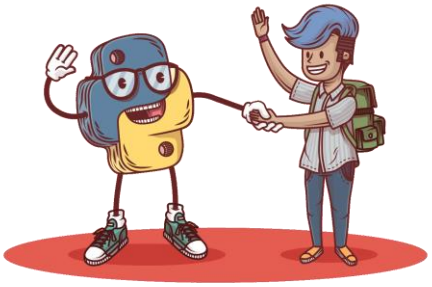
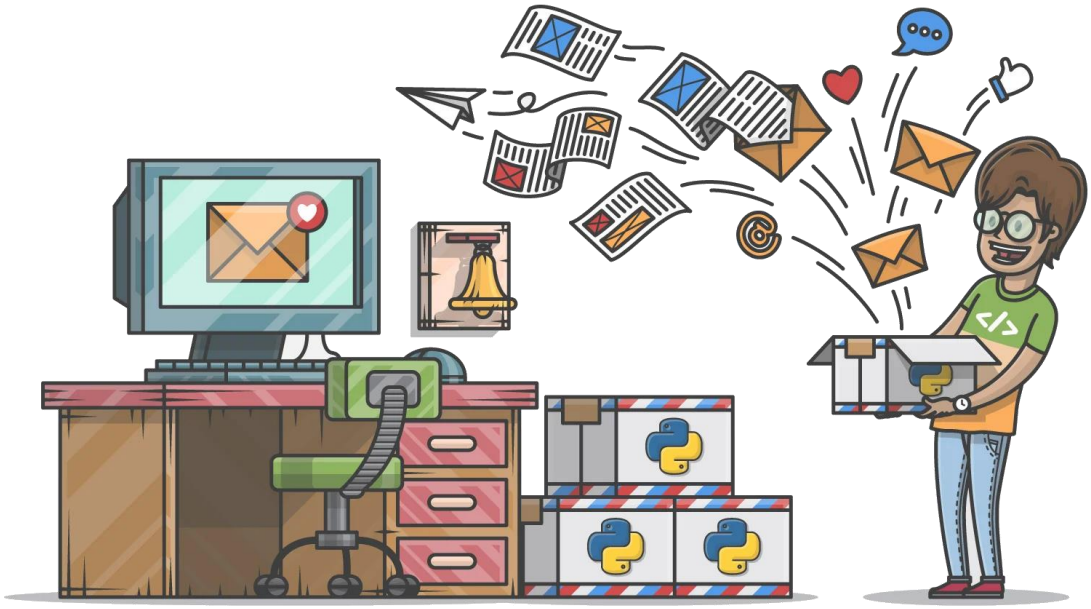


Mon Cahier d'entrainement

NSI 2024-2025

Une série de 30 exercices indispensables.

Des activités pour réviser les notions importantes.



DarkSATHI Li

Nom : ..... Prénom : .....

Exercice 1

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

Exemples :

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

Exercice 2

Écrire une fonction **`a_doublon`** qui prend en paramètre une liste triée de nombres et renvoie **`True`** si la liste contient au moins deux nombres identiques, **`False`** sinon.

Exemples :

```
>>> 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 3

On souhaite générer des grilles du jeu de démineur à partir de la position des bombes à placer.
On se limite à la génération de grilles carrées de taille $n \times n$ où n est le nombre de bombes du jeu.
Dans le jeu du démineur, chaque case de la grille contient soit une bombe, soit une valeur qui correspond aux nombres de bombes situées dans le voisinage direct de la case (au-dessus, en dessous, à droite, à gauche ou en diagonal : chaque case a donc 8 voisins si elle n'est pas située au bord de la grille).

Voici un exemple de grille 5×5 de démineur dans laquelle la bombe est représentée par une étoile :

1	1	1	0	0
1	*	1	1	1
2	2	3	2	*
1	*	2	*	3
1	1	2	2	*

On utilise une liste de listes pour représenter la grille et on choisit de coder une bombe par la valeur -1.

L'exemple ci-contre sera donc codé par la liste :

```
[[1, 1, 1, 0, 0],
 [1, -1, 1, 1, 1],
 [2, 2, 3, 2, -1],
 [1, -1, 2, -1, 3],
 [1, 1, 2, 2, -1]]
```

Compléter le code suivant afin de générer des grilles de démineur, on pourra vérifier que l'instruction **`génère_grille([(1, 1), (2, 4), (3, 1), (3, 3), (4, 4)])`** produit bien la liste donnée en exemple.

```
1  bombes = [(1, 1),
2            (2, 4),
3            (3, 1),
4            (3, 3),
5            (4, 4)]
6  grille_test = [[1, 1, 1, 0, 0],
7                 [1, -1, 1, 1, 1],
8                 [2, 2, 3, 2, -1],
9                 [1, -1, 2, -1, 3],
10                [1, 1, 2, 2, -1]]
11
```

```python
def voisinage(n, ligne, colonne):
    """ Renvoie la liste des coordonnées des voisins de la case
        (ligne, colonne) en gèrant les cases sur les bords. """
    voisins = []
    for l in range(max(0,ligne-1), min(n, ligne+2)):
        for c in range(max(0, colonne-1), min(n, colonne+2)):
            if (l, c) != (ligne, colonne):
                voisins.append((l,c))
    return voisins
```

```python
def incrémente_voisins(grille, ligne, colonne):
    """ Incrémente de 1 toutes les cases voisines d'une bombe. """
    voisins = ...
    for l, c in voisins:
        if grille[l][c] != ...: # si ce n'est pas une bombe
            ...             # on ajoute 1 à sa valeur

def génère_grille(bombes):
    """ Renvoie une grille de démineur de taille nxn où n est
        le nombre de bombes, en plaçant les bombes à l'aide de
        la liste bombes de coordonnées (tuples) passée en
        paramètre. """
    n = len(bombes)
    # Initialisation d'une grille nxn remplie de 0
    grille = [[0 for colonne in range(n)] for ligne in range(n)]

    # Place les bombes et calcule les valeurs des autres cases
    for ligne, colonne in bombes:
        grille[ligne][colonne] = ...  # place la bombe
        ...  # incrémente ses voisins

    return grille
```

Exercice 4

Dans cet exercice, les nombres sont des entiers ou des flottants.
Écrire une fonction **moyenne** renvoyant la moyenne pondérée d'une liste non vide, passée en paramètre, de tuples à deux éléments de la forme **(valeur, coefficient)** où **valeur** et **coefficient** sont des nombres positifs ou nuls.
Si la somme des coefficients est nulle, la fonction renvoie **None**, si la somme des coefficients est non nulle, la fonction renvoie, sous forme de flottant, la moyenne des valeurs affectées de leur coefficient.

Exemples :

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

Dans le premier exemple la moyenne est calculée par la formule :

$$\frac{8 \times 2 + 12 \times 0 + 13{,}5 \times 1 + 5 \times 0{,}5}{2 + 0 + 1 + 0{,}5}$$

Exercice 5

Écrire en python deux fonctions :

 **`lancer`** de paramètre **`n`**, un entier positif, qui renvoie un tableau de type **`list`** de **`n`** entiers obtenus aléatoirement entre 1 et 6 (1 et 6 inclus) ;

 **`paire_6`** de paramètre **`tab`**, un tableau de type **`list`** de **`n`** entiers entre 1 et 6 obtenus aléatoirement, 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 : Renvoie un entier aléatoire N tel que $a \leq N \leq b$.

Exemples :

```
>>> lancer1 = lancer(5)
[5, 6, 6, 2, 2]
>>> paire_6(lancer1)
True
>>> lancer2 = lancer(5)
[6, 5, 1, 6, 6]
>>> paire_6(lancer2)
True
>>> lancer3 = lancer(3)
[2, 2, 6]
>>> paire_6(lancer3)
False
>>> lancer4 = lancer(0)
[]
>>> paire_6(lancer4)
False
```

Exercice 6

Programmer la fonction **`recherche`**, prenant en paramètre un tableau non vide **`tab`** (de 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 la longueur du tableau.

Exemples :

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

Exercice 7

Programmer la fonction **fusion** prenant en paramètres deux tableaux non vides **tab1** et **tab2** (de 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]
```

Exercice 8

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 correspondantes 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 positifs ou nuls ;

renvoie un tuple dont :

- la première valeur est une clé du dictionnaire associée à la valeur maximale ;
- la seconde valeur est la 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': 220, 'Tim': 50})
('Alan', 222)
```

Exercice 9

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.

Exemples :

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

Exercice 10

On modélise la représentation binaire d'un entier non signé par un tableau d'entiers dont les éléments sont 0 ou 1. Par exemple, le tableau `[1, 0, 1, 0, 0, 1, 1]` représente l'écriture binaire de l'entier dont l'écriture décimale est

```
2**6 + 2**4 + 2**1 + 2**0 = 83.
```

À l'aide d'un parcours séquentiel, écrire la fonction **convertir** répondant aux spécifications suivantes :

```
def convertir(tab):
    """
    tab est un tableau d'entiers, dont les éléments sont 0 ou 1,
    et représentant un entier écrit en binaire.
    Renvoie l'écriture décimale de l'entier positif dont la
    représentation binaire est donnée par le tableau tab
    """
```

Exemple :

```
>>> convertir([1, 0, 1, 0, 0, 1, 1])
83
>>> convertir([1, 0, 0, 0, 0, 0, 1, 0])
130
```

Exercice 11

La fonction **`tri_insertion`** suivante prend en argument une liste **`tab`** et trie cette liste en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

On rappelle le principe du tri par insertion : on considère les éléments à trier un par un, le premier élément constituant, à lui tout seul, une liste triée de longueur 1. On range ensuite le second élément pour constituer une liste triée de longueur 2, puis on range le troisième élément pour avoir une liste triée de longueur 3 et ainsi de suite… A chaque étape, le premier élément de la sous-liste non triée est placé dans la sous-liste des éléments déjà triés de sorte que cette sous-liste demeure triée.
Le principe du tri par insertion est donc d'insérer à la nième itération, le nième élément à la bonne place.

```
liste = [9, 5, 8, 4, 0, 2, 7, 1, 10, 3, 6]

def tri_insertion(tab):
    n = len(tab)
    for i in range(1, n):
        valeur_insertion = tab[...]
        # la variable j sert à déterminer où placer la valeur à ranger
        j = ...
        # tant qu'on a pas trouvé la place de l'élément à insérer
        # on décale les valeurs du tableau vers la droite
        while j > ... and valeur_insertion < tab[...]:
            tab[j] = tab[j-1]
            j = ...
        tab[j] = ...
```

Exercice 12

On dispose d'un ensemble d'objets dont on connaît, pour chacun, la masse. On souhaite ranger l'ensemble de ces objets dans des boites identiques de telle manière que la somme des masses des objets contenus dans une boîte ne dépasse pas la capacité c de la boîte. On souhaite utiliser le moins de boîtes possibles pour ranger cet ensemble d'objets.
Pour résoudre ce problème, on utilisera un algorithme glouton consistant à placer chacun des objets dans la première boîte où cela est possible.
Par exemple, pour ranger dans des boîtes de capacité `c = 5` un ensemble de trois objets dont les masses sont représentées en Python par la liste `[1, 5, 2]`, on procède de la façon suivante :

Le premier objet, de masse 1, va dans une première boite.

Le deuxième objet, de masse 5, ne peut pas aller dans la même boite que le premier objet car cela dépasserait la capacité de la boite. On place donc cet objet dans une deuxième boîte.

Le troisième objet, de masse 2, va dans la première boîte.

On a donc utilisé deux boîtes de capacité `c = 5` pour ranger les 3 objets.

Compléter la fonction Python **`empaqueter(liste_masses, c)`** suivante pour qu'elle renvoie le nombre de boîtes de capacité **`c`** nécessaires pour empaqueter un ensemble d'objets dont les masses sont contenues dans la liste **`liste_masses`**.

```
def empaqueter(liste_masses, c):
    n = len(liste_masses)
    nb_boites = 0
    boites = [0]*n
    for masse in ... :
        i=0
        while i <= nb_boites and boites[i] + ... > c:
                i = i + 1
        if i == nb_boites + 1:
                ...
        boites[i] = ...
    return ...
```

Tester ensuite votre fonction :

```
>>> empaqueter([7, 6, 3, 4, 8, 5, 9, 2], 11)
5
```

Exercice 13

La fonction **`rendu_monnaie`** prend en paramètres deux nombres entiers positifs **`somme_due`** et **`somme_versee`**. Elle procède au rendu de la monnaie de la différence **`somme_versee - somme_due`** pour des achats effectués avec le système monétaire de la zone Euro. On utilise pour cela un algorithme glouton qui commence par rendre le maximum de pièces ou billets de plus grandes valeurs et ainsi de suite. Par la suite, on assimilera les billets à des pièces.

La fonction **`rendu_monnaie`** renvoie un tableau de type **`list`** contenant les pièces qui composent le rendu.

Toutes les sommes sont exprimées en euros. Les valeurs possibles pour les pièces sont donc contenues dans le tableau **`pieces = [1, 2, 5, 10, 20, 50, 100, 200].`**

Ainsi, l'instruction **`rendu_monnaie(452, 500)`** renvoie le tableau **`[20,20,5,2,1]`**.

En effet, la somme à rendre est de 48 euros soit 20 + 20 + 5 + 2 + 1.

Le code de la fonction **rendu_monnaie** est donné ci-après :

```
pieces = [1, 2, 5, 10, 20, 50, 100, 200]

def rendu_monnaie(somme_due, somme_versee):

    rendu = ...
    a_rendre = ...
    i = len(pieces) - 1
    while ... :
        if pieces[i] <= a_rendre :
            rendu.append(...)
            a_rendre = ...
        else :
            i = ...
    return rendu
```

Exercice 14

Écrire en langage Python une fonction **recherche** prenant comme paramètres une variable **a** de type numérique (**float** ou **int**) et un tableau **tab** (de type **list**) et qui renvoie le nombre d'occurrences de **a** dans **tab**.

Exemples d'utilisations de la fonction recherche :

```
>>> recherche(5, [])
0
>>> recherche(5, [-2, 3, 4, 8])
0
>>> recherche(5, [-2, 3, 1, 5, 3, 7, 4])
1
>>> recherche(5, [-2, 5, 3, 5, 4, 5])
3
```

Exercice 15

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 listes : l'une 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 **mini** qui prend en paramètres un tableau **releve** des relevés et un tableau **date** des dates 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 :

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

Exercice 16

Un mot palindrome peut se lire de la même façon de gauche à droite ou de droite à gauche : bob, radar, et non sont des mots palindromes.
De même certains nombres sont eux aussi des palindromes : 33, 121, 345543.
L'objectif de cet exercice est d'obtenir un programme Python permettant de tester si un nombre est un nombre palindrome.
Pour remplir cette tâche, on vous demande de compléter le code des trois fonctions ci- dessous sachant que la fonction **est_nbre_palindrome** s'appuiera sur la fonction **est_palindrome** qui elle-même s'appuiera sur la fonction **inverse_chaine**.
La fonction **inverse_chaine** inverse l'ordre des caractères d'une chaîne de caractères chaine et renvoie la chaîne inversée.
La fonction **est_palindrome** teste si une chaine de caractères chaine est un palindrome. Elle renvoie **True** si c'est le cas et **False** sinon. Cette fonction s'appuie sur la fonction précédente.
La fonction **est_nbre_palindrome** teste si un nombre nbre est un palindrome. Elle renvoie **True** si c'est le cas et **False** sinon. Cette fonction s'appuie sur la fonction précédente.

Compléter le code des trois fonctions ci-dessous.

```
1  def inverse_chaine(chaine):
2      result = ...
3      for caractere in chaine:
4          result = ...
5      return result
6
7  def est_palindrome(chaine):
8      inverse = inverse_chaine(chaine)
9      return ...
10
11 def est_nbre_palindrome(nbre):
12     chaine = ...
13     return est_palindrome(chaine)
```

Exemples :

```
>>> inverse_chaine('bac')
'cab'
>>> est_palindrome('NSI')
False
>>> est_palindrome('ISN-NSI')
True
```

```
>>> est_nbre_palindrome(214312)
False
>>> est_nbre_palindrome(213312)
True
```

Exercice 17

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

 la première liste contient les indices des valeurs de la liste **tab** strictement inférieures à **elt** ;

 la deuxième liste contient les indices des valeurs de la liste **tab** égales à **elt** ;

 la troisième liste contient les indices des valeurs de la liste **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 18

Un professeur de NSI décide de gérer les résultats de sa classe sous la forme d'un dictionnaire :

 les clefs sont les noms des élèves ;

 les valeurs sont des dictionnaires dont les clefs sont les types d'épreuves sous forme de chaîne de caractères et les valeurs sont les notes obtenues associées à leurs coefficients dans une liste.

Avec :

```
resultats = {'Dupont': {
                            'DS1': [15.5, 4],
                            'DM1': [14.5, 1],
                            'DS2': [13, 4],
                            'PROJET1': [16, 3],
                            'DS3': [14, 4]
                        },
             'Durand': {
                            'DS1': [6, 4],
                            'DM1': [14.5, 1],
                            'DS2': [8, 4],
                            'PROJET1': [9, 3],
                            'IE1': [7, 2],
                            'DS3': [8, 4],
                            'DS4':[15, 4]
                        }
            }
```

L'élève dont le nom est Durand a ainsi obtenu au DS2 la note de 8 avec un coefficient 4.
Le professeur crée une fonction **moyenne** qui prend en paramètre le **nom** d'un de ses élèves et renvoie sa moyenne arrondie au dixième.

Compléter le code du professeur ci-dessous :

```
1  def moyenne(nom, dico_result):
2      if nom in ...:
3          notes = dico_result[nom]
4          total_points = ...
5          total_coefficients = ...
6          for ...  in notes.values():
7              note, coefficient = valeurs
8              total_points = total_points + ... * coefficient
9              total_coefficients = ... + coefficient
10         return round( ... / total_coefficients, 1 )
11     else:
12         return -1
```

Exercice 19

Écrire une fonction **max_et_indice** qui prend en paramètre une liste non vide **tab** de nombres entiers et qui renvoie la valeur du plus grand élément de cette liste ainsi que l'indice de sa première apparition dans cette liste.
L'utilisation de la fonction native max n'est pas autorisée.
Ne pas oublier d'ajouter au corps de la fonction une documentation et une ou plusieurs assertions pour vérifier les pré-conditions.

Exemples :

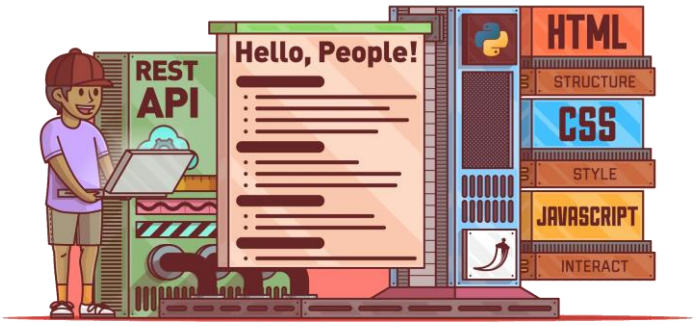
```
>>> max_et_indice([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
(9, 3)
>>> max_et_indice([-2])
(-2, 0)
>>> max_et_indice([-1, -1, 3, 3, 3])
(3, 2)
>>> max_et_indice([1, 1, 1, 1])
(1, 0)
```

Exercice 20

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, -1 sinon.

Exemples :

```
>>> recherche([2, 3, 4, 5, 6], 5)
3
>>> recherche([2, 3, 4, 6, 7], 5)
-1
```



Exercice 21

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 22

On considère une piste carrée qui contient 4 cases par côté. Les cases sont numérotées de 0 inclus à 12 exclu comme ci-dessous :

0	1	2	3
11			4
10			5
9	8	7	6

L'objectif de l'exercice est d'implémenter le jeu suivant :
Au départ, le joueur place son pion sur la case 0. A chaque coup, il lance un dé équilibré à six faces et avance son pion d'autant de cases que le nombre indiqué par le dé (entre 1 et 6 inclus) dans le sens des aiguilles d'une montre.
Par exemple, s'il obtient 2 au premier lancer, il pose son pion sur la case 2 puis s'il obtient 6 au deuxième lancer, il le pose sur la case 8, puis s'il obtient à nouveau 6, il pose le pion sur la case 2.
Le jeu se termine lorsque le joueur a posé son pion sur toutes les cases de la piste.

Compléter la fonction **nbre_coups** ci-après de sorte qu'elle renvoie le nombre de lancers aléatoires nécessaires pour terminer le jeu.

Proposer ensuite quelques tests pour en vérifier le fonctionnement.

```python
from random import randint

def nbre_coups():
    n = ...
    cases_vues = [0]
    case_en_cours = 0
    nbre_cases = 12
    while ... < ...:
        x = randint(1, 6)
        case_en_cours = (case_en_cours + ...) % ...
        if ...:
            cases_vues.append(case_en_cours)
        n = ...
    return n
```

Exercice 23

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 donc 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 de cette technique.

Exemples :

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

Exercice 24

On considère des tables (des tableaux de dictionnaires) 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 :

```python
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 **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 **table_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 25

Écrire une fonction **enumere** qui prend en paramètre une liste **L** et renvoie un dictionnaire **d** dont les clés sont les éléments de **L** avec pour valeur associée la liste des indices de l'élément dans la liste **L**.

Exemple :

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

Exercice 26

Compléter sous Python la fonction suivante en respectant la spécification. On ne recopiera pas les commentaires.

Exemples :

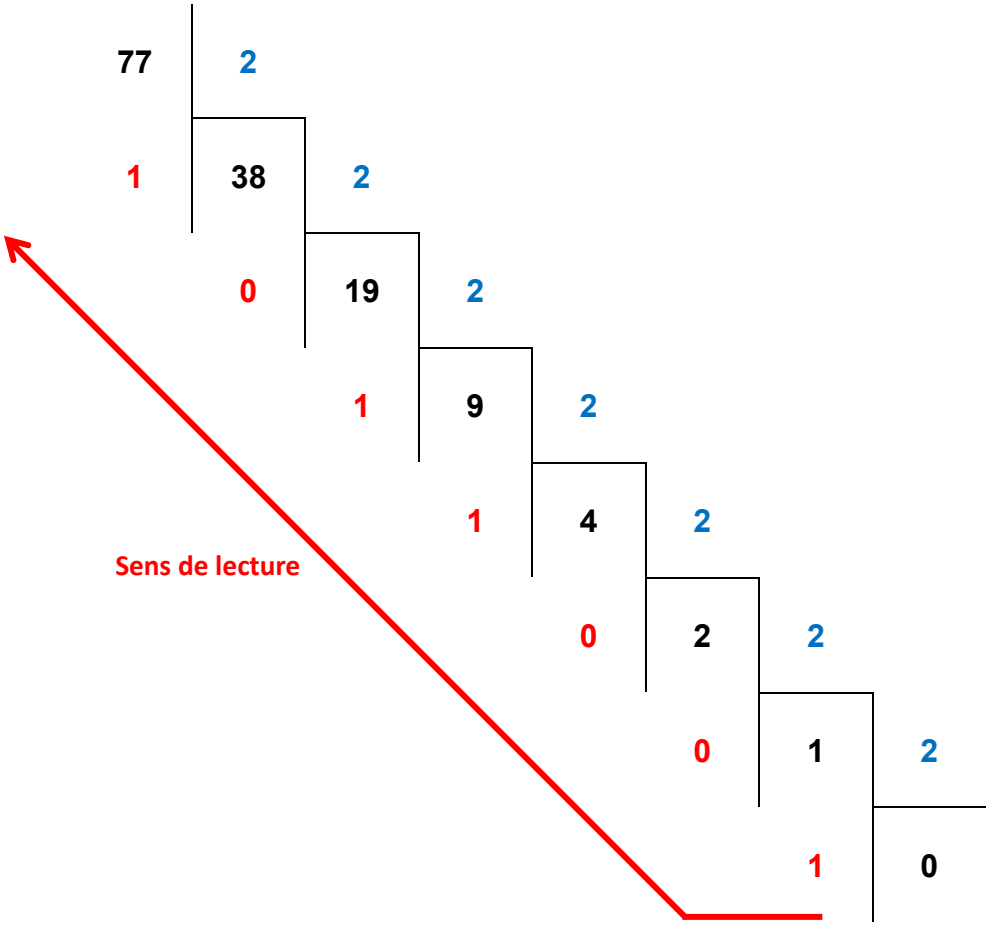
```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 27)
False
```

```python
def dichotomie(tab, x):
    """
    tab : tableau d'entiers trie dans l'ordre croissant
    x   : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """

    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
        if x > tab[m]:
            debut = m + 1
        else:
            fin = ...
    return ...
```

Exercice 27

Pour rappel, la conversion d'un nombre entier positif en binaire peut s'effectuer à l'aide des divisions successives comme illustré ici :



Voici une fonction python basée sur la méthode des divisions successives permettant de convertir un nombre entier positif en binaire :

```
def binaire(a):
    bin_a = str(...)
    a = a // 2
    while a ... :
        bin_a = ...(a%2) + ...
        a = ...
    return bin_a
```

Compléter la fonction **binaire**.

Exemples :

```
>>> binaire(0)
'0'
>>> binaire(77)
'1001101'
```

Exercice 28

On considère l'algorithme de tri de tableau suivant :
à chaque étape, on parcourt le sous-tableau des éléments non rangés et on place le plus petit élément en première position de ce sous-tableau.

Exemple avec le tableau :

```
t = [41, 55, 21, 18, 12, 6, 25]
```

Etape 1 : on parcourt tous les éléments du tableau, on permute le plus petit élément avec le premier. Le tableau devient

```
t = [6, 55, 21, 18, 12, 41, 25]
```

Etape 2 : on parcourt tous les éléments sauf le premier, on permute le plus petit élément trouvé avec le second. Le tableau devient :

```
t = [6, 12, 21, 18, 55, 41, 25]
```

Et ainsi de suite.

La code de la fonction **tri_selection** qui implémente cet algorithme est donné ci-dessous.

```
def tri_selection(tab):
    N = len(tab)
    for k in range(...):
        imin = ...
        for i in range(... , N):
            if tab[i] < ... :
                imin = i
        ... , tab[imin] = tab[imin] , ...
```

Compléter le code de cette fonction de façon à obtenir :

```
>>> liste = [41, 55, 21, 18, 12, 6, 25]
>>> tri_selection(liste)
>>> liste
[6, 12, 18, 21, 25, 41, 55]
```

On rappelle que l'instruction

```
a, b = b, a
```

échange les contenus de `a` et de `b`.

Exercice 29

Écrire une fonction ecriture_binaire_entier_positif qui prend en paramètre un entier positif n et renvoie une liste d'entiers correspondant à l'écriture binaire de n.
Ne pas oublier d'ajouter au corps de la fonction une documentation et une ou plusieurs assertions pour vérifier les pré-conditions.

Exemples :

```
>>> ecriture_binaire_entier_positif(0)
[0]
>>> ecriture_binaire_entier_positif(2)
[1, 0]
>>> ecriture_binaire_entier_positif(105)
[1, 1, 0, 1, 0, 0, 1]
```

Aide

l'opérateur `//` donne le quotient de la division euclidienne : $5//2$ donne 2 ;

l'opérateur `%` donne le reste de la division euclidienne : $5\%2$ donne 1 ;

`append` est une méthode qui ajoute un élément à une liste existante :
Soit `T = [5,2,4]`, alors `T.append(10)` ajoute `10` à la liste `T`. Ainsi, `T` devient `[5,2,4,10]`.

`reverse` est une méthode qui renverse les éléments d'une liste. Soit `T=[5,2,4,10]`.
Après `T.reverse()`, la liste devient `[10,4,2,5]`.

On remarquera qu'on récupère la représentation binaire d'un entier n en partant de la gauche en appliquant successivement les instructions :

$$b = n\%2$$
$$n = n//2$$

répétées autant que nécessaire.

Exercice 30

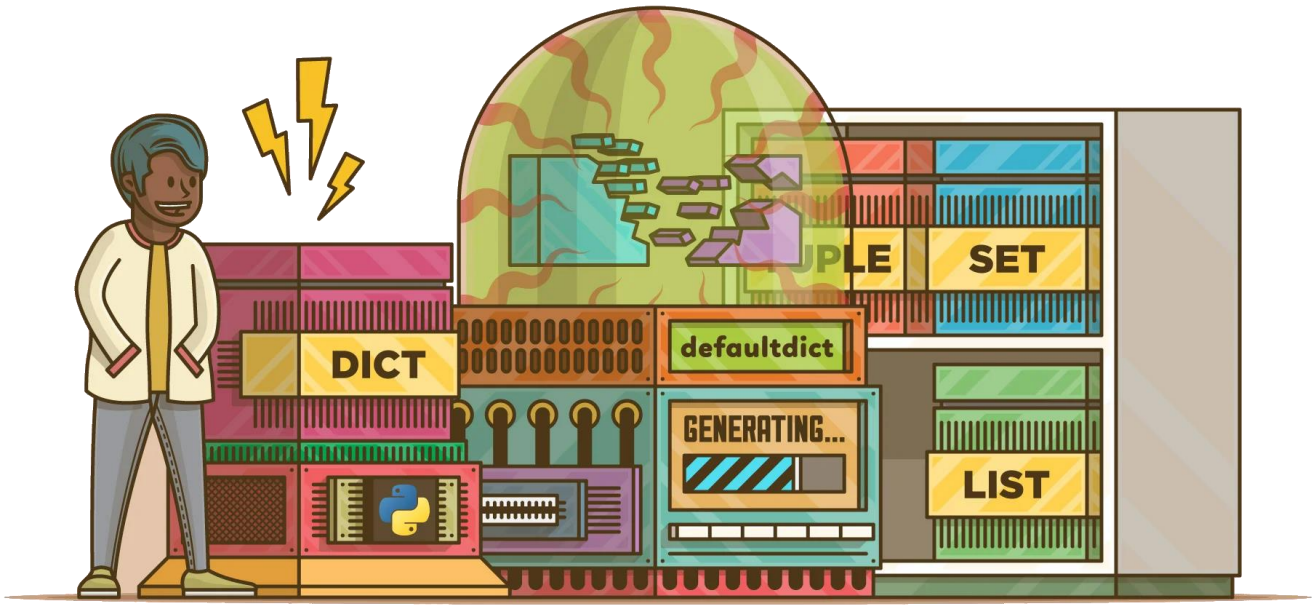
On veut trier par ordre croissant les notes d'une évaluation qui sont des nombres entiers compris entre 0 et 10 (inclus).
Ces notes sont contenues dans une liste **notes_eval.**

Écrire une fonction **rangement_valeurs** prenant en paramètre la liste **notes_eval** et renvoyant une liste de longueur 11 telle que la valeur de cette liste à chaque rang est égale au nombre de notes valant ce rang. Ainsi le terme de rang 0 indique le nombre de note 0, le terme de rang 1 le nombre de note 1, etc.

Écrire ensuite une fonction **notes_triees** prenant en paramètre la liste des effectifs des notes et renvoyant une liste contenant la liste, triée dans l'ordre croissant, des notes des élèves.

Exemple :

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



Classification des fleurs d'Iris avec KNN

Contexte : Vous avez un ensemble de données sur les fleurs d'Iris. Cet ensemble de données contient quatre caractéristiques (longueur des sépales, largeur des sépales, longueur des pétales, largeur des pétales) de 150 échantillons de fleurs d'Iris de trois espèces différentes (Iris setosa, Iris virginica et Iris versicolor).

Objectif : Votre tâche est de construire un classificateur KNN pour prédire l'espèce d'une fleur d'Iris en fonction de ces quatre caractéristiques.

Instructions :

1. **Préparation des données :**
   - Divisez votre ensemble de données en un ensemble d'entraînement et un ensemble de test (par exemple, 70% pour l'entraînement et 30% pour le test).
2. **Entraînement du modèle :**
   - Utilisez l'ensemble d'entraînement pour entraîner votre classificateur KNN. Vous pouvez commencer avec k=3.
3. **Test du modèle :**
   - Utilisez votre modèle pour prédire les espèces des fleurs dans votre ensemble de test.
4. **Évaluation du modèle :**
   - Évaluez les performances de votre modèle en calculant la précision de vos prédictions.
5. **Optimisation du modèle :**
   - Essayez différentes valeurs de k (par exemple, 1, 2, 3, 5, 10) et voyez comment cela affecte la précision de votre modèle.

Questions :

1. Quelle est la précision de votre modèle pour différentes valeurs de k ?
2. Quelle valeur de k donne la meilleure précision ?
3. Comment l'ajout ou la suppression de caractéristiques affecte-t-il la précision du modèle ?

Simulation d'un réseau simple

Contexte : Vous êtes un administrateur réseau pour une petite entreprise. L'entreprise dispose d'un réseau simple avec un serveur, cinq ordinateurs et une imprimante, tous connectés à un switch.

Objectif : Votre tâche est de simuler le fonctionnement de ce réseau en utilisant un logiciel de simulation de réseau comme Filius.

Instructions :

1. **Création du réseau :**
   - Créez un nouveau projet dans votre logiciel de simulation de réseau.
   - Ajoutez un serveur, cinq ordinateurs et une imprimante à votre espace de travail.
   - Connectez tous ces éléments à un switch.
2. **Configuration du réseau :**
   - Configurez le serveur pour fournir des services DHCP et DNS.
   - Configurez les ordinateurs pour obtenir automatiquement une adresse IP du serveur DHCP.
   - Configurez l'imprimante pour avoir une adresse IP statique.
3. **Test du réseau :**
   - Vérifiez que tous les ordinateurs peuvent obtenir une adresse IP du serveur DHCP.
   - Vérifiez que tous les ordinateurs peuvent accéder au serveur DNS.
   - Vérifiez que tous les ordinateurs peuvent imprimer sur l'imprimante.

Questions :

1. Quelle est l'adresse IP de chaque ordinateur ?
2. Quelle est l'adresse IP de l'imprimante ?
3. Quel est le rôle du serveur DHCP dans ce réseau ?
4. Quel est le rôle du serveur DNS dans ce réseau ?
5. Que se passe-t-il si le serveur tombe en panne ?