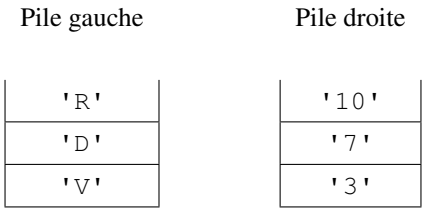
# Centres étrangers - juin 2021 - sujet 2

**Exercice 1 (Piles - 4 points)**
On cherche à obtenir un mélange d'une liste comportant un nombre pair d'éléments. Dans cet exercice, on notera `N` le nombre d'éléments de la liste à mélanger.

La méthode de mélange utilisée dans cette partie est inspirée d'un mélange de jeuxde cartes :

- ⋆ On sépare la liste en deux piles :
  - à gauche, la première pile contient les `N/2` premiers éléments de la liste ;
  - à droite, la deuxième pile contient les `N/2` derniers éléments de la liste.
- ⋆ On crée une liste vide.
- ⋆ On prend alors le sommet de la pile de gauche et on le met en début de liste, puis le sommet de la pile de droite que l'on ajoute à la liste et ainsi de suite jusqu'à ce que les piles soient vides.

Par exemple, si on applique cette méthode de mélange à la liste `['V','D','R','3','7','10']`, on obtient pour le partage de la liste en deux piles :




La nouvelle liste à la fin du mélange sera donc `['R','10','D','7','V','3']`.

1. Que devient la liste `['7','8','9','10','V','D','R','A']` si on lui applique cette méthode de mélange ?

On considère que l'on dispose de la structure de données de type `pile`, munie des seules instructions suivantes :

- `p = Pile()` : crée une pile vide nommée `p`
- `p.est_vide()` : renvoie Vrai si la liste est vide, Faux sinon
- `p.empiler(e)` : ajoute l'élément `e` dans la pile `p`
- `e = p.depiler()` : retire le dernier élément ajouté dans la pile `p`, le renvoie et l'affecte à la variable `e`
- `p2 = p.copier()` : renvoie une copie de la pile `p` sans modifier la pile `p` et l'affecte à une nouvelle pile `p2`

2. Recopier et compléter le code de la fonction suivante qui transforme une liste en pile.

```
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. On considère la fonction `partage` suivante qui partage une liste en deux piles. Lors de sa mise au point et pour aider au débuggage, des appels à la fonction `affichage_pile` ont été insérés. La fonction `affichage_pile(p)` affiche la pile `p` à l'écran verticalement sous la forme suivante :

| dernier élément empilé |
|---|
| ..... |
| ..... |
| premier élément empilé |

Code de la fonction `partage` :

```
cle = input("saisir la clé de chiffrement : ")
cle = int(cle)
c = CodeCesar(cle)
txt = input("saisir le texte à chiffrer : ")
print("le message chiffré est : "+c.cryptage(txt))
```

Quels affichages obtient-on à l'écran lors de l'exécution de l'instruction : `partage([1,2,3,4,5,6])` ?

4. (a) Dans un cas général et en vous appuyant sur une séquence de schémas, expliquer en quelques lignes comment fusionner deux piles `p_gauche` et `p_droite` pour former une liste `L` en alternant un à un les éléments de la pile `p_gauche` et de la pile `p_droite`.

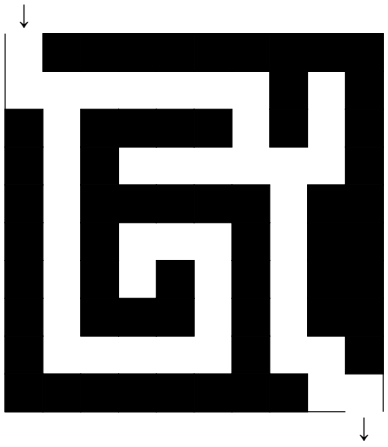
   (b) Ecrire une fonction `fusion` prenant en paramètre deux piles `p1` et `p2` et renvoyant une liste construite à partir de `p1` et `p2`.

5. Compléter la dernière ligne du code de la fonction `affichage_pile` pour qu'elle fonctionne de manière récursive.

```
def affichage_pile(p):
    p_temp = p.copier()
    if p_temp.est_vide():
        print('_____')
    else:
        elt = p_temp.depiler()
        print('| ', elt, ' |')
        affichage_pile(p_temp)
```

**Exercice 2 (Tableaux - 4 points)**
L'objectif de cet exercice est de mettre en place une modélisation d'un jeu de labyrinthe en langage `Python`.

On décide de représenter un labyrinthe par un tableau carré de taille $n$, dans lequel les cases seront des 0 si l'on peut s'y déplacer et des 1 s'il s'agit d'un mur. Voici un exemple de représentation d'un labyrinthe :



```
laby=[[0,1,1,1,1,1,1,1,1,1],
      [0,0,0,0,0,0,0,1,0,1],
      [1,0,1,1,1,1,0,1,0,1],
      [1,0,1,0,0,0,0,0,0,1],
      [1,0,1,1,1,1,1,0,1,1],
      [1,0,1,0,0,0,1,0,1,1],
      [1,0,1,0,1,0,1,0,1,1],
      [1,0,1,1,1,0,1,0,1,1],
      [1,0,0,0,0,0,1,0,0,1],
      [1,1,1,1,1,1,1,1,0,0]]
```

L'entrée du labyrinthe se situe à la première case du tableau (celle en haut à gauche) et la sortie du labyrinthe se trouve à la dernière case (celle en bas à droite).

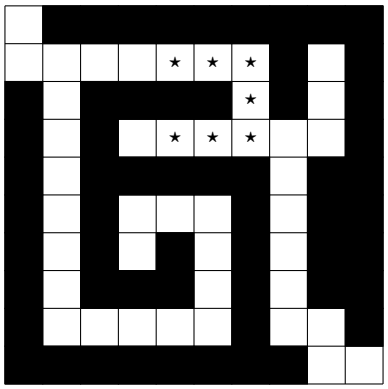
1. Proposer, en langage `Python`, une fonction `mur`, prenant en paramètre un tableau représentant un labyrinthe et deux entiers `i` et `j` compris entre `0` et `n-1` et qui renvoie un booléen indiquant la présence ou non d'un mur. Par exemple :

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

Un parcours dans le labyrinthe va être représenté par une liste de cases. Il s'agit de couples `(i,j)` où `i` et `j` correspondent respectivement aux numéros de ligne et de colonne des cases successivement visitées au long du parcours. Ainsi, la liste

`[(1,4),(1,5),(1,6),(2,6),(3,6),(3,5),(3,4)]`

correspond au parcours repéré par des étoiles $\star$ dans le schéma ci-dessous :



En revanche, la liste `[(0,0),(1,0),(1,1),(5,1),(6,1)]` ne peut pas correspondre au parcours d'un labyrinthe car toutes les cases parcourues successivement ne sont pas adjacentes.

2. On considère la fonction voisine ci-dessous, écrite en langage `Python`, qui prend en paramètres deux cases données sous forme de couple.

```
def proposition(choix):
    for v in flotte:
        if flotte[v]["type"] == choix and flotte[v]["etat"] == 1:
            return flotte[v]["station"]
```

   (a) Après avoir remarqué que les quantités `l1-l2` et `c1-c2` sont des entiers, expliquer pourquoi la fonction `voisine` indique si deux cases données sous forme de tuples `(l,c)` sont adjacentes.

(b) En déduire une fonction `adjacentes` qui reçoit une liste de cases et renvoie un booléen indiquant si la liste des cases forme une chaîne de cases adjacentes.

Un chemin sera qualifié de compatible avec le labyrinthe lorsqu'il s'agit d'une succession de cases adjacentes accessibles (non murées). On donne la fonction `teste` qui prend en paramètre un chemin `cases` et un labyrinthe `laby` et qui indique si ce chemin `cases` est un chemin possible compatible avec le labyrinthe `laby` :

```
def teste(cases, laby) :
    if not adjacentes(cases):
        return False
    possible = True
    i = 0
    while i < len(cases) and possible:
        if mur(laby, cases[i][0], cases[i][1]):
            possible = False
        i = i + 1
    return possible
```

3. Justifier que la boucle de la fonction précédente se termine.
4. En déduire une fonction `echappe` prenant en paramètre un chemin `cases` et un labyritnhe `laby` et qui indique par un booléen si le chemin `cases` permet d'aller de l'entrée à la sortie du labyrinthe `laby`.

**Exercice 3 (Booléens et caractères - 4 points)**
L'objectif de l'exercice est d'étudier une méthode de cryptage d'une chaîne de caractères à l'aide du codage ASCII et de la fonction logique `XOR`.

1. Le nombre 65, donné ici en écriture décimale, s'écrit 0100 0001 en notation binaire. En détaillant la méthode utilisée, donner l'écriture binaire du nombre 89.
2. La fonction logique `OU EXCLUSIF`, appelée `XOR` et représentée par le symbole $\oplus$, fournit une sortie égale à 1 si l'une ou l'autre des deux entrées vaut 1 mais pas les deux. On donne ci-dessous la table de vérité de la fonction `XOR` :

| A | B | A $\oplus$ B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Si on applique cette fonction à un nombre codé en binaire, elle opère bit à bit :

$$\begin{array}{rr} & 1100 \\ \oplus & 1010 \\ \hline = & 0110 \end{array}$$

Poser et calculer l'opération : 1100 1110 $\oplus$ 0110 1011

3. On donne, ci-dessous, un extrait de la table ASCII qui permet d'encoder les caractères de A à Z. On peut alors considérer l'opération `XOR` entre deux caractères en effectuant le `XOR` entre les codes ASCII des deux caractères. Par exemple : `'F' XOR 'S'` sera le résultat de 0100 0110 $\oplus$ 0101 0011.

| **Code ASCII Décimal** | **Code ASCII Binaire** | **Caractère** |
|---|---|---|
| 65 | 0100 0001 | A |
| 66 | 0100 0010 | B |
| 67 | 0100 0011 | C |
| 68 | 0100 0100 | D |
| 69 | 0100 0101 | E |
| 70 | 0100 0110 | F |
| 71 | 0100 0111 | G |
| 72 | 0100 1000 | H |
| 73 | 0100 1001 | I |
| 74 | 0100 1010 | J |
| 75 | 0100 1011 | K |
| 76 | 0100 1100 | L |
| 77 | 0100 1101 | M |

| **Code ASCII Décimal** | **Code ASCII Binaire** | **Caractère** |
|---|---|---|
| 78 | 0100 1110 | N |
| 79 | 0100 1111 | O |
| 80 | 0101 0000 | P |
| 81 | 0101 0001 | Q |
| 82 | 0101 0010 | R |
| 83 | 0101 0011 | S |
| 84 | 0101 0100 | T |
| 85 | 0101 0101 | U |
| 86 | 0101 0110 | V |
| 87 | 0101 0111 | W |
| 88 | 0101 1000 | X |
| 89 | 0101 1001 | Y |
| 90 | 0101 1010 | Z |

On souhaite mettre au point une méthode de cryptage à l'aide de la fonction `XOR`. Pour cela, on dispose d'un message à crypter et d'une clé de cryptage de même longueur que ce message. Le message et la clé sont composés uniquement des caractères du tableau ci-dessus et on applique la fonction `XOR` caractère par caractère entre les lettres du message à crypter et les lettres de la clé de cryptage. Par exemple, voici le cryptage du mot `ALPHA` à l'aide de la clé `YAKYA` :

| | | | | | |
|---|---|---|---|---|---|
| Message à crypter | A | L | P | H | A |
| Clé de cryptage | Y | A | K | Y | A |
| | ↓ | ↓ | ↓ | ↓ | ↓ |
| Message crypté | 'A' XOR 'Y' | 'L' XOR 'A' | 'P' XOR 'K' | 'H' XOR 'Y' | 'A' XOR 'A' |

Ecrire une fonction `xor_crypt` prenant en paramètres deux chaînes de caractères `message` et `cle` et renvoyant la liste des entiers correspondant au message `message` crypté à l'aide de la clé `cle`.

Aide :

- On pourra utiliser la fonction native `ord` du langage `Python` qui prend en paramètre un caractère `c` et qui renvoie un nombre représentant le code ASCII décimal du caractère `c`.
- On considère également que l'on dispose d'une fonction écrite `xor` qui prend en paramètre deux nombres `n1` et `n2` et qui renvoie le résultat de `n1` $\oplus$ `n2`.

4. On souhaite maintenant générer une clé de la taille du message à partir d'un mot quelconque. On considère que le mot choisi est plus court que le message, il faut donc le reproduire un certain nombre de fois pour créer une clé de la même longueur que le message. Par exemple, si le mot choisi est `YAK`, alors pour crypter le message `ALPHABET`, la clé sera `YAKYAKYA`.
   Créer une fonction `generer_cle` prenant en paramètre une chaîne de caractères `mot` et un entier `n`, et renvoyant la clé de longueur `n` à partir de la chaîne de caractères `mot`.
5. Recopier et compléter la table de vérité de `(A` $\oplus$ `B)` $\oplus$ `B`.

| A | B | A $\oplus$ B | (A $\oplus$ B) $\oplus$ B |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | |
| 1 | 1 | 1 | |

A l'aide de ce résultat, proposer une démarche pour décrypter un message crypté.

**Exercice 4 (SQL - 4 points)**
*Vous trouverez, en annexe 1, des rappels sur le langage SQL.*

Un club de handball souhaite regrouper efficacement toutes ses informations. Il utilise pour cela des bases de données relationnelles afin d'avoir accès aux informations classiques sur les licenciés du club ainsi que sur les matchs du championnat. Le langage SQL a été retenu.

On suppose dans l'exercice que tous les joueurs d'une équipe jouent à chaque match de l'équipe.

La structure de la base de données est composée des deux tables (ou relations) suivantes :

| Table `licencies` | |
|---|---|
| **Attributs** | **Types** |
| `id_licencie` | INT |
| `prenom` | VARCHAR |
| `nom` | VARCHAR |
| `annee_naissance` | INT |
| `equipe` | VARCHAR |

| Table `matchs` | |
|---|---|
| **Attributs** | **Types** |
| `id_match` | INT |
| `equipe` | VARCHAR |
| `adversaire` | VARCHAR |
| `lieu` | VARCHAR |
| `date` | DATE |

Ci-dessous un exemple de ce que l'on peut trouver dans la base de données :

- Exemple **non exhaustif** d'entrées de la table `licencies` :

| id_licencie | prenom | nom | annee_naissance | equipe |
|---|---|---|---|---|
| 63 | Jean-Pierre | Masclef | 1965 | Vétérans |
| 102 | Eva | Cujon | 1992 | Femmes 1 |
| 125 | Emile | Alinio | 2000 | Hommes 2 |
| 247 | Ulysse | Trentain | 2008 | -12 ans |

- Exemple **non exhaustif** d'entrées de la table `matchs` :

| id_match | equipe | adversaire | lieu | date |
|---|---|---|---|---|
| 746 | -16 ans | PHC | Domicile | 2021-06-19 |
| 780 | Vétérans | PHC | Exterieur | 2021-06-26 |
| 936 | Hommes 3 | LSC | Exterieur | 2021-06-20 |
| 1032 | -19 ans | LOH | Exterieur | 2021-05-22 |
| 1485 | Femmes 2 | CHM | Domicile | 2021-05-02 |
| 1512 | Vétérans | ATC | Domicile | 2021-04-12 |

1. (a) L'attribut nom de la table licencies pourrait-il servir de clé primaire ? Justifier.
   (b) Citer un autre attribut de cette table qui pourrait servir de clé primaire.
2. (a) Expliquer ce que renvoie la requête SQL suivante :

```
SELECT prenom,nom FROM licencies WHERE equipe = "-12ans"
```

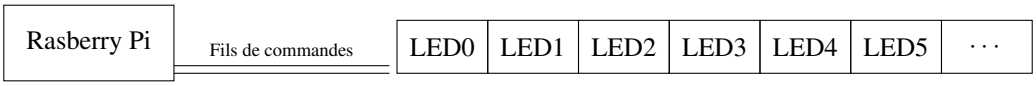
   (b) Que renvoie la requête précédente si `prenom,nom` est remplacé par une étoile $\star$ ?
   (c) Ecrire la requête qui permet l'affichage des dates de tous les matchs joués à domicile de l'équipe Vétérans.
3. Ecrire la requête qui permet d'inscrire dans la table `licencies`, Jean Lavenu né en 2001 de l'équipe Hommes 2 et qui aura comme numéro de licence 287 dans ce club.
4. On souhaite mettre à jour les données de la table `licencies` du joueur Joseph Cuviller, déjà inscrit. Il était en équipe Hommes 2 et il est maintenant en équipe Vétérans. Afin de modifier la table dans ce sens, proposer la requête adéquate.
5. Pour obtenir le nom de tous les licenciés qui jouent contre le LSC le 19 juin 2021, recopier et compléter la requête suivante :

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

**Exercice 5 (POO - 4 points)**

L'objectif de cet exercice est de commander un bandeau de diodes électroluminescentes (LED) à l'aide d'un nano-ordinateur Raspberry Pi en langage `Python`. Chacune des LEDs du bandeau pourra être commandée individuellement pour l'allumer avec une couleur choisie. Une LED est considérée comme éteinte lorsque la couleur est noire (pas de lumière), allumée pour toutes les autres couleurs.

Pour cela, nous allons utiliser un bandeau comportant au maximum 32 LEDs branché sur un nano-ordinateur Raspberry Pi sur lequel est installé le module (librairie) `Adafruit_WS2801`, à importer dans le script `Python`.




**Après avoir lu la documentation donnée en annexe 2**, répondre aux questions suivantes :

1. *Compréhension des méthodes avec un bandeau de huit LEDs.*
   Un script a été exécuté au préalable et le bandeau de LEDs est dans l'état suivant : toutes les LEDs sont éteintes sauf trois, la LED « LED0 » qui est rouge, la LED « LED1 » qui est bleue et la LED « LED5 »qui est verte.

   | | | | | | | | |
   |---|---|---|---|---|---|---|---|
   | rouge | bleu | | | | vert | | |

   L'objet bandeau est créé au début de notre script dans la variable `Obj_bandeau`

   (a) Que va renvoyer l'instruction : `Obj_bandeau.get_pixel_rgb(1)` ?

   (b) Que va renvoyer l'instruction `Adafruit_WS2801.RGB_to_color(0,0,255)` ?

   (c) On exécute les instructions suivantes :

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

   Expliquer brièvement le rôle de chacune des deux lignes.

2. On dispose désormais d'un bandeau de quinze LEDs.
   Pour chacun des deux scripts suivants écrits en langage `Python`, recopier et compléter les tableaux correspondants au bandeau dans son état final après exécution des différents scripts.

   Exemple : pour un bandeau avec les trois premières LEDs rouges, les deux suivantes blanches et les quatre dernières bleues, on obtient le tableau donné ci-dessous :

   | | | | | | | | | | | | | | | |
   |---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
   | rouge | rouge | rouge | blanc | blanc | | | | | | | bleu | bleu | bleu | bleu |

   (a) En vous inspirant de l'exemple ci-dessus, dessiner le tableau correspondant au bandeau dans son état final après exécution du script suivant :

   ```
   import RPi.GPIO as GPIO
   import Adafruit_WS2801
   import Adafruit_GPIO.SPI as SPI

   Obj_bandeau = Adafruit_WS2801.WS2801Pixels(15,spi=SPI.SpiDev(0,0),gpio=GPIO)
   Obj_bandeau.clear()
   for i in range(5):
       Obj_bandeau.set_pixel(i,16711680)
   for i in range(5,10):
       Obj_bandeau.set_pixel(i,16777215)
   for i in range(10,15):
       Obj_bandeau.set_pixel(i,255)
   Obj_bandeau.show()
   ```

   (b) Dessiner le tableau correspondant au bandeau dans son état final après exécution du script ci-dessous :

```
import RPi.GPIO as GPIO
import Adafruit_WS2801
import Adafruit_GPIO.SPI as SPI

Obj_bandeau = Adafruit_WS2801.WS2801Pixels(15,spi=SPI.SpiDev(0,0),gpio=GPIO)
Obj_bandeau.clear()
for i in range(15):
    if i%3 == 0:
        Obj_bandeau.set_pixel(i,32768)
    else:
        Obj_bandeau.set_pixel(i,65535)
Obj_bandeau.show()
```

3. *Utilisation de classe.*
On souhaite ajouter quelques fonctions supplémentaires. Pour cela, on crée une classe `Bandeau`. Le script est donnée ci-dessous :

```
1 def maximum(P):
2     m = depiler(P)
3     while not est_vide(P):
4         v = depiler(P)
5         if v > m:
6             m = v
7     return m
```

(a) On rappelle que la documentation (ou docstring) d'une fonction ou d'une méthode consiste à expliquer ce qu'elle fait, ce qu'elle prend en paramètre et ce qu'elle renvoie. Proposer une documentation à écrire à la ligne 9.

(b) Proposer également un commentaire à écrire à la ligne 38.

**Annexe 1 (exercice 4)**

**(à ne pas rendre avec la copie)**

⋆ **Types de données**

| | |
|---|---|
| `CHAR(t)`<br>`VARCHAR(t)`<br>`TEXT` | Texte fixe de `t` caractères<br>Texte de `t` caractères variables<br>Texte de 65 535 caractères au maximum |
| `INT` | Nombre entier de $-2^{31}$ à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé) |
| `FLOAT` | Nombre réel à virgule flottante |
| `DATE`<br>`DATETIME` | Date format AAAA-MM-JJ<br>Date et heure format AAAA-MM-JJHH :MI :SS |

⋆ **Quelques exemples de syntaxe SQL :**

- Insérer des enregistrements :

```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```

- Modifier des enregistrements :

```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```

- Supprimer des enregistrements :

```
DELETE FROM Table WHERE Selecteur
```

- Sélectionner des enregistrements :

```
SELECT attributs FROM Table WHERE Selecteur
```

- Effectuer une jointure :

```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE Selecteur
```

**Annexe 2 (exercice 5)**

**(à ne pas rendre avec la copie)**

Documentation de la librairie `Adafruit_WS2801`

1. **Méthodes de la librairie `Adafruit_WS2801` :**
   - ⋆ `WS2801Pixels(nb_px, spi, gpio)` : création d'un objet représentant le bandeau de LEDs.
     - `nb_px` : nombre de LEDs du bandeau
     - `spi` (Serial Peripheral Interface) : interface du micro-processeur du Raspberry.
     - `gpio` (General Purpose Input/Output) : gestion des ports entrée sortie du Raspberry.
   - ⋆ `RGB_to_color(r , g , b)` : renvoie un entier (`num_color`) correspondant à la couleur RGB. (voir tableau des correspondances des couleurs ci-dessous)
   - ⋆ `Adafruit_WS2801.WS2801Pixels(32,spi=SPI.SpiDev(0,0), gpio=GPIO)` : crée et renvoie un objet représentant le bandeau de 32 LEDs dans un script en `Python`.

2. **Méthodes sur un objet `Obj_bandeau` de type `Adafruit_WS2801.WS2801Pixels` :**
   - ⋆ `count()` : renvoie le nombre de LEDs de `Obj_bandeau`
   - ⋆ `show()` : affiche l'ensemble des modifications effectuées sur `Obj_bandeau` sur le bandeau.

   A noter que `Obj_bandeau` représente indirectement les LEDs du bandeau de LEDs, c'est-à-dire que l'on modifie d'abord `Obj_bandeau` (couleur des pixels, effacement, etc.), puis on applique les modifications sur le bandeau de LEDs avec la méthode `show()`.
   - ⋆ `clear()` : éteint toutes les LEDs de `Obj_bandeau`. Attention, cette méthode n'éteint pas directement les LEDs du bandeau de LEDs.
   - ⋆ `set_pixel(i, num_color)` : attribue à la LED numéro `i` de `Obj_bandeau` la couleur `num_color`.
     - `i` : entier compris entre `0` et `count()-1`.
     - `num_color` : entier correspondant à la couleur obtenue avec la méthode `Adafruit_WS2801.RGB_to_color(r,g,b)`

     Remarque : les modifications de `set_pixel` ne seront affichées sur le bandeau de LEDs qu'à la suite d'une instruction `show()`.
   - ⋆ `get_pixel_rgb(i)` : renvoie un tuple de trois entiers correspondant à la couleur RGB de la LED numéro `i` de `Obj_bandeau`. Pour une LED éteinte, cette méthode renvoie le tuple `(0,0,0)`.

Tableau de correspondance des couleurs :

| **Couleur** | Rouge | Bleu | Vert | Jaune | Orange | Noir | Blanc |
|---|---|---|---|---|---|---|---|
| **RGB** | `255,0,0` | `0,0,255` | `0,128,0` | `255,255,0` | `255,65,0` | `0,0,0` | `255,255,255` |
| **`num_color`** | `255` | `16711680` | `32768` | `65535` | `16895` | `0` | `16777215` |