Éléments de correction
sujet 11

Exercice 1

1.
```
class Carte() :
    def __init__(self, valeur) :
        self.valeur = valeur
        self.TdB = self.calcul_TdB()
```
2.
```
def calcul_TdB(self) :
    TdB = 0
    if self.valeur % 11 == 0 :
        TdB += 5
    if self.valeur % 10 == 0 :
        TdB += 3
    if self.valeur % 5 == 0 and self.valeur % 10 != 0 :
        TdB += 2
    if TdB == 0 :
        TdB = 1
    return TdB
```
3.
```
def est_superieure_a(self, autre) :
    return self.valeur > autre.valeur
```
4.
```
def affiher(self):
    for carte in self.contenu:
        print(carte.valeur)
def ajouter_arte(self, arte):
    self.contenu.append(carte)
```
5.
```
def nbr_TdB(self):
    nbr = 0
    for carte in self.contenu :
        nbr += carte.TdB
    return nbr
```

6.
```
def distribution(self, nbr):
    L=[Paquet([]) for _ in range(nbr)]
    for i in range(10):
        for j in range(nbr):
            carte = self.contenu.pop()
            L[j].ajouter_carte(carte)
    return L
```

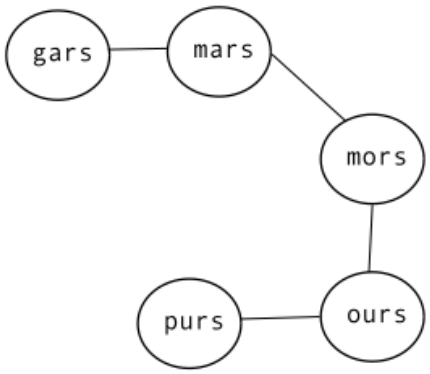
7.
```
J1 = Joueur("Joueur 1",L[0])
```
8.
```
jeu = [ Carte(i) for i in range (1,105)]
shuffle(jeu)
jeu_initial = Paquet(jeu)
distri = jeu_initial.distribuer(2)
Ordi = Joueur("Ordi", distri[0])
J1 = Joueur("J1", distri[1])
```

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
```

8.

TROISIEME TOUR
on défile 'mors'. Les voisins de 'mors' sont 'mars' et 'ours'. 'mars' est déjà dans parent. Ainsi on obtient :
`parent={'mars' : None,'gars':'mars,'mors':'mars', 'ours':'mors'}`
`file_voisins = ['ours']`

QUATRIEME TOUR
on défile 'ours'. Les voisins de 'ours' sont 'mors' et 'purs', 'mors' est déjà dans parent
On enfile 'murs'. Ainsi on obtient :
`parent = {'mars' : None, 'gars ': 'mars', 'mors ' : 'mars', 'ours':'mors', 'purs': 'ours'}}`

CINQUIEME TOUR
la variable mot est 'ours' qui est le mot final. La boucle s'arrête et renvoie le dictionnaire parent

9.

```
def renvoie_pile(parent, mot_final):
    ma_pile = Pile()
    mot = mot_final
    while mot != None :
        ma_pile.empiler(mot)
        mot = parent[mot]
    return ma_pile
```

10.

```
def construit_chemin(ma_pile):
    tab = []
    while not ma_pile.est_vide() :
        tab.append(ma_pile.depiler())
    return tab
```

11.

```
def chercher_chemin(mot1, mot2):
    parent = dic_parent(mot1, mot2)
    ma_pile = renvoie_pile(parent, mot2)
    return construit_chemin(ma_pile)
```