves):
    dico_notes_eleves["Groupe_"+str(j)] = liste_notes[i:i+size_groupe]
    i += size_groupe
    j += 1

print("dico_groupes_eleves", dico_groupes_eleves)
print()
print("dico_notes_eleves", dico_notes_eleves)
dico_notes_groupes_eleves = {key:{tuple(dico_groupes_eleves[key]):dico_no
print()
print("dico_notes_groupes_eleves",dico_notes_groupes_eleves)
```

```
dico_groupes_eleves {'Groupe_1': ['E', 'A', 'B'], 'Groupe_2': ['M', 'H',
'L'], 'Groupe_3': ['F', 'G', 'D'], 'Groupe_4': ['N', 'I', 'C'], 'Groupe_
5': ['J', 'O', 'K']}

dico_notes_eleves {'Groupe_1': [19, 4, 10], 'Groupe_2': [16, 13, 9], 'Gr
oupe_3': [3, 3, 17], 'Groupe_4': [7, 16, 3], 'Groupe_5': [3, 20, 4]}

dico_notes_groupes_eleves {'Groupe_1': {('E', 'A', 'B'): [19, 4, 10]}, '
Groupe_2': {('M', 'H', 'L'): [16, 13, 9]}, 'Groupe_3': {('F', 'G', 'D'):
[3, 3, 17]}, 'Groupe_4': {('N', 'I', 'C'): [7, 16, 3]}, 'Groupe_5': {('J
', 'O', 'K'): [3, 20, 4]}}
```

6. Ajouter une clé 'moyenne' à chaque dictionnaire contenant les notes. Comment arrondir la

moyenne à deux décimales ?.

Pour arrondir la moyenne à deux décimale on utilise la fonction round(n,m) qui arrondie le nombre n à m chiffres après la virgule.

On force une copie des items du dictionnaire que l'on modifie en parcourant car on ne peut pas ajouter ou retirer des élèments dans un dictionnaire pendant que l'on itère dessus.

In [167…

```
def moyenne(liste_note):
    return sum(liste_note)/len(liste_note)


for dico_notes_groupes_eleves_key,dico_notes_groupes_eleves_value in dico
    for(dico_notes_eleves_key, dico_notes_eleves_value) in list(dico_note
        dico_notes_groupes_eleves_value["Moyenne"] = round(moyenne(dico_n

print(dico_notes_groupes_eleves)
```

```
{'Groupe_1': {('E', 'A', 'B'): [19, 4, 10], 'Moyenne': 11.0}, 'Groupe_2'
: {('M', 'H', 'L'): [16, 13, 9], 'Moyenne': 12.67}, 'Groupe_3': {('F', '
G', 'D'): [3, 3, 17], 'Moyenne': 7.67}, 'Groupe_4': {('N', 'I', 'C'): [7
, 16, 3], 'Moyenne': 8.67}, 'Groupe_5': {('J', 'O', 'K'): [3, 20, 4], 'M
oyenne': 9.0}}
```

7. Ecrire les instructions permettant de mélanger les items du dictionnaire construit dans la

question 6) (pensez à appliquer un « shuffle » sur les clés).

Pour mélanger les items d'un dictionnaire, on recupère d'abord les clés sous forme de liste, on mélange et ensuite on ajoute les clés que l'on vient de mélanger dans un nouveau dictionnaire.

In [167…
```
print("Dictionnaire avant avoir mélangé :")
print(dico_notes_groupes_eleves)

key_list = list(dico_notes_groupes_eleves.keys())
random.shuffle(key_list)

shuffled_dico_notes_groupes_eleves = dict()

for key in key_list:
    shuffled_dico_notes_groupes_eleves[key] = dico_notes_groupes_eleves[k

print("Dictionnaire après avoir mélangé :")
print(shuffled_dico_notes_groupes_eleves)
```

```
Dictionnaire avant avoir mélangé :
{'Groupe_1': {('E', 'A', 'B'): [19, 4, 10], 'Moyenne': 11.0}, 'Groupe_2'
: {('M', 'H', 'L'): [16, 13, 9], 'Moyenne': 12.67}, 'Groupe_3': {('F', '
G', 'D'): [3, 3, 17], 'Moyenne': 7.67}, 'Groupe_4': {('N', 'I', 'C'): [7
, 16, 3], 'Moyenne': 8.67}, 'Groupe_5': {('J', 'O', 'K'): [3, 20, 4], 'M
oyenne': 9.0}}
Dictionnaire après avoir mélangé :
{'Groupe_3': {('F', 'G', 'D'): [3, 3, 17], 'Moyenne': 7.67}, 'Groupe_2':
{('M', 'H', 'L'): [16, 13, 9], 'Moyenne': 12.67}, 'Groupe_4': {('N', 'I'
, 'C'): [7, 16, 3], 'Moyenne': 8.67}, 'Groupe_5': {('J', 'O', 'K'): [3,
20, 4], 'Moyenne': 9.0}, 'Groupe_1': {('E', 'A', 'B'): [19, 4, 10], 'Moy
enne': 11.0}}
```

8. Trouver un moyen de déterminer le groupe avec la moyenne minimale, maximale ?

In [167…

```python
def find_groupe_min(dico):
    min = 20

    for key, value in dico.items():
        if value["Moyenne"] <= min:
            min = value["Moyenne"]
            groupe = key

    return groupe

def find_groupe_max(dico):
    max = 0

    for key, value in dico.items():
        if value["Moyenne"] >= max:
            max = value["Moyenne"]
            groupe = key

    return groupe

for k, v in dico_notes_groupes_eleves.items():
    print(k, v)

print(find_groupe_min(dico_notes_groupes_eleves))
print(find_groupe_max(dico_notes_groupes_eleves))
```

```
Groupe_1 {('E', 'A', 'B'): [19, 4, 10], 'Moyenne': 11.0}
Groupe_2 {('M', 'H', 'L'): [16, 13, 9], 'Moyenne': 12.67}
Groupe_3 {('F', 'G', 'D'): [3, 3, 17], 'Moyenne': 7.67}
Groupe_4 {('N', 'I', 'C'): [7, 16, 3], 'Moyenne': 8.67}
Groupe_5 {('J', 'O', 'K'): [3, 20, 4], 'Moyenne': 9.0}
Groupe_3
Groupe_2
```