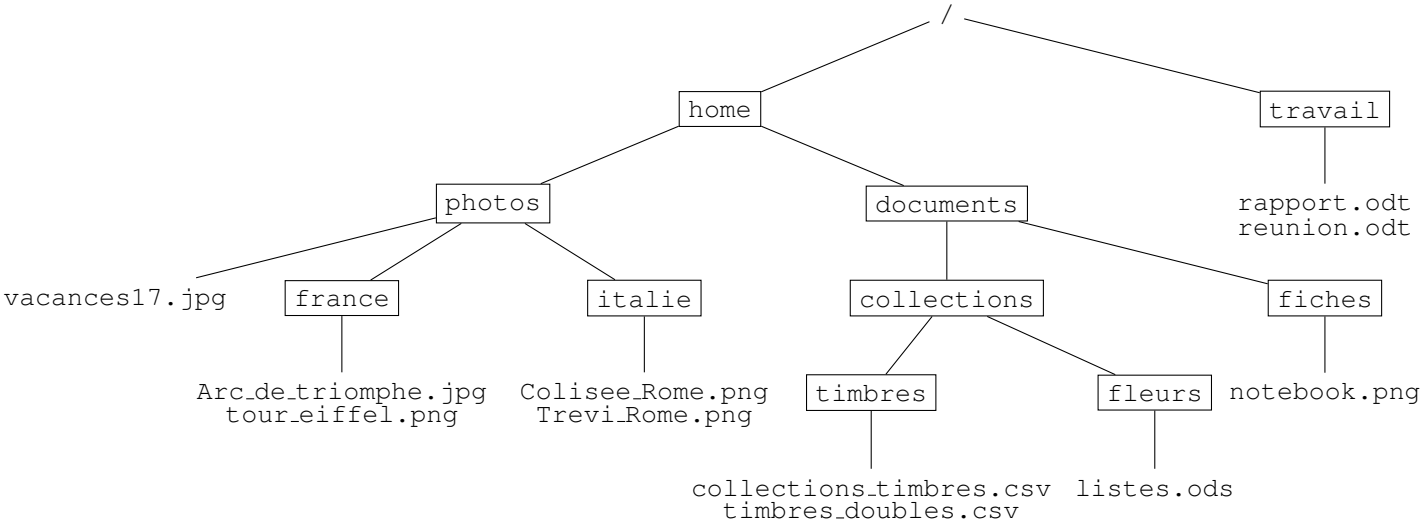
Sujet 0 - 2023 - sujet 2

Exercice 1 (OS, données en tables et bases de données - 4 points)

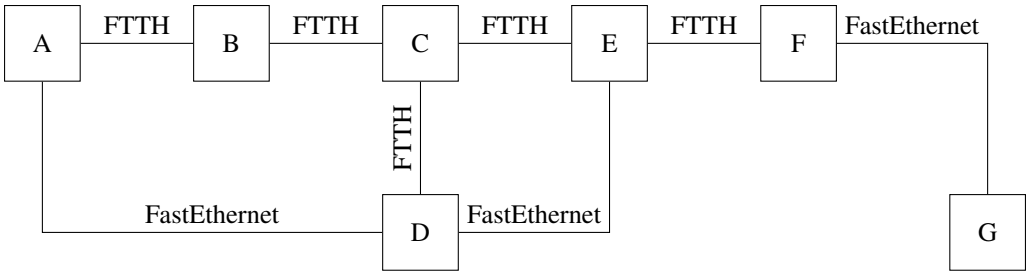
1. Sur une machine équipée du système d'exploitation GNU/Linux, les informations sont enregistrées dans un fichier du répertoire `collections`. Dans le schéma ci-dessous, on trouve des répertoires (encadrés par un rectangle, exemple : `travail`) et des fichiers (les noms non encadrés, comme `rapport.odt`).



   (a) Sachant que le répertoire courant est le répertoire `fiches`, indiquer sur la copie les numéros de toutes les commandes parmi celles proposées ci-dessous qui permettent de se positionner dans le répertoire `timbres` :
   - ⋆ Commande 1 : `cd /home/documents/collections/timbres`
   - ⋆ Commande 2 : `cd ./collections/timbres`
   - ⋆ Commande 3 : `cd /timbres`
   - ⋆ Commande 4 : `cd documents/collections/timbres`
   - ⋆ Commande 5 : `cd ../collections/timbres`
   - ⋆ Commande 6 : `cd timbres`

   (b) Donner une commande qui permet d'accéder au répertoire `timbres` à partir de la racine.

2. On considère le réseau ci-dessous dans lequel :
   - ⋆ les nœuds A, B, C, D, E, F et G sont des routeurs ;
   - ⋆ le type de liaison est précisé entre chaque routeur.



On rappelle que la bande passante des liaisons FTTH (fibre optique : Fiber To The Home) est de 10 Gbit/s et celle des liaisons FastEthernet de 100 Mbit/s.
On s'intéresse au protocole de routage OSPF. Le protocole OSPF est un protocole de routage qui cherche à minimiser la somme des coûts des liaisons entre les routeurs empruntés par un paquet. Le coût C d'une liaison est donné par la relation

$$C = \frac{10^8}{d},$$

où d est la bande passante en bit/s de la liaison.

(a) Calculer le coût d'une liaison de communication par la technologie FastEthernet.

(b) Le fichier `collections_timbres.csv` contenu dans une machine reliée au routeur A doit être envoyé à une machine reliée au routeur G. Déterminer la route permettant de relier le routeur A au routeur G et minimisant la somme des coûts selon le protocole OSPF.

3. Un extrait des informations stockées dans le fichier `collection_timbres.csv` au format CSV est donné ci-dessous :

```
nom_timbre;annee_fabrication;nom_collectionneur
Gustave Eiffel;1950;Dupont
Marianne;1989;Durand
Alan Turing;2012;Dupont
```

Donner les différents descripteurs de ce fichier CSV. Pour chacun de ces descripteurs, donner les valeurs associées.

4. On cherche maintenant à stocker une partie de ces informations dans une base de données relationnelle. La relation suivante a été proposée :

timbres

nom	annee_fabrication
Gustave Eiffel	1950
Marianne	1989
Alan Turing	2012
Gustave Eiffel	1989
Ada Lovelace	1951

(a) Définir la notion de clé primaire d'une relation.

(b) L'attribut `nom` peut-il jouer le rôle de clé primaire ?

(c) Même question pour l'attribut `annee_fabrication`.

(d) Dans le cas d'une réponse par la négative aux deux questions ci-dessus, proposer une solution permettant d'avoir une clé primaire dans la relation `timbres`.

5. On considère maintenant la relation suivante :

collectionneurs

ref_licence	nom	prenom	annee_naissance	nbre_timbres
Hqdfapo	Dupuis	Daniel	1953	53
Dfacqpe	Dupond	Jean-Pierre	1961	157
Qdfqnay	Zaouï	Jamel	1973	200
Aerazri	Pierre	Jean	1967	130
Nzxoeqg	Dupond	Alexandra	1960	61

(a) Décrire le résultat de la requête SQL ci-dessous :

```sql
UPDATE collectionneurs SET ref_licence = 'Ythpswz' WHERE nom = 'Dupond';
```

(b) Expliquer pourquoi, à la suite à cette requête, l'attribut `ref_licence` ne peut plus être une clé primaire pour la relation `collectionneurs`.

6. Ecrire une requête SQL qui affiche, pour chacune des personnes nées en 1963 ou après, le nom, le prénom et le nombre de timbres qu'elles possèdent dans leur collection, enregistrées dans la relation `collectionneurs`. On pourra utiliser certaines des clauses usuelles suivantes : `SELECT`, `FROM`, `WHERE`, `JOIN`, `UPDATE`, `INSERT`, `DELETE`.

Exercice 2 (Récursivité - 4 points)

1. (a) Expliquer en quelques mots ce qu'est une fonction récursive.
   (b) On considère la fonction Python suivante :

```
1 def compte-rebours(n):
2     """ n est un entier positif ou nul """
3     if n >= 0:
4         print(n)
5         compte_rebours(n-1)
```

   L'appel `compte_rebours(3)` affiche successivement les nombres `3`, `2`, `1` et `0`. Expliquer pourquoi le programme s'arrête après l'affichage du nombre `0`.

2. En mathématiques, la factorielle d'un entier naturel n est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. Par exemple :
   - la factorielle de 1 est 1 ;
   - la factorielle de 2 est $2 \times 1 = 2$;
   - la factorielle de 3 est $3 \times 2 \times 1 = 6$;
   - la factorielle de 4 est $4 \times 3 \times 2 \times 1 = 24$;
   - etc.

   Par convention, la factorielle de 0 est 1.

   Recopier et compléter sur votre copie le programme donné ci-dessous afin que la fonction récursive `fact` renvoie la factorielle de l'entier passé en paramètre de cette fonction.
   Exemple : `fact(4)` renvoie `24`.

```
1 def fact(n):
2     """ Renvoie le produit des nombres entiers
3     strictement positifs inférieurs à n """
4     if n == 0:
5         return ..........
6     else:
7         return ..........
```

3. La fonction `somme_entiers_rec` ci-dessous permet de calculer la somme des entiers, de `0` à l'entier naturel `n` passé en paramètre. Par exemple :
   - Pour `n = 0`, la fonction renvoie la valeur `0`.
   - Pour `n = 1`, la fonction renvoie la valeur `0 + 1 = 1`.
   - ...
   - Pour `n = 4`, la fonction renvoie la valeur `0 + 1 + 2 + 3 + 4 = 10`.

```
1 def somme_entiers_rec(n):
2     """ Permet de calculer la somme des entiers
3     de 0 à l'entier naturel n """
4     if n == 0:
5         return 0
6     else:
7         print(n) # pour vérification
8         return n + somme_entiers_rec(n-1)
```

   L'instruction `print(n)` de la ligne 7 dans le code précédent a été insérée afin de mettre en évidence le mécanisme en œuvre au niveau des appels récursifs.
   (a) Ecrire ce qui sera affiché dans la console après l'exécution de l'instruction suivante : `res = somme_entiers_rec(3)`
   (b) Quelle valeur sera alors affectée à la variable `res` ?

4. Ecrire en Python une fonction `somme_entiers` non récursive : cette fonction devra prendre en argument un entier naturel `n` et renvoyer la somme des entiers de `0` à `n` compris. Elle devra donc renvoyer le même résultat que la fonction `somme_entiers_rec` définie à la question 3.
   Exemple : `somme_entiers(4)` renvoie `10`.

Exercice 3 (Arbres binaires de recherche et programmation objet - 4 points)

1. Voici la définition d'une classe nommée `ArbreBinaire`, en Python :

```
class ArbreBinaire:
    """ Construit un arbre binaire """

    def __init__(self, valeur):
        """ Crée une instance correspondant à un état initial """
        self.valeur = valeur
        self.enfant_gauche = None
        self.enfant_droit = None

    def insert_gauche(self, valeur):
        """ Insère le paramètre valeur comme fils gauche """
        if self.enfant_gauche is None:
            self.enfant_gauche = ArbreBinaire(valeur)
        else:
            new_node = ArbreBinaire(valeur)
            new_node.enfant_gauche = self.enfant_gauche
            self.enfant_gauche = new_node

    def insert_droit(self, valeur):
        """ Insère le paramètre valeur comme fils droit """
        if self.enfant_droit is None:
            self.enfant_droit = ArbreBinaire(valeur)
        else:
            new_node = ArbreBinaire(valeur)
            new_node.enfant_droit = self.enfant_droit
            self.enfant_droit = new_node

    def get_valeur(self):
        """ Renvoie la valeur de la racine """
        return self.valeur

    def get_gauche(self):
        """ Renvoie le sous-arbre gauche """
        return self.enfant_gauche

    def get_droit(self):
        """ Renvoie le sous-arbre droit """
        return self.enfant_droit
```

   (a) En utilisant la classe définie ci-dessus, donner un exemple d'attribut, puis un exemple de méthode.

   (b) Après avoir défini la classe `ArbreBinaire`, on exécute les instructions Python suivantes :

```
r = ArbreBinaire(15)
r.insert_gauche(6)
r.insert_droit(18)
a = r.get_valeur()
b = r.get_gauche()
c = b.get_valeur()
```

   Donner les valeurs associées aux variables `a` et `c` après l'exécution de ce code.

On utilise maintenant la classe `ArbreBinaire` pour implémenter un arbre binaire de recherche.

On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel :

- ⋆ on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet ;
- ⋆ si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de `x`, alors il faut que `y.valeur <= x.valeur` ;
- ⋆ si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur > x.valeur`.

2. On exécute le code Python suivant. Représenter graphiquement l'arbre ainsi obtenu.

```
racine_r = ArbreBinaire(15)
racine_r.insert_gauche(6)
racine_r.insert_droit(18)

r_6 = racine_r.get_gauche()
r_6.insert_gauche(3)
r_6.insert_droit(7)

r_18 = racine_r.get_droit()
r_18.insert_gauche(17)
r_18.insert_droit(20)

r_3 = r_6.get_gauche()
r_3.insert_gauche(2)
```

3. On a représenté sur la figure 1 ci-dessous un arbre. Justifier qu'il ne s'agit pas d'un arbre binaire de recherche. Redessiner cet arbre sur votre copie en conservant l'ensemble des valeurs $\{2,3,5,10,11,12,13\}$ pour les nœuds afin qu'il devienne un arbre binaire de recherche.

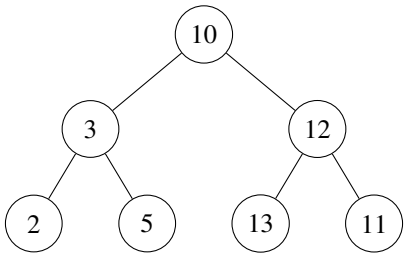


FIGURE 1

4. On considère qu'on a implémenté un objet `ArbreBinaire` nommé `A` représenté sur la figure 2.

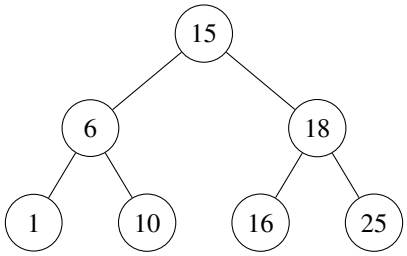


FIGURE 2

On définit la fonction `parcours_infixe` suivante, qui prend en paramètre un objet `ArbreBinaire` `T` et un second paramètre `parcours` de type liste.

```
def parcours_infixe(T, parcours):
    """ Affiche la liste des valeurs de l'arbre """
    if T is not None:
        parcours_infixe(T.get_gauche(), parcours)
        parcours.append(T.get_valeur())
        parcours_infixe(T.get_droit(), parcours)
    return parcours
```

Donner la liste renvoyée par l'appel suivant : `parcours_infixe(A, [ ])`.