



Concours Mathématiques et Physique, Physique et Chimie et Technologie Epreuve d'Informatique

Date : Mercredi 8 Juin 2022 Heure : 8 H Durée : 2 H Nombre de pages : 13
Barème : Partie I : 7 points Partie II : 6 points Partie III : 7 points

DOCUMENTS NON AUTORISÉS

L'USAGE DES CALCULATRICES EST INTERDIT

II FAUT RESPECTER IMPERATIVEMENT LES NOTATIONS DE L'ENONCE
VOUS POUVEZ EVENTUELLEMENT UTILISER LES FONCTIONS PYTHON DECRITES
À L'ANNEXE 2 (PAGE 13/13)

Le sujet comporte trois parties qui traitent les aspects suivants :

- **Partie I** : Programmation procédurale.
- **Partie II** : Programmation orientée objet.
- **Partie III** : Base de données relationnelle.

Il porte sur la même thématique présentée dans la description générale qui suit :

Description générale :

Un Quiz est un questionnaire permettant de tester une ou plusieurs compétence(s) générale(s) ou spécifique(s). Un Quiz est utilisé dans l'enseignement pour des fins d'évaluation formative et/ou sommative. Il est aussi utilisé dans les examens de certification. Différents types de questions existent, dont les plus utilisées sont : les Questions à Choix Multiples (QCM) auxquelles le candidat doit répondre en sélectionnant **au moins** une parmi une liste de propositions de réponses possibles associées à la question. Une ou plusieurs de ces proposition(s) est/sont correcte(s). Les autres sont des propositions erronées dites **distracteurs**.

La Figure 1, illustre à travers un exemple, les différentes parties d'une question QCM qui vise la compétence « Bases de Données Relationnelles ».

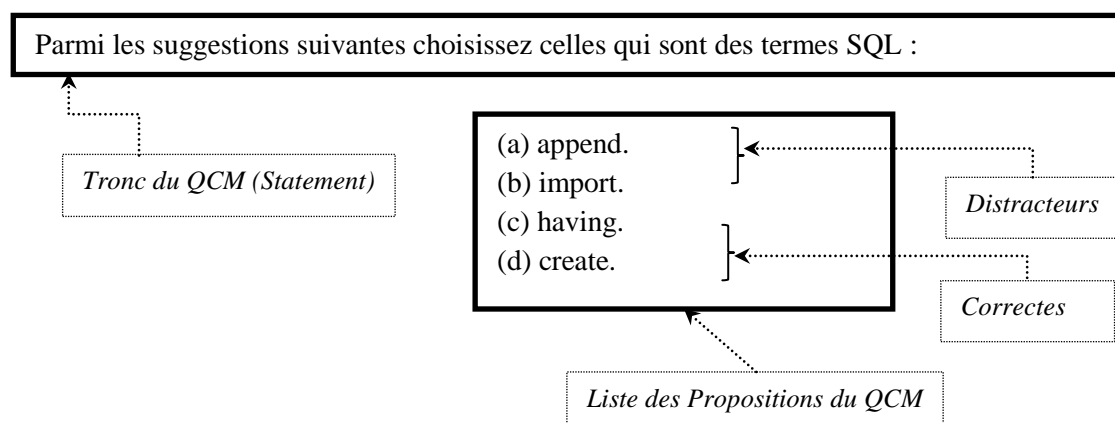


Figure 1 : Différentes parties d'une question QCM

Toute question appartenant à un Quiz est notée sur 1 point.

Au cours du passage du Quiz un candidat doit distinguer la ou les proposition(s) qu'il juge exacte(s) en les choisissant, de celles qu'il juge distracteurs.

À la fin du passage d'un Quiz, un score est calculé pour décider si un candidat est digne de certification.

Le score est la somme des notes obtenues dans toutes les questions du Quiz.

Le tableau suivant présente le barème adopté pour la notation d'une question QCM.

Tableau 1 : Barème de notation d'une question QCM.

Notation	Par réponse correcte	Par réponse incorrecte
k = nombre total de propositions c = nombre de propositions correctes	$+\frac{1}{c}$	$-\frac{1}{k-c}$

Pour la question de la Figure 1, la notation est calculée comme suit :

- c = nombre de propositions correctes = 2
 - k = nombre total de propositions = 4.
- \Rightarrow On accorde $+\frac{1}{c} = +0.5$ par proposition correcte choisie et $-\frac{1}{k-c} = -0.5$ par proposition incorrecte (distracteur) choisie.

Par exemple, pour deux candidats qui ont émis respectivement les deux réponses : "a c" et "a b d" auront dans l'ordre les scores $0 = (-0.5 + 0.5)$ et $-0.5 = (-0.5 - 0.5 + 0.5)$.

Partie I : Simulation numérique

L'objectif de cette partie est d'implémenter en Python, un mini moteur de recherche permettant de trouver les compétences visées par une question à Choix Multiples (QCM) d'un Quiz à partir du tronc de cette dernière et en se basant sur le modèle TF-IDF « Term-Frequency-Inverse-Document-Frequency ». Ci-après la description de la démarche du modèle TF-IDF, basée sur des exemples illustratifs et des définitions.

Démarche :

La démarche générale du modèle TF-IDF comporte les étapes suivantes :

- Étape 1 : Prétraitement « preprocessing » pour écarter les termes non pertinents.
- Étape 2 : Constitution du vocabulaire relatif au corpus ; l'espace de représentation des vecteurs tf-idf.
- Étape 3 : Vectorisation des documents du corpus.
- Étape 4 : Vectorisation de la question à choix multiples.
- Étape 5 : Utilisation d'une mesure de similarité pour ordonner les documents du corpus selon leur ressemblance avec la question.

Définitions et description des étapes :

Skill : un skill est une compétence spécifique. On convient de représenter les skills par un vecteur nommé **skills** de `str`.

Exemple 1:

```
skills = np.array(["python", "sql"])
```

Corpus : un corpus **D** est une collection de documents.

$$D = \{d_0, d_1, d_2, d_3, \dots, d_{n_{skills}-1}\}$$

Où chaque d_i contient le lexique relatif à un domaine de compétence spécifique $skills_i$. Par exemple, le document $d_1 = \text{"else while : in while"}$ est associé à la compétence $skills_1 = \text{"python"}$.

On convient de représenter un corpus par un dictionnaire nommé **corpus** où ses clés sont les compétences $skills_i$ et ses valeurs sont les documents associés d_i .

Exemple 2 :

```
corpus = {"python" : "else while : in while", "sql": "select insert select * in"}
```

Terme : un terme est un mot relatif à un champ lexical. Par exemple, "else" est un terme relatif au champ lexical de la compétence "python", tandis que "in" est un terme commun des deux champs lexicaux des compétences "python" et "sql".

À ce niveau, il est important d'introduire les formules propres aux mesures TF-IDF :

TF-IDF est une mesure qui quantifie le degré d'importance du terme t_j pour le document d_i (associée à la compétence $skills_i$). Elle est proportionnelle à la fréquence du terme t_j dans le document d_i et inversement proportionnelle au nombre d'autres documents d_k tel que $k \neq i$ contenant ce même terme t_j . Cette mesure est calculée comme suit :

$$tf_idf(t_j, skills_i) = tf_idf(t_j, d_i) = tf(t_j, d_i) \times idf(t_j)$$

Équation 1 : Formule TF-IDF

$$tf(t_j, skills_i) = tf(t_j, d_i) = \frac{\text{nombre d'occurrences de } t_j \text{ dans } d_i}{\sum_{t_k \in vocab} \text{nombre d'occurrences de } t_k \text{ dans } d_i}$$

Équation 2 : Formule TF

$$idf(t_j) = \log \left(\frac{n_{skills}}{\text{nombre de documents } d_k \text{ dans } D \text{ contenant } t_j} \right)$$

Équation 3 : Formule IDF

L'application de l'étape 1 sur chaque document d_i du **corpus**, produit un dictionnaire nommé **tcorpus** où ses clés sont les mêmes que **corpus** et ses valeurs sont des dictionnaires résultants dont les clés sont les termes épurés (après élimination des termes non alphabétiques) et les valeurs sont les tf relatifs calculés par l'équation 2.

Exemple 3 :

```
tcorpus = {"python" : {"else" : 1/4, "while" : 2/4, "in" : 1/4 }, "sql" : {"select" : 2/4, "in" : 1/4, "insert" : 1/4}}
```

À partir du dictionnaire **tcorpus**, on obtient directement d'une part les mesures $tf(t_j, d_i)$

Exemple 4 :

```
tf("in", "python") = tf("in", "sql") = 1/4
```

```
tf("else", "python") = 1/4 et tf("else", "sql") = 0
```

Et d'autre part calculer les mesures $idf(t_j)$ en appliquant l'équation 3.

Exemple 5 :

```
idf("else") = log(2/1) = log(2)
```

```
idf("in") = log(2/2) = log(1) = 0
```

Vocabulaire : un vocabulaire est composé de tous les termes distincts parus dans les documents du corpus. On convient de représenter un vocabulaire par le vecteur **vocab** de taille **nterms**.

$$\mathbf{vocab} = [t_0 \ t_1 \ \dots t_{nterms-1}]$$

L'application de l'étape 2 sur le **corpus** permet de produire le vocabulaire **vocab**.

Exemple 6 :

```
vocab = np.array(["else", "in", "insert", "select", "while"])
```

L'étape 3, génère à partir de **tcampus** et **vocab** la matrice **tf_idf_mat** de **nskills** lignes et **nterms** colonnes tel que :

$$tf_idf_mat_{i,j} = tf_mat_{i,j} \times idf_vect_j \quad \forall 0 \leq i < nskills \quad \forall 0 \leq j < nterms$$

Équation 4 : Matrice **tf_idf_mat**

où

$$tf_mat_{i,j} = tf(vocab_j, tcampus[skills_i]) \quad \forall 0 \leq i < nskills \quad \forall 0 \leq j < nterms$$

Équation 5 : Matrice **tf_mat**

et

$$idf_vect_j = idf(vocab_j) \quad \forall 0 \leq j < nterms$$

Équation 6 : Vecteur **idf_vect**

Exemple 7 :

```
tf_mat = np.array([[1/4, 1/4, 0, 0, 2/4],
                   [0, 1/4, 1/4, 2/4, 0]])

idf_vect = np.array([np.log(2), 0, np.log(2), np.log(2), np.log(2)])

tf_idf_mat = np.array([[np.log(2)/4, 0, 0, 0, 2 * np.log(2) / 4],
                       [0, np.log(2)/4, np.log(2)/4, 2 * np.log(2)/4, 0]])
```

L'étape 4, exploite la matrice **tf_idf_mat** et le vecteur **idf_vect** qui ont été appris à partir du **corpus** pour inférer les skills ciblés par une question d'un Quiz. Le texte du tronc de cette question **qtxt** subit les mêmes étapes de prétraitements que les documents du **corpus** afin de produire le dictionnaire **qdoc**, ce dernier est utilisé pour calculer le vecteur **qtf** en appliquant la formule suivante :

$$qtf_j = tf(vocab_j, qdoc) \quad \forall 0 \leq j < nterms$$

Équation 7 : Vecteur **qtf**

qtf est multiplié terme à terme avec le vecteur **idf_vect** afin de produire le vecteur **qtf_idf** :

$$qtf_idf_j = qtf_j \times idf_vect_j \quad \forall 0 \leq j < nterms$$

Équation 8 : Vecteur **qtf_idf**

Exemple 8 :

Pour une question de type QCM ayant le tronc :

```
qtxt = "while ... else: est une structure de :"
```

Les vecteurs associés sont :

```
qtf = np.array([1/2, 0, 0, 0, 1/2])
qtf_idf = np.array([np.log(2)/2, 0, 0, 0, np.log(2)/2])
```

L'étape 5, est l'étape finale d'inférence des compétences visées par le tronc d'une question donnée. Elle consiste à récupérer dans l'ordre les skills représentés par les lignes de la matrice **tf_idf_mat** selon leurs similarités par rapport au vecteur **qtf_idf**. Cette similarité est quantifiée en utilisant la similarité cosinus définie par :

$$\cos_sim(x, y) = \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|}$$

où :

$$\langle x, y \rangle = \sum_{i=0}^{n_{terms}-1} x_i \times y_i \quad : \text{Produit scalaire entre } x \text{ et } y$$

$$\|x\| = \sqrt{\langle x, x \rangle} \quad : \text{Norme euclidienne du vecteur } x.$$

Équation 9 : Similarité entre deux vecteurs

Les différentes similarités entre les lignes de la matrice **tf_idf_mat** et le vecteur **qtf_idf** sont stockées dans un vecteur nommé **qsim** de taille **nskills** tel que :

$$qsim_i = \cos_sim(tf_idf_mat_i, qtf_idf) \quad \forall 0 \leq i < nskills$$

Équation 10 : vecteur qsim

Exemple 9 :

Pour les vecteurs **qtf_idf** et la matrice **tf_idf_mat** :

`cos_sim(qtf_idf, tf_idf_mat[0])` donne 0.95

`cos_sim(qtf_idf, tf_idf_mat[1])` donne 0

Ceci résulte :

`qsim = np.array([0.95, 0])`

À partir du vecteur **qsim** on détermine le vecteur stochastique **sqsim** de taille **nskills** contenant les pourcentages de certitude des compétences en appliquant la transformation suivante :

$$sqsim_i = \frac{qsim_i}{\sum_{k=0}^{nskills-1} qsim_k} \quad \forall 0 \leq i < nskills$$

Équation 11 : Création d'un vecteur stochastique à partir d'un autre vecteur

Exemple 10 :

`sqsim = np.array([1.0, 0.0])`

Ceci conduit vers la conclusion que la question **qtxt** vise le domaine de compétence "python".

Travail demandé :

Dans la suite :

- Les fonctions demandées seront écrites en langage Python en utilisant impérativement la nomenclature donnée par le Tableau 2 (voir Annexe 1).
 - Le module `numpy` est importé par `import numpy as np`.
1. Écrire la fonction **is_alpha** qui prend en entrée **word**, retourne `True` si et seulement si **word** est formé uniquement par des lettres alphabétiques minuscules et `False` sinon.
 L'appel `is_alpha("python")` donne `True`.
 L'appel `is_alpha("ieee754")` donne `False`.

2. Écrire la fonction **preprocess** qui prend en entrée **txt** et **vocab** (paramètre optionnel, prend **None** par défaut). La fonction retourne le dictionnaire **doc** en appliquant les étapes suivantes :
 - a. transformer **txt** en minuscules puis le découper en une liste **words** contenant ses mots ;
 - b. éliminer de **words** les mots non alphabétiques si **vocab** est **None** sinon, éliminer les mots ne figurant pas dans **vocab** ;
 - c. construire **doc** où :
 - les clés sont les mots de **words**
 - les valeurs sont les fréquences divisées par le nombre total de mots dans **words** (voir Équation 2).

L'appel `preprocess('else while IN while a32')` donne : `{'else': 0.25, 'in': 0.25, 'while': 0.5 }`

L'appel `preprocess('else while IN while a32', np.array(["as", "else", "in"]))` donne : `{'else': 0.5, 'in': 0.5}`

3. Écrire la fonction **transform** qui prend en entrée **corpus** et retourne le dictionnaire **tcampus** en utilisant la fonction **preprocess** (voir Exemples 2 et 3).
4. Écrire la fonction **get_vocab** qui prend en entrée **tcampus** et retourne le vecteur **vocab** (voir Exemple 6).
5. Écrire la fonction **get_skills** qui prend en entrée **tcampus** et retourne le vecteur **skills** (voir Exemple 1).
6. Écrire la fonction **get_tf_vect** qui prend en entrée **vocab** et **doc**, retourne le vecteur **tf_vect** (voir Exemple 4).
7. Écrire la fonction **get_tf_mat** qui prend en entrée **vocab**, **skills** et **tcampus**, retourne la matrice **tf_mat** (voir Équation 5 & Exemple 7).
8. Écrire la fonction **get_idf_vect** qui prend en entrée la matrice **tf_mat** et retourne le vecteur **idf_vect** (voir Équation 6 & Exemple 7).
9. Écrire la fonction **cosine_sim** qui prend en entrée **tf_idf_mat** et **q_tf_idf**, calcule puis retourne le vecteur **sqsim** (voir Équations 9, 10 et 11 & Exemples 8, 9 et 10).
10. Écrire la fonction **find_skills** qui prend en entrée 5 paramètres : **qtxt**, **skills**, **vocab**, **tf_idf_mat**, **tf_idf** et **idf_vect**. La fonction retourne un dictionnaire **sim_skills** où :
 - les clés sont les domaines des compétences du vecteur **skills**.
 - les valeurs sont des pourcentages des similarités.**sim_skills** est construit comme suit :
 - a. construire **qdoc** le texte de la question prétraitée en utilisant la fonction **preprocess** appliquée sur **qtxt** et **vocab** ;
 - b. créer le vecteur **qtf** (voir Équation 7) ;
 - c. calculer le vecteur **qtf_idf** (voir Équation 8) ;
 - d. calculer le vecteur **sqsim** contenant la similarité entre **qtf_idf** et chaque ligne de **tf_idf_mat** ;
 - e. utiliser **sqsim** pour construire le dictionnaire **sim_skills**.

Partie II : Programmation orientée objet

L'objectif de cette partie est d'implémenter les deux classes :

- **SearchEngine**
- **QCM**

Description des classes :

- Classe **SearchEngine** :

Rôle :

Cette classe représente un moteur de recherche qui utilise le modèle TF-IDF pour représenter les glossaires d'un corpus de compétences et expose des méthodes permettant de chercher le(s) domaine(s) de compétences le(s) plus proche(s) d'une question donnée **qtxt**.

Attributs :

- **Vocab** : vecteur contenant les termes du corpus ;
- **Skills** : vecteur contenant les domaines de compétences du corpus ;
- **idf_vect** : vecteur contenant les **idf** des termes de **vocab** ;
- **tf_id_mat** : matrice contenant les **tf_idf** de tous les termes du **vocab** par rapport à chaque glossaire du corpus.

Méthodes :

- **__init__ (...)** : permet de créer une nouvelle instance à partir du dictionnaire **corpus** passée en paramètre.
- **ml_skills (...)** : qui à partir des deux paramètres **qtxt** et **n** ayant comme valeur par défaut 3 retourne une liste de **n** tuples tel chaque tuple contient le domaine de compétences le plus proches de **qtxt** ainsi que son pourcentage de similarités.
- **ml_skill(self, qtxt)** : qui à partir du paramètre **qtxt** retourne le domaine de compétences le plus proche.

- Classe **QCM** :

Rôle :

Cette classe représente une question d'un Quiz avec le tronc principal de la question (**statement**), les différentes propositions (**props**) ainsi que leurs véracités (**veracities**). Elle expose des méthodes permettant d'initialiser, de représenter textuellement et de calculer le score des tentatives de réponses.

Attributs :

- **skill** : domaine de compétence associé à la question ;
- **statement** : tronc principal de la question ;
- **props** : liste contenant les différentes propositions de réponses de la question ;
- **veracities** : liste contenant les véracités des propositions.

Méthodes :

- **__init__ (...)** : permet de créer une nouvelle instance à partir de 4 paramètres relatifs aux attributs.
- **__str__ (...)** : permet de retourner une représentation textuelle de la question conformément au format illustré dans la Figure 1.
- **score (...)** : qui à partir du paramètre **ans** contenant les lettres alphabétiques séparées par des espaces associées aux propositions choisies par le candidat retourne le score de sa réponse selon la méthode de notation du Tableau 1.

Travail demandé :

Pour la classe **SearchEngine** :

1. Écrire la méthode `__init__`.
2. Écrire la méthode `ml_skills`.
3. Écrire la méthode `ml_skill`.

Pour la classe **QCM** :

4. Écrire la méthode `__init__`.
5. Écrire la méthode `__str__`.
6. Écrire la méthode `score`.

Partie III : Base de Données Relationnelle

Un cabinet de formation dispose d'une base de données pour la gestion de son référentiel de questions d'évaluation et de certification.

Il y a lieu de considérer les points suivants :

- Toutes les questions sont de type **QCM** y compris les questions de type Vrai/Faux.
- **QCM** peut avoir des propositions de réponses toutes entièrement correctes mais jamais toutes fausses.
- Une question **QCM** de type Vrai/faux, supporte uniquement deux propositions de réponses, exclusivement une correcte, et une fausse.

Le schéma relationnel de la base de données est défini comme suit :

▪ **Quiz** (idQuiz, description, duree, dateCreation, dateFin)

La table **Quiz** enregistre les informations relatives à un Quiz.

- idQuiz : identifiant du Quiz, un nombre de type entier, clé primaire ;
- description : décrit l'intitulé de l'examen de certification, de type chaîne ;
- duree : durée officielle du Quiz, de type entier exprimée en minutes ;
- dateCreation : date de création relative à la mise en vigueur du Quiz, de type date ;
- dateFin : date fin de mise en vigueur du Quiz, de type date ;

▪ **Question** (idQs, tronc, dateCr)

La table **Question** enregistre les informations générales relatives à une question.

- idQs : identifiant d'une question, clé primaire, de type entier ;
- tronc : énoncé de la question, de type chaîne ;
- dateCr : date de création de la question, de type date ;

▪ **Proposition** (#idQs, idProp, textProp, veracite)

La table **Proposition** enregistre les propositions de réponses relatives à une question (les réponses correctes et les distracteurs).

- idQs : identifiant d'une question, clé étrangère qui fait référence à la table **Question** ;
- idProp : identifiant d'une réponse proposée relativement à la question identifiée par idQs, de type entier ;
- textProp : réponse proposée, de type chaîne ;
- veracite : indique l'état de la réponse proposée, de type entier. 1 si la réponse proposée est vraie, 0 sinon ;

NB : idQs et idProp ensembles forment la clé primaire de la table **Proposition**.

▪ **DetailQuiz** (#idQuiz, #idQs)

La table **DetailQuiz** enregistre les questions relatives à un Quiz ;

- idQuiz : identifiant d'un Quiz, clé étrangère qui fait référence à la table **Quiz** ;
- idQs : identifiant d'une question, clé étrangère qui fait référence à la table **Question** ;

NB : idQuiz et idQs ensembles forment la clé primaire de la table **DetailQuiz**.

III.1. Algèbre relationnelle

Travail demandé :

Pour chacune des requêtes algébriques suivantes décrire le résultat attendu.

1.
$$\prod_{tronc, textProp} \left(\sigma_{\substack{veracite=1 \\ idQs}} \left(\mathbf{Question} \triangleright \triangleleft \mathbf{Proposition} \right) \right)$$
2.
$$\prod_{idQs} \left(\sigma_{idQuiz=3} (\mathbf{DetailQuiz}) \right) \cap \prod_{idQs} \left(\sigma_{idQuiz=20} (\mathbf{DetailQuiz}) \right)$$

III.2. SQL

Travail demandé :

Exprimer en SQL les requêtes suivantes permettant de :

1. Créer la table **Proposition** en respectant les contraintes d'intégrité spécifiées dans le schéma relationnel.

Dans la suite on suppose que les tables sont créées et remplies.

2. Ajouter la question d'identifiant 17546 au Quiz d'identifiant 228.
3. Supprimer les questions dont les dates de création sont antérieures à l'année 2012.
4. Déterminer les informations des Quiz qui ne seront plus en vigueur à partir d'aujourd'hui sachant qu'une date peut être comparée à la valeur de retour de la fonction `current_date` qui représente la date courante.
5. Déterminer les identifiants, troncs et propositions des questions qui n'ont jamais été assignées dans des Quiz.
6. Déterminer les troncs des questions QCM qui proposent plus que 4 réponses possibles.
7. Déterminer les identifiants des Quiz qui ne proposent pas des questions de type Vrai/Faux.

III.3. Sqlite3

Dans ce qui suit on propose d'implémenter en Python des fonctions permettant d'élaborer des statistiques sur la base de certification. Pour cela on ajoute au schéma relationnel précédent les tables suivantes :

▪ **Candidat** (idCand, nom, prenom, email, pwd)

La table **Candidat** enregistre les informations relatives aux candidats.

- idCand : identifiant du candidat de type chaîne, clé primaire ;
- nom : nom du candidat de type chaîne ;
- prenom : prénom du candidat de type chaîne ;
- email : adresse mail du candidat de type chaîne ;
- pwd : mot de passe du candidat de type chaîne ;

▪ **Tentative** (voucher, #idCand, #idQuiz, datehDeb, datehFin, scoreG)

La table **Tentative** enregistre les informations générales qui concernent un passage d'un examen de certification par un candidat suite à l'achat d'un voucher.

- voucher : identifiant délivré à tout candidat ayant payé des frais et désirant passer une certification, de type chaîne, clé primaire ;
- idCand : identifiant d'un candidat, clé étrangère qui fait référence à la table **Candidat** ;
- idQuiz : identifiant d'un Quiz, clé étrangère qui fait référence à la table **Quiz** ;
- datehDeb : date et heure début de passage de l'examen de certif par le candidat, de type datetime ayant le format 'AAAA-MM-JJ hh:mm:ss' ;
- datehFin : date et heure fin de passage de l'examen de certif par le candidat, de type datetime ayant le format 'AAAA-MM-JJ hh:mm:ss' ;
- scoreG : score obtenu par le candidat sur la base de ses réponses aux questions proposées, de type réel ;

▪ **ReponseCandidat**(#voucher, #idQs, #idProp)

La table **ReponseCandidat** enregistre les propositions choisies explicitement par le candidat.

- voucher : identifiant d'une tentative, clé étrangère qui fait référence à la table **Tentative** ;
- idQs : identifiant d'une question, clé étrangère qui fait référence à la table **Question** ;
- idProp : identifiant d'une proposition, clé étrangère qui fait référence à la table **Proposition** ;

Conditions à considérer :

- Un candidat obtient un voucher pour passer un Quiz (examen de certification).
- Un Quiz passé par un candidat suite à l'achat d'un voucher, est appelé tentative.
- Un candidat peut acheter plusieurs vouchers pour le passage de divers examens de certifications.
- Un Quiz passé par un candidat suite à l'achat d'un voucher, est appelé tentative.
- Pour être évaluée, une tentative de candidat doit être validée et soumise au cours du délai de temps publié pour l'examen de certification.
- Dépassant la durée prévue pour un examen de certification sans soumission, le candidat est considéré défaillant.
- Pour répondre aux questions du Quiz, le candidat doit choisir les réponses qu'il juge correctes.
- Toute question appartenant à un Quiz est notée sur 1 point.
- Le calcul de note d'un candidat pour toute question du Quiz est fait sur la base du barème expliqué dans le Tableau 1.
- Le score d'un candidat dans une tentative est un pourcentage calculé sur la base de la somme des notes obtenues pour toutes les questions du Quiz et divisé par le nombre des questions.

Travail demandé :

On suppose que ces nouvelles tables sont créées et remplies.

1. Écrire la fonction **Score_Question** qui prend en paramètre **Cur**, l'identifiant **v** d'un voucher et l'identifiant **idQ** d'une question, calcule la note obtenue par le candidat.
2. Écrire la fonction **Score_Quiz** qui prend en paramètre **Cur**, l'identifiant **v** d'un voucher puis met à jour le score obtenu par le candidat.
3. Écrire la fonction **Stat_Certif** qui prend en paramètre **Cur** et renvoie une liste de tuples. Chaque tuple est constitué de cinq éléments dont l'identifiant du Quiz et quatre indicateurs de performance : le score maximum qui a été obtenu, le score moyen, le minimum et l'écart type.

La formule de l'écart type est :
$$\sigma = \sqrt{\frac{\sum_{i=1}^n |x_i - \mu|^2}{n}}$$

x_i : désigne le score d'un candidat pour un Quiz.

μ : désigne la moyenne des scores des candidats qui ont passé ce Quiz.

n : désigne l'effectif des candidats qui ont passé ce Quiz.

ANNEXE 1

Tableau 2 : Nomenclature.

Nom	Type	Description
<i>skill</i>	str	un domaine de compétence relatif à certaines questions QCM.
<i>word</i>	str	un terme potentiel.
<i>txt</i>	str	un document brut (glossaire d'un domaine de compétence).
<i>doc</i>	dict	un document prétraité représenté sous forme d'un dictionnaire où les clés sont les termes et les valeurs sont les fréquences (tf) associées.
<i>corpus</i>	dict	un corpus brut représenté sous forme d'un dictionnaire où chaque clé est un str <i>skill</i> et chaque valeur est un str <i>txt</i> .
<i>tcorpus</i>	dict	un corpus prétraité représenté sous forme d'un dictionnaire où chaque clé est un str <i>skill</i> et chaque valeur est un dictionnaire doc.
<i>vocab</i>	np.ndarray	un vecteur à composantes de type str, contenant tous les termes figurant dans les différents documents triés en ordre ascendant où chaque terme est représenté une seule fois.
<i>nterms</i>	int	la taille du vecteur <i>vocab</i> : le nombre total de termes dans les glossaires du corpus
<i>skills</i>	np.ndarray	un vecteur à composantes de type str, contenant tous les domaines de compétences triés en ordre croissant où chaque domaine de compétence (skill) est représenté une seule fois.
<i>nskills</i>	int	taille du vecteur <i>skills</i> : le nombre total de domaines de compétences.
<i>tf_vect</i>	np.ndarray	un vecteur à <i>nterms</i> composantes de type float, contenant les tf de tous les termes de <i>vocab</i> relativement à un document/question (doc).
<i>tf_mat</i>	np.ndarray	une matrice à <i>nskills</i> lignes et <i>nterms</i> colonnes, où chaque ligne i contient le <i>tf_vect</i> relatif au document associé à la compétence <i>skills[i]</i> dans le corpus.
<i>idf_vect</i>	np.ndarray	un vecteur à <i>nterms</i> composantes de type float, contenant les idf de tous les termes de <i>vocab</i> par rapport à tous les documents du corpus.
<i>tf_idf_mat</i>	np.ndarray	une matrice à <i>nskills</i> lignes et <i>nterms</i> colonnes, où chaque ligne i représente le vecteur du document associé au domaine de compétence <i>skills[i]</i> selon le modèle tf_idf.
<i>qtxt</i>	str	le texte brut d'une question d'un QCM.
<i>qdoc</i>	dict	le résultat du prétraitement du texte d'une question : représenté sous forme d'un dictionnaire où les clés sont les termes de <i>vocab</i> et les valeurs sont les fréquences (tf) associées.
<i>qtf</i>	np.ndarray	un vecteur à <i>nterms</i> composantes contenant les tf des termes de <i>vocab</i> par rapport à une question.
<i>qtf_idf</i>	np.ndarray	un vecteur à <i>nterms</i> composantes réelles décrivant une question q d'un QCM selon le modèle tf_idf, formellement.
<i>qsim</i>	np.ndarray	un vecteur stochastique à <i>nskills</i> composantes réelles positives contenant le degré de similarité entre le vecteur <i>qtf_idf</i> d'une question et chaque ligne de la matrice <i>tf_idf_mat</i> relative au corpus.

ANNEXE 2– Quelques Fonctions/Méthodes Python

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

Module numpy

- **M.shape** ou **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 **ndarray** dont les composantes sont initialisées à partir de la liste passée en entrée.
- **M.sum()** retourne la somme de tous les éléments d'un **ndarray**.
- **np.log(M)** retourne un **ndarray** de même forme que **M** contenant le logarithme terme à terme des composantes de **M**.
- **v1.dot(v2)** ou **np.dot(v1, v2)** retourne le produit scalaire entre deux vecteurs **v1** et **v2** (**ndarray** de dimension 1).
- **v1 * v2** retourne le produit terme à terme de deux **ndarray**.
- **np.linalg.norm(v)** retourne la norme du vecteur **v**.

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.
- **ord(s)** retourne le code de la lettre **s**.
- **chr(v)** retourne la lettre ayant le code **v**.
- **min(it)** retourne la valeur minimale de l'itérable **it**.
- **max(it)** retourne la valeur maximale de l'itérable **it**.
- **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.index(val)** retourne l'indice de la première occurrence **val** de la liste **lst**.
- **source.lower()** retourne un nouvel str contenant les mêmes lettres que **source** en minuscule.
- **source.isalpha()** retourne un nouvel bool indiquant si le str **source** est formé par des lettres alphabétiques uniquement.
- **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) où **k** est une clé du dictionnaire **d** et **v** est la valeur associée.
- **s.add(obj)** ajoute **obj** à un ensemble **s** (instance de la classe **set**).
- **s.remove(obj)** supprime **obj** de l'ensemble **s** (instance de la classe **set**).
- **s1.union(s2)** retourne l'ensemble union des deux ensembles **s1** et **s2**.
- **s1.difference(s2)** retourne l'ensemble différence des deux ensembles **s1** et **s2**.



Concours Mathématiques et Physique, Physique et Chimie et Technologie

Correction Epreuve d'Informatique

BAREME PARTIE I: 45 + PARTIE II: 30 + PARTIE III : 45 TOTAL 120/100

Partie I (45 points)

1. (5 points)

```
def is_alpha(word):  
    ascii_lowercase = {chr(i) for i in range(chr('a'),  
    chr('z')+1)}  
    return set(word) <= ascii_lowercase
```

version 2

```
def is_alpha(word):  
    for c in word:  
        if not 'a' <= c <= 'z':  
            return False  
    return True
```

2. (5 points)

```
def preprocess(txt, vocab):  
    txt = txt.lower()  
    if vocab is None:  
        predicate = is_alpha  
    else:  
        v = set(vocab)  
        predicate = lambda x : x in v  
    lst = [term for term in txt.split() if predicate(term)]  
    return {term: lst.count(term)/len(lst) for term in lst}
```

3. (5 points)

```
def transform(corpus):  
    return {skill:preprocess(doc) for skill, doc in  
    corpus.items() }
```

4. (5 points)

```
def get_vocab(tcorpus):  
    s = set()  
    for d in tcorpus.values():  
        s |= set(d)  
    return np.array(sorted(s))
```

5. (5 points)

```
def get_skills(tcorpus):  
    return np.array(sorted(tcorpus.keys()))
```

6. (5 points)

```
def get_tf_vect(vocab, doc):  
    return np.array([doc.get(term, 0) for term in vocab])
```

7. (5 points)

```
def get_tf_mat(vocab, skills ,tcorpus):  
    return np.array([get_tf_vect(vocab, tcorpus[skill])for skill  
in skills])
```

8. (5 points)

```
def get_idf_vect(tf_mat):  
    nskills , nterms = tf_mat.shape  
    return np.array([np.log(nskills/(tf_mat[:,j]!=0).sum()) for j  
in range(nterms)])
```

9. (2.5 points)

```
def cosine_sim(x,y):  
    if x.ndim == 1:  
        return x.dot(y) / np.sqrt(x.dot(x)) * np.sqrt(y.dot(y))  
    else:  
        sim = np.array([cosine_sim(v, y) for v in x])  
        return sim / sim.sum()
```

10. (2.5 points)

```
def find_skills(question , skills, vocab, tf idf mat, idf vect):  
    qtf idf = get tf vect(vocab, preprocess(question , vocab)) *  
idf vect  
    sim = cosine_sim(tf idf mat, qtf idf) * 100  
    return dict(zip(skills, sim))
```

Partie II (30 points = 5 points * 6 questions)

(5 points par question)

```
class SearchEngine:  
    def init (self, corpus):  
        corpus = transform(corpus)  
        self.vocab = get_vocab(corpus)  
        self.skills = get_skills(corpus)  
        tfm = get_tf_mat(self.vocab, self.skills, corpus)  
        self.idf_vect = get_idf_vect(tfm)  
        self.tf_idf_mat = tfm * self.idf_vect
```

```

def ml_skills(self, question, n = 3):
    d = find_skills(question, self.skills, self.vocab,
self.tf_idf_mat, self.idf_vect)
    skills = sorted(d, key = lambda skill : d[skill],
reverse = True)
    return tuple( (skill, d[skill]) for skill in skills[:n])

def ml_skill(self, question):
    return self.ml_skills(question, 1)[0]

class QCM:

    def __init__(self, skill, statement, props, veracities):
        self.skill = skill
        self.statement = statement
        self.props = props
        self.veracities = veracities

    def __str__(self):
        res = self.statement + "\n"
        res += "\n".join("\t({}) {}".format(chr(ord("a")+i), prop
) for i, prop in enumerate(self.props))
        return res

    def score(self, ans):
        c = sum(self.veracities)
        k = len(self.veracities)
        idx = [ord(c)-ord('a') for c in set(ans)]
        return sum((1/c) if self.veracities[i] else (-1/(k-
c)) for i in idx)

```

Partie III : (45 points)

III.1. Algèbre relationnelle (5 points = 2.5 points par question)

$$1. \prod_{trunc, textProp} \left(\sigma_{\substack{veracite=1 \\ idQs}} \left(\text{Question} \triangleright \triangleleft \text{Proposition} \right) \right)$$

La liste des questions avec leurs réponses correctes (le corrigé de la base des questions)

$$2. \prod_{idQs} \left(\sigma_{idQuiz=3}(\mathbf{DetailQuiz}) \right) \cap \prod_{idQs} \left(\sigma_{idQuiz=20}(\mathbf{DetailQuiz}) \right)$$

Les identifiants des questions affectées à la fois aux quiz 3 et 20.

Ou bien

Les identifiants des questions communes entre les quiz 3 et 20.

III.2. SQL (25 points)

1. (2.5 points)

```
create table Proposition(IdQs int,IdProp int,textProp text,veracite
int,primary key(IdQs,IdProp),foreign key(IdQs) references
Question(IdQs),check (veracite=0 or veracite=1))
```

2. (2.5 points)

```
insert into DetailQuiz values(228,17546)
```

3. (2.5 points)

```
delete from question where dateCr<"2012%"
```

4. (2.5 points)

```
select *
from quiz
where datefin <current_date
```

5. (5 points)

```
select q.idQs,tronc,textprop
from question q,proposition p
where q.idQs=p.idQs and q.idqs not in (select idQs from detailQuiz)
```

6. (5 points)

```
select tronc
from proposition p,Question q
where p.idQs=q.idQs
group by p.idQs
having count(*)>=4
```

7. (2.5 points)

```
select distinct idQuiz
from detailQuiz
where idQuiz not in(select idQuiz
                    from proposition p,detailquiz dq
                    where p.IdQs=dq.IdQs and
                    textprop in("Vrai","Faux"))
```

III.3. Sqlite3 (15 points)

1. (5 points)

```
def Score_Question(Cur,V,IdQ):
    Cur.execute("select veracite
                from Proposition P
                where IdQs={}".format(IdQ))
    Lv=Cur.fetchall()
    Lv=[i[0] for i in Lv]
    c=Lv.count(1)
    k=len(Lv)
    Cur.execute("select Rc.IdProp,veracite
                from ReponseCandidat Rc,Proposition P
                where Voucher='{}' and P.IdQs=Rc.IdQs
                and P.IdProp=Rc.IdProp
                and P.IdQs={}".format(V,IdQ))
    L=Cur.fetchall()
    ScoreQ=0
    for i in L:
        if i[1]==1:
            ScoreQ+=1/c
        else:
            ScoreQ-=1/(k-c)

    return ScoreQ
```

2. (5 points)

```
def Score_Quiz(Cur,V):
    Cur.execute("select distinct IdQs
                from ReponseCandidat
                where Voucher='{}' ".format(V))
    L=cur.fetchall()
    Lq=[i[0] for i in L]
    ScoreQz=0
    for i in Lq:
        ScoreQz+=Score_Question(Cur,V,i)
    Cur.execute("update Tentative
                set scoreG={}
                where Voucher='{}' ".format(ScoreQz,V))
```

3. (5 points)

```
from math import sqrt
def Stat_Certif(Cur):
    Cur.execute("select idQuiz,max(scoreG),avg(scoreG)
                from Tentative group by idQuiz ")
    L=cur.fetchall()
    L1=[]
    for i in L:
        Cur.execute("select scoreG
                    from Tentative
                    where idQuiz={}
                    and scoreG is not null".format(i[0]))
        LScoreGQz=Cur.fetchall()
        s=0
        for j in LScoreGQz:
            s+=abs(j[0]-i[1])**2

        Ecart=sqrt(s/len(L))
        i+=(Ecart,)
        L1.append(i)
    return(L1)
```