Baccalauréat général

Epreuve blanche d’enseignement de spécialité

AVRIL | MAI 2025

NUMÉRIQUES ET SCIENCES INFORMATIQUES

Durée de l’épreuve 3 heures 30

Aucun document autorisé

L’usage de la calculatrice n’est pas autorisé.

Dès que ce sujet est remis, assurez-vous qu’il est complet

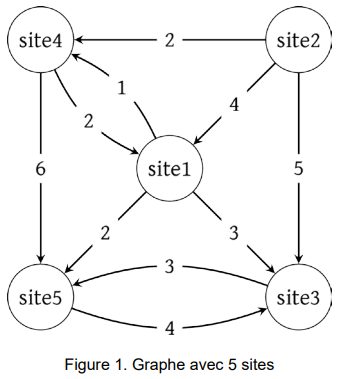
Le sujet n’est pas à rendre avec la copie

**Le sujet est composé de trois exercices indépendants**

# Exercice 1 (6 points)

Cet exercice porte sur la programmation objet en Python et les graphes.

Nous avons représenté sous la forme d’un graphe les liens entre cinq différents sites Web :



La valeur de chaque arête représente le nombre de citations (de liens hypertextes) d’un site vers un autre. Ainsi, le site **site4** contient 6 liens hypertextes qui renvoient vers le site **site5**.

Les sites sont représentés par des objets de la classe Site dont le code est partiellement donné ci-dessous. La complétion de la méthode calculPopularite fera l’objet d’une question ultérieure

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **class** Site:    **def** \_\_init\_\_(self, nom):  self.nom = nom  self.predesseurs = []  self.successeurs = []  self.popularite = 0  self.couleur = 'blanche'  **def** calculPopularite(self):  ... |

Le graphe précédent peut alors être représenté ainsi :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | # Description du graphe  s1, s2, s3, s4, s5 = Site('site1'), Site('site2'), Site('site3'), Site('site4'), Site('site5')  s1.successeurs = [(s3,3), (s4,1), (s5,3)]  s2.successeurs = [(s1,4), (s3,5), (s4,2)]  s3.successeurs = [(s5, 3)]  s4.successeurs = [(s1,2), (s5,6)]  s5.successeurs = [(s3,4)]  s1.predecesseurs = [(s2,4), (s4,2)]  s2.predecesseurs = []  s3.predecesseurs = [(s1,3), (s2,5), (s5,4)]  s4.predecesseurs = ...  s5.predecesseurs = … |

1. Expliquer la ligne 9 de ce code.
2. Les lignes 11 et 12 de cette description du graphe ne sont pas complètes.

Recopier et compléter le code des lignes 11 et 12.

1. Donner et expliquer la valeur de l’expression suivante :

s2.successeurs[1][1]

Pour mesurer la pertinence d’un site, on commence par lui attribuer un nombre appelé valeur de popularité qui correspond au nombre de fois qu’il est cité dans les autres sites, c’est-à-dire le nombre de liens hypertextes qui renvoient sur lui. Par exemple, la valeur de popularité du site **site4** est 3.

1. Donner, selon cette définition, la valeur de popularité du site **site1**.
2. Écrire sur votre copie le code de la méthode calculPopularite de la classe Site qui affecte à l’attribut popularite la valeur de popularité correspondante et renvoie cet attribut.

Afin de calculer cette valeur de popularité pour chacun des sites, nous allons faire un parcours dans le graphe de façon à exécuter la méthode calculPopularite pour chacun des objets.

Voici le code de la fonction qui permet le parcours du graphe :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **def** parcoursGraphe(sommetDepart):  parcours = []  sommetDepart.couleur = 'noire'  listeS = []  listeS.append(sommetDepart)  **while** len(listeS) != 0:  site = listeS.pop(0)  site.calculPopularite()  parcours.append(site)  **for** successeur **in** site.successeurs:  **if** successeur[0].couleur == 'blanche':  successeur[0].couleur = 'noire'  listeS.append( successeur[0] )  **return** parcours |

On rappelle les points suivants :

* la méthode append ajoute un élément à une liste Python ;

par exemple, tab.append(el) permet d’ajouter l’élément el à la liste Python tab ;

* la méthode pop enlève de la liste l'élément situé à la position indiquée et le renvoie en valeur de retour ;

par exemple, tab.pop(2) enlève l'élément à l'indice 2 et le renvoie.

Dans ce parcours, les sites non encore traités sont de couleur 'blanche' (valeur par défaut à la création de l’objet) et ceux qui sont traités de couleur 'noire'.

1. Dans ce parcours, on manipule la liste Python nommée listeS uniquement à l’aide d’appels de la forme listeS.append(sommet) et listeS.pop(0). Donner la structure de données correspondant à ces manipulations.

Donner la structure de données correspondant à ces manipulations.

1. Donner le nom de ce parcours de graphe.
2. La fonction parcoursGraphe renvoie une liste parcours. Indiquer la valeur renvoyée par l’appel de fonction :

parcoursGraphe(s1)

On cherche maintenant le site le plus populaire, celui dont la valeur de popularité est la plus grande.

Voici le code de la fonction qui renvoie le site le plus populaire, elle prend comme argument une liste non vide contenant des instances de la classe Site.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **def** lePlusPopulaire(listeSites):  maxPopularite = 0  siteLePlusPopulaire=listeSites[0]  **for** site **in** listeSites:  **if** site.popularite > maxPopularite:  ...  ...  **return** siteLePlusPopulaire |

1. Copier et compléter les lignes 6 et 7 de cette fonction.
2. Donner ce que renvoie la ligne de code suivante :

lePlusPopulaire(parcoursGraphe(s1)).nom

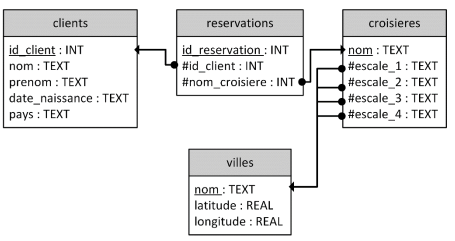
1. On envisage d’utiliser l’ensemble des fonctions proposées ci-dessus pour rechercher le site le plus populaire parmi un très grand nombre de sites (quelques milliers de sites). Expliquer si ce code est adapté à une telle quantité de sites à traiter. Justifier votre réponse.

# Exercice 2 (6 points)

Cet exercice traite de protocoles de routage, de sécurité des communications et de base de données relationnelle.

Une agence de voyage propose des croisières en bateau. Chaque croisière a un nom unique et passe par quatre escales correspondant à des villes qui ont elles aussi des noms différents.

Pour gérer les réservations de ses clients, l’agence utilise une base de données. Voici la description des trois relations de cette base dont les clés primaires ont été soulignées et les clés étrangères indiquées par un # :



Remarque : l’énoncé de cet exercice utilise tout ou une partie des mots suivants du langage SQL : SELECT, FROM, WHERE, JOIN ON, INSERT INTO, VALUES, UPDATE SET, OR, AND.

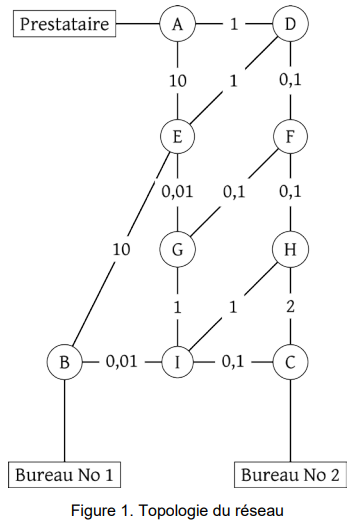
## PARTIE A

L’agence de voyage possède deux bureaux distincts

Elle passe par un prestataire de service qui héberge sa base de données et utilise un système de gestion de base de données relationnelle.

Vous trouverez ci-après un schéma du réseau entre les deux bureaux de l’agence de voyage et le prestataire.

On peut y voir les différents routeurs (nommés de A à I) ainsi que le coût des liaisons entre eux



1. Donner deux services rendus par un système de gestion de bases de données relationnelles.

Le protocole RIP (Routing Information Protocol) est un protocole de routage qui minimise le nombre de routeurs par lesquels les paquets transitent.

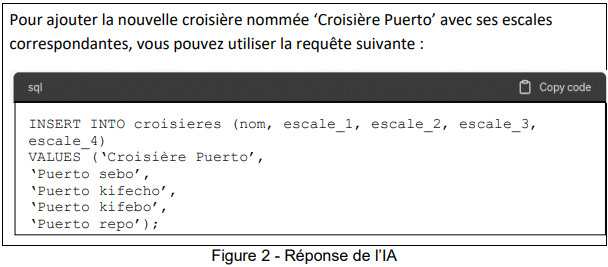
Le protocole OSPF (Open Shortest Path First) est un protocole de routage qui minimise le coût du transit des paquets.

1. Donner la route suivie par une requête issue du bureau numéro 1 jusqu’au prestataire si on utilise le protocole RIP.
2. Donner les deux routes que pourrait suivre une requête issue du bureau numéro 2 jusqu’au prestataire si on utilise le protocole OSPF. Donner le coût de chaque route.

## PARTIE B

1. Expliquer pourquoi l’attribut id\_client a été choisi comme clé primaire dans la relation clients.
2. Définir ce qu’est une clé étrangère. Donner la ou les clés étrangères de chaque relation qui en a en précisant la clé primaire qu’elles référencent.

L’agence a obtenu l’autorisation de faire escale dans quatre nouvelles villes : Puerto saibo, Puerto kifecho, Puerto kifebo et Puerto repo. Elle souhaite créer une nouvelle croisière qui passera par ces quatre villes. Un stagiaire de l’agence demande de l’aide à une Intelligence Artificielle (IA) :



Il tape alors la requête proposée mais obtient le message d’erreur suivant du SGBD (Systèmes de Gestion de Bases de Données) : FOREIGN KEY constraint failed.

1. Expliquer l’erreur commise et proposer une solution.

## PARTIE C

1. Jean Barc, un allemand né le 29 juin 1972, demande un geste commercial en raison de sa fidélité à l’agence. Expliquer les requêtes SQL suivantes saisies par le gestionnaire :

|  |
| --- |
| **SELECT** id **FROM** clients  **WHERE** nom = 'Barc' **AND** prenom = 'Jean' **AND** date\_naissance = '1972/06/29' **AND** pays = 'Allemagne';  **SELECT** id\_reservation **FROM** reservations  **WHERE** id\_client = 1243; |

1. Écrire les deux requêtes de la question 7. sous la forme d’une requête unique.
2. Un client souhaite modifier sa réservation d’identifiant 20456. Il souhaite remplacer la croisière de cette réservation par la toute dernière offre de l’agence : la Croisière Puerto. Écrire une requête SQL qui permet de mettre à jour la base de données pour lui donner satisfaction.
3. Donner une requête SQL permettant d’obtenir les noms, prénoms et dates de naissance des clients ayant choisi la croisière nommée Croisière Piano ou celle nommée Croisière Puerto.

# Exercice 3 (8 points)

Cet exercice porte sur la programmation orientée objet, sur les arbres binaires de recherche et la récursivité.

Chaque année, plusieurs courses de chiens de traîneaux sont organisées sur les terrains enneigés. L’une d’elle, La Traversée Blanche, est une course se déroulant en 9 étapes. L’organisateur de cette course est chargé de créer un programme Python pour aider à la bonne gestion de l’événement.

## PARTIE A : La classe Chien

Afin de caractériser un chien, l’organisateur décide de créer une classe Chien avec les attributs suivants :

* id\_chien, un nombre entier correspondant au numéro attribué au chien lors de son inscription à la course ;
* nom, une chaîne de caractères correspondant au nom du chien ;
* role, une chaîne de caractères correspondant au poste occupé par le chien : en fonction de sa place dans l’attelage, un chien a un rôle bien défini et peut être 'leader', 'swing dog', 'wheel dog' ou 'team dog'.
* id\_proprietaire, un nombre entier correspondant au numéro de l’équipe

Le code Python incomplet de la classe Chien est donné ci-dessous.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **class** Chien:  **def** \_\_init\_\_(self, id\_chien, nom, role, id\_prop):  self.id\_chien = id\_chien  self.nom = nom  self.role = role  self.id\_proprietaire = id\_prop  **def** changer\_role(self, nouveau\_role):  """Change le rôle du chien avec la valeur passée en paramètre."""  ... |

Voici un extrait des informations dont on dispose sur les chiens inscrits à la course.

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiens inscrits à la course** | | | |
| id\_chien | nom | role | id\_proprietaire |
| 40 | Duke | wheel dog | 10 |
| 41 | Sadie | team dog | 10 |
| 42 | Zeus | swing dog | 11 |
| 43 | Roxie | swing dog | 11 |
| 44 | Scout | team dog | 11 |
| 45 | Ginger | team dog | 11 |
| 46 | Helka | team dog | 11 |

Suite aux inscriptions, l’organisateur procède à la création de tous les objets de type Chien et les stocke dans des variables en choisissant un nom explicite. Ainsi, l’objet dont l’attribut id\_chien a pour valeur 40 est stocké dans la variable chien40.

1. Écrire l’instruction permettant d’instancier l’objet chien40 caractérisant le chien ayant le numéro d’inscription 40.
2. Selon l’état de fatigue de ses chiens ou du profil de l’étape, le musher (nom donné à la personne qui conduit le traîneau) peut décider de changer le rôle des chiens dans l’attelage.

Recopier et compléter la méthode changer\_role de la classe Chien.

1. Le propriétaire de Duke décide de lui attribuer le rôle de 'leader'.

Écrire l’instruction permettant d’effectuer cette modification.

## PARTIE B : La classe Equipe

On souhaite à présent créer une classe Equipe ayant les attributs suivants :

* num\_dossard, un nombre entier correspondant au numéro inscrit sur le dossard du musher ;
* nom\_equipe, une chaîne de caractères correspondant au nom de l’équipe ;
* liste\_chiens, une liste d’objets de type Chien dont chaque élément correspond à un chien au départ de l’étape du jour ;
* temps\_etape, une chaîne de caractères (par exemple '2h34') représentant le temps mis par l’équipe pour parcourir l’étape du jour ;
* liste\_temps, une liste de chaînes de caractères permettant de stocker les temps de l’équipe pour chacune des 9 étapes. Cet attribut peut, par exemple, contenir la liste : ['4h36', '3h57', '3h09', '5h49', '4h45', '3h26', '4h57', '5h52', '4h31'].

On donne le code Python suivant de la classe Equipe.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **class** Equipe:  **def** \_\_init\_\_(self, num\_dossard, nom\_equipe):  self.num\_dossard = num\_dossard  self.nom\_equipe = nom\_equipe  self.liste\_chiens = []  self.temps\_etape = ''  self.liste\_temps = []  **def** ajouter\_chien(self, chien):  self.liste\_chiens.append(chien)  **def** retirer\_chien(self, numero):  ...  **def** ajouter\_temps\_etape(self, temps):  self.liste\_temps.append(temps) |

Pour la première étape, le musher de l’équipe numéro 11, représentée en Python par l’objet eq11, décide de constituer une équipe avec les quatre chiens identifiés par les numéros 42, 44, 45 et 46. On donne ci-dessous les instructions Python permettant de créer l’équipe eq11 et l’attelage constitué des 4 chiens précédents.

|  |  |
| --- | --- |
| 1  2  3  4  5 | eq11 = Equipe(11, 'Malamutes Endurants')  eq11.ajouter\_chien(chien42)  eq11.ajouter\_chien(chien44)  eq11.ajouter\_chien(chien45)  eq11.ajouter\_chien(chien46) |

Malheureusement, le musher s’aperçoit que sa chienne Helka, chien numéro 46, n’est pas au mieux de sa forme et il décide de la retirer de l’attelage.

1. Recopier et compléter la méthode retirer\_chien ayant pour paramètre numero, un entier correspondant au numéro attribué au chien lors de l’inscription, et permettant de mettre à jour l’attribut liste\_chiens après retrait du chien dont la valeur de l’attribut id\_chien est numero.
2. En vous aidant de la fonction précédente, écrire l’instruction qui permet de retirer Helka de l’attelage de l’équipe eq11.
3. Indiquer le résultat renvoyé par l’appel convert('4h36').
4. Écrire une fonction temps\_course qui prend pour paramètre equipe de type Equipe et qui renvoie un nombre flottant correspondant au cumul des temps de l’équipe equipe à l’issue des 9 étapes de la course. On rappelle que la classe Equipe dispose d’un attribut liste\_temps.

## PARTIE C : classement à l’issue d’une étape

Chaque jour, à la fin de l’étape, on décide de construire un Arbre Binaire de Recherche (ABR) afin d’établir le classement des équipes. Chaque nœud de cet arbre est un objet de type Equipe.

Dans cet arbre binaire de recherche, en tout nœud :

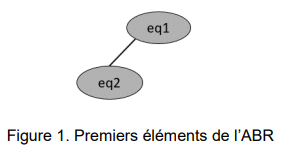
* toutes les équipes du sous-arbre gauche sont strictement plus rapides que ce nœud ;
* toutes les équipes du sous-arbre droit sont moins rapides ou sont à égalité avec ce nœud.

Voici les temps, en heure et minute, relevés à l’issue de la première étape :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Temps à l’arrivée de la première étape** | | | | | | | | | | | |
| Equipe | eq1 | eq2 | eq3 | eq4 | eq5 | eq6 | eq7 | eq8 | eq9 | eq10 | eq11 |
| Temps | 4h36 | 3h57 | 3h09 | 5h49 | 4h45 | 3h26 | 4h51 | 5h52 | 4h31 | 3h44 | 4h26 |

Dans l’arbre binaire de recherche initialement vide, on ajoute successivement, dans cet ordre, les équipes eq1, eq2, eq3, …, eq11, 11 objets de la classe Equipe tous construits sur le même modèle que l’objet eq11 précédent.

1. Dans l’arbre binaire de recherche ci-dessous, les nœuds eq1 et eq2 ont été insérés. Recopier et compléter cet arbre en insérant les 9 nœuds manquants.



1. Indiquer quel parcours d’arbre permet d’obtenir la liste des équipes classées de la plus rapide à la plus lente.

On donne ci-dessous la classe Noeud, permettant de définir les arbres binaires :

|  |  |
| --- | --- |
| 1  2  3  4  5 | **class** Noeud:  **def** \_\_init\_\_(self, equipe, gauche = None, droit = None):  self.racine = equipe  self.gauche = gauche  self.droit = droit |

On donne ci-dessous le code d’une fonction construction\_arbre qui, à partir d’une liste d’éléments de type Noeud permet d’insérer successivement chaque nœud à sa place dans l’ABR.

|  |  |
| --- | --- |
| 1  2  3  4  5 | **def** construction\_arbre(liste):  a = Noeud(liste[0])  **for** i **in** range(1,len(liste)):  inserer(a, liste[i])  **return** a |

La fonction construction\_arbre fait appel à la fonction inserer qui prend pour paramètre arb, de type Noeud, et eq, de type Equipe. Cette fonction construit le nœud à partir de eq et l’insère à sa place dans l’ABR

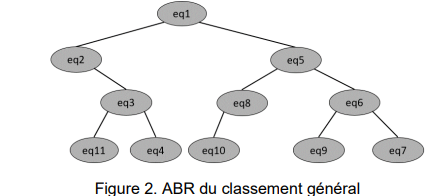
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **def** inserer(arb, eq):  """ Insertion d'une équipe à sa place dans un ABR contenant  au moins un noeud."""  **if** convert(eq.temps\_etape) < convert(arb.racine.temps\_etape):  **if** arb.gauche **is** None:  arb.gauche = ...  **else**:  inserer(..., eq)  **else**:  **if** arb.droit **is** None:  arb.droit = Noeud(eq)  **else**:  ... |

1. Expliquer en quoi la fonction inserer est une fonction récursive.
2. Recopier et compléter les lignes 6, 8 et 13 de la fonction inserer.
3. Recopier et compléter les lignes 3 et 5 de la fonction est\_gagnante ci-dessous qui prend en paramètre un ABR arbre, de type Noeud, et qui renvoie le nom de l’équipe ayant gagné l’étape.

|  |  |
| --- | --- |
| 1  2  3  4  5 | **def** est\_gagnante(arbre):  **if** arbre.gauche == None:  **return** ...  **else**:  **return** ... |

## PARTIE D : classement général

On décide d’établir un classement général obtenu à partir du cumul des temps mis par chaque équipe pour parcourir l’ensemble des 9 étapes

Sur le même principe que l’arbre de la partie précédente, on construit l’ABR ci-dessous qui permet, grâce au parcours d’arbre approprié, d’établir ce classement général des équipes. 

Le règlement prévoit la disqualification d’une équipe en cas de non-respect de celui-ci. Il s’avère que l’équipe 2 et l’équipe 5 doivent être disqualifiées pour manquement au règlement. Les nœuds eq2 et eq5 doivent donc être supprimés de l’ABR précédent.

Pour supprimer un nœud N dans un ABR, trois possibilités se présentent :

* le nœud N à supprimer est une feuille : il suffit de le retirer de l’arbre ;
* le nœud N à supprimer n’a qu’un seul fils : on relie le fils de N au père de N et on supprime le nœud N ;
* le nœud N à supprimer possède deux fils : on le remplace par son successeur (l’équipe qui a le temps immédiatement supérieur) qui est toujours le minimum de ses descendants droits.

1. Dessiner le nouvel arbre de recherche a\_final obtenu après suppression des équipes eq2 et eq5 dans l’ABR correspondant au classement général.

L’organisateur souhaite disposer d’une fonction rechercher permettant de savoir si une équipe a été disqualifiée ou non. On donne les spécifications de la fonction rechercher, prenant en paramètre arbre et equipe.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | def rechercher(arbre, equipe):  """  Paramètres  ---------  arbre : un ABR, non vide, de type Noeud, représentant le classement général.  equipe : un élément, de type Equipe, dont on veut déterminer l'appartenance  ou non à l'ABR arbre.  Résultat  ---------  Cette fonction renvoie True si equipe est un nœud de arbre,  False sinon.  """  ... |

Pour cette fonction (a\_final désigne l’arbre obtenu à la question 13, après suppression des équipes 2 et 5) :

* l’appel rechercher(a\_final, eq1) renvoie True ;
* l’appel rechercher(a\_final, eq2) renvoie False.

1. Écrire le code de la fonction rechercher