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 (NSI)

Session 2025 • Asie • Remplacement

Code sujet **BAC** 3 h 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

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

Exercice 2 — Réseaux, routage, graphes, programmation Python
12 réponses • sujet p. 3–5

Exercice 3 — Tableaux, gestion de bugs, listes, piles et POO
16 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

6 points

Bases de données relationnelles et SQL

Question 1.

Sujet p. 1

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

Une clé primaire doit identifier chaque ligne de manière unique. Or plusieurs personnes peuvent avoir le même nom de famille : dans l'extrait, Lisa Benard et Emma Benard ont toutes deux la valeur « Benard » pour l'attribut `nom`. Cet attribut ne permet donc pas de distinguer ces deux lignes.

RÉPONSE À RETENIR

L'attribut `nom` ne peut pas être choisi comme clé primaire, car ses valeurs ne sont pas nécessairement uniques.

Question 2.

Sujet p. 1

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

On sélectionne les colonnes `nom` et `prenom` de la table `joueurs`, et l'on filtre les lignes dont `genre` vaut 2, valeur indiquée pour une joueuse.

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

RÉPONSE À RETENIR

Sur l'extrait fourni, la requête renvoie :

nom	prenom
Panais	Lise
Benard	Lisa
Benard	Emma

Question 3.

Sujet p. 1

> Écrire une requête SQL permettant d'ajouter dans la table joueurs 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 de `joueurs` vont de 1 à 5. On peut choisir 6, qui n'est pas déjà utilisé dans cet extrait. On indique les colonnes explicitement ; l'ordre des valeurs de `VALUES` correspond à cet ordre. Le genre vaut 1 pour un joueur.

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


FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

La requête ajoute Nathan Gervais avec l'identifiant 6 et le genre 1.

Question 4.

Sujet p. 1, 2

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

On repère la ligne à modifier grâce à sa clé primaire, `id = 3`. La clause `WHERE` évite de modifier le nom des autres compétitions.

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

RÉPONSE À RETENIR

Après cette mise à jour, la compétition d'identifiant 3 porte le nom « Open de Tours ».

Question 5.

Sujet p. 1, 2

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

On demande les colonnes `nom` et `annee`, puis on trie par année croissante avec `ORDER BY annee ASC`. L'ordre entre compétitions de même année n'est pas déterminé par cette seule clause de tri.

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

RÉPONSE À RETENIR

Sur les données fournies, après la correction de l'identifiant 3 demandée à la question 4, le résultat est le suivant. Les lignes de 2023 peuvent apparaître dans un autre ordre.

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. 1, 2

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

Les statistiques sont dans `participe`, tandis que les noms et prénoms sont dans `joueurs`. On relie les deux tables par `joueurs.id = participe.id_joueur`, puis on garde les participations pour lesquelles le nombre de coups gagnants est strictement supérieur au nombre de fautes. Il ne faut pas utiliser `DISTINCT` : une personne doit pouvoir apparaître plusieurs fois si elle satisfait la condition dans plusieurs compétitions.

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

RÉPONSE À RETENIR

Dans l'extrait, les participations qui satisfont la condition sont celles de Durand Enzo (15 > 7), Panais Lise (17 > 10), Benard Lisa (18 > 7) et Benard Emma (15 > 11). La requête renvoie donc :

nom	prenom
Durand	Enzo
Panais	Lise
Benard	Lisa
Benard	Emma

Question 7.

Sujet p. 1, 2

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

Il faut relier `joueurs` à `participe` pour retrouver les participations, puis `participe` à `competitions` pour obtenir l'année et le nom du tournoi. On filtre les joueuses avec `genre = 2` et les compétitions de 2023 avec `annee = 2023`. Les alias permettent de distinguer le nom de la joueuse du nom de la compétition.

```
SELECT joueurs.nom AS nom_joueuse,
       joueurs.prenom,
       competitions.nom AS nom_competition
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;
```


FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Sur l'extrait fourni, les participations de joueuses en 2023 produisent les lignes suivantes :

nom_joueuse	prenom	nom_competition
Panais	Lise	Tournoi de Blois
Panais	Lise	Open de Nantes
Benard	Lisa	Open de Nantes
Benard	Emma	Open de Nantes

Question 8.

Sujet p. 1, 2

> On souhaite supprimer de la table joueurs la joueuse Emma Benard, qui ne fait plus partie du club. Déterminer quelle précaution il faut prendre avant de pouvoir le faire et justifier.

`participe.id_joueur` est une clé étrangère vers `joueurs.id`. Dans l'extrait, Emma Benard a l'identifiant 5 et une participation enregistrée pour la compétition 4. En l'absence d'une suppression en cascade précisée dans le sujet, il faut supprimer d'abord ses participations, puis sa ligne dans `joueurs` ; sinon, la clé étrangère ferait référence à un joueur supprimé.

RÉPONSE À RETENIR

Les requêtes, dans le bon ordre, sont :

```
DELETE FROM participe
WHERE id_joueur = 5;
```

```
DELETE FROM joueurs
WHERE id = 5;
```

La première supprime la participation d'Emma Benard ; la seconde peut alors supprimer sa ligne sans laisser de référence étrangère invalide.

Question 9.

Sujet p. 1, 2

> 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 la valeur 7 pour la table joueurs et la valeur 5 pour la table competitions.

Les identifiants 7 pour `joueurs` et 5 pour `competitions` sont demandés. L'identifiant 7 n'apparaît pas dans l'extrait de `joueurs`, mais l'identifiant 5 existe déjà dans `competitions` : il désigne Open de Nantes en 2021. Comme `competitions.id` est une clé primaire, il n'est pas possible d'insérer une seconde ligne portant l'identifiant 5 tout en conservant les données actuelles. La demande est donc impossible à satisfaire telle quelle sans résoudre d'abord cette collision. On ne peut pas remplacer arbitrairement 5 par 7, ni supprimer ou renuméroter la compétition existante sans autorisation, car cela modifierait les données déjà enregistrées.

RÉPONSE À RETENIR

Si l'identifiant 5 est rendu disponible au préalable par une opération autorisée, les insertions doivent ensuite respecter les clés étrangères : créer la joueuse, créer la compétition, puis enregistrer sa participation. Sous cette condition, les requêtes sont :

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

```
INSERT INTO competitions (id, nom, annee)
VALUES (5, 'Tournoi de Blois', 2024);
```

```
INSERT INTO participe (id_joueur, id_compet, nb_fautes, nb_gagnant, aces)
VALUES (7, 5, 14, 15, 2);
```

Ces insertions ne peuvent pas être exécutées avec l'état actuel de la table `competitions`, car la deuxième violerait la clé primaire.

Exercice 2

6 points

Réseaux, routage, graphes, programmation Python

Partie A : réseau et adressage

Question 1.

Sujet p. 3

> Parmi les deux adresses IP 137.254.128.200 et 137.254.128.210, donner l'adresse IP de la machine déjà connectée au sous-réseau Commerces. Ce sous-réseau a pour adresse 137.254.128.0/24 et 207 machines déjà connectées. À part le routeur, les machines déjà présentes sont numérotées dans l'ordre croissant en partant de la plus petite IP disponible.

Dans un réseau `/24`, les adresses d'hôtes utilisables vont de `137.254.128.1` à `137.254.128.254`. En appliquant la numérotation attendue pour les 207 machines, hors routeur, celles-ci occupent les adresses .1 à .207 : .200 en fait partie, mais .210 n'en fait pas partie. Cette conclusion suppose que l'adresse du routeur ne réserve pas précisément une adresse dans cette suite ; elle n'est pas précisée dans l'énoncé.

> **RÉPONSE À RETENIR**
>
> Selon cette convention de numérotation, l'adresse de la machine est **137.254.128.200**.

Question 2.

Sujet p. 3

> Préciser s'il est possible ou non d'ajouter 132 machines sur le sous-réseau Commerces, d'adresse 137.254.128.0/24, sur lequel 207 machines sont déjà connectées. Justifier la réponse.

Un réseau `/24` comporte 256 adresses au total ; après réservation de l'adresse du réseau et de l'adresse de diffusion, il reste 254 adresses utilisables pour les équipements. Même sans compter le routeur, les 207 machines existantes et les 132 machines à ajouter feraient 207 + 132 = 339 machines, soit plus que 254. Le routeur consomme lui aussi une adresse utilisable, ce qui ne peut qu'aggraver le dépassement.

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

Partie B : programmation d'un protocole de routage

Question 3.

Sujet p. 3

> 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, en utilisant le schéma du réseau et les tables de routage simplifiées.

La machine du sous-réseau Navigation transmet d'abord au routeur auquel ce sous-réseau est raccordé, R1. Le sous-réseau Commerces est raccordé à R7. Dans la table, pour la destination R7, on lit la passerelle correspondant au routeur courant : depuis R1, c'est R3 ; depuis R3, c'est R6 ; depuis R6, c'est R7. Le message arrive alors au routeur du sous-réseau Commerces.

RÉPONSE À RETENIR

Le message transite, dans l'ordre, par **R1, R3, R6 et R7**.

Question 4.

Sujet p. 3

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

Le message entre dans le réseau par R7, routeur auquel est raccordé le sous-réseau Commerces. Pour atteindre R1, la table indique, depuis R7, la passerelle R4. Depuis R4, la destination R1 conduit à R6 ; depuis R6, elle conduit de nouveau à R4. Les paquets sont donc renvoyés alternativement entre R4 et R6 au lieu d'atteindre R1. Ils peuvent finalement être supprimés lorsque leur durée de vie expire.

RÉPONSE À RETENIR

Il y a une **boucle de routage** : R7 → R4 → R6 → R4 → R6… Le message n'atteint pas le sous-réseau Navigation.

Question 5.

Sujet p. 4

Pour le réseau obtenu en se limitant aux routeurs R1, R2, R3 et R5, représenté par la figure 2, donner le dictionnaire correspondant au graphe. Les sommets sont décrits par leur nom et le dictionnaire associe à chaque sommet la liste des sommets qui lui sont liés par une arête.

Dans la Figure 2, R1 est relié à R2 et à R3 ; R2 est aussi relié à R3 et à R5. Les liaisons sont des arêtes non orientées : chaque voisin figure donc dans la liste des deux sommets liés.

RÉPONSE À RETENIR

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

Question 6.

Sujet p. 4

Rappeler le principe d'une fonction récursive.

Une fonction récursive s'appelle elle-même, directement ou indirectement. Pour pouvoir se terminer, elle comporte un ou plusieurs cas de base, traités sans nouvel appel récursif, et un ou plusieurs appels récursifs qui ramènent le problème à un cas plus proche du cas de base.

RÉPONSE À RETENIR

Le principe est donc de résoudre les cas de base directement, puis de traiter les autres cas par des appels récursifs sur des problèmes plus simples.

Question 7.

Sujet p. 4

> On dispose de la fonction liste_chemins(graphe, r_depart, r_arrivee), qui renvoie la liste de tous les chemins reliant les deux routeurs. Écrire une fonction plus_court_chemin(graphe, r_depart, r_arrivee) qui renvoie une liste représentant l'un des plus courts chemins entre r_depart et r_arrivee en utilisant le protocole RIP.

La fonction `liste_chemins` fournit tous les chemins possibles. Comme les chemins entre un même départ et une même arrivée ont les mêmes extrémités, choisir celui qui comporte le moins de routeurs revient à choisir celui dont la liste a la plus petite longueur. La fonction suivante renvoie un chemin vide si la liste fournie est vide.

```
def plus_court_chemin(graphe, r_depart, r_arrivee):
    chemins = liste_chemins(graphe, r_depart, r_arrivee)
    if len(chemins) == 0:
        return []
    return min(chemins, key=len)
```

Dans l'exemple du sujet, les chemins de R1 à R5 sont `['R1', 'R2', 'R5']` et `['R1', 'R3', 'R2', 'R5']`. Le premier contient trois routeurs, contre quatre pour le second.

RÉPONSE À RETENIR

Pour l'exemple du sujet, la fonction renvoie `['R1', 'R2', 'R5']`, soit le chemin de deux liaisons. Cette méthode doit toutefois considérer tous les chemins renvoyés par `liste_chemins`, qui peuvent être très nombreux dans un grand réseau.

Question 8.

Sujet p. 4

> Compléter la fonction plus_court_chemin_largeur(graphe, r_depart, r_arrivee), qui réalise un parcours en largeur. Le dictionnaire dict_chemins associe à un routeur le chemin reliant r_depart à ce routeur : 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] for r in L: for s_r in graphe[r]: if not s_r in sommets_marques: sommets_marques.append(...) dict_chemins[s_r] = dict_chemins[r] + [s_r] if s_r == r_arrivee: return ... L.append(s_r)

On marque chaque routeur dès sa découverte : il ne sera ainsi ajouté qu'une seule fois à la liste des routeurs à traiter. Le parcours en largeur explore les routeurs par nombre croissant de liaisons depuis le départ ; le premier chemin construit vers l'arrivée est donc minimal. La liste `L` sert de file de traitement ; les éléments ajoutés pendant son parcours seront également parcourus. Si l'arrivée est inaccessible, la fonction renvoie `None`.

```
def plus_court_chemin_largeur(graphe, r_depart, r_arrivee):
    if r_depart == r_arrivee:
        return [r_depart]

    dict_chemins = {}
    L = [r_depart]
    sommets_marques = [r_depart]
    dict_chemins[r_depart] = [r_depart]

    for r in L:
        for s_r in graphe[r]:
            if s_r not 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 Figure 2, de R1 vers R5 : en traitant R1, le parcours découvre R2 avec le chemin `['R1', 'R2']`, puis R3 avec `['R1', 'R3']`. En traitant R2, il découvre R5 et renvoie `['R1', 'R2', 'R5']`, qui comporte deux liaisons. Le chemin qui passe par R3, R2 et R5 en comporte trois.

> **RÉPONSE À RETENIR**
>
> Les deux compléments demandés dans le code fourni sont `sommets_marques.append(s_r)` et `return dict_chemins[s_r]`. La fonction complète ci-dessus traite aussi le cas où départ et arrivée sont identiques ainsi que celui où l'arrivée est inaccessible.

Question 9.

Sujet p. 4, 5

> 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 différente du routeur étudié, le premier routeur du chemin après le routeur de départ est la passerelle à contacter. Le routeur lui-même est une destination locale, sans passerelle à renseigner ; une destination inaccessible est également 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 and len(chemin) > 1:
                table[destination] = chemin[1]
    return table
```

Sur le graphe de la Figure 2, depuis R1, le chemin vers R2 commence par R2, celui vers R3 par R3 et le chemin vers R5, `['R1', 'R2', 'R5']`, commence par R2.

RÉPONSE À RETENIR

```
table_routage(g, 'R1')
# {'R2': 'R2', 'R3': 'R3', 'R5': 'R2'}
```

Partie C : utilisation du protocole OSPF

Question 10.

Sujet p. 5

Le réseau utilise trois types de connexion : Ethernet (E), de débit 10 mégabits par seconde ; Fast Ethernet (FE), de débit 100 mégabits par seconde ; Fibre (F), de débit 500 mégabits par seconde. Le coût est donné par la formule coût = 10^9 / débit, le débit étant exprimé en bits par seconde. Calculer le coût correspondant à chaque type de liaison.

On convertit d'abord les débits en bits par seconde, puis on applique la formule coût = 10^9 / débit.

Liaison	Débit en bits par seconde	Calcul du coût	Coût
Ethernet (E)	10 000 000	10^9 / 10 000 000	100
Fast Ethernet (FE)	100 000 000	10^9 / 100 000 000	10
Fibre (F)	500 000 000	10^9 / 500 000 000	2

RÉPONSE À RETENIR

Les coûts sont : **E : 100**, **FE : 10** et **F : 2**.

Question 11.

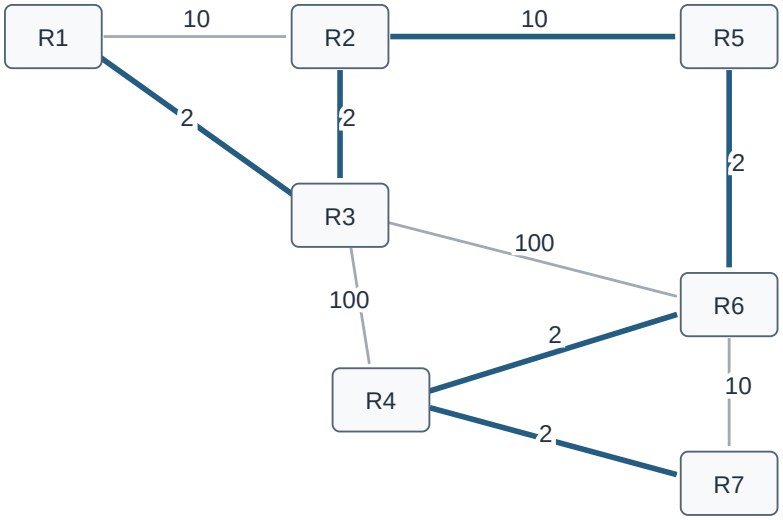
Sujet p. 5

En respectant le protocole OSPF, qui privilégie les routes dont le coût total est minimal, donner la liste des routeurs par lesquels transite un message envoyé depuis le routeur R1 à destination du routeur R7.

Les coûts des liaisons sont ceux calculés à la question 10. Pour justifier le chemin minimal, on peut appliquer l'algorithme de Dijkstra depuis R1. Les premières distances sont R3 = 2 et R2 = 10. En passant par R3, la distance de R2 devient 4 ; on obtient ensuite R5 = 14, puis R6 = 16, R4 = 18 et R7 = 20. Les prédécesseurs successifs sont R3, R2, R5, R6, R4 et R7. Le chemin correspondant a pour coût 2 + 2 + 10 + 2 + 2 + 2 = 20.

FormaV - document d'entraînement non officiel

Liaisons pondérées du réseau et chemin OSPF minimal



Les nombres indiquent les coûts OSPF calculés. Les liaisons mises en évidence composent le chemin minimal de coût total 20.

RÉPONSE À RETENIR

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

Question 12.

Sujet p. 5

> Recopier et compléter la table de routage du routeur R2 en respectant le protocole OSPF. La table comporte les destinations R1, R2, R3, R4, R5, R6 et R7, ainsi qu'une colonne « Suivant ».

Pour chaque destination, on cherche un chemin de coût total minimal depuis R2, puis on indique son premier routeur après R2. Vers R1, passer par R3 coûte 2 + 2 = 4, moins que le coût 10 de la liaison directe. Vers R3, le coût direct est 2. Vers R4, le chemin R2 → R5 → R6 → R4 coûte 10 + 2 + 2 = 14, inférieur au chemin passant par R3 et la liaison Ethernet. Vers R5, le coût direct est 10 ; vers R6, le meilleur chemin via R5 coûte 12 ; vers R7, le chemin via R5, R6 puis R4 coûte 16. La destination R2 est locale et ne nécessite pas de passerelle.

RÉPONSE À RETENIR

Destination	Suivant
R1	R3
R2	— (local)
R3	R3
R4	R5
R5	R5
R6	R5
R7	R5

FormaV - document d'entraînement non officiel

Exercice 3

8 points

Tableaux, gestion de bugs, listes, piles et POO

Partie A

Question 1.

Sujet p. 6

> 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 conservant le premier élément, puis chaque élément supérieur ou égal au dernier élément conservé.

- 31 est conservé, puis 45, car 45 ≥ 31.
- 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**
>
> `tri_dictatorial([31, 45, 41, 28, 37, 108, 127, 2, 124, 421])` renvoie `[31, 45, 108, 127, 421]`.

Question 2.

Sujet p. 6

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

Un algorithme de tri doit réordonner les éléments sans en perdre : le résultat doit contenir les mêmes éléments que la série initiale, éventuellement dans un ordre différent. Or, le tri dictatorial élimine certains éléments. Par exemple, appliqué à `[2, 1]`, il renvoie `[2]` et perd donc le 1. Même si les éléments conservés sont dans l'ordre croissant, l'algorithme ne trie pas toute la série.

> **RÉPONSE À RETENIR**
>
> Le tri dictatorial n'est pas un algorithme de tri, car il supprime des éléments au lieu de conserver tous les éléments de la série et de les réordonner.

Question 3.

Sujet p. 6

> Edgar a écrit le programme présenté pour implémenter le tri dictatorial. Il 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 programme commence par initialiser `serie_triee` avec le premier élément, soit `[8]`. Il compare ensuite chaque élément à l'élément qui le précède dans la série d'origine :

- `i = 1` : 2 ≥ 8 est faux, la liste reste `[8]`.
- `i = 2` : 9 ≥ 2 est vrai, la liste devient `[8, 9]`.
- `i = 3` : 6 ≥ 9 est faux, elle reste `[8, 9]`.
- `i = 4` : 12 ≥ 6 est vrai, elle devient `[8, 9, 12]`.

FormaV - document d'entraînement non officiel

À la dernière étape, la comparaison porte bien sur 6, l'élément précédent dans la série d'origine, et non sur le dernier élément de `serie_triee`.

> **RÉPONSE À RETENIR**
>
> À la fin de l'appel, `serie_triee` vaut `[8, 9, 12]`.

Question 4.

Sujet p. 6, 7

> Edgar réalise l'appel tri_dictatorial([]) et obtient l'erreur IndexError: list index out of range à l'instruction serie_triee = [serie[0]]. 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 dans l'introduction.

Dans une liste vide, aucun élément n'a d'indice 0. L'accès à `serie[0]` provoque donc l'erreur `IndexError: list index out of range`. L'exécution s'interrompt avant que la boucle soit exécutée. Il faut traiter le cas vide avant d'accéder au premier élément :

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

Cette modification permet à l'appel sur la liste vide de respecter la définition. La comparaison avec l'élément précédent de la série d'origine présente un autre défaut, traité à la question 6.

> **RÉPONSE À RETENIR**
>
> Avec le test préalable, `tri_dictatorial([])` renvoie `[]` sans erreur.

Question 5.

Sujet p. 7

> 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 donnée et peut révéler un bug. Mais il ne vérifie pas nécessairement toutes les entrées possibles. Pour un programme qui accepte un grand nombre d'entrées, voire une infinité d'entrées, des cas non testés peuvent encore faire apparaître une erreur. Réussir un ensemble de tests ne constitue donc pas, en général, une preuve que le programme est correct pour toutes ses entrées.

> **RÉPONSE À RETENIR**
>
> Les tests peuvent révéler la présence de bugs, mais ils ne prouvent pas leur absence en général, car ils ne couvrent pas nécessairement tous les cas d'entrée.

Question 6.

Sujet p. 7

> Pour l'appel tri_dictatorial([8, 2, 3, 5, 12]), la fonction d'Edgar renvoie [8, 3, 5, 12], qui n'est pas une liste triée.

FormaV - document d'entraînement non officiel

> Expliquer la cause du problème et proposer une modification du code d'Edgar afin de la corriger.

Le programme compare `serie[i]` à `serie[i - 1]`, alors que l'algorithme demande de comparer l'élément candidat au dernier élément conservé. Pour la série donnée, 3 est comparé à 2 et ajouté après 8, alors qu'il est inférieur à 8 et doit être éliminé. Il faut donc comparer le candidat au dernier élément de `serie_triee`. Voici une version qui corrige également le cas de la liste vide :

```
def tri_dictatorial(serie):
    if len(serie) == 0:
        return []
    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
```

La boucle examine une fois chaque élément après le premier. La comparaison au dernier élément conservé garantit que la liste produite reste dans l'ordre croissant.

RÉPONSE À RETENIR

Pour `[8, 2, 3, 5, 12]`, le programme corrigé renvoie `[8, 12]` : 2, 3 et 5 sont éliminés car ils sont inférieurs au dernier élément conservé, 8.

Partie B

Question 7.

Sujet p. 7

> Donner des instructions permettant de construire les trois maillons m1, m0 et m8 ainsi que la liste chaînée représentée dans la figure 1. Nommer cette liste chaînée ma_liste.

On construit les maillons de la fin vers le début afin de pouvoir fournir à chacun son maillon suivant. Le dernier maillon pointe vers `None`, puis la tête de la liste est le maillon de valeur 1.

RÉPONSE À RETENIR

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

La chaîne obtenue est `m1 → m0 → m8 → None`.

Question 8.

Sujet p. 7

> Indiquer ce que renvoie l'instruction m1.valeur == 1.

L'attribut `valeur` de `m1` vaut 1 ; l'égalité testée est donc vraie.

RÉPONSE À RETENIR

`m1.valeur == 1` renvoie `True`.

Question 8.

Sujet p. 7

> Indiquer ce que renvoie l'instruction m1.suivant.valeur == 8.

`m1.suivant` est `m0`, dont la valeur est 0. Cette valeur n'est pas égale à 8.

> **RÉPONSE À RETENIR**
>
> `m1.suivant.valeur == 8` renvoie `False`.

Question 8.

Sujet p. 7

> Indiquer ce que renvoie l'instruction m1.suivant.suivant == None.

`m1.suivant.suivant` désigne `m8`, qui est un maillon et non `None`.

> **RÉPONSE À RETENIR**
>
> `m1.suivant.suivant == None` renvoie `False`.

Question 8.

Sujet p. 7

> Indiquer ce que renvoie l'instruction m1.suivant.suivant.suivant == None.

`m8` est le dernier maillon de la chaîne et son attribut `suivant` vaut `None`. Ainsi, `m1.suivant.suivant.suivant` vaut `None`.

> **RÉPONSE À RETENIR**
>
> `m1.suivant.suivant.suivant == None` renvoie `True`.

Question 9.

Sujet p. 7

> Donner une instruction permettant de transformer ma_liste comme le représente le schéma : le maillon contenant 1 pointe vers le maillon contenant 8 et le maillon contenant 0 pointe également vers le maillon contenant 8.

Le schéma demandé montre que le maillon de valeur 1 doit pointer directement vers `m8`, tandis que le lien de `m0` vers `m8` demeure inchangé. Il suffit donc de remplacer le lien suivant de `m1`.

> **RÉPONSE À RETENIR**
>
> ```
> m1.suivant = m8
> ```
>
> La chaîne accessible depuis `ma_liste` commence alors par `m1`, puis `m8` ; `m0` n'est plus accessible depuis cette tête.

Question 10.

Sujet p. 7, 8

> Recopier et compléter la fonction tri_dictatorial_chaine ci-dessous. Elle 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 : def tri_dictatorial_chaine(chaine): maillon = chaine.tete while maillon.suivant ...: if maillon.valeur ... maillon = ... else: maillon.suivant = ...

La tête est le premier élément de la série et doit être conservée. On examine ensuite le maillon suivant. Si sa valeur est supérieure ou égale à celle du dernier maillon conservé, on avance. Sinon, on retire ce maillon en reliant directement le maillon conservé à son successeur ; on ne déplace pas le curseur, car le nouveau maillon suivant doit être examiné à son tour.

```
def tri_dictatorial_chaine(chaine):
    maillon = chaine.tete
    if maillon is None:
        return
    while maillon.suivant is not None:
        if maillon.suivant.valeur >= maillon.valeur:
            maillon = maillon.suivant
        else:
            maillon.suivant = maillon.suivant.suivant
```

Par exemple, pour la chaîne `2 → 3 → 1 → 8`, on avance de 2 à 3, on retire 1 en reliant 3 à 8, puis on avance à 8 : la chaîne finale est `2 → 3 → 8`. À chaque étape, soit le curseur avance, soit un maillon est retiré. Comme la chaîne est finie, la boucle termine ; le temps d'exécution est proportionnel au nombre de maillons initiaux.

RÉPONSE À RETENIR

La fonction modifie la chaîne en place, traite aussi une chaîne vide et ne renvoie aucune valeur explicite.

Partie C

Question 11.

Sujet p. 8

> Rappeler le principe du fonctionnement d'une pile.

Dans une pile, les ajouts et les retraits se font au même endroit, appelé le sommet. Elle suit le principe LIFO (« dernier entré, premier sorti ») : l'élément empilé le plus récemment est le premier que l'on dépile.

RÉPONSE À RETENIR

Une pile fonctionne selon le principe LIFO : dernier entré, premier sorti.

Question 12.

Sujet p. 8

> Remettre dans l'ordre les lignes ci-dessous afin d'obtenir l'algorithme attendu, en respectant une tabulation lorsqu'une ligne se trouve à l'intérieur d'un bloc si ou tant que. Lignes à ordonner : si p n'est pas vide ; tant que p n'est pas vide ; tant que p2 n'est pas vide ; on dépile p, on stocke l'élément obtenu dans dernier_conservé et on

FormaV - document d'entraînement non officiel

> l'empile dans p2 ; on crée une pile intermédiaire p2 vide ; on dépile p et on stocke l'élément obtenu dans candidat ; si candidat est supérieur ou égal à dernier_conservé ; on dépile p2 et on empile l'élément obtenu dans p ; dernier_conservé prend la valeur de candidat et on l'empile dans p2.

Si la pile est non vide, on dépile d'abord son premier élément pour l'initialiser comme dernier élément conservé et on l'empile dans `p2`. On examine ensuite les autres éléments dans l'ordre des dépilements : seuls les candidats supérieurs ou égaux au dernier élément conservé sont ajoutés à `p2`. Enfin, le transfert de `p2` vers `p` inverse une seconde fois l'ordre et restitue dans `p` les éléments conservés dans l'ordre du parcours initial.

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 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 candidat
        si candidat est supérieur 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
```

Question 13.

Sujet p. 8

> Écrire en Python la fonction tri_dictatorial_pile qui prend en paramètre p, une instance de la classe Pile, et modifie cette pile afin qu'elle ne conserve que des éléments triés selon le pseudo-tri dictatorial. La classe Pile dispose des méthodes __init__(self), __str__(self), depiler(self), empiler(self, elt) et est_vide(self).

La pile intermédiaire mémorise uniquement les éléments conservés. On dépile le premier élément de `p` pour initialiser `dernier_conserve`. Chaque candidat suivant est comparé à cette dernière valeur conservée. Le transfert final de `p2` vers `p` remet les éléments dans l'ordre du parcours initial. Si la pile est vide, la fonction ne fait rien.

```
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(candidat)

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

Trace sur l'exemple de l'énoncé `2, 3, 1, 8`, en supposant que les dépilements de `p` donnent les éléments dans cet ordre : 2 est conservé ; 3 est supérieur ou égal à 2, donc il est conservé ; 1 est inférieur à 3, donc il est écarté ; 8 est supérieur ou égal à 3, donc il est conservé. `p2` contient alors les éléments conservés dans l'ordre inverse du parcours ; le transfert final restitue dans `p` la série `2, 3, 8`. Chaque élément est dépilé au plus une fois et chaque élément conservé est transféré une fois : l'algorithme termine en un temps proportionnel au nombre d'éléments.

> **RÉPONSE À RETENIR**
>
> La fonction modifie `p` pour qu'elle ne contienne que les éléments conservés par le tri dictatorial, dans leur ordre de parcours initial ; elle ne renvoie rien.

Sujet : https://aeif.fr/wp-content/uploads/2025/12/Asie_septembre_2025.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.