Préambule

Ce sujet, noté sur 20, comporte trois exercices notés respectivement sur 6, 6 et 8 points. Les trois exercices sont à traiter en intégralité. Ce sujet n'est pas à rendre avec votre copie. Lors de la correction, la plus grande attention sera portée à la clarté de vos réponses.
Le fait de répondre sans faire de phrases sera sévèrement sanctionné.
Lorsque vous écrirez du code, pensez à bien marquer de manière visible les indentations éventuelles.
L'usage de la calculatrice est interdit.

Exercice 1 *6 points*

Cet exercice porte sur la programmation Python, la programmation orientée objet, les structures de données (file), l'ordonnancement et l'interblocage.

On s'intéresse aux processus et à leur ordonnancement au sein d'un système d'exploitation. On considère ici qu'on utilise un monoprocesseur.

1. Citer les trois états dans lesquels un processus peut se trouver.

On veut simuler cet ordonnancement avec des objets. Pour ce faire, on dispose déjà de la classe Processus dont voici la documentation :

Classe Processus

```
p = Processus(nom: str, duree: int)
        Crée un processus de nom <nom> et de durée <duree> (exprimée en
        cycles d'ordonnancement)

p.execute_un_cycle()
        Exécute le processus donné pendant un cycle.

p.est_fini()
        Renvoie True si le processus est terminé, False sinon.
```

Pour simplifier, on ne s'intéresse pas aux ressources qu'un processus pourrait acquérir ou libérer.

2. Citer les deux seuls états possibles pour un processus dans ce contexte.

Pour mettre en place l'ordonnancement, on décide d'utiliser une file, instance de la classe File ci-dessous.

Classe File

```
class File:
    def __init__(self):
        """ Crée une file vide """
        self.contenu = []

    def enfile(self, element):
        """ Enfile element dans la file """
        self.contenu.append(element)

    def defile(self):
        """ Renvoie le premier élément de la file et l'enlève de la file """
        return self.contenu.pop(0)

    def est_vide(self):
        """ Renvoie True si la file est vide, False sinon """
        return self.contenu == []
```

Lors de la phase de tests, on se rend compte que le code suivant produit une erreur :

```
f = File()
print(f.defile())
```

3. Rectifier sur votre copie le code de la classe `File` pour que la fonction `defile` renvoie `None` lorsque la file est vide.

On se propose d'ordonnancer les processus avec une méthode du type tourniquet telle qu'à chaque cycle :

- si un nouveau processus est créé, il est mis dans la file d'attente ;
- ensuite, on défile un processus de la file d'attente et on l'exécute pendant un cycle ;
- si le processus exécuté n'est pas terminé, on le replace dans la file.

Attention, lorsqu'un processus est créé, il se met en dernière position de la file d'attente, derrière les processus qui étaient en cours d'exécution et qui ne sont pas encore terminés.
Par exemple, avec les processus suivants :

Liste des processus		
processus	cycle de création	durée en cycles
A	2	3
B	1	4
C	4	3
D	0	5

On obtient le chronogramme ci-dessous :

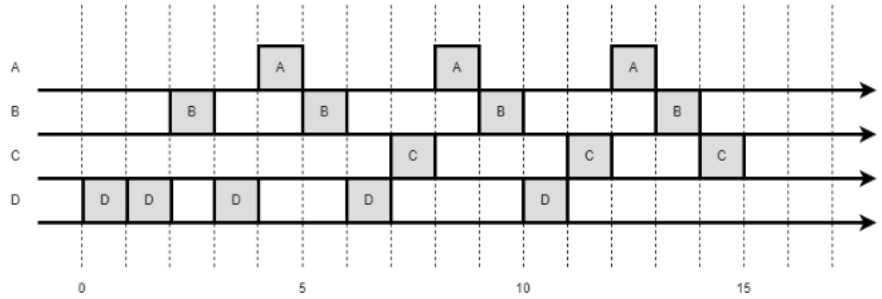

Figure 1. Chronogramme pour les processus A, B, C et D

Pour décrire les processus et le moment de leur création, on utilise le code suivant, dans lequel `depart_proc` associe à un cycle donné le processus qui sera créé à ce moment :

```
p1 = Processus("p1", 4)
p2 = Processus("p2", 3)
p3 = Processus("p3", 5)
p4 = Processus("p4", 3)
depart_proc = {0: p1, 1: p3, 2: p2, 3: p4}
```

Il s'agit d'une modélisation de la situation précédente où un seul processus peut être créé lors d'un cycle donné.

4. Recopier et compléter sur votre copie le chronogramme ci-dessous pour les processus `p1`, `p2`, `p3` et `p4`.

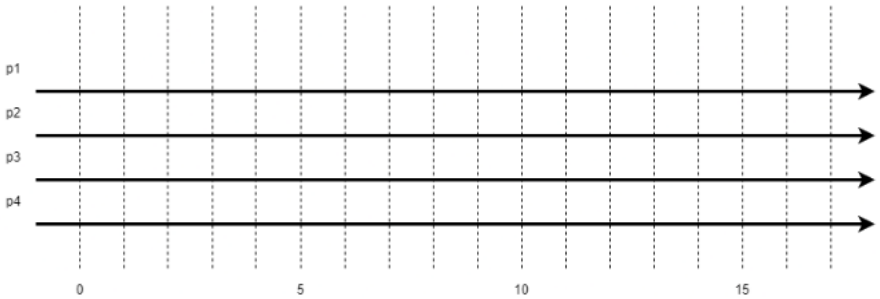

Figure 2. Chronogramme pour les processus p1, p2, p3 et p4

Pour mettre en place l'ordonnancement suivant cette méthode, on écrit la classe `Ordonnanceur` dont voici un code incomplet (l'attribut `temps` correspond au cycle en cours) :

```
1  class Ordonnanceur:
2
3      def __init__(self):
4          self.temps = 0
5          self.file = File()
6
7      def ajoute_nouveau_processus(self, proc):
8          '''Ajoute un nouveau processus dans la file de l'ordonnanceur. '''
9          ...
10
11     def tourniquet(self):
12         '''Effectue une étape d'ordonnancement et renvoie le nom du processus élu.'''
13         self.temps += 1
14         if not self.file.est_vide():
15             proc = ...
16             ...
17             if not proc.est_fini():
18                 ...
19             return proc.nom
20         else:
21             return None
```

5. Écrire sur votre copie le code manquant des lignes 9, 15, 16 et 18.

À chaque appel de la méthode `tourniquet`, celle-ci renvoie soit le nom du processus qui a été élu, soit `None` si elle n'a pas trouvé de processus en cours.

6. Écrire un programme qui :

- utilise les variables `p1`, `p2`, `p3`, `p4` et `depart_proc` définies précédemment ;
- crée un ordonnanceur ;
- ajoute un nouveau processus à l'ordonnanceur lorsque c'est le moment ;
- affiche le processus choisi par l'ordonnanceur ;
- s'arrête lorsqu'il n'y a plus de processus à exécuter.

Dans la situation donnée en exemple (voir Figure 1), il s'avère qu'en fait les processus utilisent des ressources comme :

- un fichier commun aux processus ;
- le clavier de l'ordinateur ;
- le processeur graphique (GPU) ;
- le port 25000 de la connexion Internet.

Voici le détail de ce que fait chaque processus :

Liste des processus			
A	B	C	D
acquérir le GPU	acquérir le clavier	acquérir le port	acquérir le fichier
faire des calculs	acquérir le fichier	faire des calculs	faire des calculs
libérer le GPU	libérer le clavier	libérer le port	acquérir le clavier
	libérer le fichier		libérer le clavier
			libérer le fichier

7. Montrer que l'ordre d'exécution donné en exemple aboutit à une situation d'interblocage.

Exercice 2 ***6 points***

Cet exercice porte sur la programmation Python (listes, dictionnaires) et la méthode "diviser pour régner".
Cet exercice est composé de trois parties indépendantes.

Dans cet exercice, on s'intéresse à des algorithmes pour déterminer, s'il existe, l'élément absolument majoritaire d'une liste.

On dit qu'un élément est *absolument majoritaire* s'il apparaît dans strictement plus de la moitié des emplacements de la liste.

Par exemple, la liste `[1, 4, 1, 6, 1, 7, 2, 1, 1]` admet 1 comme élément absolument majoritaire, car il apparaît 5 fois sur 9 éléments. Par ailleurs, la liste `[1, 4, 6, 1, 7, 2, 1, 1]` n'admet pas d'élément absolument majoritaire, car celui qui est le plus fréquent est 1, mais il n'apparaît que 4 fois sur 8, ce qui ne fait pas plus que la moitié.

1. Déterminer les effectifs possibles d'un élément absolument majoritaire dans une liste de taille 10.

Partie A : Calcul des effectifs de chaque élément sans dictionnaire

On peut déterminer l'éventuel élément absolument majoritaire d'une liste en calculant l'effectif de chacun de ses éléments.

2. Écrire une fonction `effectif` qui prend en paramètres une valeur `val` et une liste `lst` et qui renvoie le nombre d'apparitions de `val` dans `lst`. Il ne faut pas utiliser la méthode `count`.

3. Déterminer le nombre de comparaisons effectuées par l'appel
`effectif(1, [1, 4, 1, 6, 1, 7, 2, 1, 1])`.

4. En utilisant la fonction `effectif` précédente, écrire une fonction `majo_abs1` qui prend en paramètre une liste `lst`, et qui renvoie son élément absolument majoritaire s'il existe et renvoie `None` sinon.

5. Déterminer le nombre de comparaisons effectuées par l'appel
`majo_abs1([1, 4, 1, 6, 1, 7, 2, 1, 1])`.

Partie B : Calcul des effectifs de chaque élément dans un dictionnaire

Un autre algorithme consiste à déterminer l'élément absolument majoritaire éventuel d'une liste en calculant l'effectif de tous ses éléments en stockant l'effectif partiel de chaque élément déjà rencontré dans un dictionnaire.

6. Recopier et compléter les lignes 3, 4, 5 et 7 de la fonction `eff_dico` suivante qui prend en paramètre une liste `lst` et qui renvoie un dictionnaire dont les clés sont les éléments de `lst` et les valeurs les effectifs de chacun de ces éléments dans `lst`.

```
1  def eff_dico(lst):
2      dico_sortie = {}
3      for ......... :
4          if ... in dico_sortie:
5              ...
6          else:
7              ...
8      return dico_sortie
```

7. En utilisant la fonction `eff_dico` précédente, écrire une fonction `majo_abs2` qui prend en paramètre une liste `lst`, et qui renvoie son élément absolument majoritaire s'il existe et renvoie `None` sinon.

Partie C : par la méthode «diviser pour régner»

Un dernier algorithme consiste à partager la liste en deux listes. Ensuite, il s'agit de déterminer les éventuels éléments absolument majoritaires de chacune des deux listes. Il suffit ensuite de combiner les résultats sur les deux listes afin d'obtenir, s'il existe, l'élément majoritaire de la liste initiale.

Les questions suivantes vont permettre de concevoir précisément l'algorithme.

On considère `lst` une liste de taille `n`.

8. Déterminer l'élément absolument majoritaire de `lst` si `n = 1`. *C'est le cas de base.*

On suppose que l'on a partagé `lst` en deux listes :

- `lst1 = lst[:n//2]` (`lst1` contient les `n//2` premiers éléments de `lst`)
- `lst2 = lst[n//2:]` (`lst2` contient les autres éléments de `lst`)

9. Si ni `lst1` ni `lst2` n'admet d'élément absolument majoritaire, expliquer pourquoi `lst` n'admet pas d'élément absolument majoritaire.

10. Si `lst1` admet un élément absolument majoritaire `maj1`, donner un algorithme pour vérifier si `maj1` est l'élément absolument majoritaire de `lst`.

11. Recopier et compléter les lignes 4, 11, 13, 15 et 17 pour la fonction récursive `majo_abs3` qui implémente l'algorithme précédent.

Vous pourrez utiliser la fonction `effectif` de la question 2.

```
1  def majo_abs3(lst):
2      n = len(lst)
3      if n == 1:
4          return ...
5      else:
6          lst_g = lst[:n//2]
7          lst_d = lst[n//2:]
8          maj_g = majo_abs3(lst_g)
9          maj_d = majo_abs3(lst_d)
10         if maj_g is not None:
11             eff = ......
12             if eff > n/2:
13                 return ...
14         if maj_d is not None:
15             eff = ......
16             if eff > n/2:
17                 return ...
```

Exercice 3 ***8 points***

Cet exercice porte sur les réseaux, les protocoles réseau, les bases de données relationnelles et les requêtes SQL.
Cet exercice est composé de 3 parties indépendantes.

La société LUDOJUV est spécialisée dans la production et la distribution de magazines ludiques et pédagogiques destinés aux enfants. L'entreprise étant répartie sur plusieurs bâtiments, son réseau peut être schématisé de la manière suivante :

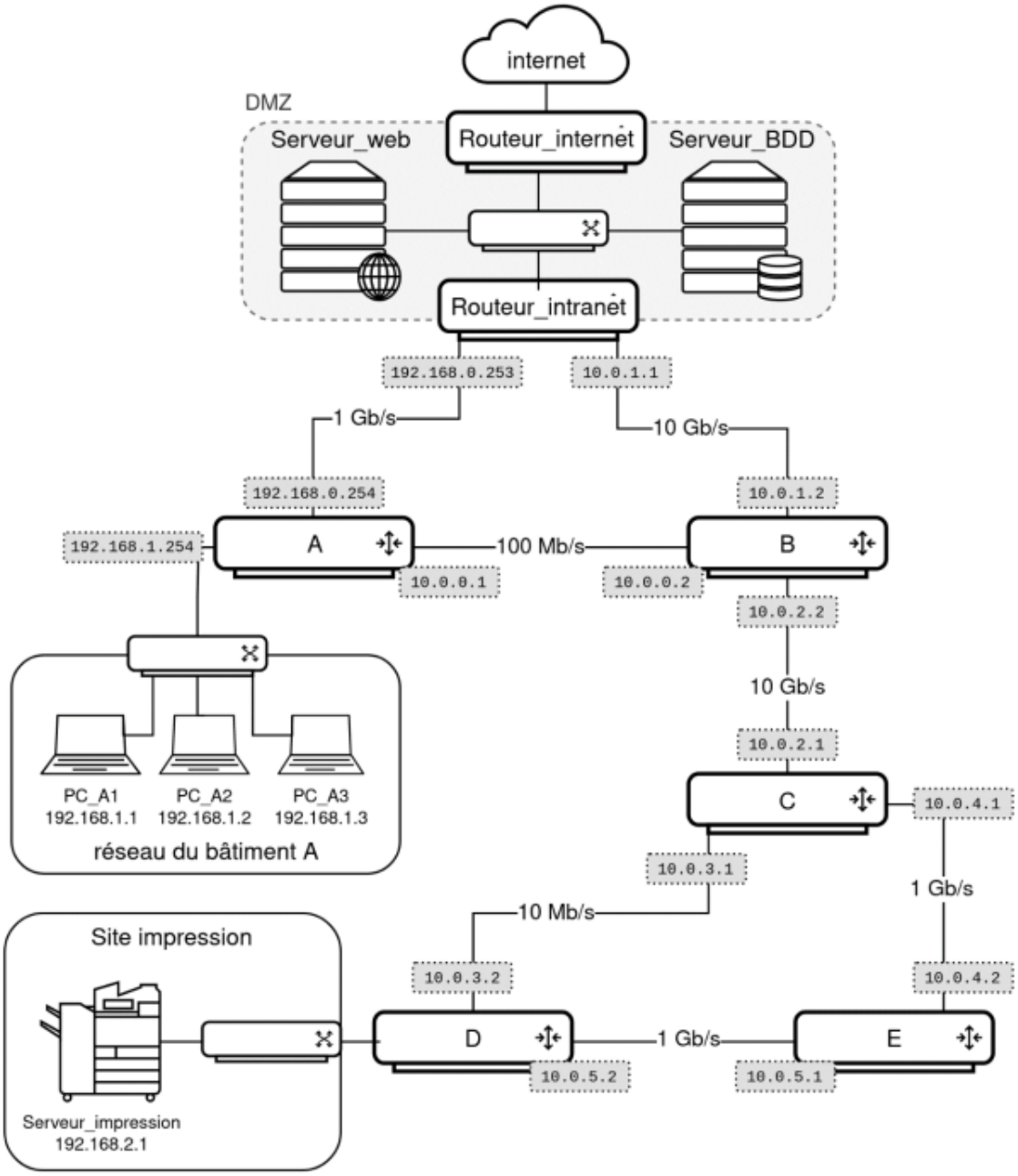

Figure 1. réseau informatique LUDOJUV

Partie A : Configuration réseau dans la DMZ

En informatique, on appelle DMZ (DeMilitarized Zone) un sous-réseau tampon entre un réseau sécurisé (le réseau local) et un réseau non sécurisé (Internet). La DMZ héberge les serveurs qui ont besoin d'accéder à Internet et d'être joignables depuis Internet et filtre l'accès au réseau local protégé.

La DMZ de l'entreprise est délimitée par les routeurs `Routeur_internet` et `Routeur_intranet`.

Dans cette partie du réseau, l'adressage des machines est construit sur l'adresse de réseau IPv4 `172.16.0.0` et le masque `255.255.255.0`.

1. Indiquer combien d'octets composent une adresse IPv4.

On attribue les adresses IP suivantes aux routeurs.

- `Routeur_internet` : dernière adresse IP du réseau : `172.16.0.254`
- `Routeur_intranet` : avant dernière adresse IP du réseau : `172.16.0.253`

2. Donner les adresses IP des serveurs de la DMZ en respectant les consignes suivantes.
    - `Serveur_web` : première adresse IP du réseau
    - `Serveur_BDD` : deuxième adresse IP du réseau

La configuration du poste `PC_A1` est la suivante :

- Adresse IP : `192.168.1.1`
- Masque de sous réseau : `255.255.255.0`
- Passerelle par défaut : `192.168.0.254`

Ce poste ne parvient pas à accéder à l'internet, ni même aux serveurs ou imprimantes de l'entreprise.
L'administrateur du réseau effectue les commandes suivantes :

- `ping 192.168.1.2` : la commande réussit.
- `ping 192.168.0.254` : la commande échoue.

3. Indiquer ce que permet de tester la commande `ping`.
4. Donner la nature du problème et proposer une solution pour y remédier.

Partie B : Routage

Dans l'état actuel du réseau, le routage est configuré manuellement et les tables de routage des routeurs sont les suivantes :

Routeur_intranet	
Destination	Prochain saut
172.16.0.0	-
192.168.0.0	-
192.168.1.0	192.168.0.254
192.168.2.0	192.168.0.254
192.168.3.0	192.168.0.254
10.0.0.0	192.168.0.254
10.0.1.0	-
10.0.2.0	192.168.0.254
10.0.3.0	192.168.0.254
10.0.4.0	192.168.0.254
10.0.5.0	192.168.0.254
0.0.0.0	172.16.0.254

Routeur A	
Destination	Prochain saut
172.16.0.0	192.168.0.253
192.168.0.0	-
192.168.1.0	-
192.168.2.0	10.0.0.2
192.168.3.0	10.0.0.2
10.0.0.0	-
10.0.1.0	10.0.0.2
10.0.2.0	10.0.0.2
10.0.3.0	10.0.0.2
10.0.4.0	10.0.0.2
10.0.5.0	10.0.0.2
0.0.0.0	192.168.0.253

Routeur B	
Destination	Prochain saut
172.16.0.0	10.0.0.1
192.168.0.0	10.0.0.1
192.168.1.0	10.0.0.1
192.168.2.0	10.0.2.1
192.168.3.0	10.0.2.1
10.0.0.0	-
10.0.1.0	-
10.0.2.0	-
10.0.3.0	10.0.2.1
10.0.4.0	10.0.2.1
10.0.5.0	10.0.2.1
0.0.0.0	10.0.1.1

Routeur C	
Destination	Prochain saut
172.16.0.0	10.0.2.2
192.168.0.0	10.0.2.2
192.168.1.0	10.0.2.2
192.168.2.0	10.0.3.2
192.168.3.0	10.0.4.2
10.0.0.0	10.0.2.2
10.0.1.0	10.0.2.2
10.0.2.0	-
10.0.3.0	-
10.0.4.0	-
10.0.5.0	10.0.4.2
0.0.0.0	10.0.2.2

Routeur D	
Destination	Prochain saut
172.16.0.0	10.0.3.1
192.168.0.0	10.0.3.1
192.168.1.0	10.0.3.1
192.168.2.0	-
192.168.3.0	10.0.5.1
10.0.0.0	10.0.3.1
10.0.1.0	10.0.3.1
10.0.2.0	10.0.3.1
10.0.3.0	-
10.0.4.0	10.0.3.1
10.0.5.0	-
0.0.0.0	10.0.3.1

Routeur E	
Destination	Prochain saut
172.16.0.0	10.0.4.1
192.168.0.0	10.0.4.1
192.168.1.0	10.0.4.1
192.168.2.0	10.0.5.2
192.168.3.0	-
10.0.0.0	10.0.4.1
10.0.1.0	10.0.4.1
10.0.2.0	10.0.4.1
10.0.3.0	10.0.4.1
10.0.4.0	-
10.0.5.0	-
0.0.0.0	10.0.4.1

5. `PC_A1` veut accéder à `Serveur_impression`. Donner le chemin suivi par les paquets sous la forme `élément1 -> élément2 -> ...`
6. Le lien entre les routeurs C et D est coupé.
   Donner le chemin suivi par les paquets lorsque `PC_A1` veut accéder à `Serveur_impression` et indiquer s'ils arrivent à destination.

Pour remédier à ce genre de problèmes, l'administrateur décide d'utiliser le protocole RIP qui est un protocole de routage automatique dont la métrique est constituée du nombre minimum de routeurs traversés.

Tous les routeurs sont de nouveau opérationnels.

7. Recopier et compléter la table de routage du routeur C si les routeurs sont configurés pour utiliser le protocole RIP et lorsque toutes les tables de routage sont stables.

Routeur C		
Destination	Prochain saut	Métrique
172.16.0.0	10.0.2.2	2
192.168.0.0	10.0.2.2	2
192.168.1.0		
192.168.2.0		
192.168.3.0		
10.0.0.0	10.0.2.2	1
10.0.1.0		
10.0.2.0	-	-
10.0.3.0	-	-
10.0.4.0	-	-
10.0.5.0		
0.0.0.0	10.0.2.2	2

8. Donner le chemin suivi par les paquets lorsque `PC_A1` veut accéder à `Serveur_impression` en appliquant le protocole RIP.
9. Expliquer pourquoi ce chemin n'est pas le meilleur choix possible.
10. Le lien entre les routeurs C et D est coupé.
    Indiquer quelle sera la modification apportée à la table de routage du routeur C et donner le nouveau chemin suivi par les paquets lorsque `PC_A1` veut accéder à `Serveur_impression`.

Partie C : Exploitation de la base de données

Une base de données relationnelle est utilisée pour suivre la création et l'impression des magazines produits par LUDOJUV. Sa description est la suivante :

- **parution(num_parution, titre_parution, redacteur, date_parution)**
- **page(id_page, numero, mise_en_forme, #num_parution)**
- **texte(num_texte, titre_texte, descriptif, nombre_lignes)**
- **image(num_image, titre_image, descriptif, largeur, hauteur, poids)**
- **comporte_texte(#num_texte, #id_page)**
- **comporte_image(#num_image, #id_page)**

Remarques :

- l'attribut ou l'association d'attributs en gras est une clé primaire de la relation ;
- les attributs précédés du symbole dièse (#) sont des clés étrangères ;
- l'attribut `mise_en_forme` dans `page` désigne la police du texte et sa taille ;
- l'attribut `nombre_lignes` dans texte désigne le nombre de lignes dans un texte ;
- l'attribut `poids` dans `image` est un nombre entier qui désigne la taille de l'image sur le disque dur, exprimée en kilooctets ;
- on rappelle que rajouter une clause `ORDER BY` expression à une requête permet de renvoyer ses résultats dans l'ordre croissant de la valeur expression ;
- il est possible de tester si le motif `MOTIF` apparaît dans une chaîne à l'aide de l'opérateur `LIKE`. Ainsi `attribut LIKE "%NSI%"` teste si les valeurs de `attribut` contiennent le motif `NSI`.

11. Écrire une requête SQL permettant d'obtenir la liste de tous les titres parus.

12. Indiquer quelles informations sont renvoyées par la requête suivante :

```
SELECT num_parution, numero
FROM page
WHERE mise_en_forme = 'Arial,12'
ORDER BY num_parution;
```

13. Écrire une requête SQL permettant d'obtenir la liste (avec numéro, titre et poids) des images dont le poids est supérieur à 1000 kilooctets.

14. Indiquer quelles informations sont renvoyées par la requête suivante :

```
SELECT num_parution
FROM parution
JOIN page ON parution.num_parution = page.num_parution
JOIN comporte_image ON comporte_image.id_page = page.id_page
JOIN image ON image.num_image = comporte_image.num_image
WHERE titre_image LIKE "%Appolo%";
```

15. Indiquer quel sera l'effet de la requête suivante :

```
INSERT INTO image
VALUES (2923, 'Volcans du massif central', '', 400, 400, 1430);
```

16. Écrire une requête SQL permettant d'ajouter les informations sur le texte suivant :

- numéro de texte : 2754
- titre : "Vulcania"
- descriptif : "Parc d'attraction"
- nombre de lignes : 250

17. Indiquer quel sera l'effet de la requête suivante si la relation `comporte_texte` ne contient aucune référence au texte concerné :

```
DELETE FROM texte WHERE num_texte = 2034;
```

18. Écrire une requête SQL permettant de supprimer les éventuelles références au texte numéro 2034 dans la relation `comporte_texte`.