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 • Polynésie • Jour 2

Code sujet **24-NSIJ2PO1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Structures de données FILE et PILE, graphes et algorithmes de parcours
11 réponses • sujet p. 2–16

Exercice 2 — Arbres binaires de recherche, POO et récursivité
12 réponses • sujet p. 6–9

Exercice 3 — Protocoles réseau, bases de données relationnelles et requêtes SQL, algorithmique et programmation en Python
12 réponses • sujet p. 10–15

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

Structures de données FILE et PILE, graphes et algorithmes de parcours

Partie A

Question 1.

Sujet p. 2

> À partir du graphe pondéré de la figure 1, déterminer le plus court chemin allant du sommet Mp au sommet Nc et préciser la longueur, en kilomètres, de ce chemin. Aucune justification n'est attendue.

Le chemin **Mp** → **Ar** → **Ax** → **Nc** a pour longueur `80 + 76 + 176 = 332` km. La comparaison des itinéraires du graphe montre qu'aucun autre chemin n'est plus court.

> **RÉPONSE À RETENIR**
>
> Le plus court chemin est **Mp** → **Ar** → **Ax** → **Nc**, de longueur **332 km**.

Question 2.

Sujet p. 2

> En partant du sommet Mp et en arrivant au sommet Nc, déterminer les deux chemins possibles qui permettent de visiter le minimum de villes.

Il faut minimiser le nombre d'arêtes. Il n'existe ni arête directe de Mp à Nc, ni sommet voisin à la fois de Mp et de Nc : un chemin minimal comporte donc trois arêtes et deux villes intermédiaires. Les deux possibilités passent par Ar, puis par Ax ou Mr.

> **RÉPONSE À RETENIR**
>
> Les deux chemins qui visitent le minimum de villes sont **Mp** → **Ar** → **Ax** → **Nc** et **Mp** → **Ar** → **Mr** → **Nc**.

Partie B

Question 3.

Sujet p. 3

> Donner l'implémentation, en langage Python, du graphe non pondéré de la figure 2, représenté par listes d'adjacence à l'aide d'un dictionnaire nommé G. Les sommets sont de type str. Écrire chaque couple clé/valeur sur une nouvelle ligne.

Le graphe est non orienté : si deux sommets sont reliés, chacun figure dans la liste de voisins de l'autre. Voici une implémentation possible, avec les voisins rangés par ordre alphabétique ; cet ordre est utilisé pour les résultats des parcours ci-dessous.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
G = {
    'Av': ['Ax', 'Mr', 'Ni'],
    'Ni': ['Ar', 'Av', 'Mp'],
    'Mp': ['Ar', 'Ni'],
    'Ar': ['Ax', 'Mp', 'Mr', 'Ni'],
    'Mr': ['Ar', 'Av', 'Ax', 'Nc', 'To'],
    'To': ['Ax', 'Mr', 'Nc'],
    'Nc': ['Ax', 'Di', 'Mr', 'To'],
    'Di': ['Ax', 'Nc'],
    'Ax': ['Ar', 'Av', 'Di', 'Mr', 'Nc', 'To']
}
```

Question 4.

Sujet p. 3

> Indiquer la signification des lettres dans les acronymes LIFO et FIFO.

Ces acronymes indiquent l'ordre dans lequel les éléments sortent de la structure.

RÉPONSE À RETENIR

- **LIFO** : « Last In, First Out », soit « dernier entré, premier sorti ».
- **FIFO** : « First In, First Out », soit « premier entré, premier sorti ».

Question 5.

Sujet p. 3

> Indiquer l'acronyme utilisé pour désigner la structure de file.

Dans une file, le premier élément ajouté est le premier retiré.

RÉPONSE À RETENIR

L'acronyme utilisé pour désigner la structure de file est **FIFO**.

Question 6.

Sujet p. 3, 4

> Donner le résultat renvoyé par l'appel parcours(G, 'Av'), en utilisant le graphe de la figure 2 et la fonction parcours fournie.

La fonction ajoute un sommet à `visite` dès qu'elle le découvre, puis l'enfile. Avec les listes d'adjacence de la question 3, Av découvre Ax, Mr et Ni. En traitant Ax, elle découvre ensuite Ar, Di, Nc et To ; en traitant Ni, elle découvre Mp.

Sommet traité	Nouveaux sommets ajoutés à visite
Av	Ax, Mr, Ni
Ax	Ar, Di, Nc, To
Mr	Aucun
Ni	Mp

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

`parcours(G, 'Av')` renvoie `['Av', 'Ax', 'Mr', 'Ni', 'Ar', 'Di', 'Nc', 'To', 'Mp']`.

Question 7.

Sujet p. 4

Recopier, parmi les deux propositions ci-dessous, celle qui correspond au type de parcours de graphe réalisé par la fonction parcours : proposition A, « parcours en largeur » ; proposition B, « parcours en profondeur ».

La fonction utilise une file : les sommets découverts sont traités dans l'ordre où ils ont été enfilés. Elle explore donc le graphe par niveaux de distance à partir du sommet de départ.

RÉPONSE À RETENIR

Proposition A : parcours en largeur.

Question 8.

Sujet p. 3, 4

En modifiant la fonction parcours, écrire une fonction distance ayant pour paramètres graphe, un dictionnaire représentant un graphe sous la forme de listes d'adjacence, et sommet, une chaîne de caractères représentant un sommet du graphe. La fonction renvoie un dictionnaire dont les clés sont les sommets du graphe et dont la valeur associée à chaque clé est la distance, en nombre d'arêtes, entre ce sommet et le sommet d'origine sommet.

On enregistre la distance dès qu'un sommet est découvert. Si le sommet traité est à la distance `d` de l'origine, tout voisin encore inconnu reçoit la distance `d + 1`. Comme la file traite les sommets par niveaux, cette première distance attribuée est la plus courte. Chaque sommet accessible est ajouté une seule fois à la file, ce qui assure la terminaison. Le graphe de la figure étant connexe, le dictionnaire contient ici tous ses sommets.

```
def distance(graphe, sommet):
    distances = {sommet: 0}
    f = creerFile()
    enfiler(f, sommet)

    while not estVide(f):
        s = defiler(f)
        for v in graphe[s]:
            if v not in distances:
                distances[v] = distances[s] + 1
                enfiler(f, v)

    return distances
```

Dans un graphe non connexe, cette version ne renvoie que les sommets accessibles depuis `sommet` ; la figure considérée ne présente pas ce cas.

RÉPONSE À RETENIR

La fonction renvoie un dictionnaire associant à chaque sommet accessible sa distance, en nombre d'arêtes, depuis le sommet d'origine.

FormaV - document d'entraînement non officiel

Question 9.

Sujet p. 3, 4

> Donner le résultat renvoyé par l'appel distance(G, 'Av').

Les voisins directs de Av, Ax, Mr et Ni, sont à distance 1. Ar, Di, Nc, To et Mp sont à distance 2 : chacun est voisin d'au moins un sommet situé à distance 1 de Av.

> **RÉPONSE À RETENIR**
>
> `distance(G, 'Av')` renvoie `{'Av': 0, 'Ax': 1, 'Mr': 1, 'Ni': 1, 'Ar': 2, 'Di': 2, 'Nc': 2, 'To': 2, 'Mp': 2}`.

Question 10.

Sujet p. 5, 16

> Traduire, dans le corps d'une fonction Python nommée parcours2, l'algorithme de parcours en pseudo-code donné. La fonction prend pour paramètres G, un dictionnaire représentant un graphe sous la forme de listes d'adjacence, et s, une chaîne de caractères représentant un sommet du graphe. Selon le corrigendum, remplacer la ligne 3 du pseudo-code par « créer une liste visite vide ».

Selon le corrigendum, la liste `visite` est créée vide. Un sommet n'est ajouté à cette liste et ses voisins ne sont empilés que lors de son premier dépilement. Les dépilements de sommets déjà visités ne provoquent donc pas de nouveau développement.

```
def parcours2(G, s):
    p = creerPile()
    empiler(p, s)
    visite = []

    while not estVide(p):
        x = depiler(p)
        if x not in visite:
            visite.append(x)
            for v in G[x]:
                empiler(p, v)

    return visite
```

Chaque sommet est développé au plus une fois. Comme le graphe est fini, un nombre fini d'éléments est empilé et l'algorithme termine.

> **RÉPONSE À RETENIR**
>
> Le code ci-dessus traduit le pseudo-code corrigé. Avec les listes d'adjacence de la question 3, les sommets ajoutés à `visite` sont, dans l'ordre, Av, Ni, Mp, Ar, Mr, To, Nc, Di et Ax.

Question 11.

Sujet p. 3, 5, 16

> Donner un résultat possible renvoyé par l'appel parcours2(G, 'Av').

La pile fonctionne selon le principe LIFO : les voisins empilés en dernier sont traités en premier. Avec l'ordre de voisins choisi à la question 3 et la liste `visite` initialement vide imposée par le corrigendum, le parcours ajoute successivement Av, Ni, Mp, Ar, Mr, To, Nc, Di et Ax. Des sommets déjà visités peuvent être dépilés, mais ne sont pas ajoutés une nouvelle fois.

RÉPONSE À RETENIR

Un résultat possible est `['Av', 'Ni', 'Mp', 'Ar', 'Mr', 'To', 'Nc', 'Di', 'Ax']`.

FormaV - document d'entraînement non officiel

Exercice 2

6 points

Arbres binaires de recherche, POO et récursivité

Partie A

Question 1.

Sujet p. 7

> Recopier et compléter le texte en choisissant parmi les expressions « au maximum », « au minimum », « exactement », « feuille », « racine », « sous-arbre gauche » et « sous-arbre droit » : le nœud initial est appelé … ; un nœud qui n'a pas de fils est appelé … ; un arbre binaire est un arbre dans lequel chaque nœud a … deux fils ; dans un arbre binaire de recherche, tout nœud est associé à une clé supérieure à chaque clé des nœuds de son … et inférieure à chaque clé des nœuds de son ….

La racine est le nœud initial ; une feuille n'a pas de fils. Dans un arbre binaire, chaque nœud a au maximum deux fils. Dans un arbre binaire de recherche, toutes les clés du sous-arbre gauche sont inférieures à celle du nœud, et toutes celles du sous-arbre droit lui sont supérieures.

RÉPONSE À RETENIR

Le nœud initial est appelé **racine**. Un nœud qui n'a pas de fils est appelé **feuille**. Chaque nœud a **au maximum** deux fils. La clé d'un nœud est supérieure à chaque clé de son **sous-arbre gauche** et inférieure à chaque clé de son **sous-arbre droit**.

Question 2.

Sujet p. 6, 7

> Donner, dans l'ordre, les clés obtenues lors du parcours préfixe de l'arbre no 1.

Un parcours préfixe visite la racine, puis le sous-arbre gauche, puis le sous-arbre droit. Dans l'arbre no 1, on visite donc d'abord 1, puis 0, puis les nœuds du sous-arbre droit dans l'ordre 2, 3, 4, 5, 6.

RÉPONSE À RETENIR

Parcours préfixe de l'arbre no 1 : 1, 0, 2, 3, 4, 5, 6.

Question 3.

Sujet p. 6, 7

> Donner, dans l'ordre, les clés obtenues lors du parcours suffixe, également appelé postfixe, de l'arbre no 2.

Un parcours postfixe visite d'abord le sous-arbre gauche, puis le sous-arbre droit, et enfin la racine. Dans l'arbre no 2, le parcours postfixe du sous-arbre gauche donne 0, 1, 2 ; celui du sous-arbre droit donne 6, 5, 4 ; on termine par la racine 3.

RÉPONSE À RETENIR

Parcours postfixe de l'arbre no 2 : 0, 1, 2, 6, 5, 4, 3.

FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 6, 7

> Donner, dans l'ordre, les clés obtenues lors du parcours infixe de l'arbre no 3.

Un parcours infixe visite le sous-arbre gauche, la racine, puis le sous-arbre droit. Pour un arbre binaire de recherche, cet ordre énumère les clés de la plus petite à la plus grande. L'arbre no 3 contient les clés de 0 à 6.

RÉPONSE À RETENIR

Parcours infixe de l'arbre no 3 : 0, 1, 2, 3, 4, 5, 6.

Question 5.

Sujet p. 6, 7

> Recopier et compléter les instructions afin de définir puis de construire, en y insérant les clés dans un ordre correct (il y a plusieurs possibilités, on en demande une), les trois instances de la classe ABR correspondant aux trois arbres représentés. Compléter les définitions de arbre_no1, arbre_no2 et arbre_no3, les listes de clés à insérer et les appels à la méthode d'insertion.

On crée trois arbres vides avec `ABR()`. Pour obtenir les formes représentées, on insère les clés dans un ordre qui place chaque clé dans le sous-arbre voulu. Les séquences suivantes construisent bien les trois arbres dessinés.

RÉPONSE À RETENIR

```
arbre_no1 = ABR()
arbre_no2 = ABR()
arbre_no3 = ABR()

for cle_a_inserer in [1, 0, 2, 3, 4, 5, 6]:
    arbre_no1.inserer(cle_a_inserer)

for cle_a_inserer in [3, 2, 4, 1, 5, 0, 6]:
    arbre_no2.inserer(cle_a_inserer)

for cle_a_inserer in [3, 1, 5, 0, 2, 4, 6]:
    arbre_no3.inserer(cle_a_inserer)
```

Question 6.

Sujet p. 6, 7, 8

> À l'aide de la méthode hauteur fournie, donner la hauteur des instances arbre_no1, arbre_no2 et arbre_no3 de la classe ABR correspondant aux trois arbres représentés.

La méthode attribue la hauteur −1 à un arbre vide et 0 à une feuille ; la hauteur d'un arbre non vide est donc le nombre d'arêtes du plus long chemin de sa racine à une feuille. Dans l'arbre no 1, le chemin 1–2–3–4–5–6 compte 5 arêtes. Dans l'arbre no 2, les plus longs chemins comptent 3 arêtes. Dans l'arbre no 3, ils comptent 2 arêtes.

RÉPONSE À RETENIR

Hauteurs : arbre_no1 : 5 ; arbre_no2 : 3 ; arbre_no3 : 2.

FormaV - document d'entraînement non officiel

Question 7.

Sujet p. 6, 8

> Recopier et compléter le code de la méthode est_presente(self, cle_a_rechercher), qui renvoie True si la clé cle_a_rechercher est présente dans l'instance d'ABR et False sinon.

Si l'arbre est vide, la clé est absente. Si la clé recherchée est égale à celle de la racine, elle est présente. Sinon, la propriété d'arbre binaire de recherche permet de poursuivre uniquement dans le sous-arbre gauche si la clé recherchée est plus petite, ou dans le sous-arbre droit si elle est plus grande. Le code fourni donne le nom de méthode `est_present` ; les appels récursifs utilisent donc ce nom.

RÉPONSE À RETENIR

```
def est_present(self, cle_a_rechercher):
    if self.est_vide():
        return False
    elif cle_a_rechercher == self.cle():
        return True
    elif cle_a_rechercher < self.cle():
        return self.sag().est_present(cle_a_rechercher)
    else:
        return self.sad().est_present(cle_a_rechercher)
```

Question 8.

Sujet p. 6, 8

> Expliquer laquelle des trois instructions suivantes nécessitera le moins d'appels récursifs avant de renvoyer son résultat : arbre_no1.est_presente(7), arbre_no2.est_presente(7) ou arbre_no3.est_presente(7).

La clé 7 n'apparaît dans aucun des arbres. La recherche suit donc le chemin vers les clés les plus grandes, puis s'arrête quand elle atteint un arbre vide. D'après les dessins, les chemins suivis sont 1, 2, 3, 4, 5, 6 dans l'arbre no 1 ; 3, 4, 5, 6 dans l'arbre no 2 ; et 3, 5, 6 dans l'arbre no 3. Il faut noter une différence de nom dans le sujet : la méthode est définie sous le nom `est_present` à la question 7, mais les appels de la question 8 écrivent `est_presente`. Pris littéralement, ces appels ne correspondent pas à la méthode définie ; on interprète la question comme visant cette méthode de recherche.

RÉPONSE À RETENIR

La recherche dans **l'arbre no 3** nécessite le moins d'appels récursifs : elle suit 3, puis 5, puis 6, avant d'atteindre le sous-arbre droit vide de 6. En comptant l'appel initial et l'appel sur l'arbre vide, il y a respectivement 7, 5 et 4 appels au total pour les arbres no 1, no 2 et no 3 ; le nombre d'appels récursifs après l'appel initial est donc 6, 4 et 3.

Partie B

Question 9.

Sujet p. 8

> À partir de la méthode est_partiellement_equilibre(self) fournie, expliquer ce qu'on appelle ici un arbre partiellement équilibré.

Pour un arbre non vide, la méthode compare les hauteurs de ses deux sous-arbres immédiats. Elle renvoie `True` lorsque la différence absolue de ces hauteurs ne dépasse pas 1. Elle ne vérifie pas si les sous-arbres sont eux-mêmes partiellement équilibrés.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Un arbre est ici **partiellement équilibré** si la différence entre les hauteurs de son sous-arbre gauche et de son sous-arbre droit est au plus 1, c'est-à-dire si |hauteur gauche − hauteur droite| ≤ 1. Ce critère est vérifié au niveau de la racine de l'instance considérée.

Question 10.

Sujet p. 6, 8, 9

> Justifier que, parmi les trois arbres définis plus haut, deux sont partiellement équilibrés.

On compare les hauteurs des deux sous-arbres de la racine. Dans l'arbre no 1, elles valent 0 à gauche et 4 à droite, donc leur différence vaut 4. Dans l'arbre no 2, elles valent 2 et 2 ; dans l'arbre no 3, elles valent 1 et 1. Le critère de la question 9 est donc vérifié pour les arbres no 2 et no 3, mais pas pour l'arbre no 1.

RÉPONSE À RETENIR

Les deux arbres partiellement équilibrés sont **l'arbre no 2 et l'arbre no 3**. L'arbre no 1 ne l'est pas, car |0 − 4| = 4, supérieur à 1. Pour l'arbre no 2, l'écart vaut |2 − 2| = 0 ; pour l'arbre no 3, il vaut |1 − 1| = 0.

Question 11.

Sujet p. 6, 8, 9

> Justifier que, parmi les trois arbres définis plus haut, un seul est équilibré. Un arbre binaire est équilibré s'il est partiellement équilibré et si ses deux sous-arbres, droit et gauche, sont eux-mêmes équilibrés ; un arbre vide est considéré comme équilibré.

Un arbre est équilibré si le critère d'équilibre partiel est respecté à chaque nœud, autrement dit si ses deux sous-arbres sont eux-mêmes équilibrés. L'arbre no 1 échoue au critère partiel dès sa racine. Dans l'arbre no 2, au nœud 2, le sous-arbre gauche a une hauteur 1 et le sous-arbre droit, vide, une hauteur −1 : l'écart vaut 2. Le même écart apparaît au nœud 4. Dans l'arbre no 3, les sous-arbres de chaque nœud ont des hauteurs qui diffèrent d'au plus 1.

RÉPONSE À RETENIR

Seul l'arbre no 3 est équilibré. L'arbre no 1 n'est pas partiellement équilibré. L'arbre no 2 est partiellement équilibré à sa racine, mais ne l'est pas à ses nœuds 2 et 4, où l'écart des hauteurs vaut 2. Dans l'arbre no 3, chaque nœud respecte le critère local et les arbres vides sont équilibrés ; l'arbre entier est donc équilibré.

Question 12.

Sujet p. 8, 9

> Définir et coder la méthode récursive est_equilibre de la classe ABR, qui renvoie True si l'instance de la classe ABR est l'implémentation d'un arbre équilibré et False sinon.

Un arbre vide est équilibré. Pour un arbre non vide, il faut vérifier le critère d'équilibre partiel au nœud courant, puis vérifier récursivement les deux sous-arbres. La récursion termine car chaque appel porte sur un sous-arbre plus petit, jusqu'à atteindre des arbres vides, qui renvoient `True`.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def est_equilibre(self):
    if self.est_vide():
        return True
    if not self.est_partiellement_equilibre():
        return False
    return self.sag().est_equilibre() and self.sad().est_equilibre()
```

Exercice 3

8 points

Protocoles réseau, bases de données relationnelles et requêtes SQL, algorithmique et programmation en Python

Partie A : Le réseau informatique d'un hôpital

Question 1.

Sujet p. 10

> En utilisant le protocole RIP, qui minimise le nombre de sauts entre routeurs, donner les chemins possibles d'un paquet de données partant du service de neurologie à destination du service d'imagerie.

Avec RIP, on minimise le nombre de sauts entre routeurs. Sur la figure, le service de neurologie est relié à R4, le service de radiologie-imagerie à R2, et R4 est directement relié à R2. Ce trajet comporte un seul saut entre routeurs, donc il est minimal.

RÉPONSE À RETENIR

Le chemin minimal est : service de neurologie → R4 → R2 → service de radiologie-imagerie. C'est le seul chemin comportant le minimum de sauts.

Question 2.

Sujet p. 10, 11

> Recopier et compléter la table des coûts du nœud R2 selon le protocole RIP. La table indique déjà un coût de 1 vers R1.

Pour RIP, le coût est le nombre de sauts entre routeurs. Depuis R2, les voisins directs R1, R4 et R8 ont un coût de 1. Les autres routeurs sont accessibles en deux sauts : par exemple, R5 par R1, et R3, R6, R7 et R9 par R8. Le coût de R2 vers lui-même est 0.

RÉPONSE À RETENIR

Destination	Coût RIP depuis R2
R1	1
R2	0
R3	2
R4	1
R5	2
R6	2
R7	2
R8	1
R9	2

FormaV - document d'entraînement non officiel

Question 3.

Sujet p. 11

> Calculer le coût d'une liaison de communication par la technologie FTTH, en utilisant la formule c = 10^8 / d, où d est la bande passante en bit/s. La bande passante des liaisons FTTH est de 10 Gbit/s.

On convertit le débit FTTH en bit/s : 10 Gbit/s = 10 × 10^9 bit/s = 10^{10} bit/s. Avec la formule donnée :

$$c = \frac{10^8}{10^{10}} = 10^{-2}.$$

RÉPONSE À RETENIR

Le coût d'une liaison FTTH est 0,01.

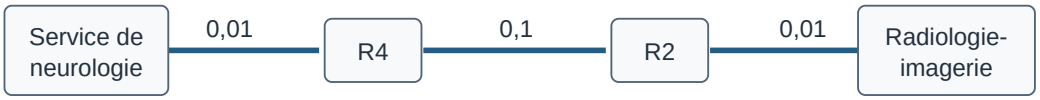
Question 4.

Sujet p. 10, 11

> Avec le protocole OSPF, qui cherche à minimiser la somme des coûts des liaisons, donner le chemin pris par un paquet partant du service de neurologie à destination du service d'imagerie. Utiliser la figure 1 et sa légende des types de liaison.

Le coût d'une liaison FTTH est 0,01. Sur la figure, le service de neurologie est relié à R4 par une liaison FTTH, le lien direct R4–R2 a un coût de 0,1, et R2 est relié au service de radiologie-imagerie par une liaison FTTH. R4 étant relié directement à R2, le trajet entre ces routeurs a pour coût 0,1. En comptant aussi les deux liaisons d'accès aux services, le coût total du trajet représenté est 0,01 + 0,1 + 0,01 = 0,12.

Chemin OSPF entre les deux services



Sous-graphe correspondant au chemin retenu ; les coûts des liaisons FTTH sont calculés à partir de la formule de l'énoncé.

RÉPONSE À RETENIR

Le paquet suit le chemin service de neurologie → R4 → R2 → service de radiologie-imagerie. Son coût est 0,1 entre routeurs, soit 0,12 en comptant aussi les deux liaisons FTTH d'accès aux services.

FormaV - document d'entraînement non officiel

Partie B : le dossier médical d'un patient

Question 5.

Sujet p. 12

> Décrire simplement le résultat obtenu avec la requête SQL : SELECT nom, prenom FROM patient WHERE num_SS LIKE '1%';

`LIKE '1%'` sélectionne les numéros de Sécurité Sociale qui commencent par le caractère « 1 ». La requête n'affiche ensuite que le nom et le prénom de ces patients. Dans l'extrait fourni, les numéros de Kylian Tardus et de Vincent Montpart commencent par 1. Sans `ORDER BY`, l'ordre d'affichage n'est pas garanti.

RÉPONSE À RETENIR

nom	prenom
Tardus	Kylian
Montpart	Vincent

Question 6.

Sujet p. 12

> Écrire une requête SQL permettant d'afficher le numéro de Sécurité Sociale des patients hospitalisés dans le service intitulé orthopédique durant l'année 2023.

La table `hospitalisation` contient les attributs `num_SS`, `date` et `service`. On la joint à `patient` par le numéro de Sécurité Sociale, puis on filtre sur le service et sur les dates de l'année 2023. `DISTINCT` évite d'afficher plusieurs fois le même numéro si plusieurs hospitalisations correspondent aux critères. La requête ci-dessous suppose que `date` est une valeur SQL de type DATE, comparée à des dates écrites au format SQL AAAA-MM-JJ.

RÉPONSE À RETENIR

```
SELECT DISTINCT p.num_SS
FROM patient AS p
JOIN hospitalisation AS h ON p.num_SS = h.num_SS
WHERE h.service = 'orthopédique'
  AND h.date >= '2023-01-01'
  AND h.date < '2024-01-01';
```

Les extraits fournis ne contiennent pas de lignes de la table `hospitalisation` : les numéros obtenus ne peuvent donc pas être déterminés à partir des seules données affichées.

Question 7.

Sujet p. 12

> Écrire une requête SQL permettant d'afficher le type et la date de chaque examen de la patiente Mme Baujean Emma.

La table `examen` contient le type, la date et le numéro de Sécurité Sociale du patient. On la joint à `patient` par `num_SS`, puis on identifie Mme Baujean Emma par son nom et son prénom.

RÉPONSE À RETENIR

```
SELECT e.type, e.date
FROM examen AS e
JOIN patient AS p ON e.num_SS = p.num_SS
WHERE p.nom = 'Baujean'
  AND p.prenom = 'Emma';
```

Aucune ligne de la table `examen` n'est fournie : les types et dates des examens ne peuvent pas être calculés à partir des extraits.

Question 8.

Sujet p. 12

> Écrire une requête SQL permettant d'afficher le nom et le prénom de tous les patients du médecin M. ARNOS Pierre.

La table `consultation` relie les patients aux médecins par les clés étrangères `num_SS` et `id_medecin`. On la joint donc aux tables `patient` et `medecin`, puis on sélectionne le médecin ARNOS Pierre. `DISTINCT` évite de répéter un patient ayant eu plusieurs consultations avec ce médecin.

RÉPONSE À RETENIR

```
SELECT DISTINCT p.nom, p.prenom
FROM patient AS p
JOIN consultation AS c ON p.num_SS = c.num_SS
JOIN medecin AS m ON c.id_medecin = m.id_medecin
WHERE m.nom = 'ARNOS'
  AND m.prenom = 'Pierre';
```

Les extraits ne fournissent pas les lignes des tables `consultation` et `medecin` nécessaires pour déterminer le résultat.

Partie C : la sécurité des mots de passe d'un médecin

Question 9.

Sujet p. 13, 14

> Compléter le script de la fonction mdp_fort(mdp), qui renvoie True si mdp est un mot de passe fort et False sinon, en recopiant les lignes 2, 10, 12 et 14. Un mot de passe fort comporte au minimum 12 caractères, au moins 2 majuscules, au moins 2 chiffres et au moins 2 symboles parmi #@!?%<>=€$+-*/&.

On vérifie d'abord la longueur minimale. On parcourt ensuite le mot de passe en comptant séparément les majuscules, les chiffres et les caractères appartenant à `liste_symboles`. Il faut enfin rejeter le mot de passe si l'un de ces trois comptes est inférieur à 2.

RÉPONSE À RETENIR

```
def mdp_fort(mdp):
    if len(mdp) < 12:
        return False
    majuscules = 0
    chiffres = 0
    symboles = 0
    for caractere in mdp:
        if caractere.isupper():
            majuscules += 1
        if caractere.isdigit():
            chiffres += 1
        if caractere in liste_symboles:
            symboles += 1
    if majuscules < 2 or chiffres < 2 or symboles < 2:
        return False
    return True
```

Les trois compteurs et la longueur doivent respecter simultanément leurs seuils minimaux.

Question 10.

Sujet p. 14

> Compléter le script de la fonction creation_mdp(n, nbr_m, nbr_c, nbr_s), qui génère aléatoirement un mot de passe respectant les minima indiqués pour sa longueur, ses majuscules, ses chiffres et ses symboles, en recopiant la ligne 9 et les lignes 13 à 19.

Une complétion directe du gabarit avec une boucle qui continue tant que la longueur est inférieure à `n` ou qu'un minimum de caractères n'est pas atteint peut produire un mot de passe plus long que `n`. Pour respecter littéralement une longueur exactement égale à `n` tout en garantissant les minima, on peut générer des chaînes de longueur `n` jusqu'à en obtenir une qui satisfait les trois conditions. Cela suppose des paramètres réalisables, notamment `n >= nbr_m + nbr_c + nbr_s` et des minima non négatifs.

RÉPONSE À RETENIR

```
from random import choice

def creation_mdp(n, nbr_m, nbr_c, nbr_s):
    caracteres = 'abcdefghijklmnopqrstuvwxyz' + \
                 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' + \
                 liste_symboles
    while True:
        mdp = ''
        majuscules = 0
        chiffres = 0
        symboles = 0
        for _ in range(n):
            c = choice(caracteres)
            mdp = mdp + c
            if c.isupper():
                majuscules += 1
            if c.isdigit():
                chiffres += 1
            if c in liste_symboles:
                symboles += 1
        if majuscules >= nbr_m and chiffres >= nbr_c and symboles >= nbr_s:
            return mdp
```

Chaque essai a exactement `n` caractères. Si les minima sont réalisables, un essai valide a une probabilité non nulle d'être obtenu ; la répétition finit donc par en produire un avec probabilité 1.

Question 11.

Sujet p. 13, 15

> Compléter le script de la fonction recherche_mot(mdp), qui renvoie les mots présents dans mdp, en recopiant les lignes 5, 6, 15, 16 et 18. Les mots entiers sont séparés par des chiffres ou des symboles ; la fonction transforme les majuscules en minuscules et utilise liste_symboles.

Après transformation, on avance l'indice sur les chiffres et les symboles, qui sont les séparateurs définis par l'énoncé. Sinon, on accumule les lettres jusqu'au prochain séparateur et on ajoute la suite obtenue à la liste. Comme la fonction `transforme` met les lettres en minuscules, le résultat de ce code est en minuscules. Les exemples imprimés dans le sujet ne sont pas cohérents avec ces règles : par exemple, avec les séparateurs annoncés, `NeuroA` forme une seule suite de lettres.

RÉPONSE À RETENIR

```
def recherche_mot(mdp):
    mot = transforme(mdp)
    trouve = []
    i = 0
    while i < len(mot):
        if mot[i].isdigit():
            i = i + 1
        elif mot[i] in liste_symboles:
            i = i + 1
        else:
            chaine = ''
            while i < len(mot) and mot[i].isalpha():
                chaine = chaine + mot[i]
                i = i + 1
            trouve.append(chaine)
    return trouve
```

Selon le code et la règle des séparateurs, `recherche_mot('@20!NeUrO&24#')` renvoie `['neuro']`. De même, `recherche_mot('Chef!14NeuroA@85!')` renvoie `['chef', 'neuroa']`, et non l'exemple affiché dans le sujet.

Question 12.

Sujet p. 14, 15

> Écrire une fonction mdp_extra_fort qui prend en paramètre une chaîne de caractères mdp et qui renvoie True si mdp est un mot de passe extra fort et False sinon. Un mot de passe extra fort ne contient pas de mot de 4 lettres ou plus de la langue française ou étrangère ; on dispose de la liste dicoFR et de la fonction recherche_mot.

On parcourt les suites de lettres renvoyées par `recherche_mot`. Si l'une d'elles a au moins quatre lettres et appartient au dictionnaire français, le mot de passe n'est pas extra fort. On normalise aussi les entrées de `dicoFR` en minuscules pour les comparer aux résultats de `recherche_mot`. L'énoncé ne fournit toutefois qu'un dictionnaire français : la fonction ci-dessous ne peut donc pas détecter les mots étrangers et ne certifie que l'absence de mots français d'au moins quatre lettres.

RÉPONSE À RETENIR

```
def mdp_extra_fort(mdp):
    mots_francais = {transforme(mot) for mot in dicoFR}
    for mot in recherche_mot(mdp):
        if len(mot) >= 4 and mot in mots_francais:
            return False
    return True
```

La fonction renvoie `False` dès qu'elle trouve un mot français d'au moins quatre lettres ; sinon elle renvoie `True` au regard du dictionnaire fourni.

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