Ce document a été mis en ligne par l'organisme FormaV®.

Toute reproduction, représentation ou diffusion, même partielle, sans autorisation préalable, est strictement interdite.

Pour consulter tous nos sujets d'annales et corrigés du Bac Général Maths + NSI, veuillez visiter :
bac-maths-nsi.fr/annales-sujets-et-corriges

Asie - juin 2026 - sujet 2 (corrigé)

Exercice 1 (Programmation orientée objet et structures linéaires)

1. Il s'agit de l'élément d'indice 5
2. La liste devient `tab = [5, 3, 8, 1, 0, 2, 7, 6, 4]`. On a donc la grille suivante :

5	3	8
1	0	2
7	6	4

3. On peut tester si la liste est triée : `tab == [0, 1, 2, 3, 4, 5, 6, 7, 8]`
4. La méthode suivante convient :

```
def est_gagnant(self):
    return self.tab == [x for x in range(9)]
```

5. Le code suivant convient :

```
def indice(self, numero):
    assert type(numero) == int, 'numero doit être entier'
    assert 0 <= numero <= 8, 'numero de case non valide'
    i = 0
    while self.tab[i] != numero :
        i = i + 1
    return i
```

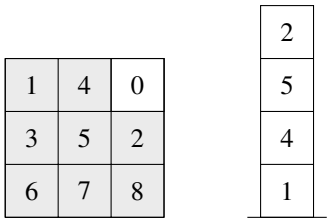
6. Le code suivant convient :

```
def jouer(self, numero):
    if self.est_possible(numero) :
        i = self.indice(numero)
        j = self.indice(0)
        self.tab[j] = numero
        self.tab[i] = 0
```

7. Le code suivant convient :

```
import random
def melanger(self, n):
    precedent = None
    i = 0
    while i < n :
        possibilites = self.coups_possibles()
        choix = random.choice(possibilites)
        if choix != precedent :
            self.jouer(choix)
            precedent = choix
            i = i + 1
```

8. Après le mélange et le coup du joueur la grille et la pile sont dans les états suivants :



Donc lors de la résolution automatique, on joue successivement :

- le numéro 2, la pile restante est (bas) `[1, 4, 5]` (haut) ;
- le numéro 5, la pile restante est (bas) `[1, 4]` (haut) ;
- le numéro 4, la pile restante est (bas) `[1]` (haut) ;
- le numéro 1, la pile est vide.

9. La fonction suivante convient :

```
def resoudre(self):
    self.mode_resolution = True
    while not self.pile.est_vide():
        numero = self.pile.depiler()
        print(numero)
        self.jouer(numero)
```

10. Si l'on oublie de passer l'attribut `mode_resolution` à `True` le comportement sera le suivant :
    - appel de `self.resolution()`
    - tant que pile est non vide :
      - dépilement d'un numéro
      - affichage de ce numéro
      - appel de `self.jouer(numero)`.

    Or dans ce dernier appel, `numero` est ré-empilé. La pile ne se videra donc jamais et la méthode ne terminera pas

11. La modification suivante convient :

```
def jouer(self, numero):
    if self.est_possible(numero) :
        i = self.indice(numero)
        j = self.indice(0)
        self.tab[j] = numero
        self.tab[i] = 0
        if not self.mode_resolution:
            if self.pile.est_vide():
                self.pile.empiler(numero)
            else:
                precedent = self.pile.depiler()
                if precedent != numero:
                    self.pile.empiler(precedent)
                    self.pile.empiler(numero)
```

Exercice 2 (Bases de données et graphes)

Partie A

1. L'attribut `id_pers` est une clé étrangère de `participation` et fait référence à `personne.id_pers`.
2. Une partie pouvant être jouée par plusieurs personnes, l'identifiant `id_partie` va apparaître à plusieurs reprises dans cette table. Il ne peut donc pas être une clé primaire de `participation` (en revanche, on peut utiliser le couple `(id_partie, id_pers)` comme clé primaire).
3. La requête suivante convient :

```
INSERT INTO personne VALUES(42, 'theorie', '2022-12-14')
```

4. La requête suivante convient :

```
SELECT id_partie FROM participation
JOIN personne ON participation.id_personne = personne.id_personne
WHERE personne.pseudo_pers = 'test'
```

5. Les requêtes suivantes conviennent :

```
DELETE FROM participation WHERE id_personne = 8
DELETE FROM personne WHERE id_personne = 8
```

Partie B

6. Le code suivant convient :

```
def indice(lettre, ordre):
    for i in range(len(ordre)):
        if ordre[i] == lettre:
            return i
```

7. Le code suivant convient :

```
def comparer(mot1, mot2, ordre):
    i = 0
    while i < len(mot1) and i < len(mot2):
        i1 = indice(mot1[i], ordre)
        i2 = indice(mot2[i], ordre)
        if i1 < i2:
            return True
        elif i1 > i2:
            return False
        i += 1
    return len(mot1) < len(mot2)
```

8. La fonction suivante convient :

```
def premiere_diff(mot1, mot2):
    i = 0
    while i < len(mot1) and i < len(mot2):
        if mot1[i] != mot2[i]:
            return i
        i += 1
    return i
```

9. On a le dictionnaire suivant :

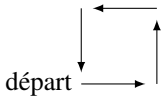
```
adj = {"a": ["i"],
       "e": ["a"],
       "o": ["u"],
       "u": ["a", "y"],
       "y": ["i", "a"]}
```

10. À chaque appel, on visite les voisins non encore visités d'un sommet et ce de façon récursive. Il s'agit donc d'un parcours en profondeur.
11. La fonction suivante convient :

```
def trier(mots):
    adj = dico_adj(mots)
    tri = []
    deja_vus = []
    for voyelle in "aeiouy":
        if voyelle not in deja_vus:
            deja_vus.append(voyelle)
            parcours(adj, voyelle, deja_vus, tri)
    tri.reverse()
    return tri
```

Exercice 3 (Programmation Python et routage)
Partie A : gestion des déplacements du robot

1. On a le parcours suivant :



2. Le code suivant convient :

```
def caracteres_valides(chaine):
    valides = "0123456789ADG()"
    intrus = [c for c in chaine if c not in valides]
    return len(intrus) == 0
```

3. Le code suivant convient :

```
def entiers_valides(chaine):
    chiffres = "0123456789"
    if chaine[len(chaine) - 1] in chiffres:
        return False
    for indice in range(1, len(chaine)):
        if chaine[indice] == ")":
            if chaine[indice - 1] in chiffres:
                return False
    return True
```

4. La fonction suivante convient :

```
def parenthesage_correct(chaine):
    parenthese = 0
    for c in chaine:
        if c == "(":
            parenthese += 1
        elif c == ")":
            parenthese -= 1
            if parenthese < 0:
                return False
    return parenthese == 0
```

5. A la fin de chaque tour de boucle :

Itération	Valeur de nombre	Valeur de indice
0	''	2
1	'1'	3
2	'17'	4
3	'179'	5

6. Le code suivant convient :

```
def lire_bloc(chaine, indice):
    indice = indice + 1
    caractere = chaine[indice]
    bloc = ""
    compteur = 1
    while compteur > 0:
        bloc = bloc + caractere
        indice = indice + 1
        caractere = chaine[indice]
        if caractere == "(":
            compteur = compteur + 1
        if caractere == ")":
            compteur = compteur - 1
    return (bloc, indice)
```

7. Le code suivant convient :

```
def lire_parcours(chaine):
    chiffres = "0123456789"
    indice = 0
    nombre = 1
    while indice < len(chaine):
        car_lu = chaine[indice]
        if car_lu in 'AGD': # commande simple
            for k in range(nombre):
                execute_mouvement(car_lu)
            nombre = 1
        elif car_lu in chiffres: # répétition
            t = lire_nombre(chaine, indice)
            nombre = t[0]
            indice = t[1]
        elif car_lu == '(': # début d'un bloc
            t = lire_bloc(chaine, indice)
            bloc = t[0]
            indice = t[1]
            for k in range(nombre):
                lire_parcours(bloc)
            nombre = 1
        indice = indice + 1
```

Partie B : communication et routage

8. On a la table de routage suivante :

modification	destination	prochain robot	distance
non	4	4	1
oui	46	4	2
non	22	22	1
non	57	22	2

9. On a la nouvelle table de routage suivante :

modification	destination	prochain robot	distance
non	4	4	1
non	46	4	2
non	22	22	1
non	57	22	2
oui	87	87	1
oui	63	87	2
oui	36	87	3

Partie C : programmation du routage

10. La fonction suivante convient :

```
def ajouter_voisin(self, identifiant):
    self.table_routage[identifiant] = {"prochain": identifiant, "distance": 1}
```

11. Le code suivant convient :

```
def nombre_sauts(self, identifiant):
    if identifiant not in self.table_routage:
        return 16
    else:
        return self.table_routage[identifiant]["distance"]
```

12. La fonction suivante convient :

```
def voisins(self):
    resultat = []
    for identifiant in self.table_routage:
        if self.table_routage[identifiant]["distance"] == 1:
            resultat.append(identifiant)
    return resultat
```

13. La fonction suivante convient :

```
def communiquer_extrait_table(self, voisin):
    resultat = {}
    for identifiant in self.table_routage:
        if self.table_routage[identifiant]["prochain"] != voisin:
            resultat[identifiant] = self.table_routage[identifiant]
            # ou
            # resultat[voisin] = self.table_routage[identifiant].copy()
    return resultat
```

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.