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.

Numérique et sciences informatiques

Session 2023 • Jour 2

Code sujet **23-NSIJ2LI1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

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

Exercice 2 — Réseaux, programmation Python et algorithmique
8 réponses • sujet p. 4–7

Exercice 3 — Arbres binaires de recherche et algorithmique
10 réponses • sujet p. 7–12

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

Bases de données relationnelles et requêtes SQL

a

Question 1.a.

Sujet p. 2, 3

> Identifier, pour chacune des deux relations « prestations » et « clients », une clé primaire possible. Justifier les réponses aux questions a. et b.

Une clé primaire doit identifier sans ambiguïté chaque ligne d'une relation. Dans les données fournies, chaque valeur de `id_prestation` est différente dans `prestations`, et chaque valeur de `id_client` est différente dans `clients`. Ce sont donc des clés primaires possibles.

> **RÉPONSE À RETENIR**
>
> `prestations` : clé primaire possible `id_prestation`.
> `clients` : clé primaire possible `id_client`.

b

Question 1.b.

Sujet p. 2, 3

> Identifier la relation qui possède une clé étrangère et donner cette clé étrangère. Justifier les réponses aux questions a. et b.

Une clé étrangère est un attribut d'une relation qui référence la clé primaire d'une autre relation. Chaque `id_client` associé à une prestation désigne le client correspondant dans la relation `clients`.

> **RÉPONSE À RETENIR**
>
> La relation qui possède une clé étrangère est `prestations`. Cette clé étrangère est `id_client` ; elle référence `clients.id_client`.

c

Question 1.c.

Sujet p. 2, 3

> Donner le type de chacun des deux premiers attributs de la relation « clients ».

Les deux premiers attributs de `clients` sont `id_client` et `nom`. Les identifiants sont des nombres entiers et les noms sont du texte.

> **RÉPONSE À RETENIR**
>
> `id_client` : entier.
> `nom` : chaîne de caractères (texte).

a

Question 2.a.

Sujet p. 2, 3

> Donner les valeurs renvoyées par la requête : SELECT nom, telephone FROM clients WHERE ville='Montélimar';

La condition `ville='Montélimar'` sélectionne les clients de Montélimar. Pour ces lignes, on relève les attributs `nom` et `telephone`.

RÉPONSE À RETENIR

nom	telephone
Ouellet	0475016031
Foucault	0475918885
Croteau	0475460794
Rivard	0475339127

b

Question 2.b.

Sujet p. 2, 3

> Écrire la requête permettant d'obtenir la date et l'heure de début des prestations de plus d'une heure, réalisées par Didier.

On sélectionne les deux attributs demandés, `date` et `heure_debut`. Les deux conditions doivent être vérifiées : la durée est strictement supérieure à une heure et l'employé est Didier.

RÉPONSE À RETENIR

```
SELECT date, heure_debut
FROM prestations
WHERE duree > 1 AND employe = 'Didier';
```

Question 3.

Sujet p. 2, 3

> Avec les informations présentées à la page précédente, donner les valeurs renvoyées par la requête : SELECT DISTINCT nom FROM clients JOIN prestations ON clients.id_client=prestations.id_client WHERE prestations.employe='Sylvie'; Le mot-clé DISTINCT permet d'éviter les doublons dans le résultat.

La jointure relie chaque prestation à son client par l'égalité des attributs `id_client`. Le filtre garde les prestations réalisées par Sylvie, puis `DISTINCT` supprime les noms répétés. Les identifiants clients concernés sont 5231, 5229, 5230, 5234 et 5232.

Comme la requête ne contient pas de clause `ORDER BY`, l'ordre des résultats n'est pas spécifié.

RÉPONSE À RETENIR

Les noms renvoyés, dans un ordre non garanti, sont : Foucault, Rouze, Bonenfant, Rivard et Croteau.

a

Question 4.a.

Sujet p. 2, 3

> Pour permettre au logiciel de préparer les factures, on souhaite ajouter le tarif horaire de chaque prestation dans ce jeu de données. Expliquer quel(s) problème(s) peuvent apparaître si l'on décide d'ajouter uniquement un attribut tarif_horaire à la relation « prestations ».

En supposant que le tarif horaire est déterminé par le type de prestation, un attribut `tarif_horaire` ajouté directement à `prestations` répéterait le même tarif pour plusieurs prestations de même type. Cela crée une redondance et un risque d'incohérence si le tarif n'est pas modifié partout. Cette organisation peut aussi entraîner des anomalies lors de l'insertion ou de la suppression de lignes. Une relation séparée permet de stocker chaque tarif une seule fois.

> **RÉPONSE À RETENIR**
>
> Si le tarif dépend du type de prestation, le stocker uniquement dans `prestations` entraîne une redondance et un risque d'incohérence lors des mises à jour. Il est préférable d'enregistrer les tarifs dans une relation dédiée.

b

Question 4.b.

Sujet p. 2, 3

> Proposer un schéma relationnel qui permet d'éviter ce(s) problème(s). La réponse peut être présentée sous la forme tarifs(premier attribut, deuxième attribut,…), prestations(premier attribut,…) et clients(premier attribut,…). Indiquer explicitement, pour chaque relation, les attributs, la clé primaire et les éventuelles clés étrangères ; les types ne sont pas demandés.

On suppose que chaque type de prestation détermine un tarif horaire. La relation `tarifs` associe alors chaque type à son tarif, et `type` en est la clé primaire. L'attribut `type` reste dans `prestations` pour indiquer le type de chaque intervention ; il y devient une clé étrangère vers `tarifs`. Le lien avec le client est conservé.

> **RÉPONSE À RETENIR**
>
> Schéma relationnel proposé :
>
> - `tarifs(type, tarif_horaire)` : clé primaire `type`.
> - `prestations(id_prestation, id_client, date, heure_debut, duree, type, employe)` : clé primaire `id_prestation` ; clés étrangères `id_client` vers `clients(id_client)` et `type` vers `tarifs(type)`.
> - `clients(id_client, nom, adresse, code_postal, ville, telephone)` : clé primaire `id_client` ; aucune clé étrangère.

Exercice 2

3 points

Réseaux, programmation Python et algorithmique

Question 1.

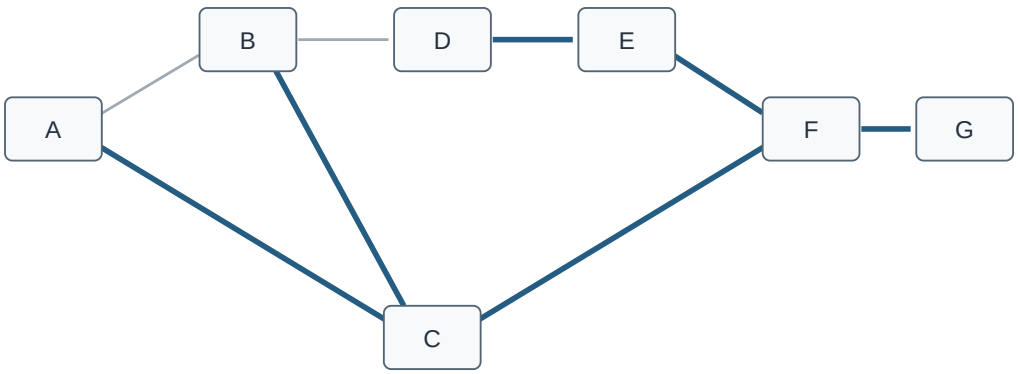
Sujet p. 4

> Selon le protocole RIP, la distance est le nombre de routeurs traversés par un chemin et le chemin choisi minimise cette distance. Reproduire et compléter la table indiquant, pour chaque routeur, la portion de sa table de routage vers le routeur G : passerelle, interface et distance. La ligne A est déjà renseignée : destination G, passerelle C, interface eth0, distance 2. Le réseau est représenté sur la page.

On suit les liaisons du graphe pour trouver, depuis chaque routeur, un chemin de distance minimale vers G. La ligne A donnée dans le sujet confirme la convention de distance : le chemin A-C-F-G a une distance de 2, correspondant aux routeurs intermédiaires C et F. La passerelle est le premier routeur rencontré sur le chemin vers G ; l'interface est celle qui permet de joindre cette passerelle.

Les chemins retenus sont B-C-F-G, C-F-G, D-E-F-G, E-F-G et F-G. Les interfaces vers les passerelles se lisent sur les ports du graphe.

Réseau des routeurs et chemins retenus vers G



Les liaisons représentées reprennent celles du sujet, sans leur attribuer de poids numériques. Les arêtes mises en évidence sont celles des chemins retenus vers G.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Routeur	destination	passerelle	interface	distance
A	G	C	eth0	2
B	G	C	eth2	2
C	G	F	eth0	1
D	G	E	eth1	2
E	G	F	eth1	1
F	G	G	eth1	0

Par exemple, depuis B, les routeurs intermédiaires sont C et F : la distance est 2. Depuis F, G est directement la destination et il n'y a aucun routeur intermédiaire : la distance est 0.

Question 2.

Sujet p. 5

Le grossiste applique une réduction de 10 % si le montant de la commande est supérieur ou égal à 100 € et strictement inférieur à 200 €, et une réduction de 20 % si le montant est supérieur ou égal à 200 €. Écrire en les complétant les lignes de code, à partir de la ligne 13, de la fonction calcul_montant afin de respecter la spécification indiquée.

À la ligne 12, le montant avant réduction est calculé. Il faut ensuite distinguer les montants de 100 € inclus à 200 € exclu, ceux d'au moins 200 €, et ceux qui sont inférieurs à 100 €. La fonction renvoie le montant après réduction éventuelle.

```
if montant >= 100 and montant < 200:
    montant = montant - montant * 0.10
elif montant >= 200:
    montant = montant - montant * 0.20

return montant
```

RÉPONSE À RETENIR

Pour la commande n°343, le montant avant réduction vaut 55 € : aucune condition de réduction n'est vérifiée, donc le montant renvoyé est 55 €. Vérification des seuils : 140 € devient 126 € et 200 € devient 160 €.

a

Question 3.a.

Sujet p. 5, 6

Appliquer, de la même façon que dans l'exemple de déroulement du tri-fusion, l'algorithme à la liste [46, 108, 55, 53] et présenter son déroulement.

On partage la liste jusqu'à obtenir des listes d'un élément, qui sont déjà triées, puis on fusionne les sous-listes triées.

1. Découpage initial : [46, 108, 55, 53] devient [46, 108] et [55, 53].

FormaV - document d'entraînement non officiel

2. Découpage suivant : [46], [108], [55] et [53].
3. Premières fusions : [46] et [108] donnent [46, 108] ; [55] et [53] donnent [53, 55].
4. Fusion finale de [46, 108] et [53, 55] : on prend successivement 46, 53 et 55, puis on ajoute l'élément restant 108.

> **RÉPONSE À RETENIR**
>
> La liste triée obtenue est [46, 53, 55, 108].

b

Question 3.b. — proposition 1

Sujet p. 6, 7

> Parmi les quatre propositions, indiquer l'instruction à placer à la ligne 4 de la fonction fusion. Proposition 1 : while i1 < len(liste1) and i2 < len(liste2):

La boucle de comparaison ne doit s'exécuter que tant que les deux listes ont un élément courant. Cette condition garantit que les accès liste1[i1] et liste2[i2] sont valides. Une fois la boucle terminée, les boucles suivantes du code ajoutent les éléments encore présents dans l'une ou l'autre des listes.

> **RÉPONSE À RETENIR**
>
> Proposition 1 : correcte.
>
> ```
> while i1 < len(liste1) and i2 < len(liste2):
> ```

Question 3.b. — proposition 2

Sujet p. 6, 7

> Parmi les quatre propositions, indiquer l'instruction à placer à la ligne 4 de la fonction fusion. Proposition 2 : while i1 < len(liste1) or i2 < len(liste2):

Avec `or`, la boucle continuerait même si une seule liste contient encore des éléments. Elle pourrait alors accéder à un indice hors limites dans l'autre liste, déjà épuisée.

> **RÉPONSE À RETENIR**
>
> Proposition 2 : incorrecte.

Question 3.b. — proposition 3

Sujet p. 6, 7

> Parmi les quatre propositions, indiquer l'instruction à placer à la ligne 4 de la fonction fusion. Proposition 3 : while i1 + i2 < len(liste1) + len(liste2):

La somme des indices ne garantit pas que chacun des indices soit valide. Si une liste est épuisée alors que l'autre contient encore des éléments, cette condition peut rester vraie ; le corps de la boucle risque alors d'accéder à un indice hors limites.

> **RÉPONSE À RETENIR**
>
> Proposition 3 : incorrecte.

Question 3.b. — proposition 4

Sujet p. 6, 7

> Parmi les quatre propositions, indiquer l'instruction à placer à la ligne 4 de la fonction fusion. Proposition 4 : while liste1[i1 + 1] < liste2[i2]:

Cette condition compare liste1[i1 + 1] à liste2[i2], alors que les éléments courants à comparer sont liste1[i1] et liste2[i2]. Elle ne vérifie pas non plus que les indices utilisés sont valides ; elle peut donc provoquer un dépassement d'indice.

> **RÉPONSE À RETENIR**
>
> Proposition 4 : incorrecte.

c

Question 3.c.

Sujet p. 5, 6, 7

> Recopier et compléter la fonction récursive tri_fusion, qui prend en paramètre une liste non triée et la renvoie sous la forme d'une nouvelle liste triée. Utiliser la fonction fusion de la question précédente. Exemple : tri_fusion([140,115,78,94]) doit renvoyer [78,94,115,140]. L'aide Python précise que L[i:j] renvoie la liste constituée des éléments de L indexés de i à j-1 ; par exemple, si L=[5,4,8,7,3], L[2:4] vaut [8,7].

Le cas de base est une liste vide ou à un élément, qui est déjà triée. Sinon, on découpe la liste en deux, on trie récursivement chaque moitié, puis on fusionne les résultats.

```
def tri_fusion(liste):
    if len(liste) <= 1:
        return liste
    milieu = len(liste) // 2
    liste1 = tri_fusion(liste[:milieu])
    liste2 = tri_fusion(liste[milieu:])
    return fusion(liste1, liste2)
```

Trace avec la liste de la question 3.a : [46, 108, 55, 53] est séparée en [46, 108] et [55, 53], puis en listes d'un élément. Les fusions donnent [46, 108] et [53, 55], puis [46, 53, 55, 108]. La fonction renvoie également l'exemple fourni dans le sujet : tri_fusion([140, 115, 78, 94]) donne [78, 94, 115, 140].

La récursion termine car chaque appel sur une liste de taille supérieure à 1 porte sur deux listes strictement plus petites. Pour une liste de taille n, chaque niveau de fusion traite au total n éléments et le nombre de niveaux est de l'ordre de $\log_2(n)$; le temps de calcul est donc de l'ordre de $n \log_2(n)$.

> **RÉPONSE À RETENIR**
>
> La fonction renvoie une nouvelle liste triée ; pour une liste vide ou à un élément, elle renvoie directement cette liste, déjà triée.

FormaV - document d'entraînement non officiel

Exercice 3

5 points

Arbres binaires de recherche et algorithmique

Question 1.

Sujet p. 7, 8

> Parmi les trois arbres dessinés, recopier le numéro de celui qui n'est pas un arbre binaire de recherche et justifier. La définition donnée précise que, pour chaque nœud, les valeurs du sous-arbre gauche lui sont strictement inférieures et celles du sous-arbre droit lui sont supérieures ou égales.

Un nœud du sous-arbre gauche d'un nœud doit avoir une valeur strictement inférieure à celle-ci ; dans son sous-arbre droit, les valeurs doivent être supérieures ou égales. Il faut vérifier ces contraintes pour chaque nœud et par rapport à ses ancêtres.

Dans l'arbre 1, le sous-arbre gauche de la racine 20 ne contient que des valeurs inférieures à 20 et son sous-arbre droit que des valeurs supérieures ou égales à 20. Les autres nœuds respectent également ces règles. L'arbre 1 est donc un ABR.

Dans l'arbre 2, 29 est le fils droit de 20 et appartient au sous-arbre gauche de 46 : on a bien $20 \leq 29 < 46$. Les autres nœuds respectent les contraintes de leurs ancêtres. L'arbre 2 est donc un ABR.

Dans l'arbre 3, 20 appartient au sous-arbre droit de la racine 29, alors que $20 < 29$. Cette disposition enfreint la définition d'un ABR.

> **RÉPONSE À RETENIR**
>
> L'arbre qui n'est pas un ABR est l'**arbre 3**.

a

Question 2.a.

Sujet p. 8, 9

> En utilisant les méthodes de la classe ABR, écrire l'instruction Python qui permet d'instancier un objet a0, de type ABR, ayant un seul nœud (la racine) de valeur 18.

Le constructeur de la classe `ABR` prend la valeur de la racine, puis les sous-arbres gauche et droit. Pour que l'arbre ne contienne que la racine de valeur 18, ces deux sous-arbres doivent être vides, représentés par `None`.

> **RÉPONSE À RETENIR**
>
> ```
> a0 = ABR(18, None, None)
> ```

Question 2.a.

Sujet p. 8, 9

> Écrire une séquence d'instructions permettant ensuite d'insérer dans l'objet a0 les deux feuilles de l'arbre représenté, de valeurs 12 et 36.

La méthode `inserer_noeud` renvoie un nouvel ABR contenant le nœud inséré. Il faut donc réaffecter le résultat à `a0` après chaque appel. Comme $12 < 18$, 12 est placé à gauche ; comme $36 > 18$, 36 est placé à droite.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
a0 = a0.inserer_noeud(12)
a0 = a0.inserer_noeud(36)
```

b

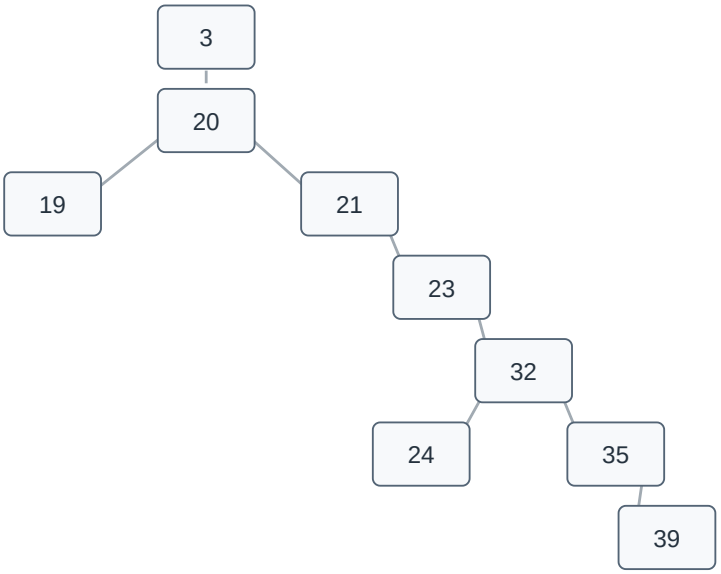
Question 2.b.

Sujet p. 7, 9

> Dessiner sur la copie l'ABR nommé a2 obtenu en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 3, puis en insérant successivement les valeurs dans l'ordre suivant : 20 ; 19 ; 21 ; 23 ; 32 ; 24 ; 35 ; 39.

On insère les valeurs dans l'ordre indiqué, chacune comme une feuille à sa place. La racine est 3 ; 20 est à sa droite. Ensuite, 19 devient le fils gauche de 20 et 21 son fils droit. Puis 23 se place à droite de 21, 32 à droite de 23, 24 à gauche de 32, 35 à droite de 32 et 39 à droite de 35.

ABR a2 obtenu par insertions successives



Chaque arête relie un nœud à l'un de ses enfants ; 24 est le fils gauche de 32 et 35 son fils droit.

RÉPONSE À RETENIR

On obtient l'ABR `a2` représenté ci-dessous.

c

Question 2.c.

Sujet p. 7, 9

> Donner la hauteur des ABR a1 et a2. La hauteur d'un arbre ne contenant qu'un nœud vaut 1.

La hauteur est le nombre de nœuds du plus long chemin entre la racine et une feuille. Dans `a1`, un plus long chemin est 24 → 21 → 19 → 3 (ou 20), soit 4 nœuds. Dans `a2`, le plus long chemin est 3 → 20 → 21 → 23 → 32 → 35 → 39, soit 7 nœuds.

RÉPONSE À RETENIR

Hauteur de `a1` : **4**. Hauteur de `a2` : **7**.

d

Question 2.d.

Sujet p. 9, 10

> On complète la classe ABR avec une méthode calculer_hauteur qui renvoie la hauteur de l'arbre. Recopier les lignes 10 et 13 en les complétant par des commentaires, puis compléter la ligne 14 par une instruction. On pourra utiliser la fonction Python max, qui prend deux nombres et renvoie leur maximum.

Une feuille a une hauteur de 1. Avec un seul sous-arbre non vide, la hauteur est égale à 1 plus la hauteur de ce sous-arbre. Si les deux sous-arbres sont non vides, on ajoute 1 à la plus grande de leurs hauteurs. Les commentaires complètent les cas prévus aux lignes 10 et 13 du sujet.

Le calcul s'effectue récursivement jusqu'aux feuilles. Pour `a2`, le plus long chemin de la racine à une feuille contient 7 nœuds, donc la méthode renvoie 7.

RÉPONSE À RETENIR

```
def calculer_hauteur(self):
    """Renvoie la hauteur de l'arbre"""

    if self.sa_droit is None and self.sa_gauche is None:
        # l'arbre est réduit à une feuille
        return 1
    elif self.sa_droit is None:
        # arbre avec une racine et seulement un sous-arbre gauche
        return 1 + self.sa_gauche.calculer_hauteur()
    elif self.sa_gauche is None:
        # arbre avec une racine et seulement un sous-arbre droit
        return 1 + self.sa_droit.calculer_hauteur()
    else:
        # arbre avec une racine et deux sous-arbres non vides
        return 1 + max(self.sa_gauche.calculer_hauteur(),
self.sa_droit.calculer_hauteur())
```


FormaV - document d'entraînement non officiel

a

Question 3.a.

Sujet p. 8, 10

> Recopier et compléter les lignes 6, 8, 11 et 13 du code de la méthode rechercher_valeur, qui permet de tester la présence ou l'absence d'une valeur donnée dans l'ABR.

On compare d'abord `v` à la valeur du nœud courant. En cas d'égalité, la valeur est trouvée. Si `v` est plus petit, la recherche continue dans le sous-arbre gauche s'il existe ; si `v` est plus grand, elle continue dans le sous-arbre droit s'il existe. Si aucun de ces cas ne permet de continuer, la valeur est absente.

RÉPONSE À RETENIR

```
def rechercher_valeur(self, v):
    """
    Renvoie True si la valeur v est trouvée dans l'ABR,
    False sinon
    """
    if v == self.valeur:
        return True
    elif v < self.valeur and self.sa_gauche is not None:
        return self.sa_gauche.rechercher_valeur(v)
    elif v > self.valeur and self.sa_droit is not None:
        return self.sa_droit.rechercher_valeur(v)
    else:
        return False
```

b

Question 3.b.

Sujet p. 9, 10

> On admet que la méthode rechercher_valeur est appelée 7 fois pour rechercher la valeur 39 dans l'ABR a2. Donner le nombre de fois où cette méthode est appelée pour rechercher la valeur 20 dans l'ABR a1.

Dans `a1`, la recherche de 20 suit le chemin 24 → 21 → 19 → 20. La méthode est appelée une fois à chacun des nœuds visités, racine comprise ; elle est appelée à nouveau au nœud 20 où la recherche s'arrête.

RÉPONSE À RETENIR

La méthode `rechercher_valeur` est appelée **4 fois**.

a

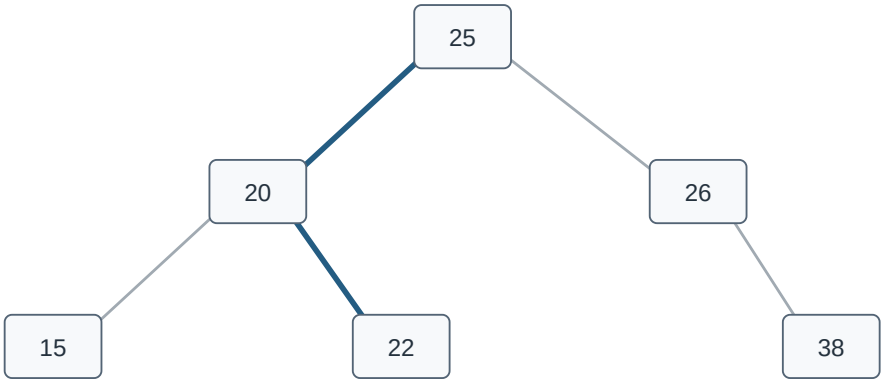
Question 4.a.

Sujet p. 12

> En suivant les différentes étapes de l'algorithme de rotation gauche, dessiner l'arbre obtenu après une rotation gauche de l'ABR représenté.

Le pivot est le sous-arbre droit de la racine : il est enraciné en 25. Son sous-arbre gauche, enraciné en 22, devient le sous-arbre droit de l'ancienne racine 20. Ensuite, 20 devient le fils gauche de 25, qui devient la nouvelle racine. Le sous-arbre 15 reste à gauche de 20, et les sous-arbres de 25 situés à droite, enracinés en 26 puis 38, conservent leur position relative.

ABR après rotation gauche autour de 20



Le sous-arbre enraciné en 22 devient le sous-arbre droit de 20.

> **RÉPONSE À RETENIR**
>
> Après la rotation gauche, la racine est **25** ; son fils gauche est 20, avec 15 à gauche et 22 à droite, et son fils droit est 26, avec 38 à droite.

b

Question 4.b.

Sujet p. 12

> Écrire le code d'une méthode Python rotation_gauche qui réalise la rotation gauche d'un ABR autour de sa racine.

On reprend les transformations de la méthode `rotation_droite` fournie dans le sujet en les adaptant au sous-arbre droit. Le pivot est `self.sa_droit` ; son sous-arbre gauche devient le sous-arbre droit de l'ancienne racine ; enfin, l'ancienne racine devient le sous-arbre gauche du pivot. Comme dans la méthode donnée, on suppose que le sous-arbre droit existe et on renvoie une nouvelle instance d'ABR.

Pour l'arbre de la question 4.a, le pivot est 25 et son sous-arbre gauche est enraciné en 22. L'affectation rattache donc 22 à droite de 20, puis 20 devient le sous-arbre gauche de 25, conformément au résultat de la rotation.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def rotation_gauche(self):
    """Renvoie une instance d'un ABR après une rotation gauche.
    On suppose qu'il existe un sous-arbre droit.
    """
    pivot = self.sa_droit
    self.sa_droit = pivot.sa_gauche
    pivot.sa_gauche = self
    return ABR(pivot.valeur, pivot.sa_gauche, pivot.sa_droit)
```

Sujet : https://www.sujetdebac.fr/annales-pdf/2023/spe-numerique-informatique-2023-liban-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.