Proposition de correction

Exercice 1

Q1

6, 7, 8, 9, 10

Partie A

Q2

```
def effectif(val : int, lst : list) -> int:
    """
    @param val -- une valeur
    @param lst -- une liste
    @return le nombre d'apparitions de val dans lst.
    """
    total = 0
    for elt in lst:
        if elt == val:
            total += 1
    return total
```

Q3

9

Q4

```
def majo_abs1(lst : list) -> int:
    """
    @param lst -- une liste
    @return élément absolument majoritaire s'il existe, None sinon
    """
    majoritaire = len(lst) // 2
    for elt in lst:
        if effectif(elt, lst) > majoritaire:
            return elt
    return None
```

Q5

9 + 1 = 10

Partie B

Q6

```
def eff_dico(lst):
    dico_sortie = {}
```

```
    for elt in lst :
        if elt in dico_sortie:
            dico_sortie[elt] += 1
        else:
            dico_sortie[elt]  = 1
    return dico_sortie
```

Q7

```
def majo_abs2(lst : list) -> int:
    """
    @param lst -- une liste lst
    @return élément absolument majoritaire s'il existe, None sinon
    """
    majoritaire = len(lst) // 2
    effectif    = eff_dico(lst)
    for clef, valeur in effectif.items():
        if valeur > majoritaire:
            return clef
    return None
```

partie C

Q8

Lst[0]

Q9

soit n ∈ N*, si ∃ m ∈ N* / m > n/2
alors ∃ m' ∈ N* / m' = m/2 > n/4
soit m' > 1/2 n/2

Q10

```
fonction majoritaire(liste : tableau d'entiers) : entier
début
    n := taille(liste)
    si ( n = 1 ) alors
        renvoyer liste[0]          { cas de base }
    sinon
        liste_gauche := liste[0..n/2 - 1]
        liste_droite  := liste[n/2..n - 1]
        nb_majoritaire := majoritaire(liste_gauche)
        si ( nb_majoritaire <> Nil ) alors
            si ( effectif(nb_majoritaire, liste) > n/2 ) alors
                renvoyer nb_majoritaire
        nb_majoritaire := majoritaire(liste_droite)
        si ( nb_majoritaire <> Nil ) alors
            si ( effectif(nb_majoritaire, liste) > n/2 ) alors
                renvoyer nb_majoritaire
fin
```

Q11

```
def majo_abs3(lst):
    n = len(lst)
    if n == 1:
        return lst[0]    # cas de base
    else:
        lst_g = lst[:n//2]
        lst_d = lst[n//2:]
        maj_g = majo_abs3(lst_g)
        maj_d = majo_abs3(lst_d)
        if maj_g is not None:
            eff = effectif(maj_g, lst)
            if eff > n/2:
                return maj_g
        if maj_d is not None:
            eff = effectif(maj_d, lst)
            if eff > n/2:
                return maj_d
```

Exercice 2

Q1

```
[
        2*
        (
                i+1
                )                       => mal parenthésé
                -3
        )
        for i in range
        (
                3, 10
        )
]
```

Partie A

Q2

```
def compte_ouvrante(txt : str) -> int:
    """
    @param txt -- une chaine de caractères
    @return le nombre de parenthèses ouvrantes qu'il contient
    """
    total = 0
    for c in txt:
        if c in "[{(":
```

```
            total += 1
    return total
```

Q3

```
def compte_fermante(txt : str) -> int:
    """
    @param txt -- une chaine de caractères
    @return le nombre de parenthèses fermantes qu'il contient
    """
    total = 0
    for c in txt:
        if c in "]})":
            total += 1
    return total
```

Q4

```
def bon_compte(txt : str) -> bool:
    """
    @param txt -- une chaine de caractères
    @return True si txt a autant de parenthèses ouvrantes que parenthèses fermantes
            et False sinon.
    """
    return compte_ouvrante(txt) == compte_fermante(txt)
```

Q5

```
assert compte_ouvrante(")(") == 1
assert compte_fermante(")(") == 1
assert bon_compte(")(") is True
```

Partie B

Q6

```
class Pile:
    def __init__(self):
        self.contenu = []

    def est_vide(self):
        return len(self.contenu) == 0

    def empiler(self, elt):
        self.contenu.append(elt)

    def depiler(self):
        if self.est_vide():
            return "La pile est vide."
        return self.contenu.pop(-1)
```

Q7

- 16 (caractères) + 2 (parenthèses ouvrantes) + 3x2 (parenthèses fermantes) = 24 comparaisons
- n (caractères) + 2xn/2 (parenthèses ouvrantes) + 3xn/2 (parenthèses fermantes) = 7/2n comparaisons

Q8

```
def est_bien_parenthesee(txt : str) -> bool:
    """
    @param txt -- une chaine de caractères
    @return True si txt est bien parenthèsé
            et False sinon.
    """
    couples = { ']':'[', '}':'{', ')':'(' }
    p = Pile()
    for c in txt:
        if c in "[{(":
            p.empiler(c)
        elif c in "]})":
            if p.est_vide() or ( p.depiler() != couple[c] ):
                return False
    return p.est_vide()
```

Exercice 3

Partie A

Q1

non, car un même titre peut être repris dans différents albums

Q2

Welcome too the Jungle Appetite for Destruction

Q3

SELECT titre

FROM Chanson

WHERE album = ‘Showbiz’

ORDER BY titre

Q4

INSERT INTO Chanson

VALUES(10, ‘Megalomania’, ‘Hullabaloo’, ‘Muse’)

Q5

UPDATE Chanson

SET titre = ‘Welcome to the Jungle’

WHERE id = 7

Partie B

Q6

- Élimination des Redondances
- Amélioration de la Cohérence et de l'Intégrité des Données
- Facilité de Mise à Jour
- Optimisation des Performances

Q7

clef étrangère qui met en relation la table Chanson avec la table Album

Q8

- Chanson(id, titre, #id_album)
- Album(id, titre, année, #id_groupe)
- Groupe(id, nom)

Q9

SELECT Album.titre

FROM Album, Chanson

WHERE Chanson.titre = ‘Showbiz’

AND Album.id = Chanson.id_album

ORDER BY Album.titre

Q10

SELECT Chanson.titre, Album.titre

FROM Album, Chanson, Groupe

WHERE Groupe.nom = ‘Muse’

AND Album.id_groupe = Groupe.id

AND Chanson.id_album = Album.id

ORDER BY Chanson.titre, Album.titre

Q11

Donne le nombre d’Albums réalisés par le groupe Muse

Partie C

Q12

```
assert ordre_lex("", "a") == True
assert ordre_lex("b", "a") == False
assert ordre_lex("aaa", "aaba") == True
```

Q13

```
def ordre_lex(mot1, mot2):
    if mot1 == "":
```

```
        return True
    elif mot2 == "":
        return False
    else:
        c1 = mot1[0]
        c2 = mot2[0]
        if c1 < c2:
            return True
        elif c1 > c2:
            return False
        else:
            return ordre_lex(mot1[1:], mot2[1:])
```

Q14

```
def ordre_lex(mot1, mot2):
    i = 0
    while (i < len(mot1)) and (i < len(mot2)):
        c1, c2 = mot1[i], mot2[i]
        if c1 < c2:
            return True
        if c1 > c2:
            return False
        i += 1
    return len(mot1) < len(mot2)
```