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 • Polynésie • Jour 2

Code sujet **25-NSIJ2PO1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Programmation orientée objet et gestion du jeu de cartes « 6 qui prend »
8 réponses • sujet p. 2–5

Exercice 2 — Bases de données relationnelles, requêtes SQL et programmation orientée objet autour des champignons
11 réponses • sujet p. 6–9

Exercice 3 — Tableaux, dictionnaires, piles, files, graphes et recherche du plus court chemin entre des mots
11 réponses • sujet p. 10–14

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

Programmation orientée objet et gestion du jeu de cartes « 6 qui prend »

Partie A : La classe Carte

Question 1.

Sujet p. 2

> Écrire le code du constructeur de la classe Carte. Il prend en paramètre une valeur entière comprise entre 1 et 104, l'affecte à l'attribut valeur et initialise l'attribut TdB à l'aide de la méthode calcul_TdB.

Le constructeur mémorise d'abord la valeur reçue, puis initialise le nombre de têtes de bœuf en appelant la méthode de l'objet. Le paramètre `self` est transmis automatiquement à une méthode d'instance.

```
class Carte:
    def __init__(self, valeur):
        self.valeur = valeur
        self.TdB = self.calcul_TdB()
```

On suppose que `valeur` est bien un entier compris entre 1 et 104, comme l'impose l'énoncé.

> **RÉPONSE À RETENIR**
>
> Le constructeur affecte la valeur à `self.valeur` et le résultat de `self.calcul_TdB()` à `self.TdB`.

Question 2.

Sujet p. 2

> Écrire le code de la méthode calcul_TdB de la classe Carte afin de calculer le nombre de têtes de bœuf d'une carte. Les règles se cumulent : la carte reçoit 5 têtes de bœuf si sa valeur est divisible par 11, 3 si elle se termine par 0 et 2 si elle se termine par 5. Si aucune de ces conditions n'est satisfaite, elle reçoit 1 tête de bœuf. Par exemple, la carte 55 en reçoit 7.

Les règles se cumulent : on teste donc chacune des trois conditions séparément et on ajoute sa pénalité au total. Si aucune condition ne s'applique, le total reste nul ; on renvoie alors 1.

```
    def calcul_TdB(self):
        total = 0
        if self.valeur % 11 == 0:
            total += 5
        if self.valeur % 10 == 0:
            total += 3
        if self.valeur % 10 == 5:
            total += 2
        if total == 0:
            total = 1
        return total
```

Les tests de terminaison utilisent le reste de la division par 10 : un reste nul indique une terminaison par 0, et un reste égal à 5 indique une terminaison par 5.

> **RÉPONSE À RETENIR**
>
> Pour la carte 55, on a `55 % 11 == 0` et `55 % 10 == 5`. La méthode renvoie donc `5 + 2 = 7`.

FormaV - document d'entraînement non officiel

Question 3.

Sujet p. 2

> Écrire le code de la méthode est_superieure_a. Elle prend en paramètre une carte autre de type Carte et renvoie True si la valeur de la carte considérée est strictement supérieure à celle de autre.

La valeur de la carte considérée est `self.valeur` et celle de la carte reçue en paramètre est `autre.valeur`. L'opérateur > réalise la comparaison stricte demandée.

```
    def est_superieure_a(self, autre):
        return self.valeur > autre.valeur
```

> **RÉPONSE À RETENIR**
>
> La méthode renvoie `True` si et seulement si `self.valeur` est strictement supérieure à `autre.valeur`.

Partie B : La classe Paquet

Question 4.

Sujet p. 3

> Compléter le code de la classe Paquet en recopiant les lignes 6, 7 et 10 : afficher doit afficher la valeur de toutes les cartes du paquet et ajouter_carte doit ajouter au contenu du paquet la carte reçue en paramètre.

La méthode `afficher` parcourt le contenu du paquet et affiche la valeur de chaque carte. La méthode `ajouter_carte` ajoute la carte à la liste `contenu`.

```
    def afficher(self):
        for carte in self.contenu:
            print(carte.valeur)

    def ajouter_carte(self, carte):
        self.contenu.append(carte)
```

> **RÉPONSE À RETENIR**
>
> Les lignes à recopier sont le parcours suivi de `print(carte.valeur)` pour `afficher`, puis `self.contenu.append(carte)` pour `ajouter_carte`.

Question 5.

Sujet p. 3

> Écrire le code de la méthode nombre_TdB de la classe Paquet, qui renvoie le nombre total de têtes de bœuf contenues sur les cartes du paquet.

On additionne les valeurs `TdB` de toutes les cartes du paquet. Pour un paquet vide, la boucle ne s'exécute pas et le total renvoyé est 0.

```
    def nombre_TdB(self):
        total = 0
        for carte in self.contenu:
            total += carte.TdB
        return total
```

RÉPONSE À RETENIR

La méthode renvoie la somme des nombres de têtes de bœuf de toutes les cartes contenues dans le paquet.

Question 6.

Sujet p. 3

> Écrire le code de la méthode distribuer de la classe Paquet. Elle prend en paramètre un entier naturel nbr, le nombre de joueurs à servir, au maximum 10, et renvoie une liste de nbr paquets initialement vides. Chaque joueur reçoit 10 cartes : la première carte du paquet est donnée au premier joueur, la deuxième au deuxième, et ainsi de suite. Chaque paquet reçoit 10 cartes retirées du paquet initial.

On crée une liste contenant une instance distincte de `Paquet` vide par joueur. On effectue ensuite 10 tours : lors de chaque tour, on retire la première carte du paquet initial et on l'ajoute au paquet du joueur suivant. `pop(0)` renvoie cette première carte tout en la supprimant du paquet initial.

```
    def distribuer(self, nbr):
        paquets = [Paquet([]) for i in range(nbr)]
        for tour in range(10):
            for joueur in range(nbr):
                carte = self.contenu.pop(0)
                paquets[joueur].ajouter_carte(carte)
        return paquets
```

Après chaque tour complet, chaque paquet distribué a reçu une carte de plus et le paquet initial a perdu `nbr` cartes. La boucle termine après 10 tours. Avec au plus 10 joueurs et 104 cartes au départ, il y a toujours assez de cartes. Si `nbr` vaut 0, la liste renvoyée est vide et aucune carte n'est retirée.

RÉPONSE À RETENIR

La méthode renvoie `nbr` paquets contenant chacun 10 cartes. Les cartes sont attribuées dans l'ordre indiqué : la première au premier joueur, la deuxième au deuxième, puis le tour recommence. Pour deux joueurs, 20 cartes sont retirées du paquet initial.

Partie C : La classe Joueur et modélisation du plateau de jeu -> classe Plateau

Question 7.

Sujet p. 4

> Écrire une commande Python permettant d'instancier le joueur nommé Joueur 1, ayant pour paquet de cartes L[0], et d'affecter l'objet créé à la variable J1.

Le constructeur de `Joueur` reçoit le nom puis le paquet de cartes. Le nom demandé est la chaîne `Joueur 1`, et le paquet est `L[0]`.

```
J1 = Joueur('Joueur 1', L[0])
```

RÉPONSE À RETENIR

Cette commande instancie le joueur demandé et affecte l'objet à la variable `J1`.

FormaV - document d'entraînement non officiel

Question 8.

Sujet p. 4, 5

> Compléter le script en recopiant les lignes 3, 7 et 9 : créer les 104 cartes du jeu initial grâce à une liste par compréhension, mélanger cette liste, instancier le paquet jeu_initial avec cette liste, puis distribuer 10 cartes aux deux joueurs nommés Ordi et J1. Les lignes données instancient Ordi avec distri[0] et J1 avec distri[1].

La borne supérieure de `range` est exclue : `range(1, 105)` produit donc les entiers de 1 à 104 inclus. On crée une carte pour chaque valeur, on mélange la liste, puis on l'utilise pour construire le paquet. Enfin, `distribuer(2)` crée les deux mains dans l'ordre attendu par les lignes fournies.

```
from random import *

jeu = [Carte(i) for i in range(1, 105)]
shuffle(jeu)
jeu_initial = Paquet(jeu)
distri = jeu_initial.distribuer(2)
Ordi = Joueur("Ordi", distri[0])
J1 = Joueur("J1", distri[1])
```

RÉPONSE À RETENIR

`jeu` contient 104 cartes. Après la distribution, `distri[0]` est la main d'`Ordi` et `distri[1]` celle de `J1` ; chacune contient 10 cartes. Il reste 84 cartes dans le paquet initial. Leur identité dépend du mélange aléatoire.

FormaV - document d'entraînement non officiel

Exercice 2

6 points

Bases de données relationnelles, requêtes SQL et programmation orientée objet autour des champignons

Partie A

Question 1.

Sujet p. 6

> Mathilde a trouvé un champignon avec des lamelles et de couleur orange. Écrire la requête permettant d'obtenir la liste des noms des champignons possibles pour sa trouvaille, à partir de la table champignon.

La requête sélectionne l'attribut `nom` et conserve les lignes dont l'attribut `lamelle` indique la présence de lamelles et dont la couleur est orange. Les deux critères doivent être vérifiés, d'où `AND`.

```
SELECT nom
FROM champignon
WHERE lamelle = 'oui' AND couleur = 'orange';
```

Dans l'extrait de la table `champignon` (page 6), une seule ligne satisfait les deux conditions.

nom
amanite des Césars

RÉPONSE À RETENIR

Dans l'extrait fourni, la requête renvoie **amanite des Césars**.

Question 2.

Sujet p. 6

> Romain a trouvé un champignon sans pied et dont le chapeau mesure 15 cm. Écrire la requête permettant d'obtenir la liste des noms des champignons possibles pour son spécimen, à partir de la table champignon.

Dans l'extrait, l'absence de pied est indiquée par `pied_min = 0` et `pied_max = 0`. Pour que le chapeau puisse mesurer 15 cm, cette mesure doit être comprise entre les bornes minimale et maximale, bornes incluses.

```
SELECT nom
FROM champignon
WHERE pied_min = 0 AND pied_max = 0
  AND chapeau_min <= 15 AND chapeau_max >= 15;
```

Le seul champignon sans pied de l'extrait est le champignon noir, dont le chapeau mesure entre 2 et 10 cm. La mesure 15 cm n'appartient pas à cet intervalle.

nom

RÉPONSE À RETENIR

Dans l'extrait fourni, la requête ne renvoie aucune ligne.

Question 3.

Sujet p. 6, 7

> Donner la clé étrangère de la table champignon faisant référence à la clé primaire id de la table ordre.

L'attribut de la table `champignon` qui indique l'ordre auquel appartient un champignon est `id_ordre`. Il référence l'identifiant de la table `ordre`.

> **RÉPONSE À RETENIR**
>
> La clé étrangère est `champignon.id_ordre` ; elle référence la clé primaire `ordre.id`.

Question 4.

Sujet p. 6, 7

> Écrire la requête permettant d'obtenir la liste des noms des champignons appartenant à la classe des agaricomycètes.

La classe est enregistrée dans la table `ordre`, alors que le nom du champignon est dans la table `champignon`. On joint donc ces tables par `id_ordre` et `id`, puis on filtre sur la classe demandée.

```
SELECT c.nom
FROM champignon AS c
JOIN ordre AS o ON c.id_ordre = o.id
WHERE o.classe = 'agaricomycètes';
```

En appliquant cette requête aux lignes visibles dans l'extrait de la page 6 et à la table `ordre` de la page 7, on obtient les noms ci-dessous. La requête ne comportant pas de `ORDER BY`, leur ordre d'affichage n'est pas garanti.

nom
champignon de Paris
coprin chevelu
bolet à pied rouge
amanite des Césars

> **RÉPONSE À RETENIR**
>
> Pour les lignes de l'extrait, les noms obtenus sont **champignon de Paris**, **coprin chevelu**, **bolet à pied rouge** et **amanite des Césars**.

Question 5.

Sujet p. 6, 7

> Écrire une requête permettant d'ajouter dans la table champignon le champignon amanite solitaire avec l'id 56. C'est un champignon blanc, de l'ordre des amanitales, qui possède des lamelles ; son chapeau mesure entre 6 et 20 cm et son pied entre 4 et 10 cm.

Dans la table `ordre` de la page 7, l'ordre `amanitales` a l'identifiant 4. Les autres valeurs viennent des caractéristiques données dans l'énoncé. À ce stade, on utilise les colonnes de la table `champignon` présentées page 6, avant l'ajout de l'attribut de toxicité.

FormaV - document d'entraînement non officiel

```
INSERT INTO champignon
    (id, nom, id_ordre, lamelle, couleur,
     chapeau_min, chapeau_max, pied_min, pied_max)
VALUES
    (56, 'amanite solitaire', 4, 'oui', 'blanc', 6, 20, 4, 10);
```

Les tailles minimale et maximale du chapeau sont respectivement 6 et 20 cm ; celles du pied sont 4 et 10 cm.

> **RÉPONSE À RETENIR**
>
> Cette instruction ajoute dans `champignon` l'amanite solitaire avec l'identifiant 56 et les caractéristiques indiquées.

Question 6.

Sujet p. 6, 7

> Écrire le schéma relationnel de la base de données contenant les trois tables champignon, ordre et toxicite, en soulignant les clés primaires et en faisant précéder les clés étrangères du caractère #. La table champignon comporte les attributs id, nom, id_ordre, lamelle, couleur, chapeau_min, chapeau_max, pied_min, pied_max et reçoit l'attribut id_toxicite ; la table ordre comporte id, nom et classe ; la table toxicite comporte id_tox, type et effets.

On ajoute à `champignon` l'attribut `id_toxicite`, qui référence `toxicite.id_tox`. L'attribut `id_ordre` référence `ordre.id`. Les clés primaires sont signalées en gras dans la représentation ci-dessous.

- `champignon(`**`id`**`, nom, #id_ordre, lamelle, couleur, chapeau_min, chapeau_max, pied_min, pied_max, #id_toxicite)`
- `ordre(`**`id`**`, nom, classe)`
- `toxicite(`**`id_tox`**`, type, effets)`

Le signe `#` devant `id_ordre` et `id_toxicite` indique qu'il s'agit de clés étrangères.

> **RÉPONSE À RETENIR**
>
> Schéma relationnel : `champignon(`**`id`**`, nom, #id_ordre, lamelle, couleur, chapeau_min, chapeau_max, pied_min, pied_max, #id_toxicite)` ; `ordre(`**`id`**`, nom, classe)` ; `toxicite(`**`id_tox`**`, type, effets)`. Les clés étrangères référencent respectivement `ordre.id` et `toxicite.id_tox`.

Question 7.

Sujet p. 7, 8

> Le champignon de nom amanite citrine est très toxique car son ingestion peut entraîner la mort. Écrire la requête permettant de corriger cette erreur dans la table champignon.

Dans la table `toxicite` de la page 7, la valeur 1 de `id_tox` correspond au type « très toxique ». Il faut donc affecter 1 à `id_toxicite` pour la ligne de l'amanite citrine.

```
UPDATE champignon
SET id_toxicite = 1
WHERE nom = 'amanite citrine';
```

La condition cible les lignes portant ce nom ; si plusieurs lignes ont ce nom, elles seront toutes modifiées.

RÉPONSE À RETENIR

La requête associe l'amanite citrine à la toxicité d'identifiant 1, qui correspond à « très toxique ».

Question 8.

Sujet p. 7, 8

Écrire une requête permettant d'obtenir les noms des champignons de l'ordre des amanitales et de toxicité très toxique.

La table `ordre` permet de filtrer les champignons sur l'ordre `amanitales`, et la table `toxicite` permet de filtrer sur le type « très toxique ». Il faut donc joindre ces deux tables à `champignon`.

```
SELECT c.nom
FROM champignon AS c
JOIN ordre AS o ON c.id_ordre = o.id
JOIN toxicite AS t ON c.id_toxicite = t.id_tox
WHERE o.nom = 'amanitales'
  AND t.type = 'très toxique';
```

L'extrait de la table `champignon` fourni page 6 ne contient pas l'attribut `id_toxicite` ni les associations entre champignons et toxicités. Le contenu de la table `toxicite` donne les correspondances entre identifiants et types, mais ne permet pas de connaître le résultat de cette requête.

RÉPONSE À RETENIR

La requête est donnée ci-dessus. Le résultat effectif ne peut pas être établi à partir des extraits fournis, car ils ne précisent pas la toxicité associée à chaque champignon.

Partie B

Question 9.

Sujet p. 8

Recopier et compléter le programme qui parcourt liste_champi et affiche le ou les noms des champignons que l'on trouve lors de la saison été.

Chaque élément `e` de `liste_champi` est un objet de la classe `Champignon`. La saison est accessible avec `e.saison` et le nom avec `e.nom`. On affiche donc le nom lorsque la saison vaut `'été'`.

```
for e in liste_champi:
    if e.saison == 'été':
        print(e.nom)
```

RÉPONSE À RETENIR

Le programme affiche le nom de chaque champignon trouvé en été.

Question 10.

Sujet p. 8, 9

Le programme donné recherche le champignon nommé Lactaire délicieux puis teste si c.cuisson est égal à la chaîne 'feu moyen'. Il renvoie False alors que le lactaire délicieux se cuisine bien à feu moyen. Expliquer pourquoi

> Quentin n'obtient pas le résultat attendu, alors qu'il n'a fait aucune erreur dans sa liste de champignons.

L'attribut `c.cuisson` contient la description complète, par exemple `'12 minutes à feu moyen'`. Le test compare cette chaîne entière à `'feu moyen'` ; les chaînes ne sont pas égales, même si les mots « feu moyen » apparaissent dans la description. C'est pourquoi, dans le contexte d'une fonction, le test renvoie `False`.

Précision : dans le code reproduit page 9, `return` apparaît sans définition de fonction. Pris littéralement comme code exécuté au niveau principal, ce serait une erreur de syntaxe en Python. L'explication du résultat `False` suppose que ces lignes sont dans une fonction, comme le suggère l'énoncé.

RÉPONSE À RETENIR

La comparaison échoue parce que `'12 minutes à feu moyen'` est différent de `'feu moyen'` : le programme compare la chaîne entière, pas seulement une partie de celle-ci.

Question 11.

Sujet p. 9

> Corriger le programme de la question précédente en utilisant la fonction recherche_textuelle, qui prend en paramètres deux chaînes de caractères texte et mot et renvoie True si mot est un mot appartenant à texte, et False sinon.

La fonction `recherche_textuelle` recherche un mot. Il faut donc vérifier séparément que les mots `'feu'` et `'moyen'` figurent dans la description de cuisson, puis combiner les deux résultats avec `and`. Voici une version complète qui renvoie `False` si le champignon recherché n'est pas dans la liste.

```
def peut_cuire_a_feu_moyen(liste_champi):
    for c in liste_champi:
        if c.nom == 'Lactaire délicieux':
            return (recherche_textuelle(c.cuisson, 'feu')
                    and recherche_textuelle(c.cuisson, 'moyen'))
    return False
```

Pour la description donnée, `'12 minutes à feu moyen'`, les deux mots recherchés sont présents ; le test renvoie donc `True`.

RÉPONSE À RETENIR

La fonction corrigée teste la présence des deux mots dans la chaîne de cuisson. Pour le lactaire délicieux de l'énoncé, elle renvoie `True`.

Exercice 3

8 points

Tableaux, dictionnaires, piles, files, graphes et recherche du plus court chemin entre des mots

Partie A

Question 1.

Sujet p. 10

> Décrire le fonctionnement d'une pile.

Une pile fonctionne selon le principe **LIFO** (« dernier entré, premier sorti »). On ajoute un élément au sommet avec une opération d'empilement et on retire l'élément situé au sommet avec une opération de dépilement. Ainsi, le dernier élément empilé est le premier que l'on récupère.

> **RÉPONSE À RETENIR**
>
> Une pile est une structure LIFO : le dernier élément ajouté est le premier retiré.

Question 2.

Sujet p. 10

> Décrire le fonctionnement d'une file.

Une file fonctionne selon le principe **FIFO** (« premier entré, premier sorti »). On ajoute les éléments à la queue de la file et on retire ceux qui se trouvent à sa tête. Ainsi, le premier élément ajouté est le premier récupéré.

> **RÉPONSE À RETENIR**
>
> Une file est une structure FIFO : le premier élément ajouté est le premier retiré.

Question 3.

Sujet p. 10

> Dans le graphe considéré, les sommets sont les mots et les arêtes relient deux sommets si les mots sont voisins. Expliquer pourquoi un graphe non orienté est adapté à la situation.

La relation « être voisin » est symétrique : si deux mots ne diffèrent que d'une lettre, le premier est voisin du second et le second est voisin du premier. Il n'y a donc pas de sens de parcours privilégié entre deux mots voisins. Un graphe non orienté, dont les arêtes peuvent être parcourues dans les deux sens, convient.

> **RÉPONSE À RETENIR**
>
> Un graphe non orienté convient car la relation de voisinage est symétrique.

Question 4.

Sujet p. 10

> Dessiner le graphe dont la liste de mots est : gars, mars, mors, ours et purs.

Deux sommets sont reliés si les mots correspondants ont une distance égale à 1. On obtient les voisinages suivants : `gars` est voisin de `mars` ; `mars` est voisin de `gars` et de `mors` ; `mors` est voisin de `mars` et d'`ours` ; `ours` est voisin de `mors` et de `purs`. Les autres paires ne sont pas voisines.

FormaV - document d'entraînement non officiel

Graphe des mots voisins



Les arêtes sont non orientées ; aucune autre paire de mots de la liste n'est voisine.

RÉPONSE À RETENIR

Le graphe est le chemin `gars` — `mars` — `mors` — `ours` — `purs`.

Partie B

Question 5.

Sujet p. 11

> Recopier et compléter la fonction chaine_vers_tab(mot). Elle prend en paramètre une chaîne de quatre caractères et renvoie un tableau Python dont les éléments sont les caractères de cette chaîne. Ainsi, chaine_vers_tab('ours') renvoie ['o', 'u', 'r', 's'].

On initialise une liste vide, puis on parcourt les caractères de `mot` dans leur ordre et on les ajoute à la liste.

```
def chaine_vers_tab(mot):
    tab_lettres = []
    for lettre in mot:
        tab_lettres.append(lettre)
    return tab_lettres
```

Trace pour `chaine_vers_tab('ours')` : la liste passe successivement par `['o']`, `['o', 'u']`, `['o', 'u', 'r']`, puis `['o', 'u', 'r', 's']`.

RÉPONSE À RETENIR

`chaine_vers_tab('ours')` renvoie `['o', 'u', 'r', 's']`.

Question 6.

Sujet p. 11

> Expliquer pourquoi la fonction distance(mot1, mot2) donnée renvoie effectivement la distance entre deux mots mot1 et mot2, qui sont des chaînes de quatre caractères.

`tab` contient d'abord les lettres de `mot1`. Pour chaque lettre de `mot2`, la fonction en retire une occurrence de `tab` si elle s'y trouve. Elle apparie donc les lettres communes en tenant compte de leurs répétitions, sans tenir compte de leur position. À la fin, `tab` contient les occurrences de lettres de `mot1` qui n'ont pas pu être appariées.

Comme les deux mots ont quatre lettres, le nombre d'occurrences restantes est le nombre minimal de lettres qu'il faut remplacer pour obtenir les lettres de l'autre mot : chaque remplacement peut corriger une occurrence en excès, et remplacer chacune de ces occurrences par une lettre manquante suffit. La fonction renvoie donc bien la distance définie dans l'énoncé.

Exemple avec `mot1 = 'mars'` et `mot2 = 'mors'` : on part de `['m', 'a', 'r', 's']`. Le parcours de `'mors'` retire `m`, ne trouve pas `o`, puis retire `r` et `s`. Il reste `['a']`, donc la distance renvoyée est 1.

> **RÉPONSE À RETENIR**
>
> La fonction renvoie le nombre d'occurrences de lettres de `mot1` qui ne peuvent pas être appariées avec celles de `mot2`. Pour `'mars'` et `'mors'`, ce nombre vaut 1.

Question 7.

Sujet p. 10, 11

> Recopier et compléter la fonction renvoie_voisins(mot), qui renvoie un tableau contenant les mots de TAB_MOTS voisins de mot, une chaîne de quatre caractères passée en paramètre. Deux mots sont voisins si et seulement si leur distance vaut 1.

On parcourt les mots de `TAB_MOTS` et on ajoute au tableau résultat ceux dont la distance à `mot` vaut exactement 1. Le test exclut notamment le mot lui-même, dont la distance à lui-même vaut 0.

```
def renvoie_voisins(mot):
    tab_voisins = []
    for voisin_possible in TAB_MOTS:
        if distance(mot, voisin_possible) == 1:
            tab_voisins.append(voisin_possible)
    return tab_voisins
```

Pour les mots de la partie A, l'appel `renvoie_voisins('mars')` retient `'gars'` et `'mors'`, puis renvoie `['gars', 'mors']`.

> **RÉPONSE À RETENIR**
>
> La fonction renvoie exactement les mots de `TAB_MOTS` dont la distance à `mot` vaut 1 ; pour `'mars'`, le résultat fourni est `['gars', 'mors']`.

Partie C

Question 8.

Sujet p. 12, 13

> Poursuivre le déroulement pas à pas de la fonction dic_parent avec mot_depart = 'mars' et mot_final = 'ours', sur le

FormaV - document d'entraînement non officiel

> modèle des étapes déjà détaillées. Décrire chaque tour de boucle jusqu'à l'issue de la fonction.

Après les deux tours déjà détaillés, `parent` vaut `{'mars': None, 'gars': 'mars', 'mors': 'mars'}` et la file contient uniquement `'mors'`.

1. **Troisième tour :** on défile `'mors'`. Ses voisins sont `'mars'` et `'ours'`. `'mars'` est déjà dans `parent`, donc on ne le traite pas à nouveau. `'ours'` n'y est pas encore : on ajoute `parent['ours'] = 'mors'` et on enfile `'ours'`. À la fin du tour, `parent` vaut `{'mars': None, 'gars': 'mars', 'mors': 'mars', 'ours': 'mors'}` et la file contient uniquement `'ours'`.
2. **Quatrième tour :** la condition de boucle est encore vraie, car la file n'est pas vide et `mot` vaut encore `'mors'`, qui n'est pas le mot final. On défile donc `'ours'`. Ses voisins, dans l'ordre fourni, sont `'mors'` et `'purs'`. `'mors'` est déjà dans `parent`. `'purs'` n'y est pas encore : on ajoute `parent['purs'] = 'ours'` et on enfile `'purs'`. La file contient alors `'purs'`. La boucle s'arrête ensuite, car `mot` vaut maintenant `'ours'`, le mot final. `'purs'` n'est donc pas défilé.

La fonction renvoie le dictionnaire `{'mars': None, 'gars': 'mars', 'mors': 'mars', 'ours': 'mors', 'purs': 'ours'}`.

> **RÉPONSE À RETENIR**
>
> Le dictionnaire renvoyé est `{'mars': None, 'gars': 'mars', 'mors': 'mars', 'ours': 'mors', 'purs': 'ours'}`. Il indique notamment que le chemin retrouvé jusqu'à `'ours'` est `'mars'`, `'mors'`, `'ours'`.

Question 9.

Sujet p. 13

> Recopier et compléter la fonction renvoie_pile(parent, mot_final). Elle prend en paramètres parent, un dictionnaire obtenu grâce à dic_parent, et mot_final, le mot final. Elle renvoie une pile dont le premier élément empilé est le mot final et dont le sommet est le mot de départ.

On part du mot final et on remonte dans le dictionnaire des parents. À chaque étape, on empile le mot courant, puis on le remplace par son parent. On s'arrête lorsque le parent vaut `None`, après avoir empilé le mot de départ. Ainsi, le premier mot empilé est le mot final et le sommet de la pile est le mot de départ.

```
def renvoie_pile(parent, mot_final):
    ma_pile = Pile()
    mot = mot_final
    while mot is not None:
        ma_pile.empiler(mot)
        mot = parent[mot]
    return ma_pile
```

Avec le dictionnaire de la question 8 et `mot_final = 'ours'`, on empile successivement `'ours'`, `'mors'`, puis `'mars'`. Le sommet est alors `'mars'`.

> **RÉPONSE À RETENIR**
>
> La pile est remplie dans l'ordre `'ours'`, `'mors'`, `'mars'` ; en la dépilant, on récupère le chemin dans l'ordre `'mars'`, `'mors'`, `'ours'`.

Question 10.

Sujet p. 13, 14

> Recopier et compléter la fonction construit_chemin. Son paramètre est une pile de mots obtenue grâce à la fonction renvoie_pile ; elle renvoie un tableau dont les éléments sont les mots du chemin recherché dans le bon ordre.

Le sommet de la pile contient le mot de départ. Il faut donc dépiler les mots un à un et les ajouter au tableau dans cet ordre. La boucle s'arrête lorsque la pile est vide.

```
def construit_chemin(ma_pile):
    tab = []
    while not ma_pile.est_vide():
        mot = ma_pile.depiler()
        tab.append(mot)
    return tab
```

Exemple : si la pile a été remplie avec `'ours'`, `'mors'`, puis `'mars'`, les dépilements donnent successivement `'mars'`, `'mors'` et `'ours'`. Le tableau renvoyé est donc `['mars', 'mors', 'ours']`.

> **RÉPONSE À RETENIR**
>
> La fonction renvoie les mots du chemin dans l'ordre allant du mot de départ au mot final ; dans l'exemple, elle renvoie `['mars', 'mors', 'ours']`.

Question 11.

Sujet p. 11, 12, 13, 14

> Coder, en utilisant les fonctions précédentes, la fonction chercher_chemin. Elle prend en paramètres mot_depart, le mot de départ, et mot_final, le mot à atteindre, et renvoie un tableau dont les éléments sont les mots qui constituent le chemin du mot de départ jusqu'au mot final.

On construit d'abord le dictionnaire des parents avec `dic_parent`, puis la pile avec `renvoie_pile`, et enfin le tableau du chemin avec `construit_chemin`.

```
def chercher_chemin(mot_depart, mot_final):
    parent = dic_parent(mot_depart, mot_final)
    ma_pile = renvoie_pile(parent, mot_final)
    return construit_chemin(ma_pile)
```

Pour les données de l'exemple, `chercher_chemin('mars', 'ours')` renvoie `['mars', 'mors', 'ours']`. Si le départ et l'arrivée sont le même mot, le dictionnaire associe ce mot à `None` et la fonction renvoie le tableau contenant ce seul mot.

`dic_parent` effectue un parcours en largeur grâce à la file : les mots sont examinés par distance croissante au mot de départ. Chaque mot n'est ajouté qu'une fois au dictionnaire des parents, ce qui permet de reconstruire un chemin de longueur minimale et évite les revisites. Le parcours termine car la liste de mots est finie et chaque mot est ajouté au plus une fois. Si `n` est le nombre de mots, chaque recherche de voisins parcourt `TAB_MOTS` et compare des mots de quatre lettres : le temps maximal est donc de l'ordre de n^2. La file, le dictionnaire et le tableau du chemin utilisent un espace de l'ordre de `n`.

> **RÉPONSE À RETENIR**
>
> La fonction complète renvoie, pour `chercher_chemin('mars', 'ours')`, le chemin le plus court `['mars', 'mors', 'ours']`.

FormaV - document d'entraînement non officiel

© 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.