

[← Index des sujets 2021](#)

## 21-NSIJ1AN1 : Corrigé

Année : **2021**

Centre : **Amérique du nord**

Jour : **1**

Enoncé : [PDF](#)

### 1. Exercice 1

*bases de données relationnelles et langage SQL*

1. a. Cette requête affiche les champs `salle` et `marque_ordi` de la table `Ordinateur`.

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

- b. Cette requête affiche les champs `salle` et `marque_ordi` de la table `ordinateur` pour les enregistrement dont le champ `video` est à `true`

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

2.  [Requête SQL](#)

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

### Note

L'ordre croissant est l'ordre par défaut, on peut donc se passer du `ASC` dans la requête précédente.  
Pour mémoire l'ordre décroissant s'obtient avec `DESC`.

1. a. Une clé primaire est unique pour chaque enregistrement. Comme plusieurs ordinateurs peuvent être dans la même salle. Le champ `salle` n'est pas unique et ne peut donc pas servir de clé primaire.
- b. On souligne la clé primaire, on repère les clés étrangères en les faisant précédées du symbole `#`.
2. a.

#### Requête SQL

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

b.

#### Requête SQL

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

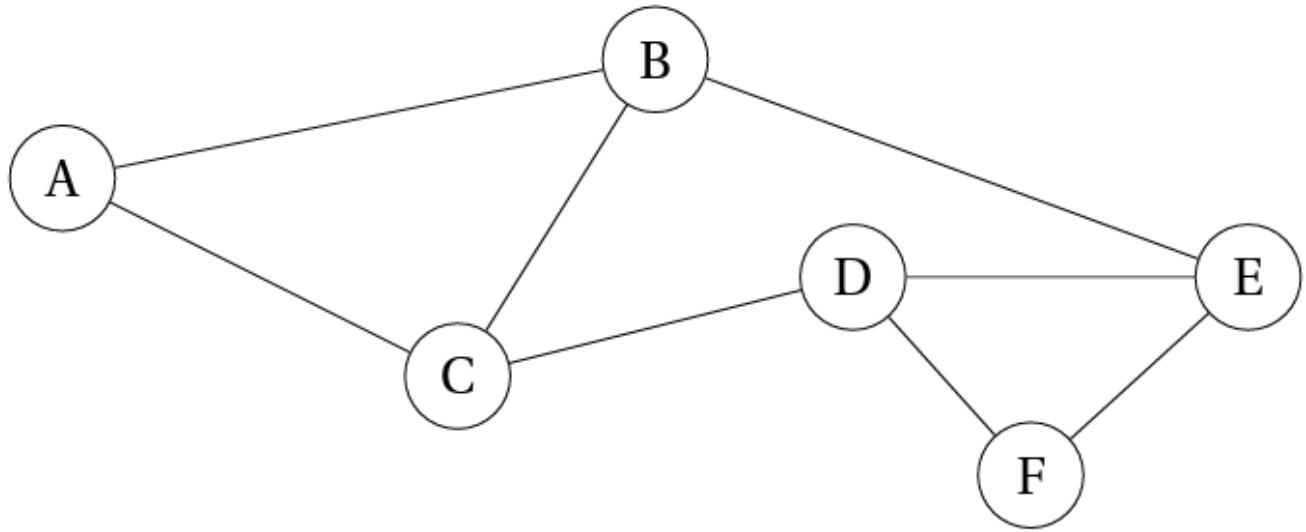
## 2. Exercice 2

*routage, processus et systèmes sur puces*

1. L'encombrement et la consommation sont plus faibles.
2. • D1 est mobilisé par le traitement de texte qui attend D2;
3. D2 est mobilisé par le SGBD qui attend D4;
4. D4 est mobilisé par la CAO qui attend D5;
5. D5 est mobilisé par le tableur qui attend D1;

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

6. A-B-E-F



7.

### 3. Exercice 3

*tableaux et programmation de base en Python*

1. a.

#### Script Python

```

def total_hors_reduction(tab):
    '''Calcul la somme des éléments de tab'''
    assert type(tab)==list, "L'argument doit être une liste"
    thr = 0
    for prix in tab:
        assert type(prix)==int or type(prix)==float,"Les prix doivent être de types numériques"
        thr += prix
    return thr
  
```

b.

#### Script Python

```

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.

#### Script Python

```
def prix_soldé(tab):
    if len(tab)>=5:
        return total_hors_reduction(tab)*0.5
    elif len(tab)==4:
        return total_hors_reduction(tab)*0.6
    elif len(tab)==3:
        return total_hors_reduction(tab)*0.7
    elif len(tab)==2:
        return total_hors_reduction(tab)*0.8
    else:
        return total_hors_reduction(tab)*0.9
```

3. a.

 Script Python

```
def minimum(tab):
    min_courant = tab[0]
    for elt in tab:
        if elt<min_courant:
            min_courant = elt
    return min_courant
```

b.

 Script Python

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

4. a.

 Script Python

```
tab = [35.0, 30.5, 20.0, 15.0, 10.5, 5.0, 6.0]
```

Le total des prix du panier est de  $35 + 30,5 + 20 + 15 + 10,5 + 6 + 5 = 122$ . Compte tenu de l'ordre des articles les articles coutant 20 € et 5 € seront offerts. Et donc le prix à payer sera 97 €.

b.

 Script Python

```
tab = [35.0, 30.5, 20.0, 15.0, 10.5, 6.0, 5.0]
```

Le prix total a payer est de 96 euros.

c. Il faut trier les objets par ordre décroissant de prix.

## 4. Exercice 4

*arbres binaires et algorithmes associés*

1. a.La racine est "Léa". L'ensemble des valeurs des feuilles est : "Marc", "Lea", "Claire", "Theo", "Marie", "Louis", "Anne" et "Kevin".  
 b. Le vainqueur est celui qui est à la racine.

 Script Python

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

c.

 Script Python

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

2. a. On utilise une fonction récursive.

 Script Python

```
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)
```

- b. Un joueur à gagné au moins un match si son nom apparaît plus d'une fois.

 Script Python

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

3. a. Cela ne fonctionne pas par le vainqueur apparaît à la racine sans effectuer de match.  
 b.

 Script Python

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

4. python

```
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))

```

## 5. Exercice 5

*notion de pile, de file et programmation de base en Python*

1. a. La file F est vide et P contient les éléments suivants : || |---| "rouge"|"vert"|"jaune"|"rouge"|"jaune"

b.

### Script Python

```

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. On utilise une pile intermédiaire pour retourner la pile. python

```

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

```

3. python

```

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
```

4. python

```
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
```