Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

# CORRIGÉ NON OFFICIEL

Proposition de corrigé rédigée par FormaV avec l'aide d'outils d'intelligence artificielle. Elle peut contenir des erreurs et ne remplace ni le corrigé ni le barème officiels de l'examen.

Les sujets officiels des sessions passées et nos sujets d'entraînement sont classés sur https://bac-maths-nsi.fr/annales-sujets-et-corriges

Document mis en ligne par FormaV®. Reproduction ou diffusion sans autorisation interdite.

BACCALAURÉAT GÉNÉRAL • CORRIGÉ DÉTAILLÉ

# Numérique et sciences informatiques

Session 2025 • Centres étrangers • Jour 2

Code sujet **25-NSIJ2G11** 3 heures 30

Proposition de corrigé pédagogique • FormaV

## Se repérer dans le corrigé


Exercice 1 — Bases de données relationnelles et requêtes SQL
11 réponses • sujet p. 2–4

Exercice 2 — Listes, dictionnaires, fonctions et récursivité
11 réponses • sujet p. 5–8

Exercice 3 — Graphes, programmation objet et récursivité
16 réponses • sujet p. 9–16


Les numéros, parties et pages renvoient au sujet original. Chaque réponse rappelle la consigne, explique la démarche et donne le résultat.



FormaV - document d'entraînement non officiel

# Exercice 1

**6 points**

Bases de données relationnelles et requêtes SQL

## Question 1.

Sujet p. 2, 3

> Expliquer, dans les relations présentées, le rôle des clés primaires.

Une clé primaire identifie sans ambiguïté chaque ligne d'une table : ses valeurs sont uniques et non nulles. D'après le schéma, les clés primaires sont `joueur.id_joueur`, `equipe.id_equipe` et `match.id_match`. Elles permettent de désigner précisément un joueur, une équipe ou un match.

**RÉPONSE À RETENIR**

Les clés primaires identifient de manière unique les lignes de leur table.

## Question 2.

Sujet p. 2

> Expliquer quelles données n'auraient pas pu être stockées dans la table match si le champ id_match n'avait pas été introduit dans cette table.

Le champ `id_match` fournit un identifiant propre à chaque rencontre. Sans lui, les autres champs ne permettraient pas toujours de distinguer deux rencontres, par exemple si les mêmes équipes se rencontraient plusieurs fois avec le même résultat. On ne pourrait alors pas identifier séparément ces matchs dans la table.

**RÉPONSE À RETENIR**

Sans `id_match`, il serait impossible de distinguer et d'identifier sans ambiguïté certaines rencontres, notamment des matchs répétés entre les mêmes équipes ayant les mêmes autres données.

## Question 3.

Sujet p. 3

> Donner le résultat de la requête SELECT prenom FROM joueur WHERE ann_naiss < 1985 en l'appliquant à l'extrait de la table joueur donné dans l'énoncé.

La condition `ann_naiss < 1985` retient Henri (1971), Laure (1980), Brigitte (1972) et Laure (1979). La requête ne sélectionne que les prénoms : Laure apparaît donc deux fois.

**RÉPONSE À RETENIR**

Résultat de la requête sur l'extrait fourni :

| prenom |
|---|
| Henri |
| Laure |
| Brigitte |
| Laure |


FormaV - document d'entraînement non officiel

## Question 4.

Sujet p. 3

> Modifier la requête précédente afin d'éviter les éventuels doublons.

Le mot-clé `DISTINCT` permet de ne conserver qu'une occurrence de chaque prénom.

```
SELECT DISTINCT prenom
FROM joueur
WHERE ann_naiss < 1985;
```

**RÉPONSE À RETENIR**

Sur l'extrait, les prénoms distincts sont Henri, Laure et Brigitte. Sans `ORDER BY`, SQL ne garantit pas leur ordre d'affichage.

| prenom |
|---|
| Henri |
| Laure |
| Brigitte |

## Question 5.

Sujet p. 2, 3

> Écrire une requête SQL permettant d'obtenir tous les noms, années de naissance et numéros de téléphone portable des personnes qui habitent à Bois-Plage.

On sélectionne les champs demandés dans la table `joueur`, puis on filtre les lignes sur la commune.

```
SELECT nom, ann_naiss, num_port
FROM joueur
WHERE commune = 'Bois-Plage';
```

**RÉPONSE À RETENIR**

Résultat sur l'extrait fourni :

| nom | ann_naiss | num_port |
|---|---|---|
| Leclerc | 2008 | 0660358945 |
| Desousa | 1980 | 0746881113 |
| Barbier | 1979 | 0787028125 |

## Question 6.

Sujet p. 2, 3, 4

> Écrire une requête SQL permettant d'obtenir le nom et le prénom du joueur j_1 de l'équipe « Les Kangourous ». L'équipe « Les Kangourous » n'apparaît pas dans l'extrait.

La colonne `equipe.j_1` contient l'identifiant du premier joueur de l'équipe. On la relie donc à la clé primaire `joueur.id_joueur`, puis on filtre sur le nom de l'équipe.

```sql
SELECT j.nom, j.prenom
FROM equipe AS e
JOIN joueur AS j ON e.j_1 = j.id_joueur
WHERE e.nom = 'Les Kangourous';
```

**RÉPONSE À RETENIR**

« Les Kangourous » ne figure pas dans l'extrait de la table `equipe`. La jointure appliquée aux extraits fournis ne renvoie donc aucune ligne. Le résultat dans la base complète ne peut pas être déduit de ces extraits.

| nom | prenom |
|---|---|

## Question 7.

Sujet p. 2, 3, 4

L'équipe « Volley Warriors » a terminé le tournoi avec un total de 5 points. Écrire une requête SQL permettant de mettre à jour la table equipe avec le nombre de points gagnés par cette équipe.

La commande `UPDATE` modifie la valeur de `points` pour la ligne dont le nom est « Volley Warriors ».

```sql
UPDATE equipe
SET points = 5
WHERE nom = 'Volley Warriors';
```

**RÉPONSE À RETENIR**

La ligne de l'équipe « Volley Warriors » reçoit la valeur `5` dans la colonne `points`.

## Question 8.

Sujet p. 2, 4

Écrire une requête SQL permettant de supprimer de la table joueur le joueur ayant pour identifiant le numéro 35.

On cible le joueur à supprimer grâce à sa clé primaire, `id_joueur`.

```sql
DELETE FROM joueur
WHERE id_joueur = 35;
```

Dans l'extrait, le joueur 35 est référencé dans `equipe.j_4` pour l'équipe 10. Si la base applique cette clé étrangère sans suppression en cascade, cette référence devra être traitée avant que la suppression puisse réussir.

**RÉPONSE À RETENIR**

La commande cible bien le joueur d'identifiant 35. La suppression effective dépend du traitement de la référence à ce joueur dans `equipe` et des contraintes de clés étrangères.

## Question 9.

Sujet p. 2, 4

> Écrire une requête SQL permettant d'obtenir la liste des identifiants de matchs auxquels a participé l'équipe ayant pour identifiant 12.

L'équipe 12 peut apparaître dans l'une ou l'autre des colonnes `eq_1` et `eq_2`. Il faut donc tester les deux colonnes.

```
SELECT id_match
FROM match
WHERE eq_1 = 12 OR eq_2 = 12;
```

**RÉPONSE À RETENIR**

Dans l'extrait, les couples d'équipes sont (3, 8), (3, 9) et (3, 10). L'équipe 12 n'y apparaît pas : le résultat sur cet extrait est vide. L'extrait ne permet pas de connaître le résultat sur la table complète.

| id_match |
|---|

## Question 10.

Sujet p. 2, 4

> Écrire une requête SQL permettant d'obtenir la liste des identifiants des matchs pour lesquels le joueur 1 de l'équipe 1 du match vient de la commune de Bois-Plage.

On relie le match à l'équipe 1, puis cette équipe à son joueur 1. On peut alors filtrer les joueurs sur leur commune.

```
SELECT m.id_match
FROM match AS m
JOIN equipe AS e ON m.eq_1 = e.id_equipe
JOIN joueur AS j ON e.j_1 = j.id_joueur
WHERE j.commune = 'Bois-Plage';
```

**RÉPONSE À RETENIR**

Dans l'extrait de `match`, les trois matchs ont `eq_1 = 3`. L'équipe 3 ne figure pas dans l'extrait de `equipe` : les jointures ne renvoient donc aucune ligne à partir des extraits fournis. Le résultat sur la base complète n'est pas déterminable avec ces seuls extraits.

| id_match |
|---|

## Question 11.

Sujet p. 2, 4

> Écrire une requête SQL permettant d'obtenir la liste, classée par ordre alphabétique, des noms et prénoms des joueurs ayant gagné au moins un match en tant que joueur 1 de l'équipe 1 du match.

On relie le match à son équipe 1, puis à son joueur 1. Ce joueur a gagné le match lorsque l'équipe gagnante est l'équipe 1, donc lorsque `eq_gagnante = eq_1`. `DISTINCT` évite de répéter un joueur qui aurait gagné plusieurs matchs. Enfin, le tri par nom puis par prénom donne un ordre alphabétique des personnes.

```
SELECT DISTINCT j.nom, j.prenom
FROM match AS m
JOIN equipe AS e ON m.eq_1 = e.id_equipe
JOIN joueur AS j ON e.j_1 = j.id_joueur
WHERE m.eq_gagnante = m.eq_1
ORDER BY j.nom, j.prenom;
```

**RÉPONSE À RETENIR**

Dans l'extrait de `match`, seul le match 33 a été gagné par l'équipe 1 : `eq_1 = 3` et `eq_gagnante = 3`. L'équipe 3 ne figure pas dans l'extrait de `equipe`, donc le nom et le prénom du joueur ne peuvent pas être obtenus à partir des extraits fournis.

| nom | prenom |
|---|---|

# Exercice 2

**6 points**

Listes, dictionnaires, fonctions et récursivité

## A

### Question 1.

Sujet p. 5, 6

> Trouver la valeur exprimée en hexadécimal de la clé qui sera associée au mot 'EW'.

Les codes ASCII donnés sont `E = 0x45` et `W = 0x57`. Leur somme vaut `0x45 + 0x57 = 0x9C`. Cette valeur tient sur un octet : aucune réduction supplémentaire n'est nécessaire.

> **RÉPONSE À RETENIR**
>
> La clé associée à `EW` est **`0x9C`**.

### Question 2.

Sujet p. 5, 6

> Comparer, sans les calculer, les valeurs des clés qui seront associées aux mots 'SAC' et 'CAS'. Justifier la réponse.

Les mots `SAC` et `CAS` contiennent les mêmes lettres. La fonction additionne leurs codes ASCII sans tenir compte de l'ordre des lettres ; par commutativité de l'addition, les sommes sont donc égales. La réduction modulo `0x100` conserve cette égalité.

> **RÉPONSE À RETENIR**
>
> Les clés associées à `SAC` et `CAS` sont égales.

### Question 3.

Sujet p. 6

> Recopier et compléter les lignes 2, 3 et 4 de la fonction code_hachage.

On initialise `somme` à zéro, puis on parcourt chaque caractère de `mot`. À chaque tour, on ajoute son code ASCII, obtenu avec `ord`.

> **RÉPONSE À RETENIR**
>
> ```
> def code_hachage(mot):
>     somme = 0
>     for caractere in mot:
>         somme = somme + ord(caractere)
>     return somme % 0x100
> ```

### Question 4.

Sujet p. 6

> Expliquer l'expression somme % 0x100 dans le code ci-dessus et donner les valeurs possibles pour la clé.

`0x100` vaut 256 en base dix. Écrire la somme sous la forme `256q + r`, où `0 ≤ r < 256`, montre que `somme % 0x100` renvoie le reste `r` : c'est l'octet de poids faible.


FormaV - document d'entraînement non officiel

Il existe une incohérence dans l'énoncé : il précise que les clés sont des entiers positifs, alors que le calcul modulo peut donner 0. Par exemple, la somme des codes ASCII de `ZZZZZZZIA` vaut `7 × 90 + 73 + 65 = 768`, multiple de 256 ; la clé calculée est donc 0.

> **RÉPONSE À RETENIR**
>
> Les valeurs possibles de `somme % 0x100` vont de **0 à 255**, soit de `0x00` à `0xFF`. La valeur 0 contredit la précision de l'énoncé selon laquelle les clés sont positives.

## B

### Question 5.

Sujet p. 7

> Déterminer, dans le pire cas, l'ordre de grandeur du nombre de comparaisons entre chaînes de caractères effectuées par un appel à ajouter_mot_liste. Exprimer le résultat en fonction de n, le nombre de mots présents dans liste.

À chaque tour de boucle, la fonction compare `mot` à un mot de la liste. Dans le pire cas, aucun mot de la liste n'est supérieur à `mot` : il y a alors une comparaison pour chacun des `n` mots avant l'ajout en fin de liste. La question porte sur les comparaisons entre chaînes, et non sur le décalage des éléments lors d'une insertion.

> **RÉPONSE À RETENIR**
>
> Dans le pire cas, le nombre de comparaisons est de l'ordre de grandeur de **O(n)** : il est linéaire en `n`.

### Question 6.

Sujet p. 7

> Rappeler avec quelle expression Python on peut tester si une clé c est déjà présente dans un dictionnaire dico.

Pour tester si un dictionnaire contient une clé, on utilise l'opérateur Python `in`, qui teste l'appartenance des clés au dictionnaire.

> **RÉPONSE À RETENIR**
>
> L'expression demandée est `c in dico`.

### Question 7.

Sujet p. 5, 6, 7

> Écrire un code pour la fonction ajouter_mot_dict(dict_mots, mot), qui a pour paramètres dict_mots, un dictionnaire qui associe à chaque clé entière possible une liste de mots, et mot, le mot à ajouter dans dict_mots, et qui réalise cet ajout sans renvoyer aucune valeur. Utiliser les fonctions code_hachage et ajouter_mot_liste.

On calcule la clé du mot. Si elle est absente, on crée la liste associée. On appelle ensuite `ajouter_mot_liste`, qui insère le mot en conservant l'ordre alphabétique. La fonction modifie le dictionnaire et, en l'absence de `return`, ne renvoie aucune valeur.

**RÉPONSE À RETENIR**

```
def ajouter_mot_dict(dict_mots, mot):
    cle = code_hachage(mot)
    if cle not in dict_mots:
        dict_mots[cle] = []
    dict_mots[cle] = ajouter_mot_liste(dict_mots[cle], mot)
```

## C

### Question 8.

Sujet p. 8

> Lors de l'exécution de l'exemple est_present(liste_mots, 'NSI', 0, len(liste_mots)), donner les valeurs des paramètres debut et fin prises lors de chaque appel de la fonction est_present.

Dans l'exemple, `len(liste_mots)` vaut 5. Au premier appel, `(debut, fin) = (0, 5)` et le milieu vaut 2. Comme `liste_mots[2]` vaut `'PYTHON'`, supérieur à `'NSI'`, le code appelle la fonction avec `(0, 1)`. Au deuxième appel, le milieu vaut 0 ; `'FONCTION'` est inférieur à `'NSI'`, donc l'appel suivant reçoit `(1, 1)`. Le milieu vaut alors 1 et le mot recherché est trouvé.

La trace correspond exactement au code imprimé. Cependant, ce code ne respecte pas de façon cohérente la convention annoncée d'une borne `fin` exclusive : ses tests et ses appels traitent les bornes comme inclusives, et il réinitialise aussi `debut` à 0 à la ligne 6.

**RÉPONSE À RETENIR**

Les couples `(debut, fin)` pris successivement sont **`(0, 5)`, `(0, 1)` et `(1, 1)`**.

### Question 9.

Sujet p. 8

> Expliquer pourquoi la méthode dichotomique permet d'effectuer, en général, moins d'opérations qu'une recherche simple qui consisterait à comparer un par un les mots de la liste avec le mot cherché.

Dans une liste triée, comparer le mot recherché avec celui du milieu permet d'éliminer environ la moitié des mots possibles : si le mot du milieu est trop grand, on poursuit dans la moitié gauche ; s'il est trop petit, on poursuit dans la moitié droite. Une recherche simple peut au contraire devoir comparer le mot recherché à chaque élément. La dichotomie suppose que la liste est triée. Or la liste de l'exemple `['FONCTION', 'NSI', 'PYTHON', 'OBJET', 'RAM']` n'est pas triée dans l'ordre alphabétique ; la méthode ne serait donc pas fiable sur cette liste en général, même si l'exemple donné aboutit à `True`.

**RÉPONSE À RETENIR**

La recherche dichotomique demande généralement moins d'opérations, car chaque comparaison permet d'éliminer environ la moitié de la zone de recherche. Cela n'est garanti que si la liste est triée.

## Question 10.

Sujet p. 8

> Donner l'ordre de grandeur du nombre de comparaisons de mots effectuées par la méthode dichotomique sur une liste de longueur n.

À chaque étape d'une recherche dichotomique correcte, la zone de recherche est divisée par environ deux. Après `k` étapes, sa taille est de l'ordre de `n / 2^k` ; il faut donc environ $\log_2(n)$ étapes pour réduire cette zone à un élément. Chaque étape comporte un nombre constant de comparaisons de mots.

Le code imprimé dans le sujet contient toutefois des erreurs : notamment, la ligne 6 réinitialise `debut` à 0, et la convention de borne finale exclusive n'est pas respectée par les tests et appels. L'ordre de grandeur demandé est celui de la méthode dichotomique correctement implémentée.

**RÉPONSE À RETENIR**

Le nombre de comparaisons est de l'ordre de grandeur de **O(log n)**.

## Question 11.

Sujet p. 5, 6, 8

> Écrire un code pour la fonction mot_present(dict_mots, mot), qui a pour paramètres dict_mots, un dictionnaire qui associe à chaque clé entière possible une liste de mots, et mot, le mot à rechercher dans dict_mots, et qui renvoie True si le mot est présent et False sinon. Utiliser les fonctions code_hachage et est_present.

La clé calculée détermine la liste où le mot pourrait se trouver. Si cette clé n'est pas dans le dictionnaire, le mot est absent. Sinon, on effectue une recherche dichotomique dans la liste alphabétiquement triée.

La fonction `est_present` imprimée dans le sujet comporte des incohérences : elle annonce un intervalle où `fin` est exclusif, mais son test d'arrêt et ses appels ne suivent pas cette convention ; de plus, la ligne 6 réinitialise à tort la borne `debut`. Pour garantir la recherche, voici une version corrigée de `est_present`, puis la fonction demandée, qui l'utilise.

**RÉPONSE À RETENIR**

```
def est_present(liste, mot, debut, fin):
    if debut >= fin:
        return False
    milieu = (debut + fin) // 2
    if liste[milieu] > mot:
        return est_present(liste, mot, debut, milieu)
    elif liste[milieu] < mot:
        return est_present(liste, mot, milieu + 1, fin)
    else:
        return True

def mot_present(dict_mots, mot):
    cle = code_hachage(mot)
    if cle not in dict_mots:
        return False
    liste = dict_mots[cle]
    return est_present(liste, mot, 0, len(liste))
```

# Exercice 3

**8 points**

Graphes, programmation objet et récursivité

### Question 1.

Sujet p. 9, 10

> Donner un chemin non prolongeable qui commence par 3 → 9 → 1 → 2.

Après la séquence imposée, on prolonge avec des nombres voisins du dernier nombre, sans réutiliser ceux déjà présents. On peut poursuivre de 2 à 4, car 4 est un multiple de 2, puis de 4 à 8, car 8 est un multiple de 4. Depuis 8, les diviseurs différents de 8 compris entre 1 et 9 sont 1, 2 et 4, déjà utilisés, et 8 n'a pas de multiple dans cet intervalle.

Le chemin obtenu respecte donc les règles et ne peut pas être prolongé.

> **RÉPONSE À RETENIR**
>
> **Un chemin non prolongeable convenable est :** 3 → 9 → 1 → 2 → 4 → 8.

## A

### Question 2.

Sujet p. 11, 12

> Donner les valeurs prises par la variable locale valeur lors de l'exécution de creer_jeu(9).

Dans `creer_jeu(9)`, la boucle parcourt `range(1, 9+1)`, soit les entiers de 1 inclus à 10 exclu. La variable `valeur` prend successivement chacun de ces entiers.

> **RÉPONSE À RETENIR**
>
> `valeur` prend successivement les valeurs 1, 2, 3, 4, 5, 6, 7, 8 et 9.

### Question 3.

Sujet p. 11, 12

> Une fois ce code exécuté, donner la valeur de jeu_9[0].valeur.

Les sommets sont ajoutés à la liste dans l'ordre des valeurs 1 à 9. Comme l'indice 0 désigne le premier élément, `jeu_9[0]` est le sommet représentant 1.

> **RÉPONSE À RETENIR**
>
> `jeu_9[0].valeur` vaut **1**.

### Question 4.

Sujet p. 12

> Expliquer le test réalisé à la ligne 5 de la fonction relier_diviseurs.

Le test `s != self` exclut le sommet lui-même : dans ce jeu, un entier n'est pas considéré comme son propre diviseur. Le test `self.valeur % s.valeur == 0` vérifie que la division de la valeur de `self` par celle de `s` a un reste nul. Si les deux conditions sont satisfaites, la valeur de `s` est un diviseur de celle de `self`, différent de lui-même.

Par exemple, pour le sommet de valeur 6, les valeurs 1, 2 et 3 satisfont ces conditions.


FormaV - document d'entraînement non officiel

**RÉPONSE À RETENIR**

La ligne 5 sélectionne les sommets différents de `self` dont la valeur divise celle de `self`.

### Question 5.

Sujet p. 11, 12

Donner le contenu de jeu_9[5].diviseurs après l'exécution de jeu_9[5].relier_diviseurs(jeu).

L'indice 5 désigne le sixième sommet, celui de valeur 6. Parmi les valeurs de 1 à 9, ses diviseurs différents de 6 sont 1, 2 et 3. La méthode parcourt `jeu` dans l'ordre et ajoute les objets correspondants à `self.diviseurs`.

**RÉPONSE À RETENIR**

`jeu_9[5].diviseurs` contient les sommets de valeurs 1, 2 et 3, soit les objets `[jeu_9[0], jeu_9[1], jeu_9[2]]`. Les valeurs de ces objets sont `[1, 2, 3]`.

### Question 6.

Sujet p. 12

Écrire la ligne 6 manquante dans le code de la nouvelle version de la fonction creer_jeu.

Il faut appeler la méthode sur le sommet qui vient d'être créé, en lui passant les sommets déjà présents dans `jeu`. La méthode relie ce sommet à ses diviseurs antérieurs et ajoute aussi ce sommet à la liste des multiples de chacun de ces diviseurs. Il peut ensuite être ajouté à `jeu`.

**RÉPONSE À RETENIR**

La ligne 6 manquante est :

```
sommet.relier_diviseurs(jeu)
```

### Question 7.

Sujet p. 13

Recopier et compléter la ligne 3 du code de la méthode lister_diviseurs.

Les éléments de `self.diviseurs` sont des objets `Sommet`. Une compréhension de liste permet de construire la liste des valeurs de ces objets en conservant leur ordre.

**RÉPONSE À RETENIR**

```
return [sommet.valeur for sommet in self.diviseurs]
```

Pour le sommet représentant 8, cette méthode renvoie `[1, 2, 4]`.

### Question 8.

Sujet p. 11, 13

Recopier et compléter le jeu de tests qui a pour but de vérifier que le sommet représentant le nombre 3 est bien relié à tous ses voisins dans le graphe jeu.

Le sommet de valeur 3 est `jeu[2]`, car l'indice 0 correspond à la valeur 1. Ses voisins sont son diviseur 1 et ses multiples compris entre 1 et 9, soit 6 et 9. Les assertions vérifient la présence de ces trois voisins dans les listes appropriées.

**RÉPONSE À RETENIR**

```
l_div_3 = jeu[2].lister_diviseurs()
l_mult_3 = jeu[2].lister_multiples()
assert 1 in l_div_3
assert 6 in l_mult_3
assert 9 in l_mult_3
```

## B

### Question 9.

Sujet p. 14

> Dans la méthode defiler, expliquer la ligne de code assert not self.est_vide().

L'instruction `assert not self.est_vide()` vérifie que la file n'est pas vide avant d'en retirer un élément. Si la file est vide, `self.est_vide()` renvoie `True`, donc la condition précédée de `not` est fausse et une erreur d'assertion est déclenchée. Cela évite ensuite d'accéder à un élément inexistant.

**RÉPONSE À RETENIR**

Cette assertion empêche donc le retrait d'un élément d'une file vide.

### Question 10.

Sujet p. 14

> Recopier et compléter le code de la méthode taille de la classe File.

La liste `donnees` conserve les éléments qui ont déjà été défilés. Les éléments encore présents dans la file correspondent aux indices allant de `decalage` à `len(self.donnees) - 1`. Leur nombre est donc la longueur de la liste moins `decalage`.

**RÉPONSE À RETENIR**

```
def taille(self):
    """Renvoie le nombre d'éléments présents dans la file"""
    return len(self.donnees) - self.decalage
```

Par exemple, après avoir enfilé 1, 2, 3 puis défilé 1, la taille vaut `3 - 1 = 2`.

### Question 11.

Sujet p. 13, 14

> À l'aide des méthodes de la classe File, écrire une séquence d'instructions qui crée une file vide, y ajoute 1 puis 2 puis 3, vérifie qu'elle est de la longueur attendue à l'aide d'un assert, défile un à un les trois éléments et vérifie à l'aide d'assert qu'ils sortent dans le bon ordre, puis vérifie que la file est bien vide.

On enfile les trois valeurs dans l'ordre demandé, on vérifie la taille, puis on contrôle que les retraits les rendent dans le même ordre. Après ces trois retraits, la file doit être vide.

RÉPONSE À RETENIR

```
f_test = File()
f_test.enfiler(1)
f_test.enfiler(2)
f_test.enfiler(3)
assert f_test.taille() == 3
assert f_test.defiler() == 1
assert f_test.defiler() == 2
assert f_test.defiler() == 3
assert f_test.est_vide()
```

Trace : après les insertions, la taille vaut 3. Les retraits renvoient 1, puis 2, puis 3 ; après le dernier retrait, la file est vide.

## C

### Question 12.

Sujet p. 15

> Recopier et compléter les lignes 6, 7, 8, 10, 13 et 14 du code de la fonction rechercher_chemins.

La boucle continue tant que la file n'est pas vide. On retire un chemin, puis on examine les voisins de son dernier sommet. `prolongeable` est initialisé à `False` et passe à `True` dès qu'un voisin n'est pas déjà dans le chemin. Le nouveau chemin est créé par concaténation, sans modifier le chemin examiné.

Voici la fonction complétée :

RÉPONSE À RETENIR

```
def rechercher_chemins(jeu):
    chemins_np = []
    f = File()
    for sommet in jeu:
        f.enfiler([sommet])
    while not f.est_vide():
        chemin = f.defiler()
        dernier = chemin[-1]
        voisins = dernier.diviseurs + dernier.multiples
        prolongeable = False
        for voisin in voisins:
            if voisin not in chemin:
                prolongeable = True
                f.enfiler(chemin + [voisin])
        if not prolongeable:
            chemins_np.append(chemin)
    return chemins_np
```

Trace sur un chemin du sujet : pour `[3, 9, 1, 2]`, les voisins de 2 sont 1, 4, 6 et 8. Le voisin 1 est déjà dans le chemin ; les chemins prolongés par 4, 6 et 8 sont donc enfilés. Sur la branche `[3, 9, 1, 2, 4, 8]`, les voisins de 8 sont 1, 2 et 4, tous déjà présents : le chemin est alors ajouté à `chemins_np`. L'algorithme termine pour un jeu fini, car aucun chemin ne peut contenir plus de sommets que le jeu n'en contient.

## Question 13.

Sujet p. 15

> Écrire une fonction valeurs_chemin qui prend en paramètre chemin, une liste de sommets, et renvoie la liste de leurs valeurs.

Chaque sommet du chemin fournit sa valeur par l'attribut `valeur`. Une compréhension de liste les rassemble dans le même ordre que les sommets.

**RÉPONSE À RETENIR**

```
def valeurs_chemin(chemin):
    return [sommet.valeur for sommet in chemin]
```

Exemple : pour les sommets représentant 4, 8 et 2, la fonction renvoie `[4, 8, 2]`.

La question 14 utilise le nom `chemin_valeur` pour la fonction de conversion. On peut rendre ce nom disponible par l'alias suivant :

```
chemin_valeur = valeurs_chemin
```

## Question 14.

Sujet p. 15, 16

> Recopier et compléter, à l'aide des fonctions rechercher_chemins et chemin_valeur, les lignes de code qui permettent d'obtenir tous les chemins non prolongeables du jeu à 9 entiers représentés par la liste des valeurs de leurs sommets.

On obtient d'abord les chemins non prolongeables sous forme de listes de sommets. On convertit ensuite chacun d'eux en liste de valeurs à l'aide de `chemin_valeur`, nom défini par l'alias proposé à la question précédente.

**RÉPONSE À RETENIR**

```
chemins = []
for chemin in rechercher_chemins(jeu_9):
    chemins.append(chemin_valeur(chemin))
```

Trace : si la recherche fournit le chemin de sommets représentant 4 → 8 → 2 → 6 → 1 → 7, la conversion ajoute `[4, 8, 2, 6, 1, 7]` à `chemins`. Cette conversion est effectuée pour chaque chemin non prolongeable.

## Question 15.

Sujet p. 16

> Écrire le code de la fonction extraire_plus_longs_chemins qui prend en paramètre une liste de chemins L_chemins et qui renvoie la liste des chemins de longueur maximale.

On détermine d'abord la longueur maximale des chemins, puis on sélectionne tous ceux dont la longueur est égale à ce maximum. Il faut conserver toutes les égalités, car plusieurs chemins peuvent être les plus longs. Le cas d'une liste vide est traité séparément.

**RÉPONSE À RETENIR**

```
def extraire_plus_longs_chemins(L_chemins):
    if L_chemins == []:
        return []

    longueur_max = len(L_chemins[0])
    for chemin in L_chemins:
        if len(chemin) > longueur_max:
            longueur_max = len(chemin)

    return [chemin for chemin in L_chemins
            if len(chemin) == longueur_max]
```

Trace sur `[[1, 7], [3, 1, 5], [4, 1, 7]]` : les longueurs sont 2, 3 et 3, donc la fonction renvoie `[[3, 1, 5], [4, 1, 7]]`. Pour l'entrée `[[1, 3, 6, 2, 4, 8]]`, elle renvoie `[[1, 3, 6, 2, 4, 8]]`, c'est-à-dire une liste contenant le chemin maximal.

## Question 16.

Sujet p. 16

> Dire si l'on peut, à l'aide de notre programme, résoudre le jeu pour un nombre de sommets aussi grand que l'on souhaite. Justifier.

Le programme énumère les chemins possibles avant d'en extraire les chemins non prolongeables. Le nombre de séquences sans répétition peut croître très rapidement avec le nombre de sommets, ce qui augmente le temps de calcul et la mémoire nécessaires. Les données du sujet illustrent cette croissance : 377 chemins pour 9 sommets et 9 896 pour 14 sommets.

Pour chaque jeu fini, la recherche termine puisque les chemins ne peuvent pas répéter de sommet. Toutefois, cette terminaison ne garantit pas que le calcul reste réalisable pour des tailles arbitrairement grandes.

**RÉPONSE À RETENIR**

**Non.** Le programme termine pour chaque jeu fini, mais l'énumération exhaustive fait croître rapidement le nombre de chemins à traiter ; le temps de calcul et la mémoire deviennent donc limitants lorsque le nombre de sommets augmente suffisamment.

---

Sujet : https://www.sujetdebac.fr/annales-pdf/2025/spe-numerique-informatique-2025-centres-etranger-2-sujet-officiel.pdf
© FormaV EI • Proposition de correction, distincte d'un corrigé officiel.

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.