République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Concours Nationaux d'Entrée aux
Cycles de Formation d'Ingénieurs
Session 2026

الجمهورية التونسية
وزارة التعليم العالي
والبحث العلمي
المناظرات الوطنية للدخول
إلى مراحل تكوين المهندسين
دورة 2026

AVIS IMPORTANT AUX CANDIDATS

Ce cahier d'examen est strictement personnel. Il ne doit comporter aucun signe distinctif.
Les réponses doivent être rédigées en noir ou en bleu.
Le non respect de l'une de ces recommandations peut conduire à l'attribution de la note ZERO à la copie.

Case à cocher si le candidat est absent ☐

CONCOURS : **Biologie et Géologie**

EPREUVE DE : **Informatique**

DATE :**05-06-2026 à 08 H 00**

DURÉE : **2 HEURES**

NE RIEN ÉCRIRE ICI

NOM : .....

PRÉNOM : .....

DATE DE NAISSANCE : .....

N° de C.I.N/PASSEPORT pour les étrangers : .....

SÉRIE : .....

IDENTIFIANT : .....

Signature des enseignants	

Signature du candidat

Annexe

Quelques Fonctions/Méthodes Python/Schéma Relationnel

Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.

Opérations sur les itérables (`str, tuple, list, dict, etc.`)

- `len(it)` retourne le nombre d'éléments de l'itérable it.
- `range(d,f,p)` retourne la séquence des valeurs entières successives comprises entre d et f, f exclu, par $pas = p$.
- `sum(it)` retourne la somme de tous les éléments de l'itérable it.
- `min(it, key = fct)` retourne la valeur minimale de l'itérable it où la fonction key définit le critère de comparaison.
- `max(it, key = fct)` retourne la valeur maximale de l'itérable it où la fonction key définit le critère de comparaison.
- `sum(it)` retourne la somme des éléments de l'itérable it.
- `x in it` vérifie si x appartient à it.
- `sorted(it, key = fct, reverse = True)` retourne une liste contenant les éléments de it dans l'ordre décroissant où la fonction key définit le critère de comparaison.
- `lst.reverse()` inverse l'ordre des éléments de la liste lst (modifie directement la liste).
- `lst.count(val)` retourne le nombre d'occurrences de val dans la liste lst.
- `lst.append(val)` ajoute val à la fin de la liste lst.
- `lst.remove(val)` supprime la première occurrence val de la liste lst.
- `lst.pop(idx)` supprime puis retourne la valeur située à la position idx de la liste lst.
- `lst.index(val)` retourne l'indice de la première occurrence val de la liste lst.
- `source.split(motif)` retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.
- `motif.join(itérable de chaînes)` retourne une chaîne de caractères résultante de la concaténation des éléments de l'itérable intercalés par le motif.
- `motif.format(paramètres)` retourne une chaîne de caractères obtenue en substituant dans l'ordre chaque caractère dans motif par un objet dans paramètres.
- `chaine.strip()` retourne une chaîne de caractères où les espaces (ou caractères blancs) au début et à la fin de la chaîne sont supprimés.
- `d.values()` retourne un itérable formé par les valeurs du dictionnaire d.
- `d.items()` retourne un itérable de couples (k,v) ou k est une clé du dictionnaire d et v est la valeur associée.
- `d.get(cle)` retourne la valeur associée à la clé `cle` dans le dictionnaire d ; retourne `None` si la clé n'existe pas.

Fonctions mathématiques (`module math`)

- `math.sin(x)` retourne le sinus de l'angle x (exprimé en radians).
- `math.cos(x)` retourne le cosinus de l'angle x (exprimé en radians).
- `math.asin(x)` retourne l'arc sinus de x (l'angle en radians dont le sinus vaut x).
- `math.radians(x)` convertit un angle x exprimé en degrés en radians.
- `math.sqrt(x)` retourne la racine carrée du nombre x.

Opérations sur les fichiers :

- `f=open(nomF,m)` permet d'ouvrir le fichier `nomF` en mode `m` ou `m='r'` ou `'w'`.

— `f.close()` permet de fermer un fichier.

— `f.readline()` permet de lire et retourner la ligne courante d'un fichier dans une chaîne de caractères.

— `f.readlines()` permet de lire et retourner le contenu de toutes les lignes d'un fichier dans une liste.

Module sqlite3

— `cur.execute(req)` exécuter la requête SQL décrite par le `str req` à partir du curseur `cur`.

— `cur.execute(req, seq)` exécuter la requête paramétrée décrite en SQL par le `str req` à partir du curseur `cur`.

— `cur.fetchall()` récupère tous les résultats de la dernière requête exécutée avec le curseur `cur` et les retourne sous forme de liste de tuples.

Schéma relationnel BD

- `Port(idPort, nomPort, paysPort, villePort, statutPort)`
- `Navire(idN, nomN, capMax, statCert, #idPort)`
- `Conteneur(idC, numSer, typeC, longC, largC, hautC, dateInsp, statMaint)`
- `Marchandise(idM, descM, typeM, poidsM, longM, largM, hautM, valDec, statDouane)`
- `Trajet(idT, dateDep, dateArr, statT, #idN, #idPortDep, #idPortArr)`
- `Affectation(idA, dateCharg, dateDecharg, statA, #idC, #idT)`
- `Placement(idP, posX, posY, posZ, rotP, #idM, #idA)`
- `Frais(idF, frDouane, frTransp, taxPort, assur, #idT, #idC)`
- `Document(idD, statValid, #idC, #idT)`

CONSIGNES GÉNÉRALES

DOCUMENT NON AUTORISÉS
L'USAGE DES CALCULATRICES EST INTERDIT

CE CAHIER D'EXAMEN COMPORTE 19 PAGES + 2 PAGES SUPPLÉMENTAIRES
LES RÉPONSES DOIVENT ÊTRE ÉCRITES DANS
LES ESPACES DE RÉPONSES DÉDIÉS
EN CAS DE BESOIN UTILISER LES PAGES VIDES EN FIN DU CAHIER EN LE SIGNALANT DANS L'ESPACE DE RÉPONSE CORRESPONDANT

IL EST IMPÉRATIF DE RESPECTER LA NOMENCLATURE DES OBJETS DE L'ÉNONCÉ
UTILISER ÉVENTUELLEMENT L'ANNEXE

LES ÉNONCÉS ET LES MODÉLISATIONS PRÉSENTÉS DANS LES DEUX PARTIES ONT ÉTÉ ADAPTÉS SPÉCIFIQUEMENT POUR LES BESOINS DE L'ÉPREUVE POUR DES FINS D'APPRENTISSAGE ET D'ÉVALUATION UNIQUEMENT.

Le sujet comporte deux parties qui traitent les aspects suivants :

Partie I : Programmation procédurale (**Q1..Q6**).

Partie II : Base de données relationnelle (**Q7..Q23**).

NE RIEN ÉCRIRE ICI

Partie I : Programmation Procédurale

Contexte général : Transport maritime

Le transport maritime constitue l'un des piliers fondamentaux du commerce international. L'optimisation des itinéraires entre ports maritimes vise à réduire les coûts, minimiser les distances parcourues, optimiser le temps de transit et améliorer l'efficacité logistique globale.

Dans ce contexte, les ports maritimes peuvent être modélisés comme des nœuds géographiques interconnectés, et les routes maritimes comme des liaisons pondérées par des distances et/ou des coûts.

Le transport maritime repose sur un réseau de ports reliés par des liaisons directes. Lorsqu'aucune liaison directe n'existe entre deux ports, le trajet s'effectue en passant par des ports intermédiaires. Dans ce problème, on s'intéresse à la modélisation de ce réseau en s'aidant de fichiers texte et de calcul des distances maritimes.

Q1. On dispose d'un fichier texte contenant les informations relatives à plusieurs ports maritimes. Chaque ligne du fichier correspond à un port et contient les champs suivants, séparés par un point-virgule ;

```
Id_port ; nom_port ; latitude ; longitude ; pays
```

Exemple de contenu du fichier :

```
0;Marseille;43.2965;5.3698;France
1;Tunis;36.8065;10.1815;Tunisie
2;Gênes;44.4056;8.9463;Italie
```

Écrire une fonction `lire_ports` qui prend en paramètres la chaine de caractères `nom_fichier` contenant le nom de fichier à lire. La fonction doit :

- Lire le contenu du fichier.
- Extraire les informations de chaque port.
- Créer et retourner une liste de listes, où chaque sous-liste contient les informations d'un port sous la forme : `[Id_port, nom_port, latitude, longitude, pays]`.

Hypothèses et contraintes :

- On suppose que les lignes du fichier texte sont triés en ordre croissant selon les identifiants des ports.
- On suppose que les identifiants des ports forment une suite d'entiers qui débute à 0.
- Faire le nécessaire pour le respect des types de données extraites.

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q1

```
def lire_ports(nom_fichier):
```

Q2. La distance entre deux ports maritimes est calculée à partir de leurs coordonnées géographiques en utilisant la **formule de Haversine** :

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

avec :

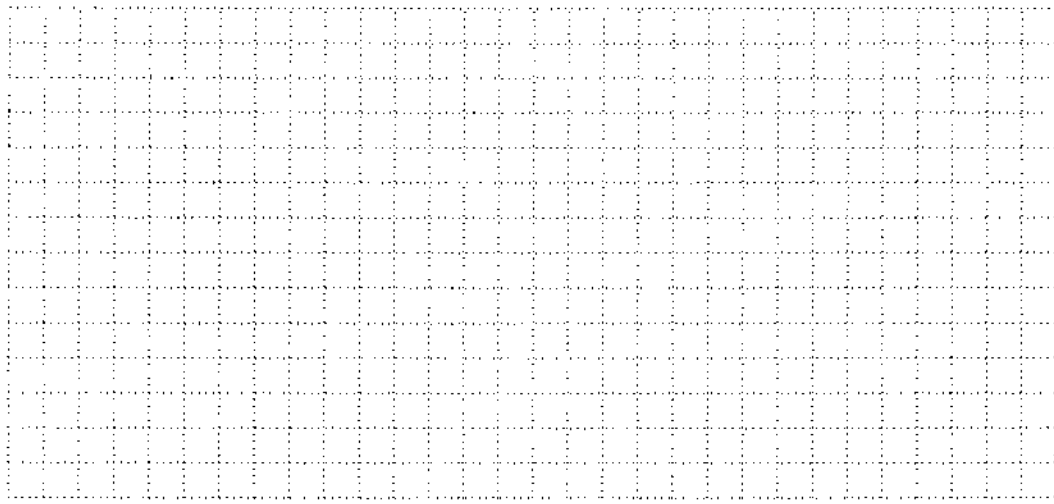
- $R = 6371\,\text{km}$: rayon moyen de la Terre ;
- φ : latitude exprimée en radians ;
- λ : longitude exprimée en radians.

Écrire une fonction `distance_ports` qui prend en paramètre deux ports `port1` et `port2` (sous forme de listes) et retourne la distance entre eux en kilomètres, en suivant les étapes suivantes :

1. Convertir les angles φ et λ en radians à partir des latitudes et longitudes des deux ports (en utilisant la fonction `radians` du module `math`) .
2. Définir $\Delta\varphi = \varphi_2 - \varphi_1$ et $\Delta\lambda = \lambda_2 - \lambda_1$.
3. Calculer $a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)$.
4. Calculer $c = \arcsin(\sqrt{a})$.
5. Calculer au final $d = 2R\,c$.

Espace de réponse pour Q2

```
import math as m
def distance_ports(port1, port2):
```



NE RIEN ÉCRIRE ICI

Q3. On dispose d'un deuxième fichier qui contient les liaisons directes entre les ports sous la forme :

```
id_port1 ; id_port2
```

Exemple de contenu du fichier :

```
0;1
0;2
1;2
1;3
2;3
```

Chaque ligne du fichier indique l'existence d'une liaison directe bidirectionnelle entre deux ports. Écrire une fonction `lire_liaisons` qui prend en paramètres la chaine de caractères `nom_fichier` contenant le nom de fichier à lire. La fonction doit :

- Lire le fichier.
- Déduire le nombre total de ports qui est donné par le maximum des identifiants des ports + 1.
- Créer et retourner la matrice d'adjacence L (sous forme d'une liste de listes) telle que :
  - $L[i][j] = 1$ s'il existe une liaison directe entre le port i et le port j,
  - $L[i][j] = 0$ sinon.

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q3

```
def lire_liaisons(nom_fichier):
```

Q4. On souhaite déterminer un parcours simple reliant deux ports dans le réseau de liaisons maritimes, représenté par une matrice d'adjacence L en utilisant l'algorithme BFS (Breadth-First Search).

Principe de l'algorithme BFS :

- Explorer d'abord tous les ports directement reliés au port de départ.
- Itérativement visiter les ports distants de 2 liaisons, puis distants de 3 liaisons, etc..
- Jusqu'à atteindre le port d'arrivée.

BFS garantit de trouver un chemin avec le plus petit nombre de liaisons.

Étapes de l'algorithme BFS

1. Initialiser les structures suivantes :
   - *a_traiter* : une liste contenant l'identifiant du port de départ,
   - *visite* : un dictionnaire vide, dédié au marquage des identifiants des ports visités,
   - *parent* : un dictionnaire vide, dédié au marquage de l'identifiant du port parent (port source relativement au parcours en cours de construction).
2. Marquer le port de départ comme visité :
   - $visite[depart] = True$
3. Identifier le parent du port de départ :
   - $parent[depart] = None$
4. Parcourir la liste *a_traiter* à l'aide un indice i :
   - assigner le port *courant* à $a\_traiter[i]$.

5. Pour chaque identifiant de port *voisin* (indice de colonne de la matrice L) :
   — s'il existe une liaison ($L[courant][voisin] == 1$)
   — et si le port n'est pas encore visité :
      — marquer *voisin* comme visité,
      — enregistrer $parent[voisin] = courant$,
      — ajouter *voisin* à la fin de la liste *a_traiter*.
6. Arrêter dès que le port d'arrivée est atteint.
7. Reconstruire le parcours à partir du dictionnaire parent.

Compléter les instructions manquantes dans l'implémentation de la fonction `determiner_parcours` qui prend en paramètres une liste de listes `L` représentant la matrice d'adjacence, ainsi que deux entiers `depart` et `arrivee`, correspondant respectivement aux identifiants des ports de départ et d'arrivée dans la liste `L`.
La fonction retourne la liste des identifiants des ports constituant le parcours trouvé.

Espace de réponse pour Q4

```
def determiner_parcours(L,depart, arrivee):
    a_traiter = ..........................................
    visite = ..........................................
    parent = ..........................................
    visite[depart] = True
    parent[depart] = None
    i = 0
    while i < len(a_traiter):
        courant = ..........................................
        if courant == arrivee:
            break
        for voisin, lien in enumerate(L[courant]):
            if lien == 1 and voisin not in visite:
                visite[voisin] = ..........................................
                parent[voisin] = courant
                a_traiter.append(..........................................)
        i += 1
    # reconstruction du parcours
    parcours = []
    courant = arrivee
    while courant is not None:
        parcours.append(courant)
        courant = parent.get(courant)
    parcours.reverse()
    return ..........................................
```

Q5. On dispose d'un parcours maritime donné sous la forme d'une liste d'identifiants de ports. L'objectif est de calculer la distance totale du parcours en additionnant les distances séparant chaque paire de ports consécutifs.

NE RIEN ÉCRIRE ICI

Écrire une fonction `distance_parcours` qui prend en entrée :

- `parcours` : une liste d'identifiants de ports $[id_1, id_2, id_3, \dots]$ représentant le trajet à suivre.
- `ports` : la liste contenant les informations de tous les ports (identifiant, nom, latitude, longitude, pays).

La fonction calcule la distance entre chaque paire de ports consécutifs du parcours en appelant la fonction `distance_ports` (voir **Q2**), puis retourne la distance totale du parcours, exprimée en kilomètres.

NB : Vérifier que le parcours contient au moins deux ports. Dans le cas contraire, la distance totale retournée est égale à 0.

Exemple

On dispose des ports suivants :

ID	Nom	Latitude	Longitude
0	Marseille	43.2965	5.3698
1	Tunis	36.8065	10.1815
3	Barcelone	41.3851	2.1734

Parcours donné : `[0, 1, 3]`

- Étape 1 : calculer la distance entre Marseille (0) et Tunis (1) ;
- Étape 2 : calculer la distance entre Tunis (1) et Barcelone (3).

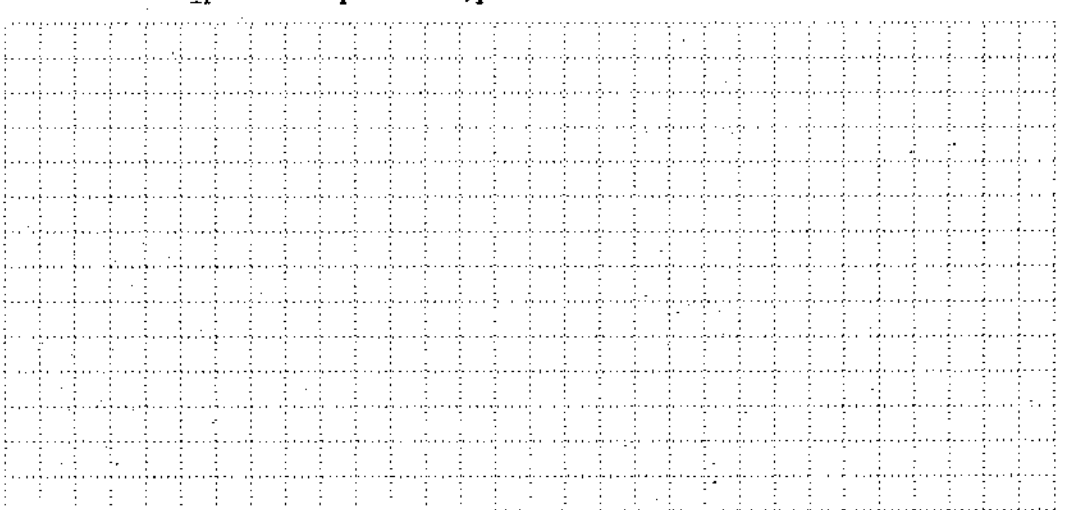
La distance totale est alors donnée par :

$$\text{distance totale} = \text{distance}(0,1) + \text{distance}(1,3) \approx 829 \text{ km} + 858 \text{ km} = 1687 \text{ km}.$$

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q5

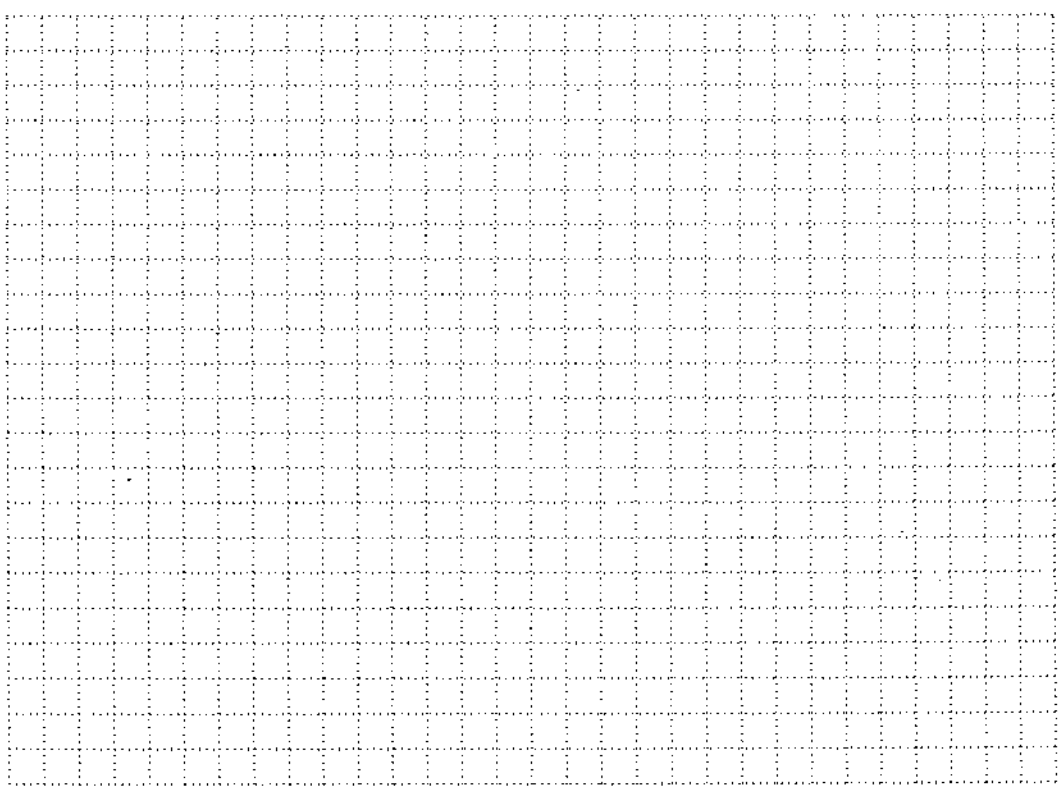
```
def distance_parcours(parcours,ports):
```



Q6. Écrire un script Python qui :

1. lit les deux fichiers `ports.txt` et `liaisons.txt` et stocke leurs contenus respectivement dans les variables `ports` et `L` ;
2. demande à l'utilisateur de saisir les identifiants de deux ports ;
3. détermine le parcours entre ces deux ports ;
4. affiche :
   - le parcours suivi (noms des ports) ;
   - la distance totale du trajet.

Espace de réponse pour Q6



NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Partie II : Base de Données Relationnelle

Le commerce maritime international, pilier de l'économie mondiale, repose sur un système complexe de gestion logistique nécessitant une coordination précise entre les acteurs du transport, les autorités portuaires et les administrations douanières. Dans ce contexte, les systèmes d'information jouent un rôle crucial pour optimiser le placement des marchandises dans les conteneurs, assurer le suivi en temps réel des navires, et calculer avec exactitude les frais de transport et droits de douane.

À l'ère de la globalisation des échanges, où près de 90% des marchandises transitent par voie maritime selon l'OMI (Organisation Maritime Internationale), une base de données relationnelle bien conçue devient l'outil indispensable pour gérer efficacement les flux maritimes, réduire les coûts logistiques, et garantir la conformité aux réglementations douanières.

Le schéma relationnel qui suit a été élaboré pour répondre à ces besoins opérationnels tout en respectant les contraintes d'un système performant et évolutif. Il s'inscrit dans le cadre des normes internationales de gestion maritime tout en intégrant les spécificités douanières régionales.

Schéma Relationnel - Base de Données Maritime `"LogiSea"` :

- `Port(idPort, nomPort, paysPort, villePort, statutPort)`
  La table `Port` enregistre les informations relatives aux ports où accostent les navires.
  - `idPort` : identifiant unique du port, de type entier clé primaire.
  - `nomPort` : nom officiel du port, de type chaîne de caractères.
  - `paysPort` : pays où se situe le port, de type chaîne de caractères.
  - `villePort` : ville où se trouve le port, de type chaîne de caractères.
  - `statutPort` : statut actuel du port (`"Actif"`, `"Inactif"`, `"En expansion"`), de type chaîne de caractères.

- `Navire(idN, nomN, capMax, statCert, #idPort)`
  La table `Navire` centralise les informations techniques et administratives des navires de transport maritime.
  - `idN` : identifiant unique du navire, de type entier clé primaire.
  - `nomN` : nom officiel du navire, de type chaîne de caractères.
  - `capMax` : capacité maximale du navire exprimée en nombre de conteneurs (TEU : Twenty-foot Equivalent Unit), de type entier.
  - `statCert` : statut des certifications du navire (`"valide"`, `"expiré"`, `"en inspection"`), de type chaîne de caractères.
  - `idPort` : identifiant du port d'attache du navire, de type entier clé étrangère faisant référence à la table `Port`.

- `Conteneur(idC, numSer, typeC, longC, largC, hautC, dateInsp, statMaint)`
  La table `Conteneur` stocke les unités de transport standardisées de type conteneur ISO.
  - `idC` : identifiant unique du conteneur, de type entier clé primaire.
  - `numSer` : numéro de série ISO du conteneur, de type chaîne de caractères unique.
  - `typeC` : type de conteneur (`"20ft"`, `"40ft"`, `"40ft HC"`, `"reefer"`, `"open top"`) de chaine de caractères.
  - `longC` / `largC` / `hautC` : longueur, largeur et hauteur internes d'un conteneur exprimées en mètre de type réel.
  - `dateInsp` : date de la dernière inspection du conteneur, de type date selon le format ISO 8601 : YYYY-MM-DD.

— `statMaint` : état de maintenance du conteneur (`"OK"`, `"à réparer"`, `"hors service"`), de type chaîne de caractères.

- `Marchandise(idM, descM, typeM, poidsM, longM, largM, hautM, valDec, statDouane)`
  La table `Marchandise` décrit les marchandises transportées ainsi que leurs caractéristiques physiques et réglementaires.
  - `idM` : identifiant unique de la marchandise, de type entier clé primaire.
  - `descM` : description détaillée de type chaine de caractères (ex : '500 cartons de smartphones neufs').
  - `typeM` : type logistique de la marchandise (`"standard"`, `"dangereux"`, `"périssable"`, `"fragile"`, `"température contrôlée"`), de type chaîne de caractères.
  - `poidsM` : poids unitaire de la marchandise en kilogrammes, de type Réel.
  - `longM` / `largM` / `hautM` : longueur, largeur et hauteur d'un carton de marchandise exprimées en mètre de type réel.
  - `valDec` : valeur déclarée en douane (en USD/EUR) de type réel.
  - `statDouane` : statut douanier de la marchandise (`"non vérifié"`, `"approuvé"`, `"retenu"`, `"refusé"`), de type chaîne de caractères.

- `Trajet(idT, dateDep, dateArr, statT, #idN, #idPortDep, #idPortArr)`
  La table `Trajet` enregistre les planifications et le suivi des voyages effectués par les navires entre ports.
  - `idT` : identifiant unique du trajet, de type entier clé primaire.
  - `dateDep` : date et heure réelles de départ du trajet, de type datetime.
  - `dateArr` : date et heure réelles d'arrivée du trajet, de type datetime.
  - `statT` : statut du trajet (`"planifié"`, `"en cours"`, `"terminé"`, `"annulé"`), de type chaîne de caractères.
  - `idN` : identifiant du navire effectuant le trajet, de type entier clé étrangère faisant référence à la table `Navire`.
  - `idPortDep` : identifiant du port de départ, de type entier clé étrangère faisant référence à la table `Port`.
  - `idPortArr` : identifiant du port d'arrivée, de type entier clé étrangère faisant référence à la table `Port`.

- `Affectation(idA, dateCharg, dateDecharg, statA, #idC, #idT)`
  La table `Affectation` stocke les liaisons temporaires entre un conteneur, un navire et un trajet donné.
  - `idA` : identifiant unique de l'affectation, de type entier clé primaire.
  - `dateCharg` / `dateDecharg` : dates de chargement/déchargement des conteneurs dans un navire pour un trajet donné de type datetime.
  - `statA` : statut de l'affectation de type chaine de caractères (`"planifié"`, `"chargé"`, `"déchargé"`, `"transbordé"`).
  - `idC` : identifiant du conteneur affecté, clé étrangère vers `Conteneur`.
  - `idT` : identifiant du trajet associé, clé étrangère vers `Trajet`.

- `Placement(idP, posX, posY, posZ, rotP, #idM, #idA)`
  La table `Placement` enregistre le positionnement optimisé d'une marchandise dans un conteneur relativement à une affectation.

NE RIEN ÉCRIRE ICI

— `idP` : identifiant unique du placement, de type entier clé primaire.
— `posX` / `posY` / `posZ` : coordonnées normalisées [0,1] d'une marchandise dans le conteneur, de type réel.
— `rotP` : rotation en degrés représentant l'orientation de la marchandise dans le conteneur, de type réel.
— `idM` : identifiant de la marchandise, clé étrangère vers `Marchandise`.
— `idA` : identifiant de l'affectation associée, clé étrangère vers `Affectation`.

- `Frais(idF, frDouane, frTransp, taxPort, assur, #idT, #idC)`
  La table `Frais` stocke les coûts liés au transport maritime et aux formalités douanières.
  — `idF` : identifiant unique des frais, de type entier clé primaire.
  — `frDouane` : montant des droits de douane, de type Réel.
  — `frTransp` : coût du transport maritime, de type Réel.
  — `taxPort` : taxes portuaires appliquées, de type Réel.
  — `assur` : coût de l'assurance, de type Réel.
  — `idT` : identifiant du trajet concerné, clé étrangère vers `Trajet`.
  — `idC` : identifiant du conteneur concerné, clé étrangère vers `Conteneur`.

- `Document(idD, statValid, #idC, #idT)`
  La table `Document` stocke les documents légaux et réglementaires liés aux trajets et aux conteneurs.
  — `idD` : identifiant unique du document, de type entier clé primaire.
  — `statValid` : statut de validité du document (`"valide"`, `"expiré"`, `"en vérification"`), de type chaîne de caractères.
  — `idC` : identifiant du conteneur associé, clé étrangère vers `Conteneur`.
  — `idT` : identifiant du trajet associé, clé étrangère vers `Trajet`.

Travail demandé

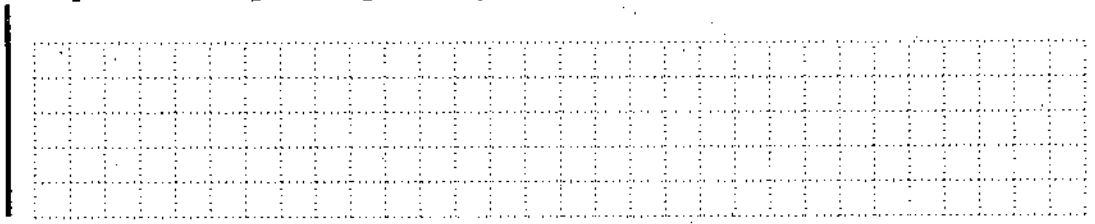
Algèbre relationnelle

Dans la suite on suppose que les tables de la base de données sont remplies en respectant les contraintes et les règles de gestion su citées.

Exprimer en algèbre relationnelle les requêtes permettant de :

Q7. Donner les identifiants et les noms des navires ainsi que les identifiants des ports de départ pour les trajets dont le statut est `"terminé"`.

Espace de réponse pour Q7



Q8. Donner les identifiants et les numéros de séries des conteneurs qui n'ont jamais transporté des marchandises de type dangereux.

Espace de réponse pour Q8

SQL

Q9. Déterminer la liste de tous les navires avec les noms de leurs ports d'attache.

Espace de réponse pour Q9

Q10. Calculer le nombre de conteneurs par type.

Espace de réponse pour Q10

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Q11. Pour les navires ayant une capacité supérieure à 10 000 TEU, lister les identifiants, les noms et les capacités.

Espace de réponse pour Q11

Q12. Lister les identifiants, les descriptions et les valeurs déclarées des marchandises dangereuses.

Espace de réponse pour Q12

Q13. Afficher les identifiants et les numéros de séries des conteneurs hors service ayant été affectés à un trajet en cours, avec les identifiants des ports de départ et d'arrivée.

Espace de réponse pour Q13

Q14. Pour chaque type de marchandise, donner le nombre total de marchandises dont le statut douanier est approuvé.

Espace de réponse pour Q14

Q15. Lister identifiants et les noms des navires qui ont transporté au moins une marchandise dangereuse, avec le nombre total de ces marchandises par navire.

Espace de réponse pour Q15

NE RIEN ECRIRE ICI

Q16. Lister les identifiants des trajets dont le statut est terminé, en affichant pour chacun la durée réelle du trajet en nombre de jours. La durée est calculée à partir des dates réelles de départ et d'arrivée.
Il est conseillé de s'inspirer de la requête suivante qui utilise la fonction prédéfinie **julianday()**.

Principe
Le jour julien est le nombre de jours écoulés depuis le 1er janvier 4713 avant J.-C. à midi (temps universel). Lorsqu'elle est utilisée pour calculer la différence entre deux dates, elle renvoie un résultat pouvant contenir une fraction de jour.
Exemple

```
SELECT
julianday('2026-02-10 18 :00 :00') - julianday('2026-02-08 12 :00 :00')
AS duree_en_jours ;
```

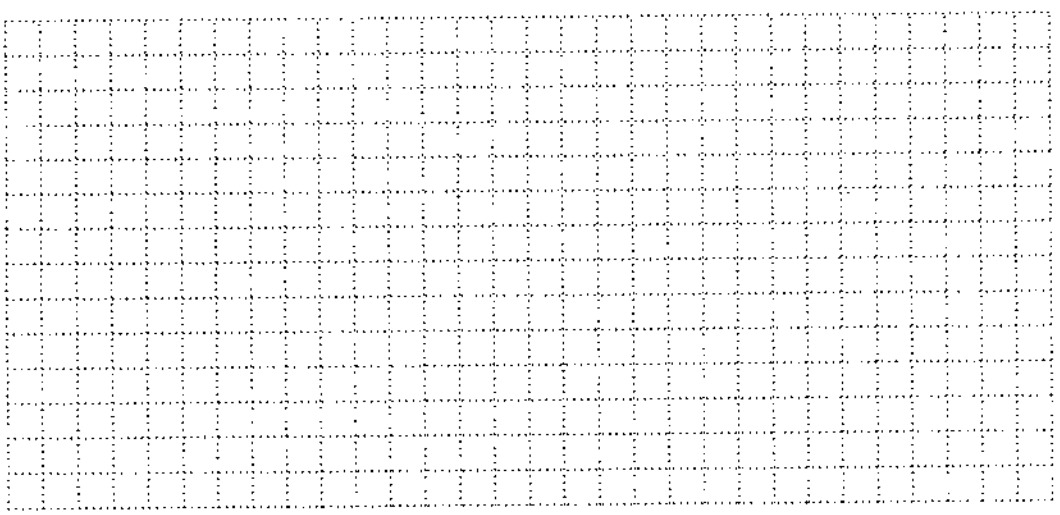
Résultat obtenu 2.25

Interprétation
On calcule la durée entre le 08 février 2026 à 12 :00 et 10 février 2026 à 18 :00
On obtient :

- 2 jours complets (du 08 février à 12 :00 au 10 février à 12 :00),
- 6 heures supplémentaires.

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q16



Q17. Lister les identifiants et les noms des Navires n'ayant jamais transporté de marchandises dangereuses.

Espace de réponse pour Q17

Q18. Pour toute affectation où le conteneur présente un taux de remplissage spatial supérieur à 80%, lister l'identifiant du conteneur, celui de l'affectation et le taux de remplissage spatial du conteneur. Ici, il y'a lieu d'utiliser le calcul de volume.

Espace de réponse pour Q18

NE RIEN ÉCRIRE ICI

Q19. Afficher les identifiants des trajets dont le coût total (douane + transport + taxes + assurance) est supérieur à la moyenne des coûts de tous les trajets.

Espace de réponse pour Q19

Q20. Lister toutes les informations concernant les conteneurs ayant tous leurs documents valides.

Espace de réponse pour Q20

Q21. Déterminer les identifants et les noms des navires pour lesquels le poids total des marchandises effectivement chargées dépasse 80 % de la capacité maximale théorique du navire (capMax), sur des trajets terminés.

Espace de réponse pour Q21

Q22. Ajout d'un attribut pertinent

Le suivi précis des navires en temps réel devient critique (sécurité, retards, sinistres). Il est donc décidé d'enrichir la base afin de stocker la position GPS actuelle du navire, indépendamment des trajets ou affectations. Changer la structure de la table navire pour ajouter l'attribut posGPS en s'aidant des données dans le tableau qui suit :

Table	Navire
Attribut	posGPS
Type	Chaîne de caractères

TABLE 1 – Spécification de l'attribut posGPS

Espace de réponse pour Q22

NE RIEN ÉCRIRE ICI

SQLITE

On s'intéresse à l'étude des placements des marchandises dans les conteneurs pour un trajet donné. Dans la suite, les fonctions demandées doivent être écrites en Python en désignant par `cur` le curseur d'exécution des requêtes.

Q23. Écrire une fonction `resumeConteneursParTrajet` qui prend en entrée `cur` le curseur de la connexion vers la base de données et un paramètre `idTrajet`et retourne un dictionnaire `resultat` où :

— chaque clé est un numéro de série du conteneur,

— chaque valeur est un dictionnaire ayant la forme suivante :

```
{
    "nb_marchandises": nbM,
    "poids_total": poids,
    "valeur_totale": somme des valeurs déclarées
 }
```

Espace de réponse pour Q23

```
def resumeConteneursParTrajet(cur, idTrajet):
```

NE RIEN ÉCRIRE ICI

Espace de réponse supplémentaire

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Espace de réponse supplémentaire

NE RIEN ECRIRE ICI

NE RIEN ECRIRE ICI

NE RIEN ECRIRE ICI

Annexe

Quelques Fonctions/Méthodes Python/Schéma Relationnel

Sans aucune obligation, les fonctions suivantes pourraient vous être utiles.

Opérations sur les itérables (`str`, `tuple`, `list`, `dict`, etc.)

- `len(it)` retourne le nombre d'éléments de l'itérable it.
- `range(d,f,p)` retourne la séquence des valeurs entières successives comprises entre d et f, f exclu, par $pas = p$.
- `sum(it)` retourne la somme de tous les éléments de l'itérable it.
- `min(it, key = fct)` retourne la valeur minimale de l'itérable it où la fonction key définit le critère de comparaison.
- `max(it, key = fct)` retourne la valeur maximale de l'itérable it où la fonction key définit le critère de comparaison.
- `sum(it)` retourne la somme des éléments de l'itérable it.
- `x in it` vérifie si x appartient à it.
- `sorted(it, key = fct, reverse = True)` retourne une liste contenant les éléments de it dans l'ordre décroissant où la fonction key définit le critère de comparaison.
- `lst.reverse()` inverse l'ordre des éléments de la liste lst (modifie directement la liste).
- `lst.count(val)` retourne le nombre d'occurrences de val dans la liste lst.
- `lst.append(val)` ajoute val à la fin de la liste lst.
- `lst.remove(val)` supprime la première occurrence val de la liste lst.
- `lst.pop(idx)` supprime puis retourne la valeur située à la position `idx` de la liste lst.
- `lst.index(val)` retourne l'indice de la première occurrence val de la liste lst.
- `source.split(motif)` retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.
- `motif.join(itérable de chaînes)` retourne une chaîne de caractères résultante de la concaténation des éléments de l'itérable intercalés par le motif.
- `motif.format(paramètres)` retourne une chaîne de caractères obtenue en substituant dans l'ordre chaque caractère dans motif par un objet dans paramètres.
- `chaine.strip()` retourne une chaîne de caractères où les espaces (ou caractères blancs) au début et à la fin de la chaîne sont supprimés.
- `d.values()` retourne un itérable formé par les valeurs du dictionnaire d.
- `d.items()` retourne un itérable de couples (k,v) ou k est une clé du dictionnaire d et v est la valeur associée.
- `d.get(cle)` retourne la valeur associée à la clé `cle` dans le dictionnaire `d`; retourne `None` si la clé n'existe pas.

Fonctions mathématiques (`module math`)

- `math.sin(x)` retourne le sinus de l'angle x (exprimé en radians).
- `math.cos(x)` retourne le cosinus de l'angle x (exprimé en radians).
- `math.asin(x)` retourne l'arc sinus de x (l'angle en radians dont le sinus vaut x).
- `math.radians(x)` convertit un angle x exprimé en degrés en radians.
- `math.sqrt(x)` retourne la racine carrée du nombre x.

Opérations sur les fichiers :

- `f=open(nomF,m)` permet d'ouvrir le fichier `nomF` en mode `m` ou `m='r'` ou `'w'`.

— `f.close()` permet de fermer un fichier.

— `f.readline()` permet de lire et retourner la ligne courante d'un fichier dans une chaîne de caractères.

— `f.readlines()` permet de lire et retourner le contenu de toutes les lignes d'un fichier dans une liste.

Module sqlite3

— `cur.execute(req)` exécuter la requête SQL décrite par le `str` req à partir du curseur cur.

— `cur.execute(req, seq)` exécuter la requête paramétrée décrite en SQL par le `str` req à partir du curseur cur.

— `cur.fetchall()` récupère tous les résultats de la dernière requête exécutée avec le curseur cur et les retourne sous forme de liste de tuples.

Schéma relationnel BD

- `Port(idPort, nomPort, paysPort, villePort, statutPort)`
- `Navire(idN, nomN, capMax, statCert, #idPort)`
- `Conteneur(idC, numSer, typeC, longC, largC, hautC, dateInsp, statMaint)`
- `Marchandise(idM, descM, typeM, poidsM, longM, largM, hautM, valDec, statDouane)`
- `Trajet(idT, dateDep, dateArr, statT, #idN, #idPortDep, #idPortArr)`
- `Affectation(idA, dateCharg, dateDecharg, statA, #idC, #idT)`
- `Placement(idP, posX, posY, posZ, rotP, #idM, #idA)`
- `Frais(idF, frDouane, frTransp, taxPort, assur, #idT, #idC)`
- `Document(idD, statValid, #idC, #idT)`