Correction 26-NSIJ2JA1

Exercice 1

1. C'est l'indice 5.

2. On obtient la grille suivante :

5	3	8
1	0	2
7	6	4

3. `tab == [i for i in range(9)]`

4.
```
    def est_gagnant(self):
        return self.tab == [i for i in range(9)]
```

5.
```
def indice(self, numero):
    assert type(numero) == int, 'numero doit être entier'
    assert 0 <= numero < 9, 'numero de case non valide'
    i = 0
    while self.tab[i] != numero and i < len(self.tab):
        i = i + 1
    return i
```

6.
```
def jouer(self, numero):
    if self.est_possible(numero):
        i = self.indice(numero)
        j = self.indice(0)
        self.tab[i] = numero
        self.tab[j] = 0
```

7.
```
def melanger(self, n):
    precedent = None
    i = 0
    while i < n:
        possibilites = self.coups_possibles()
        choix = random.choice(possibilites)
        if choix != precedent:
            self.jouer(choix)
            precedent = choix
            i = i + 1
```

8. On a, en lisant de haut en bas :

Numéro	Pile
2	| 1 4 5 >
5	| 1 4 >
4	| 1 >
1	| >

9.
```
def resoudre(self):
    self.mode_resolution = True
    while not self.est_gagnant():
        numero = self.pile.depiler()
        self.jouer(numero)
        print("coup joué :", numero)
```

10. Si on oublie de passer en mode résolution automatique, alors la méthode resoudre va continuer à empiler les coups possibles à chaque étape, même après avoir joué un coup, via la méthode `jouer`. Cela signifie que la pile va se remplir indéfiniment avec les coups possibles, et la méthode ne pourra jamais atteindre une condition d'arrêt, car elle continuera à empiler des coups sans jamais les dépiler pour les jouer. La boucle ne terminerait donc pas.

11.
```
def jouer(self, numero):
    if self.est_possible(numero):
        i = self.indice(numero)
        j = self.indice(0)
        self.tab[i] = numero
        self.tab[j] = 0
        if not self.mode_resolution:
            if self.pile.est_vide():
                self.pile.empiler(numero)
            else:
                precedent = self.pile.depiler()
                if numero != precedent:
                    self.pile.empiler(precedent)
                    self.pile.empiler(numero)
```

Exercice 2

Partie A

1. C'est la table `participation`, car une personne peut jouer plusieurs parties, donc elle ne peut être clé primaire de cette table.

2. Ce n'est pas `id_partie`, car une partie peut être jouée par plusieurs personnes.

3.
```
INSERT INTO personne
VALUES (42, "theorie", '2022-12-14');
```

4.
```
SELECT id_partie FROM participation
JOIN personne ON participation.id_pers = personne.id_pers
WHERE personne.pseudo = "test";
```

5.
```
DELETE FROM participation
WHERE id_pers = 8;

DELETE FROM personne
WHERE id_pers = 8;
```

Partie B

6.
```
def indice(lettre, ordre):
    for i in range(len(ordre)):
        if ordre[i] == lettre:
            return i
```

7.
```
def comparer (mot1, mot2, ordre):
    i= 0
    while i < len(mot1) and i < len(mot2):
        il = indice(mot1[i], ordre)
        i2 = indice(mot2[i], ordre)
        if il < i2:
            return True
        elif il > i2:
            return False
        i += 1
    return len(mot1) < len(mot2)
```

8.
```
def premiere_diff(mot1, mot2):
    i= 0
    while i < len(mot1) and i < len(mot2):
        if mot1[i] != mot2[i]:
            return i
        i += 1
    return min(len(mot1), len(mot2))
```

9. On trouve : `{'u': ['a', 'y'], 'y': ['i', 'a'], 'a': ['i'], 'o': ['u'], 'e': ['a']}`.

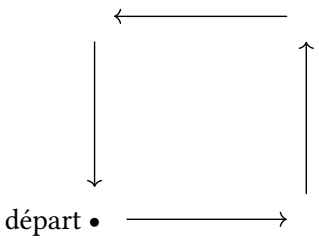
10. C'est un parcours en profondeur : si on ne connaît pas un nouveau sommet `v`, on le visite tout de suite avec l'appel récursif de la ligne 6.

11.
```
def trier(mots):
    adj = dico_adj(mots)
    tri = []
    deja_vus = []
    for s in adj:
        if s not in deja_vus:
            deja_vus.append(s)
            parcours(adj, s, deja_vus, tri)
    tri.reverse()
    return tri
```

Exercice 3

Partie A

1. Avec le parcours `'4(AG)'`, le robot va faire le parcours suivant :



2.
```
def caracteres_valides(chaine):
    valides = '0123456789ADG()'
    intrus = [c for c in chaine if c not in valides]
    return len(intrus) == 0 # Vrai si la liste est vide, faux sinon
```

3.
```
def entiers_valides(chaine):
    chiffres = '0123456789'
    if chaine[len(chaine)-1] in chiffres:
        return False
    for indice in range(1, len(chaine)):
        if chaine[indice] == ')':
            if chaine[indice - 1] in chiffres:
                return False
    return True
```

4.
```
def parenthesage_correct(chaine):
    parentheses = 0
    for c in chaine:
        if c == '(':
            parentheses += 1
        elif c == ')':
            parentheses -= 1
            if parentheses < 0:
                return False
    return parentheses == 0
```

5. On a :

n° d'itération	nombre	indice
1	1	3
2	17	4
3	179	5

6.
```
def lire_bloc (chaine, indice):
    indice = indice + 1
    caractere = chaine[indice]
    bloc = ""
    compteur = 1
    while compteur > 0:
        bloc = bloc + caractere
        indice = indice + 1
        caractere = chaine[indice]
        if caractere == '(':
            compteur = compteur + 1
        if caractere == ')':
            compteur = compteur - 1
    return (bloc, indice)
```

7.
```
def lire_parcours(chaine):
    chiffres = "0123456789"
    indice = 0
    nombre = 1
    while indice < len(chaine):
        car_lu = chaine [indice]
        if car_lu in 'AGD':       # commande simple
            for k in range(nombre):
                execute_mouvement(car_lu)
            nombre = 1
        elif car_lu in chiffres:  # répétition
            t = lire_nombre(chaine, indice)
            nombre = t[0]
            indice = t[1]
        elif car_lu == '(':       # début d'un bloc
            t = lire_bloc(chaine, indice)
            bloc = t[0]
            indice = t[1]
            for k in range (nombre):
                lire_parcours(bloc)
            nombre = 1
        indice = indice + 1
```

Partie B

8. On a la nouvelle table suivante :

destination	prochain robot	distance
4	4	1
46	4	2
22	22	1
57	22	2

9. On a la nouvelle table suivante :

destination	prochain robot	distance
4	4	1
46	4	2
22	22	1
57	22	2
87	87	1
63	87	2
36	87	3

Partie C

10.
```
    def ajouter_voisin(self, id_voisin):
        self.table_routage[id_voisin] = {"prochain": id_voisin, "distance": 1}
```

11.
```
    def nombre_sauts(self, identifiant):
        if identifiant not in self.table_routage:
            return 16
        else:
            return self.table_routage[identifiant]["distance"]
```

12.
```
    def voisins(self):
        return [e for e in self.table_routage]
```

13.
```
    def communiquer_extrait_table(self, voisin):
        return {k:v for k, v in self.table_routage.items() if v["prochain"] != voisin}
```