# Modification du logo SIN



**Savoir-faire** : savoir identifier et caractériser les fonctions d'un produit.

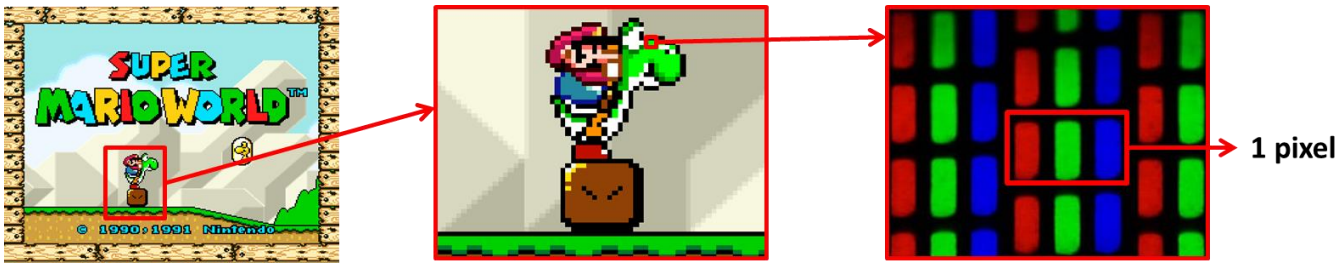
---

**Problématique** : Quel traitement informatique pour modifier les couleurs du logo SIN ?

## 1. L'image en informatique

Une image est composée de petits points appelés pixel. La définition d'une image donne le nombre de pixels qui compose l'image, par exemple une image de définition 800 x 600 (800 par 600), signifie que cette image est composée de 800 pixels en largeur et de 600 pixels en hauteur, soit en tout 800 x 600 = 480 000 pixels.

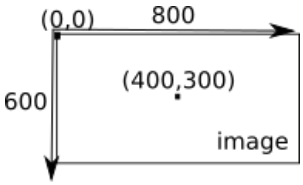
Un pixel est composé de trois parties : une partie rouge, une partie verte et une partie bleue. À chaque pixel on associe donc 3 couleurs : le rouge, le vert et le bleu. On parle de composantes rouge, vert ou bleu d'un pixel (on parle de système RVB ou RGB en anglais). La théorie physique de la synthèse additive des couleurs montre que la variation de l'intensité lumineuse de chaque composante permet d'obtenir un très grand nombre de couleurs. La valeur de l'intensité lumineuse associée à chaque composante de chaque pixel d'une image est codé sur 8 bits (un octet), chaque composante peut donc voir son intensité lumineuse variée de 0 à 255 (1 octet => 256 valeurs). On codera donc un pixel à l'aide d'un triplet de valeur (par exemple "247, 56, 98"). La première valeur donnant l'intensité de la composante rouge, la deuxième valeur donnant l'intensité de la composante verte et la troisième valeur donnant l'intensité de la composante bleue.




Quand on observe un pixel "à la loupe", on peut constater que le pixel est bien constitué de trois parties : une partie rouge, une partie verte, et une partie bleue (voir schéma ci-dessus).
Un pixel est tellement petit que l'œil humain superpose la partie rouge, la partie verte et la partie bleue du pixel, ceci explique pourquoi l'œil voit des pixels de différentes couleurs.

Le nom **pixel**, souvent abrégé **px**, provient de la locution anglaise "**pict**ure **el**ement", qui signifie "élément d'image".

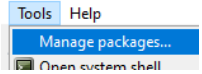
Dans une image, chaque pixel a des coordonnées x, y.
Par exemple sur le schéma ci-contre, le pixel de coordonnées (0, 0) se trouve en haut à gauche de l'image. Si l'image fait 800 pixels de large et 600 pixels de haut, le pixel ayant pour coordonnées (400, 300) sera au milieu de l'image.

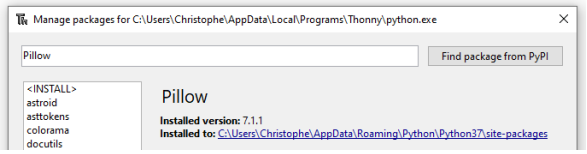



## 2. Première partie

La première partie consiste à afficher l'image.

Le traitement d'image, avec Python, va se faire à l'aide de la bibliothèque Pillow, qui est une bibliothèque de traitement d'images. Par conséquent il est nécessaire d'effectuer son installation.
**Q1 :** Dans Thonny, depuis le menu *Tools*, sélectionner *Manages packages…*, puis rechercher la bibliothèque Pillow et l'installer.



### 2.1. Afficher une image

**Q2 :** Tester le programme suivant qui permet d'afficher le logo.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
img.show()
```

Voici une analyse ligne par ligne du programme ci-dessus :

- **`from PIL import Image`** permet d'ajouter une extension (appelé bibliothèque) de Python pour travailler sur les images. Cette bibliothèque se nomme PIL.
- **`img = Image.open("logo_sin.bmp")`** permet de préciser que nous allons travailler avec l'image "**`logo_sin.jpg`** " et qui est maintenant associée à la variable **`img`**.
- **`img.show()`** permet d'afficher l'image associée à la variable **`img`**.

### 2.2. Lire un pixel

**Q3** : Tester le programme suivant qui permet de lire un pixel.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
r, v, b = img.getpixel((100, 100))
print(f"composante rouge : {r} ; composante verte : {v} ; composante bleue : {b}")
```

Ce programme donne les composantes rouge, verte et bleue du pixel de coordonnées (100, 100) de l'image "logo_sin.jpg".
Voici une analyse ligne par ligne du programme ci-dessus :

- **`r, v, b = img.getpixel((100,100))`** cette ligne récupère les valeurs du composantes rouge (r), verte (v) et bleue (b) du pixel de coordonnées (100,100). Dans la suite du programme, **`r`** correspond à la valeur de la composante rouge, **`v`** correspond à la valeur de la composante verte et **`b`** correspond à la valeur de la composante bleue.
- **`print(f"composante rouge : {r} ; composante verte : {v} ; composante bleue : {b}")`** permet d'imprimer le résultat.

**Q4** : À l'aide du programme ci-dessus et en le modifiant, donner les valeurs des composantes du pixel de coordonnée (5, 5).

```
composante rouge : 37 ; composante verte : 54 ; composante bleue : 146
```

### 2.3. Lire un pixel à l'aide d'une fonction

Une fonction est un sous-programme qui peut être appelé à différents endroits du programme principal, voir à partir d'une autre fonction.
La fonction **`lire_pixel(x, y)`** du programme suivant permet de lire un pixel dont on donne les coordonnées **`x`** et **`y`**.

```
from PIL import Image
img = Image.open("logo_sin.bmp")

def lire_pixel(x, y):
    """affiche les composantes du pixel de coordonnées x et y"""
    r, v, b = img.getpixel((x, y))
    print(f"composante rouge : {r} ; composante verte : {v} ; composante bleue : {b}")
```

**Q5** : Implémenter le programme précédent, qui contient la fonction **`lire_pixel(x, y)`**.

Tester la fonction à partir du shell comme ci-dessous :

```
>>> lire_pixel(50,50)
```

Tester la fonction en insérant le code suivant, qui permet d'appeler la fonction, dans le programme principal :

```
lire_pixel(50,50)
```

### 2.4. Modifier un pixel

**Q6** : Tester le programme suivant qui permet de modifier la couleur d'un pixel. Regarder attentivement le centre de l'image, vous devriez voir un pixel rouge à la place d'un pixel bleu.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
img.putpixel((86, 86), (255, 0, 0))
img.show()
```

L'instruction **`img.putpixel((86, 86),(255, 0, 0))`** permet de colorier le pixel de coordonnées (86, 86) en rouge (255, 0, 0)

**Q7** : Implémenter une fonction **`modif_pixel(x, y, r, v, b)`** qui permet de modifier un pixel de coordonnées **`x`** et **`y`** avec les composantes **`r`**, **`v`** et **`b`**.

### 2.5. Tracer une ligne

**Q8** : Compléter les pointillés du programme ci-dessous afin de mettre les pixels (25, 20), (25, 21), (25, 22) ... (25, 25) en vert.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
img.putpixel((..., ...), (..., ..., ...))
img.putpixel((..., ...), (..., ..., ...))
img.putpixel((..., ...), (..., ..., ...))
img.putpixel((..., ...), (..., ..., ...))
img.putpixel((..., ...), (..., ..., ...))
img.putpixel((..., ...), (..., ..., ...))
img.show()
```

Normalement, vous obtenez une ligne verticale de couleur verte prêt du bord de l'image.

**Q9** : Tester le programme ci-dessous qui permet d'éviter de réécrire plusieurs fois la même ligne à l'aide d'une boucle **`for`**.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
for y in range(50, 56, 1):
    img.putpixel((100, y), (255, 255, 0))
img.show()
```

Voici une analyse de l'instruction **`for y in range(50, 56, 1):`**

- Cette ligne **for…** est le début d'une partie du programme qui va être exécutée plusieurs fois (en boucle).

- Les lignes qui seront répétées sont placées juste après avec une marge gauche plus grande (on dit « indentation »).
- **y** est une variable qui va changer de valeur à chaque passage dans la boucle.
- **50** est la valeur de **y** au premier passage dans la boucle.
- **y** augmentera de **1** à chaque retour en début de boucle.
- Lorsque **y** aura la valeur **56**, la boucle ne sera plus exécutée et le programme continuera avec les lignes suivantes.

**Q10** : Implémenter une fonction **`ligne_horiz(coord_y_ligne, deb_ligne, fin_ligne)`** qui permet de tracer une ligne horizontale de coordonnée y (**`coord_y_ligne`**) entre les points définis par **`deb_ligne`** et **`fin_ligne`**.

**Q11** : Implémenter une fonction **`ligne_vert(coord_x_ligne, deb_ligne, fin_ligne)`** qui permet de tracer une ligne verticale de coordonnées x (**`coord_x_ligne`**) entre les points définis par **`deb_ligne`** et **`fin_ligne`**.

### 2.6. Modifier une image entière

En combinant deux boucles imbriquées, on peut agir sur une zone rectangulaire. Ici, il faut faire varier les valeurs x et y.

**Q12** : Tester le programme ci-dessous :

```
from PIL import Image
img = Image.open("logo_sin.bmp")
for x in range(10, 51, 1):
    for y in range(20, 61, 1):
        img.putpixel((x, y), (0, 255, 0))
img.show()
```

Pour changer la luminosité, le contraste ou mettre une image en noir et blanc, il faut modifier les pixels de toute l'image. Étant rectangulaire, pour agir sur l'image entière, il faut utiliser deux boucles imbriquées.
L'image du logo fait 165 pixels de large sur 172 pixels de haut.

**Q13** : Compléter les instructions des boucles **for** du programme.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
for x in range(..., ..., 1):
    for y in range(..., ..., 1):
        r, v, b = img.getpixel((x, y))
        new_r = r
        new_v = b
        new_b = v
        img.putpixel((x, y), (new_r, new_v, new_b))
img.show()
```

Le logo a changé de couleur !
Dans le programme, les composantes bleues et vertes sont inversées.

- **r, v, b = img.getpixel((x, y))** permet de récupérer les composantes rouge, verte et bleue du pixel de coordonnées (x, y).
- **new_r = r** enregistre dans la variable **new_r** (nouvelle valeur de la composante rouge) la valeur de **r** actuelle.
- **new_v = b** enregistre dans la variable **new_v** la valeur de **b** actuelle.

- **new_b = v** enregistre dans la variable new_b la valeur de v actuelle.
- **img.putpixel((x, y), (new_r, new_v, new_b))** colorie le pixel de coordonnée (x, y) avec les composantes new_r, new_v et new_b.

**Q14** : Tester le programme ci-dessous dans lequel l'instruction **size** permet de récupérer automatiquement la taille de l'image.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
largeur, hauteur = img.size
for x in range(0, largeur, 1):
    for y in range(0, hauteur, 1):
        r, v, b = img.getpixel((x, y))
        new_r = r
        new_v = b
        new_b = v
        img.putpixel((x, y), (new_r, new_v, new_b))
img.show()
```

### Filtre

L'application d'un filtre sur une image consiste à garder que la composante de la couleur du filtre.
Par exemple, pour appliquer un filtre rouge sur l'image, il faut conserver la valeur de la composante rouge et mettre les autres composantes à 0.

**Q15** : Compléter les pointillés du programme afin d'appliquer un filtre rouge sur l'image.

```
from PIL import Image
img = Image.open("logo_sin.bmp")
largeur, hauteur = img.size
for x in range(0, largeur, 1):
    for y in range(0, hauteur, 1):
        r, v, b = img.getpixel((x, y))
        new_r = ...
        new_v = ...
        new_b = ...
        img.putpixel((x, y), (new_r, new_v, new_b))
img.show()
```

**Q16** : Implémenter une fonction **filtre_rouge(image)** qui réalise un filtre rouge sur **image**.

**Q17** : Implémenter une fonction **filtre_vert(image)** qui réalise un filtre vert sur **image**.

### Négatif

Le négatif d'une image s'obtient en remplaçant la valeur du pixel par la valeur maximum des pixels (255) moins la valeur actuelle.
Donc si **x** est la valeur du pixel alors cela donne : **new_x = 255 - x**.

**Q18** : Implémenter une fonction **negatif(image)** qui donne le négatif de **image**.