

Devoir Surveillé n°1

Aucun document autorisé // Calculatrice autorisée // **Durée : 2h**

1. SQL

Exercice 1. *Maths : CCinP 2023*

On considère deux tables : **CLIENTS** et **PARTENAIRES**. La première contient des informations sur les clients et la deuxième permet d'identifier qui sont les partenaires des clients.

La table **CLIENTS** contient les attributs suivants :

- **id** : identifiant d'un individu (entier), clé primaire ;
- **nom** : (chaîne de caractères) ;
- **prenom** : (chaîne de caractères) ;
- **ville** : (chaîne de caractères) ;
- **email** : (chaîne de caractères).

La table **PARTENAIRES** contient les attributs suivants :

- **id** : identifiant de suivi (entier), clé primaire ;
 - **id_client** : identifiant du client représenté par l'attribut **id** dans la table **CLIENTS** (entier) ;
 - **partenaire** : nom du partenaire (chaîne de caractères).
1. Écrire une requête SQL permettant d'extraire les identifiants de tous les clients provenant de la ville de 'Toulouse'.
 2. Écrire une requête SQL permettant d'extraire les emails de tous les clients ayant 'SCEI' comme partenaire.

III.1 - Exploitation des données d'opération enregistrées

En option sur toutes les machines, l'équipementier propose un système de saisie de données d'opération qui enregistre en permanence les données importantes d'opérations pendant les travaux. Ces données peuvent ensuite être traitées sur un ordinateur par un logiciel qui permet d'établir des rapports de chantier.

Les données sont enregistrées dans une base de données dont une structure simplifiée est la suivante : une table **Site** répertoriant les chantiers exploités, une table **Forage** répertoriant tous les pieux de chaque site et une table **Operation** répertoriant toutes les opérations sur chaque pieu.

La structure détaillée est la suivante :

- la table **Site**, contient les trois attributs suivants :
 - **idsite** (de type *entier*) : numéro indiquant la référence du site d'exploitation
 - **nom** (de type *chaîne de caractères*) : désignation du nom du lieu du site d'exploitation
 - **dates** (de type *tuple*) : tuple contenant les dates de début et de fin de chantier
- la table **Forage**, contient les quatre attributs suivants :
 - **pieunumero** (de type *entier*) : numéro indiquant la référence du pieu de forage
 - **idsite** (de type *entier*) : numéro indiquant la référence du site d'exploitation du pieu considéré
 - **coordonneesGPS** (de type *chaîne de caractères*) : coordonnées GPS du pieu considéré
 - **quantitebeton** (de type *flottant*) : quantité de béton coulée dans le pieu considéré
- la table **Operation**, contient les douze attributs suivants :
 - **operationnumero** (de type *entier*) : numéro indiquant la référence de l'opération réalisée
 - **pieunumero** (de type *entier*) : numéro indiquant la référence du pieu de forage considéré
 - **idmachine** (de type *entier*) : numéro indiquant la référence de la machine utilisée
 - **idoutil** (de type *entier*) : numéro indiquant la référence de l'outil utilisé
 - **date** (de type *chaîne de caractères*) : date de l'opération réalisée
 - **duree** (de type *float*) : durée (en seconde) de l'opération réalisée
 - **temps** (de type *liste*) : liste des instants (en seconde) des prises de mesure des capteurs pendant l'opération considérée
 - **pression** (de type *liste*) : liste des mesures de la pression du sol (en Pa) lors de l'opération
 - **effort** (de type *liste*) : liste des mesures de l'effort de forage (en kN) lors de l'opération
 - **profondeur** (de type *liste*) : liste des mesures de profondeur (en cm) lors de l'opération
 - **couple** (de type *liste*) : liste des mesures du couple de forage (en kN.m) lors de l'opération
 - **vitesserotation** (de type *liste*) : liste des mesures de la vitesse de rotation (en tr/min) de l'outil lors de l'opération.

Q31. Écrire une requête SQL permettant de renvoyer les dates de début et de fin du chantier intitulé " Heilbronn ".

Q32. Écrire une requête SQL permettant de déterminer la quantité totale de béton qui a été coulée sur ce chantier.

Q33. Écrire une requête SQL permettant de renvoyer les listes *temps*, *effort* et *profondeur* de la première opération (numérotée 1) du premier pieu (numéroté 1) effectué sur ce chantier.

2. Tris

Exercice 3. *Tri par insertion*

Le tri par insertion est un algorithme simple et naturel. Voici son principe : on se donne une liste L en argument et on procède par en utilisant un invariant de boucle :

- La liste contenant seulement le premier élément de la liste L est triée ;
- Soit $1 \leq k < \text{len}(L) - 1$. On suppose la sous-liste des k premiers éléments de L triée. Alors on place le $(k + 1)$ ème élément de L dans la sous-liste des k premiers en le faisant glisser vers la gauche par des échanges successifs et en s'arrêtant :
 - soit si on arrive en première position ;
 - soit si l'élément à gauche est inférieur ou égal.

On remarque que l'on a besoin, pour le tri par insertion, d'une fonction qui permet d'insérer à sa place l'élément $L[k]$ dans la partie du tableau $L[0:k]$ (les k premiers éléments de L - d'indices de 0 à $k - 1$) qui est supposé trié par ordre croissant.

1. Écrire une fonction `insertion(L,k)` qui prend pour arguments une liste L de nombres et un indice k et qui place l'élément $L[k]$ au bon endroit dans la partie du tableau $L[0:k]$ (qui est supposée triée bien-sûr).
2. Écrire une fonction `tri_insertion(L)` qui prend pour argument une liste L de nombres et qui réalise l'algorithme décrit plus haut (et qui bien sûr utilise la fonction `insertion`!).

Exercice 4. *Tri à bulles*

Le but de cet exercice est de coder l'algorithme de **tri à bulles** qui permet de trier une liste de nombres. On considère une liste L de taille $n = \text{len}(L)$. Le principe du tri à bulle est le suivant :

- i) On parcourt la liste L de gauche à droite avec i allant de 0 à $n-2$ et on échange $L[i]$ et $L[i+1]$ si ces deux éléments sont mal ordonnés i.e. si $L[i] > L[i+1]$.
- ii) Une fois le premier parcours terminé, on recommence le parcours de la liste du début et on recommence les échanges. Et ainsi de suite, on recommence à parcourir/échanger tant qu'il reste des échanges à effectuer.
- iii) On renvoie la liste : elle est triée !

Dans la figure suivante, on peut voir le déroulement de l'algorithme de tri à bulles sur la liste $L = [1, 6, 4, 2, 5, 3]$.

1er Parcours						
1	6	4	2	5	3	Pas d'échange
1	6	4	2	5	3	Échange 6 et 4
1	4	6	2	5	3	Échange 6 et 2
1	4	2	6	5	3	Échange 6 et 5
1	4	2	5	6	3	Échange 6 et 3
1	4	2	5	3	6	Fin du 1 ^{er} Parcours
2eme Parcours						
1	4	2	5	3	6	Pas d'échange
1	4	2	5	3	6	Échange 4 et 2
1	2	4	5	3	6	Pas d'échange
1	2	4	5	3	6	Échange 5 et 3
1	2	4	3	5	6	Pas d'échange
1	2	4	3	5	6	Fin du 2eme Parcours
3eme Parcours						
1	2	4	3	5	6	Pas d'échange
1	2	4	3	5	6	Pas d'échange
1	2	4	3	5	6	Échange 4 et 3
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Fin du 3eme Parcours
4eme Parcours						
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Pas d'échange
1	2	3	4	5	6	Fin du 4eme Parcours
Aucun échange dans le parcours n°4 : Fin de l'algorithme, on renvoie la liste triée.						

- Écrire une fonction `parcourt(L)` qui parcourt la liste `L` de gauche à droite et qui effectue les échanges comme indiqué au point i). Cette fonction renvoie `True` si elle a effectué des échanges lors du parcours et `False` sinon.
 - Écrire une fonction `triabulles(L)` qui effectue l'algorithme de tri à bulles - on se servira bien sûr de la fonction `parcourt(L)`.
- Quel est la complexité de cet algorithme de tri dans le pire des cas en terme d'échanges ?
- (*Bonus*) Expliquer pourquoi le tri à bulles renvoie bien une liste triée.

3. Un problème SQL/Python

Problème 1. *Info Mines-Ponts 2023 (Début)*

Marchons, marchons, marchons...

Ce sujet propose d'appliquer des techniques informatiques à l'étude de trois marches de nature différente :

- une marche concrète (partie I - Randonnée)
- une marche stochastique (partie II - Mouvement brownien d'une petite particule)
- une marche auto-évitante (partie III - Marche auto-évitante)

Les trois parties sont indépendantes. L'annexe à la fin du sujet (pages 8 à 10) fournit les canevas de code en Python que vous devrez reprendre dans votre copie pour répondre aux questions de programmation.

Partie I. Randonnée

Lors de la préparation d'une randonnée, une accompagnatrice doit prendre en compte les exigences des participants. Elle dispose d'informations rassemblées dans deux tables d'une base de données :

- la table **Rando** décrit les randonnées possibles – la clef primaire entière **rid**, son nom, le niveau de difficulté du parcours (entier entre 1 et 5), le dénivelé (en mètres), la durée moyenne (en minutes) :

rid	rnom	diff	deniv	duree
1	La belle des champs	1	20	30
2	Lac de Castellane	4	650	150
3	Le tour du mont	2	200	120
4	Les crêtes de la mort	5	1200	360
5	Yukon Ho !	3	700	210
...	...	...	...	...

- la table **Participant** décrit les randonneurs – la clef primaire entière **pid**, le nom du randonneur, son année de naissance, le niveau de difficulté maximum de ses randonnées :

pid	pnom	ne	diff_max
1	Calvin	2014	2
2	Hobbes	2015	2
3	Susie	2014	2
4	Rosalyn	2001	4
...	...	...	...

Donner une requête SQL sur cette base pour :

- ❑ **Q1** – Compter le nombre de participants nés entre 1999 et 2003 inclus.
- ❑ **Q2** – Calculer la durée moyenne des randonnées pour chaque niveau de difficulté.
- ❑ **Q3** – Extraire le nom des participants pour lesquels la randonnée n°42 est trop difficile.
- ❑ **Q4** – Extraire les clés primaires des randonnées qui ont un ou des homonymes (nom identique et clé primaire distincte), sans redondance.

L'accompagnatrice a activé le suivi d'une randonnée par géolocalisation satellitaire et souhaite obtenir quelques propriétés de cette randonnée une fois celle-ci effectuée. Elle a exporté les données au format texte CSV (*comma-separated values* – valeurs séparées par des virgules) dans un fichier nommé `suivi_rando.csv` : la première ligne annonce le format, les suivantes donnent les positions dans l'ordre chronologique.

Voici le début de ce fichier pour une randonnée partant de Valmorel, en Savoie, un bel après-midi d'été :

```
lat(°),long(°),height(m),time(s)
45.461516,6.44461,1315.221,1597496965
45.461448,6.444426,1315.702,1597496980
45.461383,6.444239,1316.182,1597496995
45.461641,6.444035,1316.663,1597496710
45.461534,6.443879,1317.144,1597496725
45.461595,6.4437,1317.634,1597496740
45.461562,6.443521,1318.105,1597496755
...
```

Le module `math` de Python fournit les fonctions `asin`, `sin`, `cos`, `sqrt` et `radians`. Cette dernière convertit des degrés en radians. La documentation donne aussi des éléments de manipulation de fichiers textuels :

```
fichier = open(nom_fichier, mode) ouvre le fichier, en lecture si mode est "r".
ligne = fichier.readline() récupère la ligne suivante de fichier ouvert en lecture avec open.
lignes = fichier.readlines() donne la liste des lignes suivantes.
fichier.close() ferme fichier, ouvert avec open, après son utilisation.
ligne.split(sep) découpe la chaîne de caractères ligne selon le séparateur sep : si ligne vaut "42,43,44",
alors ligne.split(",") renvoie la liste ["42", "43", "44"].
```

On souhaite exploiter le fichier de suivi d'une randonnée – supposé préalablement placé dans le répertoire de travail – pour obtenir une liste `coords` des listes de 4 flottants (latitude, longitude, altitude, temps) représentant les points de passage collectés lors de la randonnée.

À partir du canevas fourni en annexe et en ajoutant les `import` nécessaires :

❑ **Q5** – Implémenter la fonction `importe_rando(nom_fichier)` qui réalise cette importation en retournant la liste souhaitée, par exemple en utilisant certaines des fonctions ci-dessus.

On suppose maintenant l'importation effectuée dans `coords`, avec au moins deux points d'instantanés distincts.

❑ **Q6** – Implémenter la fonction `plus_haut(coords)` qui renvoie la liste (latitude, longitude) du point le plus haut de la randonnée.

❑ **Q7** – Implémenter la fonction `deniveles(coords)` qui calcule les dénivélés cumulés positif et négatif (en mètres) de la randonnée, sous forme d'une liste de deux flottants. Le dénivélé positif est la somme des variations d'altitude positives sur le chemin, et inversement pour le dénivélé négatif.

On souhaite évaluer de manière approchée la distance parcourue lors d'une randonnée. On suppose la Terre parfaitement sphérique de rayon $R_T = 6371$ km au niveau de la mer. On utilise la formule de haversine pour calculer la distance d du grand cercle sur une sphère de rayon r entre deux points de coordonnées (latitude, longitude) (φ_1, λ_1) et (φ_2, λ_2) :

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\varphi_2 - \varphi_1}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

On prendra en compte l'altitude moyenne de l'arc, que l'on complètera pour la variation d'altitude par la formule de Pythagore, en assimilant la portion de cercle à un segment droit perpendiculaire à la verticale.

□ **Q8** – Implémenter la fonction `distance(c1, c2)` qui calcule avec cette approche la distance en mètres entre deux points de passage. On décomposera obligatoirement les formules pour en améliorer la lisibilité.

□ **Q9** – Implémenter la fonction `distance_totale(coords)` qui calcule la distance en mètres parcourue au cours d'une randonnée.

— Partie I : Randonnée

```

1  # import Python à compléter...
2
3  # importation du fichier d'une randonnée
4  def importe_rando(nom_fichier):
5      # À compléter...
6
7  coords = importe_rando("suivi_rando.csv")
8
9  # donne le point (latitude, longitude) le plus haut de la randonnée
10 def plus_haut(coords):
11     # À compléter...
12
13 print("point le plus haut", plus_haut(coords))
14 # exemple : point le plus haut [45.461451, 6.443064]
15
16 # calcul des dénivelés positif et négatif de la randonnée
17 def deniveles(coords):
18     # À compléter...
19
20 print("dénivelés", deniveles(coords))
21 # exemple : denivelés [4.059999999999945, -1.175999999999309]
22
23 RT = 6371 # rayon moyen volumétrique de la Terre en km
24
25 # distance entre deux points
26 def distance(c1, c2):
27     # À compléter...
28
29 print("premier intervalle", distance(coords[0], coords[1]), "m")
30 # exemple : premier intervalle 16.230964254992816 m
31
32 # distance totale de la randonnée
33 def distance_totale(coords):
34     # À compléter...
35
36 print("distance parcourue", distance_totale(coords), "m")
37 # exemple : distance parcourue 187.9700904658368 m

```