



PROBLÈME 1 (12 points)

Classe IndicateurClimatique

Q1 — get_X(self) (1 pt)

```
def get_X(self):  
    n = len(self.temps)  
    X = np.ones((n,2))  
    X[:, 1] = self.temps  
    return X
```

Q2 — get_Y(self) (1 pt)

```
def get_Y(self):  
    n = len(self.valeurs)  
    Y = self.valeurs.reshape((n, 1))  
    return Y
```

Classe ModeleIA

Q3 — __init__(self, ind) (1 pt)

```
def __init__(self, ind):  
    self.indicateur = ind  
    self.entraîne = False  
    self.parametres = None
```

Q4 — calcul_precision(self) (1.5 pts)

```
def calcul_precision(self):  
    v_reel = self.indicateur.valeurs  
    v_pred = self.predire()  
    v_moy = np.mean(v_reel)  
    ss_res = np.sum((v_reel - v_pred) ** 2)  
    ss_tot = np.sum((v_reel - v_moy) ** 2)  
    r2 = 1 - (ss_res / ss_tot)  
    return r2
```

Sous-classe ModeleLineaire (hérite de ModeleIA)

Q5 — entrainer(self) (2 pts)

```
def entrainer(self):  
    t = self.indicateur.temps  
    v = self.indicateur.valeurs  
    t_moy = np.mean(t)  
    v_moy = np.mean(v)  
    a = np.sum((t - t_moy) * (v - v_moy)) / np.sum((t - t_moy) ** 2)  
    b = v_moy - a * t_moy  
    self.parametres = np.array([a, b])  
    self.entraîne = True
```

Q6 — predire(self) (1.5 pts)

```
def predire(self):  
    if not self.entraîne:  
        print("Erreur : le modèle n'est pas encore entraîné.")  
        return None  
    a, b = self.parametres  
    t = self.indicateur.temps  
    vp = a * t + b  
    return vp
```

Sous-classe ModeleLineaireMatriciel (hérite de ModeleIA)

Q7 — entrainer(self) (2 pts)

```
def entrainer(self):
    X = self.indicateur.get_X()
    Y = self.indicateur.get_Y()
    beta = np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T, Y))
    self.parametres = beta
    self.entraîne = True
```

Q8 — predire(self) (2 pts)

```
def predire(self):
    if not self.entraîne:
        print("Erreur : le modèle n'est pas encore entraîné.")
        return None
    X = self.indicateur.get_X()
    Yp = np.dot(X, self.parametres)
    return Yp.flatten()
```

PROBLÈME 2 (8 points)

Q1 — Algèbre relationnelle (1 pt)

Stations_graves $\leftarrow \pi_{idStat}(\sigma_{gravite \geq 7}(Evenement))$
Stations_actives $\leftarrow \pi_{idStat}(\sigma_{statut = 'actif'}(Capteur))$
Résultat $\leftarrow Stations_graves - Stations_actives$

Q2 — CREATE TABLE Mesure (1 pt)

```
CREATE TABLE Mesure (
    idMes INT PRIMARY KEY,
    dateMes DATE NOT NULL,
    valeur REAL,
    idCap INT NOT NULL,
    FOREIGN KEY (idCap) REFERENCES Capteur(idCap) );
```

Q3 — Augmenter budget Tunisie de 5% (0.5 pt)

```
UPDATE Station
SET budget = budget * 1.05
WHERE paysStat = 'Tunisie';
```

Q4 — Stations ayant plus de 5 capteurs (0.5 pt)

```
SELECT idStat
FROM Capteur
GROUP BY idStat
HAVING COUNT(idCap) > 5;
```

Q5 — Stations : moyenne températures > 30°C après 2020 (1 pt)

```
SELECT s.idStat, s.nomStat
FROM Station s, Capteur c, Mesure m
WHERE c.idStat = s.idStat
AND m.idCap = c.idCap
AND m.dateMes > '2020-12-31'
GROUP BY s.idStat, s.nomStat
HAVING AVG(m.valeur) > 30;
```

Q6 — Stations sans événement Canicule (0.5 pt)

```
SELECT idStat, nomStat
FROM Station
WHERE idStat NOT IN (
    SELECT idStat
    FROM Evenement
    WHERE typeEvent = 'Canicule'
);
```

Q7 — TempReelStat(cur, idS) (1 pt)

```
def TempReelStat(cur, idS):
    cur.execute("""
        SELECT m.valeur
        FROM  Mesure m, Capteur c
        WHERE c.idCap = m.idCap
        AND   c.idStat = ?
        ORDER BY m.dateMes ASC
    """, (idS,))
    return [row[0] for row in cur.fetchall()]
```

Q8 — TempPredStat(cur, idS, nomM) (1 pt)

```
def TempPredStat(cur, idS, nomM):
    cur.execute("""
        SELECT tempPred
        FROM  Prediction
        WHERE idStat = ?
        AND   nomModele = ?
        ORDER BY datePred ASC
    """, (idS, nomM))
    return [row[0] for row in cur.fetchall()]
```

Q9 — GenDict(cur, idS) (1.5 pts)

```
def GenDict(cur, idS):
    # Récupérer les noms de modèles distincts pour cette station
    cur.execute("""
        SELECT DISTINCT nomModele
        FROM  Prediction
        WHERE idStat = ?
    """, (idS,))
    modeles = [row[0] for row in cur.fetchall()]

    resultat = {}
    for modele in modeles:
        y_reel = TempReelStat(cur, idS)
        y_pred = TempPredStat(cur, idS, modele)
        n = len(y_reel)
        mse = sum((y_reel[i] - y_pred[i]) ** 2 for i in range(n)) / n
        resultat[modele] = mse

    return resultat
```