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 • Antilles-Guyane • Jour 2

Code sujet **26-NSIJ2AG1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Bases de données relationnelles, requêtes SQL et programmation objet
11 réponses • sujet p. 2–5

Exercice 2 — Protocoles de routage et graphes
11 réponses • sujet p. 6–10

Exercice 3 — Tableaux, dictionnaires, récursivité et programmation dynamique appliqués à un stock de planches
16 réponses • sujet p. 11–15

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

Bases de données relationnelles, requêtes SQL et programmation objet

A

Question 1.

Sujet p. 2, 3

> Donner, en justifiant, les clés étrangères de la relation reservation.

Une clé étrangère est un attribut qui référence la clé primaire d'une autre relation. Dans `reservation`, `id_client` désigne le client associé à la réservation et `id_salle` la salle réservée. Ils référencent donc les clés primaires des relations `client` et `salle`. L'attribut `id` est la clé primaire de `reservation`, pas une clé étrangère.

RÉPONSE À RETENIR

Les clés étrangères de `reservation` sont `id_client`, qui référence `client(id_client)`, et `id_salle`, qui référence `salle(id_salle)`.

Question 2.

Sujet p. 2, 3

> Donner le résultat de la requête `SELECT intitule FROM salle WHERE nb_places > 5 ORDER BY tarif_jour;` lorsqu'on l'applique aux extraits donnés des tables.

La condition `nb_places > 5` conserve les salles RB, Ravel et RA. Leurs tarifs journaliers respectifs sont 300, 2300 et 350. Le tri croissant par `tarif_jour` donne donc l'ordre RB, RA, Ravel.

RÉPONSE À RETENIR

Le résultat de la requête est :

intitule
RB
RA
Ravel

Question 3.

Sujet p. 2, 3

> Écrire une requête permettant d'obtenir la liste sans doublon des `id_salle` des salles réservées depuis janvier 2024. On pourra utiliser le comparateur `>` pour les dates.

Les dates étant au format année-mois-jour, on retient les réservations à partir du 1er janvier 2024 inclus. `DISTINCT` élimine les répétitions d'un même identifiant de salle. L'ordre est précisé par `ORDER BY` pour rendre la présentation du résultat déterminée.

```
SELECT DISTINCT id_salle
FROM reservation
WHERE date >= '2024-01-01'
ORDER BY id_salle;
```

Dans les extraits, les réservations à partir de cette date concernent les salles 3, 3 et 4 ; après suppression des doublons, il reste les identifiants 3 et 4.

RÉPONSE À RETENIR

Sur les extraits du sujet :

id_salle
3
4

Question 4.

Sujet p. 2, 3

> Un nouveau client de nom de société « EN » et de numéro de téléphone « 0144624055 » a réservé le bureau d'identifiant 4 pour une journée, le 12 juin 2024. Écrire une suite de requêtes permettant d'ajouter ce client et sa réservation. On attribuera la valeur 3 à `id_client` et la valeur 5 à `id`.

On insère d'abord le client, puis sa réservation : cette dernière référence l'identifiant du client nouvellement ajouté. La date est écrite au format année-mois-jour et la durée est d'une journée. Le numéro de téléphone est placé entre apostrophes, notamment pour conserver son zéro initial.

```
INSERT INTO client (id_client, societe, num_tel)
VALUES (3, 'EN', '0144624055');

INSERT INTO reservation (id, id_client, id_salle, date, duree)
VALUES (5, 3, 4, '2024-06-12', 1);
```

RÉPONSE À RETENIR

La première requête ajoute le client d'identifiant 3 ; la seconde ajoute sa réservation d'identifiant 5 pour la salle 4, le 12 juin 2024, pendant une journée.

Question 5.

Sujet p. 2, 3

> La salle « RAVEL » a été modifiée, elle peut maintenant contenir 95 personnes. Écrire la requête permettant d'enregistrer cette modification dans la base de données précédente.

Dans l'extrait fourni, la salle intitulée `Ravel` a pour identifiant 2. On peut donc la désigner par sa clé primaire et mettre à jour sa capacité.

```
UPDATE salle
SET nb_places = 95
WHERE id_salle = 2;
```

RÉPONSE À RETENIR

La capacité de la salle d'identifiant 2 est enregistrée à 95 places.

Question 6.

Sujet p. 2, 3

> Écrire une requête permettant d'obtenir le temps total de réservation de la salle d'identifiant 6.

La fonction d'agrégation `SUM` additionne les durées des réservations retenues par la condition sur `id_salle`. Dans les extraits fournis, aucune réservation ne concerne la salle 6 ; en SQL, `SUM` renvoie alors `NULL` s'il n'y a aucune ligne à agréger.

```
SELECT SUM(duree)
FROM reservation
WHERE id_salle = 6;
```

RÉPONSE À RETENIR

La requête renvoie le temps total de réservation de la salle 6. Sur les extraits du sujet, le résultat est `NULL`, car aucune réservation de cette salle n'y figure.

Question 7.

Sujet p. 2, 3

> Écrire une requête permettant d'obtenir la liste sans doublon des noms de société ayant réservé une salle de réunion.

On joint `reservation` à `salle` pour filtrer les salles de type `Réunion`, puis à `client` pour obtenir les noms de société associés aux réservations. `DISTINCT` élimine les répétitions lorsqu'un client a réservé plusieurs fois une salle de réunion.

```
SELECT DISTINCT client.societe
FROM reservation
JOIN salle ON reservation.id_salle = salle.id_salle
JOIN client ON reservation.id_client = client.id_client
WHERE salle.type = 'Réunion';
```

Dans les extraits, les réservations de la salle de réunion d'identifiant 3 sont celles du client 2, dont la société est FleursJ.

RÉPONSE À RETENIR

Résultat sur les extraits du sujet :

societe
FleursJ

B

Question 8.

Sujet p. 4

> Donner un exemple d'attribut et un exemple de méthode de la classe `Salle`.

Un attribut est une donnée portée par les objets de la classe ; par exemple, `intitule`. Une méthode est une fonction définie dans la classe ; par exemple, `reserver`.

> **RÉPONSE À RETENIR**
>
> Exemple d'attribut : `intitule`. Exemple de méthode : `reserver`.

Question 9.

Sujet p. 4

> Écrire la méthode `liberer` qui modifie les attributs d'une salle de manière à la rendre disponible et sans occupant (chaîne de caractères vide).

Pour libérer la salle, la méthode rétablit l'état de disponibilité et efface le nom de l'occupant en affectant la chaîne vide à l'attribut correspondant.

```
def liberer(self):
    self.dispo = True
    self.occupant = ""
```

> **RÉPONSE À RETENIR**
>
> Après l'appel de `liberer`, `dispo` vaut `True` et `occupant` vaut `""`.

Question 10.

Sujet p. 4

> Recopier et compléter les lignes 3, 4 et 5 du code de la fonction `salles_dispos`, qui prend en paramètre une liste de salles et renvoie la liste des intitulés de celles qui sont disponibles.

On parcourt les objets de la liste, puis on teste l'attribut `dispo` de chacun. Si la salle est disponible, on ajoute son intitulé à la liste résultat.

```
def salles_dispos(liste_salles):
    res = []
    for salle in liste_salles:
        if salle.dispo:
            res.append(salle.intitule)
    return res
```

Dans l'exemple, `S1` et `S2` ont été réservées. `S3`, `S4` et `S5` sont disponibles ; la fonction renvoie donc `['RA', 'B2', 'RB']`.

> **RÉPONSE À RETENIR**
>
> Pour la liste de salles donnée, le résultat est `['RA', 'B2', 'RB']`.

Question 11.

Sujet p. 5

> Recopier et compléter la méthode `ajouter_reservation` qui ajoute `salle` à la liste des salles réservées, puis réserve cette salle en modifiant le prix total. Elle renvoie `True` si la salle a pu être réservée car elle était libre, et `False` sinon.

La condition vérifie que la salle est libre. On ajoute alors cette salle à la liste du séminaire et on appelle `reserver` avec le nom du client. Cette méthode rend la salle indisponible et renvoie son tarif, qui est ajouté au prix total. Si la salle est déjà occupée, on ne modifie ni la liste ni le prix total et la méthode renvoie `False`.

```
def ajouter_reservation(self, salle):
    if salle.dispo:
        self.liste_salles.append(salle)
        self.prix_total += salle.reserver(self.client)
        return True
    return False
```

RÉPONSE À RETENIR

La méthode renvoie `True` lorsque la salle était libre : elle est ajoutée à la liste et son tarif est comptabilisé. Si elle n'est pas disponible, la méthode renvoie `False`.

Exercice 2

6 points

Protocoles de routage et graphes

A

Question 1.

Sujet p. 6

> Expliquer pourquoi le routeur R2 ne peut pas être associé à l'adresse IP 192.168.2.3 dans le sous-réseau qui le lie au routeur R1.

Un masque ***/24*** signifie que les trois premiers octets identifient le sous-réseau. Les interfaces reliées au même sous-réseau doivent donc avoir les mêmes trois premiers octets.

La liaison entre R1 et R2 utilise le sous-réseau `192.168.1.0/24`. L'adresse `192.168.2.3` appartient au sous-réseau distinct `192.168.2.0/24` ; elle ne peut donc pas être l'adresse de R2 sur la liaison R1–R2.

> **RÉPONSE À RETENIR**
>
> R2 ne peut pas avoir l'adresse `192.168.2.3` sur la liaison R1–R2, car cette adresse appartient au sous-réseau `192.168.2.0/24`, tandis que la liaison R1–R2 utilise `192.168.1.0/24`.

Question 2.

Sujet p. 6, 7

> Recopier et compléter la table de routage du routeur R4 avec les adresses IP des destinations et des passerelles (ou routeurs suivants), selon le protocole RIP suite à l'échange avec l'ensemble des routeurs. Une destination correspond à un sous-réseau. La table fournie indique déjà les destinations 10.1.6.0/24, 10.2.6.0/24 et 192.168.2.0/24, chacune connectée avec une distance de 0, ainsi que la destination 10.1.1.0/24 via 10.1.6.1 avec une distance de 1.

R4 est directement relié aux sous-réseaux `10.1.6.0/24`, `10.2.6.0/24` et `192.168.2.0/24` : leur distance est 0. Pour un sous-réseau non directement connecté, la passerelle est l'adresse de l'interface du routeur suivant, sur le lien qui mène à ce routeur. La table fournie indique que `10.1.1.0/24` est atteint via R3 (`10.1.6.1`) à distance 1. Le sous-réseau `192.168.1.0/24` est atteint en passant ensuite par R2, donc à distance 2. Les sous-réseaux `192.168.4.0/24` et `10.2.7.0/24` sont chacun à un routeur de R4, respectivement via R6 et R5.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

La table de routage complétée de R4 est :

Destination	Passerelle	Distance
10.1.6.0/24	connecté	0
10.2.6.0/24	connecté	0
192.168.2.0/24	connecté	0
10.1.1.0/24	10.1.6.1	1
192.168.1.0/24	10.1.6.1	2
192.168.4.0/24	192.168.2.1	1
10.2.7.0/24	10.2.6.2	1

Question 3.

Sujet p. 6, 7

> Les routeurs R3 et R4 sont reliés par la technologie 5G dont le débit est de 10 Gbit/s. Calculer le débit entre les routeurs R1 et R2. On rappelle que le coût d'un lien est inversement proportionnel au débit de la liaison.

Le coût étant inversement proportionnel au débit, le produit coût × débit est constant. Pour la liaison R3–R4, le coût lu sur le graphe est 1 et le débit donné est 10 Gbit/s. La constante vaut donc 1 × 10 = 10. Le coût de la liaison R1–R2 étant 20, son débit vaut 10 ÷ 20 = 0,5 Gbit/s.

RÉPONSE À RETENIR

Le débit entre R1 et R2 est de **0,5 Gbit/s**, soit **500 Mbit/s**.

Question 4.

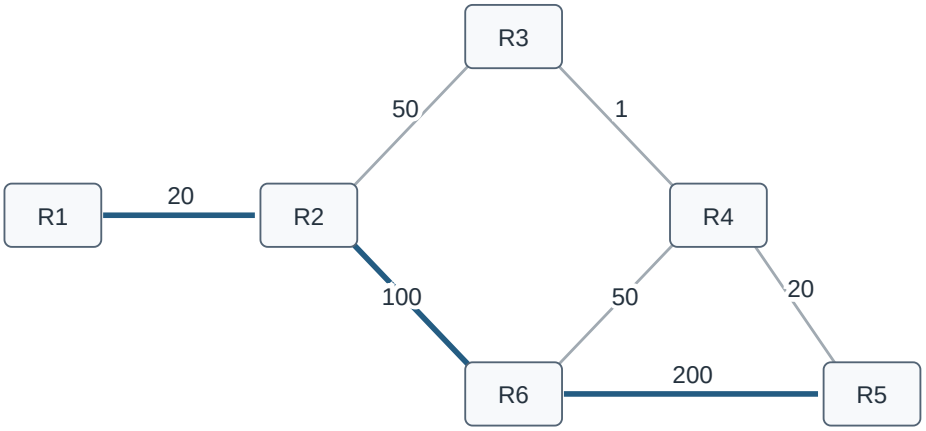
Sujet p. 6, 7

> Indiquer le chemin à suivre pour transmettre des données de R1 à R5 selon le protocole RIP.

RIP choisit le chemin qui minimise le nombre de routeurs traversés, ce qui revient ici à minimiser le nombre de liaisons. Le chemin R1–R2–R6–R5 comporte trois liaisons. Le chemin passant par R3 et R4, R1–R2–R3–R4–R5, en comporte quatre. Le premier est donc le chemin retenu par RIP.

FormaV - document d'entraînement non officiel

Chemin RIP de R1 à R5



Le chemin retenu est mis en évidence. Les étiquettes indiquent les coûts des liaisons.

> **RÉPONSE À RETENIR**
>
> Le chemin RIP est **R1 → R2 → R6 → R5**.

Question 5.

Sujet p. 6, 7

> La liaison R2–R6 tombe en panne. Indiquer le chemin à suivre pour transmettre des données de R1 à R5, toujours selon le protocole RIP.

La panne de la liaison R2–R6 rend impossible le chemin retenu à la question 4. Depuis R2, le trajet le plus court en nombre de liaisons vers R5 passe par R3, puis R4.

> **RÉPONSE À RETENIR**
>
> Le chemin RIP est **R1 → R2 → R3 → R4 → R5**.

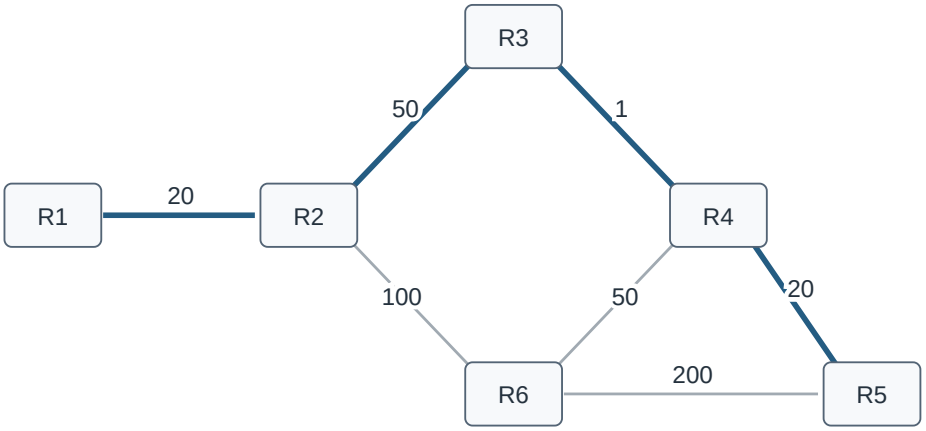
Question 6.

Sujet p. 6, 7

> La liaison R2–R6 est rétablie. Le protocole de routage OSPF est alors utilisé. Dans ce protocole, la métrique de la table de routage correspond à la somme des coûts des liaisons traversées. Indiquer le chemin à suivre pour transmettre des données de R1 à R5 selon le protocole OSPF et comparer avec le chemin obtenu précédemment selon le protocole RIP.

Avec OSPF, on compare les sommes des coûts. Le trajet R1–R2–R6–R5 coûte 20 + 100 + 200 = 320. Le trajet R1–R2–R3–R4–R5 coûte 20 + 50 + 1 + 20 = 91. Le trajet R1–R2–R6–R4–R5 coûterait 20 + 100 + 50 + 20 = 190. Le coût 91 est le plus faible : OSPF retient donc le trajet passant par R3 et R4. Il coïncide avec le chemin RIP de la question 5, lorsque R2–R6 était en panne, et diffère du chemin RIP de la question 4.

Chemin OSPF de R1 à R5



Le chemin OSPF de coût total 91 est mis en évidence. Les étiquettes indiquent les coûts des liaisons.

> **RÉPONSE À RETENIR**
>
> OSPF choisit **R1 → R2 → R3 → R4 → R5**, de coût total 91. Ce chemin est identique au chemin RIP de la question 5, mais différent du chemin RIP de la question 4.

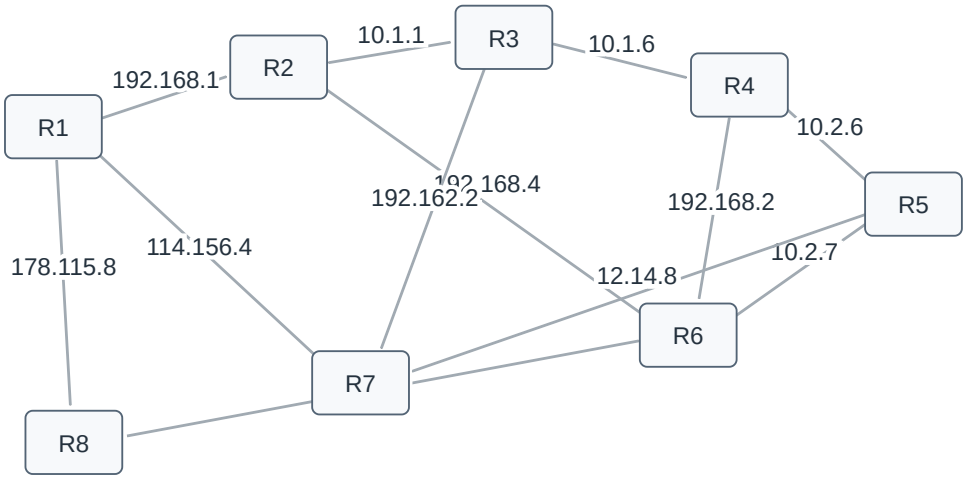
Question 7.

Sujet p. 8

> Le réseau évolue et de nouveaux routeurs sont implantés. Pour des raisons de lisibilité, le suffixe /24 a été omis. À partir du tableau récapitulant les mises à jour des adresses IP des routeurs, représenter le réseau informatique mis à jour à l'aide d'un graphe.

Deux routeurs sont reliés lorsqu'ils possèdent chacun une adresse dans le même sous-réseau. Le suffixe `/24` étant omis dans le tableau, les trois premiers octets des adresses permettent ici de repérer chaque réseau partagé. Par exemple, R1 et R7 partagent le réseau `114.156.4.0/24`, avec les adresses `114.156.4.3` et `114.156.4.2`. On déduit ainsi les douze liaisons ci-dessous. Le tableau ne donne pas de coûts pour le réseau mis à jour : le graphe n'est donc pas pondéré.

Réseau informatique mis à jour



Graphe non pondéré ; les étiquettes indiquent les identifiants de sous-réseaux, suffixe /24 omis.

RÉPONSE À RETENIR

Les liaisons déduites du tableau sont :

- R1–R2 : `192.168.1.0/24` ; R1–R7 : `114.156.4.0/24` ; R1–R8 : `178.115.8.0/24`.
- R2–R3 : `10.1.1.0/24` ; R2–R6 : `192.168.4.0/24` ; R3–R4 : `10.1.6.0/24`.
- R3–R7 : `192.162.2.0/24` ; R4–R5 : `10.2.6.0/24` ; R4–R6 : `192.168.2.0/24`.
- R5–R6 : `10.2.7.0/24` ; R5–R7 : `12.14.8.0/24` ; R6–R8 : `10.7.2.0/24`.

B

Question 8.

Sujet p. 7, 8

Pour les deux itinéraires proposés par le GPS de V1 à V5 — l'itinéraire A minimisant le temps de parcours et l'itinéraire B minimisant le nombre de villes traversées — indiquer celui qui correspond au protocole RIP et celui qui correspond au protocole OSPF d'un réseau informatique.

RIP minimise le nombre de routeurs à traverser ; par analogie, il correspond donc à l'itinéraire qui minimise le nombre de villes traversées, soit B. OSPF minimise une somme de coûts de liaison ; dans l'analogie proposée, ces coûts sont les temps de parcours. Il correspond donc à l'itinéraire A.

RÉPONSE À RETENIR

L'itinéraire B correspond à RIP et **l'itinéraire A** correspond à OSPF.

Question 9.

Sujet p. 8, 9

Écrire la liste des sommets, dans l'ordre de leur visite, lors du parcours en largeur du graphe du réseau routier de la

FormaV - document d'entraînement non officiel

> figure 2 depuis le sommet 'V3'.

On part de V3. Ses voisins sont examinés dans l'ordre indiqué par la liste d'adjacence : V2, puis V4. On traite ensuite V2, qui permet d'ajouter V1 et V6, puis V4, qui permet d'ajouter V5 (V6 a déjà été découvert). Les sommets sont ainsi visités en largeur.

> **RÉPONSE À RETENIR**
>
> L'ordre de visite est **V3, V2, V4, V1, V6, V5**.

Question 10.

Sujet p. 9, 10

> L'instruction `itineraire_court(route, 'V1', 'V5')` est saisie puis exécutée. Compléter l'évolution du contenu de la variable `precedent` à chaque tour de la boucle `while`, à la suite des états indiqués : tour n°1 `{'V1': None}`, tour n°2 `{'V1': None, 'V2': 'V1'}`, tour n°3 `{'V1': None, 'V2': 'V1', 'V3': 'V2'}`, puis `...`.

La liste d'adjacence de V2 est `['V1', 'V3', 'V6']`. Lors du traitement de V2, V1 est déjà visité, mais V3 et V6 sont ajoutés à la file et au dictionnaire `precedent`. L'état fourni au tour n°3 omet donc V6 et n'est pas compatible avec le code. En suivant le code, les états successifs sont les suivants. Le tour n°1 correspond à l'état initial fourni ; les états suivants sont pris après chaque itération de la boucle.

```
Tour n°1 : {'V1': None}
Tour n°2 : {'V1': None, 'V2': 'V1'}
Tour n°3 : {'V1': None, 'V2': 'V1', 'V3': 'V2', 'V6': 'V2'}
Tour n°4 : {'V1': None, 'V2': 'V1', 'V3': 'V2', 'V6': 'V2', 'V4': 'V3'}
Tour n°5 : {'V1': None, 'V2': 'V1', 'V3': 'V2', 'V6': 'V2', 'V4': 'V6', 'V5': 'V6'}
```

Au tour n°5, le sommet V6 est traité avant l'entrée en file de V4. Comme V4 n'est pas encore dans `deja_visites`, le code l'ajoute de nouveau à la file et remplace son prédécesseur par V6. V5 est alors rencontré ; la file est vidée et la boucle se termine.

> **RÉPONSE À RETENIR**
>
> Selon le code, le tour n°3 doit déjà contenir `'V6': 'V2'`, puis le prédécesseur de V4 est remplacé par V6 au tour n°5. Les états fournis dans le sujet sont donc partiellement incompatibles avec le programme.

Question 11.

Sujet p. 9, 10

> Écrire ce que renvoie la fonction `itineraire_court` à la suite de l'exécution de l'instruction saisie `itineraire_court(route, 'V1', 'V5')`.

Lorsque V5 est rencontré, son prédécesseur est V6. En remontant le dictionnaire `precedent`, on obtient V5, puis V6, puis V2, puis V1, dont le prédécesseur est `None`. La méthode `reverse()` inverse ensuite cette liste pour la remettre dans l'ordre du départ à l'arrivée.

> **RÉPONSE À RETENIR**
>
> La fonction renvoie `['V1', 'V2', 'V6', 'V5']`.

Exercice 3

8 points

Tableaux, dictionnaires, récursivité et programmation dynamique appliqués à un stock de planches

A

Question 1.

Sujet p. 11

> Donner une instruction Python qui permet de déclarer et d'initialiser le dictionnaire `effectif` correspondant à un stock constitué de trois planches de 1 m, de quatre planches de 2 m et d'une planche de 4 m.

Le dictionnaire associe chaque longueur présente à son effectif. On crée donc une association pour les longueurs 1, 2 et 4, avec les nombres de planches indiqués.

RÉPONSE À RETENIR

```
effectif = {1: 3, 2: 4, 4: 1}
```

Question 2.

Sujet p. 11

> Donner une expression en Python qui calcule la valeur totale en euro de l'ensemble des planches de 10 m d'un stock représenté par le dictionnaire `effectif` contenant au moins une planche de 10 m. On rappelle que la variable `prix` contient l'ensemble des prix de vente en euro des planches.

Il y a `effectif[10]` planches de 10 m, chacune vendue `prix[10]` euros. On multiplie donc ces deux valeurs. La clé 10 est présente, comme le précise l'énoncé.

RÉPONSE À RETENIR

```
effectif[10] * prix[10]
```

Question 3.

Sujet p. 11

> Donner une expression en Python qui vaut `True` si le stock représenté par le dictionnaire `effectif` contient des planches de 5 m et `False` sinon.

Une longueur est présente dans le stock exactement lorsque le dictionnaire possède la clé correspondante. Le test d'appartenance à un dictionnaire porte sur ses clés.

RÉPONSE À RETENIR

```
5 in effectif
```

Question 4.

Sujet p. 11, 12

> Recopier et compléter les lignes 2, 3 et 4 du code de la fonction `valeur_stock` ci-après.

On initialise le total à zéro, puis on parcourt les longueurs présentes dans le dictionnaire. Pour chaque longueur, on ajoute le nombre de planches de cette longueur multiplié par le prix d'une planche de cette longueur.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def valeur_stock(effectif):
    valeur_totale = 0
    for longueur in effectif:
        valeur_totale = valeur_totale + effectif[longueur] * prix[longueur]
    return valeur_totale
```

Question 5.

Sujet p. 11, 12

Écrire une fonction `mise_a_jour` qui prend en paramètre l'effectif du stock déjà présent dans la scierie nommé `effectif` et l'effectif du stock contenu dans le camion nommé `autre_effectif`, et qui met à jour le dictionnaire `effectif`. Exemple attendu : avec `effectif = {1: 3, 5: 5}` et `autre_effectif = {1: 2, 2: 2}`, après l'appel `mise_a_jour(effectif, autre_effectif)`, l'expression `effectif == {1: 5, 2: 2, 5: 5}` vaut `True`.

On parcourt les longueurs livrées. Si une longueur est déjà une clé d'`effectif`, on ajoute le nouvel effectif à celui qui existe. Sinon, on crée cette association avec l'effectif livré. La fonction modifie le dictionnaire reçu.

RÉPONSE À RETENIR

```
def mise_a_jour(effectif, autre_effectif):
    for longueur in autre_effectif:
        if longueur in effectif:
            effectif[longueur] = effectif[longueur] + autre_effectif[longueur]
        else:
            effectif[longueur] = autre_effectif[longueur]
```

Avec l'exemple de l'énoncé, la clé 1 passe de 3 à 5, la clé 2 est créée avec la valeur 2 et la clé 5 reste inchangée : `effectif` devient bien `{1: 5, 2: 2, 5: 5}`.

Question 6.

Sujet p. 12

Écrire une fonction `vendre` qui prend en paramètre l'effectif du stock nommé `effectif` et une longueur de planche nommée `longueur`, et qui modifie le dictionnaire `effectif` en conséquence. On a déjà vérifié qu'il y avait bien une planche de cette longueur dans le stock. Si le stock de planches de cette longueur devient nul, l'association devra être supprimée du dictionnaire `effectif`. On pourra utiliser `del d[c]`, qui supprime la clé `c` du dictionnaire `d`. Les exemples donnés sont `effectif = {1: 3, 5: 1}` : après la vente d'une planche de longueur 1, `effectif == {1: 2, 5: 1}` vaut `True`; après la vente d'une planche de longueur 5, `effectif == {1: 2}` vaut `True`.

L'existence de la clé `longueur` et d'au moins une planche de cette longueur a déjà été vérifiée. On diminue donc son effectif de 1, puis on supprime la clé si l'effectif devient nul.

RÉPONSE À RETENIR

```
def vendre(effectif, longueur):
    effectif[longueur] = effectif[longueur] - 1
    if effectif[longueur] == 0:
        del effectif[longueur]
```

Ainsi, une association est conservée tant qu'il reste au moins une planche de cette longueur.

B

Question 7.

Sujet p. 11, 13

> Déterminer, en justifiant, la valeur maximale que peut rapporter une planche de 4 m.

On compare toutes les possibilités de découpe en longueurs entières positives, ainsi que la vente de la planche entière :

- 4 m entière : 9 € ;
- 3 m + 1 m : 8 + 1 = 9 € ;
- 2 m + 2 m : 5 + 5 = 10 € ;
- 2 m + 1 m + 1 m : 5 + 1 + 1 = 7 € ;
- 1 m + 1 m + 1 m + 1 m : 4 €.

La meilleure recette est donc celle obtenue en deux planches de 2 m.

RÉPONSE À RETENIR

La valeur maximale d'une planche de 4 m est **10 €**, obtenus en la découpant en deux planches de 2 m.

Question 8.

Sujet p. 13

> La formule proposée indique que si n ≤ 3, `valmax(n) = prix[n]`, et que sinon `valmax(n) = max(prix[n], prix[n-1] + valmax(1), prix[n-2] + valmax(2), ..., prix[1] + valmax(n-1))`. Expliquer le cas d'arrêt de cette formule récursive.

Lorsque la longueur est au plus 3 m, l'énoncé donne directement la valeur de vente entière, égale à `prix[n]`. Il n'y a alors pas à calculer la valeur de morceaux plus petits : cette règle fournit le cas d'arrêt. Pour les longueurs supérieures, chaque appel récursif porte sur une longueur strictement inférieure à `n`.

RÉPONSE À RETENIR

Le cas d'arrêt est `n ≤ 3` : on renvoie directement `prix[n]`, sans nouvel appel récursif.

Question 9.

Sujet p. 11, 13

> Recopier et compléter les quatre lignes ci-dessous : `valmax(1) = prix[...] = ...` ; `valmax(2) = ... = ...` ; `valmax(3) = ... = ...` ; `valmax(4) = max(prix[...], prix[...] + ..., prix[...] + ..., prix[...] + ...) = ...`.

Pour les longueurs 1, 2 et 3, la valeur maximale est le prix de vente entier indiqué, puisque ces longueurs relèvent du cas d'arrêt. Pour 4 m, on compare le prix de vente entier aux trois découpages possibles en deux morceaux de longueurs positives.

RÉPONSE À RETENIR

- `valmax(1) = prix[1] = 1` ;
- `valmax(2) = prix[2] = 5` ;
- `valmax(3) = prix[3] = 8` ;
- `valmax(4) = max(prix[4], prix[3] + valmax(1), prix[2] + valmax(2), prix[1] + valmax(3)) = max(9, 8 + 1, 5 + 5, 1 + 8) = 10`.

Question 10.

Sujet p. 13

> Expliquer pourquoi, si n ≥ 4, on a bien : `valmax(n) = max(prix[n], prix[n-1] + valmax(1), prix[n-2] + valmax(2), ..., prix[1] + valmax(n-1))`.

Une planche de longueur `n` peut être vendue entière, ce qui rapporte `prix[n]`. Si elle est découpée, on peut choisir un morceau de longueur `n-k`, pour chaque entier `k` compris entre 1 et `n-1`. Ce morceau rapporte `prix[n-k]`. La longueur restante, `k`, rapporte au maximum `valmax(k)` si on optimise sa découpe. Chaque découpage est ainsi évalué par la somme des recettes des deux morceaux. Le maximum de ces sommes et de la vente entière donne la meilleure recette possible.

RÉPONSE À RETENIR

Pour tout `n ≥ 4`, on obtient bien `valmax(n) = max(prix[n], prix[n-1] + valmax(1), prix[n-2] + valmax(2), ..., prix[1] + valmax(n-1))`.

Question 11.

Sujet p. 11, 13, 14

> Recopier et compléter les lignes 2 à 7 du code de la fonction `valmax` ci-après. La fonction prend en paramètre un entier n et renvoie la valeur maximale obtenue pour une planche de longueur n selon les prix de la variable globale `prix`.

Pour `n ≤ 3`, on applique le cas d'arrêt. Sinon, on commence par envisager la vente entière, puis on compare cette recette à celle de chaque découpe : un morceau a la longueur `n - longueur` et la partie restante, de longueur `longueur`, rapporte au mieux `valmax(longueur)`. La boucle parcourt les valeurs de 1 à `n - 1`.

RÉPONSE À RETENIR

```
def valmax(n):
    if n <= 3:
        return prix[n]
    else:
        maximum = prix[n]
        for longueur in range(1, n):
            maximum = max(maximum, prix[n - longueur] + valmax(longueur))
        return maximum
```

Question 12.

Sujet p. 13, 14

> Expliquer pourquoi l'appel de `valmax(7)` effectuera deux appels de `valmax(5)`.

Dans l'évaluation de `valmax(7)`, la boucle appelle notamment `valmax(5)` pour `longueur = 5`. Elle appelle aussi `valmax(6)` pour `longueur = 6` ; pendant le calcul de ce dernier, la boucle appelle `valmax(5)` pour `longueur = 5`. Comme la fonction ne mémorise pas ses résultats, ces deux calculs constituent deux appels distincts.

RÉPONSE À RETENIR

L'appel `valmax(7)` effectue donc deux appels à `valmax(5)` : un directement et un pendant le calcul de `valmax(6)`.

FormaV - document d'entraînement non officiel

C

Question 13.

Sujet p. 11, 13, 14, 15

> La variable globale `memo` est déclarée et initialisée à `[0, 1, 5, 8, -1, -1, -1, -1, -1, -1, -1]`. Recopier et compléter les lignes 2 à 8 du code de la fonction `valmax_dynamique` ci-après, de façon à renvoyer la valeur maximale déjà mémorisée dans `memo[n]` si `memo[n]` ne vaut pas −1, et, sinon, à calculer la valeur maximale avec des appels récursifs puis à l'affecter à `memo[n]`.

Si `memo[n]` contient déjà une valeur différente de `-1`, on la renvoie immédiatement. Sinon, on reprend le calcul de la question 11 en remplaçant les appels à `valmax` par des appels à `valmax_dynamique`, puis on mémorise le maximum obtenu.

RÉPONSE À RETENIR

```
def valmax_dynamique(n):
    if memo[n] != -1:
        return memo[n]
    else:
        maximum = prix[n]
        for longueur in range(1, n):
            maximum = max(maximum, prix[n - longueur] + valmax_dynamique(longueur))
        memo[n] = maximum
    return maximum
```

Question 14.

Sujet p. 14, 15

> Donner en justifiant le nombre d'appels récursifs effectués par l'appel de `valmax_dynamique(6)`, après l'exécution de `valmax_dynamique(5)`.

Après l'exécution de `valmax_dynamique(5)`, les valeurs pour les longueurs 1, 2 et 3 sont présentes dans `memo`, et celles pour 4 et 5 ont été calculées et mémorisées. Pour calculer `valmax_dynamique(6)`, la boucle appelle la fonction une fois pour chacune des longueurs de 1 à 5. Ces appels trouvent tous leur réponse en mémoire et ne déclenchent aucun appel supplémentaire.

RÉPONSE À RETENIR

L'appel de `valmax_dynamique(6)` effectue **5 appels récursifs** directs, pour les longueurs 1, 2, 3, 4 et 5. Ils consultent tous des valeurs déjà mémorisées.

Question 15.

Sujet p. 14, 15

> Expliquer ce qui se serait passé si on avait initialisé la variable `memo` avec uniquement des valeurs égales à −1.

Si toutes les cases de `memo` valaient `-1`, aucun résultat ne serait préenregistré. La fonction calculerait alors aussi les valeurs des longueurs 1, 2 et 3 au lieu de les renvoyer immédiatement depuis le tableau. Le calcul terminerait néanmoins : pour une longueur de 1, la boucle est vide, puis le résultat est mémorisé. Les valeurs calculées seraient ensuite réutilisées grâce à la mémoïsation.

RÉPONSE À RETENIR

Le calcul terminerait et donnerait les mêmes valeurs, mais nécessiterait des calculs supplémentaires au départ, car même les cas de base devraient être calculés puis mémorisés.

Question 16.

Sujet p. 11, 15

> On rappelle que la scierie utilise un dictionnaire `effectif` qui associe à chaque longueur de planche en mètre présente dans le stock son effectif dans le stock. On suppose que la variable `memo` vient d'être initialisée à `[0, 1, 5, 8, -1, -1, -1, -1, -1, -1, -1]`. Écrire une fonction `valeur_max_stock` qui prend en paramètre un dictionnaire `effectif` et qui renvoie la valeur maximale en euro que peut rapporter le stock à l'entreprise en optimisant le découpage de chaque planche afin qu'elle rapporte le maximum.

Chaque planche de longueur `longueur` peut rapporter au maximum `valmax_dynamique(longueur)` euros. On multiplie cette valeur par l'effectif correspondant et on additionne les contributions de toutes les longueurs présentes. La fonction utilise la variable globale `memo`, initialisée comme indiqué dans l'énoncé.

RÉPONSE À RETENIR

```
def valeur_max_stock(effectif):
    valeur_totale = 0
    for longueur in effectif:
        valeur_totale = valeur_totale + effectif[longueur] *
valmax_dynamique(longueur)
    return valeur_totale
```

Cette somme optimise séparément chaque planche, comme le demande l'énoncé.

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