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 2

Code sujet **25-NSIJ2NC1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Cryptographie, signature et empreinte de message
12 réponses • sujet p. 2–6

Exercice 2 — Bases de données, requêtes SQL et gestion de bugs en Python
12 réponses • sujet p. 7–11

Exercice 3 — Siteswaps, jonglage, structures de données et graphes
13 réponses • sujet p. 12–18

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

Cryptographie, signature et empreinte de message

Partie A : Cryptographie symétrique

Question 1.

Sujet p. 2

> Donner l'instruction que doit écrire Bob pour déchiffrer le message d'Alice et affecter le résultat dans une variable m2.

Le chiffrement symétrique se déchiffre en appliquant de nouveau la même fonction avec la même clé. Puisque `m1 = code(m0, cle)` et que `code(code(m0, cle), cle) == m0`, Bob applique `code` à `m1` avec `cle`.

> **RÉPONSE À RETENIR**
>
> `m2 = code(m1, cle)`

Partie B : Cryptographie asymétrique

Question 2.

Sujet p. 3

> Donner l'instruction que doit réaliser Bob pour déchiffrer le message d'Alice afin de connaître l'heure et le lieu du rendez-vous.

Alice a chiffré le message avec la clé publique de Bob, `cle1_b`. Pour retrouver le message, Bob utilise la clé associée, sa clé privée `cle2_b`.

> **RÉPONSE À RETENIR**
>
> `m2 = code(m1, cle2_b)`

Question 3.

Sujet p. 3

> Justifier qu'il est désormais impossible pour Eve de prendre connaissance du contenu du message secret.

Eve connaît le message chiffré `m1` et la clé publique `cle1_b`. D'après les hypothèses de l'énoncé, connaître cette clé ne permet pas de retrouver la clé privée associée `cle2_b`, et il est impossible de déchiffrer le message sans cette clé privée.

Cette conclusion suppose que la clé publique utilisée est bien celle de Bob : si une personne malveillante pouvait la remplacer par sa propre clé publique, la confidentialité ne serait plus garantie.

> **RÉPONSE À RETENIR**
>
> Dans le modèle de l'exercice, Eve ne peut pas lire le message sans `cle2_b`, que seul Bob possède.

Question 4.

Sujet p. 3

> On parle de système de clé privée/clé publique. Dans l'échange précédent, indiquer quelle est la clé privée et

FormaV - document d'entraînement non officiel

> quelle est la clé publique.

Bob diffuse `cle1_b` sur Internet : c'est sa clé publique. Il ne diffuse pas `cle2_b` : c'est sa clé privée, qu'il doit garder secrète.

> **RÉPONSE À RETENIR**
>
> Clé publique de Bob : `cle1_b`. Clé privée de Bob : `cle2_b`.

Question 5.

Sujet p. 3

> Bob souhaite accuser bonne réception de ce rendez-vous et transmettre à Alice le message suivant 'Bien reçu. Rendez-vous à 16h donc.', dont Eve ne doit pas pouvoir prendre connaissance. Donner, dans l'ordre, les instructions que doivent réaliser Bob et Alice pour sécuriser ce deuxième envoi.

Pour que le message soit confidentiel, Bob le chiffre avec la clé publique d'Alice, que celle-ci a diffusée. Alice le déchiffre ensuite avec sa clé privée associée. Par exemple, en utilisant des variables distinctes des noms `m0` et `m3` employés ailleurs dans l'exercice :

1. Bob chiffre le message : `accuse_chiffre = code('Bien reçu. Rendez-vous à 16h donc.', cle1_a)`.
2. Alice déchiffre le message reçu : `accuse_recu = code(accuse_chiffre, cle2_a)`.

Bob doit s'assurer qu'il utilise bien la clé publique d'Alice.

> **RÉPONSE À RETENIR**
>
> Bob chiffre avec `cle1_a` ; Alice déchiffre avec `cle2_a`.

Question 6.

Sujet p. 3

> Expliquer pourquoi il est nécessaire d'avoir les deux clés au lieu de n'avoir que la clé privée.

Les deux clés ont des rôles complémentaires. La clé publique peut être diffusée : elle permet aux correspondants de chiffrer un message destiné à Bob. Seule la clé privée associée, que Bob garde secrète, permet de le déchiffrer. Avec une seule clé privée, Bob devrait la communiquer aux expéditeurs pour qu'ils puissent chiffrer ; toute personne qui l'intercepterait pourrait alors également déchiffrer les messages.

> **RÉPONSE À RETENIR**
>
> La clé publique permet à tous de chiffrer un message destiné à Bob, tandis que la clé privée permet à Bob seul de le déchiffrer.

Partie C : Signature

Question 7.

Sujet p. 4

> Expliquer en quoi les informations m0, m0_s et cle1_a connues de Bob permettent de garantir à la fois que le message provient bien d'Alice et que Mallory n'a pas pu envoyer le message m3 à Bob en se faisant passer pour Alice.

Bob déchiffre la signature avec la clé publique d'Alice en calculant `code(m0_s, cle1_a)`. D'après la propriété des clés associées, il doit retrouver le message qui a été signé, ici `m0`. Si le résultat est identique au message reçu, la signature correspond à ce message.

Sous l'hypothèse que `cle1_a` est bien la clé publique d'Alice et que sa clé privée `cle2_a` est restée secrète, Mallory ne peut pas fabriquer une signature valide pour le message différent `m3`. S'il remplace `m0` par `m3` tout en gardant la signature de `m0`, Bob retrouve `m0` en vérifiant la signature : la comparaison avec `m3` échoue. La signature authentifie l'origine et permet de détecter cette modification, mais elle ne chiffre pas le message.

> **RÉPONSE À RETENIR**
>
> Bob vérifie que `code(m0_s, cle1_a) == m0`. Sous les hypothèses de confiance sur la clé publique et le secret de la clé privée d'Alice, une signature valide authentifie `m0` et ne peut pas être réutilisée pour faire accepter `m3`.

Partie D : Signature par empreinte

Question 8.

Sujet p. 5

> Donner sans justifier la réduction de la chaîne de caractères 'bac'.

Pour `'bac'`, les codes des caractères sont 98, 97 et 99. La réduction vaut donc `1 × |97 − 98| + 2 × |99 − 97| = 1 + 4 = 5`.

> **RÉPONSE À RETENIR**
>
> `5`

Question 9.

Sujet p. 5

> Écrire en Python une fonction reduction qui calcule l'entier correspondant à la réduction d'une chaîne de caractères. Exemples : reduction('abca') renvoie 9, reduction(m0) renvoie 62073 et reduction(m3) renvoie 53681.

Pour chaque indice `i` de 1 à `len(s) - 1`, on ajoute `i` multiplié par la différence absolue entre les codes des caractères d'indices `i` et `i - 1`. La boucle ne s'exécute pas si la chaîne est vide ou ne contient qu'un caractère ; dans ces cas, la somme vaut 0.

```
def reduction(s):
    total = 0
    for i in range(1, len(s)):
        total += i * abs(ord(s[i]) - ord(s[i - 1]))
    return total
```

Pour `'abca'`, les contributions sont `1`, `2` et `6`, soit `9`, conformément à l'exemple.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

La fonction applique la définition de la réduction et renvoie un entier. Les autres exemples fournis dans l'énoncé sont `reduction(m0) == 62073` et `reduction(m3) == 53681`.

Question 10.

Sujet p. 4, 5

> Alice calcule m0_r = str(reduction(m0)) puis m0_s = code(m0_r, cle2_a), qui est la nouvelle signature de m0. Décrire ce que doit désormais réaliser Bob pour vérifier l'authenticité du message, c'est-à-dire qu'il a bien été envoyé par Alice.

Bob reçoit le message et la signature de son empreinte. Il déchiffre la signature avec la clé publique d'Alice, calcule la réduction du message reçu, puis transforme cette réduction en chaîne avec `str`. Il compare les deux chaînes. Si elles sont identiques, la signature correspond à l'empreinte du message reçu. Sous réserve que la clé publique soit bien celle d'Alice et que sa clé privée soit restée secrète, cela permet d'authentifier l'origine du message dans le modèle de l'exercice.

RÉPONSE À RETENIR

Bob vérifie que `code(m0_s, cle1_a) == str(reduction(m0))`, où `m0` est le message reçu. L'égalité valide la signature pour ce message. Cette réduction simple peut toutefois présenter des collisions ; elle ne constitue pas, à elle seule, une empreinte robuste pour un système réel.

Question 11.

Sujet p. 6

> Pour réduire davantage encore la taille de la signature, on propose de n'utiliser que les dix premiers indices du message m0 dans le calcul de la réduction plutôt que le message en entier. Commenter cette approche.

Si l'on ne tient compte que des dix premiers indices, les caractères d'indices 0 à 9, les caractères suivants ne participent pas au calcul de la réduction. Une modification après ces dix caractères ne changerait donc pas l'empreinte calculée et pourrait ne pas être détectée par la vérification de la signature. En outre, la réduction définie dans l'exercice est une somme et peut produire la même valeur pour des messages différents.

RÉPONSE À RETENIR

Cette méthode raccourcit le calcul, mais elle ne garantit pas l'intégrité du message : les modifications après les dix premiers indices ne sont pas prises en compte, et la réduction peut présenter des collisions.

Partie E : Chiffrement et signature

Question 12.

Sujet p. 3, 4, 5, 6

> Dans le protocole proposé dans la dernière partie, le message m0 est envoyé sans être chiffré et Eve peut en prendre connaissance. Proposer un protocole qui permet à Alice d'envoyer un message à Bob de manière confidentielle en certifiant que ce message provient bien d'elle et de manière à ce que Eve ne puisse pas en prendre connaissance. Détailler également la procédure que doit suivre Bob pour déchiffrer le message et garantir qu'il provient d'Alice.

Alice calcule l'empreinte du message entier, puis la signe avec sa clé privée. Elle chiffre ensuite le message et la signature avec la clé publique de Bob. Elle transmet à Bob les deux éléments chiffrés. En notant `m0` le message à envoyer, le protocole peut s'écrire ainsi :

```
empreinte = str(reduction(m0))
signature = code(empreinte, cle2_a)
message_chiffre = code(m0, cle1_b)
signature_chiffree = code(signature, cle1_b)
```

Bob déchiffre les deux éléments avec sa clé privée, puis vérifie la signature en la déchiffrant avec la clé publique d'Alice et en comparant l'empreinte obtenue à celle du message déchiffré :

```
m_recu = code(message_chiffre, cle2_b)
signature_recue = code(signature_chiffree, cle2_b)
empreinte_signee = code(signature_recue, cle1_a)
signature_valide = empreinte_signee == str(reduction(m_recu))
```

Bob accepte le message comme authentifié si `signature_valide` vaut `True`. Le chiffrement avec `cle1_b` assure la confidentialité dans le modèle de l'exercice ; la signature vérifiée avec `cle1_a` permet l'authentification et la détection des modifications, sous réserve que Bob connaisse bien la véritable clé publique d'Alice et qu'Alice utilise bien la véritable clé publique de Bob. La fonction de réduction donnée reste une empreinte jouet susceptible de collisions ; cette limite peut empêcher de garantir solidement l'intégrité contre des falsifications.

RÉPONSE À RETENIR

Alice signe l'empreinte du message avec `cle2_a`, puis chiffre le message et la signature avec `cle1_b`. Bob déchiffre avec `cle2_b` et vérifie la signature avec `cle1_a` en comparant l'empreinte signée à celle du message reçu. La confidentialité dépend de la clé publique de Bob ; l'authentification dépend de la clé publique d'Alice et de la confidentialité de sa clé privée. La réduction de l'exercice étant fragile, cette garantie d'intégrité est limitée par ses collisions possibles.

Exercice 2

6 points

Bases de données, requêtes SQL et gestion de bugs en Python

Question 1.

Sujet p. 7, 8

> Expliquer le rôle d'une clé primaire et rappeler la contrainte dans le choix de celle-ci.

Une clé primaire identifie sans ambiguïté chaque ligne d'une table. Sa valeur doit être unique et ne peut pas être absente (*NULL*). Elle peut être constituée d'un seul attribut ou, si nécessaire, d'une combinaison de plusieurs attributs.

RÉPONSE À RETENIR

La clé primaire identifie chaque enregistrement de manière unique ; ses valeurs doivent être uniques et non nulles.

Question 2.

Sujet p. 7, 8, 9

> À l'aide des extraits de tables, détailler un achat d'Emile Pasteur en précisant la date de son achat, le ou les article(s) acheté(s) et, si c'est le cas, le taux de remise dont il a bénéficié.

Dans la table `client`, Emile Pasteur a pour référence `25145`. La ligne de la table `vente` dont `ref_client` vaut `25145` indique un achat le `2025/03/18`, du produit de référence `89588`, en quantité 5, avec la remise `275`. Dans les tables `produit` et `remise`, ces références correspondent respectivement à « Sac sable » et à un taux de remise de 30 %.

RÉPONSE À RETENIR

Emile Pasteur a acheté 5 sacs de sable le 18 mars 2025 et a bénéficié d'une remise de 30 %.

Question 3.

Sujet p. 9

> Un nouveau client doit être entré dans la base, il s'agit de Gilles Bertaut, son email est gbertaut@fmail.fr et son numéro de téléphone est 0641424344. Son identifiant (ref_client) dans la base sera 25345. Écrire une requête SQL permettant cet ajout.

On insère une ligne dans `client` en précisant les attributs dans l'ordre de la table : référence, nom, prénom, email, téléphone. Les valeurs textuelles, notamment le numéro de téléphone, sont écrites entre apostrophes.

RÉPONSE À RETENIR

```
INSERT INTO client (ref_client, nom, prenom, email, telephone)
VALUES (25345, 'Bertaut', 'Gilles', 'gbertaut@fmail.fr', '0641424344');
```

Question 4.

Sujet p. 7, 9

> Une correction est à apporter dans la base, l'email de la cliente Shar Sofia n'a pas été correctement saisi ; voici son email correct : 'shars@fmail.fr'. Écrire une requête SQL permettant cette mise à jour.

La cliente Sofia Shar est identifiée dans l'extrait par la clé primaire `ref_client = 25322`. On cible cette clé dans la clause `WHERE` afin de ne modifier que l'enregistrement concerné.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
UPDATE client
SET email = 'shars@fmail.fr'
WHERE ref_client = 25322;
```

Question 5.

Sujet p. 8, 9

> Dans la base de données, toutes les dates sont de type chaîne de caractères au format 'aaaa/mm/jj'. Cela permet de comparer des dates entre elles. Écrire une requête qui permet d'afficher les attributs ref_client de tous les clients ayant fait un achat à partir du 1er janvier 2025. On souhaite qu'un même client n'apparaisse qu'une seule fois dans cet affichage.

La date du 1er janvier 2025 est incluse : on sélectionne donc les ventes dont la date est supérieure ou égale à `2025/01/01`. Comme les dates sont au format `aaaa/mm/jj`, leur comparaison sous forme de chaînes respecte l'ordre chronologique. `DISTINCT` empêche un client ayant plusieurs achats d'apparaître plusieurs fois.

RÉPONSE À RETENIR

```
SELECT DISTINCT ref_client
FROM vente
WHERE date >= '2025/01/01'
ORDER BY ref_client;
```

Sur les lignes de la table `vente` fournies, les quatre achats sont datés de mars 2025. Les références client distinctes sont donc :

ref_client
25123
25145
25149

Ce résultat concerne les seuls extraits visibles ; d'autres lignes éventuelles de la base complète ne sont pas fournies.

Question 6.

Sujet p. 7, 8, 9

> La tondeuse de référence 90222 vendue dans le magasin a un défaut de fabrication. Le responsable doit rappeler tous les clients qui ont acheté cette tondeuse depuis le 15 septembre 2024, date de mise en stock des tondeuses défectueuses. Écrire une requête qui permet d'obtenir le nom et le numéro de téléphone des clients à rappeler.

On relie chaque vente au client correspondant par l'égalité de leurs références client. On filtre ensuite les ventes portant sur le produit `90222` et datées à partir du 15 septembre 2024 inclus. `DISTINCT` évite de répéter un client s'il a effectué plusieurs achats concernés.

RÉPONSE À RETENIR

```
SELECT DISTINCT client.nom, client.telephone
FROM client
JOIN vente ON client.ref_client = vente.ref_client
WHERE vente.ref_produit = 90222
  AND vente.date >= '2024/09/15';
```

Dans les extraits de ventes fournis, aucune ligne ne porte la référence produit `90222`. Le résultat sur ces seuls extraits est donc le tableau vide suivant :

nom	telephone

Cela ne permet pas de conclure qu'il n'existe aucune vente correspondante dans la base complète.

Question 7.

Sujet p. 9, 10

> Recopier et compléter la ligne 2 du code de la fonction select_tel.

Dans le dictionnaire `client`, la clé est la référence client et la liste associée suit l'ordre des attributs de la table. Le téléphone est le cinquième attribut ; son indice est donc 3, car l'indexation Python commence à 0.

RÉPONSE À RETENIR

```
return client[ref_client][3]
```

Question 8.

Sujet p. 9, 10

> On considère le code de la fonction select_tel complétée. L'exécution de select_tel(client, 25137) renvoie le bon résultat '0604030201', mais l'exécution de select_tel(client, 1234) provoque l'erreur KeyError: 1234. Expliquer ce qui, concernant le dictionnaire client, est à l'origine de cette erreur et à quelle situation pour le magasin cela correspond.

L'expression `client[1234]` cherche une clé égale à `1234`. Comme cette clé est absente du dictionnaire, Python déclenche une exception `KeyError` avant de pouvoir lire le numéro de téléphone. Pour le magasin, cela correspond à une référence client qui ne désigne aucun enregistrement présent dans la base représentée par le dictionnaire.

RÉPONSE À RETENIR

L'erreur est due à l'absence de la clé `1234` dans le dictionnaire `client`. Elle correspond à une référence client qui ne désigne aucun enregistrement présent dans la base.

Question 9.

Sujet p. 10

> Recopier le code de la fonction select_tel en proposant une modification pour que la fonction renvoie None plutôt que de provoquer une KeyError dans le cas où la situation précédente se présente.

On teste d'abord si la référence est une clé du dictionnaire. Si elle en est absente, la fonction renvoie explicitement `None` ; sinon, elle renvoie le téléphone situé à l'indice 3.

RÉPONSE À RETENIR

```
def select_tel(client, ref_client):
    if ref_client in client:
        return client[ref_client][3]
    return None
```

Question 10.

Sujet p. 8, 10

Écrire le code en Python de la fonction nb_produits. Elle prend comme paramètres vente, un dictionnaire associé à la table vente, et ref_produit, un entier associé à l'attribut ref_produit du dictionnaire de la table vente. Elle renvoie le nombre total de produits de cette référence vendus.

Chaque valeur de `vente` est une liste de la forme `[date, ref_produit, ref_client, quantité, ref_remise]`. On parcourt les ventes et on additionne la quantité, à l'indice 3, lorsque la référence produit, à l'indice 1, correspond à celle recherchée. Si aucune vente ne correspond, la somme reste égale à 0.

RÉPONSE À RETENIR

```
def nb_produits(vente, ref_produit):
    total = 0
    for achat in vente.values():
        if achat[1] == ref_produit:
            total = total + achat[3]
    return total
```

Avec les ventes de l'extrait, `nb_produits(vente, 85235)` renvoie `1` : la vente `25633` porte sur une unité de ce produit.

Question 11.

Sujet p. 7, 8, 11

Dans cette question, on considère que les tables de la base de données contiennent exactement ce qui est présenté dans les extraits en début d'exercice. On exécute la requête SQL : DELETE FROM produit WHERE ref_produit = 89363;. Celle-ci n'est pas exécutée et on obtient le message d'erreur foreign key constraint failed. Expliquer pourquoi il est nécessaire que la requête demandée ne soit pas exécutée et qu'elle affiche ce message d'erreur.

La vente `25632` de la table `vente` a pour `ref_produit` la valeur `89363`. Supprimer le produit correspondant de la table `produit` laisserait cette vente avec une référence vers un produit inexistant, ce qui violerait la contrainte de clé étrangère.

RÉPONSE À RETENIR

La suppression doit être refusée pour préserver l'intégrité référentielle : la vente `25632` référence le produit `89363`. Le message `foreign key constraint failed` signale cette violation.

Question 12.

Sujet p. 8, 10, 11

On considère le code de la fonction delete_prod, qui permet de supprimer un produit dont la référence est précisée et prend en paramètres produit, vente et ref_produit. Réécrire le code de cette fonction en faisant toutes les modifications et tous les ajouts nécessaires pour qu'à son appel, elle refuse la suppression du produit lorsqu'on

> rencontre la situation présentée à la question précédente.

Avant la suppression, on parcourt les ventes pour vérifier si l'une d'elles référence le produit demandé. Si c'est le cas, la fonction refuse la suppression sans modifier le dictionnaire. Sinon, elle supprime le produit. Le code adopte une convention de retour explicite : `False` en cas de refus ou si la référence du produit est absente, et `True` si la suppression est effectuée.

RÉPONSE À RETENIR

```
def delete_prod(produit, vente, ref_produit):
    if ref_produit not in produit:
        return False

    for achat in vente.values():
        if achat[1] == ref_produit:
            return False

    del produit[ref_produit]
    return True
```

La référence produit se trouve à l'indice 1 dans chaque liste associée à une vente. Pour `89363`, la fonction renvoie `False`, car la vente `25632` référence ce produit : celui-ci n'est donc pas supprimé.

Exercice 3

8 points

Siteswaps, jonglage, structures de données et graphes

Partie A

Question 1.

Sujet p. 13

> Si on se donne l'état e2 = [1, 1, 0, 1, 0, 0], indiquer quels sont les lancers possibles.

Le premier élément de l'état vaut 1 : une balle est prête à être lancée. Un lancer positif est possible si la case d'indice égal au numéro du lancer est libre. Dans `e2 = [1, 1, 0, 1, 0, 0]`, les cases libres parmi les indices 1 à 5 sont celles d'indices 2, 4 et 5.

> **RÉPONSE À RETENIR**
>
> Les lancers possibles sont **2, 4 et 5**.

Question 2.

Sujet p. 13

> Même question pour l'état e3 = [0, 1, 1, 0, 1].

Dans `e3 = [0, 1, 1, 0, 1]`, le premier élément vaut 0 : aucune balle n'est dans la main. Par convention, le seul lancer possible dans cette situation est le lancer 0.

> **RÉPONSE À RETENIR**
>
> Le seul lancer possible est **0**.

Question 3.

Sujet p. 13

> Recopier et compléter les lignes 4, 7 et 8 du code de la fonction lancer_possible. Elle prend en argument un tableau etat représentant un état et un entier lancer, et renvoie True si le lancer est possible, et False sinon.

Le lancer 0 doit être refusé si une balle est dans la main. Pour un lancer positif, il faut une balle dans la main et une case d'arrivée libre. Les lignes complétées sont donc celles du code ci-dessous.

```
def lancer_possible(etat, lancer):
    if lancer >= len(etat) or lancer < 0:
        return False
    if lancer == 0 and etat[0] == 1:
        return False
    if lancer > 0:
        if etat[0] == 0 or etat[lancer] == 1:
            return False
    return True
```

Par exemple, dans `e1 = [1, 0, 0, 1, 1, 0]`, le lancer 2 est possible car `etat[2]` vaut 0 et une balle est en main. Le lancer 3 est impossible car `etat[3]` vaut 1 ; le lancer 0 est aussi impossible car `etat[0]` vaut 1.

RÉPONSE À RETENIR

Ligne 4 : `if lancer == 0 and etat[0] == 1:`
Ligne 7 : `if etat[0] == 0 or etat[lancer] == 1:`
Ligne 8 : `return False`

Question 4.

Sujet p. 13, 14

Depuis l'état e2 = [1, 1, 0, 1, 0, 0], on effectue un lancer de 5. Donner l'état qu'on obtient après le lancer et l'effet de la gravité.

Le lancer 5 est possible car la case d'indice 5 de `e2` vaut 0. On place d'abord la balle à cet indice : `[1, 1, 0, 1, 0, 1]`. La gravité décale ensuite les éléments d'un rang vers la gauche et ajoute un 0 à la fin : on obtient `[1, 0, 1, 0, 1, 0]`.

RÉPONSE À RETENIR

Après le lancer puis l'effet de la gravité, l'état est **[1, 0, 1, 0, 1, 0]**.

Question 5.

Sujet p. 13, 14

Recopier et compléter la ligne 4 du code de la fonction lancer_balle ci-dessous. On peut insérer plusieurs lignes si besoin. La fonction prend en paramètres un état etat de jonglage et un entier positif lancer, ne doit pas modifier l'état passé en paramètre et doit renvoyer un nouvel état correspondant au résultat du lancer. On suppose sans le vérifier que le lancer est forcément valide.

On effectue le lancer sur une copie de l'état, pour ne pas modifier la liste reçue en paramètre. On place la balle à l'indice `lancer`, puis on décale l'état d'un rang vers la gauche pour représenter la gravité.

Exemple : depuis `e1 = [1, 0, 0, 1, 1, 0]`, le lancer 2 donne d'abord `[1, 0, 1, 1, 1, 0]`, puis, après la gravité, `[0, 1, 1, 1, 0, 0]`. La liste `e1` initiale n'est pas modifiée.

RÉPONSE À RETENIR

```
def lancer_balle(etat, lancer):
    # copie de l'etat pour ne pas le modifier
    nouvel_etat = [balle for balle in etat]
    nouvel_etat[lancer] = 1
    nouvel_etat = nouvel_etat[1:] + [0]
    return nouvel_etat
```

Partie B

Question 6.

Sujet p. 12, 13, 14

Écrire une fonction liste_lancers_possibles qui prend en paramètre un état etat et qui renvoie une liste d'entiers correspondant à l'ensemble des lancers possibles à partir de cet état. Exemples : liste_lancers_possibles(e1) renvoie [1, 2, 5] et liste_lancers_possibles([0, 1, 1, 1, 0]) renvoie [0].

On teste les valeurs possibles de `lancer`, de 0 à `len(etat) - 1`, et on ajoute à la liste celles pour lesquelles `lancer_possible` renvoie `True`. Le lancer 0 est ainsi traité par cette fonction de test.

Pour `e1`, les lancers retenus sont 1, 2 et 5. Pour `[0, 1, 1, 1, 0]`, le premier élément vaut 0 et seul le lancer 0 est retenu.

RÉPONSE À RETENIR

```
def liste_lancers_possibles(etat):
    lancers = []
    for lancer in range(len(etat)):
        if lancer_possible(etat, lancer):
            lancers.append(lancer)
    return lancers
```

`liste_lancers_possibles(e1)` renvoie `[1, 2, 5]` et `liste_lancers_possibles([0, 1, 1, 1, 0])` renvoie `[0]`.

Question 7.

Sujet p. 15

Justifier qu'il s'agit d'une fonction récursive.

Dans le cas `n > 0`, la fonction `calcule_sequences` s'appelle elle-même avec un nouvel état et avec `n - 1`. Lorsque `n` vaut 0, elle atteint le cas de base et renvoie la liste contenant la séquence vide.

RÉPONSE À RETENIR

`calcule_sequences` est une fonction récursive : elle s'appelle elle-même avec un paramètre `n` diminué de 1 et possède le cas de base `n == 0`.

Question 8.

Sujet p. 15

Expliquer brièvement pourquoi elle se termine si n est un entier positif. On admet que les boucles for présentes sont bornées et donc terminent.

Si `n` est un entier positif, chaque appel récursif diminue sa valeur de 1. Après un nombre fini d'appels, `n` atteint 0 et le cas de base renvoie la séquence vide sans nouvel appel récursif. Les boucles étant supposées bornées, elles terminent également.

RÉPONSE À RETENIR

La fonction termine car `n` décroît strictement jusqu'au cas de base `n == 0`, et les boucles terminent.

Question 9.

Sujet p. 15, 16

Recopier et compléter les lignes 10, 12, 14 et 15 de la fonction calcule_sequences.

Pour chaque lancer possible, on calcule l'état obtenu, puis les séquences de longueur `n - 1` réalisables depuis cet état. On ajoute le lancer courant au début de chacune de ces séquences.

Trace avec `e1 = [1, 0, 0, 1, 1, 0]` et `n = 2` : après le lancer 1, l'état est `[1, 0, 1, 1, 0, 0]`, depuis lequel les lancers possibles sont 1, 4 et 5. Les lancers 2 et 5 depuis l'état initial conduisent chacun, après gravité, à un état dont le premier élément vaut 0 ; le lancer suivant est donc 0. Les séquences obtenues sont `[1, 1]`, `[1, 4]`, `[1, 5]`, `[2, 0]` et `[5, 0]`.

RÉPONSE À RETENIR

```
def calcule_sequences(etat, n):
    """ etat est un état de jonglerie, n est un entier.
    Calcule et renvoie les séquences de longueur n réalisables
    à partir de cet état.
    """
    if n == 0:
        return [[]]
    else:
        s_possibles = []
        l_lancers = liste_lancers_possibles(etat)
        for lancer in l_lancers:
            etat2 = lancer_balle(etat, lancer)
            s_etat2 = calcule_sequences(etat2, n - 1)
            for sequence in s_etat2:
                s_possibles.append([lancer] + sequence)
        return s_possibles
```

Partie C

Question 10.

Sujet p. 16, 17

> Recopier et compléter le code Python permettant de représenter l'automate de la figure 4 dans une variable automate.

Chaque clé du dictionnaire est un état. Chaque couple de la liste associée indique le numéro du lancer, puis l'état obtenu. Les transitions ci-dessous correspondent aux arcs et à leurs étiquettes dans la figure 4. Par exemple, depuis `'10010'`, les lancers 1, 2 et 4 mènent respectivement à `'10100'`, `'01100'` et `'00110'`.

RÉPONSE À RETENIR

```
automate = {
    '11000': [(3, '10100'), (2, '11000'), (4, '10010')],
    '01010': [(0, '10100')],
    '10100': [(1, '11000'), (3, '01100'), (4, '01010')],
    '01100': [(0, '11000')],
    '10010': [(1, '10100'), (2, '01100'), (4, '00110')],
    '00110': [(0, '01100')]
}
```

Question 11.

Sujet p. 16, 17

> Écrire le code de la fonction lancer_balle_automate. Elle prend en arguments un automate automate comme décrit plus haut, un état etat et un entier lancer représentant un lancer, et renvoie l'état obtenu lorsqu'on lance lancer depuis l'état etat. On renvoie la chaîne vide si le lancer n'est pas possible. Exemples : lancer_balle_automate(automate, '10010', 2) renvoie '01100' et lancer_balle_automate(automate, '11000', 1) renvoie ''.

On parcourt les arcs sortants de l'état donné et on cherche celui dont l'étiquette correspond au lancer demandé. Le second élément du couple donne l'état obtenu. Si aucun arc ne porte cette étiquette, la fonction renvoie la chaîne vide.

Dans l'automate, depuis `'10010'`, le lancer 2 mène à `'01100'`. Depuis `'11000'`, aucun arc n'est étiqueté 1 : le résultat est donc `''`.

RÉPONSE À RETENIR

```
def lancer_balle_automate(automate, etat, lancer):
    for numero, etat_suivant in automate.get(etat, []):
        if numero == lancer:
            return etat_suivant
    return ''
```

`lancer_balle_automate(automate, '10010', 2)` renvoie `'01100'` et `lancer_balle_automate(automate, '11000', 1)` renvoie `''`.

Question 12.

Sujet p. 17

Écrire le code de la fonction parcours_sequence_depart. Elle prend en argument un automate, un état de départ et une liste de lancers, et renvoie l'état dans lequel on arrive en suivant la séquence de lancers, ou bien None si l'un des lancers était impossible. On peut utiliser la fonction lancer_balle_automate.

On conserve l'état courant et on applique successivement chaque lancer de la séquence. Si la fonction de transition renvoie la chaîne vide, le lancer est impossible et on renvoie `None`. Sinon, l'état courant devient l'état obtenu.

Trace : depuis `'10010'`, le lancer 4 mène à `'00110'`, puis le lancer 0 mène à `'01100'`. Depuis `'10100'`, le lancer 3 mène à `'01100'`, où le lancer 4 est impossible ; le résultat est donc `None`.

RÉPONSE À RETENIR

```
def parcours_sequence_depart(automate, etat_depart, sequence):
    etat = etat_depart
    for lancer in sequence:
        etat_suivant = lancer_balle_automate(automate, etat, lancer)
        if etat_suivant == '':
            return None
        etat = etat_suivant
    return etat
```

La séquence vide laisse l'état inchangé. Si un lancer de la séquence est impossible, la fonction renvoie `None`.

Question 13.

Sujet p. 17, 18

Écrire la fonction departs_siteswap. Elle prend en argument un automate et une liste de lancers (un potentiel siteswap), et renvoie la liste des états de l'automate qui valident le siteswap. On peut utiliser la fonction parcours_sequence_depart et vérifier si le siteswap est possible à partir de chaque état de l'automate.

Une séquence valide comme siteswap doit pouvoir être répétée : à la fin de la séquence, on doit donc être revenu à l'état de départ. On teste cette condition pour chaque état, c'est-à-dire chaque clé de l'automate.

Pour `[1, 2, 3, 4, 0]`, le départ `'10100'` convient : les transitions mènent successivement à `'11000'`, `'11000'`, `'10100'`, `'01010'`, puis `'10100'`. Pour `[2, 1, 0]`, aucun état de départ ne permet de réaliser toute la séquence et de revenir à l'état initial.

RÉPONSE À RETENIR

```
def departs_siteswap(automate, sequence):
    departs = []
    for etat_depart in automate:
        etat_final = parcours_sequence_depart(automate, etat_depart, sequence)
        if etat_final == etat_depart:
            departs.append(etat_depart)
    return departs
```

`departs_siteswap(automate, [1, 2, 3, 4, 0])` renvoie `['10100']` et `departs_siteswap(automate, [2, 1, 0])` renvoie `[]`.

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