Ce document a été mis en ligne par l'organisme FormaV®.

Toute reproduction, représentation ou diffusion, même partielle, sans autorisation préalable, est strictement interdite.

Pour consulter tous nos sujets d'annales et corrigés du Bac Général Maths + NSI, veuillez visiter :
bac-maths-nsi.fr/annales-sujets-et-corriges

Amérique du sud - septembre 2024 - sujet 1 (corrigé)

Exercice 1 (Programmation Python, POO, bases de données et requêtes SQL)
Partie A

1. L'instruction suivante convient : `self._score += nb_points`
2. `'THEBEST'` étant le premier joueur de la liste `joueurs`, l'instruction suivante convient : `joueurs[0].augmenter_score(3)`
3. L'instruction suivante convient : `return self.pseudo, self.score`
4. La variable `tableau_scores` est un dictionnaire où les clés sont les pseudos des joueurs et les valeurs leur score associé.
5. On a le dictionnaire suivant : `{'THEBEST': 3, 'PAS2BOL': 0}`
6. Le dictionnaire ne contiendra que l'association correspondant au dernier joueur de la liste qui porte ce pseudo.
7. La fonction suivante convient :

```
def chercher_gagnant(tableau_scores):
    score_max, pseudo_max = -1, ''
    for pseudo in tableau_scores:
        score = tableau_scores[pseudo]
            if score > score_max:
                score_max, pseudo_max = score, pseudo
    return pseudo_max
```

Partie B

8.
   - L'enregistrement 1 appartient à la table `jeux`.
   - L'enregistrement 2 appartient à la table `scores`.
   - L'enregistrement 3 appartient à la table `joueurs`.
9. En cas de piratage de la base, les mots de passe seront directement accessibles.
10. Avec le schéma relationnel proposé, deux joueurs peuvent avoir le même pseudo puisque cet attribut ne sert pas de clé primaire.
11. On a les clés étrangères suivantes :
    - `id_jeu` qui fait référence à l'attribut `id_jeu` de la table `jeux` ;
    - `id_joueur` qui fait référence à l'attribut `id_joueur` de la table `joueurs`.
12. Avec le schéma relationnel proposé, il est tout à fait possible qu'un joueur puisse jouer à plusieurs jeux car la table `scores` prend en compte à la fois le joueur et les jeux auxquels il joue.
13. On obtient un message d'erreur après l'exécution du deuxième `INSERT INTO` car il y a violation de la contrainte d'unicité induite par la clé primaire `id_joueur`. L'insertion du second joueur est en effet impossible car il existe déjà une entrée dans la table avec la clé primaire `id_joueur` valant `1035`.
14. La requête suivante convient :

```
INSERT INTO scores VALUES (24, 1042, 0)
```

    en supposant qu'il existe dans la table `joueurs` un joueur d'identifiant `1024` et dans la table `jeux` un jeu d'identifiant `24`.
15. La requête suivante convient :

```
UPDATE scores SET score_joueur = 8 WHERE id_joueur = 1042 AND id_jeu = 24
```

Exercice 2 (Programmation Python, dictionnaires et algorithmique)
Partie A

1. On a les signatures suivantes :
   - `BEAU : ABEU`
   - `PIRATE : AEIPRT`
   - `PATRIE : AEIPRT`
2. Le tableau ci-dessous donne les valeurs prises par `i`, `k` (en sortie de boucle `while`) et `chaine`

`i`	`k` (à la fin du `while`)	`chaine`
		`'P'`
1	0	`'LP'`
2	0	`'ALP'`
3	2	`'ALNP'`
4	4	`'ALNPT'`
5	1	`'AELNPT'`

   Les affichages produits sont ceux obtenus dans la dernière colonne du tableau.
3. La fonction suivante convient :

```
def sont_annagrammes (mot1, mot2):
    return signature(mot1) == signature(mot2)
```

4. La fonction `mystere` prend en paramètre une liste de mots et renvoie un dictionnaire dont les clés sont les différentes signatures et les valeurs la liste des mots ayant ce !e signature donc dans le cas présent, le résultat (à l'ordre près) est :

```
{'AEIPRT' : ['PIRATE', 'PATRIE', 'PARTIE'],
 'ABEU' : ['AUBE', 'BEAU'],
 'AEEHLPT' : ['ELEPHANT']}
```

5. La fonction suivante convient :
   - Version itérative :

```
def inserer_lettre(mot, lettre, pos):
    res = ''
    for i in range(pos):
        res += mot[i]
    res += lettre
    for i in range(pos,len(mot)):
        res += mot[i]
    return res
```

   - Version récursive :

```
def inserer_lettre(mot, lettre, pos):
    if pos == 0 :
        return lettre+mot
    else :
        return mot[0] + inserer_lettre(mot[1 :], lettre, pos-1)
```

Partie B

6. Le script suivant convient :

```
score = 0
for lettre in mot: # ou for i in range(len(mot)) :
    score += score_lettres[lettre] # ou score += score_lettres[mot[i]]
return score
```

7. La fonction suivante convient :

```
def meilleur_mot(liste_mots):
    meilleur_mot, meilleur_score = 0, score_mot(liste_mots[0])
    for i in range(1,len(liste_mots)):
        score = score_mot(liste_mots[i])
        if score > meilleur_score:
            meilleur_mot, meilleur_score = i, score
    return liste_mots[meilleur_mot]
```

8. On complète de la façon suivante :
   - Ligne 6 : `while m < 7:`
   - Ligne 9 : `tirage = lettres[nb_alea]`
9. Le `'r'` (pour *read*) signifie que l'accès au fichier se fait en mode lecture (le fichier existe et on cherche à lire son contenu).
10. Le script suivant convient :

```
chevalet = pioche([])
mots_possibles = []
liste_signatures = signatures_depuis_tirage(chevalet)
# détermination de tous les mots quils est possible de jouer
dico = mystere(lecture('mots_valides.txt'))
for signature in liste_signatures :
    if signature in dico:
        mots_possibles += dico[signature]
# détermination dun mot optimal
mot_optimal = meilleur_mot(mots_possibles)
print('Un mot optimal : ', mot_optimal)
# mise à jour du chevalet
depot = [lettre for lettre in mot_optimal]
chevalet = chevalet_apres_depot(chevalet, depot)
chevalet = pioche(chevalet)
```

Exercice 3 (Graphes)
Partie A

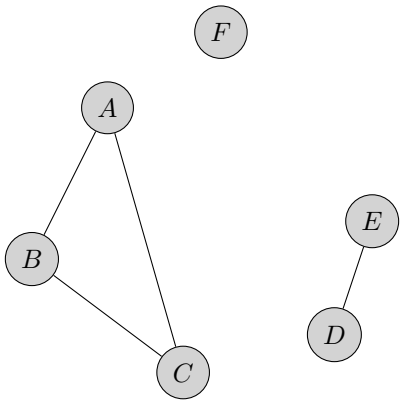
1. Le graphe à 7 sommets et 9 arêtes.
2. On a la matrice d'adjacence suivante :

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

3. On donne le résultat sous la forme d'un dictionnaire : les clés sont les sommets et les valeurs associées une liste contenant les voisins du sommet considéré :

```
{'A': ['B', 'E'],
 'B': ['A', 'D', 'G'],
 'C': ['G'],
 'D': ['B', 'G'],
 'E': ['A', 'F', 'G'],
 'F': ['E', 'G'],
 'G': ['B', 'C', 'D', 'E', 'F']}
```

4. On a le graphe suivant :



5. La fonction `fonction_mystere` prend en entrée un graphe et teste pour chacun de ses sommets si ce sommet est dans la liste des voisins de chacun de ses voisins et dans ce cas, et seulement dans ce cas, elle renvoie `True`. Au final, elle teste si le graphe est non orienté.
6. L'appel `parcours_profondeur_depuis('A', graphe_1 , marquage)` (où `marquage` est supposé être un dictionnaire dont les clés sont les sommets et les valeurs toutes égales à `False`) fera les appels successifs suivants :
   - `parcours_profondeur_depuis('B', graphe_1 , {'A':True})`
   - `parcours_profondeur_depuis('D', graphe_1 , {'A':True,'B':True})`
   - `parcours_profondeur_depuis('G', graphe_1 , {'A':True,'B':True, 'D':True})`
   - `parcours_profondeur_depuis('C', graphe_1 , {'A':True,'B':True, 'D':True, 'G':True})`
   - `parcours_profondeur_depuis('E', graphe_1 , {'A':True,'B':True, 'D':True, 'G':True,'C':True})`
   - `parcours_profondeur_depuis('F', graphe_1 , {'A':True,'B':True, 'D':True, 'G':True, 'C':True, 'E':True})`

   Donc l'ordre de parcours est `A B D G C E F`.
7. On a l'affichage suivant : `{"A":True,"B":True,"C":True, "D":False,"E":False,"F":False}`
   Remarque : les sommets dont la valeur est `True` sont ceux appartenant à la composante connexe du sommet de départ donc ici `A`.
8. Le code suivant convient :

```
def sommets_accessible(sommet, graphe):
    marquage = {sommet: False for sommet in graphe}
    parcours_profondeur_depuis(sommet, graphe, marquage)
    sommets_accessibles = []
    for s in marquage:
        if marquage[s]:
            sommets_accessibles.append(s)
    return sommets_accessibles
```

9. La fonction précédente n'affiche que les sommets accessible à partir du sommet de départ, c'est-à-dire appartenant à sa composante connexe. La fonction suivante convient :

```
def parcours_profondeur(graphe):
    marquage = {sommet: False for sommet in graphe}
    for s in marquage :
        if not marquage[s]:
            parcours_profondeur_depuis(s, graphe, marquage)
```

Partie B

10. La distance d'amitié entre deux amis vaut 1 et celle entre Aurélie et Camille vaut 3.
11. La fonction `distance_amitie_depuis` utilise un parcours en largeur.
12. Le code suivant convient :

```
def distance_amitie_depuis(personne, G):
    distance = {sommet: -1 for sommet in G}
    F = creer_file_vide()
    distance[personne] = 0
    enfiler(F,personne)
    while not(est_vide(F)):
        prenom = defiler(F)
        for ami in G[prenom]:
            if distance[ami] == -1 : # première fois que ami est découvert
                distance[ami] = distance[prenom] + 1
                enfiler(F,ami)
    return distance
```

13. L'instruction suivante convient : `distance_amitie_depuis('Aurélie', graphe_3)['Camille']`
14. La fonction suivante convient :

```
def chaine_entre(personne_1, personne_2,G):
    distance = {sommet: -1 for sommet in G}
    predecesseur = {sommet: None for sommet in G}
    F = creer_file_vide()
    distance[personne_1] = 0
    enfiler(F,personne_1)
    while not(est_vide(F) and distance[personne_2] == -1:
        prenom = defiler(F)
        for ami in G[prenom]:
            if distance[ami] == -1 : # première fois que ami est découvert
                distance[ami] = distance[prenom] + 1
                predecesseur[ami] = prenom
                enfiler(F,ami)
    if distance[personne_2] == -1 :
        return []
    else :
        chaine = [personne_2]
        personne = personne_2
        while predecesseur[personne] != None :
            personne = predecesseur[personne]
            chaine = [personne] + chaine
        return chaine
```

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.