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 : **Mathématiques et physique**

EPREUVE DE : **Informatique**

DATE :**05-06-2026 à 8 H 00**

DURÉE : **2 HEURES**

NOM : .....

PRÉNOM : .....

DATE DE NAISSANCE : .....

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

SÉRIE : .....

IDENTIFIANT : .....

Signature des enseignants	

Signature du candidat

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

CONSIGNES GÉNÉRALES

DOCUMENT NON AUTORISÉS
L'USAGE DES CALCULATRICES EST INTERDIT

CE CAHIER D'EXAMEN COMPORTE 31 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 Orientée Objet (**Q1..Q24**).

Partie II : Base de Données Relationnelle (**Q25..Q35**).

Partie I : Programmation Orientée Objet : Optimisation par Intelligence en Essaim

Application du Problème du Voyageur de Commerce (TSP) au Transport Maritime

1. 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 globaux, minimiser les distances parcourues, optimiser le temps de transit et améliorer l'efficacité logistique.

Dans ce cadre, chaque port maritime est modélisé comme une instance de la classe `MaritimePort`, héritant du concept abstrait de nœud `Node`. L'ensemble des ports est organisé au sein d'un problème de type `TSPProblem`, où les routes maritimes sont assimilées à des arêtes pondérées par une distance géographique ou un coût de transport.

La recherche d'un itinéraire optimal reliant l'ensemble des ports sans répétition, et éventuellement avec retour au port de départ, relève naturellement des problèmes d'optimisation combinatoire.

2. Le TSP comme cadre d'optimisation des problèmes de transport

Le *Traveling Salesman Problem* (TSP) consiste à déterminer une tournée hamiltonienne de coût minimal, visitant exactement une fois chaque nœud d'un graphe avant de revenir au point de départ.

Dans le contexte du transport maritime, cette modélisation se traduit par :

- chaque port maritime représenté par un nœud `Node` ;
- les distances maritimes (ou coûts logistiques) définies par la métrique `distance_to` ou par une fonction personnalisée `metric` ;
- un objectif d'optimisation visant à minimiser le coût total de la tournée, calculé via la méthode `tour_cost` de la classe `TSPProblem`.

Cependant, la complexité algorithmique du TSP croît de manière exponentielle avec le nombre de ports. Les méthodes exactes deviennent rapidement impraticables, ce qui justifie le recours à des métaheuristiques capables de produire des solutions approchées de haute qualité dans des temps de calcul raisonnables.

3. Métaheuristique inspirée des colonies de fourmis

L'*Ant Colony Optimization* (ACO) est une métaheuristique bio-inspirée fondée sur l'observation du comportement collectif des fourmis lors de la recherche de chemins optimaux entre une source et une destination.

Dans la modélisation proposée, les concepts fondamentaux sont les suivants :

- **Agent (fourmi)** : chaque agent, implémenté par la classe `AntAgent`, construit progressivement une solution sous forme de tournée en visitant successivement les ports.
- **Phéromone** (τ_{ij}) : la matrice de phéromones `tau` stockée dans la classe `Environment` représente l'attractivité collective des liaisons entre les ports.
- **Visibilité** (η_{ij}) : information heuristique inversement proportionnelle à la distance entre deux ports, stockée dans la matrice `eta`.
- **Évaporation** : le mécanisme d'évaporation, contrôlé par le paramètre ρ et implémenté par la méthode `evaporate`, permet de réduire progressivement l'intensité des phéromones afin d'éviter la stagnation prématurée.

— **Renforcement** : les meilleures tournées découvertes sont renforcées par un dépôt de phéromone proportionnel à l'inverse de leur coût total, via la méthode `reinforce_best_tour` de la classe `Colony`.

Les décisions de déplacement des agents résultent d'un compromis entre l'exploitation de la phéromone accumulée, pondérée par le paramètre `alpha`, et l'exploration guidée par l'heuristique de distance, pondérée par `beta`. Ce mécanisme collectif conduit à l'émergence de solutions optimales ou quasi-optimales.

Cette partie propose ainsi une résolution du TSP appliqué au transport maritime par ACO, fondée sur une modélisation orientée objet intégrant les classes `Environment`, `AntAgent`, `Colony` et `Simulation`.

Classes à Implémenter

Dans le but de modéliser et de résoudre le *Problème du Voyageur de Commerce* (TSP) dans un contexte de transport maritime à l'aide de la métaheuristique *Ant Colony Optimization* (ACO), nous proposons l'implémentation d'un ensemble de classes permettant de structurer les données, les agents et les mécanismes d'optimisation du système.

— `Node` : Une classe abstraite représentant un nœud générique d'un graphe. Elle définit l'interface commune à tous les nœuds du problème, notamment l'identifiant et les méthodes abstraites de calcul de distance et de validation d'état.

— `MaritimePort` : Une classe concrète dérivée de `Node` représentant un port maritime réel. Elle encapsule les informations de géolocalisation (latitude, longitude, pays, code UN/LOCODE) et permet le calcul des distances maritimes entre ports.

— `TSPProblem` : Une classe responsable de la modélisation formelle du problème TSP. Elle regroupe l'ensemble des nœuds (ports maritimes), définit la métrique de distance utilisée et fournit les méthodes nécessaires à la validation des tournées et au calcul de leur coût total.

— `Environment` : Une classe représentant l'environnement dans lequel évoluent les agents ACO. Elle gère la matrice de phéromones, la matrice de visibilité, les mécanismes d'évaporation et de renforcement, ainsi que les opérations de perception et de mise à jour de l'environnement.

— `AntAgent` : Une classe représentant une fourmi, considérée comme un agent autonome. Chaque agent construit progressivement une solution en se déplaçant de nœud en nœud, en prenant ses décisions selon des règles probabilistes fondées sur l'intensité de phéromone et l'heuristique de distance.

— `Colony` : Une classe représentant une population d'agents `AntAgent`. Elle coordonne l'exécution collective des agents, la construction des tournées, la sélection des meilleures solutions et le renforcement global des phéromones au sein de l'environnement.

— `Simulation` : Une classe façade orchestrant l'ensemble du processus ACO. Elle assure l'initialisation du problème, le lancement des itérations successives de l'algorithme, ainsi que la visualisation et l'analyse des résultats, notamment la meilleure tournée trouvée et l'évolution des niveaux de phéromone.

Interface de la Classe `Node`

Rôle & responsabilités

La classe `Node` représente une entité abstraite dans un problème de type *Traveling Salesman Problem* (TSP). Elle définit un contrat commun pour tous les types de nœuds concrets, notamment les ports maritimes modélisés par la classe `MaritimePort`.

Un nœud est caractérisé par un identifiant unique et doit être capable de calculer une distance vers un autre nœud. Il doit également indiquer s'il est actif ou non, ce qui permet :

- d'exclure certains nœuds du calcul des tournées ;
- d'activer ou désactiver dynamiquement des contraintes métier (maintenance, conditions météorologiques, conflits, indisponibilité, congestion, etc.).

Attributs

- `id` : Identifiant unique du nœud.

Méthodes à implémenter

Q1. `__init__(self, node_id)` : Initialise une nouvelle instance de la classe `Node` à partir du paramètre `node_id` représentant l'identifiant unique du nœud.

Espace de réponse pour Q1

```
class Node:
    def __init__(self, node_id):
```

La méthode `distance_to(...)` (abstraite) : calcule le coût ou la distance entre le nœud courant et un autre nœud. Cette méthode sera implémentée dans les classes dérivées.

```
    def distance_to(self, other):
        # méthode abstraite
        # sera implémentée dans les classes filles
        pass
```

La méthode `is_active(...)` (abstraite) : indique si le nœud est actif ou désactivé.

— `True` : le nœud peut être utilisé dans les algorithmes ;
— `False` : le nœud doit être ignoré.

Cette méthode permet d'introduire un comportement dynamique au niveau de l'algorithme d'optimisation (ACO) considéré dans notre cas.

```
    def is_active(self):
        # méthode abstraite
        # sera implémentée dans les classes filles
        pass
```

NE RIEN ÉCRIRE ICI

Q2. `__eq__(...)` : définit l'égalité entre deux nœuds. Deux nœuds sont considérés égaux s'ils sont des instances de la classe `Node` et s'ils possèdent le même identifiant.

Espace de réponse pour Q2

```
def __eq__(self, other):
```

Interface de la classe `MaritimePort`

Rôle & responsabilités

Cette classe hérite de la classe abstraite `Node`, elle représente un port maritime utilisé comme nœud géographique dans un problème de trafic maritime. Elle encapsule les caractéristiques communes à tous les ports maritimes, notamment leur géolocalisation selon la référence mondiale WGS-84, leur état opérationnel et leur capacité à calculer une distance nautique vers un autre port.

- La géolocalisation du port est définie par ses coordonnées latitude et longitude selon la référence WGS-84.
- L'état opérationnel du port indique s'il peut être utilisé comme nœud actif dans les algorithmes considérés.
- Le port est capable de calculer une distance nautique vers un autre port maritime.

Attributs

- `id` : identifiant unique du port, hérité de la classe `Node`.
- `name` : une chaîne de caractères représentant le nom du port.
- `latitude` : un réel représentant la latitude du port (en degrés, north-south-angle, WGS-84).
- `longitude` : un réel représentant la longitude du port (en degrés, east-west-angle, WGS-84).
- `country` (optionnel) : une chaîne de caractères indiquant le pays du port, `None` par défaut.
- `un_locode` (optionnel) : le code international UN/LOCODE du port, `None` par défaut.
- `active` (optionnel) : un booléen indiquant si le port est opérationnel ou non, `True` par défaut.

Méthodes à implémenter

Q3. `__init__(...)` : Initialise une nouvelle instance de `MaritimePort` à partir des paramètres :

- identifiant du port,
- nom,
- latitude,
- longitude,
- informations optionnelles (pays, UN/LOCODE),
- état opérationnel initial.

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q3

```
from math import radians, sin, cos, sqrt, atan2
class MaritimePort(Node):
    def __init__(self, node_id, name, latitude, longitude,
                 country=None, un_locode=None, active=True):
```

Q4. `is_active(...)` : Retourne un booléen indiquant si le port est actuellement actif (opérationnel).

Espace de réponse pour Q4

```
    def is_active(self):
```

Q5. `deactivate(...)` : Désactive le port maritime afin d'indiquer qu'il est indisponible.

Espace de réponse pour Q5

```
    def deactivate(self):
```

Q6. `activate(...)` : Réactive un port précédemment désactivé.

Espace de réponse pour Q6

```
    def activate(self):
```

NE RIEN ÉCRIRE ICI

Q7. `distance_to(...)` : Calcule la distance nautique entre le port courant et un autre port maritime. La distance est calculée à partir des coordonnées géographiques en utilisant la formule de Haversine, adaptée à la navigation maritime.

Soient deux ports maritimes P_1 et P_2 définis par :

$$P_1 = (\phi_1, \lambda_1), \quad P_2 = (\phi_2, \lambda_2)$$

où :

— ϕ_1, ϕ_2 sont les latitudes de P_1 et P_2 converties en radians,

— λ_1, λ_2 sont les longitudes de P_1 et P_2 converties en radians.

On définit :

$$\Delta\phi = \phi_2 - \phi_1, \quad \Delta\lambda = \lambda_2 - \lambda_1$$

La formule de Haversine est donnée par :

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \arctan 2\left(\sqrt{a}, \sqrt{1-a}\right)$$

La distance nautique (en kilomètres) est alors :

$$d(P_1, P_2) = R \cdot c$$

avec $R = 6371$ km le rayon moyen de la Terre.

La méthode doit s'assurer que le paramètre `other` est un port maritime actif et déclencher une exception dans le cas contraire.

Espace de réponse pour Q7

```
def distance_to(self, other):
```

NE RIEN ÉCRIRE ICI

Interface de la classe `TSPProblem`

Rôle & responsabilités

La classe `TSPProblem` modélise un problème du voyageur de commerce (TSP) à partir d'un ensemble de nœuds `MaritimePort`. Elle définit les attributs principaux nécessaires à l'évaluation et à la manipulation des trajets.

Attributs

- `nodes` : dictionnaire contenant tous les nœuds du problème où les clés sont les identifiants des nœuds et les valeurs sont les nœuds eux-mêmes.
- `metric` : fonction de distance (par ex. `distance_to`) utilisée pour évaluer le coût entre deux nœuds.
- `startnode` : nœud initial du parcours.
- `return2start` : booléen définissant si le parcours doit revenir au nœud de départ.

Méthodes à implémenter

`default_metric(...)` : Cette méthode est fournie pour être utilisée ultérieurement dans le constructeur `__init__` de la classe `TSPProblem` comme fonction de distance par défaut si aucune autre métrique n'est spécifiée.

```
class TSPProblem:
    def default_metric(self, a, b):
        return a.distance_to(b)
```

Q8. `__init__(...)` : Initialise une nouvelle instance de la classe `TSPProblem` à partir des paramètres fournis.

Les paramètres sont les suivants :

- `nodes` : une liste de nœuds `Node` à inclure dans le trajet optimal recherché. La méthode doit déclencher une exception si cette liste est vide.
- `metric` (optionnel) : une fonction qui prend deux nœuds en entrée et retourne un réel positif représentant le coût de déplacement.
  - Si `metric` est fourni ($\neq$ None), alors l'attribut `metric` est initialisé avec cette fonction.
  - Sinon, l'attribut `metric` est automatiquement défini comme la méthode par défaut `default_metric` de la classe, c'est-à-dire la distance nautique entre deux ports.

  Cette flexibilité permet soit d'utiliser une métrique personnalisée, soit d'utiliser la distance par défaut définie dans `MaritimePort`.
- `startnode` (optionnel) : désigne le nœud de départ pour le parcours.
  - Si `startnode` est fourni, l'attribut `startnode` est initialisé avec cette valeur.
  - Sinon (`startnode` est None), `startnode` est automatiquement initialisé à `nodes[0]`.

  Ce nœud doit obligatoirement appartenir à la liste `nodes`.
- `return2start` (optionnel) : booléen, `True` par défaut, indiquant si le parcours doit revenir au nœud de départ à la fin. L'attribut correspondant est `return2start`.

NE RIEN ECRIRE ICI

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q8

```
class TSPProblem:
    def default_metric(self, a, b):
        return a.distance_to(b)
    def __init__(self, nodes, metric=None, startnode=None, return2start=True):
```

Q9. `get_start_node(...)` : Retourne la valeur de l'attribut `startnode`.

Espace de réponse pour Q9

```
    def get_start_node(self):
```

Q10. `is_valid_tour(...)` : Vérifie si un parcours `tour`, défini comme une liste de nœuds `Node`, est valide pour le problème `TSPProblem` défini.

Un parcours est considéré comme valide si et seulement si tous les nœuds inclus dans `tour` appartiennent aux valeurs du dictionnaire `nodes` de l'instance.

La méthode retourne un booléen indiquant la validité du parcours :

— `True` : le parcours est valide.

— `False` : au moins un nœud du parcours n'est pas présent dans `nodes`.

Espace de réponse pour Q10

```
def is_valid_tour(self, tour):
```

Q11. `tour_cost(...)` : Calcule le coût total d'un parcours `tour`, défini comme une liste de nœuds `Node`, couvrant l'ensemble des nœuds du problème `TSPProblem`.

La méthode doit respecter les règles suivantes :

- Si la taille de `tour` est inférieure à 2, le coût total est automatiquement `0`.
- Le coût total est calculé en sommant les distances successives entre les nœuds, en utilisant la fonction `metric` de l'instance.
- Si l'attribut `return2start` est `True`, ajouter le coût de retour du dernier nœud vers le nœud de départ (`startnode`).
- Retourner le coût total du parcours.

Espace de réponse pour Q11

```
def tour_cost(self, tour):
    # Calcule du coût total d'un parcours en sommant les distances
    # successives.
    # Ajoute le coût de retour si self.return2start est True.
    # Retourne le coût total.
```

NE RIEN ÉCRIRE ICI

NE RIEN ECRIRE ICI

Interface de la classe `Environment`

Rôle & responsabilités

La classe `Environment` représente l'environnement pour résoudre un problème `TSPProblem` en utilisant l'optimisation par colonie de fourmis (ACO). Elle gère :

— La matrice des phéromones τ,
— La matrice de visibilité heuristique η,
— Les paramètres contrôlant l'influence de la phéromone et de l'heuristique α et β.

Dans l'ACO, une fourmi choisit de se déplacer du nœud n_i vers le nœud n_j en suivant une arête (lien) (n_i, n_j) avec une probabilité p_{ij} donnée par :

$$p_{ij} = \frac{\tau_{ij}^{\alpha}\, \eta_{ij}^{\beta}}{\sum_{k \in N_i} \tau_{ik}^{\alpha}\, \eta_{ik}^{\beta}}$$

où :

— N_i est l'ensemble des nœuds accessibles depuis n_i,
— τ_{ij} est la quantité de phéromone déposée sur l'arête (n_i, n_j), représentant l'expérience collective des fourmis,
— η_{ij} est l'information heuristique associée à l'arête (n_i, n_j), généralement définie comme :

$$\eta_{ij} = \frac{1}{d_{ij}}$$

— α contrôle l'influence de la phéromone (mémoire collective),
— β contrôle l'importance de l'heuristique (connaissance locale),
— p_{ij} : probabilité qu'une fourmi se déplace du nœud n_i vers le nœud n_j.

Les valeurs de τ_{ij} sont mises à jour au fil des itérations pour renforcer les bons chemins, assurant un équilibre entre exploration et exploitation.

Attributs

— `tsp_problem` : instance de `TSPProblem` sur laquelle l'environnement opère.
— `n_nodes` : entier correspondant au nombre de nœuds du problème.
— `alpha` : réel représentant l'exposant pour l'influence des phéromones.
— `beta` : réel représentant l'exposant pour l'influence de la visibilité (heuristique η).
— `tau0` : valeur initiale des phéromones.
— `id2idx` : dictionnaire où les clés sont les identifiants des nœuds et les valeurs sont les indices correspondants dans les matrices `tau` et `eta`.
— `tau` : matrice des phéromones de forme (`n_nodes`, `n_nodes`).
— `eta` : matrice de visibilité (1/distance) de forme (`n_nodes`, `n_nodes`).

Méthodes à implémenter

Q12. `__init__(...)` : Initialise une nouvelle instance de la classe **Environment** à partir des paramètres fournis :

- `tsp_problem` : instance de `TSPProblem` sur laquelle l'environnement opérera.
- `alpha` : réel représentant l'exposant pour l'influence des phéromones.
- `beta` : réel représentant l'exposant pour l'influence de la visibilité (heuristique η).
- `tau0` : valeur initiale des phéromones τ pour tous les couples de nœuds.

Le dictionnaire `id2idx` est construit en associant chaque identifiant de nœud présent dans l'attribut `nodes` de `tsp_problem` à un indice entier selon l'ordre croissant des identifiants. Cette association, bien que faite de manière arbitraire, reste **fixe et cohérente** pendant toute l'exécution de l'algorithme. Ces indices permettent de faire correspondre les lignes et colonnes des matrices de phéromones `tau` et de visibilité `eta` aux nœuds réels du problème.

Exemple concret avec des ports maritimes :

```
nodes = {
    10: MaritimePort(node_id=10, name="Tunis", latitude=36.8, longitude=10.2),
    25: MaritimePort(node_id=25, name="Sfax", latitude=34.7, longitude=10.7),
    42: MaritimePort(node_id=42, name="La Goulette", latitude=36.9, longitude=10.3)
}

id2idx = {10: 0, 25: 1, 42: 2}
```

Ainsi, l'élément $\tau_{1,2}$ représente la quantité de phéromone associée au déplacement du port Sfax ($id = 25$) vers le port La Goulette ($id = 42$).

Les matrices `tau` et `eta` sont initialisées comme suit :

$$\tau_{ij} = \texttt{tau0} \quad \forall i, j$$

$$\eta_{ij} = \begin{cases} 0 & \text{si } i = j, \\ \dfrac{1}{d(n_i, n_j)} & \text{sinon,} \end{cases}$$

où $d(n_i, n_j)$ est la distance nautique entre les ports i et j calculée via `tsp_problem.metric`.

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q12

```
class Environment:
    def __init__(self, tsp_problem, alpha, beta, tau0):
```

Q13. `get_index(...)` : Retourne les indices associés à deux nœuds du problème `TSPProblem` dans les matrices de l'environnement `Environment`.

Plus précisément, pour deux nœuds `ni` et `nj`, la méthode :

— Recherche leurs identifiants dans le dictionnaire `id2idx` de l'instance,

— Retourne un tuple (i, j) contenant leurs valeurs associées dans le dictionnaire.

Espace de réponse pour Q13

```
    def get_index(self, ni, nj):
```

Q14. `get_pheromone(...)` : Retourne la quantité de phéromone τ_{ij} déposée sur l'arête reliant deux nœuds `ni`, `nj` d'un problème `TSPProblem`.

Espace de réponse pour Q14

```
def get_pheromone(self, ni, nj):
```

Q15. `update_pheromone(...)` : Ajouter Δ à la phéromone sur la liaison entre les nœuds `ni` et `nj`.

$$\tau_{ij} = \tau_{ij} + \Delta$$

Espace de réponse pour Q15

```
def update_pheromone(self, ni, nj, delta):
```

Q16. `get_transition_attractivity(...)` : Calculer l'attractivité de transition d'une fourmi entre les nœuds `ni` et `nj`, combinant l'influence des phéromones et de la visibilité heuristique.
Cette attractivité sera normalisée ultérieurement afin d'obtenir les probabilités de transition du nœud `ni` vers le nœud `nj`.

$$a_{ij} = \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$$

Espace de réponse pour Q16

```
def get_transition_attractivity(self, ni, nj):
```

Q17. `evaporate(...)` : Applique l'évaporation des phéromones sur toutes les liaisons.
Le paramètre $\rho \in [0, 1]$ représente le **taux d'évaporation** : plus ρ est grand, plus les phéromones sont réduites à chaque itération, permettant d'éviter que certaines solutions anciennes dominent indéfiniment.

$$\tau_{ij} = \max\Big((1-\rho) \cdot \tau_{ij}, \tau_0\Big)$$

Cette opération assure que les valeurs de phéromones restent $\geq \tau_0$, garantissant ainsi un minimum de visibilité pour toutes les liaisons.

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q17

```
def evaporate(self, rho):
```

Interface de la classe `AntAgent`

Rôle & responsabilités

La classe `AntAgent` représente une fourmi dans un algorithme ACO appliqué à un problème `TSPProblem`. Elle contient les caractéristiques de l'agent (un navire dans notre cas) participant à la construction d'un trajet, à savoir :

- gestion et construction du `tour` actuel,
- sélection des prochains nœuds à visiter en fonction des informations de phéromones et de visibilité,
- interaction avec l'environnement `env` pour guider ses choix.

Attributs

- `env` : instance de `Environment` utilisée pour guider la fourmi durant la construction d'un trajet.
- `tour` : liste de nœuds représentant le parcours actuel de l'agent.

La Méthode `__init__`

`__init__(...)` : dont l'implémentation est fournie initialise les attributs `env` et `tour` de la fourmi. Le parcours `tour` est initialisé à une liste vide.

Implémentation en Python

```
import random

class AntAgent:
    def __init__(self, env):
        self.env = env
        self.tour = []
```

Q18. `select_next_node(...)` : Sélectionne et retourne le prochain nœud à visiter à partir du dernier nœud du `tour` en suivant les étapes ci-dessous :

1. **Identifier les nœuds non visités** : Construire l'ensemble des identifiants des nœuds encore disponibles :

$$\mathcal{N}_{\text{unvisited}} = \{i \in \texttt{tsp\_problem.nodes} \mid n_i \notin \texttt{tour}\}$$

2. **Calculer les attractivités des transitions** : Pour chaque nœud non visité n_j tel que $j \in \mathcal{N}_{\text{unvisited}}$, calculer son attractivité depuis le dernier nœud du tour $n_i = \texttt{tour}[-1]$:

$$a_{ij} = \text{env.get\_transition\_attractivity}(n_i, n_j)$$

et stocker ces valeurs dans une liste a.

3. **Normaliser les attractivités en probabilités** : Transformer les attractivités en probabilités de transition :

$$p_{ij} = \frac{a_{ij}}{\sum_{n_k \text{ tel que } k \in \mathcal{N}_{\text{unvisited}}} a_{ik}}$$

de sorte que $\sum_{j \in \mathcal{N}_{\text{unvisited}}} p_{ij} = 1$.

4. **Sélection aléatoire du prochain nœud** : Choisir le prochain nœud en fonction de la distribution de probabilités p. Pour cela on utilisera :

```
import random
next_node = random.choices(population=unvisited_nodes,
    weights=probabilities, k=1 )[0]
```

où :

— `population` : liste des nœuds non visités,

— `weights` : probabilités normalisées p_{ij},

— `k` : nombre d'éléments à tirer (ici $k = 1$).

Espace de réponse pour Q18

```
def select_next_node(self):
```

Q19. `construct_tour(...)` : Construit un parcours complet pour la fourmi en suivant les étapes ci-dessous :

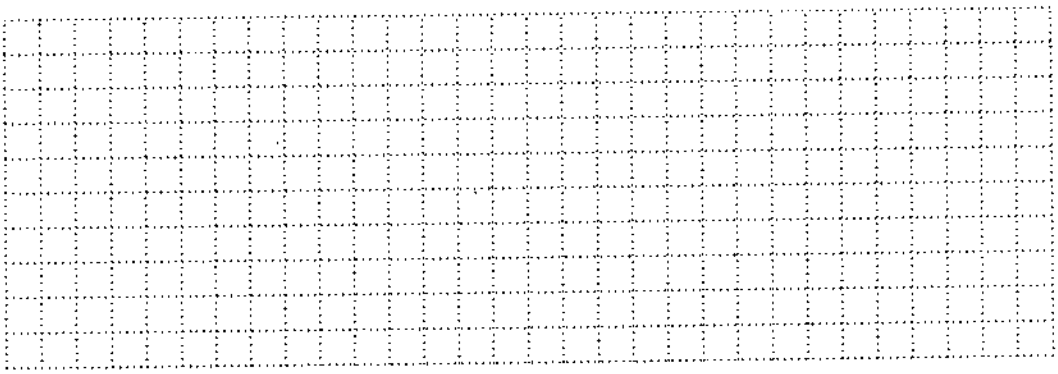
1. **Initialisation du tour** : Commencer à partir du nœud de départ du `tsp_problem`, accessible via l'environnement `env` :

$$\texttt{tour} \leftarrow [\texttt{env.tsp\_problem.startnode}]$$

2. **Sélection itérative des nœuds** : Répéter la sélection du prochain nœud jusqu'à ce que tous les nœuds actifs soient visités.
3. **Mise à jour du tour** : À chaque itération, le nœud sélectionné est ajouté à la liste `tour`.

Espace de réponse pour Q19

```
def construct_tour(self):
```



Interface de la classe `Colony`

Rôle & responsabilités

La classe `Colony` représente une colonie d'agents (fourmis) qui coopèrent pour résoudre un problème TSP. Elle permet notamment :

- la gestion de plusieurs agents construisant des parcours,
- la sélection du meilleur parcours trouvé par les agents,
- l'application des règles de renforcement des phéromones dans l'environnement,
- le contrôle du taux d'évaporation des phéromones.

Attributs

- `env` : instance de `Environment` contenant les matrices de phéromones `tau` et de visibilité `eta`.
- `n_ants` : entier représentant le nombre d'agents (`AntAgent`) dans la colonie.
- `rho` : réel représentant le taux d'évaporation des phéromones, $0 < \texttt{rho} \leq 1$.
- `ants` : liste de `AntAgent` représentant les agents de la colonie.
- `best_tour` : liste de nœuds représentant le meilleur parcours trouvé jusqu'à présent.
- `best_length` : réel représentant la longueur du meilleur parcours trouvé.

Méthodes à implémenter

Q20. `__init__(...)` : Initialise les attributs `env`, `n_ants`, `rho` de la colonie. Crée `n_ants` instances de `AntAgent`, et initialise `best_tour` par une liste vide ainsi que `best_length` à $+\infty$ (`float('inf')`).

Espace de réponse pour Q20

```
def __init__(self, env, n_agents, rho):
```

Q21. `reinforce_best_tour(...)` : Dépose des phéromones sur les liaisons du meilleur parcours `best_tour` de la colonie.

1. **Calcul du dépôt de phéromone** : La quantité de phéromone à déposer pour chaque liaison est inversement proportionnelle à la longueur du meilleur parcours :

$$\Delta\tau = \frac{1}{\texttt{best\_length}}$$

2. **Mise à jour des phéromones** : Pour chaque couple de nœuds consécutifs (n_i, n_j) du `best_tour`, mettre à jour la matrice de phéromones via :

$$\texttt{env.update\_pheromone}(n_i, n_j, \Delta\tau)$$

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Espace de réponse pour Q21

```
def reinforce_best_tour(self):
```

Q22. `run_iteration(...)` : Exécute une itération complète de l'algorithme ACO en suivant les étapes ci-dessous :

1. **Construction des parcours par les agents** : Chaque agent (`AntAgent`) construit un trajet complet en utilisant sa méthode `construct_tour` :

   pour chaque ant $\in$ `ants` : ant.construct_tour()

2. **Mise à jour du meilleur parcours** : Si un agent a trouvé un parcours plus court que `best_length`, mettre à jour :

   $$\texttt{best\_tour} \leftarrow \texttt{tour}_{\text{agent}}, \quad \texttt{best\_length} \leftarrow \textit{tour\_cost}(\texttt{tour}_{\text{agent}})$$

3. **Évaporation des phéromones** : Appliquer l'évaporation sur toutes les liaisons via :

   env.evaporate(rho)

4. **Renforcement des phéromones** : Déposer des phéromones le long du meilleur parcours trouvé avec :

   self.reinforce_best_tour()

Espace de réponse pour Q22

```
def run_iteration(self):
```

La Méthode run

La méthode run dont le code est fourni, exécute un nombre spécifié d'itérations de l'algorithme ACO, et met à jour le meilleur parcours trouvé par la colonie.

1. Répéter `run_iteration` pour `n_iterations` fois :

   pour $i \in \{1, \ldots, n\_iterations\}$: self.run_iteration()

2. À la fin des itérations, `best_tour` et `best_length` contiennent le meilleur résultat trouvé par la colonie.

Implémentation en Python

```
def run(self, n_iterations):
    for i in range(n_iterations):
        self.run_iteration()
```

Interface de la classe `Simulation`

Rôle & responsabilités

La classe `Simulation` permet de simuler un problème de transport maritime en se basant sur l'algorithme ACO. Elle orchestre les différentes étapes de la simulation :

- Initialisation du problème `TSPProblem` à partir d'un ensemble de nœuds représentant les ports maritimes.
- Création de l'environnement `Environment` avec les matrices de phéromones et de visibilité.
- Gestion d'une `Colony` d'agents représentant les navires participant à la construction des parcours.
- Exécution de la simulation pour un nombre spécifié d'itérations.
- Visualisation du meilleur parcours trouvé et de la matrice des phéromones.

Attributs

- `tsp_problem` : instance de `TSPProblem` représentant le problème TSP maritime.
- `env` : instance de `Environment` contenant les matrices de phéromones et de visibilité.
- `colony` : instance de `Colony` représentant la colonie de navires.
- `nodes` : liste de `MaritimePort` représentant tous les ports ou points du TSP.
- `rho` : un réel représentant le tau d'évaporation de la phéromone.

Méthodes à implémenter

La Méthode `__init__`

`__init__(...)` : dont le code est fourni, initialise les attributs de la simulation et crée les instances nécessaires pour l'ACO :

1. Création du problème `TSPProblem` à partir des nœuds fournis et du paramètre `return2start`.
2. Initialisation de l'environnement `Environment` avec les paramètres `alpha`, `beta` et `tau0`.
3. Création de la colonie `Colony` avec `n_agents` agents et le taux d'évaporation `rho`.

Implémentation en Python

```
import matplotlib.pyplot as plt

class Simulation:
    def __init__(self,nodes,alpha = 1.0,beta = 2.0,tau0 = None,
        n_ants = 10,rho = 0.1,return2start = True)
            self.nodes=nodes
        # Creation de TSP problem
        self.tsp_problem=TSPProblem(nodes,return2start)
        # Creation de environment
        self.env=Environment(self.tsp_problem,alpha,beta,tau0)
        # Creation de colony
        self.colony=Colony(self.env,n_agents,rho)
```

La Méthode run

La méthode run dont le code est fourni, exécute la simulation complète ACO pour `n_iterations` itérations.

Implémentation en Python

```
    def run(self, n_iterations):
        """execution de la simulation ACO pour un nombre d'itérations."""
        self.colony.run(n_iterations)
```

Q23. `plot_best_tour(...)` : Visualise le meilleur parcours trouvé par la colonie en appliquant les étapes suivantes :

1. Extraire les coordonnées longitude (`lon`) et latitude (`lat`) des nœuds du meilleur parcours.
2. Si `return2start=True`, boucler le parcours sur le nœud de départ.
3. Afficher le parcours sous forme de graphique, en reliant les nœuds par des lignes et en marquant les positions par des points.
4. Annoter chaque nœud avec son identifiant et afficher la longueur totale du parcours :

$$L_{\text{total}} = \text{self.colony.best\_length}$$

Compléter le code manquant dans les trous.

Espace de réponse pour Q23

```
def plot_best_tour(self):
    """ Visualise le meilleur tour trouvé par la colonie.  """
    best_tour = ..................................................
    if not best_tour:
        print("No tour to plot.")
        return

    # Extraction des coordonnées
    x_coords = [node.latitude for node in best_tour]
    y_coords = [..................................................]

    # Boucler le parcours sur le nœud de départ si return2start
    if self.tsp_problem.return2start:
        x_coords.append(.....................................)
        y_coords.append(.....................................)

    plt.figure(figsize=(8, 6))
    plt.plot(..............,..............., 'o-', label='Best tour')
    for i, node in enumerate(best_tour):
        plt.text(node.latitude, node.longitude, f'{node.id}')
    plt.title(f"Best Tour (Length: {self.colony.best_length:.2f})")
    plt.xlabel("X latitude")
    plt.ylabel("Y longitude")
    plt.grid(True)
    plt.legend()
    plt.show()
```

Q24. `plot_pheromone_matrix(...)` : Visualise la matrice des phéromones de l'environnement :

1. Afficher la matrice `env.tau` sous forme de heatmap.
2. Ajouter une barre de couleur pour indiquer les niveaux de phéromone.
3. Indiquer les indices des nœuds sur les axes.

Compléter le code manquant.

NE RIEN ECRIRE ICI

Espace de réponse pour Q24

```
def plot_pheromone_matrix(self):
    """ Visualise  la matrice de phéromone  en tant que heatmap. """
    plt.figure(figsize=(6, 5))
    plt.imshow(.....), cmap='hot')
    plt.colorbar(label="Pheromone Level")
    plt.title("Pheromone Matrix")
    plt.xlabel("Node Index")
    plt.ylabel("Node Index")
    plt.show()
```

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.

NE RIEN ÉCRIRE ICI

— `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.

— `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 :

Q25. 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 Q25

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Q26. Donner les identifiants et les numéros de séries des conteneurs qui n'ont jamais transporté des marchandises de type dangereux.

Espace de réponse pour Q26

SQL

Q27. Pour les navires ayant une capacité supérieure à 10 000 TEU, lister les identifiants, les noms et les capacités.

Espace de réponse pour Q27

Q28. Pour chaque type de marchandise, donner le nombre total de marchandises dont le statut douanier est approuvé.

Espace de réponse pour Q28

Q29. Lister les identifiants et les noms des Navires n'ayant jamais transporté de marchandises dangereuses.

Espace de réponse pour Q29

Q30. 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 Q30

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Q31. 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 Q31

Q32. Lister toutes les informations concernant les conteneurs ayant tous leurs documents valides.

Espace de réponse pour Q32

Q33. Lister les identifiants des trajets pour lesquels tous les conteneurs affectés possèdent une maintenance OK.

Espace de réponse pour Q33

Q34. 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 Q34

NE RIEN ÉCRIRE ICI

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.

Q35. 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 Q35

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

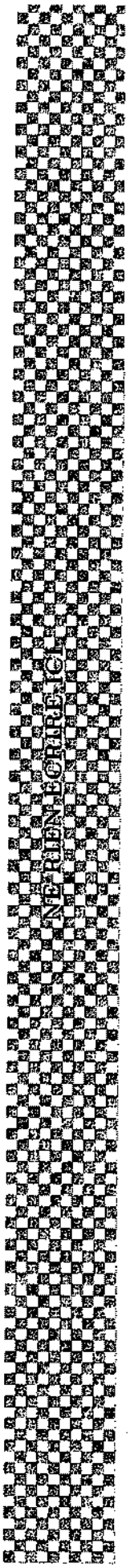
Espace de réponse supplémentaire

NE RIEN ÉCRIRE ICI

NE RIEN ÉCRIRE ICI

Espace de réponse supplémentaire

NE RIEN ÉCRIRE ICI


NE RIEN ÉCRIRE 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)` retourne la valeur minimale 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)` retourne la valeur maximale de l'itérable it.
- `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)` retourne une liste contenant les éléments de it dans l'ordre croissant.
- `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.sort()` trie la liste lst dans l'ordre croissant.
- `lst.sort(key = fct, reverse = True)` trie la liste lst en ordre décroissant en utilisant key comme critère de comparaison.
- `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.
- `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.

Module `random`

- `random.choices(population, weights=None, cum_weights=None, k=1)` retourne une liste de `k` éléments tirés aléatoirement avec remise dans `population`, avec une probabilité uniforme par défaut ou pondérée par `weights` (ou `cum\_weights`).

Module `numpy`

- `M.shape` ou `np.shape(M)` retourne un `tuple` formé par le nombre de lignes et le nombre de colonnes d'une matrice M.

— `np.array(lst)` construit une instance de la classe `np.ndarray` dont les composantes sont initialisées à partir de la liste passée en entrée.

— `np.zeros(tpl)` construit une instance de la classe `np.ndarray` dont les composantes sont initialisées par zéro ayant un `shape` comme décrit par le `tuple tpl`.

— `np.ones(tpl)` construit une instance de la classe `np.ndarray` dont toutes les composantes sont initialisées à la valeur 1 et dont le `shape` est donné par le `tuple tpl`.

— `np.full(tpl, v)` construit une instance de la classe `np.ndarray` dont toutes les composantes sont initialisées à la valeur `v` et dont le `shape` est donné par le `tuple tpl`.

— `np.maximum(A,B)` retourne un `np.ndarray` contenant le maximum élément par élément entre les deux tableaux `A` et `B` de même dimension.

— `np.max(A)` retourne la valeur maximale parmi toutes les composantes du tableau `A`.

Module `matplotlib.pyplot`

— `plt.plot(x, y)` trace une courbe (ligne 2D) où `x` et `y` sont deux itérables contenant respectivement les abscisses et les ordonnées.

— `plt.imshow(M)` affiche une image correspondant au tableau `M`. Si `M` est une matrice 2D, les valeurs sont représentées par des couleurs selon une palette (`colormap`). Si `M` est un tableau 3D de dimension $(h, w, 3)$ ou $(h, w, 4)$, il est interprété comme une image couleur RGB ou RGBA.

— `plt.axis(opt)` permet de contrôler l'affichage des axes du graphique. Par exemple `plt.axis("off")` masque les axes et `plt.axis("equal")` impose la même échelle pour les deux axes.

— `plt.xlabel(txt)` ajoute une étiquette descriptive `txt` à l'axe des abscisses.

— `plt.title(txt)` ajoute un titre `txt` au graphique courant.

— `plt.show()` affiche la figure contenant les tracés ou images précédemment définis.

— Dans `plt.plot`, la couleur et le style de la ligne peuvent être spécifiés par un argument optionnel tel que `'r'`, `'g'`, `'b'`, `'k'` correspondant respectivement aux couleurs rouge, vert, bleu et noir. Il est aussi possible d'utiliser des styles de lignes comme `'--'` (pointillé) ou `':'` (ligne pointillée).

— Dans `plt.imshow`, la palette de couleurs peut être contrôlée avec l'argument `cmap`. Par exemple `'gray'` pour une échelle de gris, `'viridis'`, `'plasma'` ou `'jet'` pour différentes palettes colorées.

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)