Épreuve pratique 💻

- Rappel des conditions de passation sur cette page
- Pdf de l'intégralité des exercices

Exercice 01.1 □

Exercice 01.1

Énoncé

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 la longueur du tableau.

Exemples

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

Correction

{{ correction(True, "

```
def recherche(tab, n):
    indice_solution = len(tab)
    for i in range(len(tab)):
        if tab[i] == n:
            indice_solution = i
    return indice_solution
```

") }}

Exercice 01.2 □

Exercice 01.2

Énoncé

On souhaite programmer une fonction donnant la distance la plus courte entre un point de départ et une liste de points. Les points sont tous à coordonnées entières. Les points sont donnés sous la forme d'un tuple de deux entiers. La liste des points à traiter est donc un tableau de tuples.

On rappelle que la distance entre deux points du plan de coordonnées $(x;y)$ et $(x';y')$ est donnée par la formule :

$$d = \sqrt{(x-x')^2+(y-y')^2}$$

On importe pour cela la fonction racine carrée (`sqrt`) du module `math` de Python.

On dispose d'une fonction `distance` et d'une fonction `plus_courte_distance` :

```
from math import sqrt     # import de la fonction racine carrée

def distance(point1, point2):
    """ Calcule et renvoie la distance entre deux points. """
    return sqrt((...)**2 + (...)**2)

assert distance((1, 0), (5, 3)) == 5.0, "erreur de calcul"

def plus_courte_distance(tab, depart):
    """ Renvoie le point du tableau tab se trouvant à la plus courte distance du point depart
    point = tab[0]
    min_dist = ...
    for i in range (1, ...):
        if distance(tab[i], depart)...:
            point = ...
            min_dist = ...
    return point

assert plus_courte_distance([(7, 9), (2, 5), (5, 2)], (0, 0)) == (2, 5), "erreur"
```

Recopier sous Python (sans les commentaires) ces deux fonctions puis compléter leur code et ajouter une ou des déclarations (`assert`) à la fonction `distance` permettant de vérifier la ou les préconditions.

Correction

```
1  from math import sqrt
2
3  def distance(point1, point2):
4      """ Calcule et renvoie la distance entre deux points. """
5      return sqrt((point1[0] - point2[0])**2 + ((point1[1] - point2[1]))**2)
6
7  assert distance((1, 0), (5, 3)) == 5.0, "erreur de calcul"
8
9
10 def plus_courte_distance(tab, depart):
11     """ Renvoie le point du tableau tab se trouvant à la plus courte distance du point dep
12     point = tab[0]
13     min_dist = distance(point, depart)
14     for i in range (1, len(tab)):
15         if distance(tab[i], depart) < min_dist:
16             point = tab[i]
17             min_dist = distance(tab[i], depart)
18     return point
19
20 assert plus_courte_distance([(7, 9), (2, 5), (5, 2)], (0, 0)) == (2, 5), "erreur"
```

Exercice 02.1 □

Exercice 02.1

Énoncé

Programmer la fonction `moyenne` prenant en paramètre un tableau d'entiers `tab` (type `list`) qui renvoie la moyenne de ses éléments si le tableau est non vide et affiche 'erreur' si le tableau est vide.

Exemples :

```
>>> moyenne([5,3,8])
5.333333333333333
>>> moyenne([1,2,3,4,5,6,7,8,9,10])
5.5
>>> moyenne([])
'erreur'
```

Correction

{{ correction(True, " L'énoncé n'est pas très clair quand il dit «d'afficher 'erreur'» (ce qui suppose un `print` et non un `return`). Nous choisissons donc dans ce cas de renvoyer `None`.

```
def moyenne(tab):
    if tab == []:
        print('erreur')
        return None
    else:
        somme = 0
        for elt in tab:
            somme += elt
        return somme / len(tab)
```

") }}

Exercice 02.2 □

Exercice 02.2

Énoncé

On considère un tableau d'entiers `tab` (type `list` dont les éléments sont des `0` ou des `1`). On se propose de trier ce tableau selon l'algorithme suivant : à chaque étape du tri,le tableau est constitué de trois zones consécutives, la première ne contenant que des `0`, la seconde n'étant pas triée et la dernière ne contenant que des `1`.

Zone de 0	Zone non triée	Zone de 1

Tant que la zone non triée n'est pas réduite à un seul élément, on regarde son premier élément :

- si cet élément vaut 0, on considère qu'il appartient désormais à la zone ne contenant que des 0 ;
- si cet élément vaut 1, il est échangé avec le dernier élément de la zone non triée et on considère alors qu'il appartient à la zone ne contenant que des 1.

Dans tous les cas, la longueur de la zone non triée diminue de 1.

Recopier sous Python en la complétant la fonction `tri` suivante :

```
def tri(tab):
    #i est le premier indice de la zone non triee, j le dernier indice.
    #Au debut, la zone non triee est le tableau entier.
    i = ...
    j = ...
    while i != j :
        if tab[i]== 0:
            i = ...
        else :
```

```
            valeur = tab[j]
            tab[j] = ...
            ...
            j = ...
    ...
```

Exemple :

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

Correction

"

```
def tri(tab):
    #i est le premier indice de la zone non triee, j le dernier indice.
    #Au debut, la zone non triee est le tableau entier.
    i = 0
    j = len(tab) - 1
    while i != j :
        if tab[i]== 0:
            i = i + 1
        else :
            valeur = tab[j]
            tab[j] = tab[i]
            tab[i] = valeur
            j = j - 1
    return tab
```

Exercice 03.1 □

Exercice 03.1

Énoncé

Programmer la fonction `multiplication`, prenant en paramètres deux nombres entiers `n1` et `n2`, et qui renvoie le produit de ces deux nombres. Les seules opérations autorisées sont l'addition et la soustraction.

Correction

Énoncé peu clair, on ne sait pas si `n1` et `n2` sont entiers naturels ou relatifs. Nous décidons qu'ils sont relatifs et donc qu'ils peuvent être négatifs, auquel cas on utilise le fait que $5times(-6) = -(5times6)$.

```
def multiplication(n1, n2):
    if n1 < 0:
        return -multiplication(-n1, n2)
    if n2 < 0:
        return -multiplication(n1, -n2)
    resultat = 0
    for _ in range(n2):
        resultat += n1
    return resultat
```

Exercice 03.2 □

Exercice 03.2

Énoncé

Recopier et compléter sous Python la fonction suivante en respectant la spécification. On ne recopiera pas les commentaires.

```
def dichotomie(tab, x):
    """
    tab : tableau d'entiers trié 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 ...
```

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
```

Correction

```
def dichotomie(tab, x):
    """
    tab : tableau d'entiers trié 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 = (debut + fin) // 2
        if x == tab[m]:
            return True
        if x > tab[m]:
            debut = m + 1
        else:
            fin = m - 1
    return False
```

Exercice 04.1 □

Exercice 04.1

Énoncé

Écrire une fonction qui prend en paramètre un tableau d'entiers non vide et qui renvoie la moyenne de ces entiers. La fonction est spécifiée ci-après et doit passer les assertions fournies.

entiers. La fonction est spécifiée ci-après et doit passer les assertions fournies.

```
def moyenne (tab):
    '''
    moyenne(list) -> float
    Entrée : un tableau non vide d'entiers
    Sortie : nombre de type float
    Correspondant à la moyenne des valeurs présentes dans le
    tableau
    '''

assert moyenne([1]) == 1
assert moyenne([1,2,3,4,5,6,7] == 4
assert moyenne([1,2]) == 1.5
```

Correction

```
1  def moyenne(tab):
2      '''
3      moyenne(list) -> float
4      Entrée : un tableau non vide d'entiers
5      Sortie : nombre de type float
6      Correspondant à la moyenne des valeurs présentes dans le
7      tableau
8      '''
9      somme = 0
10     for elt in tab:
11         somme += elt
12     return somme / len(tab)
```

Exercice 04.2 □

Exercice 04.2

Énoncé

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.

L'algorithme traite le cas du tableau vide.

L'algorithme est écrit pour que la recherche dichotomique ne se fasse que dans le cas où la valeur est comprise entre les valeurs extrêmes du tableau.

On distingue les trois cas qui renvoient `False` en renvoyant `False,1` , `False,2` et `False,3`.

Compléter l'algorithme de dichotomie donné ci-après.

```
1  def dichotomie(tab, x):
2      """
3      tab : tableau trié dans l'ordre croissant
4      x : nombre entier
5      La fonction renvoie True si tab contient x et False sinon
6      """
7      # cas du tableau vide
8      if ...:
9          return False,1
10     # cas où x n'est pas compris entre les valeurs extrêmes
11     if (x < tab[0]) or ...:
12         return False,2
13     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 ...
```

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, 3)
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],1)
(False, 2)
>>> dichotomie([],28)
(False, 1)
```

Correction

```
def dichotomie(tab, x):
    """
    tab : tableau trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    # cas du tableau vide
    if tab = []:
        return False,1
    # cas où x n'est pas compris entre les valeurs extrêmes
    if (x < tab[0]) or (x > tab[-1]):
        return False,2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = (debut + fin) // 2
        if x == tab[m]:
            return True
        if x > tab[m]:
            debut = m + 1
        else:
            fin = m - 1
    return False
```

Exercice 05.1 □

Exercice 05.1

Énoncé

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(T):
    """
    T 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 T
    """
```

Exemple :

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

Correction

{{ correction(True, "

```
def convertir(T):
    puissance = 0
    total = 0
    for i in range(len(T)-1, -1, -1):
        total += T[i]*(2**puissance)
        puissance += 1
    return total
```

") }}

Exercice 05.2 □

Exercice 05.2

Énoncé

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

```
def tri_insertion(L):
    n = len(L)

    # cas du tableau vide
    if ...:
        return L
    for j in range(1,n):
        e = L[j]
        i = j

    # A l'étape j, le sous-tableau L[0,j-1] est trié
    # et on insère L[j] dans ce sous-tableau en déterminant
    # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].

        while i > 0 and L[i-1] > ...:
            i = ...

    # si i != j, on décale le sous tableau L[i,j-1] d'un cran
    # vers la droite et on place L[j] en position i

        if i != j:
```

```
            for k in range(j,i,...):
                L[k] = L[...]
            L[i] = ...
    return L
```

Exemples :

```
>>> tri_insertion([2,5,-1,7,0,28])
[-1, 0, 2, 5, 7, 28]
>>> tri_insertion([10,9,8,7,6,5,4,3,2,1,0])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Correction

{{ correction(True, "

```
def tri_insertion(L):
    n = len(L)

    # cas du tableau vide
    if L == []:
        return L

    for j in range(1,n):
        e = L[j]
        i = j

    # A l'étape j, le sous-tableau L[0,j-1] est trié
    # et on insère L[j] dans ce sous-tableau en déterminant
    # le plus petit i tel que 0 <= i <= j et L[i-1] > L[j].

        while i > 0 and L[i-1] > e:
            i = i - 1

        # si i != j, on décale le sous tableau L[i,j-1] d'un cran
    # vers la droite et on place L[j] en position i

        if i != j:
            for k in range(j,i,-1):
                L[k] = L[k-1]
            L[i] = e
    return L
```

") }}

Exercice 06.1 □

Exercice 06.1

Énoncé

On s'intéresse au problème du rendu de monnaie. On suppose qu'on dispose d'un nombre infini de billets de 5 euros, de pièces de 2 euros et de pièces de 1 euro. Le but est d'écrire une fonction nommée `rendu` dont le paramètre est un entier positif non nul `somme_a_rendre` et qui retourne une liste de trois entiers `n1`, `n2` et `n3` qui correspondent aux nombres de billets de 5 euros (`n1`) de pièces de 2 euros (`n2`) et de pièces de 1 euro (`n3`) à rendre afin que le total rendu soit égal à `somme_a_rendre`.

On utilisera un algorithme glouton : on commencera par rendre le nombre maximal de billets de 5 euros, puis celui des pièces de 2 euros et enfin celui des pièces de 1 euros.

Exemples :

```
>>> rendu(13)
[2,1,1]
>>> rendu(64)
[12,2,0]
>>> rendu(89)
[17,2,0]
```

Correction

{{ correction(True, "

```
def rendu(somme_a_rendre):
    pieces = [5, 2, 1]
    retour = [0, 0, 0]
    reste_a_rendre = somme_a_rendre
    for i in range(3):
        retour[i] = reste_a_rendre // pieces[i]
        reste_a_rendre = reste_a_rendre % pieces[i]
    return retour
```

") }}

Exercice 06.2 □

à noter une erreur dans la version officielle, sur la méthode `enfile()`

Exercice 06.2

Énoncé

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée. On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None
```

Compléter la classe `File suivante` où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

```
class File :
    def __init__(self) :
        self.dernier_file = None

    def enfile(self,element) :
        nouveau_maillon = Maillon(...)
        nouveau_maillon.suivant = self.dernier_file
        self.dernier_file = ...

    def est_vide(self) :
```

```
        return self.dernier_file == None

    def affiche(self) :
        maillon = self.dernier_file
        while maillon != ... :
            print(maillon.valeur)
            maillon = ...

    def defile(self) :
        if not self.est_vide() :
            if self.dernier_file.suivant == None :
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = ...
            while maillon.suivant.suivant != None :
                maillon = maillon.suivant
            resultat = ...
            maillon.suivant = None
            return resultat
        return None
```

On pourra tester le fonctionnement de la classe en utilisant les commandes suivantes dans la console Python :

```
>>> F = File()
>>> F.est_vide()
True
>>> F.enfile(2)
>>> F.affiche()
2
>>> F.est_vide()
False
>>> F.enfile(5)
>>> F.enfile(7)
>>> F.affiche()
7
5
2
>>> F.defile()
2
>>> F.defile()
5
>>> F.affiche()
7
```

Correction

{{ correction(True, "

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None

class File :
    def __init__(self) :
        self.dernier_file = None

    def enfile(self,element) :
        nouveau_maillon = Maillon(element)
        nouveau_maillon.suivant =  self.dernier_file
        self.dernier_file = nouveau_maillon
```

```
14
15      def est_vide(self) :
16          return self.dernier_file == None
17
18      def affiche(self) :
19          maillon = self.dernier_file
20          while maillon != None :
21              print(maillon.valeur)
22              maillon = maillon.suivant
23
24      def defile(self) :
25          if not self.est_vide() :
26              if self.dernier_file.suivant == None :
27                  resultat = self.dernier_file.valeur
28                  self.dernier_file = None
29                  return resultat
30              maillon = self.dernier_file
31              while maillon.suivant.suivant != None :
32                  maillon = maillon.suivant
33              resultat = maillon.suivant.valeur
34              maillon.suivant = None
35              return resultat
36          return None
```

") }}

Exercice 07.1 □

Exercice 07.1

Énoncé

On s'intéresse à la suite d'entiers définie par `U1 = 1`, `U2 = 1` et, pour tout entier naturel `n`, par `Un+2 = Un+1 + Un`.

Elle s'appelle la suite de Fibonnaci.

Écrire la fonction `fibonacci` qui prend un entier `n > 0` et qui renvoie l'élément d'indice `n` de cette suite.

On utilisera une programmation dynamique (pas de récursivité).

Exemple :

```
>>> fibonacci(1)
1
>>> fibonacci(2)
1
>>> fibonacci(25)
75025
>>> fibonacci(45)
1134903170
```

Correction

{{ correction(True, " On utilise un dictionnaire pour stocker au fur et à mesure les valeurs.

```
1  def fibonnaci(n):
2      d = {}
3      d[1] = 1
```

```
    d[2] = 1
    for k in range(3, n+1):
        d[k] = d[k-1] + d[k-2]
    return d[n]
```

") }}

Exercice 07.2 □

Exercice 07.2

Énoncé

Les variables `liste_eleves` et `liste_notes` ayant été préalablement définies et étant de même longueur, la fonction `meilleures_notes` renvoie la note maximale qui a été attribuée, le nombre d'élèves ayant obtenu cette note et la liste des noms de ces élèves.

Compléter le code Python de la fonction `meilleures_notes` ci-dessous.

```
liste_eleves = ['a','b','c','d','e','f','g','h','i','j']
liste_notes = [1, 40, 80, 60, 58, 80, 75, 80, 60, 24]

def meilleures_notes():
    note_maxi = 0
    nb_eleves_note_maxi = ...
    liste_maxi = ...

    for compteur in range(...):
        if liste_notes[compteur] == ...:
            nb_eleves_note_maxi = nb_eleves_note_maxi + 1
            liste_maxi.append(liste_eleves[...])
        if liste_notes[compteur] > note_maxi:
            note_maxi = liste_notes[compteur]
            nb_eleves_note_maxi = ...
            liste_maxi = [...]

    return (note_maxi,nb_eleves_note_maxi,liste_maxi)
```

Une fois complété, le code ci-dessus donne

```
>>> meilleures_notes()
(80, 3, ['c', 'f', 'h'])
```

Correction

{{ correction(True, "

```
liste_eleves = ['a','b','c','d','e','f','g','h','i','j']
liste_notes = [1, 40, 80, 60, 58, 80, 75, 80, 60, 24]

def meilleures_notes():
    note_maxi = 0
    nb_eleves_note_maxi = 0
    liste_maxi = []

    for compteur in range(len(liste_eleves)):

```

```
            if liste_notes[compteur] == note_maxi:
                nb_eleves_note_maxi = nb_eleves_note_maxi + 1
                liste_maxi.append(liste_eleves[compteur])
            if liste_notes[compteur] > note_maxi:
                note_maxi = liste_notes[compteur]
                nb_eleves_note_maxi = 1
                liste_maxi = [liste_eleves[compteur]]

        return (note_maxi,nb_eleves_note_maxi,liste_maxi)
```

") }}

Exercice 08.1 □

Exercice 08.1

Énoncé

Écrire une fonction `recherche` qui prend en paramètres `caractere`, un caractère, et `mot`, une chaîne de caractères, et qui renvoie le nombre d'occurrences de `caractere` dans `mot`, c'est-à-dire le nombre de fois où `caractere` apparaît dans `mot`.

Exemples :

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

Correction

{{ correction(True, "

```
def recherche(caractere, mot):
    somme = 0
    for lettre in mot:
        if lettre == caractere:
            somme += 1
    return somme
```

") }}

Exercice 08.2 □

Exercice 08.2

Énoncé

On s'intéresse à un algorithme récursif qui permet de rendre la monnaie à partir d'une liste donnée de valeurs de pièces et de billets - le système monétaire est donné sous forme d'une liste `pieces=[100, 50, 20, 10, 5, 2, 1]` - (on supposera qu'il n'y a pas de limitation quant à leur nombre), on cherche à donner la liste de pièces à rendre pour une somme donnée en argument. Compléter le code Python ci-dessous de la fonction `rendu_glouton` qui implémente cet algorithme et renvoie la liste des pièces à rendre.

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

def rendu_glouton(arendre, solution=[], i=0):
    if arendre == 0:
        return ...
    p = pieces[i]
    if p <= ... :
        solution.append(...)
        return rendu_glouton(arendre - p, solution,i)
    else :
        return rendu_glouton(arendre, solution, ...)
```

On devra obtenir :

```
>>>rendu_glouton(68,[],0)
[50, 10, 5, 2, 1]
>>>rendu_glouton(291,[],0)
[100, 100, 50, 20, 20, 1]
```

Correction

{{ correction(True, "

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

def rendu_glouton(arendre, solution=[], i=0):
    if arendre == 0:
        return solution
    p = pieces[i]
    if p <= arendre :
        solution.append(p)
        return rendu_glouton(arendre - p, solution,i)
    else :
        return rendu_glouton(arendre, solution, i+1)
```

") }}

Exercice 09.1 □

Exercice 09.1

Énoncé

Soit le couple (`note` , `coefficient`):

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

Les résultats aux évaluations d'un élève sont regroupés dans une liste composée de couples (`note` , `coefficient`).

Écrire une fonction moyenne qui renvoie la moyenne pondérée de cette liste donnée en paramètre.

Par exemple, l'expression `moyenne([(15,2),(9,1),(12,3)])` devra renvoyer le résultat du calcul suivant :

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

Correction

{{ correction(True, "

```
def moyenne(tab):
    somme_notes = 0
    somme_coeffs = 0
    for devoir in tab:
        note = devoir[0]
        coeff = devoir[1]
        somme_notes += note * coeff
        somme_coeffs += coeff
    return somme_notes / somme_coeffs
```

") }}

Exercice 09.2 □

Exercice 09.2

Énoncé

On cherche à déterminer les valeurs du triangle de Pascal. Dans ce tableau de forme triangulaire, chaque ligne commence et se termine par le nombre 1. Par ailleurs, la valeur qui occupe une case située à l'intérieur du tableau s'obtient en ajoutant les valeurs des deux cases situées juste au-dessus, comme l'indique la figure suivante :

Compléter la fonction `pascal` ci-après. Elle doit renvoyer une liste correspondant au triangle de Pascal de la ligne `1` à la ligne `n` où `n` est un nombre entier supérieur ou égal à `2` (le tableau sera contenu dans la variable `C`). La variable `Ck` doit, quant à elle, contenir, à l'étape numéro `k`, la `k`-ième ligne du tableau.

```
def pascal(n):
    C= [[1]]
    for k in range(1,...):
        Ck = [...]
        for i in range(1,k):
            Ck.append(C[...][i-1]+C[...][...] )
        Ck.append(...)
        C.append(Ck)
    return C
```

Pour `n = 4`, voici ce qu'on devra obtenir :

```
>>> pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

Pour `n = 5`, voici ce qu'on devra obtenir :

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

Correction

{{ correction(True, "

```
def pascal(n):
    C = [[1]]
    for k in range(1,n+1):
        Ck = [1]
        for i in range(1,k):
            Ck.append(C[k-1][i-1]+C[k-1][i] )
        Ck.append(1)
        C.append(Ck)
    return C
```

") }}

Exercice 10.1 □

Exercice 10.1

Énoncé

Écrire une fonction `maxi` qui prend en paramètre une liste `tab` de nombres entiers et renvoie un couple donnant le plus grand élément de cette liste, ainsi que l'indice de la première apparition de ce maximum dans la liste.

Exemple :

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

Correction

{{ correction(True, "

") }}

Exercice 10.2 ☑

Exercice 10.2

Énoncé

Cet exercice utilise des piles qui seront représentées en Python par des listes (type `list`).

On rappelle que l'expression `T1 = list(T)` fait une copie de `T` indépendante de `T`, que l'expression `x = T.pop()` enlève le sommet de la pile `T` et le place dans la variable `x` et, enfin, que l'expression `T.append(v)` place la valeur `v` au sommet de la pile `T`.

Compléter le code Python de la fonction positif ci-dessous qui prend une pile `T` de nombres entiers en paramètre et qui renvoie la pile des entiers positifs dans le même ordre, sans modifier la variable `T`.

```
def positif(T):
    T2 = ...(T)
    T3 = ...
    while T2 != []:
        x = ...
        if ... >= 0:
```

```
            T3.append(...)
    T2 = []
    while T3 != ...:
        x = T3.pop()
        ...
    print('T = ',T)
    return T2
```

Exemple :

```
>>> positif([-1,0,5,-3,4,-6,10,9,-8 ])
T = [-1, 0, 5, -3, 4, -6, 10, 9, -8]
[0, 5, 4, 10, 9]
```

Correction

{{ correction(True, "

```
def positif(T):
    T2 = list(T)
    T3 = []
    while T2 != []:
        x = T2.pop()
        if x >= 0:
            T3.append(x)
    T2 = [] # <- NB : cette ligne est inutile
    while T3 != []:
        x = T3.pop()
        T2.append(x)
    print('T = ',T)
    return T2
```

") }}

Exercice 11.1 □

Exercice 11.1

Énoncé

Écrire une fonction `conv_bin` qui prend en paramètre un entier positif `n` et renvoie un couple (`b,bit)` où :

- `b` est une liste d'entiers correspondant à la représentation binaire de `n` ;
- `bit` correspond aux nombre de bits qui constituent `b`.

Exemple :

```
>>> conv_bin(9)
([1,0,0,1],4)
```

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.

Correction

{{ correction(True, "

```
def conv_bin(n):
    b = []
    bits = 0
    while n != 0:
        b.append(n % 2)
        bits += 1
        n = n // 2
    b.reverse()
    return (b, bits)
```

") }}

Exercice 11.2 □

Exercice 11.2

Énoncé

La fonction `tri_bulles` prend en paramètre une liste `T` d'entiers non triés et renvoie la liste triée par ordre croissant. Compléter le code Python ci-dessous qui implémente la fonction `tri_bulles`.

```
def tri_bulles(T):
    n = len(T)
    for i in range(...,...,-1):
        for j in range(i):
            if T[j] > T[...]:
                ... = T[j]
                T[j] = T[...]
                T[j+1] = temp
    return T
```

Écrire une autre version de l'algorithme avec

```
for i in range(n-1):
```

en lieu et place de la troisième ligne du code précédent.

Correction

{{ correction(True, "

```
def tri_bulles(T):
    n = len(T)
    for i in range(n-1,0,-1):
        for j in range(i):
            if T[j] > T[j+1]:
                temp = T[j]
                T[j] = T[j+1]
                T[j+1] = temp
    return T

#version 2

def tri_bulles(T):
    n = len(T)
    for i in range(n-1):
        for j in range(n-1,i,-1):
            if T[j] < T[j-1]:
                temp = T[j]
                T[j] = T[j-1]
                T[j-1] = temp
    return T
```

") }}

Exercice 12.1 □

Ce sujet est le même que le 10.1... ¯_(ツ)_/¯

Exercice 12.1

Énoncé

Écrire une fonction `maxi` qui prend en paramètre une liste `tab` de nombres entiers et renvoie un couple donnant le plus grand élément de cette liste, ainsi que l'indice de la première apparition de ce maximum dans la liste.

Exemple :

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

Correction

{{ correction(True, "

") }}

Exercice 12.2 □

Exercice 12.2

Énoncé

La fonction `recherche` prend en paramètres deux chaines de caractères `gene` et `seq_adn` et renvoie `True` si on retrouve `gene` dans `seq_adn` et `False` sinon. Compléter le code Python ci-dessous pour qu'il implémente la fonction `recherche`.

```
def recherche(gene, seq_adn):
    n = len(seq_adn)
    g = len(gene)
    i = ...
    trouve = False
    while i < ... and trouve == ... :
        j = 0
        while j < g and gene[j] == seq_adn[i+j]:
            ...
        if j == g:
            trouve = True
        ...
    return trouve
```

Exemples :

```
>>> recherche("AATC", "GTACAAATCTTGCC")
True
>>> recherche("AGTC", "GTACAAATCTTGCC")
False
```

Correction

{{ correction(True, "

```
def recherche(gene, seq_adn):
    n = len(seq_adn)
    g = len(gene)
    i = 0
    trouve = False
    while i < n-g and trouve == False :
        j = 0
        while j < g and gene[j] == seq_adn[i+j]:
            j += 1
        if j == g:
            trouve = True
        i += 1
    return trouve
```

") }}

Exercice 13.1 □

Exercice 13.1

Énoncé

Écrire une fonction `tri_selection` qui prend en paramètre une liste `tab` de nombres entiers et qui renvoie le tableau trié par ordre croissant.

On utilisera l'algorithme suivant :

- on recherche le plus petit élément du tableau, et on l'échange avec l'élément d'indice 0 ;
- on recherche le second plus petit élément du tableau, et on l'échange avec l'élément d'indice 1 ;
- on continue de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exemple :

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

Correction

{{ correction(True, "

```
def tri_selection(tab):
    for i in range(len(tab)-1):
        indice_min = i
        for j in range(i+1, len(tab)):
            if tab[j] < tab[indice_min]:
                indice_min = j
        tab[i], tab[indice_min] = tab[indice_min], tab[i]
    return tab
```

") }}

Exercice 13.2 □

Exercice 13.2

Énoncé

Le jeu du « plus ou moins » consiste à deviner un nombre entier choisi entre 1 et 99. Un élève de NSI décide de le coder en langage Python de la manière suivante :

- le programme génère un nombre entier aléatoire compris entre 1 et 99 ;
- si la proposition de l'utilisateur est plus petite que le nombre cherché, l'utilisateur en est averti. Il peut alors en tester un autre ;
- si la proposition de l'utilisateur est plus grande que le nombre cherché, l'utilisateur en est averti. Il peut alors en tester un autre ;
- si l'utilisateur trouve le bon nombre en 10 essais ou moins, il gagne ;
- si l'utilisateur a fait plus de 10 essais sans trouver le bon nombre, il perd.

La fonction `randint` est utilisée. Si a et b sont des entiers, `randint(a,b)` renvoie un nombre entier compris entre `a` et `b`. Compléter le code ci-dessous et le tester :

```
from random import randint

def plus_ou_moins():
    nb_mystere = randint(1,...)
    nb_test = int(input("Proposez un nombre entre 1 et 99 : "))
    compteur = ...

    while nb_mystere != ... and compteur < ... :
        compteur = compteur + ...
        if nb_mystere ... nb_test:
            nb_test = int(input("Trop petit ! Testez encore : "))
        else:
            nb_test = int(input("Trop grand ! Testez encore : "))

    if nb_mystere == nb_test:
        print ("Bravo ! Le nombre était ",...)
        print("Nombre d'essais: ",...)
    else:
        print ("Perdu ! Le nombre était ",...)
```

Correction

{{ correction(True, "

```
from random import randint

def plus_ou_moins():
    nb_mystere = randint(1,100)
    nb_test = int(input('Proposez un nombre entre 1 et 99 : '))
    compteur = 0

    while nb_mystere != nb_test and compteur < 10 :
        compteur = compteur + 1
        if nb_mystere > nb_test:
            nb_test = int(input('Trop petit ! Testez encore : '))
        else:
            nb_test = int(input('Trop grand ! Testez encore : '))

    if nb_mystere == nb_test:
        print ('Bravo ! Le nombre était ', nb_mystere)
        print('Nombre d essais: ', compteur)
    else:
        print ('Perdu ! Le nombre était ', nb_mystere)
```

") }}

Exercice 14.1 □

Exercice 14.1

Énoncé

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre et `tab` un tableau de nombres, et qui renvoie le tableau des indices de `elt` dans `tab` si `elt` est dans `tab` et le tableau vide `[]` sinon.

Exemples :

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

Correction

{{ correction(True, "

```
def recherche(elt, tab):
    tab_indices = []
    for i in range(len(tab)):
        if tab[i] == elt:
            tab_indices.append(i)
    return tab_indices
```

") }}

Exercice 14.2 □

Exercice 14.2

Énoncé

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 et les valeurs sont les notes obtenues associées à leurs coefficients.

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 ces élèves et lui renvoie sa moyenne arrondie au dixième.

Compléter le code du professeur ci-dessous :

```
def moyenne(nom):
    if nom in ...:
        notes = resultats[nom]
        total_points = ...
        total_coefficients = ...
        for ... in notes.values():
            note , coefficient = valeurs
            total_points = total_points + ... * coefficient
            total_coefficients = ... + coefficient
        return round( ... / total_coefficients , 1 )
    else:
        return -1
```

Correction

{{ correction(True, "

```
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]}}

def moyenne(nom):
    if nom in resultats:
        notes = resultats[nom]
        total_points = 0
        total_coefficients = 0
        for valeurs in notes.values():
            note , coefficient = valeurs
            total_points = total_points + note * coefficient
            total_coefficients = total_coefficients + coefficient
        return round( total_points / total_coefficients , 1 )
    else:
        return -1
```

") }}

Exercice 15.1 □

Exercice 15.1

Énoncé

Écrire une fonction `rechercheMinMax` qui prend en paramètre un tableau de nombres non triés `tab`, 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.

Exemples :

```
>>> tableau = [0, 1, 4, 2, -2, 9, 3, 1, 7, 1]
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': -2, 'max': 9}
>>> tableau = []
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': None, 'max': None}
```

Correction

{{ correction(True, "

```
def rechercheMinMax(tab):
    d = {}
    d['min'] = None
    d['max'] = None
    for val in tab:
        if val < d['min']:
            d['min'] = val
        if val > d['max']:
            d['max'] = val
    return d
```

") }}

Exercice 15.2

Exercice 15.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 16.1 □

Exercice 16.1

Énoncé

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
```

Correction

{{ correction(True, "

```
def moyenne(tab):
    somme = 0
    for val in tab:
        somme += val
    return somme / len(tab)
```

") }}

Exercice 16.2

Exercice 16.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 17.1 □

Exercice 17.1

Énoncé

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

Exemples :

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

Correction

{{ correction(True, "

```
def indice_du_min(tab):
    indice_min = 0
    for i in range(len(tab)):
        if tab[i] < tab[indice_min]:
            indice_min = i
    return indice_min
```

") }}

Exercice 17.2

Exercice 17.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 18.1 □

Exercice 18.1

Énoncé

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers, et qui renvoie l'indice de la première occurrence de `elt` dans `tab` si `elt` est dans `tab` et `-1` sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4])
-1
>>> recherche(1, [10, 12, 1, 56])
```

```
2
>>> recherche(50, [1, 50, 1])
1
>>> recherche(15, [8, 9, 10, 15])
3
```

Correction

{{ correction(True, "

```
def recherche(tab, n):
    ind_debut = 0
    ind_fin = len(tab) - 1
    while ind_debut <= ind_fin:
        ind_milieu = (ind_debut + ind_fin) // 2
        if tab[ind_milieu] == n:
            return ind_milieu
        elif tab[ind_milieu] < n:
            ind_debut = ind_milieu + 1
        else:
            ind_fin = ind_milieu - 1
    return -1
```

") }}

Exercice 18.2

Exercice 18.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 19.1 □

Exercice 19.1

Énoncé

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
```

Correction

{{ correction(True, "

") }}

Exercice 19.2

Exercice 19.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 20.1 □

Exercice 20.1

Énoncé

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 le tableau `releve` des relevés et le tableau `date` des dates et qui renvoie la plus petite valeur relevée au cours de la période et l'année correspondante.

Exemple :

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

Correction

{{ correction(True, "

") }}

Exercice 20.2

Exercice 20.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 21.1 □

Exercice 21.1

Énoncé

Écrire une fonction python appelée `nb_repetitions` qui prend en paramètres un élément `elt` et une liste `tab` et renvoie le nombre de fois où l'élément apparaît dans la liste.

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
```

Correction

{{ correction(True, "

") }}

Exercice 21.2

Exercice 21.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 22.1 □

Exercice 22.1

Énoncé

Écrire en langage Python une fonction `recherche` prenant comme paramètres une variable `a` de type numérique (`float` ou `int`) et un tableau `t` (type `list`) et qui renvoie le nombre d'occurrences de `a` dans `t` .

Exemples :

```
>>> 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
```

Correction

{{ correction(True, "

") }}

Exercice 22.2

Exercice 22.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 23.1 □

Exercice 23.1

Énoncé

L'occurrence d'un caractère dans un phrase est le nombre de fois où ce caractère est présent.

Exemples :

- l'occurrence du caractère 'o' dans 'bonjour' est 2 ;
- l'occurrence du caractère 'b' dans 'Bébé' est 1 ;
- l'occurrence du caractère 'B' dans 'Bébé' est 1 ;
- l'occurrence du caractère ' ' dans 'Hello world !' est 2.

On cherche les occurrences des caractères dans une phrase. On souhaite stocker ces occurrences 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}`

Écrire une fonction `occurence_lettres` prenant comme paramètre une variable `phrase` de type `str`. Cette fonction doit renvoyer un dictionnaire de type constitué des occurrences des caractères présents dans la phrase.

Correction

{{ correction(True, "

") }}

Exercice 23.2

Exercice 23.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 24.1 □

identique au 18.1

Exercice 24.1

Énoncé

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers, et qui renvoie l'indice de la première occurrence de `elt` dans `tab` si `elt` est dans `tab` et `-1` sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4])
-1
>>> recherche(1, [10, 12, 1, 56])
2
>>> recherche(50, [1, 50, 1])
1
>>> recherche(15, [8, 9, 10, 15])
3
```

Correction

{{ correction(True, "

") }}

Exercice 24.2

Exercice 24.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 25.1 □

Exercice 25.1

Énoncé

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

Exemples :

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

Correction

{{ correction(True, "

") }}

Exercice 25.2

Exercice 25.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 26.1 □

Exercice 26.1

Énoncé

Écrire une fonction `occurrence_max` prenant en paramètres une chaîne de caractères `chaine` et qui renvoie le caractère le plus fréquent de la chaîne. La chaine ne contient que des lettres en minuscules sans accent. On pourra s'aider du tableau

`alphabet= ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o,','p','q','r','s','t','u','v','w','x','y','z']`

et du tableau `occurrence` de 26 éléments où l'on mettra dans `occurrence[i]` le nombre d'apparitions de `alphabet[i]` dans la chaine. Puis on calculera l'indice `k` d'un maximum du tableau `occurrence` et on affichera `alphabet[k]`.

Exemple :

Correction

```
>>> ch='je suis en terminale et je passe le bac et je souhaite poursuivre des etudes pour deve
>>> occurrence_max(ch)
'e'
```

Exercice 26.2

Exercice 26.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 27.1 □

Exercice 27.1

Énoncé

Écrire une fonction `moyenne` prenant en paramètres une liste d'entiers et qui renvoie la moyenne des valeurs de cette liste.

Exemple :

```
>>> moyenne([10,20,30,40,60,110])
45.0
```

Correction

{{ correction(True, "

```
def moyenne(tab):
    somme = 0
    for val in tab:
        somme += val
    return somme / len(tab)
```

") }}

Exercice 27.2

Exercice 27.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 28.1 □

Exercice 28.1

Énoncé

Dans cet exercice, un arbre binaire de caractères est stocké sous la forme d'un dictionnaire où les clefs sont les caractères des nœuds de l'arbre et les valeurs, pour chaque clef, la liste des caractères des fils gauche et droit du nœud.

Par exemple, l'arbre

est stocké dans

```
a = {'F':['B','G'], 'B':['A','D'], 'A':['',''], 'D':['C','E'], \
'C':['',''], 'E':['',''], 'G':['','I'], 'I':['','H'], \
'H':['','']}
```

Écrire une fonction récursive `taille` prenant en paramètres un arbre binaire `arbre` sous la forme d'un dictionnaire et un caractère `lettre` qui est la valeur du sommet de l'arbre, et qui renvoie la taille de l'arbre à savoir le nombre total de nœud. On pourra distinguer les 4 cas où les deux « fils » du nœud sont `''`, le fils gauche seulement est `''`, le fils droit seulement est `''`, aucun des deux fils n'est `''`.

Exemple :

```
>>> taille(a, 'F')
9
```

Correction

{{ correction(True, "

") }}

Exercice 28.2

Exercice 28.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 29.1 □

Exercice 29.1

Énoncé

Soit un nombre entier supérieur ou égal à 1 :

- s'il est pair, on le divise par 2 ;
- s'il est impair, on le multiplie par 3 et on ajoute 1.

Puis on recommence ces étapes avec le nombre entier obtenu, jusqu'à ce que l'on obtienne la valeur 1.

On définit ainsi la suite (U_n) par :

- $U_0 = k$, où k est un entier choisi initialement;
- $U_{n+1} = \dfrac{U_n}{2}$ si U_n est pair;
- $U_{n+1} = 3 \times U_n + 1$ si U_n est impair.

On admet que, quel que soit l'entier `k` choisi au départ, la suite finit toujours sur la valeur 1.

Écrire une fonction `calcul` prenant en paramètres un entier `n` strictement positif et qui renvoie la liste des valeurs de la suite, en partant de `n` et jusqu'à atteindre 1.

Exemple :

```
>>> calcul(7)
[7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

Correction

{{ correction(True, "

") }}

Exercice 29.2

Exercice 29.2

Énoncé

Correction

{{ correction(True, "

") }}

Exercice 30.1 □

Exercice 30.1

Énoncé

Programmer la fonction `multiplication`, prenant en paramètres deux nombres entiers `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
```

Correction

{{ correction(True, "

```python
def multiplication(n1, n2):
    if n1 < 0:
        return -multiplication(-n1, n2)
    if n2 < 0:
        return -multiplication(n1, -n2)
    resultat = 0
    for _ in range(n2):
        resultat += n1
    return resultat
```

") }}

Exercice 30.2

Exercice 30.2

Énoncé

Correction

{{ correction(True, "

") }}