Amérique du nord - mai 2024 - sujet 1

Exercice 1 (POO, files, processus - 6 points)
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 :

```
class 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.

```
1  class File:
2      def __init__(self):
3          """ Crée une file vide """
4          self.contenu = []
5
6      def enfile(self, element):
7          """ Enfile element dans la file """
8          self.contenu.append(element)
9
10     def defile(self):
11         """ Renvoie le premier élément de la file et l'enlève de la file """
12         return self.contenu.pop(0)
13
14     def est_vide(self):
15         """ Renvoie True si la file est vide, False sinon """
16         return self.contenu == []
```

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

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

3. Rectifier sur votre copie le code de la classe `File` pour que la méthode `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.

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 :

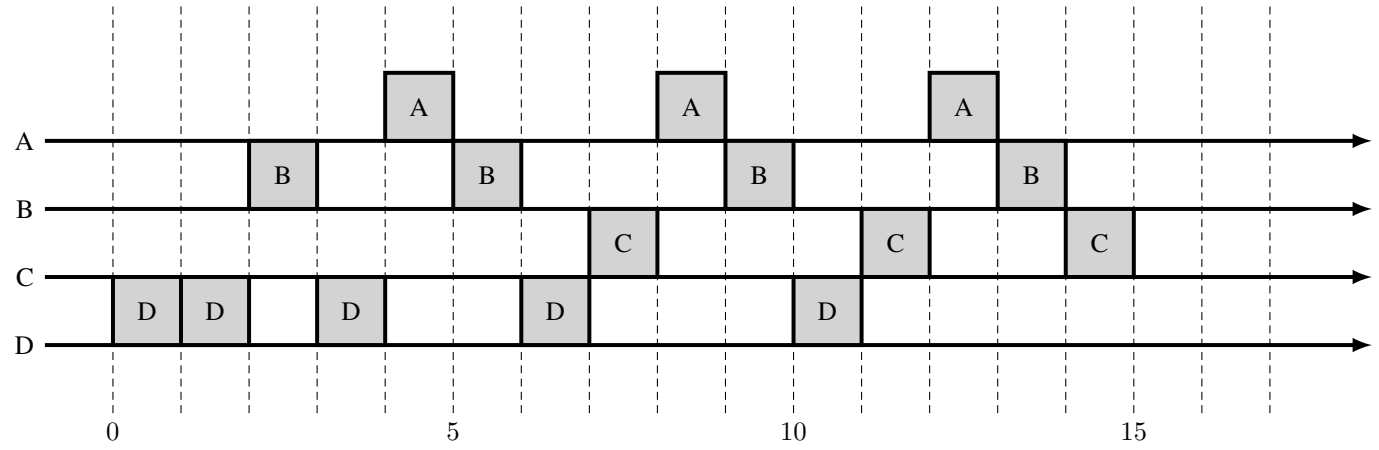

FIGURE 1 – Chronogramme pour les processus A, B, C et D

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 :

```
1 p1 = Processus('p1', 4)
2 p2 = Processus('p2', 3)
3 p3 = Processus('p3', 5)
4 p4 = Processus('p4', 3)
5 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`.

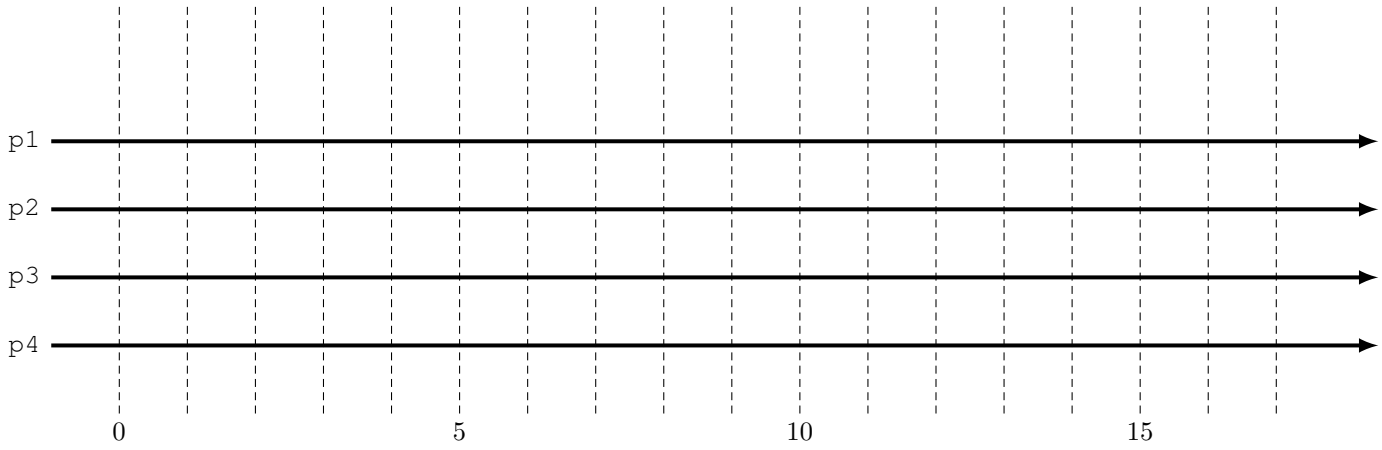

FIGURE 2 – Chronogramme pour les processus `p1`, `p2`, `p3` et `p4`

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) :

```
class Ordonnanceur:

    def __init__(self):
        self.temps = 0
        self.file = File()

    def ajoute_nouveau_processus(self, proc):
        '''Ajoute un nouveau processus dans la file de
        l'ordonnanceur'''
        ...

    def tourniquet(self):
        '''Effectue une étape d'ordonnancement et renvoie le nom
        du processus élu'''
        self.temps += 1
        if not self.file.est_vide():
            proc = ...
            ...
            if not proc.est_fini():
                ...
            return proc.nom
        else:
            return None
```

5. Compléter le code ci-dessus.

A 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. Ecrire 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 (Graphes - 6 points)
Dans cet exercice, on modélise un groupe de personnes à l'aide d'un graphe.

Le groupe est constitué de huit personnes (Anas, Emma, Gabriel, Jade, Lou, Milo, Nina et Yanis) qui possèdent entre elles les relations suivantes :

- Gabriel est ami avec Jade, Yanis, Nina et Milo ;
- Jade est amie avec Gabriel, Yanis, Emma et Lou ;
- Yanis est ami avec Gabriel, Jade, Emma, Nina, Milo et Anas ;
- Emma est amie avec Jade, Yanis et Nina ;
- Nina est amie avec Gabriel, Yanis et Emma ;
- Milo est ami avec Gabriel, Yanis et Anas ;
- Anas est ami avec Yanis et Milo ;
- Lou est amie avec Jade.

Partie A : matrice d'adjacence.
On choisit de représenter cette situation par un graphe dont les sommets sont les personnes et les arêtes représentent les liens d'amitié.

1. Dessiner sur votre copie ce graphe en représentant chaque personne par la première lettre de son prénom entourée d'un cercle et où un lien d'amitié est représenté par un trait entre deux personnes.

Une matrice d'adjacence est un tableau à deux entrées dans lequel on trouve en lignes et en colonnes les sommets du graphe.
Un lien d'amitié sera représenté par la valeur `1` à l'intersection de la ligne et de la colonne qui représentent les deux amis alors que l'absence de lien d'amitié sera représentée par un `0`.

2. Recopier et compléter l'implémentation de la déclaration de la matrice d'adjacence du graphe.

```
    # sommets : G, J, Y, E, N, M, A, L
matrice_adj = [[0, 1, 1, 0, 1, 1, 0, 0], # G
               [.......................], # J
               [.......................], # Y
               [.......................], # E
               [.......................], # N
               [.......................], # M
               [.......................], # A
               [.......................]] # L
```

On dispose de la liste suivante qui identifie les sommets du graphe :

```
sommets = ['G', 'J', 'Y', 'E', 'N', 'M', 'A', 'L']
```

On dispose d'une fonction `position(l, s)` qui prend en paramètres une liste de sommets `l` et un nom de sommet `s` et qui renvoie la position du sommet `s` dans la liste `l` s'il est présent et `None` sinon.

3. Indiquer ce que renvoie chacune des instructions suivantes :

```
>>> position(sommets, 'G')
>>> position(sommets, 'Z')
```

4. Recopier et compléter le code de la fonction `nb_amis(L, m, s)` qui prend en paramètres une liste `L` de noms de sommets, une matrice d'adjacence `m` d'un graphe et un nom de sommet `s` et qui renvoie le nombre d'amis du sommet `s` s'il est présent dans `L` et `None` sinon.

```
1 def nb_amis(L, m, s):
2     pos_s = ...
3     if pos_s == None:
4         return ...
5     amis = 0
6     for i in range(len(m)):
7         amis += ...
8     return ...
```

5. Indiquer ce que renvoie l'instruction suivante :

```
>>> nb_amis(sommets, matrice_adj, 'G')
```

Partie B : dictionnaire de listes d'adjacence.

6. Dans un dictionnaire Python `{c: v}`, indiquer ce que représentent `c` et `v`.

On appelle `graphe` le dictionnaire de listes d'adjacence associé au graphe des amis. On rappelle que Gabriel est ami avec Jade, Yanis, Nina et Milo.

```
graphe = {'G': ['J', 'Y', 'N', 'M'],
          'J': ...
          ...
         }
```

7. Recopier et compléter le dictionnaire de listes d'adjacence `graphe` sur votre copie pour qu'il modélise complètement le groupe d'amis.
8. Ecrire le code de la fonction `nb_amis(d, s)` qui prend en paramètres un dictionnaire de listes d'adjacence `d` et un nom de sommet `s` et qui renvoie le nombre d'amis du nom de sommet `s`. On suppose que `s` est bien dans `d`. Par exemple :

```
>>> nb_amis(graphe, 'L')
1
```

Milo, Gabriel et Yanis se sont fâchés, tandis qu'Anas s'est fâché avec Yanis. Le dictionnaire de listes d'adjacence du graphe qui modélise cette nouvelle situation est donné ci-dessous :

```
graphe = {'G': ['J', 'N'],
          'J': ['G', 'Y', 'E', 'L'],
          'Y': ['J', 'E', 'N'],
          'E': ['J', 'Y', 'N'],
          'N': ['G', 'Y', 'E'],
          'M': ['A'],
          'A': ['M'],
          'L': ['J']
         }
```

Pour établir la liste du cercle d'amis d'un sommet, on utilise un parcours en profondeur du graphe à partir de ce sommet. On appelle cercle d'amis de *Nom* toute personne atteignable dans le graphe à partir de *Nom*.

9. Donner la liste du cercle d'amis de Lou.

Un algorithme possible de parcours en profondeur de graphe est donné ci-dessous :

```
visités = liste vide des sommets déjà visités

fonction parcours_en_profondeur(d, s)
    ajouter s à la liste visités
    pour tous les sommets voisins v de s
        si v n'est pas dans la liste visités
            parcours_en_profondeur(d, v)
    renvoyer la liste visités
```

10. Recopier et compléter le code de la fonction `parcours_en_profondeur(d, s)` qui prend en paramètres un dictionnaire d'adjacence `d` et un sommet `s` et qui renvoie la liste des sommets issue du parcours en profondeur du graphe modélisé par `d` à partir du sommet `s`.

```
1 def parcours_en_profondeur(d, s, visites = []):
2     ...
3     for v in d[s]:
4         ...
5             parcours_en_profondeur(d, v, visites)
6     ...
```

Exercice 3 (Programmation Python, bases de données et SQL - 8 points)
Une *flashcard*, autrement appelée *carte de mémorisation*, est une carte papier sur laquelle se trouve au recto une question et au verso la réponse à cette question. On les utilise en lisant la question au recto, puis en vérifiant notre réponse à celle du verso. Une étudiante souhaite réaliser des *flashcards* numériquement.

Partie A.
L'étudiante souhaite stocker les questions/réponses de ses *flashcards* dans un fichier au format `csv`. Ce format permet de stocker textuellement des données tabulaires. La première ligne du fichier comporte les descripteurs : les noms des champs renseignés par la suite. Pour être en mesure de les identifier, chaque champ est séparé par un caractère appelé séparateur. C'est la virgule qui est le plus couramment utilisée, mais cela peut être d'autres caractères de ponctuation.

Le langage Python dispose d'un module natif nommé `csv` qui permet de traiter de tels fichiers. La méthode `DictReader` de ce module prend en argument un fichier `csv` et le séparateur utilisé. Elle permet d'extraire les données contenues dans le fichier. Voici un exemple de fonctionnement :

- Fichier `exemple.csv` :

```
champ1,champ2
a,7
b,8
c,9
```

- Code Python :

```
import csv
with open('exemple.csv', 'r') as fichier:
    donnees = list(csv.DictReader(fichier, delimiter = ','))
print(donnees)
```

- Affichage généré à l'exécution :

```
[{'champ1': 'a', 'champ2': '7'},
 {'champ1': 'b', 'champ2': '8'},
 {'champ1': 'c', 'champ2': '9'}]Y
```

Voici un extrait du fichier `flashcards.csv` réalisé par l'étudiante :

```
discipline;chapitre;question;réponse
histoire;crise de 1929;jeudi noir - date;24 octobre 1929
histoire;crise de 1929;jeudi noir - quoi;krach boursier
histoire;2GM;l'Axe;Allemagne, Italie, Japon
histoire;2GM;les Alliés;Chine, Etats-Unis, France, Royaume-Uni, URSS
histoire;2GM;Pearl Harbor - date;7 décembre 1941
philosophie;travail;Marx;aliénation de l'ouvrier
philosophie;travail;Beauvoir;donne de la valeur à l'homme
philosophie;travail;Locke;permet de fonder le droit de propriété
philosophie;travail;Crawford;satisfaction et estime de soi
```

1. Donner le séparateur choisi par l'étudiante pour son fichier `flashcards.csv`.
2. Justifier pourquoi l'étudiante a choisi ce séparateur.

Voici le code écrit par l'étudiante pour utiliser ses flashcards :

```
1  import csv
2  import time
3
4  def charger(nom_fichier):
5      with ...
6          donnees = ...
7      return
8
9  def choix_discipline(donnees):
10     disciplines = []
11     for i in range(len(donnees)):
12         disc = donnees[i]['discipline']
13         if not disc in disciplines:
14             disciplines.append(disc)
15     for i in range(len(disciplines)):
16         print(i + 1, disciplines[i])
17     num_disc = int(input('numéro de la discipline ? '))
18     return disciplines[num_disc - 1]
19
20 def choix_chapitre(donnees, disc):
21     chapitres = []
22     for i in range(len(donnees)):
23         if donnees[i]['discipline'] == disc:
24             ch = donnees[i]['chapitre']
25             if not ch in chapitres:
26                 chapitres.append(ch)
27     for i in range(len(chapitres)):
28         print(i + 1, chapitres[i])
29     num_ch = int(input('numéro du chapitre ? '))
30     return chapitres[num_ch - 1]
31
32 def entrainement(donnees, disc, ch):
33     for i in range(len(donnees)):
34         if donnees[i]['discipline'] == disc \
35         and donnees[i]['chapitre'] == ch:
36             print('QUESTION : ', donnees[i]['question'])
37             time.sleep(5)
38             print(donnees[i]['réponse'])
39             time.sleep(1)
40
41 flashcard = ...
42 d = ...
43 c = ...
44 entrainement(...)
```

3. Recopier et compléter le code de la fonction `charger(nom_fichier)` qui lit le fichier dont le nom est fourni en argument et qui renvoie les données lues sous la forme d'un dictionnaire comme dans l'exemple fourni précédemment.
4. Le module `time` est importé à la ligne 2 de ce programme. Quelle est la méthode du module `time` utilisée dans ce code ?
5. Donner le type de la variable `donnees[i]` (par exemple ligne 12).
6. Recopier et compléter les lignes 41 à 44.

Partie B.

Pour améliorer sa mémorisation sur le long terme, l'étudiante décide de mettre en oeuvre le concept de boîtes de Leitner. Dans cette méthode, il s'agit d'espacer dans le temps la révision des *flashcards* si l'étudiante répond correctement. Elle imagine une base de données qui lui permettra de conserver pour chaque question la date à laquelle elle doit de nouveau être posée. Elle décide que les questions seront réparties en cinq boîtes. Initialement, toutes les questions seront placées dans la boîte 1. Les questions de la boîte 1 sont posées tous les jours, celles de la boîte 2 tous les deux jours, celles de la boîte 3 tous les quatre jours, celles de la boîte 4 tous les huit jours et celles de la boîte 5 tous les quinze jours. Si l'étudiante donne la bonne réponse à une question et que la question n'appartient pas à la boîte 5, son numéri de boîte est incrémenté (augmenté de 1). Si l'étudiante ne donne pas la bonne réponse, la question revient dans la boîte 1.

Elle met en oeuvre une base de données relationnelles contenant quatre tables `discipline`, `chapitre`, `boite` et `flashcard`.

La table `discipline` contient la liste des disciplines étudiées. Elle a deux attributs :

- `id`, de type `INT`, l'identifiant de la discipline qui est une clé primaire pour cette table ;
- `lib`, de type `TEXT`, le libellé de la discipline.

La table `chapitre` contient la liste des chapitres des disciplines étudiées. Elle a trois attributs :

- `id`, de type `INT`, l'identifiant du chapitre qui est une clé primaire pour cette table ;
- `lib`, de type `TEXT`, le libellé du chapitre ;
- `id_disc`, de type `INT`, l'identifiant de la discipline à laquelle appartient ce chapitre.

La table `boite` contient l'ensemble des cinq boites existantes. Elle a trois attributs :

- `id`, de type `INT`, l'identifiant numéro de la boîte qui est une clé primaire pour cette table ;
- `lib`, de type `TEXT`, le libellé de la boîte ;
- `frequence`, de type `INT`, indiquant le nombre de jours séparant deux interrogations d'une question appartenant à cette boîte.

La table `flashcard` contient les questions-réponses. Elle a six attributs :

- `id`, de type `INT`, l'identifiant de la *flashcard* qui est une clé primaire pour cette table ;
- `id_ch`, de type `INT`, l'identifiant du chapitre auquel appartient la `flashcard` ;
- `id_boite`, de type `INT`, l'identifiant numéro de la boîte de la `flashcard` ;
- `question`, de type `TEXT`, le texte au recto de la *flashcard* ;
- `reponse`, de type `TEXT`, le texte au verso de la *flashcard* ;
- `date_interro`, de type `DATE`, la date de la prochaine interrogation pour cette question.

Initialement, `date_interro` sera la date d'insertion de la question dans la base de données.

Table boite		
id	lib	frequence
1	tous les jours	1
2	tous les deux jours	2
3	tous les quatre jours	4
4	tous les huit jours	8

7. Ecrire une requête SQL qui complète la table `boite` et insère la boîte 5 de libellé `tous les quinze jours` et de fréquence 15.
8. Une requête sur la table `flashcard` affiche l'enregistrement suivant :

```
5, 2, 1, Pearl Harbor - date, 6 décembre 1941
```

   Ecrire une requête SQL pour mettre à jour la date de Pearl Harbor renvoyée. La bonne date est le 7 décembre 1941.
9. Ecrire une requête SQL qui permet d'obtenir la liste des libellés des disciplines.
10. Ecrire une requête SQL qui permet d'obtenir la liste des libellés des chapitres de la discipline `histoire`.
11. Ecrire une requête SQL qui permet d'obtenir la liste des identifiants des flashcards de la discipline `histoire`.
12. Ecrire une requête SQL pour supprimer toutes les flashcards de la boîte d'identifiant 3.