POLYNÉSIE – 2026 – sujet1

EXERCICE 3 (8 POINTS)

Cet exercice porte sur le paradigme « Programmation orientée objet », les bases de données et les arbres binaires de recherche.

Introduction : au cœur du développement technologique, les interfaces conversationnelles, ou *chatbots*, jouent un rôle de plus en plus important dans l'interaction avec les utilisateurs. La start-up OpenChat souhaite développer un chatbot qui utilise une base de données pour stocker des questions et les réponses associées.

PARTIE 1 : STRUCTURES DE DONNÉES

On souhaite créer une classe `Chatbot` ayant un attribut `base_donnees` initialisé comme un dictionnaire vide.

On utilisera ensuite l'attribut `base_donnees` qui permettra de stocker les questions en tant que clés du dictionnaire et les réponses à ces questions en tant que valeurs associées à ces clés.

On définit également deux méthodes permettant de faire fonctionner le chatbot :

- une méthode `ajouter_question_reponse` permet l'ajout dans le dictionnaire `base_donnees` de la valeur `reponse` associée à la clé `question` ;
- une méthode `repondre` renvoie la réponse associée à une question.

1. Recopier et compléter les lignes 3 et 6 dans le constructeur de la classe `Chatbot` et dans la méthode `ajouter_question_reponse`.

```
1 class Chatbot:
2     def __init__(self):
3         self.base_donnees = ...   # à compléter
4
5     def ajouter_question_reponse(self, question, reponse):
6         self.base_donnees[question] = ...   # à compléter
7
8     def repondre(self, question):
9         return self.base_donnees[question]
```

Nous allons maintenant illustrer quelques interactions possibles entre un utilisateur et le chatbot, en se basant sur la classe précédemment définie.

2. Écrire les lignes de code permettant d'instancier un objet `Bot1` de la classe `Chatbot` et d'ajouter à son dictionnaire l'interaction :

   Question : « Bonjour, comment vas-tu ? » – Réponse : « Bonjour, je vais bien, merci. »

3. Écrire les lignes de code permettant d'obtenir la réponse à la question « Bonjour, comment vas-tu ? ».

PARTIE 2 : INTERACTION AVEC LA BASE DE DONNÉES

Cette partie porte sur les bases de données. On pourra utiliser les mots clés SQL suivants : AND, FROM, INSERT, INTO, JOIN, OR, ON, SELECT, SET, UPDATE, VALUES, WHERE.

On considère une table `questions_reponses` avec les colonnes `id`, `question`, `reponse` et `mot_cle`.

4. Associer à chaque mot clé sa définition :
   - mots clés : SELECT – INSERT – UPDATE ;
   - définitions : ajouter une donnée – extraire des données – mettre à jour une table.
5. Définir ce qu'est une clé primaire.
6. Indiquer quel attribut (éventuellement plusieurs) serait le plus adapté pour être une clé primaire dans la table `questions_reponses` et justifier votre choix.
7. Écrire une requête permettant de récupérer toutes les questions comportant comme mot-clé « python ».
8. Écrire une requête pour insérer la nouvelle question-réponse « Date de la bataille de Marignan ? », « 1515 » associée au mot-clé « Histoire ». On prendra comme identifiant id = 15.
9. Écrire la requête pour mettre à jour la réponse à la question d'id = 12 qui devient « Oui ».

PARTIE 3 : REPRÉSENTATION DE LA RECHERCHE AVEC LES ARBRES

On considère cet exemple d'un arbre binaire de recherche (ABR) basé sur le mot-clé des questions :

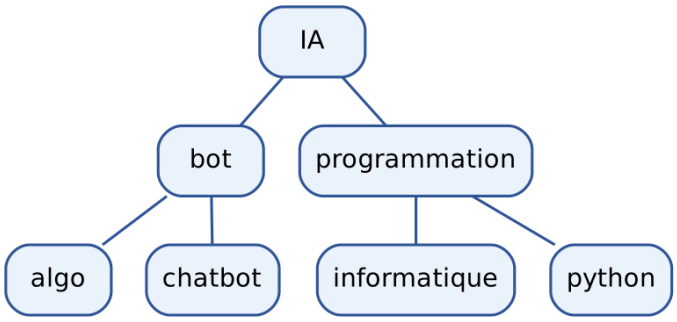


Figure 1. ABR des mots-clés

10. Indiquer quel type de parcours donnerait, dans cet ordre : IA, bot, programmation, algo, chatbot, informatique, python.
11. Indiquer dans quel ordre apparaîtraient les mots-clés avec un parcours en profondeur infixe sur l'ABR illustré.
12. Indiquer dans quel ordre apparaîtraient les mots-clés pour un parcours en profondeur postfixe (ou suffixe).
13. Indiquer quel parcours permettrait d'obtenir les mots-clés dans l'ordre alphabétique.

14. Compléter la méthode de recherche dans l'ABR :

```
class Noeud:

    def __init__(self, cle):
        self.cle = cle
        self.gauche = None
        self.droit = None

    def recherche(arbre, cle):
        if arbre is None or arbre.cle == ...:   # à compléter
            return arbre
        if cle < arbre.cle:
            return recherche(arbre. ..., cle)   # à compléter
        else:
            return recherche(arbre. ..., cle)   # à compléter
```