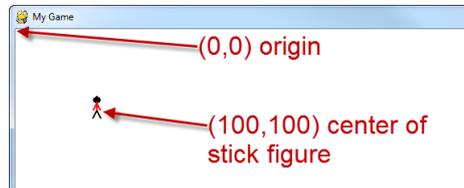


## 1 Rappels sur la représentation bitmap d'une image

Vous avez appris dans la séquence précédente qu'en informatique la représentation bitmap d'une image est une matrice (ou tableau à deux dimensions) de pixels. Dans cette séquence vous ne manipulerez que des images en niveaux de gris dont les valeurs des pixels sont codées sur 1 octet et sont des entiers compris entre 0 et 255.

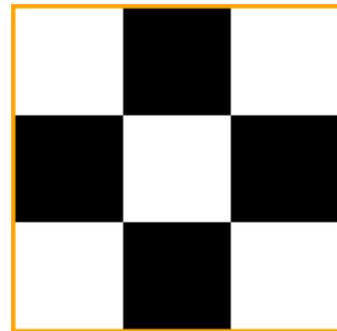
De plus les pixels sont habituellement repérés depuis une origine (0,0) située dans le coin supérieur gauche de l'image, l'axe des abscisses horizontal orienté positivement de gauche à droite, l'axe des ordonnées vertical orienté positivement de haut en bas.



Ainsi l'image ci-contre (on a rajouté un cadre orange pour

la délimiter) dont la matrice de pixels est  $\begin{pmatrix} 255 & 0 & 255 \\ 0 & 255 & 0 \\ 255 & 0 & 255 \end{pmatrix}$

est implémentée en Python sous la forme d'une liste de listes `[[255,0,255], [0,255,0], [255,0,255]]` et elle est enregistrée sur le disque par l'appel suivant (pour mieux visualiser on a choisi de représenter chacun de ses pixels par 50 pixels de l'écran) :



```
1 enregistrer_image
  ([[255,0,255], [0,255,0], [255,0,255]]),
  nom='exemple.png',n=50)
```

## 2 Traitement d'image : applications de filtres divers

### Exercice 1

#### Négatif d'une image

Pour produire une image but qui est le négatif d'une image source en niveaux de gris il suffit de créer une matrice de pixels but de mêmes dimensions que la matrice source mais remplie de 0 puis de la parcourir ligne par ligne et colonne par colonne pour appliquer à tous ses pixels la relation  $\text{pixel}_{but} = 255 - \text{pixel}_{source}$ .

On considère qu'on a déjà ouvert puis exécuté avec Pyzo le script `ISN-TP050115-Eleve.py` Pyzo (avec *Exécuter le script*).

1. Compléter le code de la fonction `palette_gris` qui prend comme paramètre un nom de fichier, enregistre sur le disque une image bitmap de dimensions  $16 \times 16$  dont les 256 pixels représentent  $2^8 = 256$  niveaux de gris (dans l'ordre croissant ligne par ligne à partir du pixel origine) et retourne sa matrice de pixels.

Sauvegarder/Exécuter le fichier puis tester les commandes suivantes.

```
1 In [7]: p = palette_gris('niveaux-gris.png')
2
3 In [8]: p
4 Out [8]:
5 array([[ 0.,  1.,  ..., 14., 15.],
6        .....,
7        [240., 241.,  ..., 254., 255.]])
```

Parmi les deux images ci-dessous, quelle est celle dont la matrice de pixels est p ?

2. Appliquer ensuite à la matrice de pixels p la fonction `negatif()` dont on donne le code ci-dessous.

Enregistrer la matrice de pixels png obtenue sous la forme d'un fichier `niveaux-gris-negatif.png` à l'aide de l'appel de fonction `enregistrer_image(png, 'niveaux-gris-negatif.png')`.

Expliquer comment les deux boucles `for` imbriquées permettent de parcourir toute la matrice de pixels `newtab` et de leur affecter les négatifs des pixels de la matrice source `tab`.

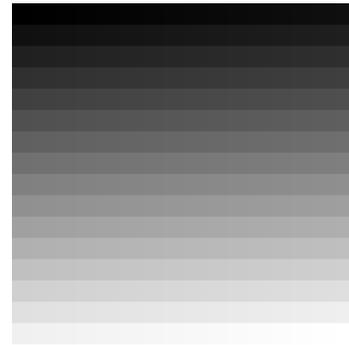
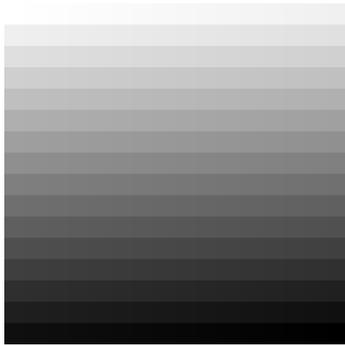
---

```

1 def negatif(tab):
2     """Retourne une nouvelle matrice de pixels en appliquant à chaque pixel de
3     tab la fonction  $x \rightarrow 255 - x$  qui transforme le blanc en noir et le noir
4     en blanc. Les pixels de tab doivent être codés en niveaux de gris de 0 à 255.
5     """
6     L, H = dimensions(tab)
7     #crée une matrice de H lignes et L colonnes remplie de 0
8     newtab = np.zeros((H, L))
9     for line in range(H):
10        for col in range(L):
11            newtab[line][col] = 255 - tab[line][col]
12    return newtab

```

---



### Exercice 2

*Des filtres pour éclaircir, assombrir, contraster*

Si on veut appliquer la même fonction numérique  $f$  (appelée fonction de filtre) à tous les pixels de la matrice de pixels d'une image, il suffit de créer une matrice de pixels but de mêmes dimensions que la matrice source mais remplie de 0 puis de la parcourir ligne par ligne et colonne par colonne pour appliquer à tous ses pixels la relation  $\text{pixel}_{but} = f(\text{pixel}_{source})$ . C'est donc le même algorithme que pour la fonction `negatif`.

Néanmoins la fonction de filtre  $f$  doit vérifier la contrainte  $f([0; 255]) \subset [0; 255]$  et même si possible  $f([0; 255]) = [0; 255]$  si on ne veut pas restreindre le spectre des niveaux de gris...

De plus, il est plus raisonnable que la formule de la fonction  $f$  ne dépende pas de l'échelle choisie pour les niveaux de gris (ici de 0 à 255). Pour normaliser, on va considérer que  $f$  est une fonction de  $[0; 1]$  dans  $[0; 1]$ . On ramène d'abord la valeur du pixel de la source entre 0 et 1 en divisant par 255, on applique la fonction de filtre  $f$  puis on retourne dans la plage  $[0; 255]$  en multipliant par 255 et enfin on convertit en entier :

---

```

1 normalsource = pixelsource / 255
2 normalbut = f(normalsource)
3 pixelbut = int( 255*normalbut )

```

---

1. Parmi les fonctions de filtre représentées sur le graphique ci-dessous, quelles sont celles qui vont :

- Eclaircir l'image ?
- Produire le négatif de l'image ?
- Assombrir l'image ?
- Augmenter le contraste ?

2. Sur le modèle de la fonction `negatif` définie dans l'exercice 1, écrire des fonctions Python `assombrir(tab)`, `eclaircir(tab)`, `contraster(tab)`, qui appliquent le filtre approprié à une matrice de pixels `tab` et retournent une nouvelle matrice de pixels.

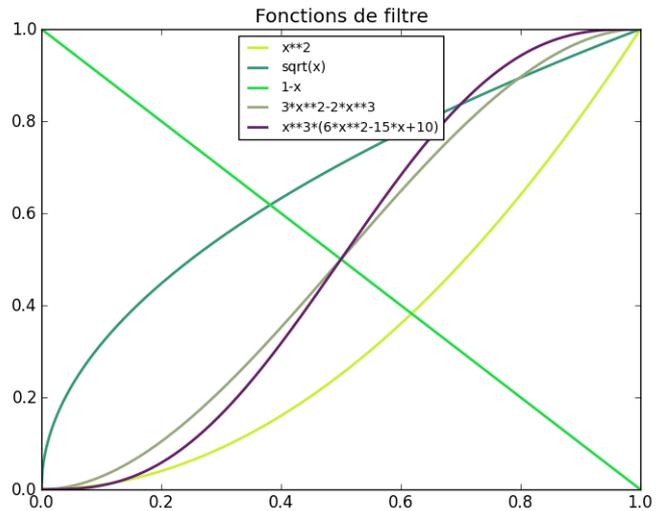
Tester ces fonctions dans le shell sur la matrice de pixels de l'image `lenagray.png`.

Pensez à transmettre ces nouvelles définitions à l'interpréteur en cliquant sur *Exécuter comme fichier*, puis avec des appels de fonctions `enregistrer_image(newtab, 'newimage.png', n=1)`, créer les fichiers images adéquats.

```

1 In [1]: tablena = ouvrir_image(
      | 'lenagray.png')
2
3 In [2]: tablenacontrast =
      | contraster(tablena)
4
5 In [3]: enregistrer_image(
      | tablenacontrast, 'lena+
      | contrast.png', n=1)

```



3. Ecrire une fonction `filtre` générique qui applique une fonction filtre à tous les pixels de la représentation matricielle d'une image bitmap.

**Exercice 3**

*Seuiller pour extraire les pixels sombres ou clairs*

Si on veut conserver uniquement les pixels sombres (ou seulement les pixels clairs), il suffit de reprendre le même algorithme que pour la fonction `negatif` et de « blanchir » (ou au contraire « noircir ») les pixels dont la valeur dépasse un certain seuil :

```

1 if pixelsource > seuil:
2     pixelbut = 255 #le nouveau pixel est blanc
3 else:
4     pixelbut = pixelsource

```

1. Ecrire le code d'une fonction `seuil(tab, s)` qui retourne une nouvelle matrice de pixels en appliquant à chaque pixel de la matrice source `tab` la fonction de seuil :  $f(x) = x$  si  $x \leq s$  et  $f(x) = 255$  sinon.

2. Cliquer sur *Exécuter le fichier* puis tester les commandes suivantes dans le shell :

```

1 In [4]: t = ouvrir_image('pendulegray.png')
2
3 In [5]: tseuil = seuil(t, 127)
4
5 In [6]: enregistrer_image(tseuil, 'penduleseuil127.png', n=1)

```

Vous devriez obtenir l'image de droite après seuillage (à 50 % du niveau de gris maximal) :

