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 (NSI)

Session 2026 • Antilles-Guyane • Jour 1

Code sujet **Non indiqué sur le sujet fourni** Non indiquée sur le sujet fourni

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Bases de données et langage SQL
12 réponses • sujet p. 1–2

Exercice 2 — Algorithmique et programmation Python : recherche d’une plus longue sous-liste commune (LCS)
8 réponses • sujet p. 3–5

Exercice 3 — Routage, sécurisation des communications et programmation orientée objet
16 réponses • sujet p. 6–8

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

Bases de données et langage SQL

Question 1.

Sujet p. 1

> Donner deux avantages que peut apporter un système de gestion de bases de données par rapport à l'utilisation d'un simple tableur dans lequel on ajouterait chaque nouvelle commande sur une ligne.

Un système de gestion de bases de données organise les informations en tables reliées et permet de contrôler les relations entre elles. Par rapport à un tableur où toutes les commandes seraient ajoutées ligne par ligne, il apporte notamment une meilleure cohérence des données et facilite les recherches.

RÉPONSE À RETENIR

- **Réduction des incohérences et des répétitions :** les informations des clients ou des pizzas peuvent être stockées une seule fois et reliées aux commandes ; les clés et les contraintes peuvent aussi contrôler les données.
- **Recherches et traitements facilités :** SQL permet de filtrer, trier et regrouper rapidement les commandes, y compris en reliant plusieurs tables.

Question 2.

Sujet p. 1

> Indiquer, en justifiant, s'il est possible d'avoir deux pizzas avec le même numéro de pizza, le même prix et une couleur de pizza différente.

Dans la relation `pizza`, l'attribut `num` est la clé primaire. Une clé primaire identifie une ligne de manière unique : deux lignes de cette relation ne peuvent donc pas avoir le même numéro.

RÉPONSE À RETENIR

Non. Il est impossible d'avoir deux pizzas de même numéro, même si elles ont le même prix et des couleurs différentes, car cela violerait l'unicité de la clé primaire `pizza.num`.

Question 3.

Sujet p. 1

> Donner un exemple de clé étrangère en précisant la relation à laquelle elle appartient, ainsi que l'attribut et la relation qu'elle référence.

Une clé étrangère est un attribut d'une relation qui référence la clé primaire d'une autre relation. On peut prendre comme exemple l'identifiant du client associé à une commande.

RÉPONSE À RETENIR

`commande.id_client` est une clé étrangère de la relation `commande` : elle référence l'attribut `client.id` de la relation `client`.

Question 4.

Sujet p. 1, 2

> En s'appuyant uniquement sur les extraits des tables fournis précédemment, écrire le résultat renvoyé par la

FormaV - document d'entraînement non officiel

> requête SQL : SELECT prix FROM pizza WHERE couleur = 'Bianca'.

La requête sélectionne les prix des lignes de la table `pizza` dont la couleur est exactement `Bianca`. Dans l'extrait, deux lignes correspondent : celles des pizzas numéros 2 et 4. La requête ne contient pas de `ORDER BY` ; l'ordre d'affichage des lignes n'est donc pas garanti par SQL.

RÉPONSE À RETENIR

Le résultat contient les valeurs suivantes :

prix
11.5
15.5

Question 5.

Sujet p. 1, 2

> Écrire une requête SQL qui donne les couleurs des pizzas dont le prix est strictement supérieur à 15 €.

Il faut sélectionner l'attribut `couleur` dans la table `pizza` et filtrer avec une condition strictement supérieure à 15. Le symbole > exclut un prix égal à 15.

RÉPONSE À RETENIR

```
SELECT couleur FROM pizza WHERE prix > 15;
```

Sur l'extrait fourni, la seule ligne qui satisfait la condition est la pizza de prix 15.5 ; la requête renvoie donc `Bianca`.

Question 6.

Sujet p. 1, 2

> Écrire une requête SQL qui donne les prénoms des clients qui n'ont pas payé une pizza qui leur a pourtant été livrée.

Les informations nécessaires sont dans `commande` et `client`. On les relie avec `commande.id_client = client.id`, puis on conserve les commandes livrées (`livraison = 1`) mais non réglées (`paiement = 0`). `DISTINCT` évite de répéter un prénom si plusieurs commandes satisfont ces conditions.

RÉPONSE À RETENIR

```
SELECT DISTINCT client.prenom
FROM client
JOIN commande ON client.id = commande.id_client
WHERE commande.livraison = 1 AND commande.paiement = 0;
```

Sur les extraits fournis, le résultat est `Raffaele` : la commande 42364 est livrée et non payée.

Question 7.

Sujet p. 1, 2

> Écrire une requête SQL qui donne le prix moyen de vente des pizzas commandées par un client de nom « Esposito

».

On relie les trois tables : le client à ses commandes par son identifiant, puis chaque commande à la pizza par son numéro. La fonction `AVG` calcule la moyenne des prix des pizzas de ces commandes. Chaque commande correspondant à une pizza, chaque pizza commandée est prise en compte.

RÉPONSE À RETENIR

```
SELECT AVG(pizza.prix)
FROM client
JOIN commande ON client.id = commande.id_client
JOIN pizza ON commande.num_pizza = pizza.num
WHERE client.nom = 'Esposito';
```

Dans l'extrait, Esposito a commandé les pizzas de prix 11.5, 12.5 et 11.5. La moyenne renvoyée est $(11.5 + 12.5 + 11.5)/3 = 35.5/3$, soit environ **11.83**.

Question 8.

Sujet p. 1, 2

La commande de code 42365 vient tout juste d'être livrée. Écrire une requête SQL qui permet de mettre à jour la base de données en conséquence.

La commande concernée est identifiée par sa clé primaire, le code 42365. Il faut mettre à jour uniquement son attribut `livraison` et lui attribuer la valeur 1, qui signifie que la pizza a été livrée.

RÉPONSE À RETENIR

```
UPDATE commande
SET livraison = 1
WHERE code = 42365;
```

Question 9.

Sujet p. 1, 2

Le 3 janvier 2025, Emanuele Girasole, qui n'a jamais encore commandé dans ce restaurant, souhaite se faire livrer la pizza numéro 2. Écrire, dans le bon ordre, les requêtes SQL qui permettent de mettre à jour la base de données en conséquence. On suppose que l'identifiant client 107 et que le code de commande 42366 n'ont pas encore été attribués dans la base de données.

Il faut d'abord créer le client, puis insérer sa commande afin que la valeur `id_client = 107` référence un client existant. Le nom est `Girasole` et le prénom `Emanuele`. La date est le 3 janvier 2025, écrite au format utilisé dans la table : `2025-01-03`. La commande souhaitée n'est pas encore livrée : on indique donc `livraison = 0`. Aucun paiement n'étant mentionné, on prend `paiement = 0` comme convention, sans supposer qu'un règlement a été effectué.

RÉPONSE À RETENIR

Dans cet ordre :

```
INSERT INTO client (id, nom, prenom)
VALUES (107, 'Girasole', 'Emanuele');

INSERT INTO commande (code, id_client, num_pizza, date, livraison, paiement)
VALUES (42366, 107, 2, '2025-01-03', 0, 0);
```

Question 10.

Sujet p. 1, 2

> La gérante de la pizzeria décide de retirer de la vente la pizza d'identifiant 1 et exécute la requête DELETE FROM pizza WHERE num = 1. Indiquer le problème que peut poser l'exécution de cette requête.

La pizza de numéro 1 est référencée par les commandes 42364 et 42365 de l'extrait. Si la clé étrangère `commande.num_pizza` référence `pizza.num` et que l'intégrité référentielle est appliquée, la suppression de cette pizza est refusée tant que ces commandes la référencent. Sans cette contrainte, ces commandes conserveraient un numéro de pizza qui ne correspondrait plus à aucune ligne de `pizza`. Une suppression en cascade risquerait, elle, d'effacer l'historique des commandes concernées.

RÉPONSE À RETENIR

Le problème est la perte de cohérence entre `pizza` et `commande` : la pizza 1 est encore référencée par des commandes. Il faut préserver les références et l'historique, par exemple en refusant la suppression ou en mettant la pizza hors vente sans supprimer sa ligne.

Question 11.

Sujet p. 1, 2

> Chaque pizza comporte divers ingrédients et un même ingrédient peut être utilisé sur plusieurs pizzas. Écrire le schéma relationnel des deux nouvelles tables, ingredient et composition, permettant de faire le lien entre les ingrédients et les pizzas, en indiquant clairement les clés primaires et les clés étrangères.

La relation `composition` modélise l'association plusieurs-à-plusieurs : une pizza peut avoir plusieurs ingrédients et un ingrédient peut apparaître dans plusieurs pizzas. La paire des identifiants est donc une clé primaire composée, ce qui interdit de répéter deux fois la même association.

RÉPONSE À RETENIR

Un schéma possible est :

- `ingredient(id_ingredient : entier, nom : texte)`, avec `id_ingredient` comme clé primaire.
- `composition(num_pizza : entier, id_ingredient : entier)`, avec la clé primaire composée `(num_pizza, id_ingredient)`, `num_pizza` comme clé étrangère référençant `pizza.num` et `id_ingredient` comme clé étrangère référençant `ingredient.id_ingredient`.

Question 12.

Sujet p. 2

> Le prestataire propose de placer la base de données sur un serveur accessible par des requêtes extérieures, et non uniquement via le réseau interne utilisé par le restaurant, afin que chaque client puisse consulter sur Internet l'historique de ses commandes. Indiquer les problèmes soulevés par cette démarche et les précautions à prendre

pour y remédier.

Rendre le serveur accessible depuis Internet augmente les risques d’accès non autorisé et d’exposition de données personnelles, notamment si un client peut consulter les commandes d’un autre. Des requêtes malveillantes pourraient aussi lire, modifier ou supprimer des données ; le service peut enfin être indisponible à la suite d’une attaque ou d’une panne.

RÉPONSE À RETENIR

- Mettre en place une authentification fiable et des autorisations vérifiées côté serveur : chaque client ne doit pouvoir consulter que son propre historique. Ne pas exposer directement la base aux requêtes des navigateurs ; passer par une application ou une API contrôlée.
- Protéger les échanges avec HTTPS, limiter les droits des comptes techniques et sécuriser les requêtes SQL, notamment avec des requêtes paramétrées, afin de réduire les risques d’interception et d’injection SQL.
- Maintenir les logiciels à jour, filtrer les accès réseau, journaliser les opérations utiles et réaliser des sauvegardes protégées pour limiter les effets des attaques et des pannes.
- Ne conserver et ne rendre accessibles que les données nécessaires, en protégeant les données personnelles conformément aux règles applicables.

Exercice 2

6 points

Algorithmique et programmation Python : recherche d'une plus longue sous-liste commune (LCS)

A

Question 1.

Sujet p. 3

> Pour un appel initial sous_listes([9, 3, 7], [[]]), recopier et compléter le tableau indiquant les valeurs des paramètres lors des appels récursifs : appel initial, 1er sous-appel, 2nd sous-appel et dernier sous-appel. Le tableau comporte les colonnes « liste » et « accumulateur ».

À chaque appel, la fonction retire le dernier élément de la liste. Elle conserve les sous-listes déjà dans l'accumulateur et ajoute celles obtenues en plaçant l'élément retiré en tête de chacune.

Après le premier appel, l'élément retiré est 7 : on ajoute `[7]`. Au sous-appel suivant, on retire 3 et on ajoute `[3]` puis `[3, 7]`. Enfin, on retire 9 et on ajoute les quatre sous-listes commençant par 9.

Appel	liste	accumulateur
appel initial	[9, 3, 7]	[[]]
1er sous-appel	[9, 3]	[[], [7]]
2nd sous-appel	[9]	[[], [7], [3], [3, 7]]
dernier sous-appel	[]	[[], [7], [3], [3, 7], [9], [9, 7], [9, 3], [9, 3, 7]]

RÉPONSE À RETENIR

Le dernier sous-appel reçoit la liste vide et l'accumulateur contenant les huit sous-listes, y compris la liste vide.

Question 2.

Sujet p. 3

> Recopier et compléter les lignes 7 et 8 du code Python de la fonction sous_listes.

Après `liste.pop()`, `liste` est privée de son dernier élément, stocké dans `element`. Pour chaque sous-liste existante, on en crée une nouvelle en plaçant `element` en tête. L'appel suivant reçoit la liste diminuée et la concaténation de l'ancien accumulateur avec `tmp`.

RÉPONSE À RETENIR

```
        tmp.append([element] + sous_liste)
    return sous_listes(liste, accumulateur + tmp)
```

Question 3.

Sujet p. 3, 4

> Expliquer pourquoi l'exécution de la fonction lcs_force_brute risque de prendre beaucoup de temps selon la longueur des listes passées en paramètre. Justifier.

Si `liste_1` comporte *m* éléments, chacune de ses sous-listes correspond à un choix, pour chaque élément, de le garder ou non. Il y a donc `2**m` sous-listes. La fonction `sous_listes` les construit toutes et `lcs_force_brute` les examine toutes.

Pour chacune, `est_extraite` peut parcourir jusqu'à *n* éléments de `liste_2`, si celle-ci a une longueur *n*.

Les tests d'extraction coûtent donc jusqu'à environ $n \times 2^m$ opérations. La construction des sous-listes ajoute elle aussi un coût, mais la croissance exponentielle en *m* explique déjà que le calcul puisse devenir très long.

> **RÉPONSE À RETENIR**
>
> La recherche exhaustive est coûteuse : elle génère et teste `2**m` sous-listes. Le nombre de traitements double lorsque la longueur de `liste_1` augmente d'une unité.

B

Question 4.

Sujet p. 4

> Indiquer ce que renvoie l'appel lcs([9, 3, 7, 5, 8], 1, [9, 7, 8, 3, 7, 3], 1).

Les paramètres 1 et 1 désignent les préfixes constitués du premier élément de chaque liste, soit `[9]` et `[9]`. Leur plus longue sous-liste commune est `[9]`.

> **RÉPONSE À RETENIR**
>
> L'appel renvoie `[9]`.

Question 5.

Sujet p. 4

> On suppose que L_1 et L_2 sont des listes non vides, de tailles respectives m et n, et que leurs derniers éléments sont égaux : $L_1[m-1] = L_2[n-1]$. Écrire la relation de récurrence permettant d'obtenir une plus longue sous-liste commune à L_1 et L_2 à partir du résultat de lcs(L_1, m−1, L_2, n−1).

Les derniers éléments des deux listes considérées sont égaux. On peut donc ajouter cet élément à la fin d'une plus longue sous-liste commune des préfixes qui le précèdent. Les paramètres de `lcs` indiquent les longueurs des préfixes : les éléments ajoutés ont donc les indices `m-1` et `n-1`.

> **RÉPONSE À RETENIR**
>
> $lcs(L_1, m, L_2, n) = lcs(L_1, m-1, L_2, n-1) + [L_1[m-1]]$.
>
> Comme $L_1[m-1] = L_2[n-1]$, l'élément ajouté appartient aux deux listes et respecte leur ordre.

FormaV - document d'entraînement non officiel

Question 6.

Sujet p. 5

> Justifier que la fonction aux contenue dans la fonction lcs_top_down est récursive.

Une fonction est récursive si son corps contient un appel à cette même fonction. Le corps de `aux` contient plusieurs appels à `aux` : un lorsque les éléments comparés sont égaux et deux lorsqu'ils sont différents. Les appels finissent par atteindre le cas d'arrêt, car au moins un indice diminue à chaque appel.

> **RÉPONSE À RETENIR**
>
> `aux` est récursive parce qu'elle s'appelle elle-même dans son propre corps.

Question 7.

Sujet p. 5

> Détailler le risque d'un tel choix dans la présente situation, lorsque les listes L_1 et L_2 sont très grandes.

Les appels récursifs occupent la pile d'exécution. Le long d'une branche, il peut y avoir jusqu'à environ $m+n$ diminutions d'indices avant d'atteindre le cas d'arrêt. Si les listes sont très grandes, la profondeur des appels peut dépasser la limite de récursion de Python et provoquer une erreur, interrompant le calcul.

Par ailleurs, la fonction ne mémorise pas les résultats des sous-problèmes : elle peut donc recalculer plusieurs fois les mêmes cas lorsque les éléments comparés sont différents, ce qui peut aussi rendre le calcul très long. La mémoïsation limiterait ces recalculs, mais ne supprimerait pas le risque d'une pile d'appels trop profonde.

> **RÉPONSE À RETENIR**
>
> Pour de très grandes listes, cette récursion peut dépasser la limite de profondeur de Python et entraîner une erreur. L'absence de mémorisation peut en outre provoquer de nombreux recalculs.

Question 8.

Sujet p. 5

> Proposer une solution pour éviter le problème évoqué à la question 7.

Pour éviter le risque lié à la profondeur de récursion, on peut calculer la LCS par programmation dynamique itérative. On remplit un tableau `dp` où `dp[i][j]` est la longueur d'une LCS des préfixes de longueurs `i` et `j`. On reconstruit ensuite une LCS en remontant dans ce tableau. Voici une fonction complète :

FormaV - document d'entraînement non officiel

```
def lcs_iterative(liste_x, liste_y):
    m = len(liste_x)
    n = len(liste_y)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if liste_x[i - 1] == liste_y[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    resultat = []
    i = m
    j = n
    while i > 0 and j > 0:
        if liste_x[i - 1] == liste_y[j - 1]:
            resultat.append(liste_x[i - 1])
            i -= 1
            j -= 1
        elif dp[i - 1][j] >= dp[i][j - 1]:
            i -= 1
        else:
            j -= 1

    resultat.reverse()
    return resultat
```

Les deux boucles remplissent `dp` à partir des préfixes vides, pour lesquels la longueur de la LCS vaut 0. Lors de la reconstruction, chaque étape diminue au moins l'un des deux indices : la boucle termine. Le calcul prend un temps de l'ordre de $m \times n$ et utilise un tableau de taille `(m + 1) × (n + 1)`.

Avec les listes du sujet, la remontée part de `(5, 6)`. Elle descend à `(4, 6)`, puis à `(3, 6)`, et se déplace à `(3, 5)`, où les deux éléments comparés valent 7. Elle retrouve ensuite 3, puis 9. Les éléments recueillis en sens inverse sont donc remis dans l'ordre par `reverse()`.

> **RÉPONSE À RETENIR**
>
> La programmation dynamique itérative évite les appels récursifs. Pour les listes `[9, 3, 7, 5, 8]` et `[9, 7, 8, 3, 7, 3]`, la fonction proposée renvoie `[9, 3, 7]`, une LCS de longueur 3.

Exercice 3

8 points

Routage, sécurisation des communications et programmation orientée objet

A

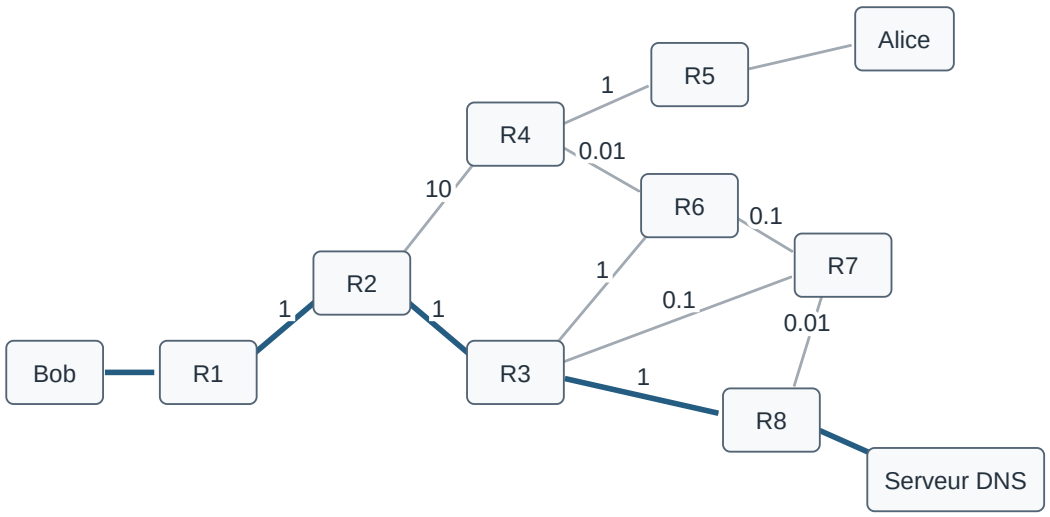
Question 1.

Sujet p. 6

> En utilisant le protocole RIP, donner le chemin que les paquets emprunteront afin qu'une requête de Bob parvienne au serveur DNS. Ne faire apparaître que les routeurs traversés.

RIP choisit un chemin qui minimise le nombre de routeurs traversés, sans tenir compte des coûts inscrits sur les liaisons. De Bob au serveur DNS, le chemin passant par R1, R2, R3 et R8 comporte quatre routeurs. Les autres chemins possibles vers le DNS traversent davantage de routeurs.

Réseau et chemin RIP de Bob au serveur DNS



Les liaisons et les coûts sont ceux de la figure du sujet ; les liaisons du chemin RIP sont mises en évidence.

RÉPONSE À RETENIR

Routeurs traversés : **R1** → **R2** → **R3** → **R8**.

Question 2.

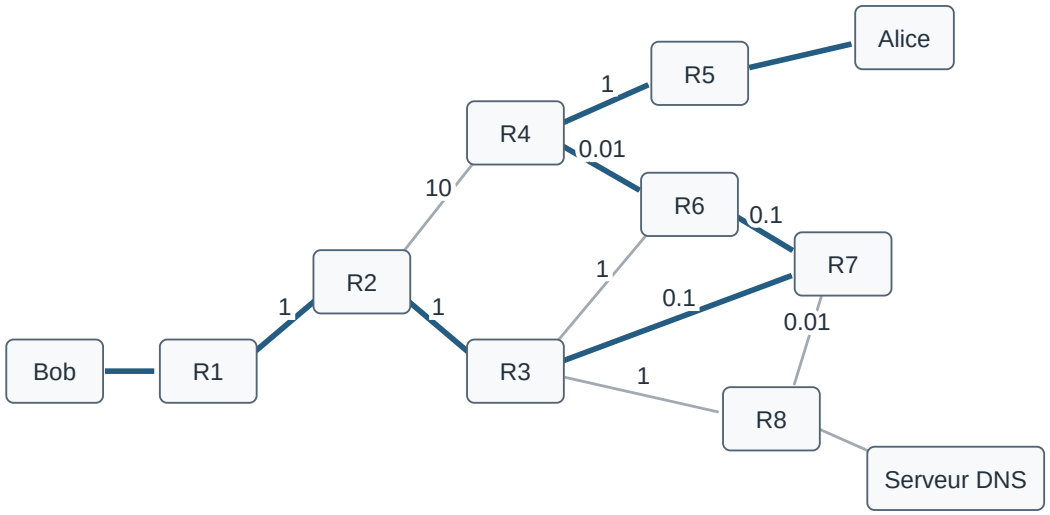
Sujet p. 6

> On suppose que Bob a l'adresse IP d'Alice. En utilisant le protocole OSPF, donner le chemin que les paquets emprunteront afin qu'une requête de Bob parvienne à Alice. Ne faire apparaître que les routeurs traversés.

OSPF minimise la somme des coûts des liaisons. Depuis R2, passer par R4 puis R5 coûte 10 + 1 = 11. En passant par R3, puis R7, R6, R4 et R5, le coût est 1 + 0,1 + 0,1 + 0,01 + 1 = 2,21 : ce trajet est donc plus court. Depuis Bob, il faut aussi traverser R1 et la liaison R1–R2. Les liaisons Bob–R1 et R5–Alice n'ont pas de coût indiqué sur la figure.

FormaV - document d'entraînement non officiel

Réseau et chemin OSPF de Bob à Alice



Les liaisons et les coûts sont ceux de la figure du sujet ; le chemin OSPF vers Alice est mis en évidence.

RÉPONSE À RETENIR

Chemin : Bob → R1 → R2 → R3 → R7 → R6 → R4 → R5 → Alice. Les routeurs traversés sont **R1, R2, R3, R7, R6, R4 et R5**.

Question 3.

Sujet p. 6

> Donner le coût du chemin identifié à la question 2.

On additionne les coûts des liaisons du chemin trouvé à la question 2 : R1–R2, R2–R3, R3–R7, R7–R6, R6–R4 et R4–R5. Les liaisons Bob–R1 et R5–Alice ne portent pas de coût indiqué sur la figure et ne sont donc pas ajoutées.

RÉPONSE À RETENIR

Coût : 1 + 1 + 0,1 + 0,1 + 0,01 + 1 = **3,21**.

Question 4.

Sujet p. 6

> Expliquer ce qui se passerait si le routeur R3 tombe en panne. L'écriture des tables de routage n'est pas attendue.

Si R3 tombe en panne, les liaisons qui le relient aux autres routeurs ne peuvent plus être empruntées. Il subsiste cependant un chemin vers Alice par R1, R2, R4 et R5, ainsi qu'un chemin vers le serveur DNS par R1, R2, R4, R6, R7 et R8. Le routage peut donc s'adapter à la panne et utiliser ces chemins de remplacement.

RÉPONSE À RETENIR

La panne de R3 ne coupe pas le réseau : les paquets peuvent être reroutés vers Alice ou vers le serveur DNS par des chemins qui évitent R3.

FormaV - document d'entraînement non officiel

Question 5.

Sujet p. 6

> Indiquer un protocole permettant à un navigateur et à un serveur web de communiquer en chiffrant leurs données.

TLS est un protocole qui sécurise les échanges en chiffrant les données. HTTPS désigne HTTP utilisé avec TLS.

> **RÉPONSE À RETENIR**
>
> On peut citer **HTTPS**, qui utilise TLS pour sécuriser les échanges entre le navigateur et le serveur web.

Question 6.

Sujet p. 6

> Bob génère une clé symétrique qu'il doit transmettre à Alice. Bob et Alice possèdent alors cette même clé symétrique. Expliquer le rôle qu'aura cette clé symétrique dans leur communication.

En cryptographie symétrique, Bob et Alice possèdent le même secret. Bob chiffre les données avec cette clé et Alice les déchiffre avec cette même clé ; Alice peut aussi chiffrer des données que Bob déchiffrera avec cette clé. Le chiffrement vise notamment la confidentialité, mais ne garantit pas à lui seul l'intégrité des données ni l'identité de leur expéditeur.

> **RÉPONSE À RETENIR**
>
> La clé symétrique sert à chiffrer les données envoyées et à les déchiffrer à leur réception, par Bob comme par Alice.

Question 7.

Sujet p. 6

> Alice génère un couple de clés composé d'une clé publique et d'une clé privée. Expliquer comment Alice et Bob vont utiliser ces deux clés afin que Bob puisse transmettre de façon sécurisée sa clé symétrique à Alice. Ne pas parler de signature ni de certificat.

Bob chiffre la clé symétrique avec la clé publique d'Alice et transmet le résultat. Alice utilise sa clé privée, qu'elle seule doit connaître, pour déchiffrer ce message et retrouver la clé symétrique. La confidentialité suppose que la clé privée d'Alice reste secrète et que Bob utilise bien la clé publique d'Alice.

> **RÉPONSE À RETENIR**
>
> Bob chiffre la clé symétrique avec la **clé publique d'Alice** ; Alice la déchiffre avec sa **clé privée**.

Question 8.

Sujet p. 6, 7

> Bob envoie sa clé publique à Alice, crée un condensé des données de la clé symétrique et le signe avec sa propre clé privée, puis transmet à Alice l'ensemble composé de la clé symétrique et de la signature, chiffré avec la clé publique d'Alice. Expliquer pourquoi Alice est la seule à pouvoir accéder à la clé symétrique et peut avoir la certitude que c'est bien Bob qui la lui a envoyée.

Alice déchiffre avec sa clé privée le message chiffré avec sa clé publique : si cette clé privée est restée secrète, un tiers qui intercepte le message ne peut pas accéder à la clé symétrique. Alice vérifie ensuite la signature avec la clé publique de Bob : elle compare le condensé obtenu en vérifiant la signature au condensé calculé à partir de la clé symétrique reçue. S'ils correspondent, la clé reçue correspond aux données signées par le détenteur de la clé privée de signature ; cela fournit un contrôle d'intégrité et d'origine.

FormaV - document d'entraînement non officiel

Cette conclusion suppose qu'Alice possède bien la véritable clé publique de Bob. Le fait que Bob lui transmette cette clé publique ne suffit pas, à lui seul, à prouver qu'elle lui appartient : un attaquant pourrait substituer sa propre clé. Il faut donc une association de confiance entre la clé publique et Bob. De même, la confidentialité dépend du secret de la clé privée d'Alice.

RÉPONSE À RETENIR

Sous ces hypothèses de confiance, le chiffrement avec la clé publique d'Alice assure la confidentialité de la clé symétrique, et la vérification de la signature avec la véritable clé publique de Bob permet de contrôler son origine et son intégrité.

B

Question 9.

Sujet p. 7

Écrire le code d'une fonction somme qui prend en paramètre un tableau (type list en Python) d'entiers et renvoie la somme de ses éléments. Par exemple, somme([1, 3, 5, 7, 9]) renvoie 25.

On initialise un accumulateur à 0 et on y ajoute chaque élément du tableau. La somme des éléments est alors renvoyée ; pour un tableau vide, la fonction renvoie 0.

RÉPONSE À RETENIR

```
def somme(tab):
    total = 0
    for element in tab:
        total += element
    return total
```

Par exemple, `somme([1, 3, 5, 7, 9])` renvoie `25`.

Question 10.

Sujet p. 7

Écrire le code d'une fonction permuter qui prend en paramètres un tableau tab (type list en Python) et deux entiers i et j, puis modifie tab en permutant ses éléments d'indices i et j. Par exemple, après permuter(liste, 1, 3) avec liste = [1, 3, 5, 7, 9], liste vaut [1, 7, 5, 3, 9].

L'affectation simultanée en Python échange les deux valeurs sans variable temporaire. Les indices `i` et `j` désignent les positions à permuter ; le tableau est modifié sur place.

RÉPONSE À RETENIR

```
def permuter(tab, i, j):
    tab[i], tab[j] = tab[j], tab[i]
```

Avec `liste = [1, 3, 5, 7, 9]`, l'appel `permuter(liste, 1, 3)` donne `[1, 7, 5, 3, 9]`.

Question 11.

Sujet p. 7

Recopier et compléter la ligne 3 du code de la fonction inverser, qui modifie le tableau passé en paramètre en

> inversant l'ordre de ses éléments : def inverser(tab): for i in range(len(tab) // 2): permuter(...).

Chaque élément d'indice `i` de la première moitié du tableau est échangé avec l'élément symétrique situé à l'indice `len(tab) - 1 - i`. La boucle s'arrête à la moitié de la longueur pour ne pas échanger deux fois les mêmes paires.

RÉPONSE À RETENIR

```
permuter(tab, i, len(tab) - 1 - i)
```

Question 12.

Sujet p. 7

> Recopier et compléter les lignes 7 à 13 du code de la fonction generer_sac. La fonction doit créer un sac de 8 entiers : le premier est choisi au hasard entre 1 et 5 inclus ; pour chacun des 7 entiers suivants, calculer la somme S de tous les éléments du sac, choisir au hasard un entier entre S+1 et 2*S inclus, puis l'ajouter à la suite des précédents. Lorsque le sac contient huit entiers, inverser l'ordre de ses éléments afin que le premier soit le plus grand, puis renvoyer le sac. La fonction randint(a, b) du module random renvoie un entier aléatoire entre a et b inclus.

On tire le premier entier entre 1 et 5 inclus. Pour chacun des sept ajouts, on calcule la somme courante `s`, puis on tire un entier entre `s + 1` et `2 * s`, bornes incluses, avant de l'ajouter au sac. Une fois les huit entiers obtenus, on appelle `inverser`, définie à la question 11, pour placer le plus grand en tête.

RÉPONSE À RETENIR

```
def generer_sac():
    '''
    renvoie un sac (8 entiers)
    : return (list) un sac
    '''
    sac = [randint(1, 5)]
    for i in range(7):
        s = sum(sac)
        nouvel_entier = randint(s + 1, 2 * s)
        sac.append(nouvel_entier)
    inverser(sac)
    return sac
```

C

Question 13.

Sujet p. 8

> La classe Cle_symetrique est initialisée avec un unique attribut sac, qui est un tableau renvoyé par la fonction generer_sac du module outils. Recopier et compléter la ligne 4 du code de la classe Cle_symetrique.

L'unique attribut de l'objet est initialisé avec le sac renvoyé par la fonction `generer_sac` du module `outils`. L'appel à cette fonction doit donc être qualifié par le nom du module.

RÉPONSE À RETENIR

```
self.sac = outils.generer_sac()
```

Question 14.

Sujet p. 8

Recopier et compléter les lignes 8 à 11 du code de la méthode chiffrer. Cette méthode chiffre un octet représenté par un tableau de huit bits en multipliant chaque bit d'indice i par l'entier d'indice i du sac, puis en renvoyant la somme des produits obtenus.

L'octet contient huit bits : on parcourt donc les indices de 0 à 7. Pour chaque indice, on multiplie le bit par la valeur correspondante du sac, puis on additionne les produits.

RÉPONSE À RETENIR

```
def chiffrer(self, octet):
    '''
    renvoie le chiffrement de octet
    : paramètre octet (list) un tableau de huit 0 ou 1
    : return (int)
    '''
    tab_produits = []
    for i in range(8):
        produit = octet[i] * self.sac[i]
        tab_produits.append(produit)
    s = sum(tab_produits)
    return s
```

Question 15.

Sujet p. 8

Justifier qu'avec la méthode dechiffrer décrite, le déchiffrement du nombre 688 est [1, 0, 1, 1, 1, 1, 0, 1]. Pour chaque indice de 0 à 7, la méthode met le bit à 1 et soustrait la valeur correspondante du sac si l'entier à déchiffrer lui est supérieur ou égal ; sinon, elle met le bit à 0.

On applique la règle décrite en parcourant le sac du plus grand poids au plus petit. À chaque étape, si le reste est supérieur ou égal au poids considéré, le bit vaut 1 et on soustrait ce poids ; sinon le bit vaut 0 et le reste ne change pas.

RÉPONSE À RETENIR

Indice i	Valeur du sac	Reste avant	Bit	Reste après
0	568	688	1	120
1	205	120	0	120
2	71	120	1	49
3	32	49	1	17
4	10	17	1	7
5	6	7	1	1
6	2	1	0	1
7	1	1	1	0

Les bits obtenus dans l'ordre des indices sont **[1, 0, 1, 1, 1, 1, 0, 1]**.

Question 16.

Sujet p. 8

Écrire le code de la méthode dechiffrer de la classe Cle_symetrique. Elle prend en paramètre un entier à déchiffrer et renvoie un tableau représentant l'octet correspondant. L'exemple fourni utilise ma_cle.sac = [568, 205, 71, 32, 10, 6, 2, 1] et demande que ma_cle.dechiffrer(688) renvoie [1, 0, 1, 1, 1, 1, 0, 1].

On conserve le reste à déchiffrer dans le paramètre `entier`. Pour chaque indice de 0 à 7, on ajoute 1 au tableau résultat et on soustrait le poids correspondant si le reste est assez grand ; sinon on ajoute 0. Le tableau ainsi construit conserve l'ordre des bits de l'octet.

RÉPONSE À RETENIR

```
def dechiffrer(self, entier):
    octet = []
    for i in range(8):
        if entier >= self.sac[i]:
            octet.append(1)
            entier -= self.sac[i]
        else:
            octet.append(0)
    return octet
```

Avec le sac indiqué dans le sujet, `dechiffrer(688)` renvoie `[1, 0, 1, 1, 1, 1, 0, 1]`.

Sujet : https://aeif.fr/wp-content/uploads/2026/06/Antilles-Guyane_juin_2026_sujet_1.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.