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 • Asie • Jour 1 • Remplacement

Code sujet **25-NSIPE4** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

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

Exercice 2 — Réseaux, routage, graphes et programmation
12 réponses • sujet p. 5–9

Exercice 3 — Algorithmique des tableaux, gestion de bugs, listes, piles et programmation orientée objet — tri dictatorial
13 réponses • sujet p. 10–14

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

> Expliquer pourquoi l'attribut nom ne peut pas être choisi comme clé primaire.

Une clé primaire doit identifier chaque ligne de manière unique. Or la valeur « Benard » apparaît deux fois dans la colonne `nom` : elle ne permet donc pas de distinguer Elsa de Emma.

RÉPONSE À RETENIR

`nom` n'est pas unique dans la table ; il ne peut donc pas servir de clé primaire.

Question 2.

Sujet p. 2

> Écrire une requête permettant d'obtenir les noms et prénoms des joueuses du club.

Le champ `genre` vaut 2 pour une joueuse. On sélectionne les colonnes demandées dans `joueurs` et on filtre les lignes sur cette valeur.

```
SELECT nom, prenom
FROM joueurs
WHERE genre = 2;
```

RÉPONSE À RETENIR

nom	prenom
Panais	Lise
Benard	Elsa
Benard	Emma

Question 3.

Sujet p. 2

> Écrire une requête permettant d'ajouter dans la table le joueur dont le prénom est Nathan et le nom est Gervais, en choisissant une valeur pour l'identifiant id cohérente avec le reste de la base.

Les identifiants présents dans l'extrait vont de 1 à 5. L'identifiant 6 est donc cohérent avec les données fournies et ne duplique aucune clé primaire. On indique explicitement les colonnes afin de faire correspondre chaque valeur au bon attribut.

```
INSERT INTO joueurs (id, nom, prenom, genre)
VALUES (6, 'Gervais', 'Nathan', 1);
```

RÉPONSE À RETENIR

La ligne insérée est `(6, 'Gervais', 'Nathan', 1)` ; `genre = 1` car Nathan est un joueur.

FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 3

> Une faute de frappe s'est glissée dans le nom de la compétition d'identifiant 3. Écrire une requête permettant de corriger le nom en Open de Tours.

On modifie la ligne de clé primaire `id = 3`. La condition `WHERE` limite la mise à jour à cette compétition.

```
UPDATE competitions
SET nom = 'Open de Tours'
WHERE id = 3;
```

RÉPONSE À RETENIR

La compétition d'identifiant 3 porte désormais le nom « Open de Tours ».

Question 5.

Sujet p. 3

> Écrire une requête permettant d'obtenir la liste des noms des tournois, ainsi que leur année, en triant par année croissante.

On sélectionne le nom et l'année dans `competitions`, puis on trie les lignes par année croissante avec `ORDER BY`. Comme aucun critère secondaire n'est demandé, l'ordre des compétitions ayant la même année n'est pas imposé.

```
SELECT nom, annee
FROM competitions
ORDER BY annee ASC;
```

Le résultat ci-dessous tient compte de la correction demandée à la question 4 : le nom de la compétition d'identifiant 3 est donc « Open de Tours ».

RÉPONSE À RETENIR

nom	annee
Open de Nantes	2021
Open de Tours	2022
Tournoi de Blois	2023
Open de Tours	2023
Open de Nantes	2023
Tournoi d'Angers	2024

Question 6.

Sujet p. 2, 3, 4

> Écrire une requête permettant d'obtenir le nom et le prénom des joueurs et des joueuses ayant obtenu un nombre de coups gagnants strictement supérieur au nombre de fautes lors d'une compétition, la même personne pouvant apparaître plusieurs fois si elle a rempli ces conditions lors de plusieurs compétitions.

On relie chaque participation à la joueuse ou au joueur correspondant avec `id_joueur = id`, puis on conserve les lignes où le nombre de coups gagnants est strictement supérieur au nombre de fautes. On ne met pas `DISTINCT`, car une personne doit pouvoir apparaître plusieurs fois si plusieurs participations satisfont la condition.

```
SELECT joueurs.nom, joueurs.prenom
FROM joueurs
JOIN participe ON joueurs.id = participe.id_joueur
WHERE participe.nb_gagnant > participe.nb_fautes;
```

Dans l'extrait, les comparaisons sont satisfaites pour les couples (joueur 1, compétition 3), (joueur 2, compétition 4), (joueur 4, compétition 4) et (joueur 5, compétition 4). Les autres participations ne vérifient pas l'inégalité stricte.

RÉPONSE À RETENIR

nom	prenom
Durand	Enzo
Panais	Lise
Benard	Elsa
Benard	Emma

Question 7.

Sujet p. 2, 3, 4

> Écrire une requête permettant d'obtenir le nom, le prénom des joueuses et le nom des compétitions auxquelles elles ont participé en 2023.

On relie `joueurs` à `participe` par l'identifiant du joueur, puis `participe` à `competitions` par l'identifiant de la compétition. Le filtre `genre = 2` retient les joueuses et `annee = 2023` les compétitions de l'année demandée.

```
SELECT joueurs.nom, joueurs.prenom, competitions.nom
FROM joueurs
JOIN participe ON joueurs.id = participe.id_joueur
JOIN competitions ON participe.id_compet = competitions.id
WHERE joueurs.genre = 2 AND competitions.annee = 2023;
```

Les participations retenues sont celles de Lise Panais aux compétitions 2 et 4, ainsi que celles d'Elsa et d'Emma Benard à la compétition 4.

RÉPONSE À RETENIR

nom	prenom	nom de la compétition
Panais	Lise	Tournoi de Blois
Panais	Lise	Open de Nantes
Benard	Elsa	Open de Nantes
Benard	Emma	Open de Nantes

Question 8.

Sujet p. 2, 3, 4

> On souhaite supprimer de la table joueurs la joueuse Emma Benard qui ne fait plus partie du club. Déterminer quelle précaution on doit prendre avant de pouvoir le faire. Justifier.

`participe.id_joueur` est une clé étrangère qui référence `joueurs.id`. Emma Benard a l'identifiant 5 et l'extrait comporte une participation qui la référence. Si la base applique normalement cette contrainte référentielle, la suppression de sa ligne dans `joueurs` est empêchée tant que cette participation existe. Il faut donc décider de traiter d'abord la ligne dépendante, ou de conserver la joueuse dans la base sous une forme archivée ou désactivée si ses statistiques doivent rester consultables.

RÉPONSE À RETENIR

Avant la suppression, il faut traiter la participation d'Emma (dans l'extrait : `id_joueur = 5`, `id_compet = 4`). On ne supprime cette ligne que si l'on accepte de supprimer également ces statistiques ; sinon, il faut privilégier une solution d'archivage ou de désactivation.

Question 9.

Sujet p. 2, 3, 4

> Une joueuse du club, de prénom Agathe et de nom Turion, a participé au Tournoi de Blois en 2024, où elle a fait 14 fautes directes, réalisé 15 coups gagnants et servi 2 aces. Écrire les différentes requêtes, dans le bon ordre, permettant d'insérer cette joueuse et ses résultats dans la base, en choisissant pour identifiant pour la table joueurs la valeur 7 et pour la table competitions la valeur 5.

La demande comporte une collision de clé primaire : dans la table `competitions` fournie, l'identifiant 5 est déjà utilisé par « Open de Nantes » en 2021. Or `competitions.id` est une clé primaire ; il est donc impossible d'insérer une nouvelle compétition avec `id = 5` sans modifier ou supprimer une donnée existante. Aucune telle modification n'est demandée ni justifiée par le sujet. Par conséquent, aucune séquence de requêtes ne peut satisfaire exactement la demande tout en conservant les données fournies.

À titre conditionnel, si l'identifiant de compétition indiqué dans la question était corrigé en 7, identifiant libre dans l'extrait, les insertions pourraient être faites ainsi. On crée la compétition avant la participation, afin que la clé étrangère `id_compet` puisse la référencer.

```
INSERT INTO joueurs (id, nom, prenom, genre)
VALUES (7, 'Turion', 'Agathe', 2);

INSERT INTO competitions (id, nom, annee)
VALUES (7, 'Tournoi de Blois', 2024);

INSERT INTO participe (id_joueur, id_compet, nb_fautes, nb_gagnant, aces)
VALUES (7, 7, 14, 15, 2);
```

Cette séquence conditionnelle conserve l'identifiant 7 demandé pour la joueuse, mais ne prétend pas respecter la valeur 5 demandée pour la compétition.

> **RÉPONSE À RETENIR**
>
> La requête demandée avec `id = 5` pour la nouvelle compétition ne peut pas être exécutée sans conflit avec la compétition déjà enregistrée sous cet identifiant. La demande est donc impossible à satisfaire telle quelle ; l'exemple avec l'identifiant 7 n'est valable que si la valeur imposée pour la compétition est corrigée.

Exercice 2

6 points

Réseaux, routage, graphes et programmation

A

Question 1.

Sujet p. 5, 6

> Parmi les deux adresses IP suivantes : 137.254.128.200 et 137.254.128.210, donner l'adresse IP de la machine déjà connectée au sous-réseau Commerces.

Avec le masque `/24`, les adresses utilisables pour les équipements vont de `137.254.128.1` à `137.254.128.254`. Les 207 machines sont numérotées en commençant par la plus petite adresse disponible. En supposant, comme le suggère cette numérotation, que l'adresse du routeur ne décale pas cette suite d'adresses attribuées aux machines, celles-ci utilisent les adresses de `137.254.128.1` à `137.254.128.207`. Ainsi, `137.254.128.200` est attribuée à une machine, mais `137.254.128.210` ne l'est pas.

> **RÉPONSE À RETENIR**
>
> L'adresse recherchée est **137.254.128.200**, sous l'hypothèse de numérotation indiquée ci-dessus.

Question 2.

Sujet p. 5, 6

> Préciser s'il est possible ou non d'ajouter 132 machines sur le sous-réseau Commerces, en justifiant la réponse.

Dans un sous-réseau `/24`, les adresses `137.254.128.0` (adresse du réseau) et `137.254.128.255` (adresse de diffusion) ne sont pas attribuables à des équipements. Il reste donc 254 adresses utilisables. Une est attribuée au routeur et 207 aux machines déjà présentes : il reste au plus $254 - 1 - 207 = 46$ adresses pour de nouvelles machines.

Comme $132 > 46$, la capacité restante est insuffisante.

> **RÉPONSE À RETENIR**
>
> **Non**, il n'est pas possible d'ajouter 132 machines sur ce sous-réseau.

B

Question 3.

Sujet p. 5, 6

> Donner la liste des routeurs par lesquels transite un message envoyé depuis une machine du sous-réseau Navigation à destination d'une machine du sous-réseau Commerces.

La machine de Navigation est reliée à R1 et celle de Commerces à R7. Le tableau donne, pour chaque routeur source, le prochain routeur à contacter pour atteindre R7 :

- depuis R1, le prochain routeur est R3 ;
- depuis R3, c'est R6 ;
- depuis R6, c'est R7.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Le message passe par **R1, R3, R6, puis R7**.

Question 4.

Sujet p. 5, 6

Décrire le problème rencontré lorsqu'une machine du sous-réseau Commerces envoie des données à destination d'une machine du sous-réseau Navigation.

La machine de Commerces est reliée à R7 et la machine de Navigation à R1. En suivant les tables pour une destination R1, R7 envoie le message à R4, R4 l'envoie à R6, puis R6 le renvoie à R4. Le message boucle donc entre R4 et R6 et n'atteint pas R1.

RÉPONSE À RETENIR

Le problème est une **boucle de routage entre R4 et R6**, qui empêche le message d'atteindre sa destination.

Question 5.

Sujet p. 6, 7

Donner le dictionnaire correspondant au réseau de la Figure 2, limité aux routeurs R1, R2, R3 et R5.

La Figure 2 représente un graphe non orienté. Les arêtes sont R1–R2, R1–R3, R2–R3 et R2–R5. Le dictionnaire doit donc donner chaque voisinage dans les deux sens.

RÉPONSE À RETENIR

Un dictionnaire correspondant au réseau est :

```
g = {
    'R1': ['R2', 'R3'],
    'R2': ['R1', 'R3', 'R5'],
    'R3': ['R1', 'R2'],
    'R5': ['R2']
}
```

Question 6.

Sujet p. 7

Rappeler le principe d'une fonction récursive.

Une fonction récursive est une fonction qui s'appelle elle-même pour résoudre un problème, généralement en le ramenant à un ou plusieurs problèmes plus simples. Elle doit comporter un **cas de base** qui arrête les appels récursifs ; sinon, ceux-ci se poursuivent jusqu'à une erreur de profondeur de pile.

RÉPONSE À RETENIR

Une fonction récursive s'appelle elle-même et possède un cas de base qui met fin à la récursion.

Question 7.

Sujet p. 7

Écrire une fonction plus_court_chemin(graphe, r_depart, r_arrivee) qui renvoie une liste représentant un des plus courts chemins entre les routeurs r_depart et r_arrivee en utilisant le protocole RIP. On utilisera la fonction

> liste_chemins définie à la question précédente.

La fonction `liste_chemins` fournit tous les chemins. Pour choisir un chemin le plus court au sens du protocole RIP, on retient un chemin contenant le moins de routeurs. Si aucun chemin n'existe, le code ci-dessous renvoie une liste vide.

```
def plus_court_chemin(graphe, r_depart, r_arrivee):
    chemins = liste_chemins(graphe, r_depart, r_arrivee)
    if chemins == []:
        return []
    return min(chemins, key=len)
```

Dans l'exemple du sujet, le chemin `['R1', 'R2', 'R5']` contient trois routeurs, tandis que `['R1', 'R3', 'R2', 'R5']` en contient quatre. Le premier est donc le plus court.

> **RÉPONSE À RETENIR**
>
> Pour l'exemple donné, un plus court chemin de R1 à R5 est `['R1', 'R2', 'R5']`.

Question 8.

Sujet p. 7, 8

> Compléter la fonction plus_court_chemin_largeur(graphe, r_depart, r_arrivee) suivante qui traduit l'idée de l'informaticien, réalisant un parcours en largeur et dans laquelle le dictionnaire dict_chemins associe à un routeur le chemin reliant r_depart à ce routeur.

On ajoute un sommet à la liste des sommets marqués dès sa découverte, puis on construit et mémorise le chemin qui y mène. Comme les sommets sont examinés par distance croissante dans le parcours en largeur, le premier chemin trouvé vers l'arrivée est de longueur minimale. Le cas où le départ est l'arrivée est traité directement ; si l'arrivée est inaccessible, la fonction renvoie `None`.

```
def plus_court_chemin_largeur(graphe, r_depart, r_arrivee):
    dict_chemins = {}
    L = [r_depart]
    sommets_marques = [r_depart]
    dict_chemins[r_depart] = [r_depart]

    if r_depart == r_arrivee:
        return [r_depart]

    for r in L:
        for s_r in graphe[r]:
            if not s_r in sommets_marques:
                sommets_marques.append(s_r)
                dict_chemins[s_r] = dict_chemins[r] + [s_r]
                if s_r == r_arrivee:
                    return dict_chemins[s_r]
                L.append(s_r)
    return None
```

Trace sur le graphe de la question 5, de R1 à R5 : le parcours découvre R2 et R3 depuis R1. En examinant R2, il découvre R5 et construit `['R1', 'R2', 'R5']`, qu'il renvoie. Chaque sommet est marqué une seule fois, ce qui garantit la terminaison sur un graphe fini. La représentation sous forme de listes de chemins entraîne des copies dont le coût peut atteindre $O(V^2)$ au total, où V est le nombre de sommets.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Sur l'exemple R1–R5, la fonction renvoie `['R1', 'R2', 'R5']`, qui est un chemin de longueur minimale.

Question 9.

Sujet p. 7, 8

> Écrire une fonction table_routage(graphe, routeur) qui renvoie la table de routage du routeur passé en paramètre sous la forme d'un dictionnaire associant à chaque routeur destination la passerelle correspondante. On pourra utiliser les fonctions écrites dans les questions précédentes.

Pour chaque destination distincte du routeur courant, on calcule un plus court chemin. Son deuxième élément est la passerelle : c'est le prochain routeur après le routeur courant. Si une destination est inaccessible, elle est omise.

```
def table_routage(graphe, routeur):
    table = {}
    for destination in graphe:
        if destination != routeur:
            chemin = plus_court_chemin_largeur(graphe, routeur, destination)
            if chemin is not None:
                table[destination] = chemin[1]
    return table
```

Sur le graphe de la question 5, pour le routeur R1, les chemins minimaux sont `['R1', 'R2']`, `['R1', 'R3']` et `['R1', 'R2', 'R5']` vers R2, R3 et R5 respectivement.

RÉPONSE À RETENIR

Pour `table_routage(g, 'R1')`, on obtient :

```
{'R2': 'R2', 'R3': 'R3', 'R5': 'R2'}
```

C

Question 10.

Sujet p. 8, 9

> Calculer le coût correspondant à chaque type de liaison.

On applique la formule $\text{coût} = 10^9/\text{débit}$, avec le débit exprimé en bits par seconde. Un mégabit par seconde vaut 10^6 bits par seconde.

- Ethernet : $10\ \text{Mbit/s} = 10^7\ \text{bit/s}$, donc $10^9/10^7 = 100$.
- Fast Ethernet : $100\ \text{Mbit/s} = 10^8\ \text{bit/s}$, donc $10^9/10^8 = 10$.
- Fibre : $500\ \text{Mbit/s} = 5 \times 10^8\ \text{bit/s}$, donc $10^9/(5 \times 10^8) = 2$.

RÉPONSE À RETENIR

Les coûts sont **100** pour Ethernet, **10** pour Fast Ethernet et **2** pour la fibre.

Question 11.

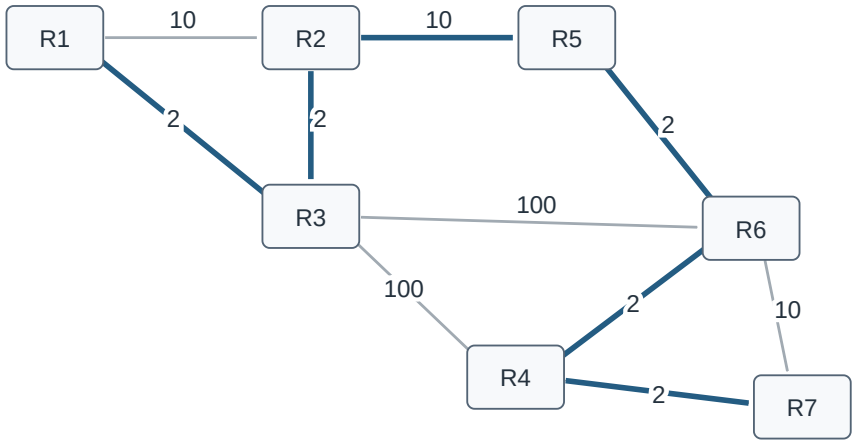
Sujet p. 8, 9

> Donner la liste des routeurs par lesquels transite un message envoyé depuis le routeur R1 à destination du routeur R7 en respectant le protocole OSPF.

Les coûts lus sur la Figure 3 sont : R1–R2 : 10 ; R1–R3 : 2 ; R2–R3 : 2 ; R2–R5 : 10 ; R3–R4 : 100 ; R3–R6 : 100 ; R4–R6 : 2 ; R4–R7 : 2 ; R5–R6 : 2 ; R6–R7 : 10.

Pour justifier le chemin minimal, on peut appliquer Dijkstra depuis R1. On obtient les distances minimales suivantes au fil des améliorations : depuis R1, R3 vaut 2 et R2 vaut d'abord 10 ; en passant par R3, la distance de R2 devient 4. Puis les distances de R5 et R6 deviennent respectivement 14 et 16, en passant par R2 puis R5. La distance de R4 devient 18 via R6, puis celle de R7 devient 20 via R4. Le chemin correspondant a pour coût $2 + 2 + 10 + 2 + 2 + 2 = 20$, inférieur aux autres possibilités évaluées par l'algorithme.

Liaisons pondérées et chemin OSPF de R1 à R7



Les poids reprennent les coûts de la Figure 3. Les arêtes mises en évidence forment le chemin OSPF minimal, de coût 20.

> **RÉPONSE À RETENIR**
>
> Le message suit le chemin **R1 → R3 → R2 → R5 → R6 → R4 → R7**, de coût total **20**.

Question 12.

Sujet p. 8, 9

> Recopier et compléter la table de routage du routeur R2 toujours en respectant le protocole OSPF.

Depuis R2, les coûts minimaux vers R1, R3 et R5 sont ceux des liaisons directes : 10, 2 et 10. Vers R6, le meilleur chemin est R2–R5–R6, de coût $10 + 2 = 12$. Vers R4, le meilleur chemin est R2–R5–R6–R4, de coût $10 + 2 + 2 = 14$. Vers R7, on poursuit par R4 : R2–R5–R6–R4–R7, de coût $10 + 2 + 2 + 2 = 16$. Dans chaque cas, le prochain routeur après R2 est indiqué dans la table. Pour la destination R2 elle-même, il n'y a pas de prochain routeur à contacter.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Destination	Suivant
R1	R1
R2	—
R3	R3
R4	R5
R5	R5
R6	R5
R7	R5

Le tiret pour R2 signifie que le routeur est lui-même la destination.

FormaV - document d'entraînement non officiel

Exercice 3

8 points

Algorithmique des tableaux, gestion de bugs, listes, piles et programmation orientée objet — tri dictatorial

A

Question 1.

Sujet p. 10, 11

> Donner le résultat que doit renvoyer l'appel : tri_dictatorial([31, 45, 41, 28, 37, 108, 127, 2, 124, 421]).

On parcourt la série de gauche à droite en comparant chaque élément au dernier élément conservé. Un élément strictement plus petit est éliminé ; un élément supérieur ou égal est conservé.

- 31 est conservé.
- 45 est supérieur à 31 : il est conservé.
- 41, 28 et 37 sont inférieurs à 45 : ils sont éliminés.
- 108 est supérieur à 45 : il est conservé.
- 127 est supérieur à 108 : il est conservé.
- 2 et 124 sont inférieurs à 127 : ils sont éliminés.
- 421 est supérieur à 127 : il est conservé.

> **RÉPONSE À RETENIR**
>
> L'appel renvoie `[31, 45, 108, 127, 421]`.

Question 2.

Sujet p. 10, 11

> Expliquer pourquoi le tri dictatorial n'est pas un algorithme de tri.

Un algorithme de tri réordonne les éléments d'une série : le résultat contient les mêmes éléments, avec les mêmes occurrences, rangés dans l'ordre demandé. Le tri dictatorial élimine les éléments plus petits que le dernier élément conservé. Il ne réordonne donc pas nécessairement tous les éléments de la série initiale.

> **RÉPONSE À RETENIR**
>
> Le tri dictatorial n'est pas un algorithme de tri, car il supprime certains éléments au lieu de simplement réordonner la série.

Question 3.

Sujet p. 10, 11

> Edgar réalise l'appel tri_dictatorial([8, 2, 9, 6, 12]). Expliquer pas à pas comment la liste serie_triee se construit après cet appel.

Le code d'Edgar initialise `serie_triee` à `[8]`. À chaque étape, il compare l'élément courant à l'élément précédent de la série initiale, et non au dernier élément conservé.

- `i = 1` : 2 ≥ 8 est faux ; la liste reste `[8]`.
- `i = 2` : 9 ≥ 2 est vrai ; 9 est ajouté, la liste devient `[8, 9]`.
- `i = 3` : 6 ≥ 9 est faux ; la liste reste `[8, 9]`.

FormaV - document d'entraînement non officiel

- `i = 4` : 12 ≥ 6 est vrai ; 12 est ajouté, la liste devient `[8, 9, 12]`.

> **RÉPONSE À RETENIR**
>
> La liste construite par le programme d'Edgar est `[8, 9, 12]`.

Question 4.

Sujet p. 10, 11

> Edgar réalise maintenant l'appel tri_dictatorial([]) et obtient l'erreur suivante : IndexError: list index out of range. Expliquer précisément l'erreur obtenue et proposer une modification du code d'Edgar afin que cet appel soit conforme à l'algorithme du tri dictatorial décrit en introduction.

Dans le code donné, `serie[0]` demande l'élément d'indice 0. Or une liste vide ne possède aucun élément à cet indice : Python déclenche donc `IndexError: list index out of range`. Il faut traiter les listes de taille 0 ou 1 avant cet accès. Pour les autres listes, il faut aussi comparer chaque élément au dernier élément conservé.

```
def tri_dictatorial(serie):
    if len(serie) <= 1:
        return serie.copy()
    serie_triee = [serie[0]]
    for i in range(1, len(serie)):
        if serie[i] >= serie_triee[-1]:
            serie_triee.append(serie[i])
    return serie_triee
```

> **RÉPONSE À RETENIR**
>
> Pour `tri_dictatorial([])`, la fonction renvoie une nouvelle liste vide, `[]`, sans erreur et sans modifier la liste reçue.

Question 5.

Sujet p. 11, 12

> Expliquer pourquoi des tests ne peuvent pas prouver de façon certaine l'absence de bugs d'un programme en général.

Un test vérifie le comportement du programme pour une entrée et une situation données. En général, le nombre d'entrées et de chemins d'exécution possibles est trop grand pour qu'on puisse tous les tester. Des tests réussis ne garantissent donc pas que les cas non testés, notamment certains cas limites, seront traités correctement.

> **RÉPONSE À RETENIR**
>
> Les tests peuvent révéler la présence de bugs, mais ne prouvent pas en général leur absence. Pour la prouver, il faudrait établir le bon fonctionnement du programme pour tous les cas pertinents.

Question 6.

Sujet p. 10, 11, 12

> Edgar réalise l'appel tri_dictatorial([8, 2, 3, 5, 12]). La fonction renvoie alors [8, 3, 5, 12], qui n'est pas une liste triée. Expliquer la cause du problème et proposer une modification du code d'Edgar afin de la corriger.

Le programme d'Edgar compare `serie[i]` à `serie[i - 1]`, c'est-à-dire à l'élément précédent de la série initiale. Or l'algorithme demande de comparer le candidat au dernier élément conservé. Ici, après avoir conservé 8, le programme élimine 2, mais compare ensuite 3 à 2 et le conserve. Pourtant, 3 est inférieur à 8 et doit être éliminé.

Il faut comparer au dernier élément de `serie_triee`. Voici une version qui traite également les listes vides et celles de taille 1 :

```
def tri_dictatorial(serie):
    if len(serie) <= 1:
        return serie.copy()
    serie_triee = [serie[0]]
    for i in range(1, len(serie)):
        if serie[i] >= serie_triee[-1]:
            serie_triee.append(serie[i])
    return serie_triee
```

Pour la série donnée, 8 est conservé ; 2, 3 et 5 sont éliminés car ils sont inférieurs au dernier élément conservé, 8 ; 12 est conservé.

RÉPONSE À RETENIR

La version corrigée renvoie `[8, 12]` pour `[8, 2, 3, 5, 12]`.

B

Question 7.

Sujet p. 12

> Donner des instructions permettant de construire les trois maillons m1, m0 et m8 et la liste chaînée représentés ci-dessus. On nommera la liste chaînée ma_liste.

On construit les maillons en commençant par le dernier, dont le suivant vaut `None`, puis on les relie dans l'ordre représenté : 1, puis 0, puis 8.

```
m8 = Maillon(8, None)
m0 = Maillon(0, m8)
m1 = Maillon(1, m0)
ma_liste = Liste(m1)
```

RÉPONSE À RETENIR

La liste obtenue est `ma_liste.tete` → 1 → 0 → 8 → `None`.

Question 8.

Sujet p. 12, 13

> Indiquer ce que renvoie chacune des instructions ci-dessous : m1.valeur == 1 ; m1.suivant.valeur == 8 ;

FormaV - document d'entraînement non officiel

> m1.suivant.suivant == None ; m1.suivant.suivant.suivant == None.

Dans la liste construite, `m1` contient 1, son suivant est `m0` (qui contient 0), puis vient `m8` (qui contient 8), dont le suivant vaut `None`.

> **RÉPONSE À RETENIR**
>
> - `m1.valeur == 1` renvoie `True`.
> - `m1.suivant.valeur == 8` renvoie `False`, car `m1.suivant` est `m0`, de valeur 0.
> - `m1.suivant.suivant == None` renvoie `False`, car cette expression désigne `m8`.
> - `m1.suivant.suivant.suivant == None` renvoie `True`, car le suivant de `m8` est `None`.

Question 9.

Sujet p. 12, 13

> Donner une instruction permettant de transformer ma_liste en la liste chaînée représentée ci-dessous.

La figure cible conserve la tête 1, mais son suivant doit désormais être 8. Il suffit de remplacer le lien de `m1`, qui pointait vers `m0`, par le suivant actuel de `m0`.

> **RÉPONSE À RETENIR**
>
> ```
> ma_liste.tete.suivant = ma_liste.tete.suivant.suivant
> ```
>
> La chaîne accessible depuis `ma_liste.tete` devient 1 → 8 → `None`. Le maillon 0 n'est plus accessible depuis la tête de la liste.

Question 10.

Sujet p. 10, 12, 13

> Recopier et compléter la fonction tri_dictatorial_chaine ci-dessous. La fonction prend en paramètre une instance de liste chaînée chaine, modifie cette liste chaînée en suivant l'algorithme du pseudo-tri dictatorial et ne renvoie rien.

On conserve le premier maillon. À chaque étape, on examine son suivant : si la valeur de ce suivant est supérieure ou égale à celle du maillon courant, on le conserve et on avance ; sinon, on le retire en le contournant. Après un retrait, on ne déplace pas le pointeur courant, car son nouveau suivant doit encore être examiné. Le test sur la tête traite aussi le cas d'une liste vide.

```
def tri_dictatorial_chaine(chaine):
    if chaine.tete is None:
        return

    maillon = chaine.tete
    while maillon.suivant is not None:
        if maillon.valeur <= maillon.suivant.valeur:
            maillon = maillon.suivant
        else:
            maillon.suivant = maillon.suivant.suivant
```

Par exemple, sur une chaîne 8 → 2 → 3 → 5 → 12, les maillons de valeur 2, 3 et 5 sont successivement retirés ; la chaîne devient 8 → 12 → `None`.

RÉPONSE À RETENIR

La fonction modifie les liens de la liste sur place et ne renvoie pas de résultat utile. La comparaison accepte l'égalité, conformément à la règle qui élimine seulement un élément strictement inférieur au dernier élément conservé.

C

Question 11.

Sujet p. 13

> Rappeler le principe du fonctionnement d'une pile.

Une pile est une structure de données où les ajouts et les retraits se font au même endroit, appelé le sommet. Elle suit la règle LIFO (« dernier entré, premier sorti ») : le dernier élément empilé est le premier dépilé. Les opérations usuelles sont empiler, dépiler et tester si la pile est vide.

RÉPONSE À RETENIR

Une pile est donc une structure LIFO, avec accès au seul élément situé au sommet.

Question 12.

Sujet p. 10, 13, 14

> Remettre dans l'ordre les lignes ci-dessous afin d'obtenir l'algorithme attendu, en respectant une tabulation lorsque la ligne est à l'intérieur d'un bloc si ou tant que.

On suppose que le sommet de `p` représente le premier élément de la série à traiter. Le premier dépilé est conservé et placé dans `p2`. Chaque candidat suivant est conservé dans `p2` s'il est supérieur ou égal au dernier élément conservé. La dernière boucle transfère les éléments de `p2` dans `p` : cette seconde inversion rétablit leur ordre initial. Il faut bien placer l'instruction de dépilement de `p2` dans cette dernière boucle.

RÉPONSE À RETENIR

```
si p n'est pas vide :
    on crée une pile intermédiaire p2 vide
    on dépile p, on stocke l'élément obtenu dans la variable dernier_conservé et on
l'empile dans p2
    tant que p n'est pas vide :
        on dépile p et on stocke l'élément obtenu dans la variable candidat
        si candidat est supérieure ou égal à dernier_conservé :
            dernier_conservé prend la valeur de candidat et on l'empile dans p2
    tant que p2 n'est pas vide :
        on dépile p2 et on empile l'élément obtenu dans p
```

Si `p` est vide au départ, elle reste vide.

Question 13.

Sujet p. 10, 13, 14

> Écrire en Python la fonction tri_dictatorial_pile qui prend en paramètre p une instance de Pile et modifie cette pile afin qu'elle ne conserve que des éléments triés selon le pseudo-tri dictatorial.

On suppose que le sommet de `p` représente le premier élément de la série, comme dans l'algorithme de la question 12. Le premier élément dépilé est donc conservé. Chaque candidat suivant est conservé uniquement s'il est supérieur ou égal au dernier élément conservé ; il est alors ajouté à `p2`. Enfin, le transfert de `p2` vers `p` rétablit l'ordre initial.

```
def tri_dictatorial_pile(p):
    if p.est_vide():
        return

    p2 = Pile()
    dernier_conserve = p.depiler()
    p2.empiler(dernier_conserve)

    while not p.est_vide():
        candidat = p.depiler()
        if candidat >= dernier_conserve:
            dernier_conserve = candidat
            p2.empiler(dernier_conserve)

    while not p2.est_vide():
        p.empiler(p2.depiler())
```

Trace sur la série de l'énoncé `[8, 2, 3, 5, 12]`, avec 8 au sommet initial : après le premier dépilement, `p2` contient 8 ; 2, 3 et 5 sont éliminés ; 12 est conservé et `p2` contient alors 12 au sommet, puis 8. Le transfert final place 8 puis 12 au sommet de `p`, dans l'ordre de la série.

RÉPONSE À RETENIR

Après l'appel, `p` contient les éléments conservés par le pseudo-tri dictatorial, dans leur ordre initial, du sommet vers le fond. La fonction modifie la pile et ne renvoie rien.

Sujet : https://www.sujetdebac.fr/annales-pdf/2025/spe-numerique-informatique-2025-asie-1-remplacement-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.