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 2024 • Asie • Jour 2

Code sujet **24-NSIJ2JA1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Recherche d’un élément absolument majoritaire dans une liste
11 réponses • sujet p. 2–4

Exercice 2 — Vérification du parenthésage d’une expression à l’aide d’une pile
8 réponses • sujet p. 5–7

Exercice 3 — Bases de données relationnelles, requêtes SQL et ordre lexicographique en Python
14 réponses • sujet p. 8–12

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

Recherche d'un élément absolument majoritaire dans une liste

Question 1.

Sujet p. 2

> Déterminer les effectifs possibles d'un élément absolument majoritaire dans une liste de taille 10.

Un élément absolument majoritaire doit apparaître dans strictement plus de la moitié des 10 emplacements, donc plus de 5 fois. Son effectif est un entier et ne peut pas dépasser 10.

> **RÉPONSE À RETENIR**
>
> Les effectifs possibles sont **6, 7, 8, 9 et 10**.

Partie A : Calcul des effectifs de chaque élément sans dictionnaire

Question 2.

Sujet p. 2

> Écrire une fonction effectif qui prend en paramètres une valeur val et une liste lst et qui renvoie le nombre d'apparitions de val dans lst. Il ne faut pas utiliser la méthode count.

On parcourt la liste et on compare chaque élément à `val`. On incrémente un compteur à chaque égalité. Après le parcours complet, ce compteur est le nombre d'apparitions de `val`. La méthode `count` n'est pas utilisée.

> **RÉPONSE À RETENIR**

```
def effectif(val, lst):
    nb = 0
    for elt in lst:
        if elt == val:
            nb += 1
    return nb
```

Question 3.

Sujet p. 2

> Déterminer le nombre de comparaisons effectuées par l'appel effectif(1, [1, 4, 1, 6, 1, 7, 2, 1, 1]).

La fonction examine les 9 éléments de la liste et effectue une comparaison d'égalité entre chacun d'eux et `1`. Le fait que certaines comparaisons soient vraies ne change pas leur nombre.

> **RÉPONSE À RETENIR**
>
> L'appel effectue **9 comparaisons d'égalité**.

FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 2

> En utilisant la fonction effectif précédente, écrire une fonction majo_abs1 qui prend en paramètre une liste lst et qui renvoie son élément absolument majoritaire s'il existe et renvoie None sinon.

On considère successivement chaque élément de la liste comme candidat et on calcule son effectif avec `effectif`. Il est absolument majoritaire si son effectif dépasse strictement la moitié de la taille de la liste. Si aucun candidat ne vérifie cette condition, on renvoie `None`. Pour une liste vide, la boucle ne s'exécute pas et la fonction renvoie bien `None`.

RÉPONSE À RETENIR

```
def majo_abs1(lst):
    for val in lst:
        if effectif(val, lst) > len(lst) / 2:
            return val
    return None
```

Question 5.

Sujet p. 2

> Déterminer le nombre de comparaisons effectuées par l'appel à majo_abs1([1, 4, 1, 6, 1, 7, 2, 1, 1]).

Le premier candidat est `1`. L'appel `effectif(1, lst)` effectue 9 comparaisons d'égalité et renvoie 5. La fonction teste ensuite si cet effectif dépasse la moitié de 9 : le test `5 > 9 / 2` est une comparaison supplémentaire et il est vrai, donc la fonction s'arrête. Les tests de contrôle de la boucle ne sont pas comptés comme des comparaisons entre valeurs.

RÉPONSE À RETENIR

En comptant les 9 comparaisons d'égalité et la comparaison de majorité, on obtient **10 comparaisons**. Si l'on ne compte que les comparaisons d'égalité entre les éléments de la liste et le candidat, il y en a **9**.

Partie B : Calcul des effectifs de chaque élément dans un dictionnaire

Question 6.

Sujet p. 3

> Recopier et compléter les lignes 3, 4, 5 et 7 de la fonction eff_dico. Elle prend en paramètre une liste lst et renvoie un dictionnaire dont les clés sont les éléments de lst et les valeurs les effectifs de chacun de ces éléments dans lst.

Pour chaque élément rencontré, on vérifie s'il est déjà une clé du dictionnaire. Si oui, on augmente son effectif de 1 ; sinon, on l'ajoute avec l'effectif 1. Ainsi, chaque clé présente dans le dictionnaire correspond à un élément de la liste et sa valeur donne son nombre d'apparitions.

RÉPONSE À RETENIR

```
def eff_dico(lst):
    dico_sortie = {}
    for elt in lst:
        if elt in dico_sortie:
            dico_sortie[elt] += 1
        else:
            dico_sortie[elt] = 1
    return dico_sortie
```

Question 7.

Sujet p. 3

> En utilisant la fonction eff_dico précédente, écrire une fonction majo_abs2 qui prend en paramètre une liste lst et qui renvoie son élément absolument majoritaire s'il existe et renvoie None sinon.

On calcule les effectifs avec `eff_dico`, puis on cherche un effectif strictement supérieur à la moitié de la taille de la liste. Si aucun effectif ne dépasse ce seuil, on renvoie `None`. Pour une liste vide, le dictionnaire est vide et la fonction renvoie également `None`.

RÉPONSE À RETENIR

```
def majo_abs2(lst):
    effectifs = eff_dico(lst)
    for val in effectifs:
        if effectifs[val] > len(lst) / 2:
            return val
    return None
```

Partie C : par la méthode « diviser pour régner »

Question 8.

Sujet p. 3

> On considère lst une liste de taille n. Déterminer l'élément absolument majoritaire de lst si n = 1. C'est le cas de base.

Pour une liste de taille 1, son unique élément apparaît une fois. Comme 1 est strictement supérieur à la moitié de 1, cet élément est absolument majoritaire.

RÉPONSE À RETENIR

L'élément absolument majoritaire est l'unique élément de la liste, `lst[0]`.

Question 9.

Sujet p. 3

> Après avoir partagé lst en lst1 = lst[:n//2] et lst2 = lst[n//2:], si ni lst1 ni lst2 n'admet d'élément absolument majoritaire, expliquer pourquoi lst n'admet pas d'élément absolument majoritaire.

Raisonnons par contradiction. Si `x` était absolument majoritaire dans la liste entière, mais pas dans l'une ou l'autre des deux sous-listes, son effectif dans chacune serait inférieur ou égal à la moitié de la taille de cette sous-liste. En additionnant ces deux effectifs, celui de `x` dans la liste entière serait inférieur ou égal à la

moitié de sa taille. Cela contredirait le fait que `x` y est absolument majoritaire.

> **RÉPONSE À RETENIR**
>
> Tout élément absolument majoritaire de `lst` doit être absolument majoritaire dans au moins une des deux sous-listes. Si aucune sous-liste n'en admet, `lst` n'en admet donc pas non plus.

Question 10.

Sujet p. 3

> Si lst1 admet un élément absolument majoritaire maj1, donner un algorithme pour vérifier si maj1 est l'élément absolument majoritaire de lst.

La majorité de `maj1` dans `lst1` ne suffit pas à conclure pour la liste entière. Il faut compter ses apparitions dans `lst` et vérifier que cet effectif est strictement supérieur à la moitié de `n`.

> **RÉPONSE À RETENIR**
>
> Calculer `effectif(maj1, lst)`. Si le résultat est strictement supérieur à `n / 2`, alors `maj1` est absolument majoritaire dans `lst` ; sinon, il ne l'est pas.

Question 11.

Sujet p. 2, 3, 4

> Recopier et compléter les lignes 4, 11, 13, 15 et 17 de la fonction récursive majo_abs3 qui implémente l'algorithme précédent. Vous pourrez utiliser la fonction effectif de la question 2.

Pour le cas de base, la liste ne contient qu'un élément : il est donc absolument majoritaire. Pour les listes plus grandes, on cherche les candidats éventuels dans chaque moitié, puis on compte leurs apparitions dans la liste entière. On ne renvoie un candidat que si son effectif dépasse strictement `n / 2`. Si aucun candidat ne convient, il faut renvoyer `None`.

La fonction donnée dans le sujet ne traite pas la liste vide : si celle-ci est admise, il faut ajouter un garde-fou avant le cas `n == 1`, comme dans le code ci-dessous. Pour toute liste non vide, chaque appel récursif porte sur des listes plus petites, donc la récursion termine. À chaque niveau, les appels récursifs traitent deux sous-listes et les vérifications de candidats coûtent au plus un temps proportionnel à `n` ; le temps total est donc de l'ordre de `n log n`.

RÉPONSE À RETENIR

```
def majo_abs3(lst):
    n = len(lst)
    if n == 0:
        return None
    if n == 1:
        return lst[0]
    else:
        lst_g = lst[:n // 2]
        lst_d = lst[n // 2:]
        maj_g = majo_abs3(lst_g)
        maj_d = majo_abs3(lst_d)
        if maj_g is not None:
            eff = effectif(maj_g, lst)
            if eff > n / 2:
                return maj_g
        if maj_d is not None:
            eff = effectif(maj_d, lst)
            if eff > n / 2:
                return maj_d
        return None
```

Les lignes demandées dans le squelette du sujet sont donc : ligne 4, `return lst[0]` ; ligne 11, `effectif(maj_g, lst)` ; ligne 13, `return maj_g` ; ligne 15, `effectif(maj_d, lst)` ; ligne 17, `return maj_d`. Le `return None` final est nécessaire lorsque aucun candidat ne convient.

Exercice 2

6 points

Vérification du parenthésage d'une expression à l'aide d'une pile

Question 1.

Sujet p. 5

> Déterminer si l'expression '[2*(i+1)-3) for i in range(3, 10)]' est bien parenthésée. Justifier votre réponse.

En parcourant l'expression de gauche à droite, on rencontre d'abord le crochet ouvrant `[`, puis la parenthèse ouvrante `(`. Le premier `)` ferme correctement cette parenthèse. Mais le `)` suivant devrait fermer le crochet ouvrant, qui attend un `]` : les types ne correspondent pas.

RÉPONSE À RETENIR

L'expression `[2*(i+1)-3) for i in range(3, 10)]` n'est pas bien parenthésée.

Partie A

Question 2.

Sujet p. 5

> Écrire une fonction compte_ouvrante qui prend en paramètre une chaîne de caractères txt et qui renvoie le nombre de parenthèses ouvrantes qu'il contient.

On parcourt les caractères de `txt` et on incrémente un compteur pour chaque caractère ouvrant, c'est-à-dire `(`, `[` ou `{`.

RÉPONSE À RETENIR

```
def compte_ouvrante(txt):
    total = 0
    for caractere in txt:
        if caractere in '([{':
            total += 1
    return total
```

Question 3.

Sujet p. 5

> Écrire une fonction compte_fermante qui prend en paramètre une chaîne de caractères txt et qui renvoie le nombre de parenthèses fermantes qu'il contient.

On applique le même principe, en comptant cette fois les caractères fermants `)`, `]` et `}`.

RÉPONSE À RETENIR

```
def compte_fermante(txt):
    total = 0
    for caractere in txt:
        if caractere in ')]}':
            total += 1
    return total
```


FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 5

> En utilisant les deux fonctions précédentes, écrire une fonction bon_compte qui prend en paramètre une chaîne de caractères txt et qui renvoie True si txt a autant de parenthèses ouvrantes que de parenthèses fermantes et False sinon.

La fonction renvoie le résultat de la comparaison des deux effectifs. Avoir le même nombre de caractères ouvrants et fermants est nécessaire, mais ne garantit pas que la chaîne est bien parenthésée : leur ordre et leur type comptent aussi.

RÉPONSE À RETENIR

```
def bon_compte(txt):
    return compte_ouvrante(txt) == compte_fermante(txt)
```

Question 5.

Sujet p. 5

> Donner un exemple de chaîne de caractères pour laquelle bon_compte renvoie True alors qu'elle n'est pas bien parenthésée.

Dans `)(`, il y a un caractère ouvrant et un caractère fermant, donc les deux effectifs sont égaux. Cependant, le caractère fermant apparaît avant le caractère ouvrant : la chaîne n'est pas bien parenthésée.

RÉPONSE À RETENIR

Par exemple, `bon_compte(')(')` renvoie `True`, alors que `)('` n'est pas bien parenthésée.

Partie B

Question 6.

Sujet p. 6

> Compléter les lignes 9 et 14 du code de la classe Pile pour que la méthode empiler permette d'empiler un élément elt dans une pile et que la méthode depiler permette de dépiler une pile en renvoyant l'élément dépilé. Vous n'écrirez que le code des deux méthodes.

La liste `self.contenu` représente la pile. L'ajout à la fin de la liste empile un élément ; `pop()` retire et renvoie le dernier élément. La gestion de la pile vide est celle fournie dans l'énoncé.

RÉPONSE À RETENIR

```
def empiler(self, elt):
    self.contenu.append(elt)

def depiler(self):
    if self.est_vide():
        return "La pile est vide."
    return self.contenu.pop()
```

Question 7.

Sujet p. 5, 6

> Déterminer le nombre de comparaisons effectuées si on applique l'algorithme précédent à la chaîne de caractères 'tab[2*(i + 4)] - tab[3]'. En déduire le nombre maximum de comparaisons effectuées si on applique l'algorithme précédent à une chaîne de caractères de taille n, en tenant compte des comparaisons de la classe Pile.

Le nombre exact de comparaisons dépend de la façon d'écrire les tests, car l'énoncé ne fournit pas le code de l'algorithme. Pour donner un décompte précis, on adopte ici la convention de la fonction proposée à la question 8 : chaque test de catégorie `caractere in '(['{` ou `caractere in ')]}'` compte pour une comparaison ; on compte également les tests d'égalité ou d'inégalité explicites, dont celui de `est_vide`. Les comparaisons internes éventuellement réalisées par Python pour évaluer `in` ne sont pas décomptées séparément.

La chaîne donnée a 23 caractères : 3 ouvrants, 3 fermants et 17 autres caractères. Pour les 3 ouvrants, seul le test de catégorie « ouvrante » est exécuté. Pour chacun des 20 caractères restants, les tests « ouvrante » puis « fermante » sont exécutés. Cela fait donc $3 + 2 \times 20 = 43$ tests de catégorie.

La chaîne est correctement parenthésée. Pour chacune de ses 3 fermetures, on compte le test de pile vide dans `depiler`, le test distinguant le message de pile vide d'un caractère dépilé, et le test de correspondance des types : $3 \times 3 = 9$ comparaisons. Le test final de pile vide ajoute 1 comparaison. Le total est donc $43 + 9 + 1 = 53$.

Pour une chaîne de taille n, notons c le nombre de fermetures effectivement traitées. Elles nécessitent au moins c ouvertures ; les tests de catégorie sont donc au plus $2n - c$. Les fermetures ajoutent au plus $3c$ comparaisons, et le test final au plus 1. Comme $c \leq \lfloor n/2 \rfloor$, le total est au plus $2n + 2\lfloor n/2 \rfloor + 1$. Cette borne est atteignable, avec une chaîne formée de couples valides et, si n est impair, d'un caractère ordinaire supplémentaire.

> **RÉPONSE À RETENIR**
>
> Avec la convention explicitée ci-dessus, l'exécution sur `'tab[2*(i + 4)] - tab[3]'` effectue **53 comparaisons**. Pour une chaîne de taille n, le maximum est $\mathbf{2n + 2\lfloor n/2 \rfloor + 1}$, soit $3n + 1$ si n est pair et $3n$ si n est impair.

Question 8.

Sujet p. 5, 6, 7

> Écrire une fonction est_bien_parenthesee qui prend en paramètre une chaîne de caractères et qui implémente l'algorithme précédent.

On empile chaque caractère ouvrant. Pour un caractère fermant, on dépile le dernier ouvrant et on vérifie que les deux caractères forment un couple. Le message renvoyé par `depiler` en cas de pile vide permet de détecter une fermeture sans ouvrant correspondant. Enfin, il faut vérifier qu'il ne reste aucun ouvrant dans la pile.

RÉPONSE À RETENIR

```
def est_bien_parenthesee(txt):
    p = Pile()
    couples = {'(': ')', '[': ']', '{': '}'}

    for caractere in txt:
        if caractere in '([{':
            p.empiler(caractere)
        elif caractere in ')]}':
            ouvrante = p.depiler()
            if ouvrante == 'La pile est vide.':
                return False
            if couples[ouvrante] != caractere:
                return False

    return p.est_vide()
```

Exercice 3

8 points

Bases de données relationnelles, requêtes SQL et ordre lexicographique en Python

Partie A

Question 1.

Sujet p. 8

> L'attribut titre peut-il être une clé primaire pour la table Chanson ? Justifier.

Une clé primaire doit identifier sans ambiguïté chaque ligne de la table. Or, dans l'extrait, « Muscle Museum » apparaît deux fois et « Showbiz » apparaît deux fois : ces titres ne permettent donc pas de distinguer toutes les lignes.

RÉPONSE À RETENIR

Non. L'attribut `titre` ne peut pas être une clé primaire de `Chanson`, car plusieurs lignes peuvent avoir le même titre.

Question 2.

Sujet p. 8, 9

> Donner le résultat de la requête : SELECT titre, album FROM Chanson WHERE groupe = 'Guns N' Roses';

La clause `WHERE` sélectionne les lignes dont l'attribut `groupe` vaut exactement « Guns N' Roses ». Dans l'extrait de la table, seule la chanson d'identifiant 7 satisfait cette condition. La projection `SELECT titre, album` ne conserve que ces deux attributs.

RÉPONSE À RETENIR

titre	album
Welcome too the Jungle	Appetite for Destruction

Question 3.

Sujet p. 8, 9

> Écrire une requête SQL permettant d'obtenir tous les titres des chansons de l'album Showbiz dans l'ordre croissant.

On filtre d'abord les lignes sur l'album Showbiz, puis on sélectionne les titres et on les trie par ordre croissant avec `ORDER BY titre`. Dans l'extrait, les titres de cet album sont Sunburn, Muscle Museum et Showbiz. L'ordre lexicographique croissant les place comme indiqué dans le résultat.

```
SELECT titre
FROM Chanson
WHERE album = 'Showbiz'
ORDER BY titre;
```

RÉPONSE À RETENIR

titre
Muscle Museum
Showbiz
Sunburn

Question 4.

Sujet p. 8, 9

Écrire une requête SQL permettant d'ajouter la chanson dont le titre est Megalomania, de l'album Hullabaloo et du groupe Muse.

La table `Chanson` possède aussi un attribut `id`, mais l'énoncé ne précise pas s'il est généré automatiquement. Sous l'hypothèse qu'il l'est, on peut ne préciser que les trois autres colonnes dans l'insertion. Si cet identifiant n'est pas généré par la base, il faut également fournir une valeur disponible pour `id`.

```
INSERT INTO Chanson (titre, album, groupe)
VALUES ('Megalomania', 'Hullabaloo', 'Muse');
```

RÉPONSE À RETENIR

Cette requête ajoute la chanson Megalomania à l'album Hullabaloo du groupe Muse, sous l'hypothèse que la base attribue automatiquement l'identifiant.

Question 5.

Sujet p. 8, 9

La chanson Welcome too the Jungle s'écrit normalement Welcome to the Jungle. Écrire une requête SQL permettant de corriger cette erreur.

Il faut modifier l'attribut `titre` de la ligne concernée. Dans l'extrait, cette ligne a l'identifiant 7 ; le filtre sur cet identifiant cible uniquement la chanson à corriger.

```
UPDATE Chanson
SET titre = 'Welcome to the Jungle'
WHERE id = 7;
```

RÉPONSE À RETENIR

Après cette mise à jour, le titre de la ligne d'identifiant 7 est « Welcome to the Jungle ».

Partie B

Question 6.

Sujet p. 9, 10

Expliquer l'intérêt d'utiliser trois tables Chanson, Album et Groupe au lieu de regrouper toutes les informations dans

> une seule table.

Les trois tables séparent les informations relatives aux chansons, aux albums et aux groupes. Ainsi, les informations d'un album ou d'un groupe ne sont pas répétées dans chaque ligne de chanson. Cette organisation limite les redondances et réduit les risques d'incohérence lors d'une modification ; les identifiants permettent de relier les informations.

> **RÉPONSE À RETENIR**
>
> Utiliser trois tables évite de répéter les mêmes informations sur les albums et les groupes dans de nombreuses lignes de chansons et facilite le maintien de données cohérentes.

Question 7.

Sujet p. 9, 10

> Expliquer le rôle de l'attribut id_album de la table Chanson.

L'attribut `id_album` de `Chanson` contient l'identifiant de l'album auquel appartient la chanson. Il référence l'attribut `id` de la table `Album` : c'est une clé étrangère qui permet de retrouver l'album associé.

> **RÉPONSE À RETENIR**
>
> `Chanson.id_album` est la clé étrangère qui relie chaque chanson à son album dans `Album`.

Question 8.

Sujet p. 9, 10

> Proposer un schéma relationnel pour cette version de la base de données. Spécifier les clés primaires en les soulignant et les clés étrangères en les faisant précéder par le symbole #.

Les identifiants `id` sont les clés primaires des trois tables. La clé étrangère `id_album` de `Chanson` référence `Album.id`, et la clé étrangère `id_groupe` de `Album` référence `Groupe.id`. Les identifiants primaires sont soulignés ci-dessous ; le symbole `#` précède les clés étrangères.

```
Chanson(id, titre, #id_album)
Album(id, titre, année, #id_groupe)
Groupe(id, nom)
```

> **RÉPONSE À RETENIR**
>
> Un schéma relationnel possible est : `Chanson(id, titre, #id_album)`, `Album(id, titre, année, #id_groupe)` et `Groupe(id, nom)`. Les clés étrangères renvoient respectivement à `Album.id` et `Groupe.id`.

Question 9.

Sujet p. 9, 10

> Écrire une requête SQL permettant d'obtenir tous les noms des albums contenant la chanson Showbiz.

On relie les chansons à leurs albums grâce à l'égalité entre `Chanson.id_album` et `Album.id`. On conserve les chansons dont le titre est Showbiz, puis on sélectionne le nom de leur album.

```
SELECT a.titre
FROM Chanson AS c
JOIN Album AS a ON c.id_album = a.id
WHERE c.titre = 'Showbiz';
```


FormaV - document d'entraînement non officiel

Dans l'extrait, les deux chansons portant le titre Showbiz sont associées aux albums Showbiz et Hullabaloo. Sans clause `ORDER BY`, aucun ordre de résultat n'est garanti.

RÉPONSE À RETENIR

titre
Showbiz
Hullabaloo

Question 10.

Sujet p. 9, 10

> Écrire une requête SQL permettant d'obtenir tous les titres avec le nom de l'album des chansons du groupe Muse.

On joint `Chanson` à `Album`, puis `Album` à `Groupe`. Le filtre sur le nom Muse conserve les chansons dont l'album est associé à ce groupe. On sélectionne le titre de la chanson et celui de l'album.

```
SELECT c.titre, a.titre AS album
FROM Chanson AS c
JOIN Album AS a ON c.id_album = a.id
JOIN Groupe AS g ON a.id_groupe = g.id
WHERE g.nom = 'Muse';
```

Voici les lignes correspondantes dans l'ordre des chansons de l'extrait fourni. En l'absence de `ORDER BY`, SQL ne garantit pas cet ordre de présentation.

RÉPONSE À RETENIR

titre	album
Sunburn	Showbiz
Muscle Museum	Showbiz
Showbiz	Showbiz
New Born	Origin of Symmetry
Sing for Absolution	Absolution
Hysteria	Absolution
Muscle Museum	Hullabaloo
Showbiz	Hullabaloo

Question 11.

Sujet p. 10

> Décrire par une phrase ce qu'effectue la requête SQL suivante : SELECT COUNT(*) AS tot FROM Album AS a JOIN Groupe AS g ON a.id_groupe = g.id WHERE g.nom = 'Muse';

La jointure associe chaque album au groupe référencé par `Album.id_groupe`. Le filtre conserve les albums associés à Muse, puis `COUNT(*)` compte les lignes obtenues. Dans l'extrait, les quatre albums associés à Muse sont Showbiz, Origin of Symmetry, Hullabaloo et Absolution.

RÉPONSE À RETENIR

La requête compte le nombre d'albums associés au groupe Muse. Sur les données fournies, elle renvoie `tot = 4`.

tot
4

Partie C

Question 12.

Sujet p. 11

> Compléter les assertions suivantes : assert ordre_lex('', 'a') == True ; assert ordre_lex('b', 'a') == ... ; assert ordre_lex('aaa', 'aaba') == ... .

Lorsque les premiers caractères différents sont rencontrés, leur ordre alphabétique décide du résultat. Ainsi, `b` ne précède pas `a`, tandis que la comparaison de `aaa` avec `aaba` se décide au troisième caractère : `a` précède `b`.

RÉPONSE À RETENIR

```
assert ordre_lex('', 'a') == True
assert ordre_lex('b', 'a') == False
assert ordre_lex('aaa', 'aaba') == True
```

Question 13.

Sujet p. 11, 12

> Recopier et compléter la fonction récursive ordre_lex ci-dessous, qui prend pour paramètres deux chaînes de caractères mot1 et mot2 et qui renvoie True si mot1 précède mot2 dans l'ordre lexicographique.

On compare les premiers caractères. Si l'un est plus petit que l'autre, la comparaison est décidée. S'ils sont égaux, on recommence sur les chaînes privées de leur premier caractère. La récursion se termine dès qu'une chaîne est vide. Avec le cas de base fourni, deux chaînes égales donnent `True` lorsque les deux chaînes s'épuisent ensemble : le code réalise donc aussi cette convention d'égalité.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

```
def ordre_lex(mot1, mot2):
    if mot1 == "":
        return True
    elif mot2 == "":
        return False
    else:
        c1 = mot1[0]
        c2 = mot2[0]
        if c1 < c2:
            return True
        elif c1 > c2:
            return False
        else:
            return ordre_lex(mot1[1:], mot2[1:])
```

À chaque appel récursif, les deux chaînes perdent leur premier caractère : la fonction termine après un nombre fini d'appels. La profondeur de récursion peut toutefois atteindre la longueur de la plus courte chaîne.

Question 14.

Sujet p. 11, 12

> Proposer une version itérative de la fonction ordre_lex.

On parcourt les caractères de même indice tant qu'il en existe dans les deux chaînes. Dès que deux caractères diffèrent, leur comparaison donne le résultat. Si les caractères comparés sont égaux jusqu'à l'épuisement d'une chaîne, c'est la chaîne la plus courte qui précède l'autre. Si les deux chaînes s'épuisent ensemble, on renvoie `True`, conformément au cas de base de la fonction récursive.

L'indice augmente à chaque tour : la boucle termine en au plus autant d'itérations que la longueur de la plus courte chaîne. Cette version évite la profondeur de pile de la version récursive.

RÉPONSE À RETENIR

```
def ordre_lex(mot1, mot2):
    i = 0
    while i < len(mot1) and i < len(mot2):
        if mot1[i] < mot2[i]:
            return True
        elif mot1[i] > mot2[i]:
            return False
        i = i + 1
    return i == len(mot1)
```

Par exemple, pour `mot1 = "aaa"` et `mot2 = "aaba"`, les deux premières paires de caractères sont égales, puis `a < b` au troisième caractère : la fonction renvoie `True`.

Sujet : https://www.sujetdebac.fr/annales-pdf/2024/spe-numerique-informatique-2024-asie-2-sujet-officiel.pdf

© FormaV EI • Proposition de correction, distincte d'un corrigé officiel.

FormaV - document d'entraînement non 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.