Épreuves écrites de NSI
Correction du sujet
Centres étrangers 2021
Journée 2

Exercice 2

1. On peut écrire le code suivant :

```
def mur(laby, i, j) :
    return laby[i][j] == 1
```

ou bien :

```
def mur(laby, i, j) :
    if laby[i][j] == 1 :
        return True
    else :
        return False
```

2. a. Les coordonnée des deux cases sont données respectivement par `(l1 ; c1)` et `(l2 ; c2)`. L'expression `(l1 - l2)**2 + (c1 - l2)**2` permet donc de calculer le carré de la distance entre ces deux cases. Elle correspond à un entier positif puisque `l1, l2, c1` et `c2` sont des entiers.

Donc, à la dernière ligne de la fonction, la variable `d` est égale à 1 si et seulement si les cases sont adjacentes. En effet, si `d` est égale à 0 alors les cases coïncident et si d est supérieure ou égale à 2 alors les cases sont distinctes et non adjacentes.

2. b. On peut écrire le code suivant :

```
def adjacentes(cases) :
    lg = len(cases)
    for i in range(lg - 1):
        if not voisines(cases[i], cases[i+1]):
            return False
    return True
```

3. La variable `i` est initialisée à 0 et elle est incrémentée de 1 à chaque passage dans la boucle. Il est donc certain qu'elle dépassera la valeur `len(cases)` en un nombre fini d'étapes. La condition `i < len(cases) and possible` renverra alors la valeur `False` quelle que soit la valeur de la variable `possible` et on sortira donc de la boucle `while`.

4. On peut écrire le code suivant :

```
def echappe(cases, laby) :
    lg1 = len(cases)
    lg2 = len(laby)
    return teste(cases, laby) and cases[0] == (0,0) and
cases[lg1-1] == (lg2-1,lg2-1)
```

ou bien :

```
def echappe(cases, laby) :
    lg1 = len(cases)
    lg2 = len(laby)
```

```
    if not teste(cases, laby) :
        return False
    if  cases[0] != (0,0) :
        return False
    if cases[lg1-1] != (lg2-1,lg2-1):
        return False
    return True
```

Exercice 3 :

1. Décomposons 89 en somme de puissances de 2. On a :

$$89 = 64 + 25 = 64 + 16 + 9 = 64 + 16 + 8 + 1$$

Donc

$$89 = 2^6 + 2^4 + 2^3 + 2^0$$
$$= 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Ainsi l'écriture binaire du nombre 89 est 01011001.

2. On effectue l'opération suivante :

$$\begin{array}{r} 1100\ 1110 \\ \oplus\ 0110\ 1011 \\ \hline \mathbf{1010\ 0101} \end{array}$$

3. On peut écrire le code suivant :

```
def xor_crypt(message, cle) :
    """On suppose que les chaines message et cle sont de
même longueur"""
    tab_res = []
    lg = len(message)
    for i in range(lg) :
        c1 = message[i]
        n1 = ord(c1)
        c2 = cle[i]
        n2 = ord(c2)
        n3 = xor(n1, n2)
        tab_res.append(n3)
    return tab_res
```

4. On peut écrire le code suivant :

```
def generer_cle(mot, n) :
    cle = ""
    lg = len(mot)
    q = n//lg
    r = n % lg
    for i in range(q):
        cle = cle + mot
    for i in range(r):
        cle = cle + mot[i]
    return cle
```

ou bien en utilisant certains opérateurs utiles en Python :

```
def generer_cle(mot, n) :
    lg = len(mot)
    q = n//lg
    r = n%lg
    cle = q*mot + mot[:r]
    return cle
```

5. On a la table de vérité suivante :

E_1	E_2	$E_1 \oplus E_2$	$(E_1 \oplus E_2) \oplus E_2$
0	0	0	**0**
0	1	1	**0**
1	0	1	**1**
1	1	0	**1**

On constate que dans tous les cas la valeur de $(E_1 \oplus E_2) \oplus E_2$ est identique à celle de E_1. Pour déchiffrer le message codé obtenu comme résultat de l'appel de fonction `xor_crypt(message, cle)`, il suffit donc de faire à nouveau l'appel de fonction `xor_crypt(..., cle)`.

Ainsi, supposons qu'Alice a placé le message à chiffrer dans une variable `monMessage` puis a exécuté l'instruction suivante :

```
mes_chiffre = xor_crypt(message, cle)
```

puis qu'elle a envoyé à Bob la chaîne de caractère contenue dans la variable `mes_chiffre`. Si Bob connaît la variable `cle` utilisée par Alice, il peut déchiffrer le message d'Alice en exécutant les deux instructions suivantes :

```
mes_clair = xor_crypt(mes_chiffre, cle).
print(mes_clair)
```