CORRECTION BAC NSI 2026 AN J1

Exercice 1 : jeu du Puissance 4 *(POO et récursivité, difficile !)*

Partie A

1.
```
def __init__(self):
    self.grille = [[0 for colonne in range(7)] for ligne in range(6) ]
```

2.
```
def joue(self, colonne, joueur):
    ligne = 5    # on part de la rangee la plus basse
    while ligne != -1 and self.grille[ligne][colonne] != 0:
        ligne = ligne - 1
    if ligne != -1:
        self.grille[ligne][colonne]= joueur
        return True
    else:
        return False
```

3.
```
jeu1 = Grille()
jeu1.jouer(2, 1)
jeu1.jouer(3, 2)
jeu1.jouer(3, 1)
```

4. Le joueur 1 occupe les cases: (5,2) et (4,3), et le joueur 2 occupe la case: (5,3)
On obtient: valeur_case(5, 3) = 7 valeur_case(5, 2) = 5 valeur_case(4, 3) = 10.
Ainsi : score(jeu1) = 7 − 5 −10 = −8

5.
```
def score(self):
    result = 0
    for ligne in range(6):
        for colonne in range(7):
            if grille[ligne][colonne] == 2 :
                result = result + valeur_case(ligne, colonne)
            elif grille[ligne][colonne] == 1 :
                result = result - valeur_case(ligne, colonne)
    return result
```

Partie B

6.
```
class Noeud :
    def __init__(self, colonne):
        self.colonne = colonne
        self.score = 0
        self.suivants = []     # liste de Noeuds fils
```

7.
```
    def colonne_score_min(self):
        noeud_min = self.suivants[0]
        for noeud in self.suivants:
            if noeud.score < noeud_min.score:
                noeud_min = noeud
        return (noeudmin.colonne, noeud_min.score)
```

8.
```
def calcule_score(self, niveau, joueur, grille):
    g = grille.gagnant()
    if g == 1:
        self.score = -(100 + 10*(niveau_max-niveau))
    elif g ==2:
        self.score = (100 + 10*(niveau_max-niveau))
    elif niveau == niveau_max:
        self.score = grille.score()
    else:
        for colonne in range(7):
            grille2 = grille.copie_grille()
            if grille2.joue(colonne, joueur):
                nouveau_noeud = Noeud(colonne)
                self.suivants.append(nouveau_noeud)
                 nouveau_noeud.calcule_score(niveau+1, 3-joueur, grille2)
    if joueur == 1:
        self.score = self.colonne_score_min()[1]
    else:
        self.score = self.colonne_score_max()[1]
```

9. Il n'est pas réaliste de prendre niveau_max = 42 car cela fait trop de possibilités, le temps de calcul sera excessif, car on a une complexité exponentielle.

Partie C

10.
```
def choisit_coup(grille, joueur):
    racine = Noeud(-1)
    racine.calcule_score(0, joueur, grille)
    if joueur == 1:
        colonne, score = racine.colonne_score_min()
    else:
        colonne, score = racine.colonne_score_max()
    return colonne
```

Exercice 2 réseau entreprise Gamerzz

1. Le masque porte sur 29 bits : il reste donc 3 bits pour la partie hostid, donc 2^3 = 8 adresses IP, mais 2 sont réservées (adresse réseau et adresse de diffusion) donc il y a 6 adresses pour des machines.
   80 s'écrit en binaire sur un octet : **01010** 000
   Seuls les 3 derniers bits sont modifiables : l'adresse du routeur peut donc être 10.42.0.81

2. 70 s'écrit en binaire sur un octet : 0100 0110
   Cela peut être une adresse du réseau Salle Gaming Online : **0100** 0000

3. Dans le réseau 10.42.0.16 /30 : Routeur C : 10.42.0.17 Routeur F : 10.42.0.18
   Dans le réseau 10.42.0.12 /30 : Routeur C : 10.42.0.13 Routeur D : 10.42.0.14

4. Table de routage du Routeur C :

Réseau	*Passerelle*	*Nombre de sauts*
DMZ	10.42.0.1 (routeur B)	1
Gaming Online	connecté	0
Internet	10.42.0.1 (routeur B)	2
Gaming VR	10.42.0.18 (routeur F)	1
Administration	10.42.0.1 (routeur B)	2
Application	10.42.0.14 (routeur D)	1
SGBD	10.42.0.14 (routeur D)	2

5. Débit 10 Gb/s : coût $10^{10}/10^{10} = 1$
   Débit 1 Gb/s : coût $10^{10}/10^{9} = 10$ Débit 100 Mb/s : coût $10^{10}/10^{8} = 100$

6. Il peut faire le chemin A-B-E-D car le coût est minimal : 12.

7. Les données d'un segment TCP sont encapsulées dans un paquet IP.

8. Une file suit le principe : « premier arrivé, premier servi » donc cela permet de traiter les segments dans leur ordre de départ, sans jamais changer (alors qu'une pile inverserait à chaque routeur).

9.
```
class Routeur_DROP_TAIL :
    def __init__(self, t_max):
        self.f = creer_file_vide()
        self.t_max = t_max      # entier positif
        self.t = 0
```

10.
```
    def recoit(self, p):
        if self.t < self.t_max :
            enfile(self.f, p)
            self.t = self.t + 1
            return True
        return False
```

```
11.     def recoit(self, p):
            if self.t < self.t_min :
                enfile(self.f, p)
                self.t = self.t + 1
                return True
            elif self.t_min <= self.t < self.t_max :
                if self.tirage_au_sort() :
                    enfile(self.f, p)
                    self.t = self.t + 1
                    return True
            return False
```

Exercice 3 gestion d'immeubles

Partie A

1. L'attribut numero_immeuble n'est pas unique car les mêmes numéros peuvent se retrouver dans différentes rues, or la clé primaire doit être unique.

2. SELECT id_immeuble
   FROM immeuble
   WHERE rue_immeuble = "la mer"
   ORDER BY id_immeuble

3. SELECT id_appart
   FROM appartement
   WHERE id_immeuble = 16 AND etage_appart >= 5

4. Cela risque de rompre l'intégrité car dans la table **appartement** il y a probablement 16 en *#id_immeuble* qui est une clé étrangère, et alors ne pointerait vers aucune référence dans la table **immeuble**.

5. INSERT INTO immeuble
   VALUES : (140, 6, 13, "Turing")

6. UPDATE appartement
   SET prix_appart = prix_appart * 2
   WHERE id_appart = 603

7. SELECT MAX(prix_appart)
   FROM appartement
   JOIN immeuble
   ON appartement.id_immeuble = immeuble.id_immeuble
   WHERE rue_immeuble = "la mer"

Partie B

8. Il y a 6 sous-séquences de longueur 2 : [3, 8] [3, 5] [1, 8] [1, 2] [1, 5] [2, 5]

9. La plus longue sous-séquence est : [1, 2, 5] sa longueur est 3.

10.
```
def est_strictement_croissante(seq):
    for i in range(0, len(seq) - 1):
        if seq[i] >= seq[i+1]:
            return False
    return True
```

11.
```
def llsc_fin(tab, i):
    if i == 0 :
        return 1:
    max_len = 1
    for j in range(i) :
        if tab[j] < tab[i]
             max_len = max(max_len, llsc_fin(tab, j) + 1)
    return max_len
```

Remarque : les 2 premieres lignes sont inutiles en fait car cela retournerait 1 aussi sans elles

12.
```
def llsc_dyn(tab):
    n = len(tab)
    dyn = [1] * n
    for i in range(1, n) :
        for j in range(i) :
            if tab[j] < tab[i] :
                dyn[i] = max(dyn[i], dyn[j]+1 )
    return max_len
```

13. La programmation dynamique permet de ne pas recalculer plusieurs fois les mêmes valeurs en les stockant dans un tableau, elle sera donc beaucoup plus rapide.