Centres étrangers – 2024 – sujet2 - Correction

Exercice 2 (6 points)

Partie A

1. Commande pour connaître le contenu du dossier documents

Depuis le répertoire `home`, la commande est : ls `documents`

---

2. Effet de la commande

```
mv ../../multimedia /home/documents
```

Le dossier `multimedia` est déplacé dans le dossier `documents`.
Il n'est plus à son emplacement initial dans l'arborescence.

---

3. Pourquoi la classe `Arbre` ne modélise pas l'arborescence ?

Un système de fichiers n'est pas un arbre binaire.
Un dossier peut avoir plus de deux sous-dossiers.
Or la classe `Arbre` ne permet que deux fils (`gauche` et `droit`).

---

4. Nom du parcours réalisé

Le parcours affiche :

1. la racine
2. le sous-arbre gauche
3. le sous-arbre droit

C'est un **parcours préfixe** (préordre).

---

5. Parcours en largeur

Un parcours en largeur consiste à parcourir niveau par niveau, de gauche à droite.

(On liste d'abord la racine, puis tous ses fils, puis les fils de ses fils, etc.)

Partie B

6. Méthode `est_vide`

```
def est_vide(self):
    return len(self.fils) == 0
```

👉*Commentaire : Un dossier est vide si sa liste de fils est vide.*

---

7. Instanciation du dossier multimedia

Exemple (structure indicative) :

```
images = Dossier("images", [])
videos = Dossier("videos", [])
musique = Dossier("musique", [])

multimedia = Dossier("multimedia", [images, videos, musique])
```

👉*Commentaire : On doit d'abord créer les sous-dossiers, puis créer le dossier parent en leur passant la liste des fils.*

---

8. Méthode `parcours` en préfixe

```
def parcours(self):
    print(self.nom)
    for f in self.fils:
        f.parcours()
```

👉*Commentaire : On affiche le dossier courant puis on parcourt récursivement tous ses fils.*

---

9. Pourquoi cette méthode termine toujours ?

L'arborescence de fichiers est finie.
Chaque appel récursif traite un sous-dossier plus petit.
La récursion s'arrête quand un dossier n'a plus de fils.

---

10. Version en parcours suffixe (postfixe)

```
def parcours(self):
    for f in self.fils:
        f.parcours()
    print(self.nom)
```

👉*Commentaire : On affiche le dossier après avoir parcouru tous ses fils.*

---

11. Différence avec la commande UNIX `ls`

- `parcours()` affiche tous les descendants récursivement.
- `ls` affiche uniquement le contenu direct du dossier courant.

---

12. Méthode `mkdir`

```
def mkdir(self, nom):
    nouveau = Dossier(nom, [])
    self.fils.append(nouveau)
```

👉*Commentaire : On crée un nouveau dossier vide et on l'ajoute à la liste des fils.*

---

13. Méthode `contient`

```
def contient(self, nom_dossier):
    if self.nom == nom_dossier:
        return True
    for f in self.fils:
        if f.contient(nom_dossier):
            return True
    return False
```

👉*Commentaire : On teste le dossier courant, puis on cherche récursivement dans les sous-dossiers.*

14. Déterminer le dossier parent (idée en français)

On parcourt récursivement l'arborescence.
Pour chaque dossier, on vérifie si l'un de ses fils correspond au dossier recherché.
Si oui, le dossier courant est son parent.

---

15. Modification plus efficace

On peut ajouter un attribut `parent` dans la classe :

```
def __init__(self, nom, liste, parent=None):
    self.nom = nom
    self.fils = liste
    self.parent = parent
```

Lorsqu'on crée un sous-dossier, on lui passe son parent.

👉*Cela permet d'accéder directement au dossier parent sans parcourir toute l'arborescence, ce qui est beaucoup plus efficace.*
