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

25NSIJ2AS1

Exercice 1 (6 points)

Cet exercice porte sur les bases de données relationnelles, les requêtes SQL, la programmation en Python et la manipulation de listes.

Partie A

1. La requête `SELECT nom FROM station WHERE latitude = -22.276000 AND longitude = 166.452833` renvoie la table :

nom
NOUMEA

2. La requête `SELECT nom FROM station ORDER BY nom ;` permet d'obtenir le nom de toutes les stations météorologiques triées par ordre alphabétique.
3. La requête `SELECT forceVent, dirVent FROM observation NATURAL JOIN station WHERE nom = 'BOURAKE' AND date = '2023010214' ;` permet d'obtenir la force et la direction du vent à BOURAKE le 2 janvier 2023 à 14h.
4. La requête `SELECT COUNT(idObs) FROM observation ;` permet d'obtenir le nombre total de relevés en Nouvelle Calédonie.
5. Le schéma relationnel de la table `meteo` en supprimant les données `hauteur`, `precip`, `forceVent` et `dirVent` est `meteo(idStat : INT, nom : TEXT, latitude : REAL, longitude : REAL, idObs : INT, date : TEXT('AAAAMMJJHH')`.

Partie B

6. Les commandes créent une liste à partir du fichier `'observations.csv'`, suppriment les noms des champs, puis transforment les chaînes de caractères en entiers ou en flottants pour certains champs ; le résultat obtenu est une version correctement typée du premier enregistrement de la table.
7. L'instruction nécessaire à l'utilisation du module Python `math` est `import math`.
8. On complète la fonction `coord`.

```
36 def coord(l_obs, stat_ref):
37     """ ... """
38     # ....
39     for obs in l_obs :
40         if obs[1] == stat_ref :
41            return (obs[2], obs[3])
```

9. On écrit un algorithme en pseudo-code de la fonction `liste_stations`.

```
def liste_stations(l_obs, stat_ref, dist):
    # on initialise une liste vide l_ident qui contiendra la liste des identifiants
    # pour chaque station stat de la liste l_obs
        # si la distance entre stat et stat_ref est inférieure à dist
            # appendre l'identifiant de stat à la liste l_ident
    # renvoyer l_ident
```

10. On écrit une fonction `nettoyage`.

```
1  def nettoyage(l_obs, stat_ref):
2      l_stations = liste_stations(l_obs, stat_ref, 2000)
3      l_temp = []
4      for obs in l_obs:
5          if obs[0] in l_stations:
6              l_temp.append(obs[9])
7      return l_temp
```

11. On écrit la fonction `moyenne`.

```
def moyenne(L):
    return sum(L)/len(L)
```

ou bien, de manière plus explicite,

```
def moyenne(L):
    somme = 0
    nombre = 0
    for x in L:
        somme = somme + x
        nombre = nombre + 1
    return somme / nombre
```

12. Ces commandes permettent d'obtenir la moyenne des températures des stations situées à moins de 2000 unités de la station Paris_11 le 1er janvier 2024.

```
>>> liste_obs = creation_liste_obs('observations.csv')
>>> liste_obs = supp_champs(liste_obs)
>>> transtype(liste_obs)
>>> L = [obs[9] for obs in liste_obs if obs[5] // 100 == 20240101 and distance('Paris_11', ob
>>> moyenne(L)
```

Exercice 2 (6 points)

Cet exercice porte sur la structure de pile, la programmation objet et l'algorithmique.

1. On peut terminer le jeu en versant le tube 4 dans le tube 3 en partant de la situation de la figure 4.

Partie A : Les tubes

2. La structure de pile est une structure linéaire de type LIFO last-in first-out, dont les méthodes `empiler` et `depiler` permettent d'ajouter un élément au sommet et de récupérer le sommet en le supprimant de la pile, respectivement.
3. Les lignes 11 et 12 du code de la classe `tube` permettent d'empiler la couleur et de mettre à jour la prochaine position où on pourra empiler une autre couleur.
4. On complète le code de la méthode `depiler`.

```
14      def depiler(self):
15          if self.taille > 0:
16              self.taille = self.taille - 1
17              couleur = self.contenu[self.taille]
18              self.contenu[self.taille] = 0
19              return couleur
20          else:
21              return -1
```

5. On écrit une méthode `est_plein` de la classe `tube`.

```
    def est_plein(self):
        return self.taille == 3
```

6. On écrit une méthode `est_homogene` de la classe `tube`.

```
def est_homogene(self):
    if self.taille < 2:
        return True
    if self.contenu[1] != self.contenu[0]:
        return False
    if self.taille == 2:
        return True
    return self.contenu[2] == self.contenu[1]
```

7. On écrit une méthode `derniere_couleur` de la classe `tube`.

```
def derniere_couleur(self):
    if self.taille == 0:
        return -1
    return self.contenu[self.taille - 1]
```

8. On complète le code de la méthode `verser`.

```
1   def verser(self, other):
2       while not self.est_vide() and (other.est_vide() or other.taille<3 \
3             and  self.derniere_couleur() == other.derniere_couleur()) :
4           couleur = self.depiler()
5           other.empiler(couleur)
```

Partie B : Le jeu

9. Le code `tube2.verser(tube1)` permet de faire passer la variable `etat` de la représentation en figure 2 à celle de la figure 3.
10. On écrit une fonction booléenne `gagne`.

```
def gagne(etat):
    for pile in etat:
        if not etat.est_homogene():
            return False
    return True
```

Exercice 3 (8 points)

Cet exercice porte sur la programmation Python, les graphes et les réseaux.

Partie A

1. Donner la valeur associée à la clé 1 dans ce dictionnaire est `[2, 5]`.
2. On écrit une fonction `voisins`.

```
def voisins(graphe, k):
    return graphe[k]
```

3. On écrit la fonction `degre_du_sommet`.

```
def degre_du_sommet(graphe, sommet):
    return len(graphe[sommet])
```

4. On écrit la fonction `degre_sommets`.

```
def degre_sommets(graphe):
    return [(sommet, degre_du_sommet(graphe, sommet)) for sommet in graphe]
```

5. Ligne 4 la boucle `for` fait varier `i` de `0` à `len(l_deg)` **inclus** mais les éléments de `l_deg` sont indexés de `0` à `len(l_deg)` **exclus**, ce qui explique l'erreur d'index déclenché à la ligne 6 lorsque `i` vaut 3 dans le cas d'usage testé. Supprimer le `+1` suffit à corriger l'erreur.

```
4       for i in range(len(l_deg)):
```

6. Le tri de `tri_liste` est un tri par sélection.
7. On écrit une fonction `tri_sommets`.

```
def tri_sommets(graphe):
    liste_couples = tri_liste(degre_sommets(graphe))
    return [t[0] for t in liste_couples]
```

```
1  def coloration(g):
2      """Renvoie une coloration du graphe g"""
3      # Algorithme de Welsh-Powell, limité à 4 couleurs
4
5      couleur = ['Rouge', 'Bleu', 'Vert', 'Jaune']
6      coloration_sommets = {}
7      for s_i in g:
8          coloration_sommets[s_i] = None
9      for s_i in tri_sommets(g):
10         couleurs_voisins_s_i = [coloration_sommets[s_j] for s_j in voisins(g, s_i)]
11         k = 0
12         while couleur[k] in couleurs_voisins_s_i :
13             k = k + 1
14         coloration_sommets[s_i] = couleur[k]
15     return coloration_sommets
```

8. La variable `coloration_sommets` est un dictionnaire qui après exécution de la boucle des lignes 7 et 8 vaut `{1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None,8: None, 9: None}`.
9. La fonction fournit le coloriage :

```
{1: 'Vert', 2: 'Bleu', 3: 'Rouge', 4: 'Vert', 5: 'Rouge', 6: 'Bleu', 7: 'Bleu',
8: 'Bleu', 9: 'Rouge'}
```

Partie B

10. La commande `cp prog1.py ../travail/TP` convient.
11. La commande `ping 190.12.10.25` convient.
12. Une adresse possible pour l'ordinateur P2 est 12.128.42.42.
13. Le chemin emprunté par un paquet de données allant de l'ordinateur P1 à l'ordinateur P2 est P1-S1-R1-R2-R3-R8-R9-S2-P2. 1
14. Le protocole de routage qui semble être utilisé est RIP, puisque avec OSPF R1-R5-R4-R3 serait plus court que R1-R3.
15. Les coûts pour des liaisons de 100 Mbits/s, 1 Gbits/s et 10 Gbits/s sont respectivement $\frac{10^8}{100 \times 10^6} = 1$, $\frac{10^8}{10^9} = 0,1$ arrondi à 1 [1], et $\frac{10^8}{10 \times 10^9} = 0,01$ arrondi à 1.
    Par conséquent les coûts sont tous égaux à 1, et utiliser OSPF revient à utiliser RIP, même si les tables de routage ne sont pas calculées par le même algorithme.
16. La route qui sera empruntée par le paquet de données envoyé de l'ordinateur P1 à l'ordinateur P2, en respectant le protocole OSPF, sera donc P1-S1-R1-R2-R3-R8-R9-S2-P2, ce qui remet en question la réponse à la question 14 évidemment.

1. OSPF travaille avec des coûts entiers.

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.