Éléments de correction
sujet 05

Exercice 1

1.
```
self.jour = jour
self.mois = mois
self.annee = annee
```
2.
1er mai 2000
3.
```
d = Date(19,6,2024)
```
4.
```
def get_annee(self):
    return self.annee
```
5.
```
def set_mois(self,mois):
    self.mois = mois
```
6.
```
if self.est_bissextile() :
    self.nb_jours_par_mois[1] = 29
```
7.
```
def est_bissextile(self):
    return (self.annee % 4 == 0 and self.annee % 100 != 0) or self.annee % 400 == 0
```
8.
```
>>> 79
```
9.
```
def nb_jours_restants(self):
    j = 365
    if self.est_bissextile():
        j = 366
    return j - self.nb_jours_passes()
```
10.
```
>>> d1.nb_jours_depuis(d2)
>>> 0
>>> d1.nb_jours_depuis(d3)
>>> -1
>>> d1.nb_jours_depuis(d4)
>>> -1
>>> d1.nb_jours_depuis(d5)
>>> 731
```
11.
```
def timestamp(self):
    d = Date(1,1,1970)
    return self.nb_jours_depuis(d) * 24 * 3600
```

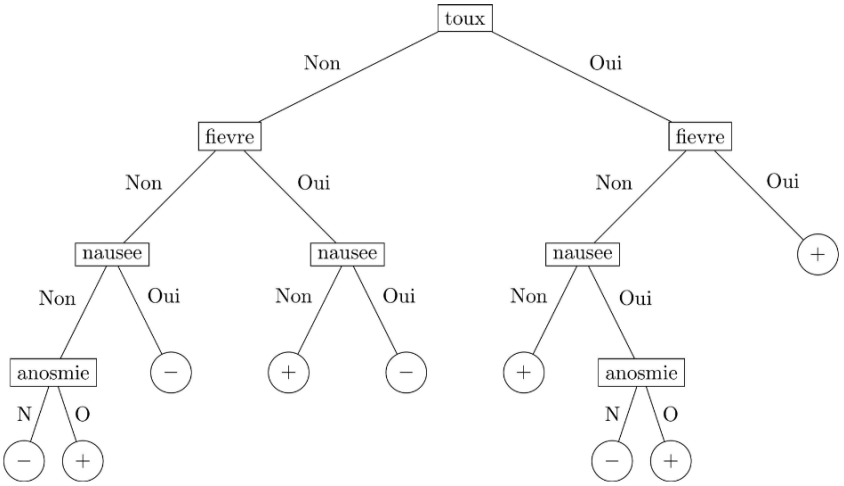
Exercice 2

1. un ordonnanceur
2. Prêt, élu et bloqué
3. File
4.
```
class Processus:
    def __init__(self, pid, priorite, temps_CPU):
        self.priorite = priorite
        self.PID = pid
        self.temps_utilisation = 0
        self.temps_CPU = temps_CPU
```
5.
```
Cycle 1: CPU=P1 liste_files=[[P3, P2], [], []]
Cycle 2: CPU=P2 liste_files=[[P3],[P1],[]]
Cycle 3: CPU=P3 liste_files=[[], [P2,P1],[]]
Cycle 4: CPU=P3 liste_files=[[], [P2,P1], []]
Cycle 5: CPU=P1 liste_files=[[], [P2], [P3]]
```
6.
Le processus 1000 va avoir une priorité qui va diminuer de 1 après chaque exécution. Comme il y a en permanence de nouveaux processus qui arrive avec une priorité de 0, au bout d'un moment le processus 1000 ne sera plus jamais élu.
7.
L'attente finit par faire remonter sa priorité, au bout d'un moment, il redeviendra prioritaire (au moins pour un cycle)
8.
```
def meilleur_priorite(liste_files):
    i = 0
    for f in liste_files:
        if len(f) > 0:
            return i
        i += 1
    return None
```
9.
```
def prioritaire(liste_files):
    index_prio = meilleur_priorite(liste_files)
    if index_prio == None:
        return None
    else :
        return liste_files[index_prio].pop(0)
```

10.
```
def gerer(p, liste_files):
    if p == None or p.temps_utilisation >= p.temps_CPU:
        return prioritaire(liste_files)
    else:
        p.temps_utilisation += 1
        if meilleur_priorite(liste_files) <= p.priorite:
            p.priorite += 1
            liste_files[p.priorite].append(p)
            return prioritaire(liste_files)
        else:
            p.priorite += 1
            return p
```

Exercice 3

1.
```
SELECT nom_patient, prenom
FROM Patient
WHERE age > 60
```
2.
```
UPDATE Symptome
SET toux = 'Non'
WHERE nom_patient = 'Heartman'
```
3.
```
SELECT COUNT(nom_patient)
FROM Diagnostic
JOIN Symptome ON Symptome.nom_patient = Diagnostic.nomp_patient
WHERE nom_maladie = 'Covid-19' AND toux = 'Oui'
```
4.
Il a fait deux erreurs dans sa requête : c'est Patient et pas Patients et le nom Douglas a déjà été utilisé (nom_patient est une clé primaire)
5.
Il faut choisir l'attribut numero_secu comme clé primaire
6.
On a (+)
7. Provoque une erreur (assertion error) si le noeud est une feuille
8. attribut : `self.valeur ;` méthode : `est_feuille`
9.
```
def applique(arbre, patient):
    if arbre.est_feuille():
        return arbre.diagnostic()
    else:
        if patient[arbre.symptome()]:
            return applique(arbre.droit, patient)
        else:
            return applique(arbre.gauche, patient)
```

10. 31
11. 

12.
```
def reduire(self):
    if self.est_feuille():
        return
    self.gauche.reduire()
    self.droit.reduire()
    if self.gauche.est_feuille() and self.droit.est_feuille() and
self.gauche.diagnostic() == self.droit.diagnostic():
        self.valeur = self.gauche.diagnostic()
        self.gauche = None
        self.droit = None
```
13.
```
def verifie(num_secu):
    n = num_secu // 100
    k = num_secu % 100
    return (k + n) % 97 == 0
```
14.
```
def cle(n):
    return 97 - (n % 97)
```