Métropole - septembre 2024 - sujet 2

Exercice 1 (Programmation Python et décidabilité - 6 points)
Dans cet exercice, on dira qu'un appel `f(x)`, où `f` est une fonction Python prenant un argument et `x` est une valeur, termine lorsque l'évaluation de `f(x)` renvoie toujours une valeur au bout d'un nombre fini d'étapes. A l'opposé, un tel appel ne termine pas s'il est possible qu'il effectue des instructions à l'infini.

Partie A : boucle `while`
Commençons par un premier exemple, avec une fonction prenant un entier en argument et utilisant une boucle `while`.

```
1  def f1(n):
2      i = n
3      while i != 10:
4          i = i + 1
5      return i
```

1. Sur votre copie, donner les valeurs successives de la variable `i` lors de l'exécution de `f1(7)`, et indiquer si `f1(7)` termine.
2. Indiquer si l'appel `f1(-2)` se termine. Si oui, indiquer la valeur renvoyée.
3. Sur votre copie, donner les cinq premières valeurs prises par la variable `i` lors de l'exécution de `f1(12)`, et indiquer si l'appel `f1(12)` va terminer ou non.
4. Préciser pour quels entiers `n` l'appel `f1(n)` se termine.

Partie B : fonction récursive
Prenons maintenant un deuxième exemple, avec une fonction récursive (prenant elle aussi un entier en argument).

```
1  def f2(n):
2      if n == 0:
3          return 0
4      else:
5          return n + f2(n-2)
```

5. L'appel `f2(4)` termine-t-il ? Si oui, indiquer la valeur renvoyée par `f2(4)` ; sinon, justifier brièvement.
6. L'appel `f2(5)` termine-t-il ? Si oui, indiquer la valeur renvoyée par `f2(5)` ; sinon, justifier brièvement.
7. Déterminer l'ensemble des entiers naturels `n` pour lesquels l'appel `f2(n)` termine.
8. Ecrire une fonction Python `infini`, récursive, telle que l'appel `infini(n)` ne termine pour aucun entier `n`.

Partie C : le problème de l'arrêt
On se demande maintenant s'il est possible d'écrire une fonction `arret` qui prend en arguments une chaîne de caractères `code_f` contenant le code d'une fonction `f` et un argument `x` de `f`, et tel que `arret(code_f, x)` renvoie `True` si l'appel `f(x)` va terminer et `False` sinon.

Dans la suite de cet exercice, on suppose disposer d'une telle fonction `arret` et on implémente la fonction suivante, utilisant cette fonction `arret`, ainsi que la fonction `infini` de la question précédente dont l'appel `infini(n)` ne termine jamais quelle que soit la valeur de `n`.

```
1  def paradoxe(x):
2      if arret(x, x):
3          infini(42)
4      else:
5          return 0
```

De même, on suppose disposer d'une variable `code_paradoxe` contenant le code de la fonction `paradoxe` sous la forme d'une chaîne de caractères, et on s'intéresse à l'appel `paradoxe(code_paradoxe)`.

Cet appel de fonction commence par effectuer le test `arret(code_paradoxe, code_paradoxe)` dans le `if` de la ligne 2.

9. Dans le cas où `arret(code_paradoxe, code_paradoxe)` renvoie `True`, préciser la prochaine instruction à être exécutée. Dans ce cas, expliquer si l'appel `paradoxe(code_paradoxe)` termine.
10. Dans le cas où `arret(code_paradoxe, code_paradoxe)` renvoie `False`, préciser la prochaine instruction à être exécutée. Dans ce cas, expliquer si l'appel `paradoxe(code_paradoxe)` termine.
11. En déduire qu'une telle fonction `arret` ne peut exister.

Exercice 2 (Programmation Python, POO et pile - 6 points)
Le *mélange faro* consiste à partager un jeu de cartes en deux moitiés et intercaler les cartes de ces deux moitiés.

Pour tout l'exercice on notera `n` le nombre de cartes et on considérera qu'il est pair.

Pour modéliser le jeu de cartes, on décide d'utiliser une pile qui sera une instance de la classe `Pile` dont on donne ici l'interface :

- Le constructeur `Pile` ne prend pas de paramètres et renvoie une pile vide.
  ```
  jeu = Pile() #crée une pile vide référencée par jeu
  ```
- La méthode `empile` prend en paramètre une valeur et l'empile sur la pile.
  ```
  jeu.empile(1) #empile la valeur 1 sur la pile jeu
  ```
- La méthode `depile` ne prend pas de paramètres et retire le dernier élément empilé d'une pile non vide et renvoie sa valeur.
  ```
  print(jeu.depile()) #dépile 1 et affiche cette valeur
  ```
- La méthode `est_vide` ne prend pas de paramètres et renvoie un booléen indiquant si la pile est vide.
  ```
  print(jeu.est_vide()) #affiche True puisque la pile est vide
  ```

Le jeu de cartes est alors modélisé par une pile appelée `jeu` de sommet 1, puis 2 en dessous, *et cætera* jusqu'au bas de la pile qui contient `n`, comme illustré sur la figure ci-dessous.

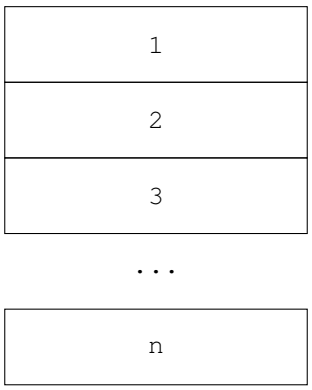

FIGURE 1 – Pile représentant un jeu de cartes

Le mélange faro est réalisé ainsi :

- Etape 1 : on dépile la moitié de `jeu` et chaque élément dépilé est empilé dans une deuxième pile appelée `moitie1` ;
- Etape 2 : on dépile le reste de `jeu` et chaque élément dépilé est empilé dans une troisième pile appelée `moitie2` ;
- Etape 3 : on empile alternativement dans `jeu` et dans cet ordre un élément de `moitie1` puis un élément de `moitie2` jusqu'à vider ces deux piles.

Dans l'exemple suivant les contenus initiaux de `jeu`, `moitie1` et `moitie2` sont représentés ci-dessous :

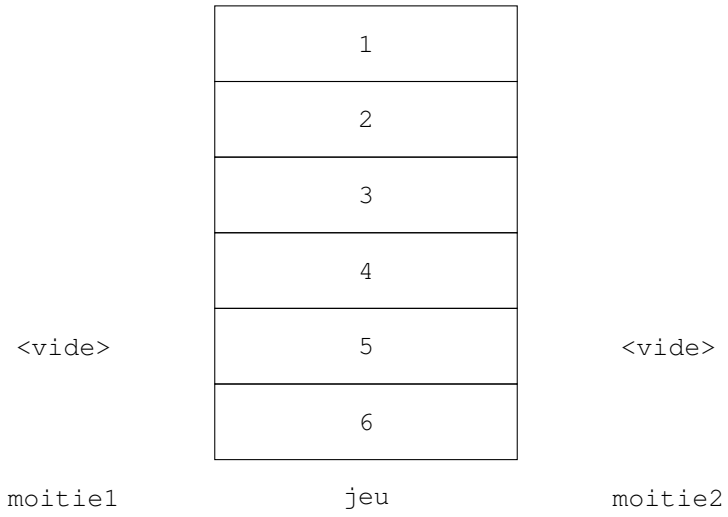

FIGURE 2 – Contenus initiaux des 3 piles

1. Représenter sur votre copie les contenus de ces trois piles à la fin de chaque étape du mélange faro.

Voici le code de la fonction `produire_jeu` qui prend en paramètre un entier `n` supposé pair et qui renvoie une instance de la classe `Pile` qui représente le jeu de cartes.

```
def produire_jeu(n):
    resultat = Pile()
    for i in range(...):
        resultat.empile(...)
    return resultat
```

2. Recopier et compléter sur votre copie le code de la fonction `produire_jeu`.

Ci-après figure le code de la fonction `scinder_jeu` qui prend en paramètres une instance de taille paire de la classe `Pile` qui est le jeu que l'on veut partager en deux moitiés, un entier `n` qui est la taille de la pile et qui renvoie deux piles qui sont les deux moitiés du jeu.

```
def scinder_jeu(p, n):
    m1 = Pile()
    m2 = Pile
    for i in range(n):
        m1.empile(p.depile())
    for i in range(n):
        m2.empile(p.depile())
    return m1, m2
```

3. Ce code comporte des erreurs. Indiquer les numéros de lignes à rectifier ainsi que les rectifications à apporter.
4. Ecrire une fonction `recombiner` qui prend en paramètres deux instances `m1` et `m2` de la classe `Pile` qui sont respectivement la première et la deuxième moitié d'un jeu de cartes et qui renvoie une instance de la classe `Pile` qui est le jeu obtenu en y empilant alternativement et dans cet ordre les éléments de `m1` et de `m2`.
5. Ecrire une fonction `faro` qui prend en paramètres une instance de la classe `Pile` qui est le jeu que l'on veut mélanger, un entier `n` qui est la taille de la pile et qui renvoie une instance de la classe `Pile` qui contient le jeu obtenu en appliquant le mélange faro.

Une propriété mathématique assure qu'étant donné un jeu de `n` cartes (`n` pair), en répétant suffisamment de fois le mélange faro, on finira par remettre le jeu dans l'ordre initial. On aimerait trouver, pour un entier `n` donné, ce nombre minimal de répétitions nécessaires. Pour cela, on considère une fonction `identiques` qui prend en arguments deux instances de la classe `Pile` et qui renvoie un booléen indiquant si ces deux piles ont les mêmes éléments, en même nombre et dans le même ordre.

La fonction `identiques` ne modifie pas les piles données en entrée.

Pour s'assurer que la fonction `identiques` fonctionne correctement, on a commencé à produire un jeu de tests :

```
p1 = Pile()
p1.empile(1)
p2 = Pile()
assert not identiques(p1, p2)
```

6. Compléter ce jeu de tests pour s'assurer que l'on couvre les cas suivants : les piles sont différentes, mais de même taille ; les piles sont identiques.
7. Ecrire une fonction `ordre_faro` qui prend en paramètres un entier `n` pair et qui renvoie le plus petit nombre de répétitions du mélange faro pour qu'un jeu de `n` cartes soit remis dans son ordre initial.

Exercice 3 (Réseaux, protocoles de routage et bases de données - 8 points)

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 :

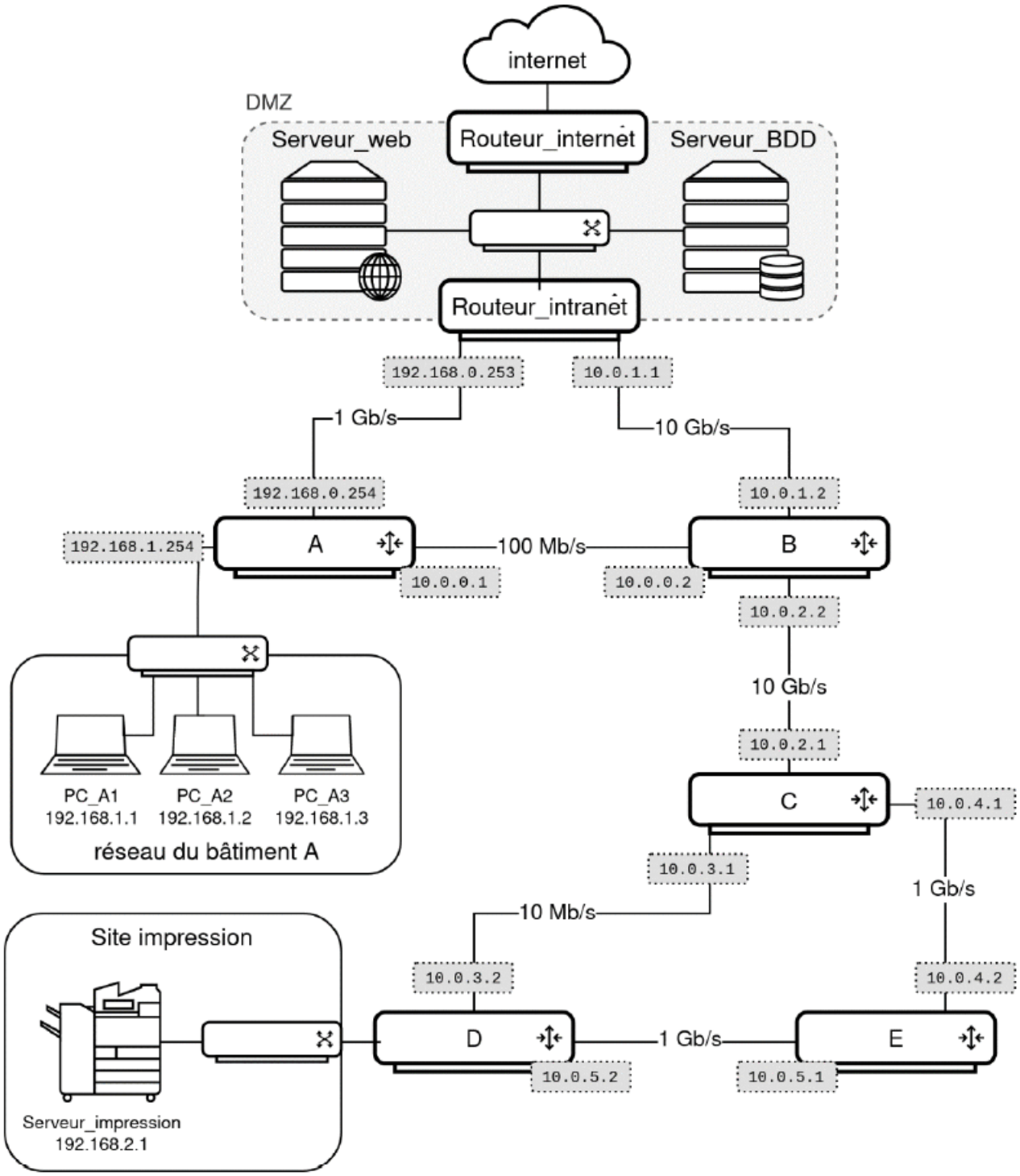

FIGURE 1 – Réseau informatique LUDOJUV

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 :

- `Router_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. 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 souligné 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. Ecrire 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. Ecrire 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. Ecrire 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. Ecrire une requête SQL permettant de supprimer les éventuelles références au texte numéro 2034 dans la relation `comporte_texte`.