Exercice 1 — 6 points

Questions	*Contenu et notions*	*Capacités exigibles / Niveau*	*Éléments de réponses et commentaires*
1	écrire un constructeur	N1	
2	instruction conditionnelle - critères de divisibilité	N1	
3	créer une méthode	N1	
4	compléter des méthodes	N1	
5	écrire une méthode avec utilisation d'un compteur	N1	

```
class Carte() :
    def __init__(self, valeur) :
        self.valeur = valeur
        self.TdB = self.calcul_TdB()
```

```
def calcul_TdB(self) :
    TdB = 0
    if self.valeur % 11 == 0 :
        TdB = 5
    if self.valeur % 10 == 0 :
        TdB = TdB + 3
    if self.valeur % 5 == 0 and self.valeur % 10 != 0 :
        TdB = TdB + 2
    if TdB == 0 :
        TdB = 1
    return TdB
```

```
def est_superieure_a(self,autre) :
     #retourne True si carte1 (self) a une valeur supérieure à carte2 (autre)
     return self.valeur > autre.valeur
```

```
6   for carte in self.contenu:
7       print(carte.valeur,end=" ")
10  self.contenu.append(carte)
```

```
def nbr_TdB(self):
    nbr = 0
    for carte in self.contenu :
        nbr = nbr + carte.TdB
    return nbr
```

Exercice 1	6 points		

6	création d'une méthode avec gestion des autres méthode, manipulation de liste	N1	`def distribution(self,nbr):`
			`  # initialiser la liste avec 'nbr' paquets vides`
			`  L=[Paquet([]) for i in range(nbr)]`
			
			`  # distribution des 10 cartes aux nbr joueurs`
			`  for i in range(10):`
			`  for j in range(nbr):`
			`  # récupération de la 1ère carte en haut du paquet`
			`  carte = self.contenu.pop()`
			`  # ajout de la carte dans la main du joueur`
			`  L[j].ajouter_carte(carte)`
			`  return L`

7	instancier un objet d'une classe	N1	`J1 = Joueur("Joueur 1",L[0])`

8	création liste par compréhension, instanciation, appel de méthodes	N2	`3  jeu = [ Carte(i) for i in range (1,105)]`
			`7  jeu_initial = Paquet(jeu)`
			`9  distri = jeu_initial.distribution(2)`

Exercice 2 — 6 points

Questions	*Contenu et notions*	*Capacités exigibles / Niveau*	*Éléments de réponses et commentaires*

1	Base de données requete	N2	`SELECT nom`
			`FROM champignon`
			`WHERE lamelles = 'oui' and couleur = 'orange';`

2	Base de données requete	N2	`SELECT nom`
			`FROM champignon`
			`WHERE pied_max = 0 and chapeau_max >= 15 and chapeau_min <= 15;`

3	bases de données, vocabulaire	N2	`id_ordre`

4	reqête SQL avec jointure	N3	`SELECT champignon.nom`
			`FROM champignon`
			`JOIN ordre ON champignon.id_ordre = ordre.id`
			`WHERE ordre.classe = 'agaricomycètes'  ;`
			ou
			`SELECT champignon.nom`
			`FROM champigon, ordre`
			`WHERE champignon.id_ordre = ordre.id`
			`AND ordre.classe = 'agaricomycètes';`

5	requete de mise à jour	N2	`INSERT INTO champignon VALUES (56, 'amanite solitaire', 4, 'oui',` `'blanc',6, 20, 4, 10);`

6	requete de mise à jour	N2	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	base de données, requête de mise à jour	N2	`UPDATE champignon`
			`SET id_toxicite = 1`
			`WHERE nom = 'amanite citrine';`

8	programmation Python, POO	N3	`SELECT champignon.nom`
			`FROM champignon`
			`JOIN ordre ON champignon.id_ordre = ordre.id`
			`JOIN toxicite ON champignon.id_toxicite = toxicite.id_tox`
			`WHERE ordre.nom = 'amanitales' and toxicite.type = 'très toxique' ;`

Exercice 2	6 points		

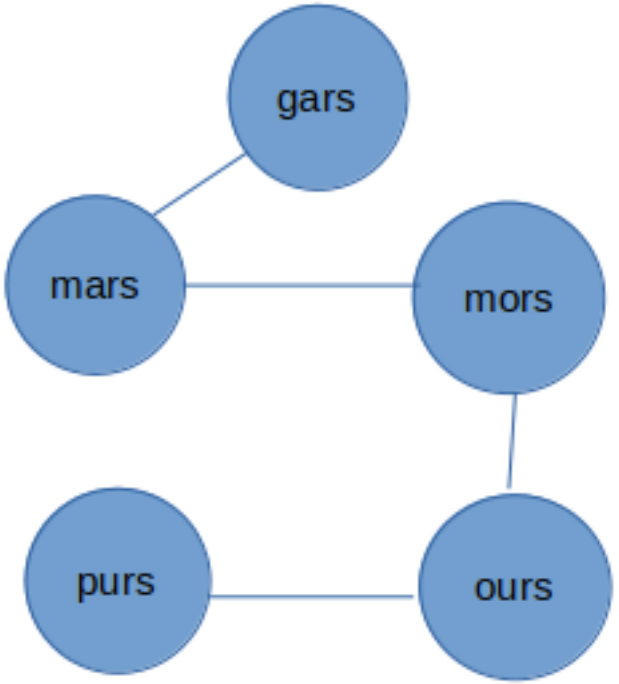
9	POO, python	N2	`for e in liste_champi:`
			`  if e.saison == 'été':`
			`  print(e.nom)`

10	POO, programmation python	N2	L'attribut `cuisson` du lactaire délicieux est égal à '12 minutes à feu moyen' et il est comparé à 'feu moyen'. Ces valeurs ne sont pas égales.

11	POO et programmation python	N3	`for c in range liste_champi:`
			`  if c. nom == 'Lactaire délicieux':`
			`  return recherche_textuelle(c.cuisson, 'feu moyen')`

Exercice 3 — 8 points

Questions	*Contenu et notions*	*Capacités exigibles / Niveau*	*Éléments de réponses et commentaires*
1	Listes, Piles, Files : strutures linéaires	N1	Une pile est une structure où l'élément empilé en dernier est celui qui sera dépilé en premier (structure LIFO). L'interface d'une Pile permet de `creer une pile vide`, `savoir si une pile est vide`, `empiler un élément au sommet de la pile`, `dépiler le sommet de la pile`.
2	Listes, Piles, Files : strutures linéaires	N1	Une file est une structure où le permier élément enfilé est celui qui sera défilé en premier (structure FIFO). L'interface d'une `File` permet de `créer une file vide`, `savoir si une file est vide`, `enfiler un élement en queue de file`, `défiler l'élément en tête de file`.
3	Graphes : structures relationnelles	N1	si mot1 est voisin de mot2 alors mot2 est voisin de mot1. Un graphe non orienté est donc adapté à la situation.
4	Graphes : structures relationnelles	N1-2	



Exercice 3	8 points		
5	structures linéaires	N1-2	

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

6	Structures linéaires	N2	Les élements de *tab* sont les lettres du mot1. A chaque tour de boucle, on en retire la première occurence 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	Graphes : structures relationnelles	N2	

```
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	Algorithmes sur les graphes	N3	

```
TROISIEME TOUR
* on défile 'mors'
* les voisins de 'mors' sont 'mars' et 'ours'. 'mars' est déjà dans parent. Le
enfile 'ours'. Ainsi on obtient :
    * parent={'mars:None,'gars':'mars,'mors':'mars','ours':'mors'}
    * file_voisins contient de la tête à la queue uniquement 'ours'
QUATRIEME TOUR
    * on défile 'ours'
    * les voisins de 'ours' sont 'mors' et 'purs'. 'mors' est déjà dans parent
    enfile 'murs'. Ainsi on obtient :
    * parent = {'mars : None, 'gars': 'mars, 'mors':'mars','ours':'mors','purs
CINQUIEME TOUR
la variable mot est 'ours' qui est le mot final. La boucle s'arrête et renvoie
```

9	Algorithmes sur les Graphes et structures linéaires	N3	

```
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
```

Exercice 3	8 points		

10	Algorithmes sur les Graphes et structures linéaires	N2-3	`def construit_chemin(ma_pile):`
			`  tab = []`
			`  while not ma_pile.est_vide() :`
			`  tab.append(ma_pile.depiler())`
			`  return tab`

11	Algorithmes sur les Graphes et structures linéairesGraphes : structures relationnelles	N2-3	`def chercher_chemin(mot1, mot2):`
			`  parent = dic_parent(mot1, mot2)`
			`  ma_pile = renvoie_pile(parent, mot2)`
			`  return construit_chemin(ma_pile)`