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 2023 • Jour 2

Code sujet **23-NSIJ2NC1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Base de données relationnelle et requêtes SQL
8 réponses • sujet p. 2–3

Exercice 2 — Palindromes, récursivité et jeu Motus en Python
6 réponses • sujet p. 4–5

Exercice 3 — Concurrence des processus, exclusion mutuelle et ordonnancement
5 réponses • sujet p. 6–8

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

4 points

Base de données relationnelle et requêtes SQL

1

Question 1.a

Sujet p. 2

> Dans le schéma Benevole (idbenevole, nom, prenom, rue, code_postal, ville, telephone, #idmission) et Mission (idmission, designation), les clés primaires sont soulignées et les clés étrangères sont précédées de #. idbenevole est déclaré comme clé primaire de la table Benevole. Préciser son rôle.

Une clé primaire identifie de manière unique chaque ligne d'une table. Sa valeur ne peut pas être dupliquée ; elle ne peut pas être absente. Ici, `idbenevole` distingue chaque bénévole, même si plusieurs ont le même nom.

> **RÉPONSE À RETENIR**
>
> `idbenevole` identifie de façon unique chaque ligne de la table `Benevole`.

Question 1.b

Sujet p. 2

> Dans le schéma Benevole (idbenevole, nom, prenom, rue, code_postal, ville, telephone, #idmission) et Mission (idmission, designation), idmission est déclaré comme clé étrangère de la table Benevole. Préciser son rôle.

Une clé étrangère établit un lien avec une autre table et contribue à garantir la cohérence de ce lien. Ici, la valeur de `Benevole.idmission` doit correspondre à une valeur existant dans la clé primaire `Mission.idmission`. Elle indique donc la mission associée au bénévole.

> **RÉPONSE À RETENIR**
>
> `idmission` relie un bénévole à une mission de la table `Mission` et empêche de référencer une mission inexistante, sous réserve des contraintes d'intégrité définies dans la base.

2

Question 2)

Sujet p. 2

> Le secrétaire exécute la requête INSERT INTO Benevole (nom, prenom, rue, code_postal, ville, telephone, idmission) VALUES avec les deux enregistrements ('POHU', 'Dominique', '12 rue de la Frairie', '49000', 'Angers', '+336123456', 1) et ('MOISY', 'Camille', '33 rue Maurice Bot', '49400', 'Cholet', '+337987654', 2). Expliquer l'objectif de cette requête.

La commande `INSERT INTO` ajoute des lignes à la table indiquée. Pour chaque ligne, les valeurs fournies correspondent aux colonnes nommées, dans le même ordre. Les deux lignes de valeurs sont séparées par une virgule.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Cette requête a pour objectif d'ajouter à la table `Benevole` deux bénévoles avec les coordonnées indiquées : Dominique POHU, associé à la mission d'identifiant 1, et Camille MOISY, associé à la mission d'identifiant 2.

3

Question 3.a

Sujet p. 2

> La table Collecte (idcollecte, #idbenevole, annee_ventes, montant) répertorie chaque année les ventes réalisées. idbenevole fait référence à Benevole ; annee_ventes indique l'année de la collecte et montant sa valeur en euros, sous la forme d'un nombre entier. Écrire une requête SQL permettant d'afficher l'extrait de la table Collecte avec les ventes réalisées en 2020.

On sélectionne les lignes de `Collecte` dont l'année des ventes vaut 2020. Comme l'énoncé demande l'extrait de la table, `*` permet d'afficher toutes ses colonnes.

```
SELECT *
FROM Collecte
WHERE annee_ventes = 2020;
```

Aucun extrait de données n'est fourni dans le sujet : les lignes retournées ne peuvent donc pas être précisées.

RÉPONSE À RETENIR

La requête affiche toutes les collectes enregistrées pour l'année 2020.

Question 3.b

Sujet p. 2

> Écrire une requête SQL permettant d'afficher les noms et prénoms des bénévoles ayant apporté plus de 50 € de dons en 2020.

Les noms et prénoms sont dans `Benevole`, tandis que l'année et le montant sont dans `Collecte`. Il faut relier les tables par `idbenevole`, puis filtrer sur l'année 2020 et sur un montant strictement supérieur à 50. `DISTINCT` évite de répéter un bénévole s'il a plusieurs collectes correspondant aux critères.

```
SELECT DISTINCT Benevole.nom, Benevole.prenom
FROM Benevole
JOIN Collecte ON Benevole.idbenevole = Collecte.idbenevole
WHERE Collecte.annee_ventes = 2020
  AND Collecte.montant > 50;
```

Aucun extrait de données n'est fourni : les lignes du résultat ne peuvent pas être déterminées.

RÉPONSE À RETENIR

La requête renvoie les noms et prénoms des bénévoles ayant une collecte de 2020 dont le montant dépasse 50 €.

Question 3.c

Sujet p. 2

> Écrire une requête SQL permettant de modifier une erreur de saisie sur le montant des ventes en 2020 pour la personne dont idbenevole est égal à 1 : cette personne a collecté 300 €, alors que le montant apparaissant dans la base est de 3 €.

La commande `UPDATE` modifie les lignes qui satisfont la condition `WHERE`. Il faut limiter la modification au bénévole d'identifiant 1, à l'année 2020, et à l'ancien montant erroné de 3 €.

```
UPDATE Collecte
SET montant = 300
WHERE idbenevole = 1
  AND annee_ventes = 2020
  AND montant = 3;
```

> **RÉPONSE À RETENIR**
>
> La collecte concernée est corrigée : son montant passe de 3 € à 300 €.

4

Question 4.a

Sujet p. 3

> La table Presence (idpresence, jour, nb_heures) permet de suivre l'activité des bénévoles. Le secrétaire exécute INSERT INTO Presence(idpresence, jour, nb_heures) VALUES ('11-02-2020', 2) ; pour insérer un premier enregistrement. Expliquer pourquoi cette requête échoue.

La requête nomme trois colonnes (`idpresence`, `jour` et `nb_heures`), mais ne fournit que deux valeurs. Le nombre de valeurs doit correspondre au nombre de colonnes indiquées.

Si `idpresence` est généré automatiquement par le SGBDR, on peut omettre cette colonne :

```
INSERT INTO Presence (jour, nb_heures)
VALUES ('11-02-2020', 2);
```

Si cet identifiant n'est pas généré automatiquement, il faut fournir une troisième valeur. Par exemple, sous l'hypothèse que `idpresence` est un entier et que l'identifiant 1 est disponible :

```
INSERT INTO Presence (idpresence, jour, nb_heures)
VALUES (1, '11-02-2020', 2);
```

> **RÉPONSE À RETENIR**
>
> La requête échoue parce qu'elle indique trois colonnes mais seulement deux valeurs. Il faut soit omettre `idpresence` s'il est généré automatiquement, soit fournir une valeur d'identifiant disponible.

FormaV - document d'entraînement non officiel

Question 4.b

Sujet p. 3

> Après correction de l'erreur précédente, le secrétaire exécute SELECT nom FROM Presence WHERE nb_heures > 2 ; pour obtenir la liste des noms des bénévoles présents plus de 2 heures sur une journée. Expliquer pourquoi cette requête échoue et proposer une solution de correction.

La requête échoue car la table `Presence` ne contient pas de colonne `nom` : son schéma comporte seulement `idpresence`, `jour` et `nb_heures`. De plus, aucune colonne de ce schéma ne relie une présence à un bénévole. On ne peut donc pas retrouver les noms des bénévoles à partir des tables et attributs fournis.

Pour pouvoir répondre à la demande, on peut ajouter à `Presence` un attribut `idbenevole`, clé étrangère vers `Benevole.idbenevole`. Une fois ce lien renseigné, les tables peuvent être jointes ainsi :

```
SELECT DISTINCT Benevole.nom
FROM Benevole
JOIN Presence ON Benevole.idbenevole = Presence.idbenevole
WHERE Presence.nb_heures > 2;
```

Aucun extrait de données n'est donné pour calculer les lignes du résultat.

> **RÉPONSE À RETENIR**
>
> La requête initiale échoue parce que `nom` n'est pas une colonne de `Presence`. Pour afficher les noms, il faut disposer d'un lien entre une présence et un bénévole, par exemple une colonne `idbenevole` dans `Presence`, puis effectuer la jointure indiquée.

Exercice 2

4 points

Palindromes, récursivité et jeu Motus en Python

1

Question 1.a

Sujet p. 4

> Pour la fonction palindrome fournie, qui teste si un mot s'écrit de la même façon à l'endroit et à l'envers, écrire les spécifications sous la forme d'une chaîne de documentation (docstring). Préciser l'entrée et son type, la sortie et son type, ainsi que le rôle de la fonction.

La docstring précise le paramètre, le type de la valeur renvoyée et le rôle de la fonction. Ici, la fonction compare les caractères situés symétriquement dans le mot et indique si le mot se lit de la même façon à l'endroit et à l'envers.

RÉPONSE À RETENIR

```
def palindrome(mot):
    """Teste si un mot est un palindrome.

    Paramètre : mot (str), le mot à tester.
    Retour : bool, True si mot est un palindrome, False sinon.
    """
```

Question 1.b

Sujet p. 4

> Indiquer sans justification la complexité en temps de calcul de l'algorithme palindrome pour un mot de longueur n. Ne compter que les comparaisons de caractères.

La boucle parcourt les valeurs de `k` de 0 à `n//2 - 1`. À chaque itération, elle compare une paire de caractères. Le nombre de comparaisons est donc exactement `n//2`, y compris si une différence est trouvée, car la boucle ne s'interrompt pas à ce moment-là.

RÉPONSE À RETENIR

La complexité en temps est linéaire, soit Θ(n). Plus précisément, l'algorithme effectue exactement `n//2` comparaisons de caractères.

Question 1.c

Sujet p. 4

> Indiquer ce qu'est une fonction récursive.

Une fonction récursive est une fonction qui s'appelle elle-même, directement ou indirectement. Pour que le calcul s'arrête, elle comporte un cas de base qui ne déclenche pas de nouvel appel récursif.

RÉPONSE À RETENIR

Une fonction récursive s'appelle elle-même, directement ou indirectement, et comporte un cas de base qui permet d'arrêter les appels.

Question 1.d

Sujet p. 4

> Écrire et compléter, sur la copie, les lignes 2, 4 et 5 de la fonction récursive palindrome_recursive, qui prend en paramètre un mot et teste si ce mot est un palindrome. Le squelette fourni comporte notamment une condition de base à la ligne 2, return True à la ligne 3, une comparaison des caractères aux lignes 4 et 5, puis l'appel palindrome_recursive(mot[1:len(mot)-1]) à la ligne 6.

Un mot de longueur 0 ou 1 est nécessairement un palindrome : c'est le cas de base. Pour un mot plus long, on compare ses deux caractères extrêmes. S'ils sont différents, le mot n'est pas un palindrome ; s'ils sont identiques, on recommence sur le mot privé de ses deux extrémités.

RÉPONSE À RETENIR

```
def palindrome_recursive(mot):
    if len(mot) <= 1:
        return True
    if mot[0] != mot[len(mot) - 1]:
        return False
    return palindrome_recursive(mot[1:len(mot) - 1])
```

Les compléments sont : ligne 2, `1` ; ligne 4, `len(mot) - 1` ; ligne 5, `False`.

2

Question 2.a

Sujet p. 4, 5

> Écrire la fonction test_mot, qui prend en paramètres le mot à deviner en majuscules et un mot proposé en majuscules par le joueur. Cette fonction renvoie le mot formé des lettres correctement placées et de tirets ailleurs. On suppose que le mot proposé a le bon nombre de lettres. Par exemple, test_mot('MOMIE','MINCE') renvoie 'M---E'.

On parcourt les positions des deux mots, qui ont le même nombre de lettres selon l'énoncé. À chaque position, on conserve la lettre si elle est identique dans le mot mystère et la proposition ; sinon, on ajoute un tiret. On assemble ensuite les caractères obtenus en une chaîne.

```
def test_mot(mot, proposition):
    resultat = []
    for i in range(len(mot)):
        if mot[i] == proposition[i]:
            resultat.append(mot[i])
        else:
            resultat.append('-')
    return ''.join(resultat)
```

Pour `mot = 'MOMIE'` et `proposition = 'MINCE'`, les lettres aux indices 0 et 4 sont bien placées ; les trois positions intermédiaires sont remplacées par des tirets.

RÉPONSE À RETENIR

`test_mot('MOMIE', 'MINCE')` renvoie `'M---E'`.

Question 2.b

Sujet p. 5

> Le programme jeu_motus prend en paramètre le mot à deviner. Il doit afficher sa première lettre suivie de tirets, inviter le candidat à saisir une proposition avec « donner votre proposition : », afficher les lettres bien placées et des tirets ailleurs, puis recommencer jusqu'à la victoire ou jusqu'au nombre maximal d'essais autorisés, égal au nombre de lettres du mot. Il affiche « Félicitations » si le candidat a gagné et donne le mot mystère s'il a perdu. En vous appuyant sur l'exemple d'affichage et le programme fournis, expliquer pourquoi ce programme ne donne pas toujours le résultat attendu.

Le compteur `nb_coups` est initialisé à 0, mais il n'est jamais modifié dans le programme fourni. Ainsi, après une proposition incorrecte, `nb_coups < n` reste vrai et `gagne` reste faux : la boucle ne s'arrête pas, même après plus de `n` propositions. Si le joueur gagne, le message annonce aussi 0 coup, puisque le compteur n'a pas été incrémenté.

Pour corriger ces défauts, il faut incrémenter le compteur après chaque proposition saisie. Dans le code ci-dessous, chaque proposition compte pour un essai, y compris la proposition gagnante. On suppose, comme dans l'exercice, que le mot saisi a le bon nombre de lettres et que le mot à deviner n'est pas vide.

```
def jeu_motus(mot):
    n = len(mot)
    nb_coups = 0
    gagne = False
    print(mot[0] + '-' * (n - 1))
    while nb_coups < n and not gagne:
        mot_joueur = input('donner votre proposition : ')
        nb_coups += 1
        if mot_joueur == mot:
            gagne = True
        else:
            print(test_mot(mot, mot_joueur))
    if gagne:
        print('Félicitations, vous avez gagné en', nb_coups, 'coups.')
    else:
        print('perdu, il fallait trouver :', mot)
```

Dans l'exemple, les trois propositions incorrectes `MATIN`, `MATHS` et `MOTOS` portent le compteur à 3. La proposition gagnante `MOTUS` le porte à 4, puis la boucle se termine.

RÉPONSE À RETENIR

L'absence d'incrémentation de `nb_coups` empêche la limite de `n` essais de fonctionner : après des propositions incorrectes, la boucle peut continuer indéfiniment. Le nombre de coups affiché en cas de victoire est également faux. Incrémenter le compteur à chaque proposition saisie corrige ces deux problèmes.

Exercice 3

4 points

Concurrence des processus, exclusion mutuelle et ordonnancement

1

Question 1.a

Sujet p. 6

> Deux guichets utilisent une variable compteur partagée, initialement égale à 100 en base 10, qui diminue de 1 à chaque billet délivré. Suivre l'ordonnancement indiqué dans le tableau : à l'intervalle 1, P1 et P2 lisent compteur dans leurs variables locales compteur1 et compteur2 ; à l'intervalle 2, P2 décrémente compteur2 et écrit sa valeur dans compteur ; à l'intervalle 3, P1 décrémente compteur1 et écrit sa valeur dans compteur. Montrer que la valeur de compteur après l'intervalle de temps 3 est égale à 99.

Au départ, `compteur` vaut 100. À l'intervalle 1, les deux processus lisent cette valeur : `compteur1` et `compteur2` valent donc chacun 100. À l'intervalle 2, P2 décrémente sa copie à 99 et écrit 99 dans `compteur`. À l'intervalle 3, P1 décrémente sa propre copie, qui vaut toujours 100, à 99 et écrit à son tour 99 dans `compteur`. Cette dernière écriture ne tient pas compte de la mise à jour de P2 : celle-ci est perdue.

> **RÉPONSE À RETENIR**
>
> Après l'intervalle de temps 3, `compteur` vaut 99.

Question 1.b

Sujet p. 6

> Modifier l'ordonnancement des processus pour qu'à la fin la valeur de la variable compteur soit égale à 98.

Pour obtenir deux décrémentations effectives, il faut que le second processus lise la valeur écrite par le premier. On peut exécuter les opérations de P1 l'une après l'autre, puis celles de P2 :

1. P1 lit 100, décrémente sa copie à 99 et écrit 99 dans `compteur`.
2. P2 lit alors 99, décrémente sa copie à 98 et écrit 98 dans `compteur`.

On pourrait aussi exécuter P2 entièrement avant P1.

> **RÉPONSE À RETENIR**
>
> Un ordonnancement possible est : lecture, décrémentation et écriture par P1, puis lecture, décrémentation et écriture par P2. La valeur finale de `compteur` est 98.

2

Question 2.a

Sujet p. 6, 7

> La mémoire partagée est gérée par un jeton mutex d'exclusion mutuelle. Chaque processus verrouille mutex, lit compteur dans sa variable locale, décrémente cette variable, écrit sa valeur dans compteur, puis déverrouille mutex. Justifier que cette solution permet à la variable compteur de comptabiliser correctement le nombre de places restantes.

Le verrouillage du mutex garantit qu'un seul processus à la fois accède à la mémoire partagée dans la séquence critique : lecture de `compteur`, décrémentation de sa copie locale et écriture dans `compteur`. L'autre processus doit attendre la libération du mutex. Ainsi, il lit la valeur actualisée par le premier

FormaV - document d'entraînement non officiel

processus, au lieu d'utiliser une ancienne copie.

Par exemple, si le compteur vaut initialement 100, le premier processus qui entre dans la section critique écrit 99. Le suivant lit alors 99 et écrit 98. Cette conclusion suppose que les deux processus utilisent bien le même mutex et que ses opérations de verrouillage et de déverrouillage assurent l'exclusion mutuelle.

> **RÉPONSE À RETENIR**
>
> Les deux billets délivrés entraînent successivement le passage de 100 à 99, puis de 99 à 98. Le compteur comptabilise donc correctement le nombre de places restantes.

Question 2.b

Sujet p. 7

> Proposer un algorithme identique pour les processus P1 et P2 produisant le même résultat.

On peut définir un algorithme commun qui reçoit la variable locale propre au processus. P1 l'utilise avec `compteur1` et P2 avec `compteur2`. Le mutex est partagé par les deux processus.

```
Algorithme traiter_billet(variable_locale) :
    mutex.verrouillage()
    variable_locale ← compteur
    variable_locale ← variable_locale − 1
    compteur ← variable_locale
    mutex.deverrouillage()

Processus P1 : traiter_billet(compteur1)
Processus P2 : traiter_billet(compteur2)
```

Le verrouillage a lieu avant la lecture et le déverrouillage après l'écriture : toute la mise à jour du compteur est ainsi protégée.

> **RÉPONSE À RETENIR**
>
> Les deux processus appliquent le même algorithme : verrouiller, lire, décrémenter, écrire, puis déverrouiller. Seule la variable locale passée à l'algorithme diffère.

3

Question 3)

Sujet p. 7, 8

> Quatre processus P1, P2, P3 et P4 exécutent chacun l'algorithme de la question 2.b. L'ordonnancement est un ordonnancement avec priorité, sans nouveau calcul de la priorité. Le quantum est de 15 ms et insécable ; une priorité de 0 définit une priorité haute et une priorité de 20 une priorité basse. Les données sont : P1, priorité 5, arrivée sur le processeur à 30 ms, durée de service 30 ms ; P2, priorité 5, arrivée à 15 ms, durée de service 30 ms ; P3, priorité 8, arrivée à 15 ms, durée de service 30 ms ; P4, priorité 20, arrivée à 15 ms, durée de service 30 ms. Compléter le diagramme sur le document réponse, à l'image de la figure 1 de la question 1.

Le quantum dure 15 ms et est insécable. Aucun processus n'étant arrivé avant 15 ms, le processeur reste inactif de 0 à 15 ms. À 15 ms, P2, P3 et P4 sont disponibles ; P2 a la priorité la plus haute, puisque 5 est inférieur à 8 et à 20. Il exécute donc un premier quantum de 15 à 30 ms. P1 arrive à 30 ms et a la même priorité que P2.

FormaV - document d'entraînement non officiel

L’énoncé ne précise pas comment départager deux processus de même priorité. En adoptant la convention FIFO pour les priorités égales, P2 passe avant P1 : il achève son service de 30 à 45 ms, puis P1 exécute ses deux quanta de 45 à 75 ms. P3 exécute ensuite ses deux quanta de 75 à 105 ms, puis P4 les siens de 105 à 135 ms. Les quatre durées de service sont ainsi de 30 ms. Le processeur reste inactif de 135 à 180 ms, fin de l’axe du document réponse.

Intervalle (ms)	**0–15**	**15–30**	**30–45**	**45–60**	**60–75**	**75–90**	**90–105**	**105–120**	**120–135**	**135–150**	**150–165**	**165–180**
Processeur	Inactif	P2	P2	P1	P1	P3	P3	P4	P4	Inactif	Inactif	Inactif

Si, à 30 ms, le départage entre P1 et P2 se fait en faveur de P1, un autre ordonnancement valable est : P2 de 15 à 30 ms, P1 de 30 à 60 ms, P2 de 60 à 75 ms, P3 de 75 à 105 ms et P4 de 105 à 135 ms ; le processeur reste ensuite inactif jusqu’à 180 ms.

Ordonnancement (départage FIFO entre priorités égales)

> **0–15 ms** Processeur inactif : aucun processus n’est encore arrivé.
>
> **15–30 ms** P2 — priorité 5, premier quantum.
>
> **30–45 ms** P2 — priorité 5, second quantum ; départage FIFO choisi face à P1.
>
> **45–60 ms** P1 — priorité 5, premier quantum.
>
> **60–75 ms** P1 — priorité 5, second quantum.
>
> **75–90 ms** P3 — priorité 8, premier quantum.
>
> **90–105 ms** P3 — priorité 8, second quantum.
>
> **105–120 ms** P4 — priorité 20, premier quantum.
>
> **120–135 ms** P4 — priorité 20, second quantum.
>
> **135–180 ms** Processeur inactif : les quatre processus ont terminé leur service.

Chaque quantum dure 15 ms. Les arrivées sont P2, P3 et P4 à 15 ms, puis P1 à 30 ms. Le départage FIFO entre P1 et P2 est une convention, car l’énoncé ne fixe pas le traitement des égalités de priorité.

RÉPONSE À RETENIR

Avec un départage FIFO entre les priorités égales, l’ordonnancement est : processeur inactif de 0 à 15 ms, P2 de 15 à 45 ms, P1 de 45 à 75 ms, P3 de 75 à 105 ms, P4 de 105 à 135 ms, puis processeur inactif de 135 à 180 ms. L’ordre entre P1 et P2 dépend du départage choisi, qui n’est pas fixé dans l’énoncé.

Sujet : https://www.sujetdebac.fr/annales-pdf/2023/spe-numerique-informatique-2023-nouv-caledonie-2-sujet-officiel.pdf
© FormaV EI • Proposition de correction, distincte d'un corrigé officiel.

FormaV - document d'entraînement non 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.