# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

**Sujet n°21**

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 3 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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
```

## EXERCICE 2 (4 points)

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
```

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

## Sujet n°22

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

Programmer une fonction `renverse`, prenant en paramètre une chaîne de caractères non vide `mot` et renvoie une chaîne de caractères en inversant ceux de la chaîne `mot`.

Exemple :

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

## EXERCICE 2 (4 points)

Un nombre premier est un nombre entier naturel qui admet exactement deux diviseurs distincts entiers et positifs : 1 et lui-même.

Le crible d'Ératosthène permet de déterminer les nombres premiers plus petit qu'un certain nombre N fixé.
On considère pour cela un tableau `tab` de N booléens, initialement tous égaux à `True`, sauf `tab[Criblecrible0]` et `tab[1]` qui valent `False`, 0 et 1 n'étant pas des nombres premiers.

On parcourt alors ce tableau de gauche à droite.

Pour chaque indice `i` :

- si `tab[i]` vaut `True` : le nombre `i` est premier et on donne la valeur False à toutes les cases du tableau dont l'indice est un multiple de `i`, à partir de `2*i` (c'est-à-dire `2*i`, `3*i` …).

- si `tab[i]` vaut `False` : le nombre `i` n'est pas premier et on n'effectue aucun changement sur le tableau.

On dispose de la fonction `crible`, incomplète et donnée ci-dessous, prenant en paramètre un entier N strictement positif et renvoyant un tableau contenant tous les nombres premiers plus petits que N.

```
def crible(N):
    """
    Renvoie un tableau contenant tous les nombres premiers plus petits que N
    """
    premiers = []
    tab = [True] * N
    tab[0], tab[1] = False, False
    for i in range(..., N):
        if tab[i] == ...:
            premiers.append(...)
            for multiple in range(2*i, N, ...):
                tab[multiple] = ...
```

```
    return premiers

assert crible(40) == [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

Compléter le code de cette fonction.

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

## Sujet n°23

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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 ;
- 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': 220, 'Tim': 50})
('Alan', 222)
```

## EXERCICE 2 (4 points)

Nous avons l'habitude de noter les expressions arithmétiques avec des parenthèses comme par exemple : $(2 + 3) \times 5$.

Il existe une autre notation utilisée par certaines calculatrices, appelée notation postfixe, qui n'utilise pas de parenthèses. L'expression arithmétique précédente est alors obtenue en saisissant successivement 2, puis 3, puis l'opérateur +, puis 5, et enfin l'opérateur $\times$. On modélise cette saisie par le tableau `[2, 3, '+', 5, '*'].`

Autre exemple, la notation postfixe de $3 \times 2 + 5$ est modélisée par le tableau :

```
[3, 2, '*', 5, '+'].
```

D'une manière plus générale, la valeur associée à une expression arithmétique en notation postfixe est déterminée à l'aide d'une pile en parcourant l'expression arithmétique de gauche à droite de la façon suivante :

- Si l'élément parcouru est un nombre, on le place au sommet de la pile ;
- Si l'élément parcouru est un opérateur, on récupère les deux éléments situés au sommet de la pile et on leur applique l'opérateur. On place alors le résultat au sommet de la pile.
- A la fin du parcours, il reste alors un seul élément dans la pile qui est le résultat de l'expression arithmétique.

Dans le cadre de cet exercice, on se limitera aux opérations $\times$ et $+$.

Pour cet exercice, on dispose d'une classe `Pile` qui implémente les méthodes de base sur la structure de pile.

Compléter le script de la fonction `eval_expression` qui reçoit en paramètre une liste python représentant la notation postfixe d'une expression arithmétique et qui renvoie sa valeur associée.

Exemple :

```
>>> eval_expression([2, 3, '+', 5, '*'])
25
```

```
class Pile:
    """Classe définissant une structure de pile."""
    def __init__(self):
        self.contenu = []

    def est_vide(self):
        """
        Renvoie le booléen True si la pile est vide, False sinon.
        """
        return self.contenu == []

    def empiler(self, v):
        """Place l'élément v au sommet de la pile."""
        self.contenu.append(v)

    def depiler(self):
        """
        Retire et renvoie l'élément placé au sommet de la pile,
        si la pile n'est pas vide.
        """
        if not self.est_vide():
            return self.contenu.pop()


def eval_expression(tab):
    p = Pile()
    for ... in tab:
        if element != '+' ... element != '*':
            p.empiler(...)
        else:
            if element == ...:
                resultat = p.depiler() + ...
            else:
                resultat = ...
            p.empiler(...)
    return ...
```

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

## Sujet n°24

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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

Exemples :

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

## EXERCICE 2 (4 points)

On dispose de chaînes de caractères contenant uniquement des parenthèses ouvrantes et fermantes.

Un parenthésage est correct si :

- le nombre de parenthèses ouvrantes de la chaîne est égal au nombre de parenthèses fermantes.
- en parcourant la chaîne de gauche à droite, le nombre de parenthèses déjà ouvertes doit être, à tout moment, supérieur ou égal au nombre de parenthèses déjà fermées.

Ainsi, `"((()))(())"` est un parenthésage correct.
Les parenthésages `"())(()"` et `"(())(()"` sont, eux, incorrects.

On dispose du code de la classe `Pile` suivant :

```
class Pile:
    """Classe définissant une pile"""
    def __init__(self, valeurs=[]):
        self.valeurs = valeurs

    def est_vide(self):
        """Renvoie True si la pile est vide, False sinon"""
        return self.valeurs == []

    def empiler(self, c):
        """Place l'élément c au sommet de la pile"""
        self.valeurs.append(c)

    def depiler(self):
        """
        Supprime l'élément placé au sommet de la pile, à condition qu'elle
        soit non vide
        """
        if self.est_vide() == False:
```

```
            self.valeurs.pop()
```

On souhaite programmer une fonction `parenthesage` qui prend en paramètre une chaîne `ch` de parenthèses et renvoie `True` si la chaîne est bien parenthésée et `False` sinon.
Cette fonction utilise une pile et suit le principe suivant : en parcourant la chaîne de gauche à droite, si on trouve une parenthèse ouvrante, on l'empile au sommet de la pile et si on trouve une parenthèse fermante, on dépile (si possible !) la parenthèse ouvrante stockée au sommet de la pile.
La chaîne est alors bien parenthésée si, à la fin du parcours, la pile est vide.
Elle est, par contre, mal parenthésée :
- si dans le parcours, on trouve une parenthèse fermante, alors que la pile est vide ;
- ou si, à la fin du parcours, la pile n'est pas vide.

```
def parenthesage (ch):
    """Renvoie True si la chaîne ch est bien parenthésée et False sinon"""
    p = Pile()
    for c in ch:
        if c == ...:
            p.empiler(c)
        elif c == ...:
            if p.est_vide():
                return ...
        else:
            ...
    return p.est_vide()

assert parenthesage("((()())(()))") == True
assert parenthesage("())(()") == False
assert parenthesage("(())(()") == False
```

Compléter le code de la fonction `parenthesage`.

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

**Sujet n°25**

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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 :

```
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 2 (4 points)

On considère des tableaux de nombres dont tous les éléments sont présents exactement trois fois et à suivre, sauf un élément qui est présent une unique fois et que l'on appelle « l'intrus ». Voici quelques exemples :

```
tab_a = [3, 3, 3, 9, 9, 9, 1, 1, 1, 7, 2, 2, 2, 4, 4, 4, 8, 8, 8, 5, 5, 5]
#l'intrus est 7

tab_b = [8, 5, 5, 5, 9, 9, 9, 18, 18, 18, 3, 3, 3]
#l'intrus est 8

tab_c = [5, 5, 5, 1, 1, 1, 0, 0, 0, 6, 6, 6, 3, 8, 8, 8]
#l'intrus est 3
```

On remarque qu'avec de tels tableaux :

- pour les indices multiples de 3 situés strictement avant l'intrus, l'élément correspondant et son voisin de droite sont égaux,
- pour les indices multiples de 3 situés après l'intrus, l'élément correspondant et son voisin de droite - s'il existe - sont différents.

Ce que l'on peut observer ci-dessous en observant les valeurs des paires de voisins marquées par des caractères ^ :

```
[3, 3, 3, 9, 9, 9, 1, 1, 1, 7, 2, 2, 2, 4, 4, 4, 8, 8, 8, 5, 5, 5]
 ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^  ^     ^
 0        3        6        9        12       15       18       21
```

Dans des listes comme celles ci-dessus, un algorithme récursif pour trouver l'intrus consiste alors à choisir un indice `i` multiple de 3 situé approximativement au milieu des indices parmi lesquels se trouve l'intrus.

Puis, en fonction des valeurs de l'élément d'indice `i` et de son voisin de droite, à appliquer récursivement l'algorithme à la moitié droite ou à la moitié gauche des indices parmi lesquels se trouve l'intrus.

Compléter la fonction ci-dessous qui met en œuvre cet algorithme.

```
def trouver_intrus(tab, g, d):
    '''
    Renvoie la valeur de l'intrus situé entre les indices g et d
    dans la liste tab où :
        tab vérifie les conditions de l'exercice,
        g et d sont des multiples de 3.
    '''

    if g == d:
        return ...

    else:
        nombre_de_triplets = (d - g)// ...
        indice = g + 3 * (nombre_de_triplets // 2)
        if ... :
            return ...
        else:
            return ...
```

*Exemples :*

```
>>> trouver_intrus([3, 3, 3, 9, 9, 9, 1, 1, 1, 7, 2, 2, 2, 4, 4, 4, 8, 8,
8, 5, 5, 5], 0, 21)
7

>>> trouver_intrus([8, 5, 5, 5, 9, 9, 9, 18, 18, 18, 3, 3, 3], 0, 12)
8

>>> trouver_intrus([5, 5, 5, 1, 1, 1, 0, 0, 0, 6, 6, 6, 3, 8, 8, 8], 0, 15)
3
```

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

### Sujet n°26

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 2 à 2 / 2**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

Écrire une fonction `RechercheMin` 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 :

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

## EXERCICE 2 (4 points)

On considère la fonction `separe` ci-dessous qui prend en argument un tableau `tab` dont les éléments sont des `0` et des `1` et qui sépare les `0` des `1` en plaçant les `0` en début de tableau et les `1` à la suite.

```
def separe(tab):
    i = 0
    j = ...
    while i < j :
        if tab[i] == 0 :
            i = ...
        else :
            tab[i], tab[j] = ...
            j = ...
    return tab
```

Compléter la fonction `separe` ci-dessus.

Exemples :

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

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

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

### Sujet n°27

---

DUREE DE L'EPREUVE : 1 heure

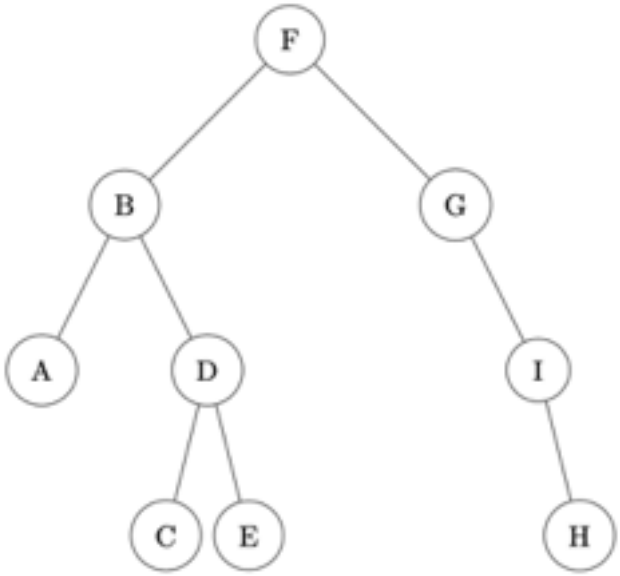
**Le sujet comporte 3 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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
```

## EXERCICE 2 (4 points)

On considère l'algorithme de tri de tableau suivant : à chaque étape, on parcourt depuis le début du tableau tous les éléments non rangés et on place en dernière position le plus grand élément.

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 grand élément avec le dernier. Le tableau devient

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

Etape 2 : on parcourt tous les éléments **sauf le dernier**, on permute le plus grand élément trouvé avec l'avant dernier. Le tableau devient :

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

Et ainsi de suite.

La code de la fonction `tri_iteratif` qui implémente cet algorithme est donné ci-dessous.

```
def tri_iteratif(tab):
    for k in range( ... , 0, -1):
        imax = ...
        for i in range(0 , ... ):
            if tab[i] > ... :
                imax = i
        if tab[imax] > ... :
            ... , tab[imax] = tab[imax] , ...
    return tab
```

Compléter le code qui doit donner :

```
>>> tri_iteratif([41, 55, 21, 18, 12, 6, 25])
[6, 18, 12, 21, 25, 41, 55]
```

On rappelle que l'instruction

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

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

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

**Sujet n°28**

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 2 à 2 / 2**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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 2 (4 points)

On considère la fonction `dec_to_bin` ci-dessous qui prend en paramètre un entier positif `a` en écriture décimale et qui renvoie son écriture binaire sous la forme d'une chaine de caractères.

```
def dec_to_bin(a):
    bin_a = ...
    a = a//2
    while a ... :
        bin_a = ... + bin_a
        a = ...
    return bin_a
```

Compléter la fonction `dec_to_bin`.

Exemples :

```
>>> dec_to_bin(83)
'1010011'
>>> dec_to_bin(127)
'1111111'
```

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

## Sujet n°29

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 3 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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

$U_1 = 1$, $U_2 = 1$ et, pour tout entier naturel n, par $U_{n+2} = U_{n+1} + U_n$.

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).

Exemples :

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

## EXERCICE 2 (4 points)

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'])
```

# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

## Sujet n°30

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 3 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

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]
```

## EXERCICE 2 (4 points)

Les chiffres romains sont un système ancien d'écriture des nombres.

Les chiffres romains sont: I, V, X, L, C, D, et M.
Ces symboles représentent respectivement 1, 5, 10, 50, 100, 500, et 1000 en base dix.

Lorsque deux caractères successifs sont tels que le caractère placé à gauche possède une valeur supérieure ou égale à celui de droite, le nombre s'obtient en additionnant le caractère de gauche à la valeur de la chaîne située à droite.

Ainsi, `"XVI"` est le nombre 16 car X + VI = 10 + 6.

Lorsque deux caractères successifs sont tels que le caractère placé à gauche possède une valeur strictement inférieure à celui de droite, le nombre s'obtient en retranchant le caractère de gauche à la valeur de la chaîne située à droite.

Ainsi, `"CDIII"` est le nombre 403 car DIII – C = 503 – 100.

On dispose d'un dictionnaire `dico`, à compléter, où les clés sont les caractères apparaissant dans l'écriture en chiffres romains et où les valeurs sont les nombres entiers associés en écriture décimale.

On souhaite créer une fonction récursive `rom_to_dec` qui prend en paramètre une chaîne de caractères (non vide) représentant un nombre écrit en chiffres romains et renvoyant le nombre associé en écriture décimale :

```
def rom_to_dec (nombre):
      """ Renvoie l'écriture décimale du nombre donné en chiffres romains """

      dico = {"I":1, "V":5, ...}
      if len(nombre) == 1:
            return ...

      else:

    ### on supprime le premier caractère de la chaîne contenue dans la
variable nombre et cette nouvelle chaîne est enregistrée dans la variable
nombre_droite
            nombre_droite = nombre[1:]

            if dico[nombre[0]] >= dico[nombre[1]]:
                  return dico[nombre[0]] + ...
            else:
                  return ...


assert rom_to_dec("CXLII") == 142
```