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

Centres étrangers - juin 2026 - sujet 1 (corrigé)

Exercice 1 (Tri, ABR et POO)
Partie A : dictionnaire et tri

1. Calcul du score :
   - Nombre de points au 100 mètres : $(20 - 13) \times 10 = 70$;
   - Nombre de points au saut en longueur : $5{,}2 \times 20 = 104$;
   - Nombre de points au lancer de poids : $9 \times 10 = 90$;
   - Nombre de points au 1500 mètres : $500 - 310 = 190$.

   Le score d'Alex est donc de $70 + 104 + 90 + 190 = 454$ points.
2. Le code suivant convient :

```
def nb_points(epreuve, valeur):
    points = 0
    if epreuve == '100m':
        points = (20 - valeur) * 10
    elif epreuve == 'longueur':
        points = valeur * 20
    elif epreuve == 'poids':
        points = valeur * 10
    elif epreuve == '1500m':
        points = 500 - valeur
    return points
```

3. Le code suivant convient :

```
def score(athlete):
    total = 0
    performances = athlete['performances']
    for epreuve in performances:
        valeur = performances[epreuve]
        total += nb_points(epreuve, valeur)
    athlete['score'] = total
```

4. Le code suivant convient :

```
def classer(l):
    n = len(l)
    for i in range(n):
        max_index = i
        for j in range(i + 1, n):
            if l[j]['score'] > l[max_index]['score']:
                max_index = j
        temp = l[i]
        l[i] = l[max_index]
        l[max_index] = temp
```

5. Il s'agit du tri par sélection dans l'ordre décroissant
6. D'après la question précédente, le coût de la fonction `classer` est quadratique (en $O(n^2)$).

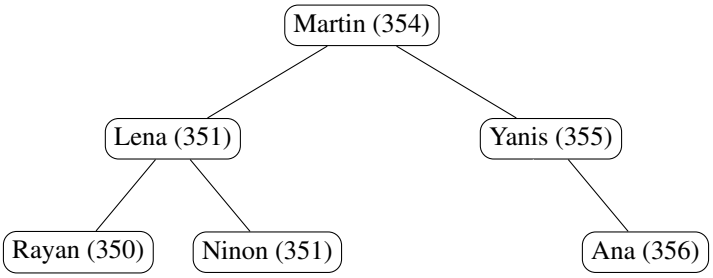
Partie B : arbre binaire de recherche

7. L'instruction suivante convient : `alex = Athlete('Alex', 13.0, 5.2, 9.0, 310.0)`

8. La fonction suivante convient :

```
def calculer_score(self):
    points = (20 - self.m100) * 10
    points += self.longueur * 20
    points += self.poids * 10
    points += 500 - self.m1500
    # on renvoie la valeur qui est affectée à self.score dans le constructeur
    return points
```

9. On obtient l'arbre suivant :



10. Le code suivant convient :

```
def inserer(self, athlete):
    if athlete.score < self.valeur.score:
        if self.gauche == None: # ou is None
            self.gauche = Noeud(athlete)
        else:
            self.gauche.inserer(athlete)
    else:
        if self.droite == None: # ou is None
            self.droite = Noeud(athlete)
        else:
            self.droite.inserer(athelte)
```

11. Il s'agit d'un parcours en profondeur dans lequel on visite les sous-arbres droits, puis l'on traite le nœud en cours et enfin on visite le sous-arbre gauche. Il s'agit donc d'un parcours infixe inversé. Dans cet arbre, les grands scores sont placés à droite. On obtient donc un classement des scores dans l'ordre décroissant.

12. ⋆ Si l'on utilise les dictionnaires et le tri par sélection :
    - Avantage : légèreté, pas de données superflues ;
    - Inconvénient : tri de coût quadratique.

    ⋆ Si l'on utilise les arbres binaires de recherche :
    - Avantage : tri efficace, de coût pseudo-linéaire $O(n \log_2(n))$ en moyenne ;
    - Inconvénient : empreinte mémoire plus lourde.

Exercice 2 (Sécurisation des communications et programmation)

1. D'après le tableau, le message est `CODE`
2. On a la table de correspondance suivante :

	1	2	3	4	5	6
1	S	E	C	U	R	I
2	T	Y	1	0	2	4
3	A	B	D	F	G	H
4	J	K	L	M	N	O
5	P	Q	V	W	X	Z
6	3	5	6	7	8	9

3. La donnée du tableau permet de chiffrer et de déchiffrer le message. Il s'agit donc d'une méthode de chiffrement symétrique.
4. On obtient : `'AXU7BCDEFGHIJKLMNOPQRSTVWYZ012345689'`
5. La fonction suivante convient :

```
def grille_vide(n):
    return [['' for j in range(n)] for i in range(n)]
```

6. Le code suivant convient :

```
def generer_grille(cle):
    ordre_insertion = generer_ordre(cle)
    grille = grille_vide(6)
    indice = 0
    for i in range(6):
        for j in range(6):
            grille[i][j] = ordre_insertion[indice]
            indice = indice + 1
    return grille
```

7. Le code suivant convient :

```
def dechiffrer(cle, message):
    resultat = ''
    grille = generer_grille(cle)
    for t in message:
        ligne = t[0] - 1
        colonne = t[1] - 1
        resultat = resultat + grille[ligne][colonne]
    return resultat
```

8. Les deux fonctions suivantes conviennent.
   - ⋆ En parcourant une fois la grille :

```
def generer_dico(cle):
    grille = generer_grille(cle)
    dico = {}
    for i in range(6):
        for j in range(6):
            caractere = grille[i][j]
            dico[caractere] = (i + 1, j + 1)
    return dico
```

   - ⋆ Sans parcourir la grille, mais en parcourant l'ordre d'insertion :

```
def generer_dico(cle):
    ordre_insertion = generer_ordre(cle)
    dico = {}
    i = 1
    j = 1
    for caractere in ordre_insertion:
        dico[caractere] = (i, j)
        j += 1
        if j == 7:
            i += 1
            j = 1
    return dico
```

9. La fonction suivante convient :

```
def chiffrer(cle, message):
    dico = generer_dico(cle)
    resultat = []
    for caractere in message:
        resultat.append(dico[caractere])
    return resultat
```

10. Dans un chiffrement symétrique, la même clé est utiliser pour chiffrer et déchiffrer. En revanche, dans un chiffrement asymétrique, ces opérations nécessitent deux clés distinctes.

Exercice 3 (POO, récursivité, langage SQL)

Partie A : la classe `Demineur`

1. L'assertion suivante convient : `assert 0.1 <= pourcentage_mines <= 0.3, 'Le pourcentage ...'`
2. Le code suivant convient :

```
class Demineur :
    def __init__(self, hauteur, largeur, pourcentage_mines):
        assert 0.1 <= pourcentage_mines <= 0.3, 'Le pourcentage de mines ...'
        self.hauteur = hauteur
        self.largeur = largeur
        self.pourcentage_mines = pourcentage_mines
```

3. L'instruction suivante convient : `demineur_intermediaire = Demineur(16, 16, 0.156)`

Partie B : création de la grille du démineur

4. Le code suivant convient :

```
def grille_demineur_vide(self):
    return [[0 for _ in range(self.largeur)] for _ in range(self.hauteur)]
```

5. Le code suivant convient :

```
while compteur_mines < nombre_bombes:
    ligne = randint(0, self.hauteur - 1)
    colonne = randint(0, self.largeur - 1)
    if self.grille_demineur[ligne][colonne] == 0:
        self.grille_demineur[ligne][colonne] = -1
        compteur_mines = compteur_mines + 1
```

6. La méthode suivante convient :

```
def nombre_voisines_avec_mines(self, coordonnees_case):
    total = 0
    for (ligne, colonne) in self.voisines(coordonnees_case):
        if self.grille_demineur[ligne][colonne] == -1:
            total += 1
    return total
```

7. La méthode suivante convient :

```
def generer_demineur(self):
    for ligne in range(self.hauteur):
        for colonne in range(self.largeur):
            if self.grille_demineur[ligne][colonne] != -1:
                self.grille_demineur[ligne][colonne] = self.nombre_voisines_avec_mines((ligne, colonne))
```

Partie C : l'interface utilisateur du jeu du démineur

8. La méthode suivante convient :

```
def visibilite(self, coords):
    ligne, colonne = coords
    if self.grille_demineur[ligne][colonne] == -1:
        self.grille_visibilite = [[True for _ in range(self.largeur)] for _ in range(self.hauteur)]
    else:
        self.grille_visibilite[ligne][colonne] = True
        if self.grille_demineur[ligne][colonne] == 0:
            for i, j in self.voisines(coords):
                if not self.grille_visibilite[i][j]:
                    self.visibilite((i, j))
```

Partie D : jouer en ligne au démineur

9. Les clés étrangères de la table `Meilleur_score` sont `joueur` qui fait référence à `Joueur.id_joueur`, et `niveau` qui fait référence à `Demineur.niveau`.

10. La requête suivante convient :

```
SELECT niveau, score FROM Meilleur_score WHERE joueur = 2
```

11. La requête suivante convient :

```
UPDATE Joueur SET mot_de_passe = 'cGhhxDE4' WHERE pseudo = 'Kirna'
```

12. Cette requête permet d'obtenir les pseudos des joueurs ayant réalisé un score strictement supérieur à 1000 sur des grilles de type expert en strictement moins de 400 secondes. On obtient donc un seul pseudo qui est `Raptor`

13. La requête suivante convient :

```
UPDATE Demineur SET niveau = 'débutant' WHERE niveau = 'facile'
```

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.