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 2

Code sujet **25-NSIJ2JA1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Programmation orientée objet et algorithmique : manipulation de dates
11 réponses • sujet p. 2–5

Exercice 2 — Programmation Python et gestion de l’ordre d’exécution des processus
10 réponses • sujet p. 6–8

Exercice 3 — Bases de données, arbres de décision, programmation Python et intégrité des données
14 réponses • sujet p. 9–14

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

Programmation orientée objet et algorithmique : manipulation de dates

Partie A : Accès et modification des données

Question 1.

Sujet p. 2

> Le constructeur de la classe Date prend en paramètres trois entiers représentant le jour, le mois et l'année, puis les affecte respectivement aux attributs jour, mois et annee. Recopier et compléter les lignes 3 à 5 du code fourni.

Le constructeur reçoit les paramètres `jour`, `mois` et `annee`. Chaque paramètre est affecté à l'attribut correspondant de l'instance avec le préfixe `self.`.

RÉPONSE À RETENIR

```
self.jour = jour
self.mois = mois
self.annee = annee
```

Question 2.

Sujet p. 2

> Indiquer à quelle date correspond l'instance de la classe Date suivante : d = Date(1, 5, 2000).

Les paramètres de `Date` sont donnés dans l'ordre jour, mois, année. L'instance représente donc le premier jour du cinquième mois de l'année 2000.

RÉPONSE À RETENIR

1er mai 2000.

Question 3.

Sujet p. 2

> Écrire le code permettant de créer une instance d de la classe Date qui représente la date du 19 juin 2024.

On fournit à `Date` les trois entiers correspondant au jour, au mois (juin est le mois 6) et à l'année.

RÉPONSE À RETENIR

```
d = Date(19, 6, 2024)
```

Question 4.

Sujet p. 2

> La méthode get_annee renvoie la valeur de l'attribut annee. Recopier et compléter les lignes 15 et 16 du code fourni.

La méthode doit renvoyer l'attribut `annee` de l'objet courant ; on y accède avec `self.annee`.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def get_annee(self):
    return self.annee
```

Question 5.

Sujet p. 3

> La méthode set_mois modifie l'attribut mois en lui affectant la valeur passée en argument. Recopier et compléter les lignes 21 et 22 du code fourni.

La valeur reçue en paramètre doit être affectée à l'attribut `mois` de l'objet courant.

RÉPONSE À RETENIR

```
def set_mois(self, mois):
    self.mois = mois
```

Question 6.

Sujet p. 2, 3

> La méthode est_bissextile utilise uniquement l'attribut annee et renvoie True si l'année de l'instance courante est bissextile et False sinon. Compléter la méthode __init__ pour ajuster le nombre de jours par mois pour les années bissextiles : recopier et compléter les lignes 7 et 8, à la suite de la liste nb_jours_par_mois.

L'année doit être enregistrée avant l'appel à `est_bissextile`, puisque cette méthode teste l'attribut `annee`. Les mois sont numérotés à partir de 1, tandis que les indices de liste commencent à 0 : février est donc à l'indice 1.

RÉPONSE À RETENIR

```
if self.est_bissextile():
    self.nb_jours_par_mois[1] = 29
```

Par exemple, pour une instance d'année 2024, l'appel renvoie vrai et la case d'indice 1 passe de 28 à 29. Pour 2001, elle reste à 28.

Partie B : sur l'année de l'instance courante

Question 7.

Sujet p. 3

> Une année est bissextile si elle est divisible par 4 mais pas par 100, ou si elle est divisible par 400. Écrire le code de la méthode est_bissextile. On rappelle qu'un entier a est divisible par l'entier n si a % n == 0.

Une année est bissextile si elle est divisible par 400, ou si elle est divisible par 4 sans être divisible par 100. Le `or` sépare ces deux cas ; dans le second cas, le `and` impose les deux conditions.

RÉPONSE À RETENIR

```
def est_bissextile(self):
    return (self.annee % 400 == 0) or (
        self.annee % 4 == 0 and self.annee % 100 != 0
    )
```

Trace sur des années du sujet : pour 2024, `2024 % 4 == 0` et `2024 % 100 != 0`, donc la méthode renvoie `True`. Pour 2001, `2001 % 4 != 0`, donc elle renvoie `False`.

Question 8.

Sujet p. 3

Indiquer quel sera l'affichage en console après l'exécution des deux instructions suivantes : d1 = Date(20, 3, 2001), puis d1.nb_jours_passes(), en utilisant la méthode nb_jours_passes fournie.

La méthode commence par le jour du mois, soit 20. Elle ajoute ensuite les jours des mois précédents, janvier et février : `31 + 28`. Mars 2001 n'ajoute pas de jour bissextile.

RÉPONSE À RETENIR

Le calcul est `20 + 28 + 31 = 79`. L'affichage de `d1.nb_jours_passes()` est donc **79**.

Question 9.

Sujet p. 4

La méthode nb_jours_restants renvoie le nombre de jours restants dans l'année de l'instance courante, soit 366 ou 365 moins le nombre de jours déjà passés, selon que l'année est bissextile ou non. Recopier et compléter le script fourni de cette méthode.

On choisit d'abord la durée de l'année : 365 jours par défaut, puis 366 si l'année de l'objet est bissextile. Il faut ensuite soustraire le nombre de jours déjà passés, renvoyé par `nb_jours_passes`.

RÉPONSE À RETENIR

```
def nb_jours_restants(self):
    j = 365
    if self.est_bissextile():
        j = 366
    return j - self.nb_jours_passes()
```

Par exemple, pour le 20 mars 2001, le nombre de jours passés est 79 et l'année compte 365 jours : la méthode renvoie `365 - 79 = 286`.

Partie C : entre deux dates

Question 10.

Sujet p. 4

Pour les instances d1 = Date(15, 6, 2024), d2 = Date(15, 6, 2024), d3 = Date(15, 7, 2024), d4 = Date(15, 6, 2025) et d5 = Date(15, 6, 2022), indiquer les affichages en console après l'exécution de chacune des instructions d1.nb_jours_depuis(d2), d1.nb_jours_depuis(d3), d1.nb_jours_depuis(d4) et d1.nb_jours_depuis(d5). On précise que l'année 2024 est bissextile. La méthode nb_jours_depuis fournie renvoie le nombre de jours écoulés entre la

> date de other et celle de l'instance courante.

La méthode renvoie `-1` si la date passée en paramètre est postérieure à celle de l'objet courant, et `0` si les deux dates sont identiques. Sinon, elle additionne les jours passés dans l'année de `self`, les jours restants dans l'année de `other` et, s'il y en a, les années entières intermédiaires.

Pour vérifier le dernier cas, le 15 juin 2024 est le 167e jour de l'année bissextile 2024, et le 15 juin 2022 est le 166e jour d'une année de 365 jours. La méthode calcule donc `167 + (365 - 166) + 365 = 731`.

RÉPONSE À RETENIR

Instruction	Résultat	Justification
`d1.nb_jours_depuis(d2)`	`0`	Les dates sont identiques.
`d1.nb_jours_depuis(d3)`	`-1`	Le 15 juillet 2024 est postérieur au 15 juin 2024.
`d1.nb_jours_depuis(d4)`	`-1`	L'année 2025 est postérieure à l'année 2024.
`d1.nb_jours_depuis(d5)`	`731`	`199 + 365 + 167 = 731` : jours restants après le 15 juin 2022, année 2023 entière, puis jours passés jusqu'au 15 juin 2024.

Question 11.

Sujet p. 4, 5

> Le timestamp est le nombre de secondes qui se sont écoulées depuis le 1er janvier 1970 à 00h00. Recopier et compléter le code de la méthode timestamp, qui renvoie ce nombre de secondes. Le sujet donne l'exemple du 01/01/2024 à 00:00:00, qui correspond au timestamp 1704063600.

La date de référence à construire est le 1er janvier 1970. La méthode `nb_jours_depuis` donne le nombre de jours entre cette date et l'instance courante ; on convertit ces jours en secondes en multipliant par `24 * 3600`.

Il y a une incohérence entre l'exemple fourni et ce calcul par jours entiers : du 1er janvier 1970 au 1er janvier 2024, il s'écoule 19 723 jours, soit `19 723 * 86 400 = 1 704 067 200` secondes. La valeur annoncée dans le sujet, `1704063600`, est inférieure de 3 600 secondes. Elle peut correspondre à une convention de fuseau horaire, mais aucun fuseau n'est précisé. Le code demandé, fondé uniquement sur les dates et la conversion en jours indiquée, est celui ci-dessous.

RÉPONSE À RETENIR

```
def timestamp(self):
    d = Date(1, 1, 1970)
    return self.nb_jours_depuis(d) * 24 * 3600
```

Exercice 2

6 points

Programmation Python et gestion de l'ordre d'exécution des processus

Question 1.

Sujet p. 6

> On souhaite élaborer un programme système permettant de gérer l'ordre d'exécution des processus sur le processeur. Donner le nom de ce type de programme.

Le programme qui décide quel processus obtient le processeur et dans quel ordre est un **ordonnanceur de processus**, ou ordonnanceur CPU.

> **RÉPONSE À RETENIR**
>
> Il s'agit d'un ordonnanceur de processus.

Question 2.

Sujet p. 6

> Donner les différents états possibles d'un processus.

Dans le modèle usuel, un processus peut être **créé**, **prêt** (en attente du processeur), **en cours d'exécution**, **bloqué** (en attente d'un événement ou d'une ressource) ou **terminé**. Certains modèles simplifiés ne distinguent pas l'état créé.

> **RÉPONSE À RETENIR**
>
> Les états usuels sont : créé, prêt, en cours d'exécution, bloqué et terminé.

Question 3.

Sujet p. 6, 7

> Parmi les propositions suivantes, donner la structure la plus adaptée pour stocker les processus d'une même priorité : proposition 1, Liste ; proposition 2, File ; proposition 3, Pile.

La structure adaptée est une file : elle permet de sélectionner les processus d'une même priorité selon le principe FIFO, premier entré, premier sorti.

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

Question 4.

Sujet p. 7

> Pour représenter un processus, la classe Processus possède les variables d'instance PID (identifiant du processus), priorite, temps_utilisation sur le CPU et temps_CPU (temps nécessaire à son exécution). Compléter le constructeur de la classe Processus dans le code fourni.

Le constructeur reçoit les paramètres nommés `PID`, `priorite` et `temps_CPU`. Il mémorise l'identifiant, la priorité et le temps nécessaire, puis initialise le temps d'utilisation à 0.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
class Processus:
    def __init__(self, PID, priorite, temps_CPU):
        self.priorite = priorite
        self.PID = PID
        self.temps_utilisation = 0
        self.temps_CPU = temps_CPU
```

Question 5.

Sujet p. 6, 7

On considère P1 = Processus(PID=1, priorite=0, temps_CPU=10), P2 = Processus(PID=2, priorite=0, temps_CPU=7) et P3 = Processus(PID=3, priorite=0, temps_CPU=5), avec liste_files = [[P3, P2, P1], [], []]. Compléter la simulation des cycles 3, 4 et 5, où la variable CPU contient le processus en cours d'exécution. Les cycles 1 et 2 sont indiqués dans le sujet.

Dans les états fournis, `P1` est choisi dans `[P3, P2, P1]` : le prochain processus à sortir est donc à droite. On utilise ainsi `pop()` pour retirer un processus et on insère à gauche les processus qui rejoignent une file, afin de respecter l'ordre FIFO.

Au cycle 3, `P2` utilise une unité de temps CPU. `P3` attend en priorité 0 ; `P2` est donc relégué en priorité 1, puis `P3` est élu. La file de priorité 1 contient alors `[P2, P1]`. Au cycle 4, `P3` est en priorité 0 et les processus en attente sont en priorité 1 : aucun n'a une priorité supérieure ou égale à la sienne. `P3` continue donc, et sa valeur de priorité passe à 1. Au cycle 5, les processus `P2` et `P1` attendent en priorité 1 : `P3` est relégué en priorité 2, et `P1`, en tête de file selon la convention retenue, est élu.

En appliquant les règles écrites dans l'énoncé, la troisième file est vide au cycle 4. Le dernier emplacement représenté par `[...]` dans le modèle du cycle 4 ne peut donc pas être non vide tout en conservant `CPU=P3`.

RÉPONSE À RETENIR

Cycle	CPU	liste_files
3	P3	[[], [P2, P1], []]
4	P3	[[], [P2, P1], []]
5	P1	[[], [P2], [P3]]

Question 6.

Sujet p. 6, 7, 8

On dispose d'un processus qui nécessite un temps d'utilisation de 1000 pour terminer, et d'un nombre important de processus dont le temps d'utilisation pour terminer est de 4, chaque processus terminé étant remplacé par un nouveau processus similaire. Expliquer pourquoi le processus long risque de ne jamais terminer avec le programme de gestion de l'ordre des processus décrit, en indiquant notamment sa priorité au bout de quelques temps.

Le processus long est relégué d'un niveau chaque fois qu'il utilise le processeur alors que des processus de priorité supérieure ou égale attendent. Comme une valeur numérique plus grande représente une priorité moins élevée, sa priorité peut passer de 0 à 1, puis 2, puis continuer à augmenter. Les processus courts nouvellement créés, eux, entrent toujours en priorité 0. Avec leur renouvellement continu, ils peuvent donc être choisis avant le processus long, qui risque d'être affamé et de ne jamais obtenir les 1000 unités de temps CPU nécessaires.

RÉPONSE À RETENIR

Sans remontée de priorité liée à l'attente, la valeur numérique de la priorité du processus long peut augmenter indéfiniment, tandis que les nouveaux processus restent en priorité 0. Le processus long risque donc de ne jamais terminer.

Question 7.

Sujet p. 8

Pour régler ce phénomène, on ajoute au processus la variable d'instance temps_d_attente et on définit une constante Max_Temps, correspondant au temps maximum qu'un processus attend avant de remonter sa priorité. À chaque cycle, temps_d_attente augmente et, si sa valeur dépasse Max_Temps, sa priorité augmente. Expliquer pourquoi le processus qui nécessite un temps long d'utilisation du CPU ne risque plus de ne jamais terminer avec ce nouveau programme de gestion.

Le temps d'attente du processus long augmente à chaque cycle. Lorsque le seuil `Max_Temps` est dépassé, sa priorité est améliorée : sa valeur numérique diminue. Il finit ainsi par atteindre une priorité suffisante pour être sélectionné avant les processus moins prioritaires. Cette conclusion suppose que les remontées de priorité sont bien appliquées et qu'un processus ne peut pas être indéfiniment dépassé par d'autres processus de même priorité.

RÉPONSE À RETENIR

La remontée de priorité déclenchée par l'attente empêche le processus long d'être repoussé indéfiniment par les nouveaux processus en priorité 0 : il finit par être élu et peut terminer.

Question 8.

Sujet p. 8

Écrire une fonction meilleur_priorite(liste_files) qui renvoie None s'il n'y a plus de processus et, dans le cas contraire, la priorité de l'un des processus les plus prioritaires de la liste des files d'attente. Dans l'exemple liste_files = [[], [p2], [p3, p1]], la fonction renvoie 1.

La priorité la plus élevée correspond au plus petit indice de file. On parcourt donc les files dans l'ordre et on renvoie l'indice de la première file non vide. S'il n'y en a aucune, on renvoie `None`. La fonction examine au plus autant de files que la longueur de `liste_files`.

Dans l'exemple, la file d'indice 0 est vide et celle d'indice 1 contient `p2` : la fonction renvoie 1.

RÉPONSE À RETENIR

```
def meilleur_priorite(liste_files):
    for i in range(len(liste_files)):
        if liste_files[i]:
            return i
    return None
```

Pour l'exemple fourni, le résultat est `1`. Si toutes les files sont vides, le résultat est `None`.

Question 9.

Sujet p. 8

Écrire une fonction prioritaire(liste_files) qui renvoie None si aucune des files d'attente de la liste ne contient un processus et qui renvoie sinon l'un des processus parmi les plus prioritaires. La fonction doit supprimer le processus choisi de la file d'attente dans laquelle il se trouvait. On pourra utiliser liste.pop(i) pour renvoyer l'élément

> à la position i tout en le supprimant de la liste.

On cherche d'abord la file non vide de meilleure priorité. S'il n'y en a pas, on renvoie `None`. Sinon, on retire et renvoie le prochain processus de cette file. La simulation donnée impose de choisir à droite : on emploie donc `pop()`. La recherche parcourt au plus toutes les files ; le retrait à droite d'une liste Python est en temps constant.

Exemple avec les files `[[], [p2], [p3, p1]]` : la fonction renvoie `p2` et modifie les files en `[[], [], [p3, p1]]`.

RÉPONSE À RETENIR

```
def prioritaire(liste_files):
    i = meilleur_priorite(liste_files)
    if i is None:
        return None
    return liste_files[i].pop()
```

Question 10.

Sujet p. 6, 8

> Écrire une fonction gerer(p, liste_files) qui récupère le processus p en cours d'exécution ainsi que la liste des files d'attente liste_files, et qui implémente le programme donné au début de l'énoncé pour gérer les processus.

La fonction renvoie le processus qui occupera le CPU après le traitement du cycle, ou `None` si le CPU est libre. Si aucun processus n'est en cours, elle élit le processus prioritaire en attente. Si le processus courant a terminé, elle le retire du processeur. Sinon, elle augmente son temps d'utilisation d'une unité. Lorsqu'un processus de priorité supérieure ou égale attend, le processus courant est relégué d'un niveau, remis dans sa file, puis remplacé par le processus le plus prioritaire en attente. Sinon, il continue sur le CPU, avec une valeur numérique de priorité augmentée de 1.

La condition `meilleure <= p.priorite` traduit bien la règle : une valeur numérique plus petite correspond à une priorité plus élevée. Les files manquantes sont créées si nécessaire lorsque la priorité du processus relégué dépasse les indices existants. Les processus remis en attente sont insérés à gauche et retirés à droite, conformément à la simulation.

Trace avec les données de la question 5 : CPU libre, on élit `P1` ; au cycle suivant, `P1` est relégué et `P2` est élu ; ensuite `P2` est relégué et `P3` est élu. `P3` continue un cycle et passe en priorité 1, puis est relégué en priorité 2 ; `P1` est alors élu.

RÉPONSE À RETENIR

```
def gerer(p, liste_files):
    if p is None:
        return prioritaire(liste_files)

    if p.temps_utilisation >= p.temps_CPU:
        return None

    p.temps_utilisation += 1
    meilleure = meilleur_priorite(liste_files)

    if meilleure is not None and meilleure <= p.priorite:
        p.priorite += 1
        while len(liste_files) <= p.priorite:
            liste_files.append([])
        liste_files[p.priorite].insert(0, p)
        return prioritaire(liste_files)

    p.priorite += 1
    return p
```

Après sélection initiale de `P1`, la trace est : `P2` élu avec les files `[[P3], [P1], []]` ; puis `P3` élu avec `[[], [P2, P1], []]` ; `P3` continue en priorité 1 avec les mêmes files ; enfin `P1` est élu avec `[[], [P2], [P3]]`.

Exercice 3

8 points

Bases de données, arbres de décision, programmation Python et intégrité des données

Partie A

Question 1.

Sujet p. 9, 10

> À partir de la table Patient, écrire une requête SQL permettant d'obtenir les noms et prénoms des patients ayant strictement plus de 60 ans.

On sélectionne les colonnes `nom_patient` et `prenom` de la table `Patient`. La condition `age > 60` exclut les personnes âgées de 60 ans exactement.

```
SELECT nom_patient, prenom
FROM Patient
WHERE age > 60;
```

Dans l'extrait fourni, seul Woods, âgé de 65 ans, satisfait cette condition.

RÉPONSE À RETENIR

La requête renvoie :

nom_patient	prenom
Woods	Caroll

Question 2.

Sujet p. 9, 10

> Alice Heartman ne tousse plus. Écrire une requête SQL permettant de mettre à jour la base de données avec cette information.

Le symptôme est stocké dans la table `Symptome`. La ligne d'Alice Heartman est repérée par `nom_patient = 'Heartman'`. Il faut donc remplacer la valeur de `toux` par `'Non'`.

RÉPONSE À RETENIR

```
UPDATE Symptome
SET toux = 'Non'
WHERE nom_patient = 'Heartman';
```

Question 3.

Sujet p. 9, 10

> On souhaite effectuer des statistiques sur les symptômes des patients atteints de Covid-19. Écrire une requête SQL permettant de connaître le nombre de patients avec un diagnostic de Covid-19 qui toussent.

On joint `Diagnostic` et `Symptome` grâce à leur attribut commun `nom_patient`. On filtre ensuite sur le diagnostic Covid-19 et sur la valeur `'Oui'` du symptôme `toux`. Enfin, `COUNT(*)` compte les lignes correspondantes.

```
SELECT COUNT(*)
FROM Diagnostic
JOIN Symptome ON Diagnostic.nom_patient = Symptome.nom_patient
WHERE Diagnostic.nom_maladie = 'Covid-19'
  AND Symptome.toux = 'Oui';
```

La table `Diagnostic` est laissée vide dans l'extrait du sujet : on ne peut donc pas calculer le résultat numérique à partir des données fournies.

> **RÉPONSE À RETENIR**
>
> La requête renvoie le nombre de patients ayant un diagnostic Covid-19 et toussant. Le résultat chiffré n'est pas déterminable avec l'extrait fourni, car les diagnostics ne sont pas renseignés.

Question 4.

Sujet p. 9, 10

> Un employé de l'hôpital saisit la requête INSERT INTO Patients VALUES ('Douglas', 'Patrick', 168077230253829, 55). Expliquer pourquoi cette requête produit une erreur, en tenant compte du schéma relationnel fourni.

Le schéma fourni nomme la table `Patient`, alors que la requête utilise `Patients` : si aucune table `Patients` n'existe, le nom de table provoque une erreur. Il y a également un problème de clé : `nom_patient` est la clé primaire et la valeur `'Douglas'` est déjà présente dans la table `Patient`. Une fois le nom de table corrigé, cette insertion ne peut donc pas ajouter un second enregistrement portant cette même clé primaire.

> **RÉPONSE À RETENIR**
>
> La requête vise un nom de table différent de celui du schéma (`Patients` au lieu de `Patient`). Même en corrigeant ce nom, l'insertion serait refusée en raison du doublon de clé primaire `Douglas`.

Question 5.

Sujet p. 9, 10

> Proposer une modification du schéma relationnel qui permettrait de résoudre le problème mentionné à la question précédente.

Un nom de famille n'identifie pas nécessairement un patient de manière unique. On peut remplacer la clé primaire `nom_patient` par `numero_secu`, supposé unique, et faire référencer cette nouvelle clé par les autres tables qui identifient un patient. Le nom de famille devient alors un attribut ordinaire et peut être partagé par plusieurs patients.

```
Patient(
    numero_secu INT PRIMARY KEY,
    nom_patient TEXT,
    prenom TEXT,
    age INT
)

Symptome(
    numero_secu INT PRIMARY KEY REFERENCES Patient(numero_secu),
    toux TEXT,
    fievre TEXT,
    nausee TEXT,
    anosmie TEXT
)

Diagnostic(
    numero_secu INT PRIMARY KEY REFERENCES Patient(numero_secu),
    nom_maladie TEXT REFERENCES Maladie(nom_maladie)
)
```

Dans cette proposition, le nom de table reste `Patient`, au singulier, conformément au schéma. L'insertion doit aussi préciser les colonnes si l'ordre ou le nombre des attributs est susceptible de changer.

> **RÉPONSE À RETENIR**
>
> La clé primaire de `Patient` devient `numero_secu` ; les références au patient dans `Symptome` et `Diagnostic` utilisent cette même clé. Deux patients peuvent ainsi avoir le même nom de famille sans conflit de clé primaire.

Partie B

Question 6.

Sujet p. 11

> D'après l'arbre de la figure 1, donner le diagnostic pour un patient qui tousse et qui a de la fièvre, mais n'a ni nausée ni anosmie. Dans l'arbre, un symptôme présent conduit au sous-arbre droit et un symptôme absent au sous-arbre gauche.

Le patient tousse : depuis la racine `toux`, on suit donc la branche droite. Il a de la fièvre : au nœud `fievre` atteint, on suit encore la branche droite. Il n'a pas de nausée : au nœud `nausee`, on prend la branche gauche, qui mène à une feuille portant le signe `+`. L'anosmie n'intervient pas dans ce parcours.

> **RÉPONSE À RETENIR**
>
> Le diagnostic est positif pour la Covid-19 (+).

Question 7.

Sujet p. 12

> Préciser la signification de l'assertion de la méthode symptome de la classe Noeud.

L'instruction `assert not self.est_feuille()` vérifie que le nœud courant n'est pas une feuille avant que la méthode ne renvoie sa valeur comme symptôme. Si le nœud est une feuille, l'assertion échoue et provoque une exception `AssertionError` (lorsque les assertions sont activées).

RÉPONSE À RETENIR

La méthode `symptome` doit être appelée sur un nœud interne et non sur une feuille.

Question 8.

Sujet p. 11, 12

> Nommer un attribut et une méthode de la classe Noeud.

Parmi les attributs de la classe figurent `valeur`, `gauche` et `droit`. Parmi ses méthodes figurent `est_feuille`, `symptome` et `diagnostic`.

RÉPONSE À RETENIR

Par exemple : l'attribut `valeur` et la méthode `est_feuille`.

Question 9.

Sujet p. 11, 12

> Compléter la fonction applique(arbre, patient), définie récursivement, qui renvoie le diagnostic établi en utilisant un arbre de décision binaire implémenté à l'aide de la classe Noeud. Les symptômes du patient sont représentés par un dictionnaire dont les clés sont les symptômes possibles et dont les valeurs sont True si le patient présente le symptôme et False sinon. Compléter les cas correspondant à une feuille, à un symptôme présent et à un symptôme absent dans le code fourni.

Si le nœud courant est une feuille, on renvoie son diagnostic. Sinon, la valeur associée au symptôme dans le dictionnaire indique la branche à suivre : le sous-arbre droit si le symptôme est présent, le gauche s'il est absent.

RÉPONSE À RETENIR

```
def applique(arbre, patient):
    if arbre.est_feuille():
        return arbre.diagnostic()
    else:
        if patient[arbre.symptome()]:
            return applique(arbre.droit, patient)
        else:
            return applique(arbre.gauche, patient)
```

Avec le dictionnaire fourni, le parcours suit les branches droites pour `toux` et `fievre`, puis la branche gauche pour `nausee`. Il aboutit au diagnostic `+`.

Question 10.

Sujet p. 11, 12

> Donner la taille de l'arbre représenté en figure 1. On considère que la taille d'un arbre constitué d'une unique feuille est 1.

La figure est un arbre binaire complet de quatre niveaux de nœuds internes, suivis des feuilles. On compte les nœuds à chaque niveau : 1 à la racine, puis 2 nœuds `fievre`, 4 nœuds `nausee`, 8 nœuds `anosmie` et 16 feuilles.

RÉPONSE À RETENIR

La taille de l'arbre est donc `1 + 2 + 4 + 8 + 16 = 31` nœuds.

Question 11.

Sujet p. 11, 12, 13

> Appliquer à l'arbre de la figure 1 la règle de réduction illustrée en figure 2 : un nœud dont les deux sous-arbres sont des feuilles correspondant au même diagnostic peut être remplacé par une feuille correspondant à ce diagnostic. Dessiner le nouvel arbre.

On applique la règle aux nœuds `anosmie` dont les deux feuilles ont le même diagnostic. Ensuite, les nœuds `nausee` dont les deux sous-arbres sont devenus des feuilles de même valeur peuvent aussi être remplacés. On obtient l'arbre suivant ; chaque branche gauche correspond à un symptôme absent (`Non`) et chaque branche droite à un symptôme présent (`Oui`).

RÉPONSE À RETENIR

```
toux
├── Non → fievre
│   ├── Non → nausee
│   │   ├── Non → anosmie
│   │   │   ├── Non → −
│   │   │   └── Oui → +
│   │   └── Oui → −
│   └── Oui → nausee
│       ├── Non → +
│       └── Oui → −
└── Oui → fievre
    ├── Non → nausee
    │   ├── Non → +
    │   └── Oui → anosmie
    │       ├── Non → −
    │       └── Oui → +
    └── Oui → +
```

Question 12.

Sujet p. 11, 12, 13

> Compléter la méthode reduire, fournie dans le sujet, afin d'appliquer récursivement la règle de réduction et de réduire la taille d'un arbre de décision binaire sans changer les décisions prises.

La méthode réduit d'abord récursivement les deux sous-arbres. Si les deux enfants sont alors des feuilles portant la même valeur, le nœud courant devient une feuille de cette valeur : on remplace sa valeur par le diagnostic commun et on met ses deux références vers les enfants à `None`.

RÉPONSE À RETENIR

```
def reduire(self):
    """Fonction récursive qui réduit la taille d'un arbre de
    décision sans changer les décisions prises."""
    if self.est_feuille():
        return
    self.gauche.reduire()
    self.droit.reduire()
    if (self.gauche.est_feuille() and self.droit.est_feuille()
            and self.gauche.valeur == self.droit.valeur):
        self.valeur = self.gauche.valeur
        self.gauche = None
        self.droit = None
```

Cette méthode suppose que chaque nœud interne possède ses deux sous-arbres, comme dans l'arbre binaire présenté.

Partie C

Question 13.

Sujet p. 13, 14

Sur les 15 chiffres du numéro de sécurité sociale, les 13 premiers forment n et les 2 derniers forment la clé k, choisie pour que n + k soit un multiple de 97. Recopier et compléter la fonction verifie(num_secu), qui renvoie un booléen indiquant si un numéro de sécurité sociale représenté par un entier de type int est bien formé. Le code fourni calcule n = num_secu // 100 et k = num_secu % 100.

Les deux dernières opérations fournies extraient la clé `k` et le nombre `n` formé par les treize premiers chiffres. Le numéro est bien formé, selon la règle du sujet, lorsque `n + k` est divisible par 97, donc lorsque son reste modulo 97 est nul.

RÉPONSE À RETENIR

```
def verifie(num_secu):
    n = num_secu // 100
    k = num_secu % 100
    return (n + k) % 97 == 0
```

Pour l'exemple du sujet, `n = 2070535238001` et `k = 87` ; leur somme est divisible par 97, donc la fonction renvoie `True`.

Question 14.

Sujet p. 13, 14

Compléter la fonction cle, qui renvoie la clé k d'un numéro de sécurité sociale en prenant pour paramètre le nombre n formé des 13 premiers chiffres de ce numéro.

Il faut que `n + k` soit divisible par 97. Le reste de `-n` modulo 97 fournit une valeur de clé comprise entre 0 et 96 qui vérifie cette condition.

RÉPONSE À RETENIR

```
def cle(n):
    return (-n) % 97
```

Pour `n = 2070535238001`, la clé renvoyée est `87`, comme dans l'exemple du sujet.

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