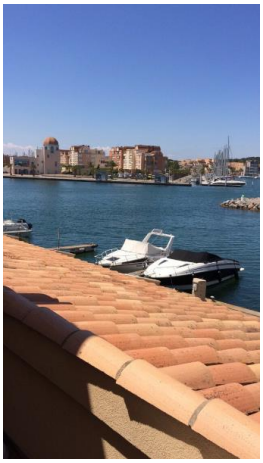
# *Un enseignement au numérique pour notre académie : Différents traitements d'une même image en 9 fiches*

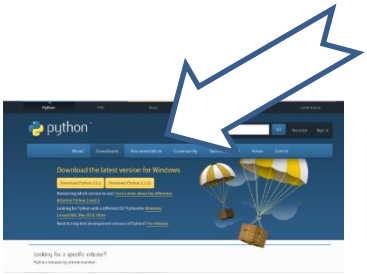


*L'objet de cet article est de partir d'une même image numérique qui sera conservée sur le bureau de notre ordinateur et d'opérer sur cette même image différents traitements classiques.*

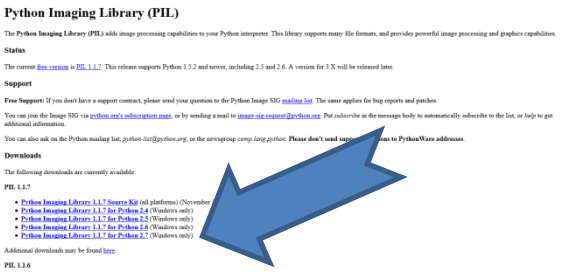
*On propose ainsi de parcourir en 9 fiches un chemin partant du traitement d'une image couleur en noir et blanc en allant jusqu'à l'agrandissement d'une telle image en passant par différents traitements rangés du plus simple au plus compliqué.*

**Utilitaire pour les fiches :**
**Un logiciel, une bibliothèque et une image numérique**

**Pour ce qui est du logiciel à utiliser** :
Il est nécessaire de télécharger :
- Le logiciel Python dans sa version 2.7.12 ;
- Le module PIL de Python pour Python 2.7.



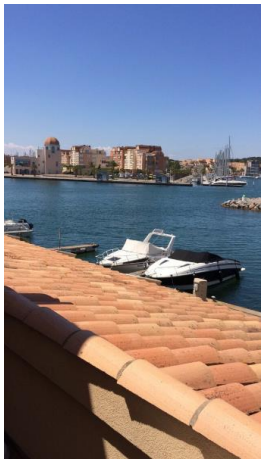

Nous vous proposons ainsi de télécharger un logiciel de programmation qui s'appelle Python (comme le serpent), attention à bien télécharger une version déjà relativement ancienne : la version 2.7.12.
Pour ce faire, utiliser le lien : https://www.python.org/downloads/




Il faut ensuite aller chercher ce que l'on appelle un module Python c'est-à-dire une bibliothèque de petits programmes Python que l'on pourra ainsi utiliser à souhait. Ce module s'appelle PIL, il est accessible à l'adresse suivante :
http://www.pythonware.com/products/pil/
Attention à bien sélectionner la version de PIL correspondant à Python 2.7.

**Résultat des opérations :** vous avez sur votre ordinateur le logiciel Python et sa bibliothèque PIL.

## Pour ce qu'il en est de la photo numérique :

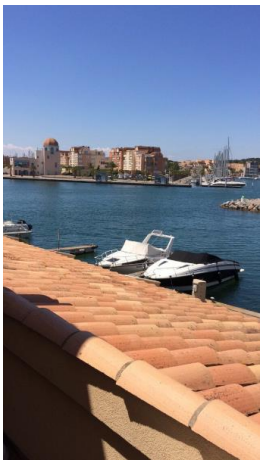


Nous allons utiliser systématiquement dans les 9 fiches la même image, nous vous proposons ainsi de prendre une photo numérique de votre choix que vous allez sauvegarder sur votre bureau en nommant le fichier Photo de départ. Pour ce faire, il vous suffit d'opérer un clic droit sur la photo ci-contre puis de choisir l'option *Enregistrer sous* puis de choisir le dossier *Bureau* et prendre le format jpg.

Pour obtenir l'adresse de votre photo, il vous suffit d'aller à présent sur votre bureau, de cliquer droit sur votre image et de lire l'adresse dans le menu *Propriétés* puis *Sécurité*. L'adresse est de la forme :

C:\Users\Nom de l'utilisateur\Bureau\Photo de départ.JPG

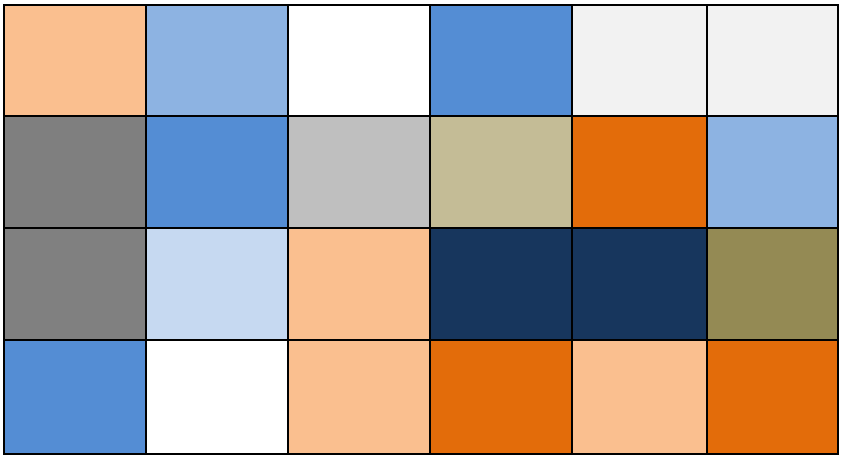
# Découvrir la notion de dimension d'une image numérique



Lorsque l'on fait un dessin avec des feutres et des crayons de couleurs, on obtient une image qui n'a rien de numérique.

Lorsqu'en revanche, on prend une photo avec un téléphone portable, la photo que l'on obtient est une image numérique. On qualifie cette image de numérique car elle est définie avec des nombres. Si l'on regarde en effet de plus près une image numérique, on se rend compte qu'elle est formée avec plein de petits carrés de couleurs identiques ou différentes qui sont mis l'un à côté de l'autre.

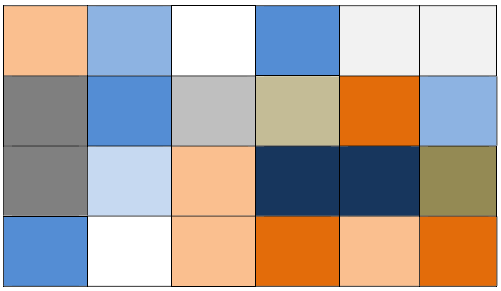
Voici ci-dessous une image numérique, elle est formée de plein de petits carreaux (les pixels qui sont placés l'un contre l'autre).

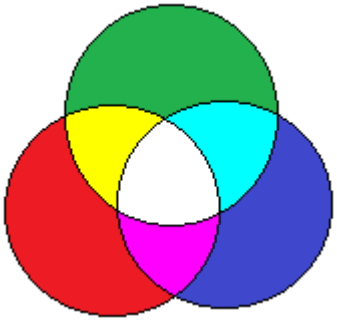


Au total dans cette image, nous avons 6 colonnes et 4 lignes (donc au total 24 pixels). Nous avons donc des nombres 6, 4, 24 mais ce n'est pas uniquement à cause de ces nombres que nous allons parler d'image numérique. Le couple (6,4) s'appelle **les dimensions** de l'image, on dira que l'on a une image 6 par 4 que l'on note : 6 X 4.

# Découvrir le codage numérique des couleurs

Reprenons l'image numérique de la page précédente, nous allons voir à présent comment l'ordinateur va savoir que la couleur à placer en haut à gauche est de couleur ocre pale proche du rose ; que sa voisine est bleue, que la suivante est blanche …





Pour ce qui concerne les couleurs, le numérique procède comme un artiste peintre qui arrive avec les trois couleurs primaires à reconstituer toutes les couleurs. En effet par addition à doses différentes des trois couleurs Rouge, Vert et Bleu, on arrive à reconstituer toutes les couleurs de l'arc-en-ciel.

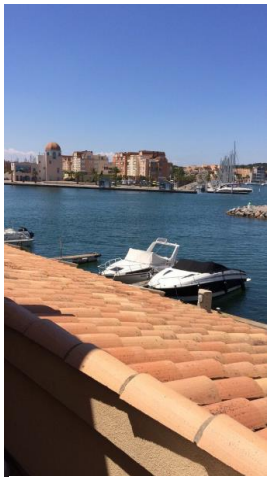
**Une image est dite au format RGB lorsque la couleur de chacun de ses pixels est construite à partir d'un triplet de nombres entiers (r,g,b) où :**
**- l'entier r compris entre 0 et 255 représente le niveau de rouge ;**
**- l'entier g compris entre 0 et 255 représente le niveau de vert (green en anglais) ;**
**- l'entier b compris entre 0 et 255 représente le niveau de bleu.**

Afin d'illustrer ce codage RGB des couleurs, nous allons construire une image dont vous pourrez faire varier à souhait la couleur.

```
>>> from PIL import Image
>>> def new():
        colonne,ligne=300,150
        imagearrivee=Image.new('RGB',(colonne,ligne))
        for x in range(colonne):
            for y in range(ligne):
                imagearrivee.putpixel((x,y),(120,200,90))
        imagearrivee.save("C:\Users\Nom de l'utilisateur\Bureau\Photo.JPG")
>>> new()
```

On reprend l'adresse de l'image de départ, adresse que l'on modifie juste à la fin en remplaçant *Photo de départ* par *Photo*. L'image ainsi construite est placée sur votre bureau à l'adresse que vous allez donner sur cette ligne. Vous pouvez modifier à souhait le triplet (120,200,90) afin d'obtenir une image d'une autre couleur.

## Fiche n°1 : Transformer une image en noir et blanc



Nous allons effectuer ce que l'on appelle un seuillage. Pour ce faire, nous choisissons un seuil (par exemple 127). Nous rappelons que la couleur de chaque pixel d'une image au format RGB est construite à partir d'un triplet (r,g,b) de niveaux des couleurs Rouge, Vert et bleu. Dès qu'une des trois valeurs r, g ou b dépasse la valeur du seuil, nous attribuons au pixel le code couleur (255,255,255) (couleur blanche) ; si tel n'est pas le cas, nous lui attribuons le code (0,0,0) (couleur noire).
Si l'on prend par exemple, le code couleurs (123, 74, 210) du petit carré ci-contre ; ce code va être remplacé par (255,255,255). On obtient la couleur blanche que l'on a utilisée pour remplir le carré ci-contre. Les seules couleurs qui vont émerger de ce seuillage sont au nombre de 2 sont la couleur noire (0,0,0) et la couleur blanche (255,255,255).

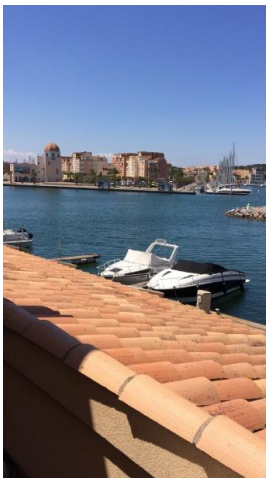
**Nous allons à présent effectuer cette tâche afin de traiter notre photo, ainsi pour chaque pixel de coordonnées (x,y),:**
**1) nous relevons son code couleurs (r,g,b) ;**
**2) nous construisons l'image d'arrivée, en affectant le nouveau code couleurs à chaque pixel (une fois effectué le test sur chacune des valeurs des variables r, g et b).**

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

```
>>> from PIL import Image
>>> a=" C:\Users\Nom de l'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def noirblanc(image):
        (c,l)=image.size                    ***on récupère ici les dimensions de l'image***
        imagearrivee=Image.new('RGB',(l,c))    ***On crée une image d'arrivée de même dimensions***
        for x in range(c):                  *** On va parcourir toutes les colonnes***
            for y in range(l):              *** Pour chaque colonne, on va parcourir toutes les lignes***
                pixel=image.getpixel((x,y))          ***on récupère le triplet (r,g,b)***
                if pixel[0]>127 or pixel[1]>127 or pixel[2]>127 :
                                p=(255,255,255)
                        else :    p=(0,0,0)
                imagearrivee.putpixel((x,y),p)
        imagearrivee.save(''C:\Users\Nom de l'utilisateur\Bureau\Photo n°1.JPG'')
>>> noirblanc(im)                           ***on lance la fonction noirblanc() appliquée à notre image ***
```

Bien mettre l'adresse de l'image de départ qui correspond à votre ordinateur.
L'image obtenue sera placée sur votre bureau à l'adresse que vous allez donner.

# Fiche n°2 : Filtrer une image couleur



Quoi de plus beau pour une photo de la mer que de ne conserver que les tons de bleu et de faire disparaître les autres. C'est ce que nous proposons de réaliser ici en produisant le même effet que si l'on regardait la photo en lui plaçant dessus **un filtre transparent bleu (qui ne laisse passer que la couleur bleue)**.

Nous savons que dans le codage (r,g,b) le niveau de couleur bleue est donné par la valeur de b. Pour ne conserver que ce niveau de bleu de chaque pixel, il suffit de **remplacer le triplet (r,g,b) par le triplet (0,0,b)**.

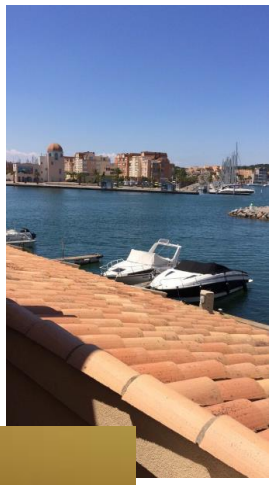
**Nous allons à présent effectuer cette tâche de façon concrète afin de traiter notre photo, ainsi pour chaque pixel de coordonnées (x,y),:**
**1) nous relevons son code couleurs (r,g,b) ;**
**2) nous construisons l'image d'arrivée, en affectant le nouveau code couleurs à chaque pixel (en remplaçant tout simplement le triplet (r,g,b) par (0,0,b)).**

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Desktop\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def bleu(image):
        (c,l)=image.size
        imagearrivee=Image.new('RGB',(c,l))
        for x in range(c):
            for y in range(l):
                pixel=image.getpixel((x,y))
                p=(0,0,pixel[2])
                imagearrivee.putpixel((x,y),p)
        imagearrivee.save("C:\Users\Nom d'utilisateur\Bureau\Photo n°2.JPG")
>>> bleu(im)
```

L'image ainsi construite est placée sur votre bureau à l'adresse que vous allez donner sur cette ligne.

# Fiche n°3 : Inverser les couleurs d'une image RGB



Nous nous répétons en disant que la couleur de chaque pixel d'une image au format RGB est construite à partir d'un triplet (r,g,b) de niveaux des couleurs Rouge, Vert et bleu.
Si l'on prend par exemple (255,0,0), on obtient la couleur Rouge, si l'on prend (123, 74, 210) on obtient la couleur violette que l'on a utilisée pour remplir le carré ci-contre.

Dans cette fiche, nous allons vous faire découvrir une méthode classique de transformation d'une image couleur. Cette méthode dite d'inversion des couleurs consiste à modifier chaque pixel de code couleurs (r,g,b) en un pixel de code couleurs (255-r,255-g,255-b).
Si l'on reprend le code couleurs précédent (123,74,210), nous allons obtenir le triplet (132,181,45) qui correspond à la couleur verte que l'on a utilisée pour remplir le carré ci-contre.

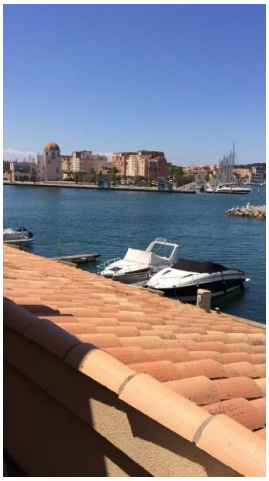
**Nous allons à présent effectuer cette tâche afin de traiter notre photo, ainsi pour chaque pixel de coordonnées (x,y),:**
**1) nous relevons son code couleurs (r,g,b) ;**
**2) nous construisons l'image d'arrivée, en affectant le code couleurs (255-r,255-g,255-b) au pixel de coordonnées (x,y) de cette nouvelle image.**

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

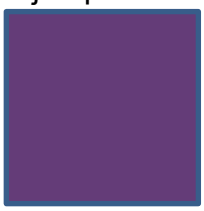
```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def gris(image):
        imagearrivee=Image.new('RGB',(c,l))
        for x in range(c):
            for y in range(l):
                pixel=image.getpixel((x,y))
                p=(255-pixel[0], 255-pixel[1], 255-pixel[2])
                imagearrivee.putpixel((x,y),p)
        imagearrivee.save("C:\Users\Nom de l'utilisateur\Bureau\Photo n°3.JPG")
>>> gris(im)
```

L'image ainsi construite est placée sur votre bureau à l'adresse que vous allez donner sur cette ligne.

# Fiche n°4 : Transformer une image couleurs en une image en niveaux de gris



Dans cette fiche, nous allons voir comment à transformer cette belle photo en une image en niveaux de gris c'est-à-dire une image construite uniquement avec des pixels gris (en partant du blanc pour aller jusqu'au noir).



Nous avons vu que la couleur de chaque pixel d'une image au format RGB est construite à partir d'un triplet (r,g,b) de niveaux des couleurs Rouge, Vert et bleu. Si l'on prend par exemple (255,0,0), on obtient la couleur Rouge, si l'on prend (123, 74, 210) on obtient la couleur violette que l'on a utilisée pour remplir le carré ci-contre.

Pour obtenir du gris, rien de plus facile : il suffit de prendre un triplet de trois valeurs égales. Par exemple, si l'on prend le triplet (100,100,100), on obtient le gris foncé du carré ci-contre.

**Nous allons construire à présent l'image ci-contre à partir de notre photo en couleurs. Pour chaque pixel :**

**1) nous prenons le triplet (r,g,b) et nous calculons la partie entière n de la moyenne des trois valeurs de ce triplet. Par exemple : pour le triplet (200,78,135), on obtient 137 ;**

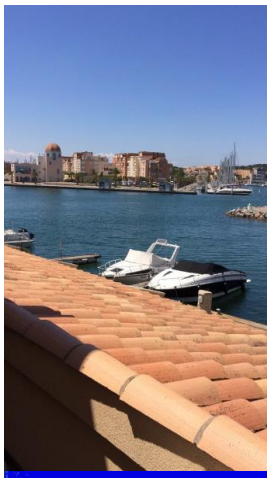
**2) nous remplaçons la couleur (r,g,b) du pixel par (n,n,n). Comme les trois éléments de ce triplet ont la même valeur, on obtient du gris.**

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def gris(image):
        (l,c)=image.size
    imagearrivee=Image.new('RGB',(c,l))
    for x in range(l):
        for y in range(c):
            pixel=image.getpixel((x,y))
            gris=int((pixel[0]+pixel[1]+pixel[2])/3)
            p=(gris,gris,gris)
            imagearrivee.putpixel((x,y),p)
    imagearrivee.save("C:\Users\Nom de l'utilisateur\Bureau\Photo n°4.JPG")
>>> gris(im)
```

L'image ainsi construite est placée sur votre bureau à l'adresse que vous allez donner sur cette ligne.

# Fiche n°5 : Transformer une image en 8 couleurs



Nous allons effectuer ce que l'on appelle un seuillage pour ce faire, nous choisissons un seuil (par exemple 127). Nous rappelons que la couleur de chaque pixel d'une image au format RGB est construite à partir d'un triplet (r,g,b) de niveaux des couleurs Rouge, Vert et bleu. Dès qu'une des trois valeurs r, g ou b dépasse la valeur du seuil, nous la remplaçons par 255, si tel n'est pas le cas, nous la remplaçons par 0. Si l'on prend par exemple, le code couleurs (123, 74, 210) ; celui-ci va être remplacé par (0,0,255) on obtient la couleur violette que l'on a utilisée pour remplir le carré ci-contre. Les seules couleurs qui vont émerger de ce seuillage sont au nombre de 8, celles-ci sont les couleurs codées par les triplets : (0,0,0),(255,0,0), (0,255,0),(255,255,0), (0,0,255), (255,0,255), (0,255,255), (255,255,255).

**Nous allons à présent effectuer cette tâche de façon concrète afin de traiter notre photo, ainsi pour chaque pixel de coordonnées (x,y),:**

**1) nous relevons son code couleurs (r,g,b) ;**

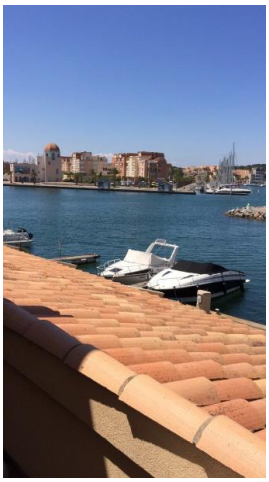
**2) nous construisons l'image d'arrivée, en affectant le code couleurs effectuant un test sur chacune des valeurs des variables r, g et b**

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def huitcouleurs(image):
        (l,c)=image.size
        imagearrivee=Image.new('RGB',(l,c))
        for x in range(l):
                for y in range(c):
                        pixel=image.getpixel((x,y))
                        if pixel[0]>127:
                                r=225
                        else:
                                r=0
                        if pixel[1]>127:
                                g=225
                        else:
                                g=0
                        if pixel[2]>127:
                                b=225
                        else:
                                b=0
                        imagearrivee.putpixel((x,y),(r,g,b))
        imagearrivee.save(''C:\Users\Nom de l'utilisateur\Bureau\Photo n°5.JPG'')
>>> huitcouleurs(im)
```

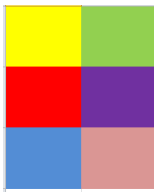
L'image ainsi construite est placée sur votre bureau à l'adresse que vous allez donner sur cette ligne.

# Fiche n°6 : Faire pivoter une image

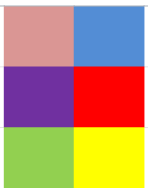


De nombreux logiciels proposent de faire pivoter une image d'un angle donné. Nous allons réaliser cette opération sur notre photographie en construisant une nouvelle image obtenue en faisant pivoter notre image de départ.

Nous allons parcourir notre image de départ et lire pour chaque pixel la valeur du triplet (r,g,b) ; il nous faudra ensuite reporter ce triplet sur le pixel qui lui correspondra dans l'image symétrique.



Afin de bien comprendre ce qu'il se passe, nous allons prendre une image plus simple constituée de 2 colonnes et 3 lignes. Ainsi l'image que nous avons placée à gauche doit devenir l'image placée à droite. Pour ce faire, il suffit de faire correspondre :



- au pixel (0,0), le pixel (1,2) ;
- au pixel (1,0), le pixel (0,2) ;
- au pixel (0,1), le pixel (1,1) ;
- au pixel (1,1), le pixel (0,1) ;
- au pixel (0,2), le pixel (1,0) ;
- au pixel (1,2), le pixel (0,0).

Autrement dit : au pixel (x,y) on associe le pixel (1-x,2-x).

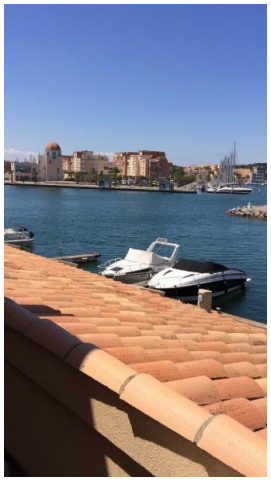
Plus généralement :

**au pixel (x,y) on associera le pixel (nombre de colonnes -1 – x , nombre de lignes -1- y)**

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def pivot(image):
        (l,c)=image.size
    imagearrivee=Image.new('RGB',(l,c))
    for x in range(l):
        for y in range(c):
            p=image.getpixel((x,y))
            imagearrivee.putpixel((l-1-x,c-1-y),p)
    imagearrivee.save("C:\Users\Nom d'utilisateur\Bureau\Photo N°6.JPG")
>>> pivot(im)
```

# Fiche n°7 : Couper une image en 4 puis former une nouvelle image



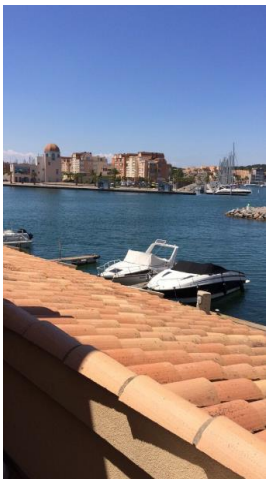
Nous proposons ici de couper notre photo en 4 puis de recoller les 4 morceaux ainsi obtenus. En d'autres termes : passer de l'image placée à gauche à celle placée à droite.

Pour cela, **nous prenons les dimensions de l'image : c colonnes et l lignes**. Puis **on s'occupe de chacun des quarts d'image, en construisant quart après quart notre image d'arrivée**.

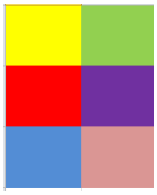
Programmons à présent le parcours de notre image et la modification de tous ses pixels.

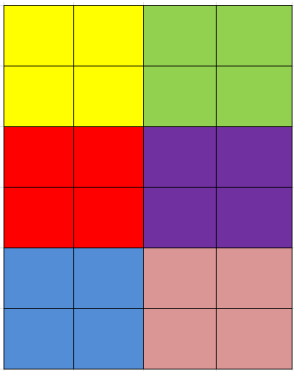
```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def couperenquatre(image):
        (c,l)=image.size
    imagearrivee=Image.new('RGB',(c,l))
    for i in range(l/2):                                    ***On va prendre le quart en haut à gauche ***
        for j in range(c/2):
            p=image.getpixel((j,i))
            imagearrivee.putpixel((c/2+j,l/2+i),p)          *** On le place en bas à droite***
    for i in range(l/2):                                    ***On va prendre le quart en haut à droite***
        for j in range(c/2,c):
            p=image.getpixel((j,i))
            imagearrivee.putpixel((j-c/2,l/2+i),p)          *** On le place en bas à gauche***
    for i in range(l/2,l):                                  *** on va prendre le quart en bas à gauche***
        for j in range(c/2):
            p=image.getpixel((j,i))
            imagearrivee.putpixel((c/2+j,i-l/2),p)          ***On le place en haut à droite***
    for i in range(l/2,l):                                  ***On va prendre le quart à droite en bas***
        for j in range(c/2,c):
            p=image.getpixel((j,i))
            imagearrivee.putpixel((j-c/2,i-l/2),p)          ***On le place en haut à gauche***
    imagearrivee.save("C:\Users\Noms d'utilisateur\Bureau\Photo N°7.JPG")
>>> couperenquatre(im)
```

# Fiche n°8 : Agrandir une image



Nous nous proposons de construire une image dont les dimensions sont deux fois celles de la photographie de départ et qui lui soit semblables.
On peut se dire a priori que cela doit être très facile. Il n'en est rien car il va s'agir de suivre un processus précis pour construire cette image ; processus que nous choisissons de présenter avec notre image très simple de 2 colonnes et 3 lignes. L'image que nous souhaitons construire une image de 4 colonnes et 6 lignes. Pour réaliser la tête en bas, nous allons parcourir notre image de départ et lire pour chaque pixel la valeur du triplet (r,g,b) ; il nous faudra ensuite reporter ce triplet sur 4 pixels qui lui correspondront dans l'image que nous allons construire. Nous allons ainsi construire la grande image à partir de la petite. Pour ce faire, il suffit de faire correspondre :





- au pixel (0,0), le pixel **(0,0)** puis (1,0),(0,1),(1,1);
- au pixel (1,0), le pixel **(2,0)** puis (3,0),(2,1),(3,1) ;
- au pixel (0,1), le pixel **(0,2)** puis (1,2),(0,3),(1,3) ;
- au pixel (1,1), le pixel **(2,2)** puis (3,2),(2,3),(3,3) ;
- au pixel (0,2), le pixel **(0,4)** puis (1,4),(0,5),(1,5) ;
- au pixel (1,2), le pixel **(2,4)** puis (3,4),(2,5),(3,5).

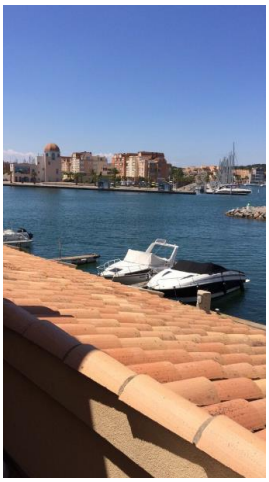
Plus généralement :

**au pixel (x,y) on associera le pixel (2x , 2y) puis (2x+1,2y), (2x, 2y+1), (2x+1,2y+1)**.

Programmons à présent le parcours de notre image et la modification de tous ses pixels.

```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def agrandissement(image):
        (c,l)=image.size
    imagearrivee=Image.new('RGB',(2*c,2*l))
    for x in range(c):
        for y in range(l):
            p=image.getpixel((x,y))
            imagearrivee.putpixel((2*x,2*y),p)
                imagearrivee.putpixel((2*x+1,2*y),p)
                imagearrivee.putpixel((2*x,2*y+1),p)
              imagearrivee.putpixel((2*x+1,2*y+1),p)
    imagearrivee.save('C:\Users\Nom d'utilisateur\Bureau\Photo n°8.JPG')
>>> agrandissement(im)
```

# Fiche n°9 : Conservation uniquement des contours



La transformation que nous proposons dans cette fiche est certainement la plus spectaculaire : Imaginez-vous être devant ce paysage et à avoir à le reproduire rapidement au crayon papier sur une feuille. Pour ce faire, vous allez conserver uniquement les contours de tous les éléments présents sur cette image, autrement dit vous allez identifier tous les points qui ressortent par rapport à leurs proches voisins. Nous allons faire produire cette opération de façon automatique en retenant pour chaque pixel de la photo ceux qui ont une couleur « très » différente de celle de leurs voisins. Soit un pixel de triplet de couleurs (r1,g1,b1) et un autre pixel de triplet de couleurs (r2,g2,b2), nous pouvons avoir une idée de la différence entre les couleurs de ces deux pixels en calculant $(r1-r2)^2+(g1-g2)^2+(b1-b2)^2$ que nous noterons quantite((r1,g1,b1),(r2,g2,b2)). Nous conservons à l'idée que :

« *Plus la valeur trouvée est grande, plus les couleurs des deux pixels sont différentes* » et « *Plus la valeur trouvée est petite, plus les couleurs des deux pixels sont proches l'une de l'autre* ». Pour chaque pixel (x,y), nous allons calculer les huit distances entre son triplet de couleurs et celui de chacun de ses voisins. Si la somme des huit quantités obtenues est inférieure à 2400, nous affectons à ce pixel, la couleur blanche (ce point n'a en effet rien de remarquable) ; si tel n'est pas le cas, nous affectons à ce pixel, la couleur noire (ce point est certainement sur le bord d'un objet de l'image).

```
>>> from PIL import Image
>>> a="C:\Users\Nom d'utilisateur\Bureau\Photo de départ.JPG"
>>> im=Image.open(a)
>>> def quantite(tri1,tri2):
        d=(tri1[0]-tri2[0])**2+(tri1[1]-tri2[1])**2+(tri1[2]-tri2[2])**2
        return(d)
>>> def contours(image):
    (c,l)=image.size
        imagearrivee=Image.new('RGB',(c,l))
        for x in range(1,c-1):
            for y in range(1,l-1):
                p=image.getpixel((x,y))
            p1=image.getpixel((x-1,y-1))
            q1=quantite(p,p1)
            p2=image.getpixel((x-1,y))
            q2=quantite(p,p2)
            p3=image.getpixel((x-1,y+1))
            q3=quantite(p,p3)
            p4=image.getpixel((x,y-1))
            q4=quantite (p,p4)
            p5=image.getpixel((x,y+1))
            q5=quantite (p,p5)
            p6=image.getpixel((x+1,y-1))
            q6=quantite(p,p6)
            p7=image.getpixel((x+1,y))
            q7=quantite(p,p7)
            p8=image.getpixel((x+1,y+1))
            q8=quantite (p,p8)
            if q1+q2+q3+q4+q5+q6+q7+q8<2400:
                imagearrivee.putpixel((x,y),(255,255,255))
            else :
                imagearrivee.putpixel((x,y),(0,0,0))
    imagearrivee.save("C:\Users\Nom d'utilisateur\Bureau\Photo n°9.JPG")
>>> contours(im)
```