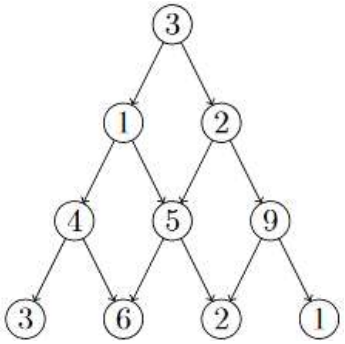
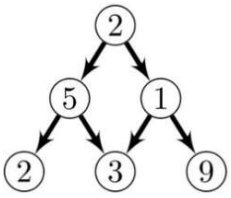
Exercice 1 : Récursivité et optimisation

1. ex2 = [[3], [1,2], [4,5,9], [3,6,2,1]] représente la pyramide :



2. Il y a deux conduits égaux de score maximal :
3 + 2 + 5 + 6 = 16
3 + 2 + 9 + 2 = 16

3. Conduits pour la pyramide :
2 + 5 + 2
2 + 5 + 3
2 + 1 + 3
2 + 1 + 9



4. Pour n = 2, il y a 2 = 2^1 conduits. Pour n = 3, il y a 4 = 2^2 conduits. Pour n = 4, il y a 8 = 2^3 conduits. Ainsi pour n niveaux, il y a $\mathbf{2^{n-1}}$ conduits

5. L'algorithme testant tous les conduits a une complexité **exponentielle O(2^n)** ce qui n'est pas raisonnable.

6.

```
def score_max(i, j, p):  # indice j du niveau i de la pyramide p
    # Cas trivial : dernier niveau
    if i == len(p)-1:
        return p[i][j]
    # Cas récursif : on ajoute le score max des deux sous-arbres
    return p[i][j] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))
```

7.

```
def pyramide_nulle(n):
    p = []
    for k in range(1, n+1):
        p.append([0]*k)
    return p
```

Solution plus sophistiquée :

```
def pyramide_nulle(n): # par compréhension
    return [[0]*k for k in range(1, n+1)]
```

8.

```
def prog_dyn(p):
    n = len(p)
    s = pyramide_nulle(n)
    # remplissage du dernier niveau
    for j in range(n):
        s[n-1][j] = p[n-1][j]
    # remplissage des autres niveaux
    for i in range(n-1, -1, -1):
        for j in range(i):
            s[i][j] = p[i][j] + max(s[i+1][j], s[i+1][j+1])
    # renvoie du score maximal
    return s[0][0]
```

9. On peut observer deux boucles imbriquées s'exécutant chacune au plus n fois, ce qui justifie le coût d'exécution **quadratique $O(n^2)$** de la fonction.

10. Le principe de la programmation dynamique est d'éviter de refaire des calculs déjà effectués. Ainsi, il est possible de stocker le résultat des calculs dans un dictionnaire et de s'en servir à chaque appel récursif pour y puiser les résultats des calculs déjà rencontrés.
Cette technique de stockage en mémoire des résultats redondants est la mémoïsation.

```
def score_max_mem(i, j, p, mem={}):
    # Cas trivial : dernier niveau
    if i == len(p)-1:
        return p[i][j]
    # Cas déjà traité et enregistré dans la mémoire
    if (i, j) in mem:
        return mem[(i,j)]
    # Cas récursif : on ajoute le score max des deux sous-arbres
    r = p[i][j] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))
    mem[(i, j)] = r # enregistrement du résultat dans la mémoire
    return r
```

Exercice 2 : Systèmes d'exploitation et gestion des processus

1. Un logiciel est **libre** quand son code est distribué dans le cadre d'une licence qui en autorise la consultation, la modification et la redistribution, sans contrepartie financière. L'usage de ces systèmes s'avère sans restriction et gratuit, à l'opposé d'un système **propriétaire** dont le code source n'est pas accessible, qui interdit la redistribution et nécessite un paiement initial ou un abonnement.

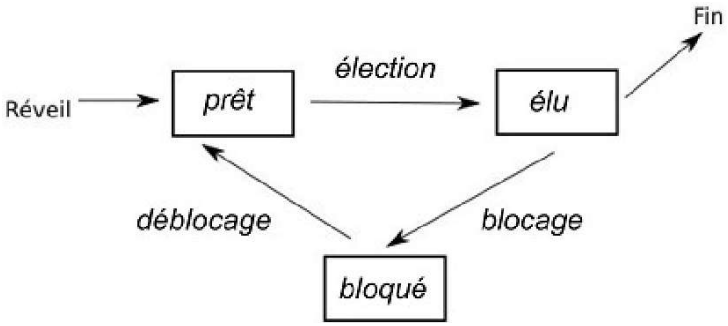
2. Un **système d'exploitation** assure la gestion des accès au processeur, à la mémoire et aux périphériques d'entrée/sortie (clavier, souris, carte réseau, …) et la gestion des fichiers, des utilisateurs et leurs droits. En résumé, il fait le pont entre les ressources de la machines, les autres logiciels et l'utilisateur.

3. Chemin absolu :
`/home/elsa/documents/boulot/rapport.odt`

4. Chemin relatif depuis le répertoire `elsa` :
`../max/images/photos_vac/photo_1.jpg`

5. La commande déplace le fichier `fiche.ods` du répertoire `documents` vers le répertoire `boulot`. Ainsi après l'exécution, le répertoire `document` est vide et le répertoire `boulot` contient les deux fichiers : `rapport.odt` et `fiche.ods`.

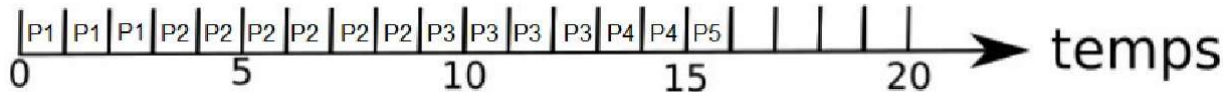
6.



7. Un processus passera de l'état élu à l'état bloqué s'il n'a plus accès à une ressource nécessaire à son exécution.

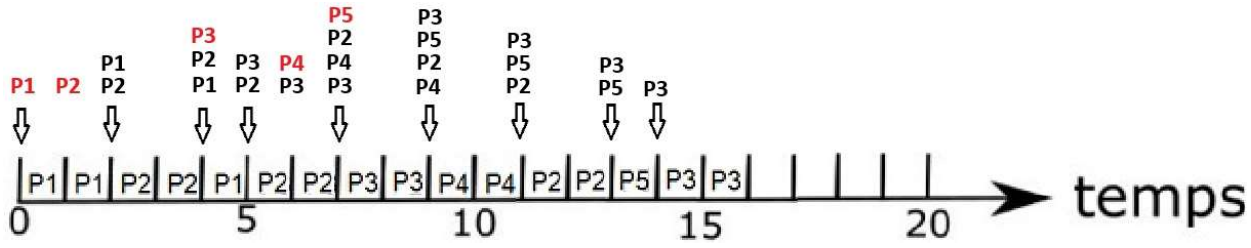
8. Une **pile** est une structure de données linéraire de type LIFO (Last In First Out).

9.

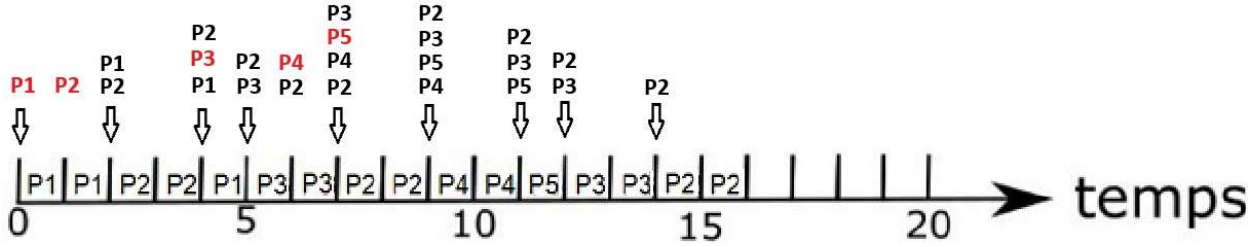


10. Une ambiguïté demeure dans le sujet pour savoir qui du processus perdant l'accès au processeur ou du nouveau processus est empilé en premier lorsque ces évènements se produisent en même temps. C'est le cas aux dates 4 et 7.

Si c'est le processus qui perd l'accès au processeur qui est empilé avant le **nouveau processus**, la réponse est :



Si c'est le **nouveau processus** qui est empilé avant le processus perdant l'accès au processeur, la réponse est :



11. Deux processus P1 et P2 seront en situation d'interblocage si chacun bloque une ressource nécessaire au second pour poursuivre son exécution.

Exercice 3 : Programmation, POO et Base de Données

Partie A

1.
a a la valeur `[10, 8, 9, 9, 8, 10, 6, 7, 8, 8]`.
b a la valeur `"Fondation"`.

2.
```
def titre_livre(dico, id_livre):
    for i in range(len(dico['id'])):
        if dico['id'][i] == id_livre :
            return dico['titre'][i]
    return None
```

3.
```
def note_maxi(dico):
    m = 0
    for i in range(len(dico['id'])):
        if dico['note'][i] > m:
            m = dico['note'][i]
    return m
```

4.
```
def livres_note(dico, n):
    livres = []
    for i in range(len(dico['id'])):
        if dico['note'][i] == n:
            livres.append(dico['titre'][i])
    return livres
```

5. En utilisant les deux fonctions précédentes :
```
def livre_note_maxi(dico):
    return livres_note(dico, note_maxi(dico))
```

Partie B

6. La classe **`Livre`** possède 5 attributs (**`id`**, **`titre`**, **`auteur`**, **`ann_pub`** et **`note`**) et 4 méthodes en excluant le constructeur (**`get_id`**, **`get_titre`**, **`get_auteur`**, **`get_ann_pub`**).

7.
```
def get_note(self):
    return self.note
```

8.
```
bi = Bibliotheque() # Instanciation de la bibliothèque
bi.ajout_livre(Livre(8, "Blade Runner", "K.Dick", 1968, 8))
```

9.

Réponse à privilégier étant donné les définitions des accesseurs de la classe **`Livre`** :

```
def titre_livre(self, id_livre):
    for livre in self.liste_livre :
        if livre.get_id() == id_livre :
            return livre.get_titre()
    return None
```

Étant donné qu'en python il n'y a pas de restriction d'accès aux attributs, une autre réponse acceptable serait :

```
def titre_livre(self, id_livre):
    for livre in self.liste_livre :
        if livre.id == id_livre :
            return livre.titre
    return None
```

Partie C

10. Un auteur peut avoir écrit plusieurs livres et ne peut donc pas permettre d'identifier un livre de manière unique en jouant le rôle de clé primaire.

11.

titre
Ubik
Blade Runner

12.

```
SELECT titre FROM livres WHERE auteur = 'Asimov' AND ann_pub > 1950;
```

13.

```
UPDATE livres SET note = 10 WHERE id = 4 ;
```

14. Séparer les informations dans deux tables permet d'éviter les redondances. Les données sur les auteurs n'apparaissent ainsi qu'une fois au lieu d'être répétées pour chacun de leurs livres.

15. L'attribut **`id_auteur`** de la table **`livres`** est une clé étrangère qui fait référence à la clé primaire **`id`** de la table **`auteurs`**.

16.

```
SELECT nom, prenom FROM auteurs JOIN livres ON auteurs.id = livres.id_auteur
WHERE ann_pub > 1960 ;
```

17. La requête renvoie les titres des livres pour lesquels les auteurs avaient moins de 30 ans lors de leur publication.

18. La création d'une telle base de données contenant des données personnelles devra faire l'objet d'une déclaration préalable auprès de la Commission Nationale Informatique et Libertés (CNIL). De plus, le recueil de ces données personnelles est soumis au consentement préalable des personnes concernées et à l'exercice de leurs droits d'accès et de rectification d'après le Règlement Général sur la Protection des Données (RGPD) de l'Union Européenne.