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 2024 • Métropole • Jour 1

Code sujet **24-NSIJ1ME1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Programmation objet en Python et graphes
11 réponses • sujet p. 2–5

Exercice 2 — Protocoles de routage, sécurité des communications et bases de données relationnelles
10 réponses • sujet p. 6–8

Exercice 3 — Programmation orientée objet, arbres binaires de recherche et récursivité
14 réponses • sujet p. 9–15

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

Programmation objet en Python et graphes

Question 1.

Sujet p. 3

> Expliquer la ligne 9 de la description du graphe : s2.predecesseurs = [].

La liste `s2.predecesseurs` contient les sites qui ont au moins un lien dirigé vers `s2`, ainsi que le nombre de ces liens. La liste vide signifie qu'aucun lien du graphe ne pointe vers `site2`.

RÉPONSE À RETENIR

`s2` n'a aucun prédécesseur dans le graphe.

Question 2.

Sujet p. 2, 3

> Les lignes 11 et 12 de la description du graphe ne sont pas complètes. Recopier et compléter le code des lignes 11 et 12.

On relève, pour chaque site, les arêtes qui arrivent sur lui et leur poids. `site4` reçoit un lien de `site1` et deux de `site2`. `site5` reçoit trois liens de `site1`, trois de `site3` et six de `site4`.

RÉPONSE À RETENIR

```
s4.predecesseurs = [(s1, 1), (s2, 2)]
s5.predecesseurs = [(s1, 3), (s3, 3), (s4, 6)]
```

Question 3.

Sujet p. 3

> Donner et expliquer la valeur de l'expression s2.successeurs[1][1].

`s2.successeurs` vaut `[(s1, 4), (s3, 5), (s4, 2)]`. L'indice 1 sélectionne son deuxième élément, le couple `(s3, 5)` ; le second indice 1 sélectionne le nombre de liens dans ce couple.

RÉPONSE À RETENIR

`s2.successeurs[1][1]` vaut **5** : il y a 5 liens de `site2` vers `site3`.

Question 4.

Sujet p. 2, 3

> Selon la définition donnée, la valeur de popularité d'un site correspond au nombre de fois où il est cité dans les autres sites, c'est-à-dire au nombre de liens hypertextes qui renvoient vers lui. Donner la valeur de popularité du site site1.

Les prédécesseurs de `site1` sont `site2` et `site4`. Ils ont respectivement 4 et 2 liens vers `site1`. La popularité est le nombre total de liens entrants, et non le nombre de sites prédécesseurs.

RÉPONSE À RETENIR

La popularité de `site1` est **4 + 2 = 6**.

FormaV - document d'entraînement non officiel

Question 5.

Sujet p. 2, 3

> Écrire le code de la méthode calculPopularite de la classe Site qui affecte à l'attribut popularite la valeur de popularité correspondante et renvoie cet attribut.

Chaque élément de `self.predecesseurs` contient un site prédécesseur et le nombre de liens de ce site vers l'objet courant. On additionne ces nombres, on affecte le total à `self.popularite`, puis on renvoie cet attribut.

RÉPONSE À RETENIR

```
def calculPopularite(self):
    self.popularite = 0
    for site, nombre_liens in self.predecesseurs:
        self.popularite += nombre_liens
    return self.popularite
```

Question 6.

Sujet p. 4

> Dans ce parcours, la liste Python nommée listeS est manipulée uniquement à l'aide d'appels de la forme listeS.append(sommet) et listeS.pop(0). Donner la structure de données correspondant à ces manipulations.

`append(sommet)` ajoute le sommet à la fin de la liste, tandis que `pop(0)` retire celui qui est en première position. Le premier sommet ajouté est donc le premier retiré : c'est le principe FIFO (« premier entré, premier sorti »).

RÉPONSE À RETENIR

La structure de données est une **file**.

Question 7.

Sujet p. 4

> Donner le nom de ce parcours de graphe.

Le parcours utilise une file : il traite d'abord le sommet de départ, puis les sommets découverts à distance croissante.

RÉPONSE À RETENIR

Il s'agit d'un **parcours en largeur** du graphe.

Question 8.

Sujet p. 3, 4

> La fonction parcoursGraphe renvoie une liste parcours. Indiquer la valeur renvoyée par l'appel de fonction parcoursGraphe(s1).

Le parcours commence par `s1`. Ses successeurs sont ajoutés dans l'ordre donné, soit `s3`, `s4`, puis `s5`. En examinant leurs successeurs, aucun nouveau sommet n'est découvert : ils sont déjà colorés en noir. Le sommet `s2` n'est pas accessible depuis `s1`.

RÉPONSE À RETENIR

La fonction renvoie `[s1, s3, s4, s5]`.

Question 9.

Sujet p. 5

> Recopier et compléter les lignes 6 et 7 de la fonction lePlusPopulaire, qui prend comme argument une liste non vide contenant des instances de la classe Site et renvoie le site le plus populaire.

Quand la popularité du site courant dépasse la plus grande valeur déjà rencontrée, il faut mettre à jour cette valeur maximale et mémoriser le site correspondant.

RÉPONSE À RETENIR

```
            maxPopularite = site.popularite
            siteLePlusPopulaire = site
```

Question 10.

Sujet p. 3, 4, 5

> Donner ce que renvoie la ligne de code suivante : lePlusPopulaire(parcoursGraphe(s1)).nom.

Le parcours renvoie `[s1, s3, s4, s5]`. Les popularités correspondantes sont 6, 12, 3 et 12. La fonction retient d'abord `s3`, qui atteint 12. Comme la condition est strictement supérieure, elle ne remplace pas `s3` par `s5`, qui a également une popularité de 12.

RÉPONSE À RETENIR

L'expression renvoie **`site3`**. `site3` et `site5` sont ex æquo avec une popularité de 12, et la fonction conserve le premier rencontré.

Question 11.

Sujet p. 2, 3, 4, 5

> On envisage d'utiliser l'ensemble des fonctions proposées pour rechercher le site le plus populaire parmi un très grand nombre de sites (quelques milliers). Expliquer si ce code est adapté à une telle quantité de sites à traiter et justifier votre réponse.

Le parcours en largeur examine les sommets et les arêtes, mais la file est ici représentée par une liste Python : chaque appel à `pop(0)` peut décaler les autres éléments. Dans le pire cas, le coût cumulé des retraits peut donc être de l'ordre de V^2, où `V` est le nombre de sommets. Une file adaptée, par exemple une deque avec retrait en tête efficace, permettrait un parcours en `O(V + E)`, où `E` est le nombre d'arêtes.

Le parcours lancé depuis un seul sommet ne visite que les sommets accessibles depuis celui-ci. Dans le graphe fourni, `s2` n'est pas accessible depuis `s1` et n'est donc pas traité ; sa popularité vaut toutefois 0 ici, ce qui ne change pas le maximum trouvé parmi les sites de cet exemple. Pour garantir une recherche sur tous les sites d'un graphe quelconque, il faudrait aussi lancer un parcours depuis chaque sommet encore blanc. Ainsi, le code peut fonctionner sur quelques milliers de sites, mais il n'est pas pleinement adapté tel quel à une recherche globale et à grande échelle.

RÉPONSE À RETENIR

Le code n'est pas pleinement adapté tel quel : `pop(0)` peut entraîner un coût quadratique et un parcours depuis `s1` ne couvre pas nécessairement tous les sites. Une file efficace et le traitement des sommets encore non visités sont nécessaires pour garantir une recherche globale plus efficace.

Exercice 2

6 points

Protocoles de routage, sécurité des communications et bases de données relationnelles

A

Question 1.

Sujet p. 6, 7

> Donner deux services rendus par un système de gestion de bases de données relationnelles.

Un SGBDR permet de stocker et d'organiser durablement des données dans des tables. Il permet aussi de consulter et de modifier ces données au moyen de requêtes. Il peut également vérifier des contraintes d'intégrité et gérer les accès.

> **RÉPONSE À RETENIR**
>
> Deux services rendus par un SGBDR sont le **stockage organisé des données** et leur **consultation ou modification par requêtes**.

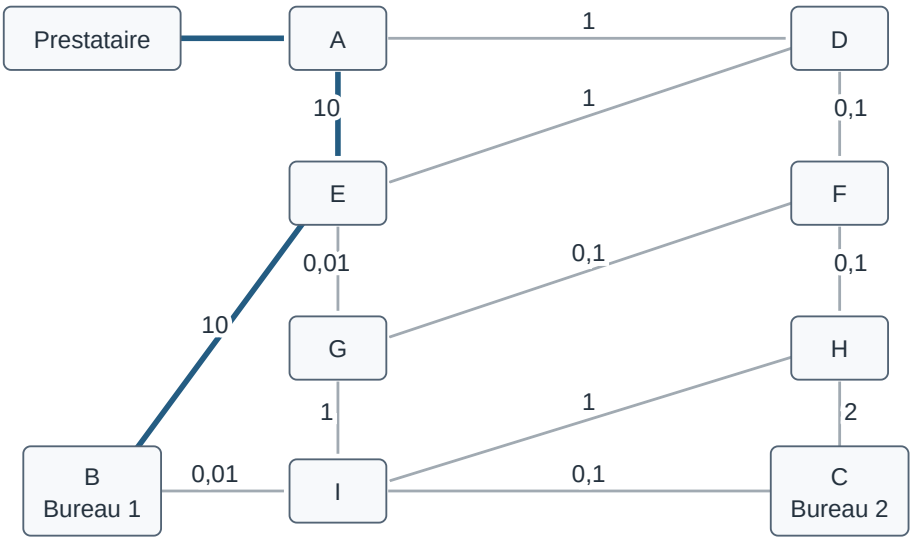
Question 2.

Sujet p. 7

> Le protocole RIP minimise le nombre de routeurs par lesquels les paquets transitent. Donner la route suivie par une requête issue du bureau numéro 1 jusqu'au prestataire si l'on utilise ce protocole.

RIP choisit un chemin qui minimise le nombre de routeurs traversés, sans tenir compte des coûts inscrits sur les liaisons. Depuis le bureau numéro 1, on entre dans le réseau par le routeur B. Le chemin B → E → A comporte deux liaisons entre routeurs et est le plus court en nombre de liaisons pour atteindre le routeur A du prestataire. Le lien entre A et le prestataire est ensuite emprunté.

Réseau de la figure 1 — route RIP depuis le bureau numéro 1



Les poids sont ceux de la figure 1. Le chemin mis en évidence minimise le nombre de liaisons entre routeurs ; RIP ne tient pas compte de leurs coûts.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Route : Bureau numéro 1 → B → E → A → Prestataire.

Question 3.

Sujet p. 7

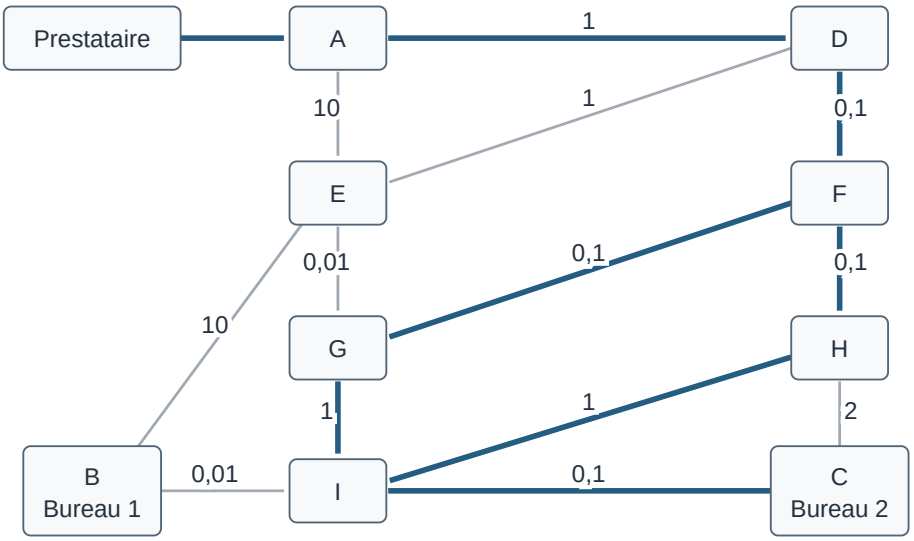
Le protocole OSPF minimise le coût du transit des paquets. Donner les deux routes que pourrait suivre une requête issue du bureau numéro 2 jusqu'au prestataire si l'on utilise ce protocole. Donner le coût de chaque route.

Avec OSPF, on compare la somme des coûts des liaisons. En partant du routeur C, les plus petites distances vers les routeurs utiles sont : C : 0 ; I : 0,1 ; H : 1,1 et G : 1,1 ; F : 1,2 ; D : 1,3 ; A : 2,3. Pour atteindre F, deux possibilités ont le même coût minimal : passer par G ou par H. Elles donnent donc les deux routes minimales suivantes :

- C → I → G → F → D → A : 0,1 + 1 + 0,1 + 0,1 + 1 = 2,3 ;
- C → I → H → F → D → A : 0,1 + 1 + 0,1 + 0,1 + 1 = 2,3.

Les routes passant par H directement depuis C coûtent déjà 2 avant de poursuivre vers A ; elles ne sont donc pas moins coûteuses. Le lien entre A et le prestataire n'a pas de coût indiqué sur la figure.

Réseau de la figure 1 — chemins OSPF depuis le bureau numéro 2



Les deux chemins mis en évidence partagent C–I et F–D–A. Ils se séparent entre I et F et ont chacun un coût total de 2,3.

RÉPONSE À RETENIR

Routes OSPF de coût minimal : Bureau numéro 2 → C → I → G → F → D → A → Prestataire, coût 2,3 ; ou Bureau numéro 2 → C → I → H → F → D → A → Prestataire, coût 2,3.

FormaV - document d'entraînement non officiel

B

Question 4.

Sujet p. 6, 7

> Expliquer pourquoi l'attribut id_client a été choisi comme clé primaire dans la relation clients.

Une clé primaire identifie de manière unique chaque ligne d'une relation : ses valeurs ne sont pas nulles et deux lignes ne peuvent pas avoir la même valeur de clé. L'attribut `id_client` joue ce rôle pour les clients. Un nom, un prénom ou une date de naissance ne conviendrait pas aussi bien, car plusieurs clients peuvent partager ces valeurs.

> **RÉPONSE À RETENIR**
>
> `id_client` est la clé primaire car il permet d'identifier chaque client de façon unique.

Question 5.

Sujet p. 6, 8

> Définir ce qu'est une clé étrangère. Donner la ou les clés étrangères de chaque relation qui en possède, en précisant la clé primaire qu'elles référencent.

Une clé étrangère est un attribut (ou un ensemble d'attributs) d'une relation qui référence la clé primaire d'une autre relation. Elle garantit que la valeur référencée correspond à une ligne existante dans la relation cible.

- Dans `reservations`, `id_client` référence `clients.id_client`.
- Dans `reservations`, `nom_croisiere` référence `croisieres.nom`.
- Dans `croisieres`, chacun des attributs `escale_1`, `escale_2`, `escale_3` et `escale_4` référence `villes.nom`.
- Les relations `clients` et `villes` ne montrent pas de clé étrangère dans le schéma fourni.

Le schéma indique toutefois `reservations.nom_croisiere : INT` alors que `croisieres.nom : TEXT`. Il y a donc une incohérence de types dans le schéma imprimé ; le lien de clé étrangère indiqué est néanmoins bien celui entre ces deux attributs.

> **RÉPONSE À RETENIR**
>
> Clés étrangères : `reservations.id_client → clients.id_client` ; `reservations.nom_croisiere → croisieres.nom` ; `croisieres.escale_1`, `escale_2`, `escale_3` et `escale_4 → villes.nom`.

Question 6.

Sujet p. 6, 8

> L'agence a obtenu l'autorisation de faire escale dans quatre nouvelles villes : Puerto saibo, Puerto kifecho, Puerto kifebo et Puerto repo. Elle souhaite créer une croisière nommée « Croisière Puerto » qui passera par ces villes. La requête proposée par l'IA insère dans la relation croisieres les valeurs « Croisière Puerto », « Puerto sebo », « Puerto kifecho », « Puerto kifebo » et « Puerto repo » pour les attributs nom, escale_1, escale_2, escale_3 et escale_4. Le SGBD affiche le message « FOREIGN KEY constraint failed ». Expliquer l'erreur commise et proposer une solution.

La première escale est mal orthographiée dans la requête proposée : l'énoncé donne `Puerto saibo`, tandis que la requête utilise `Puerto sebo`. De plus, les quatre valeurs d'escale sont des clés étrangères vers `villes.nom` : les quatre noms exacts doivent donc déjà être présents dans la relation `villes`. Il faut d'abord

y ajouter les quatre nouvelles villes, en renseignant leurs latitudes et longitudes réelles, puis insérer la croisière avec les noms exacts. Les coordonnées n'étant pas fournies, il ne faut pas les inventer.

Après l'ajout des villes dans `villes`, l'insertion de la croisière peut s'écrire ainsi :

```
INSERT INTO croisieres (nom, escale_1, escale_2, escale_3, escale_4)
VALUES ('Croisière Puerto', 'Puerto saibo', 'Puerto kifecho',
        'Puerto kifebo', 'Puerto repo');
```

RÉPONSE À RETENIR

Corriger `Puerto sebo` en `Puerto saibo`, vérifier que les quatre noms d'escale exacts figurent d'abord dans `villes`, puis insérer la croisière.

C

Question 7.

Sujet p. 6, 8

> Jean Barc, un allemand né le 29 juin 1972, demande un geste commercial en raison de sa fidélité à l'agence. Expliquer les deux requêtes SQL saisies par le gestionnaire : SELECT id FROM clients WHERE nom = 'Barc' AND prenom = 'Jean' AND date_naissance = '1972/06/29' AND pays = 'Allemagne'; puis SELECT id_reservation FROM reservations WHERE id_client = 1243;

La première requête cherche l'identifiant du client dont le nom est Barc, le prénom Jean, la date de naissance `1972/06/29` et le pays Allemagne. Les quatre conditions reliées par `AND` doivent être satisfaites simultanément. Cependant, le schéma désigne la clé de la relation `clients` par `id_client`, et non par `id` : telle qu'elle est écrite, la requête ne correspond donc pas au schéma. Il faut écrire `SELECT id_client FROM clients`.

La seconde requête sélectionne les identifiants des réservations dont l'attribut `id_client` vaut 1243. Elle suppose que 1243 est l'identifiant obtenu pour Jean Barc avec la première recherche corrigée. Aucun extrait de lignes des tables n'est fourni : on ne peut donc pas déterminer les résultats effectifs des deux requêtes.

RÉPONSE À RETENIR

La première requête vise à identifier Jean Barc, mais doit sélectionner `id_client` conformément au schéma. La seconde renvoie les `id_reservation` associés au client 1243.

Question 8.

Sujet p. 6, 8

> Écrire les deux requêtes de la question 7 sous la forme d'une requête unique.

On peut obtenir directement les réservations du client en joignant `clients` à `reservations` sur leur attribut commun `id_client`. Le filtre identifie ensuite Jean Barc avec les quatre caractéristiques données. Comme le schéma nomme l'identifiant du client `id_client`, la requête combinée correcte est :

```sql
SELECT r.id_reservation
FROM clients AS c
JOIN reservations AS r ON c.id_client = r.id_client
WHERE c.nom = 'Barc'
  AND c.prenom = 'Jean'
  AND c.date_naissance = '1972/06/29'
  AND c.pays = 'Allemagne';
```

Aucun extrait de lignes n'étant fourni, les identifiants de réservation effectivement renvoyés ne sont pas déterminables.

> **RÉPONSE À RETENIR**
>
> Cette requête unique sélectionne les identifiants des réservations de Jean Barc, en joignant les deux relations sur `id_client`.

Question 9.

Sujet p. 6, 8

> Un client souhaite modifier sa réservation d'identifiant 20456. Il souhaite remplacer la croisière de cette réservation par la toute dernière offre de l'agence : la Croisière Puerto. Écrire une requête SQL qui permet de mettre à jour la base de données pour lui donner satisfaction.

Il faut mettre à jour la ligne de la relation `reservations` dont `id_reservation` vaut 20456, en remplaçant son attribut `nom_croisiere` par le nom de la nouvelle offre. La croisière doit déjà être enregistrée dans `croisieres` pour que la référence puisse être respectée.

```sql
UPDATE reservations
SET nom_croisiere = 'Croisière Puerto'
WHERE id_reservation = 20456;
```

> **RÉPONSE À RETENIR**
>
> La requête met la réservation 20456 à la croisière `Croisière Puerto`, sous réserve que cette croisière ait déjà été ajoutée à la base.

Question 10.

Sujet p. 6, 8

> Donner une requête SQL permettant d'obtenir les noms, prénoms et dates de naissance des clients ayant choisi la croisière nommée Croisière Piano ou celle nommée Croisière Puerto.

On relie `clients` à `reservations` par `id_client`, puis `reservations` à `croisieres` par le nom de la croisière. Le filtre conserve les réservations correspondant à l'un ou l'autre des deux noms demandés. `DISTINCT` évite de répéter un client ayant plusieurs réservations correspondantes.

```sql
SELECT DISTINCT c.nom, c.prenom, c.date_naissance
FROM clients AS c
JOIN reservations AS r ON c.id_client = r.id_client
JOIN croisieres AS cr ON r.nom_croisiere = cr.nom
WHERE cr.nom = 'Croisière Piano'
   OR cr.nom = 'Croisière Puerto';
```

Aucun extrait de lignes n’étant fourni, on peut établir la requête mais pas les noms, prénoms et dates de naissance qui apparaîtraient dans son résultat. Le schéma comporte par ailleurs l’incohérence de types entre `reservations.nom_croisiere` et `croisieres.nom` signalée à la question 5.

> **RÉPONSE À RETENIR**
>
> Cette requête renvoie les noms, prénoms et dates de naissance des clients ayant réservé `Croisière Piano` ou `Croisière Puerto`, sans doublonner un client.

Exercice 3

8 points

Programmation orientée objet, arbres binaires de recherche et récursivité

A

Question 1.

Sujet p. 9, 10

> Écrire l'instruction permettant d'instancier l'objet chien40 caractérisant le chien ayant le numéro d'inscription 40. D'après le tableau, il s'agit de Duke, de rôle « wheel dog », dont l'id_proprietaire est 10.

Le constructeur de `Chien` attend dans l'ordre l'identifiant, le nom, le rôle et l'identifiant du propriétaire. Le tableau indique pour le chien 40 : Duke, de rôle `wheel dog`, appartenant à l'équipe 10.

RÉPONSE À RETENIR

```
chien40 = Chien(40, 'Duke', 'wheel dog', 10)
```

Question 2.

Sujet p. 9, 10

> Selon l'état de fatigue de ses chiens ou le profil de l'étape, le musher peut décider de changer le rôle des chiens dans l'attelage. Recopier et compléter la méthode changer_role de la classe Chien.

La méthode modifie l'attribut `role` de l'objet courant en lui affectant la valeur fournie par le paramètre `nouveau_role`.

RÉPONSE À RETENIR

```
def changer_role(self, nouveau_role):
    """Change le rôle du chien avec la valeur passée en
    paramètre."""
    self.role = nouveau_role
```

Question 3.

Sujet p. 10

> Le propriétaire de Duke décide de lui attribuer le rôle de 'leader'. Écrire l'instruction permettant d'effectuer cette modification.

L'objet représentant Duke est `chien40`. On utilise sa méthode `changer_role` avec la chaîne de caractères correspondant au nouveau rôle.

RÉPONSE À RETENIR

```
chien40.changer_role('leader')
```

B

Question 4.

Sujet p. 10, 11

> Recopier et compléter la méthode retirer_chien ayant pour paramètre numero, un entier correspondant au numéro attribué au chien lors de l'inscription, et permettant de mettre à jour l'attribut liste_chiens après retrait du chien dont la valeur de l'attribut id_chien est numero.

On parcourt les chiens de `liste_chiens` et on compare leur attribut `id_chien` au numéro recherché. Lorsque le chien est trouvé, on supprime l'élément correspondant puis on arrête la méthode. Si aucun numéro ne correspond, la liste reste inchangée.

RÉPONSE À RETENIR

```
def retirer_chien(self, numero):
    for i, chien in enumerate(self.liste_chiens):
        if chien.id_chien == numero:
            del self.liste_chiens[i]
            return
```

Par exemple, dans la liste des chiens 42, 44, 45 et 46, l'appel avec `numero = 46` enlève Helka et laisse les chiens 42, 44 et 45.

Question 5.

Sujet p. 10, 11

> En vous aidant de la méthode précédente, écrire l'instruction qui permet de retirer Helka, chien numéro 46, de l'attelage de l'équipe eq11.

Helka a pour identifiant 46. On transmet donc ce numéro à la méthode de l'équipe `eq11`.

RÉPONSE À RETENIR

```
eq11.retirer_chien(46)
```

Question 6.

Sujet p. 11, 12

> Indiquer le résultat renvoyé par l'appel convert('4h36').

Pour `'4h36'`, les caractères aux indices 0, 2 et 3 sont respectivement `'4'`, `'3'` et `'6'`. La fonction convertit donc 4 heures et 36 minutes en heures décimales.

RÉPONSE À RETENIR

`convert('4h36')` renvoie **4.6**, car `4 + 36 / 60 = 4.6`.

Question 7.

Sujet p. 10, 11, 12

> Écrire une fonction temps_course qui prend pour paramètre equipe, de type Equipe, et qui renvoie un nombre flottant correspondant au cumul des temps de l'équipe equipe à l'issue des 9 étapes de la course. La classe Equipe dispose d'un attribut liste_temps.

Chaque durée de `equipe.liste_temps` est convertie en heures décimales par `convert`, puis ajoutée au cumul. L'initialisation à `0.0` donne un cumul de type flottant.

RÉPONSE À RETENIR

```
def temps_course(equipe):
    total = 0.0
    for temps in equipe.liste_temps:
        total = total + convert(temps)
    return total
```

Avec la liste d'exemple fournie, la somme des heures est 35 et celle des minutes est 362 minutes, soit 6 h 02. Le cumul vaut donc 41 h 02, c'est-à-dire environ `41.03333333333333` heures.

C

Question 8.

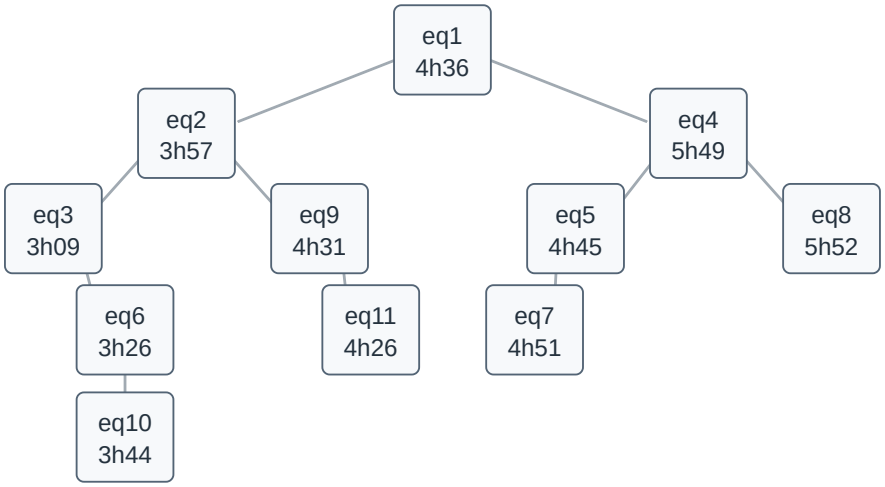
Sujet p. 12

> Dans l'arbre binaire de recherche représenté, les nœuds eq1 et eq2 ont été insérés. Recopier et compléter cet arbre en insérant les 9 nœuds manquants, dans l'ordre eq3 à eq11. Les temps relevés à l'issue de la première étape sont : eq1 4h36, eq2 3h57, eq3 3h09, eq4 5h49, eq5 4h45, eq6 3h26, eq7 4h51, eq8 5h52, eq9 4h31, eq10 3h44 et eq11 4h26. Dans l'ABR, les équipes du sous-arbre gauche sont strictement plus rapides que le nœud et celles du sous-arbre droit sont moins rapides ou à égalité avec lui.

On insère les équipes dans l'ordre indiqué. Un temps inférieur à celui du nœud courant conduit à gauche ; un temps supérieur conduit à droite. Par exemple, `eq6` (3 h 26) va à gauche de `eq1`, puis à gauche de `eq2`, puis à droite de `eq3`. Les chemins et emplacements de toutes les insertions sont indiqués ci-dessous.

Équipe insérée	Chemin suivi	Emplacement
eq3 (3 h 09)	eq1 gauche, eq2 gauche	gauche de eq2
eq4 (5 h 49)	eq1 droite	droite de eq1
eq5 (4 h 45)	eq1 droite, eq4 gauche	gauche de eq4
eq6 (3 h 26)	eq1 gauche, eq2 gauche, eq3 droite	droite de eq3
eq7 (4 h 51)	eq1 droite, eq4 gauche, eq5 droite	droite de eq5
eq8 (5 h 52)	eq1 droite, eq4 droite	droite de eq4
eq9 (4 h 31)	eq1 gauche, eq2 droite	droite de eq2
eq10 (3 h 44)	eq1 gauche, eq2 gauche, eq3 droite, eq6 droite	droite de eq6
eq11 (4 h 26)	eq1 gauche, eq2 droite, eq9 gauche	gauche de eq9

ABR des temps de la première étape



Une équipe plus rapide que le nœud est à gauche ; une équipe moins rapide est à droite.

> **RÉPONSE À RETENIR**
>
> L'arbre complété, en conservant les positions initiales de `eq1` et `eq2`, est le suivant.

Question 9.

Sujet p. 12

> Indiquer quel parcours d'arbre permet d'obtenir la liste des équipes classées de la plus rapide à la plus lente.

Les équipes les plus rapides se trouvent dans les sous-arbres gauches. Pour les énumérer par temps croissant, on visite donc d'abord le sous-arbre gauche, puis le nœud, puis le sous-arbre droit.

> **RÉPONSE À RETENIR**
>
> Il faut effectuer un **parcours infixe** : gauche, racine, droite.

Question 10.

Sujet p. 13

> Expliquer en quoi la fonction inserer est une fonction récursive.

Une fonction est récursive lorsqu'elle s'appelle elle-même. Dans `inserer`, si le fils où l'équipe doit être ajoutée existe déjà, la fonction se rappelle sur ce sous-arbre. Chaque appel descend d'un niveau. Dans un arbre fini, on atteint un fils vide : on y crée le nouveau nœud, ce qui termine l'insertion.

> **RÉPONSE À RETENIR**
>
> `inserer` est donc récursive parce que son code appelle de nouveau `inserer` sur le sous-arbre approprié.

Question 11.

Sujet p. 13

> Recopier et compléter les lignes 6, 8 et 13 de la fonction inserer.

Si le fils nécessaire est absent, on y crée un nœud contenant `eq`. Sinon, on poursuit récursivement dans ce sous-arbre. Par exemple, `eq10` (3 h 44) suit le chemin `eq1` gauche, `eq2` gauche, `eq3` droite, `eq6` droite ; on le place donc à droite de `eq6`.

RÉPONSE À RETENIR

```
def inserer(arb, eq):
    """Insertion d'une équipe à sa place dans un ABR contenant
    au moins un noeud."""
    if convert(eq.temps_etape) < convert(arb.racine.temps_etape):
        if arb.gauche is None:
            arb.gauche = Noeud(eq)
        else:
            inserer(arb.gauche, eq)
    else:
        if arb.droit is None:
            arb.droit = Noeud(eq)
        else:
            inserer(arb.droit, eq)
```

Le cas d'égalité passe à droite, conformément à la règle de l'énoncé.

Question 12.

Sujet p. 13

> Recopier et compléter les lignes 3 et 5 de la fonction est_gagnante, qui prend en paramètre un ABR arbre, de type Noeud, et renvoie le nom de l'équipe ayant gagné l'étape.

Le gagnant est l'équipe dont le temps est le plus petit. Dans cet ABR, elle se trouve au nœud le plus à gauche. Si le fils gauche est absent, le nœud courant contient le gagnant ; sinon, on poursuit la recherche dans le sous-arbre gauche.

RÉPONSE À RETENIR

```
def est_gagnante(arbre):
    if arbre.gauche == None:
        return arbre.racine.nom_equipe
    else:
        return est_gagnante(arbre.gauche)
```

Dans l'arbre construit à la question 8, le nœud le plus à gauche est `eq3`, avec un temps de 3 h 09 ; la fonction renvoie l'attribut `nom_equipe` de ce nœud.

D

Question 13.

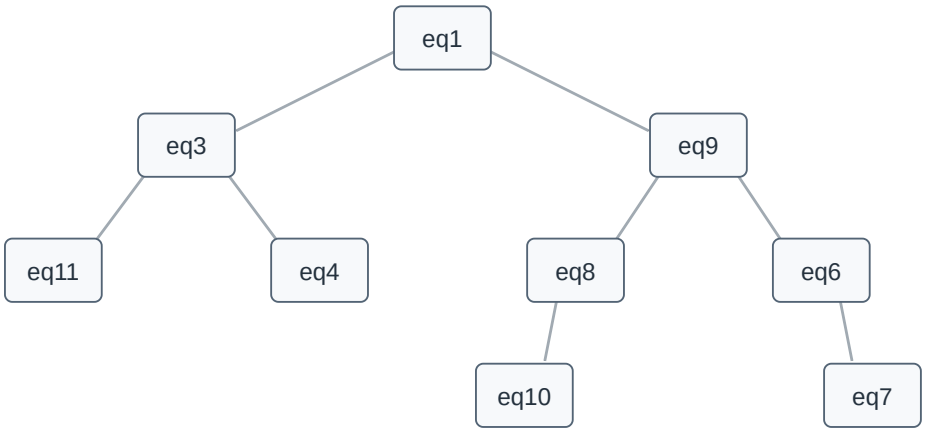
Sujet p. 14

> Dessiner le nouvel arbre de recherche a_final obtenu après suppression des équipes eq2 et eq5 dans l'ABR correspondant au classement général. Pour supprimer un nœud, s'il s'agit d'une feuille, le retirer de l'arbre ; s'il n'a qu'un seul fils, relier ce fils au père du nœud puis supprimer le nœud ; s'il a deux fils, le remplacer par son

successeur, qui est le minimum de ses descendants droits.

Dans l'arbre de la figure 2, `eq2` n'a qu'un fils, `eq3` : après sa suppression, `eq3` devient le fils gauche de `eq1`. Le nœud `eq5` a deux fils. Son successeur est le minimum de son sous-arbre droit, soit `eq9`, le fils gauche de `eq6`. On remplace donc `eq5` par `eq9`, puis on supprime l'ancien nœud `eq9`, qui est une feuille.

a_final après suppression de eq2 et eq5



eq2 est remplacée par son unique fils eq3 ; eq5 est remplacée par son successeur eq9.

RÉPONSE À RETENIR

Le nouvel arbre `a_final` est représenté ci-dessous.

Question 14.

Sujet p. 14, 15

Écrire le code de la fonction rechercher(arbre, equipe). Elle prend en paramètre un ABR non vide de type Noeud représentant le classement général et une équipe de type Equipe ; elle renvoie True si equipe est un nœud de l'arbre et False sinon. Pour a_final, obtenu après suppression des équipes 2 et 5 à la question 13, rechercher(a_final, eq1) renvoie True et rechercher(a_final, eq2) renvoie False.

La recherche compare les objets par leur cumul des neuf temps, qui sert de critère d'ordre dans l'ABR. Si l'équipe recherchée est celle du nœud courant, on renvoie `True`. Sinon, un cumul inférieur conduit à gauche et un cumul supérieur ou égal conduit à droite, conformément à la règle d'insertion des égalités. Lorsqu'on atteint un sous-arbre vide, l'équipe n'est pas présente. Le test `is` vérifie qu'il s'agit bien du même objet `Equipe`.

RÉPONSE À RETENIR

```
def rechercher(arbre, equipe):
    """Renvoie True si equipe est un nœud de l'ABR, False sinon."""
    if arbre is None:
        return False
    if arbre.racine is equipe:
        return True
    if temps_course(equipe) < temps_course(arbre.racine):
        return rechercher(arbre.gauche, equipe)
    else:
        return rechercher(arbre.droit, equipe)
```

Pour `a_final`, la recherche de `eq1` réussit dès la racine et renvoie `True`. L'équipe `eq2` ayant été supprimée, sa recherche aboutit à un sous-arbre vide et renvoie `False`, comme indiqué dans le sujet.

Sujet : https://www.sujetdebac.fr/annales-pdf/2024/spe-numerique-informatique-2024-metropole-1-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.