Exercice 1 :

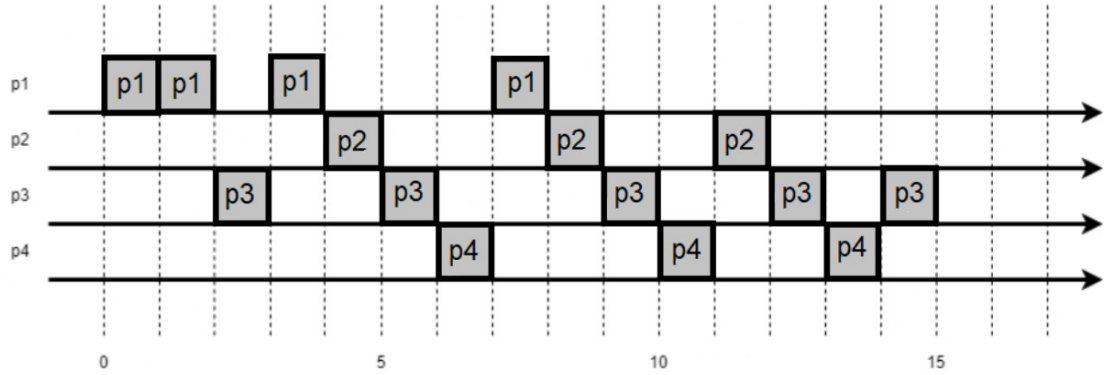
1. Les trois états possibles d'un processus sont : **prêt** (en attente d'un quantum de temps processeur), **bloqué** (en attente de l'accès à une ressources) et **élu** (en cours d'exécution).

2. Dans ce contexte, les deux seuls états possibles sont **prêt** et **élu**.

3.
```
def defile(self):
    if self.est_vide():
        return None
    return self.contenu.pop(0)
```

4. [Attention, un processus élu est prioritaire sur processus créé pour l'ajout dans la file d'après l'exemple de chronogramme donné.]

Contenu de la file															
			p2	p4	p1	p2	p3	p4							
		p3	p1	p2	p3	p4	p1	p2	p3	p4	p2	p3	p4	p3	
élu	p1	p1	p3	p1	p2	p3	p4	p1	p2	p3	p4	p2	p3	p4	p3



5. Code complété :
```
class Ordonnanceur:
    def __init__(self):
        self.temps = 0
        self.file = File()

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

```
    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 = self.file.defile()
            proc.execute_un_cycle()
            if not proc.est_fini():
                self.file.enfile(proc)
            return proc.nom
        else:
            return None
```

6.

```
def gestion_processus(liste_proc, depart_proc):
    tps = 0
    ordo = Ordonnanceur()
    ordo.ajoute_nouveau_processus(depart_proc[tps])
    while not ordo.file.est_vide():
        proc = ordo.tourniquet()
        print(proc)
        tps += 1
        if tps in depart_proc:
            ordo.ajoute_nouveau_processus(depart_proc[tps])
```

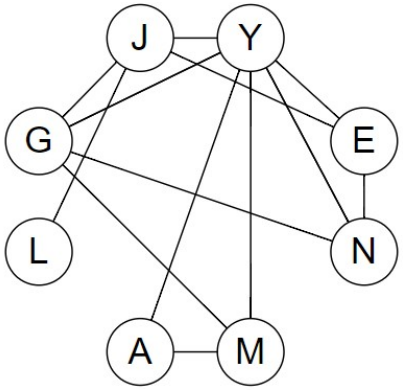
7.

	fichier	**clavier**	**processeur**	**port internet**
D: acquérir le fichier	bloqué par D			
D: faire des calculs	bloqué par D			
B: acquérir le clavier	bloqué par D	bloqué par B		
D: acquérir le clavier (bloqué par B)	bloqué par D	bloqué par B		
A: acquérir le GPU	bloqué par D	bloqué par B	bloqué par A	
B: acquérir le fichier (bloqué par D)	bloqué par D	bloqué par B	bloqué par A	
C: acquérir le port	bloqué par D	bloqué par B	bloqué par A	bloqué par C
A: faire des calculs	bloqué par D	bloqué par B	bloqué par A	bloqué par C
C: faire des calculs	bloqué par D	bloqué par B	bloqué par A	bloqué par C
A: libérer le GPU	bloqué par D	bloqué par B		bloqué par C
C: libérer le port	bloqué par D	bloqué par B		

La situation d'interblocage intervient avec les deux processus D et B qui se bloquent mutuellement en réservant respectivement le fichier et le clavier.
[Par contre les processus A et C peuvent se terminer.]

Exercice 2 : Partie A : Matrice d'adjacence

1. Graphe :



2.

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

3. Avec `sommets = ['G', 'J', 'Y', 'E', 'N', 'M', 'A', 'L']`:
`position(sommets, 'G')` renvoie 0
`position(sommets, 'Z')` renvoie None

4.

```
def nb_amis(L, m, s):
    pos_s = position(L, s)
    if pos_s == None:
        return None
    amis = 0
    for i in range(len(m)):
        amis += m[i][pos_s]
    return amis
```

5. `nb_amis(sommets, matrice_adj, 'G')` renvoie 4.

Exercice 2 : Partie B : Dictionnaire de listes d'adjacence

6. Dans un dictionnaire, **c** représente une clé et **v** la valeur qui lui est associée.

7.
```
graphe = {'G' : ['J', 'Y', 'N', 'M'],
          'J' : ['G', 'Y', 'E', 'L'],
          'Y' : ['G', 'J', 'E', 'N', 'M', 'A'],
          'E' : ['J', 'Y', 'N'],
          'N' : ['G', 'Y', 'E'],
          'M' : ['G', 'Y', 'A'],
          'A' : ['Y', 'M'],
          'L' : ['J']
         }
```

8.
```
def nb_amis(d, s):
    return len(d[s])
```

9. Le cercle d'amis de Lou est Jade, Gabriel, Nino, Yanis et Emma.

10.
```
def parcours_en_profondeur(d, s, visites = []):
    visites += [s]
    for v in d[s]:
        if v not in visites:
            parcours_en_profondeur(d, v)
    return visites
```

Exercice 3 : Partie A :

1. Le séparateur est le point-virgule ; .

2. Certaines données comportent en elles-mêmes des virgules ce qui élimine d'office ce caractère comme pouvant servir de séparateur.

3.
```
def charger(nom_fichier):
    with open(nom_fichier, 'r') as fichier:
        donnees = list(csv.DictReader(fichier, delimiter=';'))
    return donnees
```

4. Le module `time` est utilisé ligne 37 avec sa méthode `sleep()`.

5. `donnees` est une liste de dictionnaire dont `donnees[i]` est un dictionnaire.

6.
```
flashcard = charger("flashcards.csv")
d = choix_discipline(flashcard)
c = choix_chapitre(flashcard, d)
entrainement(flashcard, d, c)
```

Exercice 3 : Partie B :

7.
```
INSERT INTO boite (id, lib, frequence)
VALUES(5, 'tous les quinze jours', 15);
```

8.
```
UPDATE flashcard
SET reponse = 'Pearl Harbor – date, 7 décembre 1941'
WHERE reponse = 'Pearl Harbor – date, 6 décembre 1941';
```

9.
```
SELECT lib FROM discipline;
```
ou pour les avoir par ordre alphabétique :
```
SELECT lib FROM discipline ORDER BY lib;
```

10.
```
SELECT chapitre.lib FROM chapitre
JOIN discipline ON discipline.id = chapitre.id_disc
WHERE discipline.lib = 'histoire' ;
```

11.
```
SELECT flashcard.id FROM flashcard
JOIN chapitre ON chapitre.id = flashcard.id_ch
JOIN discipline ON discipline.id = chapitre.id_disc
WHERE discipline.lib = 'histoire';
```

12.
```
DELETE FROM flashcard WHERE id_boite = 3;
```