Images numériques
Manipulation d'une image au format PGM avec une classe Python

Vous disposez d'une classe `pgm` écrite en Python 3.
Cette classe est présente dans le module `pgm.py`

La classe `pgm` est partiellement fonctionnelle.
L'objectif de cet exercice est de la compléter avec de nouvelles méthodes.

Ressources

```
├── doc_eleve
│   ├── camera_thermique
│   │   ├── thermal-imaging-house.jpg
│   │   └── thermal-imaging-house.pgm
│   ├── module_pgm_a_completer
│   │   └── pgm.py
│   ├── visionneuse_pgm
│   │   ├── image.pgm
│   │   └── visionneuse_pgm.py
│   └── module_pgm_complet_bytecode
│       ├── image2.pgm
│       ├── image.pgm
│       ├── pgm_python_34.pyc
│       ├── pgm_python_35.pyc
│       ├── pgm_python_37.pyc
│       ├── resultat
│       └── test.py
```

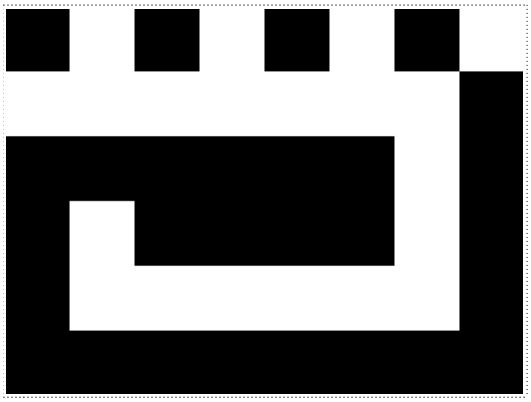
0. Classe `pgm` : documentation et exemple d'utilisation

Pour obtenir de l'aide sur la classe `pgm`, vous pouvez exécuter directement le module `pgm.py`

Le script `test.py` donne un exemple d'utilisation de la classe `pgm` dans sa version complète (modifier la commande `import` en fonction de la version de Python).

1. Création d'une image pixel par pixel

Avec un IDE Python 3, écrire le script `prog1.py` qui permet de créer le fichier `image_test_py.pgm` de l'image suivante :



Afficher également les propriétés de l'image, avec la méthode `properties()`

2. Modification de quelques pixels

Nouveau script `prog2.py`

A l'aide de la méthode `put_pixel()`, compléter le script précédent de manière à colorer :
- le pixel en haut à droite avec un gris `0xB2`
- le pixel en bas à gauche en blanc

Sauvegarder l'image : `image_test2_py.pgm`

3. Inversion du niveau de couleurs

Ouvrir le fichier `pgm.py` et compléter la méthode `invertcolor()` :

Pour cela, on dispose de la liste `self.datas_image`
Cette liste contient la valeurs des pixels de l'image originale.

Pour voir son contenu :
```
print(self.datas_image)
```

Le but est de créer la liste `self.datas_image_dest` de l'image inversée.

Deux techniques sont possibles :

Première technique : on balaye l'image originale pixel par pixel en commençant par le pixel n°0 jusqu'au dernier pixel (ici pixel n°47) :

```
# on copie la liste
self.datas_image_dest = self.datas_image.copy()

# index est le numéro du pixel
# self.__imagesize contient le nombre de pixels de l'image

for index in range(self.__imagesize):
    self.datas_image_dest[index] = self.datas_image[index]   # à modifier

# on retourne une nouvelle image
return self.__new_object(width=self.__width, height=self.__height)
```

Deuxième technique : on balaye de gauche à droite et de haut en bas, et on utilise les coordonnées (x, y) des pixels :

```
self.datas_image_dest = self.datas_image.copy()

for y in range(self.__height):
    for x in range(self.__width):
        # lecture de la couleur du pixel (x, y)
        pixel = self.get_pixel(x, y)
        # modification de la couleur du pixel (x, y)
        self.__set_pixel(x, y, pixel)    # à modifier

return self.__new_object(width=self.__width, height=self.__height)
```

Remarque : on peut aussi construire la liste `self.datas_image_dest` de cette manière :

```
self.datas_image_dest = list()  # liste vide

for y in range(self.__height):
    for x in range(self.__width):
        # lecture de la couleur du pixel (x, y)
        pixel = self.get_pixel(x, y)
        # on ajoute un élément en fin de liste
        self.datas_image_dest.append(pixel)    # à modifier

return self.__new_object(width=self.__width, height=self.__height)
```

Compléter le script `prog2.py` (à renommer `prog3.py`) de façon à inverser les couleurs du fichier `image_test2_py.pgm`
Sauvegarder l'image : `image_test3_py.pgm`

4. Un peu de bruit !

On veut ajouter du bruit à une image.

La méthode `noise(level)` est à compléter.

Vous aurez besoin du module `random` et de sa méthode `randint` :

```
import random
random.randint(-level, level)
```

Attention : il faut limiter le résultat dans l'intervalle 0 à 255.

5. Seuillage

Méthode `threshold(level)` à compléter.

6. Transformations miroirs

Méthodes `flipH()` et `flipV()` à compléter.

7. Rotations

Méthodes `rotate180()` `rotate90()` et `rotate270()` à compléter.

8. Découpage

Méthode `crop(x0, yo, width, height)` à compléter.

9. Filtres de convolution

En s'inspirant de la méthode `blur()`, compléter les méthodes :

```
edgedetect()
edgeenhance()
emboss()
sharpen()
```

Bibliographie et exemples :
https://docs.gimp.org/2.8/fr/plug-in-convmatrix.html
https://docs.gimp.org/2.8/en/plug-in-convmatrix.html

10. Caméra thermique

thermal-imaging-house.pgm

Il s'agit de l'image thermique d'une maison (échelle de température allant de 0 °C à +20 °C).

Traiter l'image de manière à obtenir les zones où la température est supérieure à T, puis les zones où la température est égale à T (isotherme).

Par exemple, pour T = 15 °C :