Proposition de correction

Exercice 1

Q1a

salle	marque_ordi
012	HP
114	Lenovo
223	Dell
223	Dell
223	Dell

Q1b

nom_ordi	salle
Gen-24	012
Tech-62	114
Gen-132	223

Q2

SELECT * FROM Ordinateur WHERE annee >= 2017 ORDER BY annee ASC

Q3a

Des ordinateurs différents peuvent avoir le même n° de salle.

Q3b

Imprimante(nom_imprimante : String, marque_imp : String, modele_imp : String, salle : String, #nom_ordi : String)

indique une clef étrangère sur la relation Ordinateur

Q4a

INSERT INTO Videoprojecteur VALUES(315, 'NEC', 'ME402X', false)

Q4b

SELECT Ordinateur.salle, Ordinateur.nom_ordi, Videoprojecteur.marque_video

FROM Ordinateur, Videoprojecteur

WHERE Ordinateur.video = true AND Ordinateur.salle = Videoprojecteur.salle AND Videoprojecteur.tni = true

Exercice 2

Q1

- Plus grande intégration du Hardware (taille composants plus petite)
- Plus grande rapidité de traitement (programmation Hardware des fonctions)

Q2

Les application SGBD et CAO attendent mutuellement la Donnée D3. Il y a interblocage (deadlock).

D2, D4 et D5 ne pourront pas être libérées.

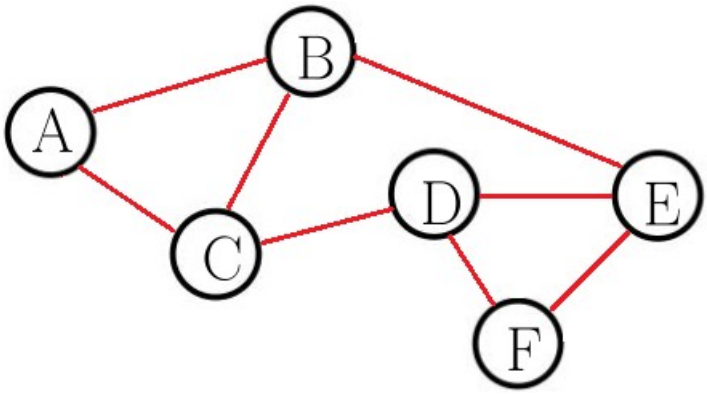
Le traitement de texte ne pourra obtenir D2.

Le tableur ne pourra obtenir D5

Q3

A → B → F

Q4



Exercice 3

Q1a

```
def total_hors_reduction(articles_panier : list) -> list:
    """ renvoie le total des prix des articles du panier."""
    total = 0
    for article in articles_panier:
        total += article
    return total
```

Q1b

```
def offre_bienvenue(tab):
    """ tableau -> float """
    somme    = 0
    longueur = len(tab)
```

```
    if longueur > 0 :
        somme = tab [0] * 0.8
    if longueur > 1 :
        somme = somme + (somme * 0.7)
    if longueur > 2 :
        for i in range(2, longueur ):
            somme = somme + tab[i]
    return somme
```

Q2

```
def prix_solde(prix_articles : list) -> float:
    """ renvoie le total des prix de ces articles
    lorsqu'on leur applique la réduction des soldes."""
    total = total_hors_reduction(prix_articles)
    if len(prix_articles) >= 5:
        total *= 0.5
    else:
        total *= 1 - (0.1 * len(prix_articles))
    return total
```

Q3a

```
def minimum(tab : list) -> float:
    """ renvoie la valeur minimum présente dans le tableau."""
    if len(tab) == 0:
        return 0
    min = tab[0]
    for i in tab:
        if i < min:
            min = i
    return min
```

Q3b

```
def offre_bon_client(prix_articles : list) -> float:
    """ renvoie le total à payer
    lorsqu'on leur applique l'offre bon client."""
    if len(prix_articles) >= 2:
        return total_hors_reduction(prix_articles) - minimum(prix_articles)
    elif len(prix_articles) == 1:
        return prix_articles[0]
    else:
        return 0
```

Q4a

Tab = [30.5, **15.0**, 35.0, **6.0**, 20.0, 10.5, 5.0]

promotion déstockage = 122.0 – 15.0 – 6.0 = 101.0€

Q4b

Tab = [35.0, 30.5, **20.0**, 15.0, 10.5, **6.0**, 5.0]

promotion déstockage = 122.0 – 20.0 – 6.0 = 96.0€

Q4c

Le client doit trier ses articles par prix décroissants.

Exercice 4

Q1a

- Racine = "Lea"
- Feuilles = ["Marc", "Lea", "Claire", "Theo", "Marie", "Louis", "Anne", "Kevin"]

Q1b

```
def vainqueur(arb : object) -> str:
    """ renvoie le nom du vainqueur du tournoi."""
    return racine(arb)
```

Q1c

```
def finale(arb : object) -> list:
    """ renvoie les noms des deux compétiteurs finalistes."""
    return [racine(gauche(arb)), racine(droite(arb))]
```

Q2a

```
    def occurrences(arb : object, nom : str, total : int = 0) -> int:
        """ renvoie le nombre d'occurrences du joueur """
        if est_vide(arb):
            return 0
        else:
            total = int(racine(arb) == nom)
            return total + occurrences(gauche(arb), nom, total) + occurrences(droit(arb), nom, total)
```

Q2b

```
def a_gagne(arb : object, nom : str) -> bool:
    """ détermine si le joueur a gagné au moins un match dans la compétition """
    return occurrences(arb, nom) > 1
```

Q3a

La racine de l'arbre détermine le nom du gagnant et ne doit pas être pris en compte.

Q3b

```
def nombre_matchs(arb, nom):
    """ arbre_competition , str -> int """
    total = occurrences(arb, nom)
    if racine(arb) == nom:
        return total - 1
    else:
        return total
```

Q4

```
def liste_joueurs(arb):
    """ arbre_competition -> tableau """
    if est_vide(arb):
        return []
    elif est_vide(gauche(arb)) and est_vide(droit(arb)) :
        return [racine(arb)]
    else:
        return liste_joueurs(gauche(arb)) + liste_joueurs(droit(arb))
```

Exercice 5

Q1a

- La file F sera vide.
- La pile P contiendra : "jaune" "rouge" "jaune" "vert" "rouge" → dépilement

Q1b

```
def taille_file(F : object) -> int:
    """ renvoie le nombre d'éléments de la file"""
    total = 0
    File  = creer_file_vide ()
    while not est_vide(F):
        total += 1
        enfiler(File, defiler(F))
    F = File    # restoration de F
    return total
```

Q2

```
def former_pile(F : object) -> object:
    """ renvoie une pile contenant les mêmes éléments que la file """
    P = creer_pile_vide ()
    L = []
    while not est_vide(F):
        L.append(defiler(F))
```

```
    L.reverse()    # permet de respecter le même ordre entre empilement et enfilement
    for i in L:
        P.empiler(i)
    return P
```

Q3

```
def nb_elements(F : object, elt : object) -> int:
    """ renvoie le nombre de fois où elt est présent dans la file F"""
    total = 0
    File  = creer_file_vide ()
    while not est_vide(F):
        temp = defiler(F):
        if elt == temp:
            total += 1
        enfiler(File, temp)
    F = File    # restoration de F
    return total
```

Q4

```
def verifier_contenu(F : obect, nb_rouge : int, nb_vert : int, nb_jaune : int) -> bool:
    """ renvoie True si "rouge" apparaît au plus nb_rouge fois dans la file F,
    "vert" apparaît au plus nb_vert fois dans la file F
    et "jaune" apparaît au plus nb_jaune fois
    Elle renvoie False sinon"""
    r, v, j = nb_element(F, 'rouge'), nb_element(F, 'vert'), nb_element(F, 'jaune'),
    return r <= nb_rouge and v <= nb_vert and j <= nb_jaune
```