

EXERCICES PYTHON

Exercice 1 -

Écrire une fonction `max_et_indice` qui prend en paramètre un tableau non vide `tab` (type Python list) de nombres entiers et qui renvoie la valeur du plus grand élément de ce tableau ainsi que l'indice de sa première apparition dans ce tableau.

L'utilisation de la fonction native `max` n'est pas autorisée.

Exemples :

```
>>> max_et_indice([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
(9, 3)
>>> max_et_indice([-2])
(-2, 0)
>>> max_et_indice([-1, -1, 3, 3, 3])
(3, 2)
>>> max_et_indice([1, 1, 1, 1])
(1, 0)
```

Exercice 2 -

On s'intéresse à la suite d'entiers définie par :

- les deux premières valeurs sont égales à 1 ;
- ensuite, chaque valeur est obtenue en faisant la somme des deux valeurs qui la précèdent.

La troisième valeur est donc $1+1 = 2$, la quatrième est $1+2 = 3$, la cinquième est $2+3 = 5$, la sixième est $3 + 5 = 8$, et ainsi de suite.

Cette suite d'entiers est connue sous le nom de suite de Fibonacci.

Écrire en Python une fonction `fibonacci` qui prend en paramètre un entier `n` supposé strictement positif et qui renvoie le terme d'indice `n` de cette suite.

Exemples :

```
>>> fibonacci(1)
1
>>> fibonacci(2)
1
>>> fibonacci(25)
75025
```

Exercice 3 -

Écrire une fonction `ecriture_binaire_entier_positif` qui prend en paramètre un entier positif `n` et renvoie une chaîne de caractère correspondant à l'écriture binaire de `n`.

On rappelle que :

- l'écriture binaire de 25 est 11001 car $25 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$;
- $n \% 2$ vaut 0 ou 1 selon que `n` est pair ou impair ;
- $n // 2$ donne le quotient de la division euclidienne de `n` par 2.

Il est interdit dans cet exercice d'utiliser la fonction `bin` de Python.

Exemples :

```
>>> 5 % 2
1
>>> 5 // 2
2
>>> ecriture_binaire_entier_positif(0)
'0'
>>> ecriture_binaire_entier_positif(2)
'10'
>>> ecriture_binaire_entier_positif(105)
'1101001'
```

Exercice 4 -

Programmer une fonction renverse, prenant en paramètre une chaîne de caractères non vide et renvoie cette chaîne de caractères en ordre inverse.

Exemple :

```
>>> renverse("")
""
>>> renverse("abc")
"cba"
>>> renverse("informatique")
"euqitamrofni"
```

Exercice 5 -

On rappelle que :

- le nombre a^n est le nombre $a \times a \times a \times \dots \times a$, où le facteur a apparaît n fois,
- en langage Python, l'instruction `t[-1]` permet d'accéder au dernier élément du tableau `t`.

Dans cet exercice, l'opérateur `**` et la fonction `pow` ne sont pas autorisés.

Programmer en langage Python une fonction `liste_puissances` qui prend en argument un nombre entier a , un entier strictement positif n et qui renvoie la liste de ses puissances $[a^1, a^2, \dots, a^n]$.

Programmer également une fonction `liste_puissances_borne` qui prend en argument un nombre entier a supérieur ou égal à 2 et un entier borne, et qui renvoie la liste de ses puissances, à l'exclusion de a^0 , strictement inférieures à borne.

Exemples :

```
>>> liste_puissances(3, 5)
[3, 9, 27, 81, 243]
>>> liste_puissances(-2, 4)
[-2, 4, -8, 16]
>>> liste_puissances_borne(2, 16)
[2, 4, 8]
>>> liste_puissances_borne(2, 17)
[2, 4, 8, 16]
>>> liste_puissances_borne(5, 5)
[]
```

Exercice 6 -

Le nombre d'occurrences d'un caractère dans une chaîne de caractère est le nombre d'ap-

paritions de ce caractère dans la chaîne.

Exemples :

- le nombre d'occurrences du caractère 'o' dans 'bonjour' est 2 ;
- le nombre d'occurrences du caractère 'b' dans 'Bébé' est 1 ;
- le nombre d'occurrences du caractère 'B' dans 'Bébé' est 1 ;
- le nombre d'occurrences du caractère ' ' dans 'Hello world !' est 2.

On cherche les occurrences des caractères dans une phrase. On souhaite stocker ces occur-

rences dans un dictionnaire dont les clefs seraient les caractères de la phrase et les valeurs

l'occurrence de ces caractères.

Par exemple : avec la phrase 'Hello world !' le dictionnaire est le suivant :

```
{ 'H': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 2, 'w': 1, 'r': 1, 'd': 1, '!': 1 }
```

L'ordre des clefs n'a pas d'importance.

Écrire une fonction `nbr_occurrences` prenant comme paramètre une chaîne de caractères `chaine` et renvoyant le dictionnaire des nombres d'occurrences des caractères de cette chaîne.

Exercice 7 -

Écrire la fonction `maximum_tableau`, prenant en paramètre un tableau non vide de nombres `tab` (de type `list`) et renvoyant le plus grand élément de ce tableau.

Exemples :

```
>>> maximum_tableau([98, 12, 104, 23, 131, 9])
131
>>> maximum_tableau([-27, 24, -3, 15])
24
```

Exercice 8 -

Programmer la fonction `multiplication`, prenant en paramètres deux nombres entiers relatifs `n1` et `n2`, et qui renvoie le produit de ces deux nombres.

Les seules opérations autorisées sont l'addition et la soustraction.

```
>>> multiplication(3, 5)
15
>>> multiplication(-4, -8)
32
>>> multiplication(-2, 6)
-12
>>> multiplication(-2, 0)
0
```

Exercice 9 -

Écrire une fonction `recherche` qui prend en paramètres un tableau `tab` de nombres entiers triés par ordre croissant et un nombre entier `n`, et qui effectue une recherche dichotomique du nombre entier `n` dans le tableau non vide `tab`.

Cette fonction doit renvoyer un indice correspondant au nombre cherché s'il est dans le tableau, `None` sinon.

Exemples :

```
>>> recherche([2, 3, 4, 5, 6], 5)
3
>>> recherche([2, 3, 4, 6, 7], 5) # renvoie None
```

Exercice 10 -

Un arbre binaire est soit vide, représenté en Python par la valeur `None`, soit un nœud représenté par un triplet `(g, x, d)` où `x` est l'étiquette du nœud et `g` et `d` sont les sous-arbres gauche et droit.

On souhaite écrire une fonction `parcours_largeur` qui prend en paramètre un arbre binaire et qui renvoie la liste des étiquettes des nœuds de l'arbre parcourus en largeur.

Exemples :

```
>>> arbre = ( ( (None, 1, None), 2, (None, 3, None) ),
4,
( (None, 5, None), 6, (None, 7, None) ) )
>>> parcours_largeur(arbre)
[4, 2, 6, 1, 3, 5, 7]
```

Exercice 11 -

Programmer la fonction fusion prenant en paramètres deux tableaux non vides tab1 et tab2 (type list) d'entiers, chacun dans l'ordre croissant, et renvoyant un tableau trié dans l'ordre croissant et contenant l'ensemble des valeurs de tab1 et tab2.

Exemples :

```
>>> fusion([3, 5], [2, 5])
[2, 3, 5, 5]
>>> fusion([-2, 4], [-3, 5, 10])
[-3, -2, 4, 5, 10]
>>> fusion([4], [2, 6])
[2, 4, 6]
>>> fusion([], [])
[]
>>> fusion([1, 2, 3], [])
[1, 2, 3]
```

Exercice 12 -

Écrire une fonction recherche qui prend en paramètres elt nombre entier et tab un tableau de nombres entiers (type list), et qui renvoie l'indice de la première occurrence de elt dans tab si elt est dans tab et None sinon.

L'objectif de cet exercice est de parcourir un tableau, il est interdit d'utiliser la méthode index des listes Python.

Exemples :

```
>>> recherche(1, [2, 3, 4]) # renvoie None
>>> recherche(1, [10, 12, 1, 56])
2
>>> recherche(50, [1, 50, 1])
1
>>> recherche(15, [8, 9, 10, 15])
3
```

Exercice 13 -

Écrire en python deux fonctions :

- lancer de paramètre n, un entier positif, qui renvoie un tableau de n entiers obtenus aléatoirement entre 1 et 6 (1 et 6 inclus) ;
- paire_6 de paramètre tab, un tableau de n entiers compris entre 1 et 6 et qui renvoie un booléen égal à True si le nombre de 6 est supérieur ou égal à 2, False sinon.

On pourra utiliser la fonction randint(a,b) du module random pour laquelle la documentation officielle est la suivante :

random.randint(a, b)

Renvoie un entier aléatoire N tel que $a \leq N \leq b$.

Exemples :

```
>>> lancer1 = lancer(5)
>>> lancer1
[5, 6, 6, 2, 2]
>>> paire_6(lancer1)
True
>>> lancer2 = lancer(5)
>>> lancer2
[6, 5, 1, 6, 6]
>>> paire_6(lancer2)
True
>>> lancer3 = lancer(3)
>>> lancer3
[2, 2, 6]
>>> paire_6(lancer3)
False
```

```
>>> lancer4 = lancer(0)
>>> lancer4
[]
>>> paire_6(lancer4)
False
```

Exercice 14 -

Programmer la fonction multiplication qui en paramètres deux nombres entiers relatifs n1 et n2, et qui renvoie le produit de ces deux nombres.

Les seules opérations arithmétiques autorisées sont l'addition et la soustraction.

Exemples :

```
>>> multiplication(3, 5)
15
>>> multiplication(-4, -8)
32
>>> multiplication(-2, 6)
-12
>>> multiplication(-2, 0)
0
```

Exercice 15 -

Écrire une fonction moyenne(notes) qui renvoie la moyenne pondérée des résultats contenus dans le tableau notes, non vide, donné en paramètre. Ce tableau contient des couples (note, coefficient) dans lesquels :

- note est un nombre de type flottant (float) compris entre 0 et 20 ;
- coefficient est un nombre entier strictement positif.

Ainsi l'expression moyenne([(15.0,2),(9.0,1),(12.0,3)]) devra renvoyer 12.5 comme résultat du calcul suivant :

$$\frac{2 \times 15 + 1 \times 9 + 3 \times 12}{2 + 1 + 3} = 12,5$$

Exercice 16 -

Écrire une fonction moyenne qui prend en paramètre un tableau d'entiers non vide et qui renvoie un nombre flottant donnant la moyenne de ces entiers.

Attention : il est interdit d'utiliser la fonction sum ou la fonction mean (module statistics) de Python.

Exemples

```
>>> moyenne([1])
1.0
>>> moyenne([1, 2, 3, 4, 5, 6, 7])
4.0
>>> moyenne([1, 2])
1.5
```

Exercice 17 -

Écrire une fonction recherche_min qui prend en paramètre un tableau de nombres tab non vide, et qui renvoie l'indice de la première occurrence du minimum de ce tableau. Les tableaux seront représentés sous forme de liste Python.

Exemples :

```
>>> recherche_min([5])
0
>>> recherche_min([2, 4, 1])
2
>>> recherche_min([5, 3, 2, 2, 4])
2
>>> recherche_min([-1, -2, -3, -3])
2
```

Exercice 18 -

Écrire une fonction `min_et_max` qui prend en paramètre un tableau de nombres `tab` non vide, et qui renvoie la plus petite et la plus grande valeur du tableau sous la forme d'un dictionnaire à deux clés `min` et `max`.

Les tableaux seront représentés sous forme de liste Python.

L'utilisation des fonctions natives `min`, `max` et `sorted`, ainsi que la méthode `sort` n'est pas autorisée.

Exemples :

```
>>> min_et_max([0, 1, 4, 2, -2, 9, 3, 1, 7, 1])
{'min': -2, 'max': 9}
>>> min_et_max([0, 1, 2, 3])
{'min': 0, 'max': 3}
>>> min_et_max([3])
{'min': 3, 'max': 3}
>>> min_et_max([1, 3, 2, 1, 3])
{'min': 1, 'max': 3}
>>> min_et_max([-1, -1, -1, -1, -1])
{'min': -1, 'max': -1}
```

Exercice 19 -

Écrire une fonction `indices_maxi` qui prend en paramètre un tableau non vide de nombre entiers `tab`, représenté par une liste Python et qui renvoie un tuple (`maxi`, `indices`)

où :

- `maxi` est le plus grand élément du tableau `tab` ;
- `indices` est une liste Python contenant les indices du tableau `tab` où apparaît ce plus grand élément.

Exemple :

```
>>> indices_maxi([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
(9, [3, 8])
>>> indices_maxi([7])
(7, [0])
```

Exercice 20 -

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers (type list), et qui renvoie l'indice de la dernière occurrence de `elt` dans `tab` si `elt` est dans `tab` et `None` sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4]) # renvoie None
>>> recherche(1, [10, 12, 1, 56])
2
>>> recherche(1, [1, 0, 42, 7])
0
>>> recherche(1, [1, 50, 1])
2
>>> recherche(1, [8, 1, 10, 1, 7, 1, 8])
5
```

Exercice 21 -

On veut trier par ordre croissant les notes d'une évaluation qui sont des nombres entiers compris entre 0 et 10 (inclus).

Ces notes sont contenues dans un tableau `notes_eval` (type list).

Écrire une fonction `effectif_notes` prenant en paramètre le tableau `notes_eval` et renvoyant un tableau de longueur 11 tel que la valeur d'indice `i` soit le nombre de notes valant `i` dans le tableau `notes_eval`.

Écrire ensuite une fonction `notes_triees` prenant en paramètre le tableau des effectifs des notes et renvoyant un tableau contenant les mêmes valeurs que `notes_eval` mais triées dans l'ordre croissant.

Exemple :

```
>>> notes_eval = [2, 0, 5, 9, 6, 9, 10, 5, 7,
9, 9, 5, 0, 9, 6, 5, 4]
>>> eff = effectif_notes(notes_eval)
>>> eff
[2, 0, 1, 0, 1, 4, 2, 1, 0, 5, 1]
>>> notes_triees(eff)
[0, 0, 2, 4, 5, 5, 5, 5, 6, 6, 7, 9, 9, 9, 9, 9, 10]
```

Exercice 22 -

Écrire une fonction `enumere` qui prend en paramètre un tableau `tab` (type list) et renvoie un dictionnaire `d` dont les clés sont les éléments de `tab` avec pour valeur associée la liste des indices de l'élément dans le tableau `tab`.

Exemple :

```
>>> enumere([])
{}
>>> enumere([1, 2, 3])
{1: [0], 2: [1], 3: [2]}
>>> enumere([1, 1, 2, 3, 2, 1])
{1: [0, 1, 5], 2: [2, 4], 3: [3]}
```

Exercice 23 -

On a relevé les valeurs moyennes annuelles des températures à Paris pour la période allant de 2013 à 2019. Les résultats ont été récupérés sous la forme de deux tableaux (de type list) : l'un pour les températures, l'autre pour les années :

```
t_moy = [14.9, 13.3, 13.1, 12.5, 13.0, 13.6, 13.7]
annees = [2013, 2014, 2015, 2016, 2017, 2018, 2019]
```

Écrire la fonction `annee_temperature_minimale` qui prend en paramètres ces deux tableaux et qui renvoie la plus petite valeur relevée au cours de la période et l'année correspondante.

On suppose que la température minimale est atteinte une seule fois.

Exemple :

```
>>> annee_temperature_minimale(t_moy, annees)
(12.5, 2016)
```

Exercice 24 -

Écrire une fonction `ajoute_dictionnaires` qui prend en paramètres deux dictionnaires `d1` et `d2` dont les clés sont des nombres et renvoie le dictionnaire `d` défini de la façon suivante :

- les clés de `d` sont celles de `d1` et celles de `d2` réunies ;
- si une clé est présente dans les deux dictionnaires `d1` et `d2`, sa valeur associée dans le dictionnaire `d` est la somme de ses valeurs dans les dictionnaires `d1` et `d2` ;
- si une clé n'est présente que dans un des deux dictionnaires, sa valeur associée dans le dictionnaire `d` est la même que sa valeur dans le dictionnaire où elle est présente.

Exemples :

```
>>> ajoute_dictionnaires({1: 5, 2: 7}, {2: 9, 3: 11})
{1: 5, 2: 16, 3: 11}
>>> ajoute_dictionnaires({}, {2: 9, 3: 11})
{2: 9, 3: 11}
>>> ajoute_dictionnaires({1: 5, 2: 7}, {})
{1: 5, 2: 7}
```

Exercice 25 -

Écrire une fonction `verifie` qui prend en paramètre un tableau de valeurs numériques et qui renvoie `True` si ce tableau est trié dans l'ordre croissant, `False` sinon.

Un tableau vide est considéré comme trié.

Exemples :

```
>>> verifie([0, 5, 8, 8, 9])
True
>>> verifie([8, 12, 4])
False
>>> verifie([-1, 4])
True
>>> verifie([])
True
>>> verifie([5])
True
```

Exercice 26 -

Écrire une fonction `a_doublon` qui prend en paramètre un tableau trié de nombres dans l'ordre croissant et renvoie `True` si ce tableau contient au moins deux nombres identiques, `False` sinon.

Exemple :

```
>>> a_doublon([])
False
>>> a_doublon([1])
False
>>> a_doublon([1, 2, 4, 6, 6])
True
>>> a_doublon([2, 5, 7, 7, 7, 9])
True
>>> a_doublon([0, 2, 3])
False
```

Exercice 27 -

On considère des tables, c'est-à-dire des tableaux de dictionnaires ayant tous les mêmes clés, qui contiennent des enregistrements relatifs à des animaux hébergés dans un refuge. Les attributs des enregistrements sont 'nom', 'espece', 'age', 'enclos'.

Voici un exemple d'une telle table :

```
animaux = [ {'nom':'Medor', 'espece':'chien', 'age':5, 'enclos':2},
             {'nom':'Titine', 'espece':'chat', 'age':2, 'enclos':5},
             {'nom':'Tom', 'espece':'chat', 'age':7, 'enclos':4},
             {'nom':'Belle', 'espece':'chien', 'age':6, 'enclos':3},
             {'nom':'Mirza', 'espece':'chat', 'age':6, 'enclos':5}]
```

Programmer une fonction `selection_enclos` qui :

- prend en paramètres :
 - une table `animaux` contenant des enregistrements relatifs à des animaux (comme dans l'exemple ci-dessus),
 - un numéro d'enclos `num_enclos` ;
- renvoie une table contenant les enregistrements de animaux dont l'attribut 'enclos' est `num_enclos`.

Exemples avec la table `animaux` ci-dessus :

```
>>> selection_enclos(animaux, 5)
[{'nom':'Titine', 'espece':'chat', 'age':2, 'enclos':5},
 {'nom':'Mirza', 'espece':'chat', 'age':6, 'enclos':5}]
>>> selection_enclos(animaux, 2)
[{'nom':'Medor', 'espece':'chien', 'age':5, 'enclos':2}]
>>> selection_enclos(animaux, 7)
[]
```

Exercice 28 -

Le codage par différence (delta encoding en anglais) permet de compresser un tableau de données en indiquant pour chaque donnée, sa différence avec la précédente (plutôt que la donnée elle-même). On se retrouve alors avec un tableau de données plus petit, nécessitant moins de place en mémoire. Cette méthode se révèle efficace lorsque les valeurs consécutives sont proches.

Programmer la fonction `delta(liste)` qui prend en paramètre un tableau non vide de nombres entiers et qui renvoie un tableau contenant les valeurs entières compressées à l'aide cette technique.

Exemples :

```
>>> delta([1000, 800, 802, 1000, 1003])
[1000, -200, 2, 198, 3]
>>> delta([42])
[42]
```

Exercice 29 -

Écrire une fonction `recherche_motif` qui prend en paramètre une chaîne de caractères `motif` non vide et une chaîne de caractères `texte` et qui renvoie la liste des positions de `motif` dans `texte`. Si `motif` n'apparaît pas, la fonction renvoie une liste vide.

Exemples:

```
>>> recherche_motif("ab", "")
[]
>>> recherche_motif("ab", "cdcdcdcd")
[]
>>> recherche_motif("ab", "abracadabra")
[0, 7]
>>> recherche_motif("ab", "abracadabraab")
[0, 7, 11]
```

Exercice 30 -

Écrire une fonction `occurrences(caractere, chaine)` qui prend en paramètres `caractere`, une chaîne de caractère de longueur 1, et `chaine`, une chaîne de caractères.

Cette fonction renvoie le nombre d'occurrences de `caractere` dans `chaine`, c'est-à-dire le nombre de fois où `caractere` apparaît dans `chaine`.

Exemples :

```
>>> occurrences('e', "sciences")
2
>>> occurrences('i', "mississippi")
4
>>> occurrences('a', "mississippi")
0
```

Exercice 31 -

Écrire une fonction `tri_selection` qui prend en paramètre un tableau `tab` de nombres entiers (type `list`) et qui le modifie afin qu'il soit trié par ordre croissant.

On utilisera l'algorithme suivant :

- on recherche le plus petit élément du tableau, en le parcourant du rang 0 au dernier rang, et on l'échange avec l'élément d'indice 0 ;
- on recherche ensuite le plus petit élément du tableau restreint du rang 1 au dernier rang, et on l'échange avec l'élément d'indice 1 ;
- on continue de cette façon jusqu'à ce que le tableau soit entièrement triée.

Exemple :

```
>>> tab = [1, 52, 6, -9, 12]
>>> tri_selection(tab)
>>> tab
[-9, 1, 6, 12, 52]
```

Exercice 32 -

Sur le réseau social TipTop, on s'intéresse au nombre de « like » des abonnés. Les données sont stockées dans des dictionnaires où les clés sont les pseudos et les valeurs correspon-

dantes sont les nombres de « like » comme ci-dessous :

```
{ 'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50 }
```

Écrire une fonction `max_dico` qui :

- prend en paramètre un dictionnaire `dico` non vide dont les clés sont des chaînes de caractères et les valeurs associées sont des entiers ;
- et qui renvoie un tuple dont :
 - la première valeur est la clé du dictionnaire associée à la valeur maximale ;
 - la seconde valeur est la première valeur maximale présente dans le dictionnaire.

Exemples :

```
>>> max_dico({ 'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50 })
('Ada', 201)
>>> max_dico({ 'Alan': 222, 'Ada': 201, 'Eve': 222, 'Tim': 50 })
('Alan', 222)
```

Exercice 33 -

Dans cet exercice, on considère des phrases composées de mots.

- On appelle mot une chaîne de caractères composée avec des caractères choisis parmi les 26 lettres minuscules ou majuscules de l'alphabet.

- On appelle phrase une chaîne de caractères :

- composée avec un ou plusieurs mots séparés entre eux par un seul caractère espace ' ',

- se finissant :

- * soit par un point '.' qui est alors collé au dernier mot,

- * soit par un point d'exclamation '!' ou d'interrogation '?' qui est alors séparé du dernier mot par un seul caractère espace ' '.

Voici deux exemples de phrases :

```
'Cet exercice est simple.'
```

```
'Le point d exclamation est separe !'
```

Après avoir remarqué le lien entre le nombre de mots et le nombre de caractères espace dans une phrase, programmer une fonction `nombre_de_mots` qui prend en paramètre une phrase et renvoie le nombre de mots présents dans cette phrase.

```
>>> nombre_de_mots('Cet exercice est simple.')
```

```
4
```

```
>>> nombre_de_mots('Le point d exclamation est séparé !')
```

```
6
```

```
>>> nombre_de_mots('Combien de mots y a t il dans cette phrase ?')
```

```
10
```

```
>>> nombre_de_mots('Fin.')
```

```
1
```

Exercice 34 -

On considère dans cet exercice une représentation binaire d'un entier non signé en tant que tableau de booléens.

Si

```
tab = [True, False, True, False, False, True, True]
```

est un tel tableau, alors l'entier qu'il représente est $26 + 24 + 21 + 20 = 83$. Cette représentation consistant à placer en premier le booléen indiquant la puissance la plus élevée de 2 est dite big-endian ou grand-boutiste.

Écrire une fonction `gb_vers_entier` qui prend en paramètre un tel tableau et renvoie l'entier qu'il représente.

Exemple :

```
>>> gb_vers_entier([])
```

```
0
```

```
>>> gb_vers_entier([True])
```

```
1
```

```
>>> gb_vers_entier([True, False, True, False, False, True, True])
```

```
83
```

```
>>> gb_vers_entier([True, False, False, False, False, False, True, False])
```

```
130
```

Exercice 35 -

Écrire une fonction moyenne qui prend en paramètre un tableau non vide de nombres flottants et qui renvoie la moyenne des valeurs du tableau. Les tableaux seront représentés sous forme de liste Python.

Exemples :

```
>>> moyenne([1.0])
1.0
>>> moyenne([1.0, 2.0, 4.0])
2.3333333333333335
```

Exercice 36 -

Programmer la fonction moyenne prenant en paramètre un tableau d'entiers tab (de type list) qui renvoie la moyenne de ses éléments si le tableau est non vide. Proposer une façon de traiter le cas où le tableau passé en paramètre est vide.

Dans cet exercice, on s'interdira d'utiliser la fonction Python sum.

Exemples :

```
>>> moyenne([5, 3, 8])
5.333333333333333
>>> moyenne([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
5.5
>>> moyenne([])
# Comportement différent suivant le traitement proposé.
```

Exercice 37 -

Écrire une fonction recherche_indices_classement qui prend en paramètres un entier elt et un tableau d'entiers tab représenté par une liste Python, et qui renvoie trois listes Python d'entiers:

- la première liste contient les indices des valeurs du tableau tab strictement inférieures à elt ;
- la deuxième liste contient les indices des valeurs du tableau tab égales à elt ;
- la troisième liste contient les indices des valeurs du tableau tab strictement supérieures à elt.

Exemples :

```
>>> recherche_indices_classement(3, [1, 3, 4, 2, 4, 6, 3, 0])
([0, 3, 7], [1, 6], [2, 4, 5])
>>> recherche_indices_classement(3, [1, 4, 2, 4, 6, 0])
([0, 2, 5], [], [1, 3, 4])
>>> recherche_indices_classement(3, [1, 1, 1, 1])
([0, 1, 2, 3], [], [])
>>> recherche_indices_classement(3, [])
([], [], [])
```

Exercice 38 -

L'opérateur « ou exclusif » entre deux bits renvoie 0 si les deux bits sont égaux et 1 s'ils sont différents. Il est symbolisé par le symbole $\oplus$. Ainsi :

- $0 \oplus 0 = 0$
- $0 \oplus 1 = 1$
- $1 \oplus 0 = 1$
- $1 \oplus 1 = 0$

Écrire une fonction `ou_exclusif` qui prend en paramètres deux tableaux de 0 ou de 1 de même longueur et qui renvoie un tableau où l'élément situé à position *i* est le résultat, par l'opérateur « ou exclusif », des éléments à la position *i* des tableaux passés en paramètres.

Exemples :

```
>>> ou_exclusif([1, 0, 1, 0, 1, 1, 0, 1], [0, 1, 1, 1, 0, 1, 0, 0])
[1, 1, 0, 1, 1, 0, 0, 1]
>>> ou_exclusif([1, 1, 0, 1], [0, 0, 1, 1])
[1, 1, 1, 0]
```

Exercice 39 -

Écrire une fonction Python appelée `nb_repetitions` qui prend en paramètres un élément `elt` et un tableau `tab` (type `list`) d'éléments du même type et qui renvoie le nombre de fois où l'élément apparaît dans le tableau.

Exemples :

```
>>> nb_repetitions(5, [2, 5, 3, 5, 6, 9, 5])
3
>>> nb_repetitions('A', ['B', 'A', 'B', 'A', 'R'])
2
>>> nb_repetitions(12, [1, '!', 7, 21, 36, 44])
0
```

Exercice 40 -

Écrire une fonction `couples_consecutifs` qui prend en paramètre un tableau de nombres entiers `tab` non vide (type `list`), et qui renvoie la liste Python (éventuellement vide) des couples d'entiers consécutifs successifs qu'il peut y avoir dans `tab`.

Exemples :

```
>>> couples_consecutifs([1, 4, 3, 5])
[]
>>> couples_consecutifs([1, 4, 5, 3])
[(4, 5)]
>>> couples_consecutifs([1, 1, 2, 4])
[(1, 2)]
>>> couples_consecutifs([7, 1, 2, 5, 3, 4])
[(1, 2), (3, 4)]
>>> couples_consecutifs([5, 1, 2, 3, 8, -5, -4, 7])
[(1, 2), (2, 3), (-5, -4)]
```

Exercice 41 -

Dans cet exercice on cherche à calculer la moyenne pondérée d'un élève dans une matière

donnée. Chaque note est associée à un coefficient qui la pondère.

Par exemple, si ses notes sont : 14 avec coefficient 3, 12 avec coefficient 1 et 16 avec coeffi-

cient 2, sa moyenne pondérée sera donnée par

$$\frac{14 \times 3 + 12 \times 1 + 16 \times 2}{3 + 1 + 2} = 14,333...$$

Écrire une fonction moyenne :

- qui prend en paramètre une liste notes non vide de tuples à deux éléments entiers de la forme (note, coefficient) (int ou float) positifs ou nuls ;
- et qui renvoie la moyenne pondérée des notes de la liste sous forme de flottant si la somme des coefficients est non nulle, None sinon.

Exemple :

```
>>> moyenne([(8, 2), (12, 0), (13.5, 1), (5, 0.5)])
9.142857142857142
>>> moyenne([(3, 0), (5, 0)])
None
```

Exercice 42 -

On considère des chaînes de caractères contenant uniquement des majuscules et des caractères * appelées mots à trous.

Par exemple INFO*MA*IQUE, ***|***E** et *S* sont des mots à trous.

Programmer une fonction correspond :

- qui prend en paramètres deux chaînes de caractères mot et mot_a_trous où mot_a_trous est un mot à trous comme indiqué ci-dessus ;
- et qui renvoie :
 - True si on peut obtenir mot en remplaçant convenablement les caractères '*' de mot_a_trous ;
 - False sinon.

Exemple :

```
>>> correspond('INFORMATIQUE', 'INFO*MA*IQUE')
True
>>> correspond('AUTOMATIQUE', 'INFO*MA*IQUE')
False
>>> correspond('STOP', 'S*')
False
>>> correspond('AUTO', '*UT*')
True
```

Exercice 43 -

Écrire une fonction compte_occurrences prenant en paramètres une valeur x et un tableau tab (de type list) et renvoyant le nombre d'occurrences de x dans tab.

L'objectif de cet exercice étant de parcourir un tableau, il est interdit d'utiliser la méthode count des listes Python.

Exemples :

```
>>> compte_occurrences(5, [])
0
>>> compte_occurrences(5, [-2, 3, 1, 5, 3, 7, 4])
1
>>> compte_occurrences('a', ['a', 'b', 'c', 'a', 'd', 'e', 'a'])
3
```

Exercice 44 -

Programmer la fonction recherche, prenant en paramètre un tableau non vide tab (type list) d'entiers et un entier n, et qui renvoie l'indice de la dernière occurrence de l'élément cherché. Si l'élément n'est pas présent, la fonction renvoie None.

Exemples

```
>>> recherche([5, 3], 1) # renvoie None
>>> recherche([2, 4], 2)
0
>>> recherche([2, 3, 5, 2, 4], 2)
3
```