Éléments de correction
sujet 02

Exercice 1

1. 01100001
2. `[0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1]`
3.
```
def nb_occurences(tab,i):
    nb_occ = {}
    for j in range(3*i,3*(i+1)):
        x = tab[j]
        if x in nb_occ:
            nb_occ[x] += 1
        else :
            nb_occ[x] = 1
    return nb_occ
```

4.
```
def majorite(dict):
    cle_max = None
    valeur_max = -1
    for cle in dict.keys():
        if dict[cle] > valeur_max:
            valeur_max = dict[cle]
            cle_max = cle
    return cle_max
```

5.

1	0	1
1	1	0
0	1	1

6.
```
def erreur_colonne(mat):
    for i in range(len(mat)):
        s = 0
        for j in range(len(mat)):
            s += mat[j][i]
        if s % 2 == 1:
            return i
```

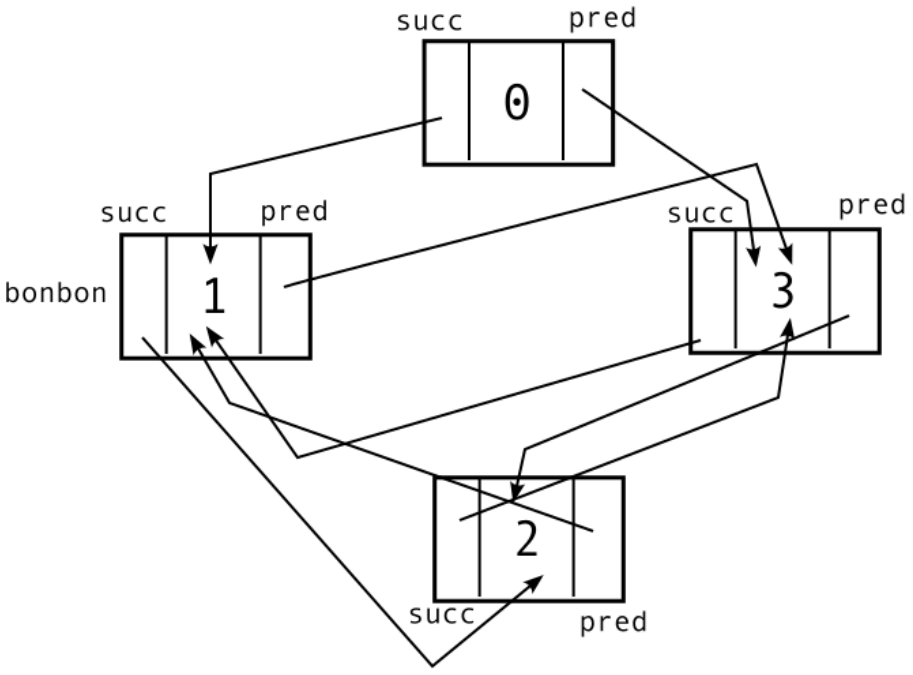
7.
1110000 diffère de 1010000 de 1 bit, le mot est donc 1000

8.
```
def corriger_erreur(code_recu):
    if code_recu in hamming_4_47:
        return code_recu
    else :
        code = [v for v in code_recu]
        for indice in range(7):
            conde[indice] = (code[indice] + 1) % 2
            if code in hamming_4_47:
                return code
            else :
                code[indice] = (code[indice] + 1) % 2
```
9.
2^7 = 128 feuilles (mais seulement 2^4 = 16 feuilles différentes)
10.
```
def decode(arbre, code, i):
    if i == len(code):
        return arbre.etiquette
    if code[i] == 0:
        return decode(arbre.gauche,code,i+1)
    if code[i] == 1:
        return decode(arbre.droite,code,i+1)
```

Exercice 2

1. 0 - 3 - 6 - 2 - 7 - 5 - 1 reste 4
2. Le nom “true” n’est pas reconnu par le système. Il faudrait écrire “True” à la place.
3.
```
def dernier(n):
    collier = [True]*n
    indice = 0
    collier[indice] = False
    for etape in range(n-1):
        nb_bonbons_vus = 0
        while nb_bonbons_vus < 3:
            indice += 1
            if indice == n:
                indice = 0
            if collier[indice] :
                nb_bonbons_vus += 1
        collier[indice] = False
    return indice
```

4. La file est de type FIFO
5. affiche 3, 4, 1, 2
6.
```
def dernier_file(n):
    f = File()
    for i in range(n):
        f.enfile(i)
    while n > 1:
        f.defile()
        f.enfile(f.defile())
        f.enfile(f.defile())
        n -= 1
    return f.defile()
```
7. Ce sont des attributs
8. a = 1 ; b = 2
9.
```
def creer_collier(n):
    premier = Bonbon(0)
    actuel = premier
    for i in range(1,n):
        nouveau = Bonbon(i)
        actuel.succ = nouveau
        nouveau.pred = actuel
        actuel = nouveau
    actuel.succ = premier
    premier.pred = actuel
    return premier
```
10.



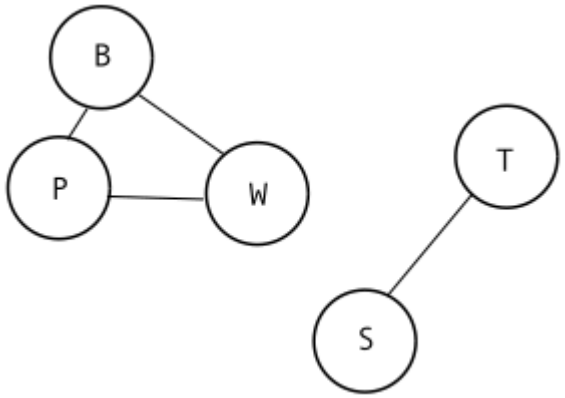
Le nouveau bonbon est le 1. 0 pointe encore vers 1 et vers 3, mais plus aucun bonbon ne pointe vers 0 : 0 ne fait donc plus partie du collier (il a été mangé)

11. Proposition C
12.
```
def dernier_chaine(n):
    bonbon = creer_collier(n)
    while bonbon.valeur != bonbon.succ.valeur:
        bonbon.pred.succ = bonbon.succ
        bonbon.succ.pred = bonbon.pred
        bonbon = bonbon.succ.succ.succ
    return bonbon.valeur
```

Exercice 3

1. On retrouve plusieurs fois la même valeur pour l'atrribut id_vol, or, la clé primaire doit être unique, id_vol ne peut donc pas être une clé primaire
2. Le couple id_vol et id_passager peut être une clé primaire, car un passager ne peut pas être plusieurs fois sur le même vol.
3. Une clé étrangère permet d'établir un lien (jointure) entre deux tables.
4. AI0015, AI0258, AI0292
5.
```
SELECT ville
FROM aeroport
JOIN vol ON id_aeroport = aeroport_arr
WHERE aeroport_dep = 'CDG'
```
6.
```
UPDATE passager
SET d_totale = 16
WHERE id_passager = 5
```
7.
id_vol est la clé primaire. AI0256 est déjà utilisé dans la table vol, ce qui va entrainer une erreur. Il est donc nécessaire de choisir une autre valeur qui n'a pas encore été utilisée.
8. valeur expression : 10
9.
```
def vol_direct(graphe, ville1, ville2):
    return ville2 in graphe[ville1]
```
10.
```
def liste_villes_proches(graphe,ville,d_max):
    t = []
    d_ville = graphe[ville]
    for k,v in d_ville.items():
        if v <= d_max:
            t.append(k)
    return t
```

11.



12. A
13. à la ligne 13 la fonction parcours s'appelle elle-même, elle est donc récursive.
14.
```
visitees1 = ['W', 'P', 'T', 'B', 'S']
visitees2 = ['W', 'P', 'B']
```
15.
proposition C
16.
```
def est_connexe(graphe):
    depart = ville_arbitraire(graphe)
    visitees = []
    parcours(graphe, visitees, depart)
    return len(visitees) == len(graphe)
```
17.
```
['W', 'P', 'T', 'B'] 25
['W', 'P', 'S', 'T', 'B'] 40
```
18.
Permet d'afficher tous les chemins entre 'ville' et 'arrivee' sous forme de liste de sommets, ainsi que la distance totale parcourue pour chaque chemin.