Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

CORRIGÉ NON OFFICIEL

Proposition de corrigé rédigée par FormaV avec l'aide d'outils d'intelligence artificielle. Elle peut contenir des erreurs et ne remplace ni le corrigé ni le barème officiels de l'examen.

Les sujets officiels des sessions passées et nos sujets d'entraînement sont classés sur https://bac-maths-nsi.fr/annales-sujets-et-corriges

Document mis en ligne par FormaV®. Reproduction ou diffusion sans autorisation interdite.

BACCALAURÉAT GÉNÉRAL • CORRIGÉ DÉTAILLÉ

Numérique et sciences informatiques

Session 2025 • Jour 1

Code sujet **25-NSIJ1NC1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Graphes, protocoles réseaux et programmation orientée objet
13 réponses • sujet p. 2–6

Exercice 2 — Bases de données et programmation orientée objet
10 réponses • sujet p. 7–10

Exercice 3 — Algorithmique, représentation binaire des entiers positifs et programmation en langage Python
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

Graphes, protocoles réseaux et programmation orientée objet

Partie A

Question 1.

Sujet p. 2

> On désire parcourir en largeur depuis le sommet A le graphe de la figure 1. Dire lequel des parcours ABCDEF, ABCEDF et ABCDFE est un parcours en largeur, en justifiant.

En largeur, on visite d'abord le sommet de départ, puis tous ses voisins directs avant les sommets situés à deux arêtes, et ainsi de suite. Depuis A, les niveaux sont : niveau 0 : A ; niveau 1 : B, C et E ; niveau 2 : D ; niveau 3 : F. L'ordre des sommets d'un même niveau peut dépendre de l'ordre de parcours des voisins.

ABCDEF et ABCDFE placent D, à distance 2 de A, avant E, à distance 1 : ces parcours ne sont pas en largeur.

RÉPONSE À RETENIR

Le seul parcours en largeur proposé est **ABCEDF** : il visite B, C et E avant D, puis F.

Question 2.

Sujet p. 2, 3

> Donner la table de routage de F à l'initialisation du protocole RIP.

À l'initialisation, chaque routeur inscrit uniquement ses voisins directs. Sur la figure 1, F est relié directement à D et à aucun autre routeur. D est donc accessible en un saut ; le prochain routeur est noté « – » dans la table, comme pour les liaisons directes de la table fournie pour A.

RÉPONSE À RETENIR

Table de routage de F à l'initialisation :

routeur	nombre de sauts	prochain routeur
D	1	–

Question 3.

Sujet p. 2, 3

> Donner la table de routage de A après une première itération de RIP. Deux réponses sont possibles.

Lors de la première itération, A reçoit les tables initiales de ses voisins B, C et E. C et E connaissent chacun D en deux sauts depuis eux-mêmes : A peut donc apprendre une route vers D en deux sauts, en passant par C ou par E. Ces deux routes ayant la même longueur, le prochain routeur retenu dépend de l'ordre de réception des tables. Les routes directes de A vers B, C et E restent inchangées.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Après une première itération, la table de A est :

routeur	nombre de sauts	prochain routeur
B	1	–
C	1	–
D	2	C ou E
E	1	–

Question 4.

Sujet p. 2, 3

> Donner le numéro de l'itération de RIP à partir duquel les tables des routeurs du réseau ne varient plus.

En comptant l'initialisation comme l'état de départ, une itération complète permet de propager les routes d'un saut supplémentaire. Dans le graphe, la distance maximale est de 4 sauts, par exemple de B à F par B–A–C–D–F. Il faut donc trois itérations pour que les routes les plus longues de 4 sauts soient connues ; l'itération suivante ne modifie plus les tables.

RÉPONSE À RETENIR

Les tables sont stabilisées à la fin de la **3e itération** et ne varient donc plus à partir de la **4e itération**.

Question 5.

Sujet p. 2, 3

> On suppose que les routeurs E et F sont reliés. Donner la nouvelle table de routage de A après stabilisation de RIP. Deux réponses sont possibles.

Après l'ajout de la liaison E–F, A atteint F en deux sauts par A–E–F. D reste accessible en deux sauts, soit par C, soit par E. Ces deux routes vers D étant de même longueur, le prochain routeur retenu dépend de l'ordre de réception des tables. Les voisins directs de A restent B, C et E.

RÉPONSE À RETENIR

Après stabilisation, la table de A est :

routeur	nombre de sauts	prochain routeur
B	1	–
C	1	–
D	2	C ou E
E	1	–
F	2	E

Partie B

Question 6.

Sujet p. 3, 4

> Recopier et compléter le code de la méthode relie de la classe Routeur. Cette méthode prend en argument self et un routeur autre et met à jour si nécessaire les attributs voisins, nb_sauts et prochain de self et autre pour indiquer leur connexion ; si les routeurs sont déjà connectés, elle ne fait rien.

On ajoute d'abord la liaison et les informations de voisinage dans `self`. La liaison étant réciproque, on appelle ensuite la méthode dans l'autre sens si nécessaire. Lors de cet appel réciproque, le test d'appartenance empêche une récursion infinie.

RÉPONSE À RETENIR

```
def relie(self, autre):
    if autre not in self.voisins:
        self.voisins.append(autre)
        self.nb_sauts[autre] = 1
        self.prochain[autre] = None
        if self not in autre.voisins:
            autre.relie(self)
```

Question 7.

Sujet p. 4, 5

> Écrire la méthode relie_liste de la classe Routeur, qui prend en paramètre une liste de routeurs lst et relie le routeur self à chacun des routeurs de cette liste.

Il suffit de parcourir la liste et d'appeler `relie` pour chaque routeur. Cette méthode prend en charge la réciprocité de la liaison et ne modifie rien si la liaison existe déjà.

RÉPONSE À RETENIR

```
def relie_liste(self, lst):
    for autre in lst:
        self.relie(autre)
```

Question 8.

Sujet p. 2, 5

> Écrire les instructions manquantes pour relier les routeurs de manière à obtenir le graphe de la figure 1.

Les arêtes de la figure 1 sont A–B, A–C, A–E, C–D, C–E, E–D et D–F. Les appels ci-dessous établissent exactement ces liaisons ; la méthode `relie` ajoute automatiquement chaque liaison dans les deux sens.

RÉPONSE À RETENIR

```
A.relie_liste([B, C, E])
C.relie_liste([D, E])
E.relie(D)
D.relie(F)
```

Question 9.

Sujet p. 2, 5

> Recopier et compléter le code de la méthode met_a_jour_table afin qu'elle implémente la règle c du protocole RIP.

Si `autre` connaît une route vers `r` en `autre.nb_sauts[r]` sauts, une route passant par `autre` depuis `self` coûte un saut de plus. Elle est ajoutée si elle est nouvelle, ou remplace la route connue seulement si elle est strictement plus courte. Le premier routeur de ce chemin est `autre`.

RÉPONSE À RETENIR

```
def met_a_jour_table(self, autre):
    for r in autre.nb_sauts:
        if r != self:
            if (r not in self.nb_sauts or
                self.nb_sauts[r] > autre.nb_sauts[r] + 1):
                self.nb_sauts[r] = autre.nb_sauts[r] + 1
                self.prochain[r] = autre
```

Question 10.

Sujet p. 3, 5

> Écrire la méthode itere_rip qui prend en paramètre le routeur self et met à jour sa table de routage lorsqu'il reçoit la table de routage de chacun des routeurs présents dans la liste de ses voisins.

Le routeur reçoit successivement la table de chacun de ses voisins et applique à chaque réception la règle de mise à jour. La boucle parcourt donc tous les éléments de `self.voisins`.

RÉPONSE À RETENIR

```
def itere_rip(self):
    for autre in self.voisins:
        self.met_a_jour_table(autre)
```

Question 11.

Sujet p. 3, 5

> Écrire une fonction qui prend en paramètre une liste de routeurs l_routeurs et réalise une itération du protocole RIP pour tous les routeurs de cette liste.

Une itération pour le réseau consiste à faire réaliser son itération locale à chacun des routeurs de la liste. Chaque routeur est traité une fois.

RÉPONSE À RETENIR

```
def itere_reseau(l_routeurs):
    for routeur in l_routeurs:
        routeur.itere_rip()
```

Question 12.

Sujet p. 5

> On suppose que la méthode met_a_jour_table renvoie True si le routeur self a procédé à une mise à jour de sa table de routage, et False sinon. Écrire une version modifiée de la méthode itere_rip de la classe Routeur qui

> renvoie True si self a modifié sa table de routage au cours de son exécution, et False sinon.

On mémorise si au moins une mise à jour a eu lieu. Il faut néanmoins appeler `met_a_jour_table` pour chacun des voisins : la boucle ne doit pas s'arrêter au premier appel qui renvoie `True`.

RÉPONSE À RETENIR

```
def itere_rip(self):
    modifie = False
    for autre in self.voisins:
        if self.met_a_jour_table(autre):
            modifie = True
    return modifie
```

Question 13.

Sujet p. 5, 6

> Compléter le code du programme principal afin qu'il mette à jour les tables de routage des routeurs de liste_routeurs jusqu'à ce qu'il ne soit plus nécessaire de faire des mises à jour. Ne pas réécrire les instructions permettant de connecter les routeurs entre eux.

L'indicateur est initialisé à `True` pour garantir au moins une itération. Au début de chaque tour, on le remet à `False`, puis on le repasse à `True` dès qu'un routeur a modifié sa table. La boucle s'arrête après un tour complet sans modification.

RÉPONSE À RETENIR

```
modification = True
while modification:
    modification = False
    for routeur in liste_routeurs:
        if routeur.itere_rip():
            modification = True
```

Exercice 2

6 points

Bases de données et programmation orientée objet

Question 1.

Sujet p. 8

> Indiquer ce que renvoie, lorsqu'on l'applique aux extraits précédents des relations, la requête : SELECT nom FROM lac WHERE altitude <= 2000;

La requête sélectionne les noms des lacs dont l'altitude est inférieure ou égale à 2 000 mètres. Dans l'extrait, seul le Lac Vert, à 1 266 mètres, satisfait cette condition.

RÉPONSE À RETENIR

Résultat sur les extraits fournis :

nom
Lac Vert

Question 2.

Sujet p. 7, 8

> D'après la base de données de Susie, indiquer les noms des lacs que l'on peut atteindre depuis le parking de Chamonix.

Le parking de Chamonix a l'identifiant 1. Dans la relation `rando`, les randonnées dont `depart` vaut 1 ont pour valeurs d'arrivée 1 et 2. Ces identifiants désignent respectivement le Lac Blanc et les Lacs Noirs dans la relation `lac`.

idR	depart	arrivee	nom du lac
1	1	1	Lac Blanc
3	1	2	Lacs Noirs

RÉPONSE À RETENIR

On peut atteindre le Lac Blanc et les Lacs Noirs depuis le parking de Chamonix, d'après les extraits de la base.

Question 3.

Sujet p. 7, 8

> À partir de la totalité des enregistrements, donner une requête permettant d'obtenir les coordonnées GPS des parkings situés dans la commune de Passy à une altitude strictement comprise entre 800 et 1 000 mètres.

On filtre la relation `parking` sur la commune Passy et sur une altitude strictement supérieure à 800 et strictement inférieure à 1 000. On projette ensuite l'attribut `coord_GPS`.

FormaV - document d'entraînement non officiel

```
SELECT coord_GPS
FROM parking
WHERE commune = 'Passy'
  AND altitude > 800
  AND altitude < 1000;
```

RÉPONSE À RETENIR

Sur les extraits fournis, les deux parkings de Passy sont à 600 et 1 181 mètres : aucun ne satisfait l'intervalle strict. Le résultat sur ces extraits est donc vide. La requête s'applique à l'ensemble des enregistrements, dont les autres lignes ne sont pas fournies.

coord_GPS

Question 4.

Sujet p. 7, 8

Donner une requête permettant d'obtenir les noms des lacs qu'il est possible d'atteindre depuis le parking situé à 1 300 mètres d'altitude dans la commune de Cordon. On admet qu'un tel parking existe dans la base de données.

On relie `parking` à `rando` grâce à `rando.depart = parking.idP`, puis `rando` à `lac` grâce à `rando.arrivee = lac.idL`. Le filtre identifie le parking de Cordon à 1 300 mètres. `DISTINCT` évite de répéter un nom si plusieurs randonnées conduisent au même lac.

```
SELECT DISTINCT lac.nom
FROM parking
JOIN rando ON rando.depart = parking.idP
JOIN lac ON rando.arrivee = lac.idL
WHERE parking.commune = 'Cordon'
  AND parking.altitude = 1300;
```

RÉPONSE À RETENIR

Dans les extraits fournis, aucun parking de Cordon à 1 300 mètres n'apparaît : la requête renvoie donc un résultat vide sur ces extraits. Comme la question porte sur l'ensemble de la base, les noms renvoyés par la base complète ne peuvent pas être déduits de l'extrait seul.

nom

Question 5.

Sujet p. 7, 8, 9

L'ordre des requêtes SQL est important. Le parking d'identifiant 3 à Passy est déjà dans la base, mais ni la randonnée ni le lac n'y figurent. Donner les requêtes permettant d'ajouter la randonnée qui part de ce parking et conduit au lac d'Anterne situé à 2 059 mètres d'altitude. On peut utiliser l'identifiant 42 pour le lac et l'identifiant 100 pour la randonnée.

La relation `rando` référence le lac par sa clé étrangère `arrivee`. Il faut donc d'abord ajouter le lac à `lac`, puis ajouter la randonnée. Le parking d'identifiant 3 existe déjà.

```
INSERT INTO lac (idL, nom, altitude)
VALUES (42, 'Lac d''Anterne', 2059);

INSERT INTO rando (idR, depart, arrivee)
VALUES (100, 3, 42);
```

RÉPONSE À RETENIR

Ces deux insertions ajoutent le lac d'Anterne sous l'identifiant 42, puis la randonnée 100 qui part du parking 3 et arrive au lac 42.

Question 6.

Sujet p. 7, 9

> Susie a fait une erreur de saisie en insérant le nom du lac : elle a écrit « Lc d Anterne ». Donner la requête permettant de corriger cette erreur.

On met à jour le nom du lac dont l'identifiant est 42, choisi pour ce lac à la question 5. L'identifiant permet de viser précisément l'enregistrement concerné.

```
UPDATE lac
SET nom = 'Lac d''Anterne'
WHERE idL = 42;
```

RÉPONSE À RETENIR

Le nom enregistré pour le lac d'identifiant 42 devient `Lac d'Anterne`.

Question 7.

Sujet p. 7, 9

> Le parking dont l'identifiant est 28 a été transformé en parc et n'existe plus. Donner les requêtes permettant de supprimer ce parking de la base de données.

`rando.depart` est une clé étrangère qui référence `parking.idP`. En l'absence d'une règle de suppression en cascade précisée dans le sujet, il faut d'abord supprimer les randonnées qui référencent le parking 28, puis supprimer ce parking.

```
DELETE FROM rando
WHERE depart = 28;

DELETE FROM parking
WHERE idP = 28;
```

RÉPONSE À RETENIR

Les randonnées partant du parking 28 sont supprimées avant le parking lui-même, ce qui respecte la contrainte de clé étrangère.

Question 8.

Sujet p. 9

> Recopier et compléter les lignes 3 et 5 de la fonction get_parking, qui prend en paramètre une liste de randonnées

FormaV - document d'entraînement non officiel

> et renvoie, sans doublon, la liste des identifiants des différents parkings points de départ de ces randonnées.

On parcourt les objets de la liste `randos`. Pour chaque randonnée, on ajoute son identifiant de parking uniquement s'il ne figure pas déjà dans la liste `parkings`.

```
def get_parking(randos):
    parkings = []
    for rando in randos:
        if rando.depart not in parkings:
            parkings.append(rando.depart)
    return parkings
```

Trace sur l'exemple : la première randonnée ajoute 1, la deuxième ajoute 2, et la troisième ne réajoute pas 1 car il est déjà présent.

Randonnée traitée	**Liste après traitement**
Rando(1, 1, 1)	[1]
Rando(2, 2, 1)	[1, 2]
Rando(3, 1, 2)	[1, 2]

RÉPONSE À RETENIR

La fonction renvoie une liste sans doublon des identifiants de départ. Pour l'exemple du sujet, le résultat est `[1, 2]`.

Question 9.

Sujet p. 9, 10

> Recopier et compléter la ligne 4 de la fonction get_nb_rando, qui prend en paramètres un identifiant de parking et une liste de randonnées, et renvoie le nombre de randonnées qui partent de ce parking.

Une randonnée part du parking recherché lorsque son attribut `depart` est égal à l'identifiant donné par le paramètre `parking`. La ligne à compléter est donc `if rando.depart == parking:`.

```
def get_nb_rando(parking, randos):
    nb = 0
    for rando in randos:
        if rando.depart == parking:
            nb = nb + 1
    return nb
```

Dans l'exemple du sujet, les randonnées 1 et 3 ont `depart` égal à 1. Le compteur passe donc de 0 à 1, puis à 2 ; la randonnée 2 ne le modifie pas.

RÉPONSE À RETENIR

La ligne complétée est `if rando.depart == parking:`. Pour l'appel donné dans le sujet, la fonction renvoie 2.

Question 10.

Sujet p. 9, 10

> Écrire une fonction nb_rando_par_parking qui prend en paramètre une liste de randonnées et renvoie un dictionnaire associant à chaque identifiant de parking le nombre de randonnées qui partent de ce parking.

On parcourt chaque randonnée et on compte les départs selon l'identifiant de son parking. Le dictionnaire est initialisé à 0 à la première rencontre d'un identifiant, puis son compteur est incrémenté. Seuls les parkings apparaissant comme départ d'au moins une randonnée figurent dans le dictionnaire, ce qui correspond aux données accessibles à la fonction.

```
def nb_rando_par_parking(randos):
    resultat = {}
    for rando in randos:
        parking = rando.depart
        if parking not in resultat:
            resultat[parking] = 0
        resultat[parking] = resultat[parking] + 1
    return resultat
```

Trace sur l'exemple du sujet :

Randonnée traitée	Dictionnaire après traitement
Rando(1, 1, 1)	{1: 1}
Rando(2, 2, 1)	{1: 1, 2: 1}
Rando(3, 1, 2)	{1: 2, 2: 1}

RÉPONSE À RETENIR

Pour la liste de l'exemple, la fonction renvoie `{1: 2, 2: 1}` : deux randonnées partent du parking 1 et une du parking 2. Si la liste est vide, elle renvoie le dictionnaire vide `{}`.

Exercice 3

8 points

Algorithmique, représentation binaire des entiers positifs et programmation en langage Python

Partie A : Modélisation du problème

Question 1.

Sujet p. 11

> Dans le jeu Objectif somme, on suppose que les cibles sont nécessairement des entiers entre 1 et 45. Expliquer pourquoi on fait cette hypothèse.

Une ligne ou une colonne comporte cinq cases, et chaque chiffre initial est compris entre 1 et 9. La somme maximale de cinq chiffres est donc 5 × 9 = 45. Pour une ligne, la règle impose de conserver au moins un chiffre : sa somme est donc au moins égale à 1. La borne inférieure 1 suppose que les cibles sont positives, comme l'indique l'intervalle donné dans l'énoncé. Attention : la règle « conserver au moins un chiffre » ne concerne explicitement que les lignes ; si une colonne était entièrement vidée, sa somme serait 0. La borne inférieure 1 est donc une hypothèse sur les cibles, et non une conséquence de cette seule règle pour les colonnes.

RÉPONSE À RETENIR

Les cibles positives sont comprises entre 1 et 45 : 1 est le minimum positif et 45 est la somme maximale de cinq chiffres égaux à 9.

Question 2.

Sujet p. 11

> Donner la plus petite valeur de cible que la ligne [6, 4, 5, 8, 2] peut atteindre. Donner aussi la plus grande valeur de cible qu'elle peut atteindre.

La ligne doit conserver au moins un chiffre. Sa plus petite somme possible est donc son plus petit chiffre, 2. Pour la somme maximale, on conserve les cinq chiffres : 6 + 4 + 5 + 8 + 2 = 25.

RÉPONSE À RETENIR

La plus petite cible possible est 2 et la plus grande est 25.

Question 3.

Sujet p. 12

> Écrire une fonction extraireLigne qui prend en paramètre un plateau et un indice i compris entre 0 et 4 inclus, et renvoie la ligne Li du plateau.

Chaque ligne est déjà une liste du plateau. L'indice `i` désigne directement la ligne à renvoyer.

RÉPONSE À RETENIR

```
def extraireLigne(plateau, i):
    return plateau[i]
```

Avec `plateau_ex`, l'appel `extraireLigne(plateau_ex, 0)` renvoie `[7, 9, 2, 3, 2]`.

FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 12

> Écrire une fonction extraireColonne qui prend en paramètre un plateau et un indice i compris entre 0 et 4 inclus, et renvoie la colonne Ci du plateau.

Pour former la colonne d'indice `i`, on récupère l'élément d'indice `i` dans chacune des cinq lignes, dans l'ordre.

RÉPONSE À RETENIR

```
def extraireColonne(plateau, i):
    colonne = []
    for j in range(5):
        colonne.append(plateau[j][i])
    return colonne
```

Avec `plateau_ex` et `i = 1`, on recueille 9, 6, 7, 4 et 6. La fonction renvoie donc `[9, 6, 7, 4, 6]`.

Partie B : Simplification du problème

Question 5.

Sujet p. 11, 12

> Donner la représentation en langage Python du plateau de la solution proposée dans la figure 1, en représentant les cases vides par des zéros.

On lit les chiffres conservés dans le plateau de droite de la figure 1 et on remplace chacune des cases vides par 0. Les lignes sont données dans l'ordre L0 à L4 et les colonnes dans l'ordre C0 à C4.

RÉPONSE À RETENIR

```
plateau_solution = [
    [0, 9, 2, 0, 2],
    [8, 0, 0, 0, 1],
    [7, 0, 3, 2, 0],
    [0, 4, 0, 0, 2],
    [0, 0, 0, 0, 4]
]
```

Vérification : les sommes des lignes sont 13, 9, 12, 6 et 4, et celles des colonnes sont 15, 13, 5, 2 et 9, conformément aux cibles de la figure.

Question 6.

Sujet p. 11, 12

> Pour le jeu représenté à gauche sur la figure 1, donner en Python le plateau obtenu en appliquant la règle 1 à chaque ligne.

On applique ici la règle 1 aux lignes seulement : une case est remplacée par 0 si son chiffre est strictement supérieur à la cible de sa ligne. Les trois premières lignes restent inchangées, car leurs chiffres sont tous inférieurs ou égaux à leurs cibles. Dans L3, le 8 dépasse la cible 6 ; dans L4, seul le 4 ne dépasse pas la cible 4.

RÉPONSE À RETENIR

```
plateau_apres_regle1_lignes = [
    [7, 9, 2, 3, 2],
    [8, 6, 3, 5, 1],
    [7, 7, 3, 2, 7],
    [6, 4, 5, 0, 2],
    [0, 0, 0, 0, 4]
]
```

Question 7.

Sujet p. 12, 13

> Recopier et compléter les lignes 4 et 10 pour compléter le code de la fonction regle1.

Pour la ligne d'indice `i`, la cible est `ciblesLignes[i]`. Pour la colonne d'indice `j`, elle est `ciblesColonnes[j]`. Par exemple, sur le plateau de l'énoncé, la cible de L3 vaut 6 : le 8 de cette ligne est donc mis à 0. Lors du passage sur les colonnes, la cible de C3 vaut 2 ; ses valeurs 3 et 5 sont alors mises à 0.

RÉPONSE À RETENIR

```
def regle1(plateau, ciblesLignes, ciblesColonnes):
    for i in range(5):
        tab = extraireLigne(plateau, i)
        cible = ciblesLignes[i]
        for j in range(5):
            if tab[j] > cible:
                plateau[i][j] = 0
    for j in range(5):
        tab = extraireColonne(plateau, j)
        cible = ciblesColonnes[j]
        for i in range(5):
            if tab[i] > cible:
                plateau[i][j] = 0
```

Sur le plateau de la figure 1, après l'étape des lignes, L3 devient `[6, 4, 5, 0, 2]` et L4 devient `[0, 0, 0, 0, 4]`. À l'étape des colonnes, C3, de cible 2, met à zéro ses valeurs initiales 3 et 5. Les autres colonnes ne changent pas.

Question 8.

Sujet p. 13

> Écrire une fonction unImpair qui prend comme paramètre une liste d'entiers et renvoie True si la liste ne contient qu'un seul entier impair et False sinon.

On compte les nombres impairs et on renvoie vrai si, et seulement si, leur nombre vaut 1. Le parcours examine chaque élément de la liste et se termine donc après un nombre fini d'étapes.

RÉPONSE À RETENIR

```
def unImpair(liste):
    nb_impairs = 0
    for nombre in liste:
        if nombre % 2 == 1:
            nb_impairs = nb_impairs + 1
    return nb_impairs == 1
```

Trace avec `[6, 4, 5, 8, 2]` : le compteur vaut successivement 0, 0, 1, 1, 1. La fonction renvoie `True`. Avec `[1, 2, 3, 5, 2]`, il y a trois impairs ; elle renvoie `False`.

Question 9.

Sujet p. 13, 14

> Recopier et compléter les lignes 4, 5, 11 et 12 pour compléter le code de la fonction regle2, qui prend comme paramètres un plateau, ciblesLignes et ciblesColonnes et applique la règle 2.

Pour chaque ligne, on vérifie que sa cible est paire et que la ligne contient exactement un impair. Si ces deux conditions sont remplies, cet impair est remplacé par 0. On applique ensuite la même démarche à chaque colonne, avec sa cible.

RÉPONSE À RETENIR

```
def regle2(plateau, ciblesLignes, ciblesColonnes):
    for i in range(5):
        ligne = extraireLigne(plateau, i)
        impair = unImpair(ligne)
        if ciblesLignes[i] % 2 == 0 and impair:
            for j in range(5):
                if plateau[i][j] % 2 == 1:
                    plateau[i][j] = 0
    for j in range(5):
        colonne = extraireColonne(plateau, j)
        impair = unImpair(colonne)
        if ciblesColonnes[j] % 2 == 0 and impair:
            for i in range(5):
                if plateau[i][j] % 2 == 1:
                    plateau[i][j] = 0
```

Exemple sur L3 du plateau initial : la ligne vaut `[6, 4, 5, 8, 2]` et sa cible vaut 6. Il y a un seul impair, le 5 ; il est remplacé par 0, ce qui donne `[6, 4, 0, 8, 2]`.

Partie C : Problème sur une ligne et représentation binaire

Question 10.

Sujet p. 14

> Expliquer pourquoi [1, 1, 0, 0, 1] est un masque solution pour la liste [1, 2, 3, 5, 2] et la cible 5. Donner tous les autres masques solutions.

Les bits 1 du masque `[1, 1, 0, 0, 1]` sélectionnent les valeurs 1, 2 et 2. Leur somme vaut 1 + 2 + 2 = 5. Pour trouver les autres masques, on cherche les autres choix de positions dont les valeurs totalisent 5.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

Les autres masques solutions sont `[0, 0, 0, 1, 0]` (le 5 seul), `[0, 1, 1, 0, 0]` (2 + 3) et `[0, 0, 1, 0, 1]` (3 + 2). Avec le masque donné, cela fait quatre masques solutions au total.

Question 11.

Sujet p. 14

Écrire une fonction somme qui prend comme paramètres une liste de 5 entiers et un masque (une liste de taille 5 de 0 et de 1), et renvoie la somme des chiffres du tableau correspondant au masque. Par exemple, somme([1, 5, 3, 4, 8], [0, 1, 1, 0, 1]) doit renvoyer 5 + 3 + 8 = 16.

On parcourt les cinq positions et on ajoute à un total le chiffre correspondant chaque fois que le bit du masque vaut 1.

RÉPONSE À RETENIR

```
def somme(liste, masque):
    total = 0
    for i in range(5):
        if masque[i] == 1:
            total = total + liste[i]
    return total
```

Trace avec l'exemple de l'énoncé : les contributions successives sont 0, 5, 3, 0 et 8. Le total vaut 0 + 5 + 3 + 0 + 8 = 16 ; la fonction renvoie `16`.

Question 12.

Sujet p. 14

Donner la représentation binaire sur 5 bits de l'entier 26 sous la forme d'une liste de taille 5.

On décompose 26 en puissances de 2 parmi les cinq poids possibles : 26 = 16 + 8 + 2. Les bits correspondant aux poids 16, 8, 4, 2 et 1 sont donc 1, 1, 0, 1 et 0.

RÉPONSE À RETENIR

La représentation binaire sur 5 bits est `[1, 1, 0, 1, 0]`.

Question 13.

Sujet p. 14

Expliquer pourquoi on ne représente que les entiers compris entre 0 et 31 sur 5 bits.

Chaque bit peut prendre deux valeurs, 0 ou 1. Il existe donc $2^5 = 32$ listes de cinq bits. La plus petite représente 00000, soit 0 ; la plus grande représente 11111, soit 16 + 8 + 4 + 2 + 1 = 31.

RÉPONSE À RETENIR

Avec 5 bits, on représente tous les entiers de 0 à 31 inclus.

Question 14.

Sujet p. 14

Écrire une fonction dec2bin qui prend comme paramètre un entier compris entre 0 et 31 et renvoie sa

FormaV - document d'entraînement non officiel

représentation binaire sous la forme d'une liste de 5 bits. Par exemple, dec2bin(9) renvoie [0, 1, 0, 0, 1].

La division entière répétée par 2 fournit les bits de droite à gauche : le reste donne le bit courant, puis le quotient sert à l'étape suivante. On remplit les cinq positions de la liste en partant de l'indice 4.

RÉPONSE À RETENIR

```
def dec2bin(n):
    bits = [0, 0, 0, 0, 0]
    for i in range(4, -1, -1):
        bits[i] = n % 2
        n = n // 2
    return bits
```

Trace pour `n = 9` : on inscrit les restes 1, 0, 0, 1 et 0 aux indices 4, 3, 2, 1 et 0, respectivement. La liste finale est `[0, 1, 0, 0, 1]`, comme dans l'énoncé.

Question 15.

Sujet p. 14, 15

Écrire une fonction masques_solutions qui prend comme paramètres une liste de taille 5 d'entiers et une cible, et renvoie la liste de tous les masques solutions correspondants.

Chaque entier de 0 à 31 fournit un masque différent grâce à `dec2bin`. On calcule la somme correspondant à chacun des 32 masques et on ajoute à la liste résultat ceux dont la somme est égale à la cible. La boucle termine après ces 32 essais.

RÉPONSE À RETENIR

```
def masques_solutions(liste, cible):
    solutions = []
    for n in range(32):
        masque = dec2bin(n)
        if somme(liste, masque) == cible:
            solutions.append(masque)
    return solutions
```

Exemple avec la liste `[1, 2, 3, 5, 2]` et la cible 5 : les masques retenus correspondent aux entiers 2, 5, 12 et 25. Leurs sommes sélectionnées valent respectivement 2, 5, 3 + 2 et 1 + 2 + 2. La fonction renvoie `[[0, 0, 0, 1, 0], [0, 0, 1, 0, 1], [0, 1, 1, 0, 0], [1, 1, 0, 0, 1]]`.

Partie D : Retour au jeu « Objectif Somme »

Question 16.

Sujet p. 12, 15

Écrire une fonction teste_solution qui prend comme paramètres un plateau, la liste des cibles des lignes et la liste des cibles des colonnes, et retourne True si les valeurs des cases restantes du plateau vérifient bien les cibles sur chaque ligne et chaque colonne, et False sinon.

On vérifie successivement la somme de chaque ligne, puis celle de chaque colonne. Si une seule somme diffère de sa cible, la fonction renvoie `False`. On vérifie aussi qu'aucune ligne n'est entièrement vide, conformément à la règle du jeu. Avec des cibles de lignes positives et des sommes correctes, cette dernière condition est déjà satisfaite, mais le test la rend explicite.

RÉPONSE À RETENIR

```
def teste_solution(plateau, ciblesLignes, ciblesColonnes):
    for i in range(5):
        if sum(plateau[i]) != ciblesLignes[i]:
            return False
        if not any(plateau[i][j] != 0 for j in range(5)):
            return False
    for j in range(5):
        total = 0
        for i in range(5):
            total = total + plateau[i][j]
        if total != ciblesColonnes[j]:
            return False
    return True
```

Trace sur le plateau solution de la figure 1 : les sommes des lignes sont `[13, 9, 12, 6, 4]` et celles des colonnes `[15, 13, 5, 2, 9]`. Chaque ligne contient au moins un chiffre non nul ; ces sommes correspondent aux cibles. La fonction renvoie donc `True`.

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