Exercice 1 *Préparation*

1. Copier le sous-dossier `Activite2` du dossier `ICN\Images` situé dans le partage `Donnees` de sa classe. Coller ce dossier dans son espace personnel.
2. `Activite2` contient un sous-dossier `ressources` avec les fichiers images que nous allons manipuler ainsi qu'un fichier `cadeau.py` contenant les codes `Python` à compléter. Ouvrir `Pyzo`, enregistrer dans `ressources` un fichier `Images-Activite2.py`, puis exécuter ce fichier avec l'option `Run as script`.

1 Traitement d'une image pixel par pixel

Exercice 2 *Inversion*

1. Deux pixels sont complémentaires si la somme de leurs valeurs est égale à la valeur maximale, celle d'un pixel blanc. La fonction `invpixel_gray(pix)` ci-dessous prend en paramètre la valeur de niveau de gris d'un pixel, comprise entre 0 et 255, et retourne sa valeur complémentaire. Écrire une fonction `invpixel_rgb(pix)` qui retourne la valeur complémentaire d'un pixel représenté en (R,G,B) ; `invpixel_rgb((255,0,240))` doit retourner $(0,255,15)$.

```
def invpixel_gray(pix):
    return 255 - pix
```

2. Compléter le code de la fonction `inversion(imsource)`, pour qu'elle qu'elle retourne une image créée en appliquant la fonction d'inversion adaptée au mode de l'image ('L' ou 'RGB') à tous les pixels de l'image source `imsource`.

```
def inversion(imsource):
    pixelsource = imsource.load()
    (L, H) = imsource.size
    imbut = Image.new(imsource.mode, (L, H))
    pixelbut = imbut.load()
    if imsource.mode == 'L':
        invpixel = invpixel_gray
    else:
        invpixel = invpixel_rgb
    for y in range(H):
        for x in range(L):
            #  a completer
    return imbut
```

Tester la fonction sur l'image du tableau « Cardinal Fernando Niño de Guevara (1541–1609) » peint par *El Greco* [1] :

```
In [1]: im1 = Image.open("cardinal-el-greco.jpg")

In [2]: im2 = inversion(im1)

In [3]: im2.save("cardinal-inverse.jpg")
```

1. Metropolitan Museum of New York licence CC0 1.0

Original

Inversion

Flip

Exercice 3 *Une transformation géométrique*

Compléter le code de la fonction `flip(imsource)` pour qu'elle retourne l'image obtenue par symétrie de l'image source par rapport à l'un de ces bords verticaux.

```
def flip(imsource):
    pixelsource = imsource.load()
    (L, H) = imsource.size
    imbut = Image.new(imsource.mode, (L, H))
    pixelbut = imbut.load()
    # a completer
    return imbut
```

Exercice 4 *Réduction*

Pour réduire une image source de dimension `(L, H)` d'un coefficient `coef` qui divise L et H, il suffit de créer une image de dimension `(L // coef, H // coef)` puis pour chaque pixel `pixelbut[x, y]` de cette image reçoit la valeur du pixel `pixelbut[x * coef, y * coef]` de l'image source.

1. Compléter le code de la fonction `reduire_image(imsource, coef)`.

   Comparer la réduction agrandie avec l'image originale.

```
def reduire_image(imsource, coef):
    pixelsource = imsource.load()
    (L, H) = imsource.size
    imbut = Image.new(imsource.mode, (L // coef, H // coef))
    pixelbut = imbut.load()
    # a completer
    return imbut
```

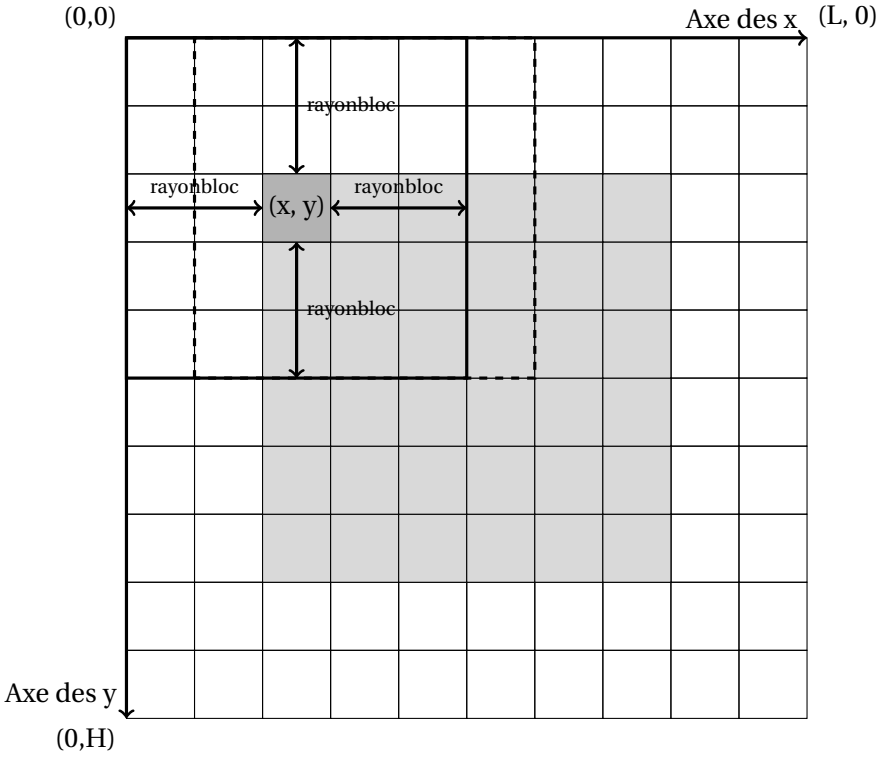
2. Comment modifier cette fonction pour un agrandissement ? Le résultat sera-t-il satisfaisant ?

2 Traitement d'une image pixel par blocs de pixel

Exercice 5 *Floutage*

1. Récupérer la fonction `moyenne_bloc(pixel, xmin, ymin, xmax, ymax, mode)` dans le fichier `cadeau.py`. Ses paramètres sont : `pixel` la matrice de pixels d'une image, `mod` de valeur 'L' ou 'RGB' et les coordonnées de deux pixels `(xmin, ymin)` et `(xmax, ymax)` qui sont respectivement le coin supérieur gauche et le coin inférieur droit d'un bloc rectangulaire de pixels. La fonction retourne la moyenne des valeurs des pixels de ce bloc.

Floutage



2. Pour flouter, une image de dimension (L, H) on va créer une nouvelle image où chaque pixel en position `(x,y)` de l'image source est remplacé par la moyenne de ses pixels voisins. Le voisinage choisi est un bloc carré de centre `(x,y)`, de coin supérieur gauche `(x - rayonbloc, y - rayonbloc)` et de coin inférieur droit `(x + rayonbloc, y + rayonbloc)` (en gras sur la figure). Pour que le bloc ne sorte pas des limites de l'image source, on ne prend comme centre des blocs que les pixels à une distance `rayonbloc` des bords (zone grisée sur la figure)

   Compléter la fonction `floutage(imsource, rayonbloc)` qui retourne une image floutée d'une image source avec les blocs de voisinage carrés décrits précédemment.

```
def floutage(imsource, rayonbloc):
    (L, H) = imsource.size
    imbut = Image.new(imsource.mode, (L, H))
    pixelsource = imsource.load()
    pixelbut = imbut.load()
    #a completer
    return imbut
```

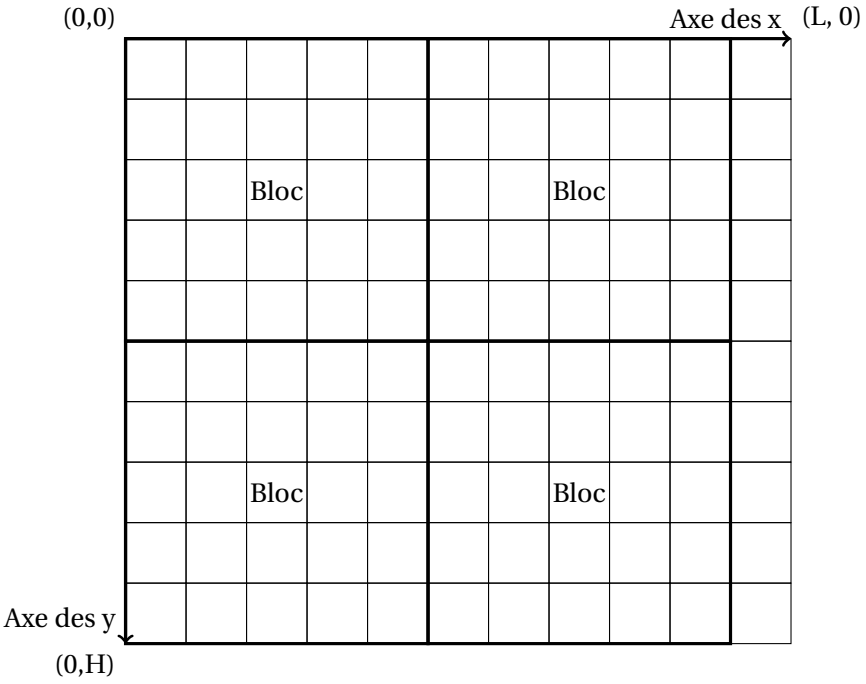
3. Comment expliquer la petite bande noire autour de l'image floutée.

   Peut-on modifier la fonction de floutage pour la supprimer ?

Exercice 6 *Pixellisation*

Pour pixelliser une image de dimension (L, H), on va découper sa matrice de pixels en blocs rectangulaires puis créer une nouvelle image où tous les pixels d'un bloc reçoivent la même valeur qui est la moyenne du bloc dans l'image source. Si le côté du bloc n'est pas un diviseur commun, certaines bandes sur les côtés ne seront pas traitées.

Pixellisation



1. Écrire une fonction `bloc_monochrome(pixel, xmin, ymin, xmax, ymax, couleur)` qui prend en paramètre une matrice de pixels et affecte la même couleur à tous les pixels d'un bloc de coin supérieur gauche `(xmin, ymin)` et de coin inférieur droit `(xmax, ymax)`.

2. Compléter la fonction `pixellisation(imsource, taillebloc)` où le paramètre `taillebloc` est le côté d'un bloc.

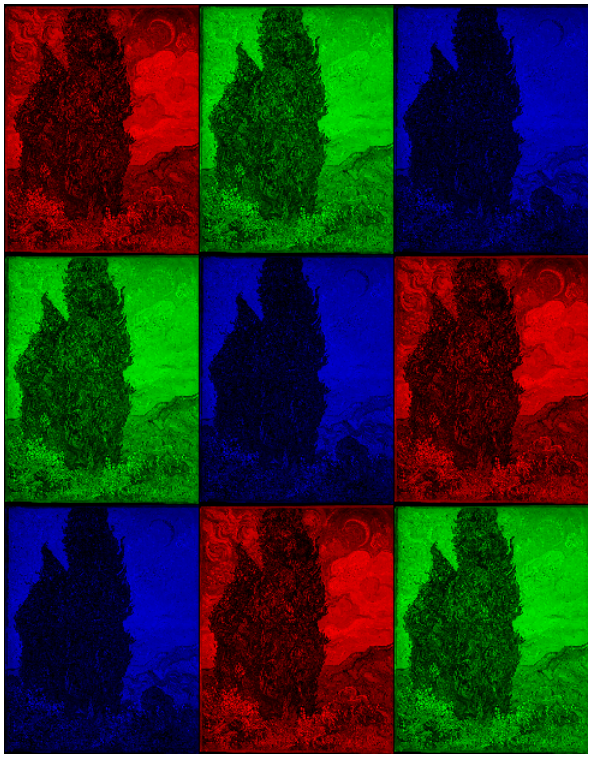
```
def pixellisation(imsource, taillebloc):
    (L, H) = imsource.size
    imbut = Image.new(imsource.mode, (L, H))
    pixelsource = imsource.load()
    pixelbut = imbut.load()
    for y in range(..............):
        for x in range(............):
            (xmin, ymin) = ............
            (xmax, ymax) = ..............
            moyenne = moyenne_bloc(.......................)
            bloc_monochrome(.......................)
    return imbut
```

3 Image constituée de vignettes / blocs images

Exercice 7 *Sérialisation*

À partir de l'image contenue dans le fichier `cypres.bmp`[a], on veut réaliser une mosaique constituée de trois vignettes ou blocs images qui sont les composantes (Rouge ,Vert ,Bleu) de l'image source après réduction d'un certain facteur.
On va utiliser les fonctions `filtre_composante` vue dans l'activité 1 et `reduire_image` pour concevoir les trois vignettes.

a. Metropolitan Museum of New York licence CC0 1.0



1. Compléter la fonction `creer_bloc(imsource, nbloc)` qui doit retourner une liste contenant les matrices de pixels des trois blocs/vignettes (Rouge ,Vert ,Bleu) obtenus après réduction de l'image source du facteur `nbloc`.

```
def creer_bloc(imsource, nbloc):
    pixelbloc = []
    for canal in range(3):
        im = reduire_image(..............)
        pixelbloc.append(..............)
    return pixelbloc
```

2. Le nombre de blocs en largeur et en hauteur est donné par `(Lbloc, Hbloc) = (L // nbloc, H // nbloc)`.

Si un pixel a pour coordonnées `(x,y)` dans l'image source, alors si on compte en blocs par rapport à l'origine située dans le coin supérieur gauche `(0,0)` de l'image ses coordonnées sont `(x // Lbloc, y // Hbloc)`.

De plus le pixel appartient à un bloc et ses coordonnées dans le repère d'origine le coin supérieur de ce bloc sont `(x % Lbloc, y % Hbloc)`.

Compléter la fonction `mosaique_rgb(imsource, bloc, nbloc)` qui doit retourner une mosaique similaire à celle-ci dessus pour la valeur 3 du paramètre `nbloc`.

```
def mosaique_rgb(imsource, bloc, nbloc):
    (L, H) = imsource.size
    pixelbloc = creer_bloc(imsource, nbloc)
    imbut = Image.new(imsource.mode, (L, H))
    pixelbut = imbut.load()
    (Lbloc, Hbloc) = (L // nbloc, H // nbloc)
    # a completer
    return imbut
```