Proposition de correction

Exercice 1

Partie A

Q1.a

1024

Q1.b

tab[1]["ad_lieu"]

Q2

- Proposition 2
- Proposition 3

Q3

renvoie le nombre de parking de nature « Sortie d'autoroute » dont les places disponibles sont ≥ à 100

ici : 0

Q4

```
def insee_max_places(table):
    """
    Entrée : un tableau de dictionnaires
    Sortie : une chaîne de caractères (le code INSEE du lieu possédant le plus grand nombre de
    places)
    """
    maxi = table[0]["nb_places"]
    code_insee = table[0]["insee"]
    for dico in table:
            if dico["nb_places"] > maxi :
                    maxi = dico["nb_places"]
                    code_insee = dico["insee"]
return code_insee
```

Q5

```
def moyenne_par_type(table, nom_type):
    """
    Entrée : un tableau de dictionnaires et une chaîne de caractères (type de lieu)
    Cette fonction renvoie, parmi les lieux de covoiturage dont le type est nom_type,
    la moyenne du nombre de places pour le type choisi.
    Sortie : un flottant
    """
    nb_places = [dico["nb_places"] for dico in table if dico["type"] == type]
    moyenne = sum(nb_places) / len(nb_places) if len(nb_places) else 0
    return moyenne
```

Partie B

Q6

code_insee est une clef primaire

Q7

SELECT id_lieu, code_insee

FROM LIEU

WHERE type = "Sortie autoroute"

Q8

SELECT COUNT(LIEU.id_lieu)

FROM LIEU, COMMUNE

WHERE COMMUNE.departement = "JURA"

AND LIEU.code_insee = COMMUNE.code_insee

Q9.a

la clé étrangère code_insee enregistrée dans la table LIEU pointe sur la clé primaire code_insee de la table COMMUNE

Q9.b

UPDATE LIEU

SET code_insee = "40146"

WHERE code_insee = "40714"

Exercice 2

Partie A

Q1

gimp

Q2

gnome_shell

Q3

1611

Q4

bash

Q5

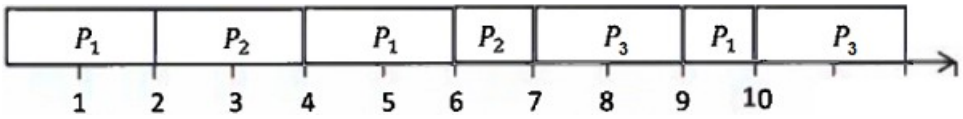
- 1
- 1611
- 5571
- 5622
- 5629

Partie B

Q6

- Prêt : en attente d'exécution
- bloqué : en attente de ressource
- élu : en exécution

Q7



Partie C

Q8

NON ET		
a	b	NON (a ET b)
0	0	1
0	1	1
1	0	1
1	1	0

Q9

a	b	NON a	NON b	(NON a) OU (NON b)
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Exercice 3

Partie A

Q1

```
class Personnage:
    def _ init_ (self, nom_clan, pts_vie, pts_force):
            self.clan = nom_clan
            self.vie = pts_vie
            self.force = pts_force
```

Q2

- attributs : clan, vie, force
- méthodes : __init__(), attaque(), defense()

Q3

```
Luther = Personnage("Umbrella", 100, 15)
```

Q4

```
def attaque(self, autre_perso : Personnage) -> int :
    autre_perso.vie -= self.force
    return int(autre_perso.vie > 0)
```

Q5

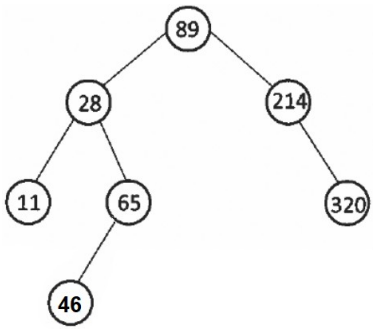
une pile

Partie B

Q6.a

3

Q6.b



Q7.a

infixe

Q7.b

11, 28, 65, 89, 214, 320

Q8

```
def recherche_dichotomique (arbre, ref_materiel):
    """
    Entrée : un arbre binaire de recherche dont les noeuds sont des tableaux ; un entier (la référence du matériel recherché).
    Sortie : un entier -1 si la référence du matériel n'est pas présente dans l'arbre ou la quantité si la référence est présente.
    """
    if est_vide(arbre):         # référence absente de l'arbre
            return -1
    elif ref materiel < valeur_racine(arbre)[0]:         # recherche dans le sous-arbre gauche
            return recherche_dichotomique (gauche(arbre), ref_materiel)
    elif ref materiel > valeur_racine(arbre)[0]:         # recherche dans le sous-arbre droit
            return recherche_dichotomique (droit(arbre), ref_materiel)
    else:                       # référence présente, on renvoie la quantité correspondante
            return valeur racine(arbre)[1]
```