2021 amérique du nord sujet 1

Exercice 1

1. a. [0.5 point]

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

1. b. [0.5 point]

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

2. [0.5 point]

```
SELECT * FROM Ordinateurs WHERE annee >= 2017 ORDER BY annee ASC;
```

3. a. [0.5 point] Car elle n'est pas unique.

3. b. [0.5 point]

Imprimantes(nom_imprimante : String, marque_imp : String, modele_imp : String, salle : String, nom_ordi : String)

nom_ordi est une clé étrangère pour la table Ordinateurs.

4. a. [0.5 point]

```
INSERT INTO Videoprojecteur VALUES ("315", "NEC", "ME402X", false);
```

4. b. [1 point]

```
SELECT salle, nom_ordi, marque_video
FROM Ordinateurs
JOIN Videoprojecteur ON Ordinateurs.salle = Videoprojecteur.salle
WHERE video = true AND tni = true;
```

Exercice 2

1. [1 point] L'encombrement et la consommation sont plus faibles.

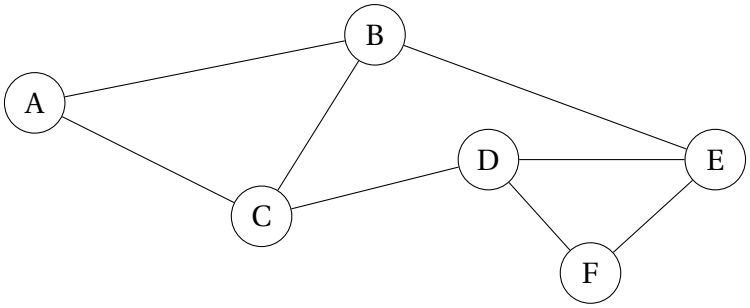
2. [1 point]

- D1 est mobilisé par le traitement de texte qui attend D2;
- D2 est mobilisé par le SGBD qui attend D4;
- D4 est mobilisé par la CAO qui attend D5;
- D5 est mobilisé par le tableur qui attend D1;

Nous avons bien une boucle. C'est ce qu'on appelle *l'interblocage*.

3. [1 point] A-B-E-F

4. [1 point]



Exercice 3

1. a. [0.5 point]

```
def total_hors_reduction(tab):
    s = 0
    for p in tab:
        s = s + p
    return s
```

1. b. [0.5 point]

```
def offre_bienvenue(tab):
    """ tableau -> float """
    somme = 0
    longueur = len(tab)
    if longueur > 0 :
        somme = tab[0]*0.8
    if longueur > 1 :
        somme = somme + tab[1]*0.7
    if longueur > 2 :
        for i in range(2, longueur) :
            somme = somme + tab[i]
    return somme
```

2. [1 point]

```
def prix_solde(tab):
    total = total_hors_reduction(tab)
    longueur = len(tab)
    if longueur >= 5 :
        somme = total*0.5
    elif longueur == 4 :
        somme = total*0.6
    elif longueur == 3 :
        somme = total*0.7
    elif longueur == 2 :
        somme = total*0.8
    elif longueur == 1 :
        somme = total*0.9
    return somme
```

3. a. [0.5 point]

```
def minimum(tab):
    min = 0
    for p in tab:
        if p < min:
            min = p
    return min
```

3. b. [1 point]

```
def offre_bon_client(tab):
    mini = minimum(tab)
    total = total_hors_reduction(tab)
    if len(tab) >= 2:
        return total - mini
    else:
        return total
```

4. a. [1 point] `[30.5, 15.0, 20.0, 6.0, 5.0, 35.0, 10.5]`

4. b. [1 point] `[35.0, 30.5, 20.0, 15.0, 10.5, 6.0, 5.0]`

4. c. [0.5 point] Il faut trier les articles par ordre de prix décroissant. Le client peut donc utiliser un algorithme de tri comme le tri par sélection.

Exercice 4

1. a. [0.5 point] La racine est "Léa". L'ensemble des valeurs des feuilles est : "Marc", "Lea", "Claire", "Theo", "Marie", "Louis", "Anne" et "Kevin".

1. b. [0.25 point] Le vainqueur est celui qui est à la racine.

```
def vainqueur(arb):
    return racine(arb)
```

1. c. [0.5 point]

```
def finale(arb):
    return [racine(gauche(arb)), racine(droit(arb))]
```

2. a. [0.5 point] On utilise une fonction récursive.

```
def occurences(arb, nom):
    if est_vide(arb):
        return 0
    else:
        if racine(arb) == nom:
            return 1 + occurences(gauche(arb), nom) + occurences(droit(arb), nom)
        else:
            return occurences(gauche(arb), nom) + occurences(droit(arb), nom)
```

2. b. [0.5 point] Un joueur à gagné au moins un match si son nom apparaît plus d'une fois.

```
def gagne(arb, nom):
    return occurences(arb, nom) > 1
```

3. a. [0.25 point] Cela ne fonctionne pas par le vainqueur apparaît à la racine sans effectuer de match.

3. b. [0.5 point]

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

4. [0.75 point]

```
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

1. a. [0.5 point] La file F est vide et P contient les éléments suivants :

"rouge"
"vert"
"jaune"
"rouge"
"jaune"

1. b. [0.75 point]

```
def taille_file(F):
    """File -> Int"""
    FS = creer_file_vide()
    k = 0
    while not(est_vide(F)):
        k = k + 1
        enfiler(FS, defiler(F))
    while not(est_vide(FS)):
        enfiler(F, defiler(FS))
    return k
```

2. [0.75 point] On utilise une pile intermédiaire pour retourner la pile.

```
def former_pile(F):
    """File -> Pile"""
    Q = creer_pile_vide()
    while not(est_vide(F)):
        enpiler(Q, defiler(F))
    P = creer_pile_vide()
    while not(est_vide(Q)):
        enpiler(P, depiler(Q))
    return P
```

3. [1 point]

```
def nb_elements(F, elt):
    """File -> Int"""
    FS = creer_file_vide()
    k = 0
    while not(est_vide(F)):
        e = defiler(F)
        if e == elt:
            k = k + 1
        enfiler(FS, e)
    while not(est_vide(FS)):
        enfiler(F, defiler(FS))
    return k
```

Une autre possibilité sans la file intermédiaire :

```
def nb_elements(F, elt):
    """File -> Int"""
    k = 0
    for i in range(taille_file(F)):
        e = defiler(F)
        if e == elt:
            k = k + 1
        enfiler(F, e)
    return k
```

4. [1 point]

```
def verifier_contenu(F, nb_rouge, nb_vert, nb_jaune):
    """File -> Booléen"""
    return nb_elements(F, "rouge") <= nb_rouge
        and nb_elements(F, "vert") <= nb_vert
        and nb_elements(F, "jaune") <= nb_jaune
```