Terminale NSI **Devoir surveillé n°2** le 16-10-20
Durée : 2 heures ***__Calculatrices interdites__***

Exercice 1 (6 points)

1. Qu'est-ce qu'un **système d'exploitation** ? Décrivez brièvement son rôle et ses principales fonctions.

Le reste de l'exercice porte sur les commandes du terminal Linux. Dans la mesure du possible, on utilisera les caractères spéciaux (wildcard) plutôt que de lister tous les noms des fichiers concernés un par un.

On obtient l'affichage suivant dans un terminal Linux.

```
perso@pcA212:~$ pwd
/home/perso
perso@pcA212:~$ tree
.
├── dir1
│   ├── file11.jpg
│   ├── file11.txt
│   ├── file12.jpg
│   ├── file12.txt
│   ├── file13.jpg
│   ├── file13.txt
│   ├── file1.txt
│   ├── file2.txt
│   └── file3.txt
├── dir2
└── jeu.sh

2 directories, 10 files
perso@pcA212:~$ ls -l dir1
total 0
-rw-r--r-- 1 perso perso 7 Oct 10 08:51 file11.jpg
-rw-r--r-- 1 perso perso 7 Oct 10 08:50 file11.txt
-rw-r--r-- 1 perso perso 7 Oct 10 08:51 file12.jpg
-rw-r--r-- 1 perso perso 7 Oct 10 08:50 file12.txt
-rw-r--r-- 1 perso perso 7 Oct 10 08:51 file13.jpg
-rw-r--r-- 1 perso perso 7 Oct 10 08:50 file13.txt
-rwxrw-r-- 1 perso perso 7 Oct 10 08:50 file1.txt
-rwxr----- 1 perso perso 7 Oct 10 08:50 file2.txt
-rw-r--r-- 1 perso perso 7 Oct 10 08:50 file3.txt
perso@pcA212:~$ ls /
bin boot dev etc home init lib lib64 media mnt opt
proc root run sbin srv sys tmp usr var
perso@ pcA212:~$
```

À chaque question, on suppose que les fichiers sont toujours dans la configuration initiale affichée ci-dessus.

2. Que signifie le ~ dans la ligne **perso@ pcA212**:~$?

3. Manipulations de fichiers

a. Quelle commande permet de copier les fichiers file1.txt, file2.txt et file3.txt dans le dossier dir2 ?

b. Quelles commandes permettent de créer un dossier images dans dir1 puis de déplacer les fichiers images dans ce dossier ?

c. On suppose que l'on accède à une clé USB via le dossier usbperso situé dans le dossier media (à la racine). Quelle commande permet de copier les fichiers file11.txt, file12.txt et file13.txt dans cette clé ?

4. Gestion des droits

a. Dans la ligne -rwxrw-r-- 1 perso perso 7 Oct 10 08:50 file1.txt, que signifie -rwxrw-r-- ?

b. Indiquer les droits sur le fichier file2.txt au format linux (par exemple -rwxrw-r--) après la commande

chmod u-x,g+w,o=r dir1/file2.txt .

c. Après la commande chmod 523 dir1/file2.txt, quels sont les droits sur le fichier file2.txt ?

d. Alors qu'on essaie de renommer le fichier file11.txt, on obtient le message :

mv: cannot move 'dir1/file11.txt' to 'dir1/file20.txt': Permission denied

Quelle peut être la cause de cette erreur ? Proposer une commande autorisant cette manipulation.

Exercice 2 (4 points)

On dispose d'une implémentation des listes munie des fonctions primitives suivantes :

- liste_vide() : renvoie une liste vide
- ajoute(e, lst) : renvoie la liste (e, lst), qui est égale à la liste lst complétée par l'élément e en tête de liste.
- est_vide(lst) : renvoie un booléen indiquant si la liste lst est vide
- valeur(lst) : à partir d'une suite lst = (e, lst2) renvoie la valeur e
- suite(lst) : à partir d'une suite lst = (e, lst2) renvoie la liste lst2

1. On applique la fonction traitement ci-dessous à la liste représentée par [2, 4, 5]. Que renvoie-t-elle ?

```
def traitement(lst):
    nlst = liste_vide()
    while not est_vide(lst):
        nlst = ajoute(valeur(lst), nlst)
        lst = suite(lst)
    return nlst
```

2. Recopier et compléter la dernière ligne de la fonction récursive copie qui renvoie une vraie copie d'une liste.

```
def copie(lst):
    if est_vide(lst):
        return liste_vide()
    else:
        return ajoute(……..)
```

3. Écrire une fonction cherche_min qui renvoie la valeur minimale d'une liste de nombres.

Exercice 3 (3 points)

On dispose de deux implémentations des files, que nous désignerons par « implémentation classique » et « implémentation tête-queue ».

1. Indiquer pour chaque implémentation si les objets instanciés de la classe File sont mutables ou immuables.

2. On utilise l'implémentation classique. On souhaite rajouter une méthode len qui renvoie la longueur d'une file en temps constant (sans parcourir toute la file). Pour cela, on modifie la méthode __init__ de la classe File :

```
def __init__(self, c):
    self.cellule = c
    self.longueur = 0
```

a. Quelles autres modifications faut-il apporter à la classe File pour que la propriété longueur soit toujours égale à la longueur de la file ?
b. Écrire la méthode len renvoyant la longueur d'une file.

3. Quel est l'intérêt principal de l'implémentation tête-queue par rapport à l'implémentation classique ?

```
### Implémentation classique ###

class Cellule:
    def __init__(self, valeur, suivant):
        self._valeur = valeur
        self._suivant = suivant

class File:
    def __init__(self, c):
        self.cellule = c

    def enfiler(self, valeur):
        if self.est_vide():
            self.cellule = Cellule(valeur, self.cellule)
        else:
            cel = self.cellule
            while cel._suivant != None:
                cel = cel._suivant
            cel._suivant = Cellule(valeur, cel._suivant)

    def defiler(self):
        valeur = self.cellule._valeur
        self.cellule = self.cellule._suivant
        return valeur

    def est_vide(self):
        return self.cellule == None

def file_vide():
    return File(None)
```

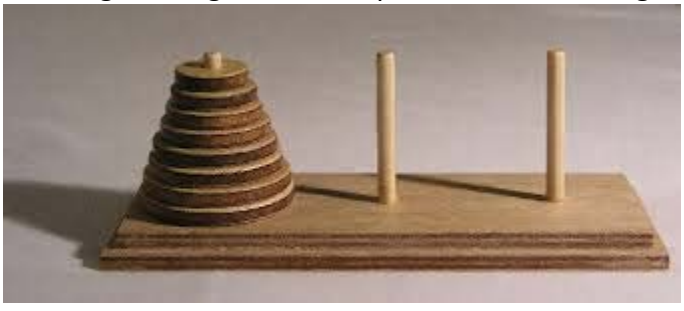
```
### Implémentation tête-queue ###

class Cellule:
    def __init__(self, valeur, suivant):
        self._valeur = valeur
        self._suivant = suivant

class File:
    def __init__(self, c):
        self.tete = c
        self.queue = c

    def enfiler(self, valeur):
        if self.est_vide():
            self.queue = Cellule(valeur, self.queue)
            self.tete = self.queue
        else:
            self.queue._suivant = Cellule(valeur, self.queue._suivant)
            self.queue = self.queue._suivant

    def defiler(self):
        valeur = self.tete._valeur
        self.tete = self.tete._suivant
        return valeur

    def est_vide(self):
        return self.tete == None

def file_vide():
    return File(None)
```

Exercice 4 (7 points)

Le jeu des tours de Hanoï, imaginé par le mathématicien français Édouard Lucas, consiste à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois ;
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ.



Jeu de tours de Hanoi avec 8 disques (source : wikipedia)

On souhaite implémenter le jeu des tours de Hanoi en modélisant chaque tour par une pile et chaque disque par un nombre entier, le plus petit disque étant représenté par le nombre 1, le suivant par 2, …

On dispose d'une implémentation des piles munie des fonctions primitives suivantes :

- pile_vide() : renvoie une pile vide
- est_vide(pile) : renvoie un booléen indiquant si la pile est vide
- depiler(pile) : renvoie le sommet **et le supprime de la pile**
- empiler(pile, element) : ajoute un élément au sommet de la pile, **ne renvoie rien**

Le nombre de disques utilisés est stocké dans la constante NBR_DISQUES.

On a écrit la fonction dessin_disque suivante.

```
def dessin_disque(disque):
    chaine = (2*disque-1)*"="
    nb_espaces = NBR_DISQUES - disque
    chaine = nb_espaces*" " + chaine + nb_espaces*" "
    return chaine
```

1. Si NBR_DISQUES = 4, que renvoie l'appel de dessin_disque(2) ? Indiquer clairement les espaces dans votre réponse.

On a écrit une fonction dessin_jeu représentant les trois tours côte à côte avec leurs disques respectifs. Le jeu est initialisé par le script suivant.

```
def initialisation():
    for i in range(NBR_DISQUES, 0, -1):
        empiler(pileA, i)
    dessin_jeu()

pileA = pile_vide()
pileB = pile_vide()
pileC = pile_vide()
NBR_DISQUES = 3
initialisation()
```

Les trois piles pileA, pileB, pileC représentent les trois tours. Après cette initialisation, l'appel de dessin_jeu() produit l'affichage suivant :

```
  =    |    |
 ===   |    |
=====  |    |
```

Après l'instruction, empiler(pileC, depiler(pileA)), dessin_jeu() affiche :

```
  |    |    |
 ===   |    |
=====  |    =
```

2. Dessiner l'affichage produit par dessin_jeu() après l'initialisation suivie des trois instructions suivantes :

```
empiler(pileC, depiler(pileA))
empiler(pileB, depiler(pileA))
empiler(pileB, depiler(pileC))
```

3. Écrire une fonction d'entête **def** deplacer(pile1, pile2): permettant de déplacer un disque de la pile pile1 vers la pile pile2.

4. Améliorer la fonction précédente en écrivant une fonction deplacer_verification qui effectue le déplacement d'un disque de la pile1 vers la pile2 uniquement si la pile2 est vide ou si le sommet de la pile2 est plus grand que le sommet de la pile1. Si ce n'est pas le cas, cette fonction ne modifiera pas les piles et affichera « déplacement impossible ».

On considère la fonction suivante.

```
def hanoi(n, pileA, pileC, pileB):
    if n > 0:
        hanoi(n-1, pileA, pileB, pileC)
        deplacer(pileA, pileC)
        hanoi(n-1, pileB, pileC, pileA)
```

5. Après l'initialisation du jeu, on exécute l'instruction hanoi(NBR_DISQUES, pileA, pileC, pileB).
Quel affichage produira l'appel de dessin_jeu() après cette instruction ?
Justifier.

Eléments de correction du devoir surveillé n°2

Exercice 1 (6 points)

1. Le système d'exploitation est la couche intermédiaire entre la machine et les applications.
Il gère les ressources matérielles : il cache la complexité du matériel, arbitre les requêtes d'accès aux ressources, évite les conflits, permet un usage équitable du matériel, gère les erreurs, empêche les usages impropres de la machine.
Il facilite l'utilisation de la machine grâce à un ensemble de bibliothèques. Il rend la programmation plus facile, plus simple.
Il permet à un même programme de fonctionner sur des machines très différentes (processeurs, nombre de cœurs, mémoire, type de disque dur ou carte réseau) munies du même système d'exploitation.

2. ~ signifie qu'on est actuellement dans le répertoire utilisateur de l'utilisateur courant (perso).

3. a. cp dir1/file?.txt dir2
b. mkdir dir1/images
mv dir1/*.jpg dir1/images
c. cp dir1/file1?.txt /media/usbperso

4. a. -rwxrw-r-- signifie que :
- il s'agit d'un fichier,
- le propriétaire (user) a les droits en lecture, écriture, exécution
- le groupe propriétaire (group) a les droits en lecture et écriture
- les autres ont uniquement les droits en lecture

b. Les droits sur file2.txt seront : -rw-rw-r—
c. Les droits sur file2.txt sont alors : -r-x-w--wx
d. Le dossier dir1 n'est pas accessible en écriture. On le modifie avec : chmod u+w dir1

Exercice 2 (4 points)

1. La fonction traitement reconstitue la liste élément par élément, mais en inversant l'ordre.
Ainsi, traitement([2, 4, 5]) renvoie [5, 4, 2].

2.
```
def copie(lst):
    if est_vide(lst):
        return liste_vide()
    else:
        return ajoute(valeur(lst), copie(suite(lst)))
```

3.
```
def cherche_min(lst):
    mini = valeur(lst)
    lst = suite(lst)
    while not est_vide(lst):
        if valeur(lst) < mini:
            mini = valeur(lst)
        lst = suite(lst)
    return mini
```

Exercice 3 (3 points)

1. Dans les deux cas, il s'agit d'objets mutables.

2. a. On rajoute les lignes self.longueur += 1 dans la méthode enfiler et self.longueur -=1 dans la méthode defiler.

b. def len(self):

return self.longueur

3. Dans le cas de l'implémentation classique, pour enfiler un élément, on doit parcourir toute la file. La méthode enfiler a donc une complexité linéaire : en O(*n*) pour une file de longueur *n*.
Dans le cas de l'implémentation tête-queue, la propriété queue permet un accès direct à la queue de la file pour enfiler les éléments en temps constant. Il n'est plus nécessaire de parcourir toute la file pour enfiler un élément.

Exercice 4 (7 points)

1. Si NBR_DISQUES = 4, l'appel de dessin_disque(2) renvoie la chaine de caractères ' === '
(deux espaces, trois égals, deux espaces).

2. On obtient :

```
  |    |    |
  |    =    |
=====  ===   |
```

3.

```
def deplacer(pile1, pile2):
    empiler(pile2, depiler(pile1))
```

4.

```
def deplacer_verification(pile1, pile2):
    disque1 = depiler(pile1)
    if est_vide(pile2):
        empiler(pile2, disque1)
    else:
        disque2 = depiler(pile2)
        empiler(pile2, disque2)
        if disque1 < disque2:
            empiler(pile2, disque1)
        else:
            print("déplacement impossible")
            empiler(pile1, disque1)
```

5. On obtient :

```
|   |    =
|   |   ===
|   |  =====
```

On prouve par récurrence pour tout entier $n \geq 1$ que l'appel de hanoi(n, pileA, pileC, pileB) déplace n disques de la pile A vers la pile C en s'aidant de la pile B.
Initiation : pour n = 1, il n'y a qu'un disque et l'instruction deplacer(pileA, pileC) le déplace bien sur la pile C. Les rappels de la fonction hanoi avec n = 0 ne font rien.
Hérédité : on suppose que l'appel de hanoi(n-1, pileA, pileC, pileB) déplace n-1 disques de la pile A vers la pile C en s'aidant de la pile B. Alors, hanoi(n, pileA, pileC, pileB) va exécuter les opérations suivantes :

- hanoi(n-1, pileA, pileB, pileC) déplace n-1 disques de la pile A vers la pile B
- deplacer(pileA, pileC) déplace le disque le plus grand de la pile A vers la pile C
- hanoi(n-1, pileB, pileC, pileA) déplace les n-1 disques restant de la pile B vers la pile C

Suite à ces trois instructions, les n disques ont été déplacés de la pile A vers la pile C.
D'où, par récurrence, la fonction hanoi résout bien le problème des tours de Hanoi pour tout $n \geq 1$.