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 2026 • Polynésie • Jour 1

Code sujet **26-NSIJ1PO1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Complexité et calcul des termes d’une suite récurrente
7 réponses • sujet p. 2–3

Exercice 2 — Annuaire téléphonique : programmation Python, programmation orientée objet, dictionnaires et arbres binaires de recherche
16 réponses • sujet p. 4–8

Exercice 3 — Chatbot : programmation orientée objet, bases de données et arbres binaires de recherche
14 réponses • sujet p. 9–11

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

Complexité et calcul des termes d'une suite récurrente

Question 1.

Sujet p. 2

> Compléter la fonction récursive suite, qui prend en paramètre un entier n et renvoie la valeur de U_n, pour la suite définie par $U_0 = 0$, $U_1 = 1$ et $U_n = U_{n-1} + U_{n-2}$ pour tout $n \geq 2$.

Pour les indices 0 et 1, la valeur de U_n est directement `n`. Pour `n ≥ 2`, on applique la relation de récurrence en additionnant les deux termes précédents. On considère ici, comme dans la définition de la suite, que `n` est un entier naturel.

```
def suite(n):
    if n < 2:
        valeur = n
    else:
        valeur = suite(n - 1) + suite(n - 2)
    return valeur
```

Par exemple, pour `n = 5`, la fonction calcule `suite(4) + suite(3) = 3 + 2 = 5`.

> **RÉPONSE À RETENIR**
>
> Les compléments sont : ligne 3, `n` ; ligne 5, `n - 2` ; ligne 6, `valeur`.

Question 2.

Sujet p. 2

> Écrire une fonction nb_additions_suite prenant en paramètre un entier n et renvoyant la valeur de p(n), où p(0) = 0, p(1) = 0 et p(n) = 1 + p(n − 1) + p(n − 2) pour tout n ≥ 2.

On traduit la définition de `p` : les valeurs de base sont nulles pour `n = 0` et `n = 1`, puis, pour `n ≥ 2`, on compte l'addition effectuée en plus des additions des deux appels récursifs.

```
def nb_additions_suite(n):
    if n < 2:
        return 0
    return 1 + nb_additions_suite(n - 1) + nb_additions_suite(n - 2)
```

Par exemple, `p(4) = 1 + p(3) + p(2)`. Or `p(3) = 2` et `p(2) = 1`, donc `p(4) = 4`.

> **RÉPONSE À RETENIR**
>
> Cette fonction renvoie `p(n)` conformément à la récurrence donnée, pour un entier naturel `n`.

Question 3.

Sujet p. 2

> Calculer les valeurs p(2), p(4) et p(7).

On applique la récurrence en partant de `p(0) = p(1) = 0` :

- `p(2) = 1 + p(1) + p(0) = 1`.
- `p(3) = 1 + p(2) + p(1) = 2`, puis `p(4) = 1 + p(3) + p(2) = 1 + 2 + 1 = 4`.

FormaV - document d'entraînement non officiel

- `p(5) = 1 + p(4) + p(3) = 7`, `p(6) = 1 + p(5) + p(4) = 12`, puis `p(7) = 1 + p(6) + p(5) = 20`.

> **RÉPONSE À RETENIR**
>
> **`p(2) = 1`, `p(4) = 4` et `p(7) = 20`.**

Question 4.

Sujet p. 2, 3

> Recopier et compléter le tableau avec le nombre d'additions effectuées par la fonction suite2 pour les calculs des premiers termes de la suite U_n. Le tableau donne déjà 1 addition pour n = 2, 2 additions pour n = 3 et 3 additions pour n = 4 ; compléter les lignes pour n = 5, 6 et 7.

Pour compter les additions nécessaires au calcul d'un terme avec `suite2`, on part de la liste initiale `[0, 1]` et on considère un calcul isolé de chaque terme. Lorsqu'un nouvel indice est atteint, l'addition qui forme ce terme est effectuée une fois ; les termes déjà présents dans la liste sont réutilisés. Ainsi, après les valeurs fournies pour `n = 2`, `3` et `4`, chaque indice supplémentaire demande une addition de plus.

Attention : en Python, la liste par défaut de la fonction est partagée entre les appels. Les nombres du tableau correspondent donc au calcul de chaque terme à partir de la liste initiale, comme le suppose la question ; dans une même exécution où la liste aurait déjà été remplie par un appel antérieur, un nouvel appel pourrait réutiliser ces valeurs.

> **RÉPONSE À RETENIR**

n	Nombre d'additions
2	1
3	2
4	3
5	4
6	5
7	6

Question 5.

Sujet p. 3

> Expliquer pourquoi suite2 est plus efficace que suite et donner le nom du procédé algorithmique utilisé pour obtenir cette efficacité.

La fonction `suite` recalcule plusieurs fois certains termes : les deux appels récursifs peuvent demander à nouveau des valeurs déjà calculées dans une autre branche. À l'inverse, `suite2` conserve les valeurs dans `valeurs_calculees` et les réutilise. Pour un calcul à partir de la liste initiale, chaque terme jusqu'à l'indice demandé est donc calculé une seule fois. Par exemple, pour `n = 7`, `suite` effectue `p(7) = 20` additions, contre `6` additions pour `suite2`.

Le procédé utilisé est la **mémoïsation** : on mémorise les résultats déjà obtenus pour éviter de les recalculer. `suite2` reste cependant récursive, donc sa profondeur d'appels peut augmenter avec `n`.

RÉPONSE À RETENIR

`suite2` est plus efficace parce qu'elle mémorise les termes déjà calculés et évite les recalculs. Ce procédé s'appelle la **mémoïsation**.

Question 6.

Sujet p. 3

> Proposer une version itérative, non récursive, suite3 du calcul des termes de la suite U_n.

On calcule les termes dans l'ordre, de l'indice 2 jusqu'à `n`, en conservant les deux termes consécutifs nécessaires à la récurrence. Pour `n = 0` ou `n = 1`, le terme vaut directement `n`.

```
def suite3(n):
    if n < 2:
        return n
    precedent = 0
    courant = 1
    for i in range(2, n + 1):
        precedent, courant = courant, precedent + courant
    return courant
```

Trace pour `n = 5` : le couple initial `(precedent, courant)` vaut `(0, 1)`. Après les itérations d'indices 2, 3, 4 et 5, il vaut respectivement `(1, 1)`, `(1, 2)`, `(2, 3)` et `(3, 5)`. La fonction renvoie donc U_5 `= 5`. Pour `n` ≥ `2`, la boucle effectue `n - 1` additions.

RÉPONSE À RETENIR

La fonction itérative `suite3` ci-dessus renvoie U_n sans appel récursif.

Question 7.

Sujet p. 3

> Expliquer pourquoi la version itérative est préférable à la version récursive pour obtenir un terme quelconque de la suite U_n.

La version itérative calcule chaque terme à partir des deux précédents, une seule fois, et ne conserve que deux valeurs auxiliaires. Elle évite les nombreux recalculs de la fonction récursive `suite` et ne crée pas de chaîne d'appels récursifs susceptible de dépasser la profondeur de pile disponible. Pour `n` ≥ `2`, elle effectue `n - 1` additions, soit un nombre d'opérations proportionnel à `n`, et utilise un espace auxiliaire constant.

La version mémoïsée `suite2` limite elle aussi les recalculs, mais elle demeure récursive : la mémorisation ne supprime pas le risque lié à une profondeur d'appels trop grande.

RÉPONSE À RETENIR

La version itérative est préférable : elle effectue un nombre linéaire d'additions sans appels récursifs et n'utilise qu'un espace auxiliaire constant. Elle évite ainsi les recalculs de la récursion simple et le risque de dépassement de profondeur de pile des versions récursives.

Exercice 2

6 points

Annuaire téléphonique : programmation Python, programmation orientée objet, dictionnaires et arbres binaires de recherche

Partie A

Question 1.

Sujet p. 4

> Donner la structure de données de la variable contacts_fictifs.

La variable est délimitée par des crochets et ses éléments sont des dictionnaires : c'est donc une liste de dictionnaires. Chaque dictionnaire associe un prénom, clé de type chaîne de caractères, à un numéro de téléphone, valeur également de type chaîne de caractères.

> **RÉPONSE À RETENIR**
>
> `contacts_fictifs` est une liste contenant dix dictionnaires, chacun associant un prénom à un numéro de téléphone.

Question 2.

Sujet p. 4, 5

> Justifier que l'arbre binaire ci-dessus est bien un arbre binaire de recherche pour l'ordre alphabétique.

Dans un arbre binaire de recherche, toutes les clés du sous-arbre gauche d'un nœud sont inférieures à sa clé, et toutes celles du sous-arbre droit lui sont supérieures, selon l'ordre alphabétique.

À la racine, Olivier, les prénoms du sous-arbre gauche (Benjamin, Antoine, Eric et Emilie) sont tous avant Olivier ; ceux du sous-arbre droit (Sophie, Roxanne, Victor, Yann et Zoé) sont tous après lui. Dans le sous-arbre de Benjamin, Antoine est avant Benjamin et Eric est après Benjamin ; Emilie, fils gauche d'Eric, est bien après Benjamin et avant Eric. Dans le sous-arbre de Sophie, Roxanne est avant Sophie et Victor est après Sophie ; Yann est après Victor et Zoé après Yann.

> **RÉPONSE À RETENIR**
>
> L'arbre représenté respecte donc la règle d'ordre d'un arbre binaire de recherche pour les prénoms.

Question 3.

Sujet p. 4, 5

> Indiquer la racine de cet arbre, puis donner l'ensemble des feuilles de cet arbre.

La racine est le nœud situé tout en haut de l'arbre. Une feuille est un nœud qui n'a ni fils gauche ni fils droit. En examinant l'arbre, les nœuds sans enfant sont Antoine, Emilie, Roxanne et Zoé.

> **RÉPONSE À RETENIR**
>
> La racine est **Olivier**. L'ensemble des feuilles est **{Antoine, Emilie, Roxanne, Zoé}**.

FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 4, 5

> Donner la hauteur et la taille de l'arbre.

La taille est le nombre total de nœuds : l'arbre contient les dix contacts. La hauteur est le nombre de nœuds du plus long chemin de la racine à une feuille, puisque le sujet définit la hauteur d'un arbre réduit à un nœud comme étant 1. Le plus long chemin est Olivier → Sophie → Victor → Yann → Zoé ; il compte cinq nœuds.

> **RÉPONSE À RETENIR**
>
> La hauteur de l'arbre est **5** et sa taille est **10**.

Question 5.

Sujet p. 4, 5

> Citer l'avantage d'utiliser un arbre de recherche par rapport au type de données de la variable contacts_fictifs.

Dans la liste de dictionnaires, rechercher un prénom impose en général d'examiner les éléments l'un après l'autre. Dans un arbre binaire de recherche, chaque comparaison permet de continuer dans le seul sous-arbre pertinent. Lorsque l'arbre est équilibré, la recherche et l'insertion peuvent donc être plus rapides ; dans un arbre très déséquilibré, cet avantage peut disparaître.

> **RÉPONSE À RETENIR**
>
> L'arbre binaire de recherche permet notamment de rechercher un contact plus efficacement qu'une recherche séquentielle dans la liste, si l'arbre est suffisamment équilibré.

Question 6.

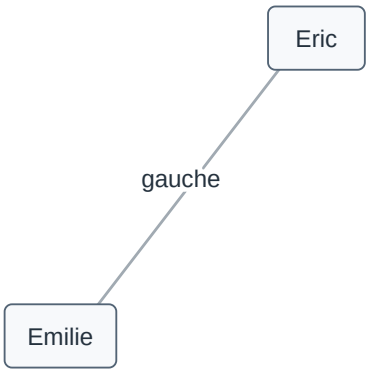
Sujet p. 4, 5

> Donner le sous-arbre droit du nœud Benjamin.

Le sous-arbre droit d'un nœud est l'arbre dont la racine est son fils droit. Sur l'arbre fourni, le fils droit de Benjamin est Eric, qui a Emilie comme fils gauche.

FormaV - document d'entraînement non officiel

Sous-arbre droit de Benjamin



Eric est la racine du sous-arbre ; Emilie est son fils gauche.

> **RÉPONSE À RETENIR**
>
> Le sous-arbre droit de Benjamin a pour racine **Eric** ; Emilie est son fils gauche et il n'a pas de fils droit.

Question 7. Instruction A

Sujet p. 5

> Parmi les instructions (A), (B) et (C), écrire la lettre correspondant à celle qui construit et stocke dans la variable abr le sous-arbre ayant Benjamin pour racine. Instruction A : abr = Noeud(Contact('Benjamin', '06 12 34 56 78'), Noeud(Contact('Antoine', '09 87 65 43 21')), Noeud(Contact('Emilie', '04 23 45 67 89'), Noeud(Contact('Eric', '03 45 67 89 01')))).

Dans l'instruction A, le fils droit de Benjamin est Emilie, avec Eric comme fils gauche. Dans l'arbre demandé, le fils droit de Benjamin est Eric, et Emilie est le fils gauche d'Eric. La structure ne correspond donc pas.

> **RÉPONSE À RETENIR**
>
> L'instruction A ne convient pas.

Question 7. Instruction B

Sujet p. 5

> Parmi les instructions (A), (B) et (C), écrire la lettre correspondant à celle qui construit et stocke dans la variable abr le sous-arbre ayant Benjamin pour racine. Instruction B : abr = Noeud(Contact('Benjamin', '06 12 34 56 78'), Noeud(Contact('Antoine', '09 87 65 43 21')), Noeud(Contact('Eric', '03 45 67 89 01'), Noeud(Contact('Emilie', '04 23 45 67 89')))).

Dans l'instruction B, Benjamin est la racine, Antoine son fils gauche et Eric son fils droit. Emilie est construite comme fils gauche d'Eric, ce qui correspond à l'arbre fourni.

> **RÉPONSE À RETENIR**
>
> L'instruction correcte est **B**.

Question 7. Instruction C

Sujet p. 5

> Parmi les instructions (A), (B) et (C), écrire la lettre correspondant à celle qui construit et stocke dans la variable abr le sous-arbre ayant Benjamin pour racine. Instruction C : abr = Noeud(Noeud(Contact('Benjamin', '06 12 34 56 78'), ('Antoine', '09 87 65 43 21')), Noeud(Contact('Eric', '03 45 67 89 01'), ('Emilie', '04 23 45 67 89'))).

Dans l'instruction C, le premier argument de `Noeud` est lui-même un objet `Noeud`, alors que le constructeur attend un objet `Contact` pour le contact du nœud. De plus, les prénoms et numéros fournis sous forme de tuples ne sont pas construits en objets `Contact`.

RÉPONSE À RETENIR

L'instruction C ne convient pas.

Question 8.

Sujet p. 6

> Donner les instructions à placer aux lignes 14, 23 et 26 pour compléter les méthodes inserer_recursive et trouver_recursive de la classe ContactManager.

Pour l'insertion, si le prénom est inférieur à celui du nœud courant, le code descend à gauche ; sinon, il descend à droite. La ligne 14 complète donc la branche `else`. Pour la recherche, si le prénom recherché est égal à celui du nœud courant, on renvoie son numéro ; s'il est inférieur, on poursuit à gauche. On suppose que les prénoms insérés sont distincts, comme l'indique le sujet pour les ABR.

RÉPONSE À RETENIR

Les instructions à placer sont :

```
Ligne 14 : noeud.droite = self.inserer_recursive(noeud.droite, contact)
Ligne 23 : if prenom == noeud.contact.prenom:
Ligne 26 : return self.trouver_recursive(noeud.gauche, prenom)
```

La recherche renvoie `None` si le nœud est absent, le numéro si le prénom est trouvé, et poursuit à gauche ou à droite selon l'ordre alphabétique.

Question 9.

Sujet p. 6, 7

> En utilisant la variable contact_manager définie à la ligne 2 et en particulier sa méthode trouver_contact, écrire en langage Python une fonction recherche_contact qui prend en paramètre prenom_recherche, une chaîne de caractères à rechercher, et qui affiche dans la console le prénom recherché puis le numéro du contact s'il existe (exemple avec « Roxanne »), ou le message « Contact 'Adrien' introuvable » si le contact est introuvable.

La méthode `trouver_contact` renvoie le numéro si le prénom est présent et `None` sinon. On affiche d'abord le prénom recherché, puis on teste cette valeur de retour pour choisir le message.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def recherche_contact(prenom_recherche):
    print('Prénom recherché :', prenom_recherche)
    numero = contact_manager.trouver_contact(prenom_recherche)
    if numero is None:
        print("Contact '" + prenom_recherche + "' introuvable")
    else:
        print(prenom_recherche + ' : ' + numero)
```

Pour `Roxanne`, la fonction affiche :

```
Prénom recherché : Roxanne
Roxanne : 07 45 67 89 01
```

Pour `Adrien`, elle affiche :

```
Prénom recherché : Adrien
Contact 'Adrien' introuvable
```

Question 10. Parcours en largeur

Sujet p. 4, 7

Indiquer, en le justifiant, si le parcours en largeur est le type de parcours à utiliser pour afficher la liste des prénoms dans l'ordre alphabétique.

Un parcours en largeur visite les nœuds niveau par niveau. Cet ordre dépend de la forme de l'arbre et ne garantit pas un ordre alphabétique.

RÉPONSE À RETENIR

Non : le parcours en largeur ne convient pas pour afficher les prénoms dans l'ordre alphabétique.

Question 10. Parcours en profondeur dans l'ordre préfixe

Sujet p. 4, 7

Indiquer, en le justifiant, si le parcours en profondeur dans l'ordre préfixe est le type de parcours à utiliser pour afficher la liste des prénoms dans l'ordre alphabétique.

Un parcours préfixe visite le nœud courant avant ses sous-arbres. Il affiche donc la racine avant les prénoms de son sous-arbre gauche, qui sont pourtant alphabétiquement antérieurs à elle.

RÉPONSE À RETENIR

Non : le parcours en profondeur préfixe ne convient pas.

Question 10. Parcours en profondeur dans l'ordre infixe

Sujet p. 4, 7

Indiquer, en le justifiant, si le parcours en profondeur dans l'ordre infixe est le type de parcours à utiliser pour afficher la liste des prénoms dans l'ordre alphabétique.

Un parcours infixe visite d'abord le sous-arbre gauche, puis le nœud courant, puis le sous-arbre droit. Dans un arbre binaire de recherche, les clés du sous-arbre gauche sont inférieures à la clé courante et celles du sous-arbre droit lui sont supérieures ; ce parcours énumère donc les prénoms dans l'ordre alphabétique.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Oui : il faut utiliser le parcours en profondeur dans l'ordre infixe.

Question 10. Parcours en profondeur dans l'ordre postfixe

Sujet p. 4, 7

> Indiquer, en le justifiant, si le parcours en profondeur dans l'ordre postfixe est le type de parcours à utiliser pour afficher la liste des prénoms dans l'ordre alphabétique.

Un parcours postfixe visite les sous-arbres avant le nœud courant. Il ne visite donc pas les clés dans l'ordre croissant d'un arbre binaire de recherche.

RÉPONSE À RETENIR

Non : le parcours en profondeur postfixe ne convient pas.

Question 11.

Sujet p. 7, 8

> Écrire les instructions des lignes 4 et 5 de la fonction récursive affichage_contacts_trie afin d'obtenir la liste des contacts dans l'ordre alphabétique et sous la forme donnée dans le sujet.

Pour effectuer un parcours infixe, la fonction appelle d'abord récursivement l'affichage sur le fils gauche, affiche ensuite le contact courant, puis l'appel déjà fourni sur le fils droit termine le parcours. L'affichage utilise le prénom et le numéro de téléphone du contact.

RÉPONSE À RETENIR

Les instructions à insérer aux lignes 4 et 5 sont :

```
affichage_contacts_trie(noeud.gauche)
print(noeud.contact.prenom, ':', noeud.contact.num_tel)
```

La fonction complète est :

```
def affichage_contacts_trie(noeud):
    """Afficher les contacts triés par nom"""
    if noeud is not None:
        affichage_contacts_trie(noeud.gauche)
        print(noeud.contact.prenom, ':', noeud.contact.num_tel)
        affichage_contacts_trie(noeud.droite)
```

Pour l'arbre fourni, le parcours affiche successivement Antoine, Benjamin, Emilie, Eric, Olivier, Roxanne, Sophie, Victor, Yann et Zoé.

Exercice 3

8 points

Chatbot : programmation orientée objet, bases de données et arbres binaires de recherche

Partie 1 : Structures de données

Question 1.

Sujet p. 9

> Recopier et compléter les lignes 3 et 6 dans le constructeur de la classe Chatbot et dans la méthode ajouter_question_reponse.

La base de données doit être un dictionnaire vide à la création de chaque objet. La méthode d'ajout associe ensuite la réponse à la question utilisée comme clé, conformément à la structure demandée.

RÉPONSE À RETENIR

```
self.base_donnees = {}
self.base_donnees[question] = reponse
```

Question 2.

Sujet p. 9

> Écrire les lignes de code permettant d'instancier un objet Bot1 de la classe Chatbot et d'ajouter à son dictionnaire l'interaction suivante : question « Bonjour, comment vas-tu ? » ; réponse « Bonjour, je vais bien, merci. »

On instancie la classe `Chatbot`, puis on appelle sa méthode d'ajout avec la question et sa réponse.

RÉPONSE À RETENIR

```
Bot1 = Chatbot()
Bot1.ajouter_question_reponse(
    "Bonjour, comment vas-tu ?",
    "Bonjour, je vais bien, merci."
)
```

Question 3.

Sujet p. 9

> Écrire les lignes de code permettant d'obtenir la réponse à la question « Bonjour, comment vas-tu ? ».

La méthode `repondre` renvoie la valeur du dictionnaire associée à la question fournie. On peut afficher cette valeur avec `print`.

RÉPONSE À RETENIR

```
print(Bot1.repondre("Bonjour, comment vas-tu ?"))
```

Le texte affiché est : `Bonjour, je vais bien, merci.`

FormaV - document d'entraînement non officiel

Partie 2 : Interaction avec la base de données

Question 4.

Sujet p. 10

> Associer à chaque mot-clé sa définition : SELECT, INSERT et UPDATE ; ajouter une donnée, extraire des données et mettre à jour une table.

`SELECT` sert à sélectionner, donc à extraire des données. `INSERT` ajoute une nouvelle donnée et `UPDATE` modifie des données déjà présentes dans une table.

> **RÉPONSE À RETENIR**
>
> **SELECT** : extraire des données.
>
> **INSERT** : ajouter une donnée.
>
> **UPDATE** : mettre à jour une table.

Question 5.

Sujet p. 10

> Définir ce qu'est une clé primaire.

Une clé primaire sert à identifier de manière unique chaque ligne d'une table. Sa valeur ne peut donc pas être dupliquée ni être absente.

> **RÉPONSE À RETENIR**
>
> Une clé primaire est un attribut, ou un ensemble d'attributs, qui identifie de façon unique chaque enregistrement d'une table ; elle est unique et non nulle.

Question 6.

Sujet p. 10

> Indiquer quel attribut, éventuellement plusieurs, serait le plus adapté pour être une clé primaire dans la table questions_reponses et justifier votre choix. La table comporte les colonnes id, question, reponse et mot_cle.

L'identifiant `id` est le choix le plus adapté : il permet de distinguer chaque enregistrement, même si plusieurs questions ont le même texte ou le même mot-clé. Le sujet ne donne pas de contrainte d'unicité sur `question` ou `mot_cle`.

> **RÉPONSE À RETENIR**
>
> La clé primaire la plus adaptée est `id`, car chaque ligne peut ainsi être identifiée sans ambiguïté.

Question 7.

Sujet p. 10

> Écrire une requête permettant de récupérer toutes les questions comportant comme mot-clé « python ».

Les questions figurent dans la colonne `question` et leur mot-clé dans `mot_cle`. On sélectionne donc les questions dont le mot-clé est exactement `python`.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
SELECT question
FROM questions_reponses
WHERE mot_cle = 'python';
```

Aucun extrait de la table n'étant fourni, le résultat des lignes sélectionnées ne peut pas être détaillé.

Question 8.

Sujet p. 10

> Écrire une requête pour insérer la nouvelle question-réponse « Date de la bataille de Marignan ? », « 1515 », associée au mot-clé « Histoire ». On prendra comme identifiant Id = 15.

La nouvelle ligne fournit une valeur pour chacune des quatre colonnes : l'identifiant 15, le texte de la question, la réponse et le mot-clé. Les valeurs textuelles sont encadrées par des apostrophes SQL.

RÉPONSE À RETENIR

```
INSERT INTO questions_reponses (id, question, reponse, mot_cle)
VALUES (15, 'Date de la bataille de Marignan ?', '1515', 'Histoire');
```

Question 9.

Sujet p. 10

> Écrire la requête pour mettre à jour la réponse à la question d'Id = 12, qui devient « Oui ».

La clause `SET` donne la nouvelle valeur de la réponse. La condition sur `id` limite la modification à la ligne d'identifiant 12.

RÉPONSE À RETENIR

```
UPDATE questions_reponses
SET reponse = 'Oui'
WHERE id = 12;
```

Partie 3 : Représentation de la recherche avec les arbres

Question 10.

Sujet p. 10

> Indiquer quel type de parcours donnerait, dans cet ordre, les mots-clés IA, bot, programmation, algo, chatbot, informatique, python.

On visite d'abord la racine IA, puis ses enfants bot et programmation, puis les nœuds du niveau suivant. C'est un parcours par niveaux, appelé parcours en largeur.

RÉPONSE À RETENIR

Le parcours demandé est le **parcours en largeur** : IA, bot, programmation, algo, chatbot, informatique, python.

FormaV - document d'entraînement non officiel

Question 11.

Sujet p. 10

> Indiquer dans quel ordre apparaîtraient les mots-clés avec un parcours en profondeur infixe sur l'ABR illustré.

Un parcours en profondeur infixe visite le sous-arbre gauche, le nœud courant, puis le sous-arbre droit. Le sous-arbre gauche de IA donne algo, bot, chatbot ; on visite ensuite IA, puis le sous-arbre droit donne informatique, programmation, python.

> **RÉPONSE À RETENIR**
>
> L'ordre est : **algo, bot, chatbot, IA, informatique, programmation, python**.

Question 12.

Sujet p. 10

> Indiquer dans quel ordre apparaîtraient les mots-clés pour un parcours en profondeur postfixe (ou suffixe).

Un parcours postfixe (ou suffixe) visite d'abord le sous-arbre gauche, puis le sous-arbre droit, et termine par le nœud courant. Le sous-arbre gauche donne algo, chatbot, bot ; le droit donne informatique, python, programmation ; la racine IA est visitée en dernier.

> **RÉPONSE À RETENIR**
>
> L'ordre est : **algo, chatbot, bot, informatique, python, programmation, IA**.

Question 13.

Sujet p. 10

> Indiquer quel parcours permettrait d'obtenir les mots-clés dans l'ordre alphabétique.

Dans un arbre binaire de recherche, les clés du sous-arbre gauche précèdent la clé du nœud, et celles du sous-arbre droit lui succèdent. Le parcours infixe visite donc les clés dans l'ordre alphabétique.

> **RÉPONSE À RETENIR**
>
> Il faut utiliser un **parcours en profondeur infixe**.

Question 14.

Sujet p. 11

> Compléter la méthode de recherche dans l'ABR : compléter le test de la ligne 9 et les attributs à utiliser dans les appels récursifs des lignes 12 et 14.

Si l'arbre est vide ou si sa clé est celle recherchée, la recherche renvoie le nœud courant. Sinon, la propriété de l'ABR permet de poursuivre à gauche lorsque `cle` est inférieure à la clé du nœud, et à droite dans l'autre cas. Les attributs déclarés sont `gauche` et `droit`.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def recherche(arbre, cle):
    if arbre is None or arbre.cle == cle:
        return arbre
    if cle < arbre.cle:
        return recherche(arbre.gauche, cle)
    else:
        return recherche(arbre.droit, cle)
```

La fonction renvoie le nœud contenant la clé si elle est présente, et `None` si la recherche atteint un sous-arbre vide.

Sujet : https://www.sujetdebac.fr/annales-pdf/2026/spe-numerique-informatique-2026-polynesie-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.