Correction

NSI - 2021 Étranger Jour 2 (21-NSIJ2G11)

Exercice 1 - Mélange

1. La liste mélangée obtenue est `['10', 'A', '9', 'R', '8', 'D', '7', 'V']`.

2.

```
def liste_vers_pile(L):
    '''Prend en paramètre une liste et renvoie une pile.'''
    N = len(L)
    p_temp = Pile()
    for i in range(N):
        p_temp.empiler(L[i])
    return p_temp
```

3. L'affichage sera un message d'erreur indiquant que la fonction range n'accepte pas de flottant.
`'float' object cannot be interpreted as an integer`
Il faudrait remplacer la division simple par le quotient de la division euclidienne N**//**2 en argument de la fonction `range`.

Si on passe cette erreur, on obtient un autre message d'erreur indiquant que l'objet `p_gauche` ne possède pas de méthode `empile`. La méthode définie est `empile`**r**.
`'Pile' object has no attribute 'empile'`

En considérant qu'il s'agit d'erreurs du sujet, l'affichage obtenu serait :

3
2
1
6
5
4

4. a.

Pile Gauche	Pile Droite	Liste	Explications
3 2 1	6 5 4	[]	La liste initiale est vide.
2 1	6 5 4	[3]	On dépile Pile Gauche et on insère dans la liste.
2 1	5 4	[3, 6]	On dépile Pile Droite et on insère dans la liste.
1	5 4	[3, 6, 2]	On dépile Pile Gauche et on insère dans la liste.
1	4	[3, 6, 2, 5]	On dépile Pile Droite et on insère dans la liste.
	4	[3, 6, 2, 5, 1]	On dépile Pile Gauche et on insère dans la liste.
		[3, 6, 2, 5, 1, 4]	On dépile Pile Droite et on insère dans la liste.

Correction

NSI - 2021 Étranger Jour 2 (21-NSIJ2G11)

4. b.

```
def fusion(p1, p2):
    """Fusion de deux piles possédant le même nombre d'éléments."""
    liste = []
    while not p1.est_vide():
        liste.append(p1.depiler())
        liste.append(p2.depiler())
    return liste
```

5.

```
def affichage_pile(p):
    p_temp = p.copier()
    if p_temp.est_vide():
        print('____')
    else:
        elt = p_temp.depiler()
        print('| ', elt, ' |')   # !! Une parenthèse en trop dans le sujet
        affichage_pile(p_temp)
```

Correction

NSI - 2021 Étranger Jour 2 (21-NSIJ2G11)

Exercice 2 - Labyrinthe

1.

Solution 1

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

Solution 2

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

2. a. Pour que deux cases soit adjacente, il faut que la distance qui les sépare vaille 1. La distance est ici calculée selon Pythagore $c = \sqrt{a^2 + b^2}$. La racine carré est omise car $\sqrt{1} = 1$ est le seul cas qui nous intéresse.

2. b.

```
def adjacentes(liste):
    """ Vérifie que la liste est une suite de cases voisines."""
    if len(liste) <= 1:
        return True
    for i in range(1, len(liste)):
        if not voisine(liste[i-1], liste[i]):
            return False
    return True
```

3. La preuve de terminaison de la boucle est apportée par l'incrémentation de l'indice `i` à chaque itération de la boucle, qui conduira à rendre faux la condition de continuation de la boucle (`i < len(cases) and possible`). En effet `i` dépassera obligatoirement `len(cases)` à un moment donné.

4.

```
def echappe(cases, laby):
    if not cases[0] == (0, 0):
        return False        # La première case est l'entrée.
    if not cases[-1] == (len(laby-1), len(laby-1)):
        return False        # La première case est l'entrée.
    if not adjacentes(cases):
        return False        # Les cases sont adjacentes.
    if not teste(cases, laby):
        return False        # Le chemin ne percute pas de mur.
    return True
```

Correction

NSI - 2021 Étranger Jour 2 (21-NSIJ2G11)

Exercice 3 - Codage XOR

1. Conversion de $89_{(10)}$ en binaire :

$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
89 – 64 = 25	25	25 – 16 = 9	9 – 8 = 1	1	1	1 – 1 = 0
1	0	1	1	0	0	1

$89_{(10)} = 01011001_{(2)}$ sur 8 bits.

2.

	1	1	0	0	1	1	1	0
$\oplus$	0	1	1	0	1	0	1	1
=	1	0	1	0	0	1	0	1

3.

```
def xor_crypt(message, cle):
    """ Renvoie la liste des entiers correspondant au message codé.
        Le message et la clé ont la même longueur."""
    code = []
    for i in range(len(message)):
        code.append(xor(ord(message[i]), ord(cle[i])))
    return code
```

Remarque : On parle plutôt de « chiffrage » que de « cryptage ». Seule la notion de décryptage a un sens puisque qu'elle consiste à décrypter un code sans en connaître la clé. On parle de « chiffrage » puisque la clé est forcément connue au moment du chiffrage, on ne peut pas crypter sans connaître la clé de chiffrement.

4.

```
def generer_cle(mot, n):
    """ Répète le mot jusqu'à obtenir une chaîne de n caractère."""
    cle = ""
    i = 0
    while len(cle) < n:
        cle += mot[i % len(mot)] # la clé peut être plus longue que n
        i += 1
    return cle
```

5.

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

Pour décoder un message codé par cette méthode, il suffit d'appliquer la même opération que celle réalisée lors du codage, c'est-à-dire l'opération XOR bit à bit pour chaque lettre du message codé et la clé.

Correction

NSI - 2021 Étranger Jour 2 (21-NSIJ2G11)

Exercice 4 - Gestion d'un club de handball

1. a. L'attribut `nom` de la table `licencies` ne peut pas servir de clé primaire à cause des homonymies. Une clé primaire doit permettre d'identifier de façon unique un enregistrement d'une table.

1. b. L'attribut pouvant servir de clé primaire est `id_licencie`.

2. a. Cette requête renvoie le prénom et le nom de tous les licenciés appartenant à l'équipe des -12 ans.

2. b. Elle renvoie toutes les valeurs (`id_licencie`, `prenom`, `nom`, `annee_naissance`, `equipe`) de tous les licenciés appartenant à l'équipe des -12 ans.

2. c.

```
SELECT date FROM matchs WHERE lieu = 'Domicile' AND equipe = 'Vétérans';
```

3.

```
INSERT INTO licencies(id_licencie, prenom, nom, annee_naissance, equipe) VALUES (287, 'Jean', 'Lavenu', '2001', 'Hommes 2');
```

ou, puisque toutes les valeurs de la table sont présentes, la syntaxe raccourcie :

```
INSERT INTO licencies VALUES (287, 'Jean', 'Lavenu', 2001, 'Hommes 2');
```

4.

```
UPDATE licencies SET equipe = 'Vétérans' WHERE prenom = 'Joseph' AND nom = 'Cuviller';
```

5.

```
SELECT nom FROM licencies
JOIN Matchs ON licencies.equipe = matchs.equipe
WHERE matchs.adversaire = 'LSC' AND matchs.date = '2021-06-19';
```

Correction

NSI - 2021 Étranger Jour 2 (21-NSIJ2G11)

Exercice 5 - Bandeau à LED

1. a. L'instruction `Obj_bandeau.get_pixel_rgb(1)` retourne le tuple `(0, 0, 255)`.

1. b. L'instruction `Adafruit_WS2801.RGB_to_color(0,0,255)` retourne l'entier `16711680`.

1. c.

```
coul = Obj_bandeau.get_pixel_rgb(0)
```

Récupère le tuple `(255, 0, 0)` associé au 1er pixel et l'affecte à la variable `coul`.

```
print(Adafruit_WS2801.RGB_to_color(coul[0],coul[1],coul[2]))
```

Affiche l'entier correspondant `255`.

2. a.

16711680 → Bleu pour les 5 premières LED.

16777245 → Blanc pour les 5 suivantes

255 → Rouge pour les 5 dernières.

Bleu	Bleu	Bleu	Bleu	Bleu	Blanc	Blanc	Blanc	Blanc	Blanc	Rouge	Rouge	Rouge	Rouge	Rouge

2. b.

32768 → Vert pour les LED dont l'indice est un multiple de 3.

65535 → Jaune pour les autres.

Vert	Jaune	Jaune	Vert	Jaune	Jaune	Vert	Jaune	Jaune	Vert	Jaune	Jaune	Vert	Jaune	Jaune

3. a.

```
""" Initialise un bandeau avec Pixel_COUNT LEDs."""
```

3. b.

```
# Met en bleu les pixels d'indice 6 et 7.
```