Le miniprojet est à faire par groupes de deux ou individuellement. Les groupes peuvent s'échanger des idées ou des approches générales, mais pas du code directement.

Préparation

Ouvrez Visual Studio Code et depuis le menu Terminal, choisissez New Terminal. Tapez-y ces lignes l'une après l'autre pour installer si nécessaire les modules de traitement d'image:

```
test -d venv && source venv/bin/activate
python3 -m pip install numpy Pillow types-Pillow
```

(Si la première ligne ne fonctionne pas mais que la seconde fonctionne, partez du principe que c'est OK.)

Téléchargez depuis Moodle le fichier **`miniproject-start.zip`** et décompressez-le. Déplacez ensuite à la racine de votre workspace les deux fichiers **`miniprojectutils.py`** et **`miniproject.py`**, ainsi que le dossier **`imgs`**, qui contient quelques images exemples.

- **`miniprojectutils.py`** contient les fonctions que vous pouvez appeler sans avoir besoin de forcément tout comprendre. Vous pouvez consulter leur implémentation et voir comment elles utilisent les bibliothèques **`numpy`** et **`Pillow`** pour faire leur travail. Merci de ne **pas** modifier ce fichier.
- **`miniproject.py`** est le fichier que vous devrez petit à petit compléter. Il contient déjà le code de base, partiellement commenté, décrit au cours.

Lancez le fichier **`miniproject.py`**. Il devrait afficher deux lignes sur la console en disant qu'il commence le travail et qu'il charge une image, puis s'arrêter juste après. Si cela ne fonctionne pas, appelez un·e assistant·e.

Partie 1. Conversion en niveaux de gris

(a) Commencez par implémenter la fonction **`rgb_to_gray`**, dont le but est de convertir une couleur RGB en un niveau de gris. Cette fonction reçoit donc trois arguments, **`r`**, **`g`** et **`b`**, tous entre 0 et 255 (compris), et doit retourner un **`int`** entre 0 et 255 aussi.

On pourrait se dire qu'une bonne méthode de conversion en niveau de gris serait de faire la moyenne entre les trois composantes RGB. Mais cela fonctionne mal: notre œil perçoit les contributions des composantes à la clarté d'une couleur de manière différente. Basez-vous plutôt sur cette formule pour faire une somme pondérée, dont vous voyez qu'elle donne une importance prépondérante à la composante verte:

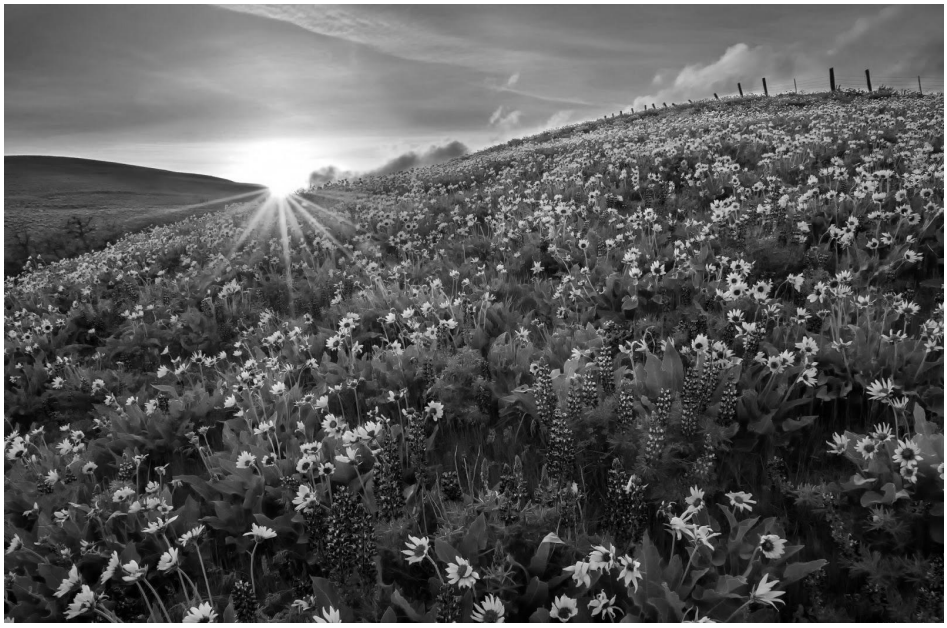
$$gray = 0.2126 \cdot r + 0.7152 \cdot g + 0.0722 \cdot b$$

Rajoutez en bas du fichier (mais *dans* la condition **`if __name__ == "__main__"`**) quelques lignes pour vérifier que votre fonction retourne toujours bien un int entre 0 et 255.

(b) Implémentez la fonction **`to_grayscale`**. Elle reçoit en paramètre une image couleur et doit retourner une nouvelle image en niveaux de gris de la même taille, obtenue en convertissant chaque pixel avec la fonction **`rgb_to_gray`**.

Pour tester votre code, décommentez dans la fonction **`highlight_rectangle_test`** le code lié à l'étape 1. Si tout se passe bien, un fichier image est généré ici dans votre workspace: **`imgs/sunrise/1_gray.jpg`**. Ouvrez-le et vérifiez que l'image obtenue est celle en haut de la page suivante.

Si vous avez des difficultés, essayez avec une petite image de 10×10 pixels que vous générez aléatoirement (avec la fonction **`new_random_rgb_image`** de **`miniprojectutils.py`** par exemple) plutôt que directement avec la grande image. C'est plus simple de voir les valeurs intermédiaires (en faisant **`print`** ou via le débuggeur) sur de petites images.



Partie 2. Recherche d'un motif

(a) Essayons d'abord de dessiner par dessus une image pour mettre en avant une zone donnée. Nous allons pour cela utiliser et implémenter la fonction **`highlight_rectangle`**. Cette fonction prend en paramètre une image en niveau de gris, les coordonnées d'un point, une taille, une valeur int entre 0 et 255 (inclus), et une largeur de ligne. L'idée de cette fonction et qu'elle doit créer une **nouvelle** image (c'est important !) dont les pixels sont identiques à celle passée en paramètre, à l'exceptions de ceux présents sur les contours d'un rectangle. Ce rectangle est défini par les coordonnées du coin supérieur gauche, et la taille du rectangle, tout deux passé en paramètres. Les contours doivent être dessinés à l'extérieur du rectangle avec une ligne dont la largeur et la valeur des pixel est donné en paramètres.

Faites bien attention à ne pas dessiner à l'intérieur du rectangle mais bien à l'extérieur. Par exemple, si on appelle **`highlight_rectangle(img, (1,2), (4,2), 255, 3)`** sur une image de taille 10x10, alors sur l'image retourné, les pixels (1, 2) et (4, 3) doivent être inchangés car ils sont à l'intérieur du rectangle, mais les pixel (0, 3) et (3, 4) seront blanc car ils sont tout les deux sur les contours du rectangle. Faites aussi attention à ne pas dessiner en dehors de l'image: Si le rectangle ou ses contours dépassent de l'image, dessinez seulement la partie visible (La méthode **`clamp`** disponible dans **`miniprojectutils.py`** peut vous aider).

Testez votre fonction en l'appelant avec quelques valeurs, et en décommentant le code lié à cette étape de la fonction **`highlight_rectangle_test`**.

(b) Vous allez maintenant implémenter la fonction **`pattern_difference`**. Cette fonction prend en argument une grande image "source", une petite image "pattern" (ou motif en français), et une position, et retourne un float. La source et le pattern doivent être en niveau de gris. Le but de cette fonction et de calculer un score de différence entre le pattern et un morceau (de même taille) de la deuxième image. Plus le score est faible, plus le pattern est similaire au morceau de la deuxième image, avec un score de 0 indiquant qu'ils sont identiques. Ce morceau est un rectangle de même taille que le pattern et dont le coin supérieur gauche (coordonnée minimales) est défini par la position donnée en argument. Cette position doit être de telle sorte que le rectangle ainsi formé soit compris entièrement dans l'image source, sinon la fonction doit retourner 1. Pour calculer le score de similarité, on commence par calculer la différence absolue entre la valeur des pixels du pattern et la valeur des pixels dans le morceau de la deuxième images. On fait ensuite la moyenne des différences et on obtiens ainsi une valeur entre 0 et 255 (inclus). On divise enfin par 255 pour obtenir un score de différence entre 0 et 1.

Pour cette fonction, il est **très important** que vous n'utilisiez **PAS** de boucle (for ou while) car sinon l'exécution de la prochaine étape sera trop longue et vous n'aurez pas tout les points. À la place, vous devez utiliser les opérateurs numpy, permettant de faire vos calculs plus rapidement.

Vous pouvez à nouveau ajouter en bas du fichier (mais *dans* la condition **`if __name__ == "__main__"`**) quelques lignes pour tester votre programme. (Par exemple, que se passe t'il si l'on passe 2x la même image et les coordonnées (0,0) ?) Vous pouvez aussi créer vos propres images de tests (en rognant et modifiant des images pour créer un pattern similaire) avec des outils libres et gratuits comme GIMP, ou simplement en définissant vos propres arrays numpy.

(c) Vous allez maintenant implémenter la fonction **`find_similar_pattern_position`**. Cette fonction prend en argument une image source et un pattern (tout deux en niveaux de gris), et retourne une position. Le but et de trouver le morceau de l'image source qui ressemble le plus au pattern. Pour cela, vous devez utiliser la fonction **`pattern_difference`** que vous venez d'implémenter pour trouver la position qui donne la plus faible différence, et retourner cette position. Si la taille du pattern ne rentre pas dans l'image source, retournez (0,0).

Attention: Cette fonction peut prendre quelques secondes à s'exécuter, même avec un petit pattern, et potentiellement beaucoup plus si le pattern est grand.

(d) Pour finir, vous allez maintenant implémenter la fonction **`highlight_similar_pattern`**. Cette fonction prend en argument une image source et une image pattern (tout deux en niveaux de gris), une valeur de gris (entre 0 et 255 inclus), et une largeur de ligne, et doit retourner une copie de l'image source dont la seule différence est que le morceau le plus similaire est entouré par une ligne de la largeur et valeur donnée en argument. Cette fonction doit utiliser les fonctions précédentes.

Testez votre code en décommentant les étapes de la fonction **`highlight_similar_pattern_test`**.

Vous pouvez modifier l'appel à **`highlight_similar_pattern_test`** pour tester votre code avec les autres images et patterns d'exemple.