

7.  
J1 = Joueur("Joueur 1",L[0])
8.  
jeu = [ Carte(i) for i in range (1,105)]  
shuffle(jeu)  
jeu\_initial = Paquet(jeu)  
distri = jeu\_initial.distribuer(2)  
Ordi = Joueur("Ordi", distri[0])  
J1 = Joueur("J1", distri[1])

## Exercice 2

1.  
SELECT nom  
FROM champignon  
WHERE lamelle = 'oui' and couleur = 'orange'
2.  
SELECT nom  
FROM champignon  
WHERE pied\_max = 0 AND chapeau\_max >= 15 AND chapeau\_min = 15 AND  
chapeau\_max = 15
3.  
id\_ordre
4.  
SELECT nom  
FROM champignon  
JOIN ordre ON id\_ordre = ordre.id  
WHERE classe = 'agaricomycètes'
5.  
INSERT INTO champignon  
VALUES  
(56, 'amanite solitaire', 4,'oui','blanc', 6, 20, 4, 10)
6.  
champignon(id, nom, #id\_ordre, lamelle, couleur, chapeau\_min,  
chapeau\_max, pied\_min, pied\_max, # id\_toxicite)  
ordre(id, nom, classe)  
toxicite(id\_tox, type, effet)
7.  
UPDATE champignon  
SET id\_toxicite = 1  
WHERE nom = 'amanite citrine'
8.  
SELECT nom  
FROM champignon  
JOIN ordre ON id\_ordre = ordre.id  
JOIN toxicite ON id\_toxicite = id\_tox  
WHERE nom = 'amanitales' AND type= 'très toxique'

9.
 

```

for e in liste_champi:
    if e.saison == 'été':
        print(e.nom)
      
```
10.
 

'12 minutes à feu moyen' est différent de 'feu moyen'.
11.
 

```

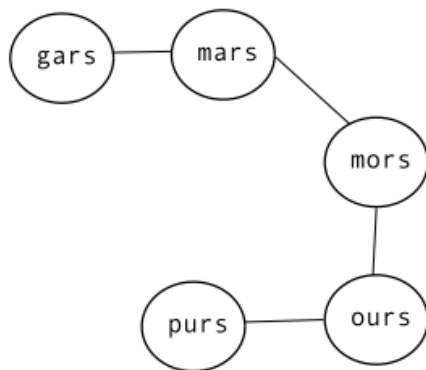
for c in range liste_champi:
    if c.nom == 'Lactaire délicieux':
        return recherche_textuelle(c.cuisson, 'feu moyen')
      
```

### Exercice 3

1. La pile est de type LIFO : dernier arrivé, premier sortie
2. La file est de type FIFO : premier arrivé, premier sortie
3.
 

si mot1 est voisin de mot2 alors mot2 est donc adapté à la situation, comme la relation est symétrique, un graphe non orienté est adapté.

4.



5.
 

```

def chaine_vers_tab(mot):
    tab_lettres = []
    for lettre in mot :
        tab_lettres.append(lettre)
    return tab_lettres
      
```
6.
 

Les éléments de tab sont les lettres du mot1. A chaque tour de boucle, on en retire la première occurrence de chaque lettre du mot2. Ainsi, il reste dans le tableau les lettres du mot2 qui ne sont pas dans le mot1. La longueur de tab correspond donc au nombre de lettres dont mot1 et mot2 diffèrent : c'est bien la distance entre eux.
7.
 

```

def renvoie_voisins(mot):
    tab_voisins = []
    for voisin_possible in TAB_MOTS :
        if distance(mot, voisin_possible) == 1 :
            tab_voisins.append(voisin_possible)
    return tab_voisins
      
```