TP 1 : Rappel sur les librairies Numpy et matplotlib

1 - Introduction

L'objectif de ce TP est de faire des rappels sur les librairies numpy et matplotlib. Il sera à réaliser en python 3. Les librairies utilisées sont installées sur les machines de l'université, vous pouvez néanmoins les installer sur vos propres machines à l'aide de l'utilitaire pip présent par défaut avec python.

N'hésitez pas à regarder régulièrement la documentation de ces librairies, des exemples d'utilisation accompagnent généralement l'explication de chaque fonction.

- Python 3: https://docs.python.org/3/
- Numpy: https://docs.scipy.org/doc/numpy/reference/
- Scipy: https://docs.scipy.org/doc/scipy/reference/
- Matplotilb: https://matplotlib.org/contents.html

À part si cela est précisé, vous ne devez pas utiliser directement de boucle (f or , $whi \leq$) ou de branchement conditionnel (if) durant ce TP..

In [1]:
```
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt
```

2 - Vecteurs

2.1 Création de vecteurs

Créez un vecteur de taille 5 contenant que des zéros. Affichez-le et affichez sa taille.

In [2]:
```
vecteur_zero = np.zeros(5)
print("Affiche  du vecteur: ", vecteur_zero)
print("Dimension du vecteur: {} . Nombre d'élément du vecteur : {} ".forma
```

```
Affiche  du vecteur:  [0. 0. 0. 0. 0.]
Dimension du vecteur: (5,) . Nombre d'élément du vecteur : 5
```

Créez un vecteur contenant les nombres compris entre 3 et 12 (exclus) avec un pas de 0.5

In [3]:
```
np.arange(3, 12, 0.5)
```

Out[3]:
```
array([ 3. ,  3.5,  4. ,  4.5,  5. ,  5.5,  6. ,  6.5,  7. ,  7.5,  8. ,
        8.5,  9. ,  9.5, 10. , 10.5, 11. , 11.5])
```

Créez un vecteur contenant les carrés des entiers compris entre -5 et 5 (exclus).

In [4]:
```
np.arange(-5, 5, 1) ** 2
```

Out[4]:
```
array([25, 16,  9,  4,  1,  0,  1,  4,  9, 16])
```

Créez un vecteur contenant toutes les puissances de deux des entiers compris entre 1 et 65536 ($= 2^{16}$) inclus. Le résultat attendu ressemble à :

<p style="text-align:center";> [1, 2, 4, 8, 16,..., 32768, 65536] </p>

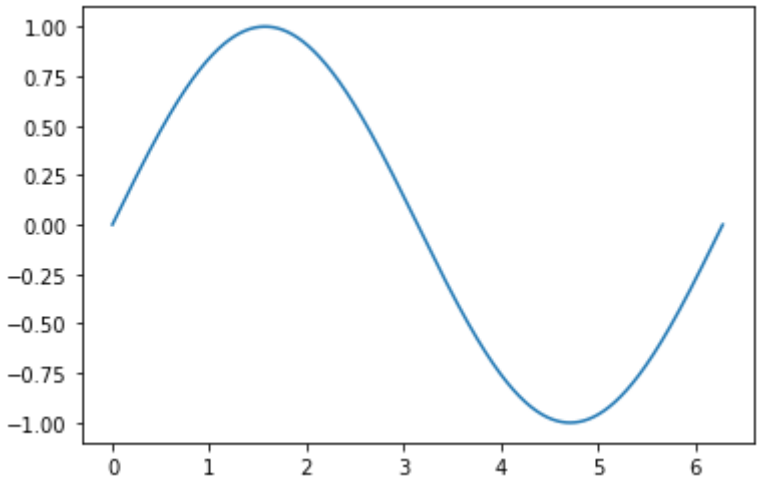
In [5]:
```
2 ** np.arange(0, 17, 1)
```

Out[5]:
```
array([    1,     2,     4,     8,    16,    32,    64,   128,   256,
         512,  1024,  2048,  4096,  8192, 16384, 32768, 65536])
```

2.2 Visualisation de fonctions

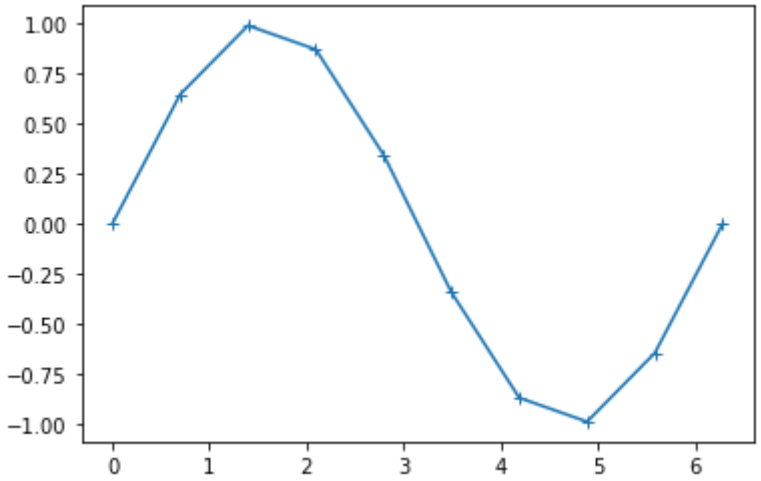
Tracez avec matplotlib la fonction *sinus* entre 0 et 2*np.pi en calculant un point tous les 1e-3. Vérifiez que les valeurs en abscisse et en ordonnée sont correctes.

In [6]:
```
x = np.arange(0, 2 * np.pi, 1e-3)

plt.plot(x, np.sin(x))
plt.show()
```



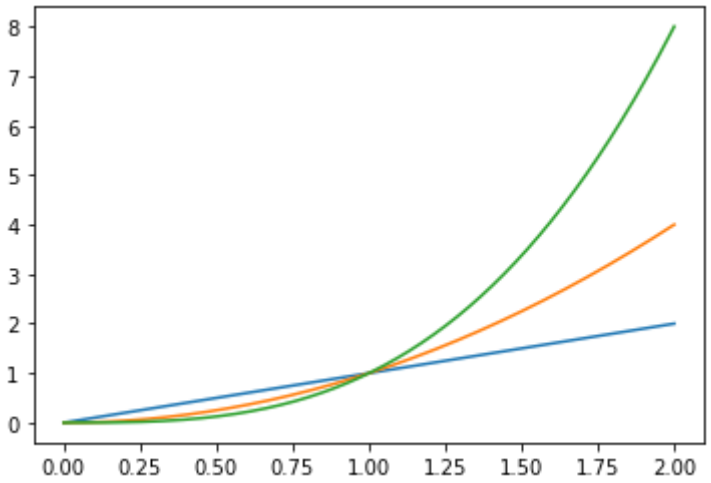
Faites l'affichage de la précédente question, mais avec uniquement 10 points. Vous pouvez visualiser ces points en ajoutant '+-' aux arguments de la fonction `plot` .

In [7]:
```
x1 = np.linspace(0, 2 * np.pi, 10)

plt.plot(x1, np.sin(x1), "+-")
plt.show()
```



Tracez sur la même figure les fonctions x,x^2,x^3 entre 0 et 2. Choisissez un écart entre chaque point suffisamment petit pour que les courbes paraissent lisses.

In [8]:
```
x2 = np.arange(0, 2, 1e-6)

plt.plot(x2, x2)
plt.plot(x2, x2 ** 2)
plt.plot(x2, x2 ** 3)

plt.show()
```



3 - Comparaison python classique et fonction numpy

Le code suivant permet de mesurer le temps d exéuction de plusieurs instructions.

In [9]:
```
%%time
1+1
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.72 µs
```

Out[9]: 2

Calculez la somme des entiers entre 0 et 100 millions avec une **boucle for** en python sans utiliser de librairie. Vous mesurez le temps d'exéuction de votre code.

In [10]:
```
%%time
somme = 0
for nombre in range(0, 10 ** 8 + 1):
    somme += nombre
print(somme)
```

```
5000000050000000
CPU times: user 10.7 s, sys: 6.31 ms, total: 10.7 s
Wall time: 10.7 s
```

Créez un vecteur contenant tous les entiers entre 0 et 100 millions puis calculez la somme des valeurs de ce vecteur en utilisant numpy. **Aucun for ou while ne devra être utilisé.**

Comparez les vitesses d'exécution des deux scripts.

In [11]:
```
%%time
np.sum(np.arange(0, 10 ** 8 +1))
```

```
CPU times: user 130 ms, sys: 120 ms, total: 250 ms
Wall time: 248 ms
```

Out[11]:
```
5000000050000000
```

Question bonus: Calculez cette somme en utilisant vos connaissances sur la somme des suites arithmétiques. Retrouvez-vous le même résultat ?

In [12]:
```
%%time
n = 0
p = 10 ** 8
(p- n +1) * (n + p) / 2
```

```
CPU times: user 25 µs, sys: 1 µs, total: 26 µs
Wall time: 30 µs
```

Out[12]:
```
5000000050000000.0
```

4 - Les matrices

4.1 - Création de matrices

Créez une matrice de taille (12,4) contenant que des zéros. Affichez la et affichez sa taille.

In [13]:
```
matrice_zero = np.zeros((12, 4))
print("Matrice: \n {} \n Dimension de la matrice: {}".format(matrice_zero,
```

```
Matrice:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
 Dimension de la matrice: (12, 4)
```

Créez une matrice identitée de dimension (16,16). Affichez la et affichez sa taille.

In [14]:
```
matrice_identite = np.identity(16)
print("Matrice: \n {} \n Dimension de la matrice: {}".format(matrice_identi
```

```
Matrice:
 [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
 Dimension de la matrice: (16, 16)
```

Créez et affichez la matrice $\begin{bmatrix} 1 & 31 \\ 51 & 12 \end{bmatrix}$

In [15]:
```
matrice = np.array([[1, 31], [51, 12]])
print(matrice)
```

```
[[ 1 31]
 [51 12]]
```

Affichez uniquement la ligne 0 de la matrice précédente. Affichez la colonne 1.

In [16]:
```
print("Ligne 0: {} \n Colonne 1: {}".format(matrice[0], matrice[:, 1]))
```

```
Ligne 0: [ 1 31]
 Colonne 1: [31 12]
```

5. Application à du traitement d'image simple

Dans cette partie, nous mettrons en application des opérations sur les matrices pour effectuer des traitements d'images simples.

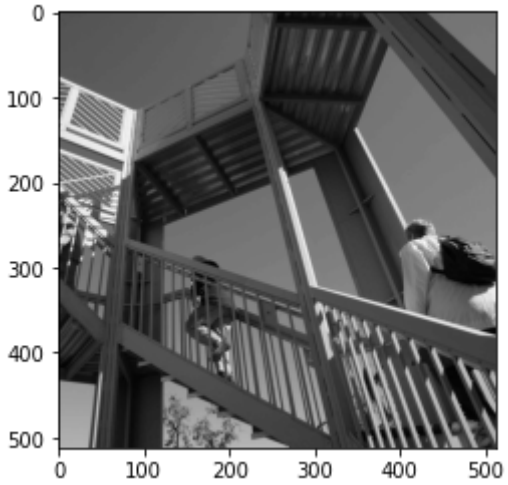
5.1 Lecture et manipulation d'une image

Il existe dans la librairie scipy deux images de test que vous pouvez utiliser. L'une d'elles peut être récupéré par le code suivant:

In [17]:
```
import scipy as sc
import scipy.misc
im = sc.misc.ascent()
```

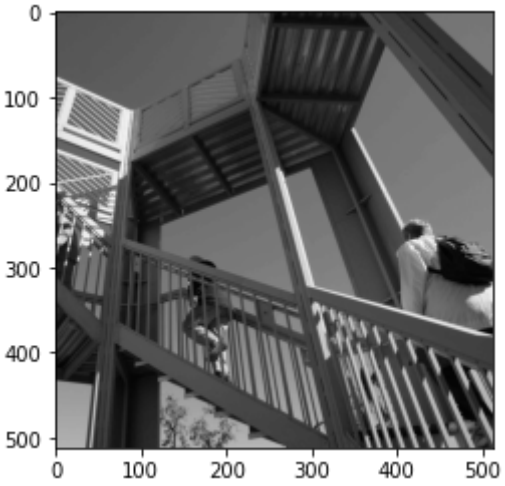
À l'aide de la fonction plt.imshow de matplotlib affichez l'image précédente. L'option `cmap='gray'` permet de préciser que l'image est en noir et blanc. Les options `vmin = 0` et `vmax = 255` permet de préciser que les pixels sont définis par des valeurs allant de 0 (noir) à 255 (blanc). N'oubliez pour la fonction plt.show() après plt.imshow .

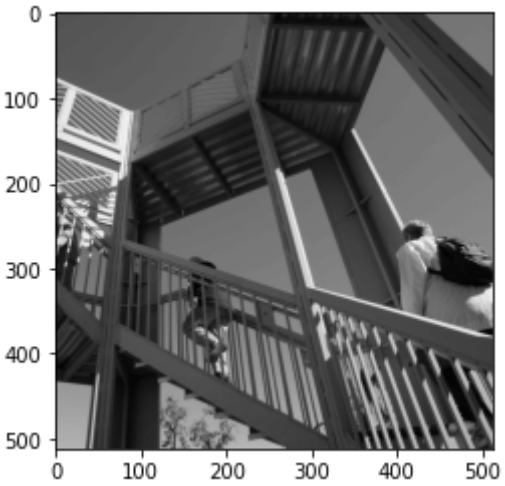
In [18]:
```
plt.imshow(im, cmap='gray', vmin = 0, vmax = 255)
plt.show()
```



La fonction plt.show permet d'exécuter toutes les instructions d'affichage demandées. Il est parfois nécessaire d'afficher plusieurs fenêtres d'affichage. Pour cela vous pouvez appeler la fonction `plt.figure(n)` avec n le numéro de la fenêtre à gérer. Lorsque plt.show sera appelé, toutes les fenêtres seront affichées. Affichez dans deux fenêtres en parallèle l'image précédente. **Vous utiliserez qu'une seule fois la fonction plt.show().**

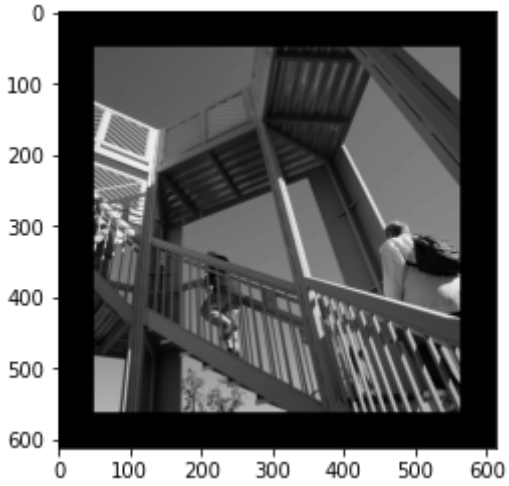
In [19]:
```
plt.figure(1)
plt.imshow(im, cmap='gray', vmin = 0, vmax = 255)

plt.figure(2)
plt.imshow(im, cmap='gray', vmin = 0, vmax = 255)

plt.show()
```





Nous allons ajouter un cadre noir à notre image. Pour cela, construisez un tableau numpy contenant uniquement des zéros dont les dimensions sont celles de l'image augmentées de 100 en ligne et colonne. Recopiez ensuite l'image à partir de la ligne 50, colonne 50 (sans utiliser de for ou de while).

In [20]:
```
tableau_zero = np.zeros((np.shape(im)[0] + 100, np.shape(im)[1] + 100))


tableau_zero[50:-50, 50:-50] = im

plt.imshow(tableau_zero, cmap='gray', vmin = 0, vmax = 255)

plt.show()
```



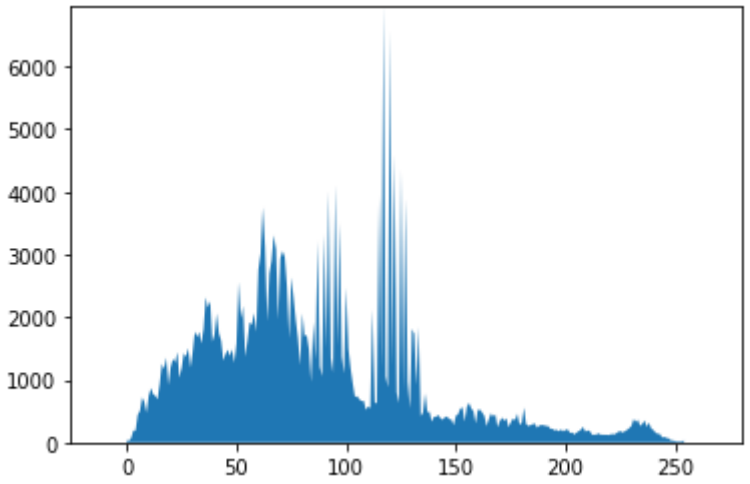
5.2- L'histogramme des niveaux de gris

En imagerie numérique, l'histogramme représente la distribution des intensités (ou des couleurs) de l'image. C'est une représentation graphique donnant pour chaque niveau de gris (ou couleur) le nombre de points de l'image ayant ce niveau de gris (couleur).

L'histogramme permet ainsi de visualiser la répartition des différents niveaux de gris de l'image. Pour chaque niveau de gris entre 0 (noir) jusqu'à 255 (blanc) en abscisse, vous avez le nombre de pixels de l'image ayant cette valeur en ordonnée.

Nous allons maintenant afficher l'histogramme des niveaux de gris d'une image, grace à la fonction suivante:

In [21]:
```
def hist(im):
  h,x = np.histogram(im,bins=255,range=(0,255))
  plt.fill_between(x[:-1],0,h)
  plt.axis((-1/10*(x[-1]-x[0]),x[-1]+1/10*(x[-1]-x[0]),0,np.max(h)))

hist(im)
plt.show()
```



Expliquez à quoi servent les fonctions `np.histogram` , `plt.fill_between` et `plt.axis` .

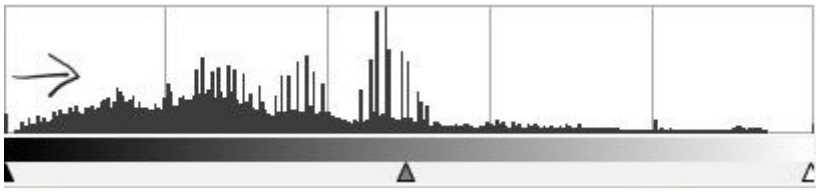
à compléter

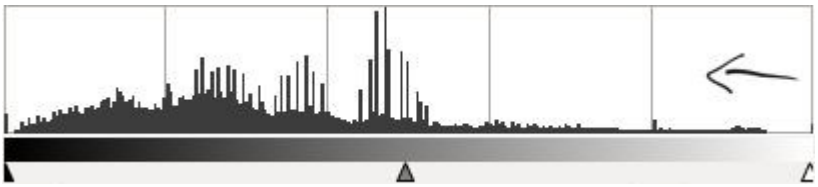
Que pouvez vous dire sur l'image en regardant son histogramme ?

à compléter

5.3 - Éclaircir et assombrir une image

Nous allons dans un premier temps chercher à éclaircir l'image. Pour cela, nous allons augmenter l'ensemble des valeurs des niveaux de gris des pixels. Il faut néanmoins faire en sorte que les valeurs restent entre 0 et 255.

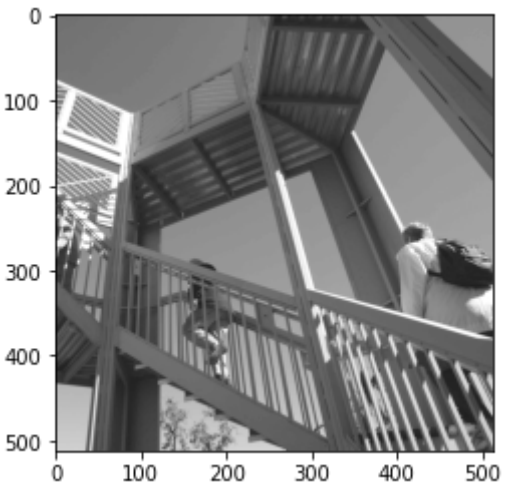
Éclaircir l'image: 

Assombrir l'image: 

Soit n notre paramètre de réglage de l'effet. n Varie entre 0 (aucun effet) et 255 (effet maximal, l'image devient entièrement blanche). Réalisez un programme qui sature (mettre à 255) les pixels dont la valeur de départ est supérieure à $255 - n$ et augmente de n, les autres pixels.

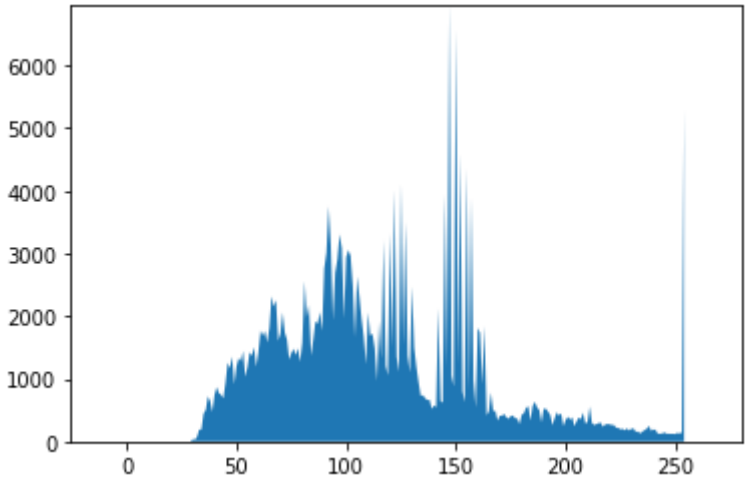
In [22]:
```
def eclaicir(im,n):
    # a compléter
    im += n
    im[im > 255] = 255
    return im
```

In [23]:
```
n = 30
im2 = eclaicir(im,n)
plt.imshow(im2,cmap='gray',vmin=0,vmax=255)
plt.show()
```



Affichez l'histogramme avant et après l'éclaircissement et regarder les conséquences du traitement pour différentes valeurs de n.

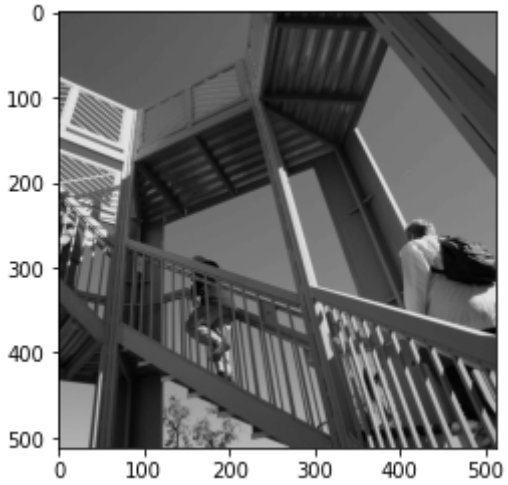
In [24]:
```
hist(im2)
plt.show()
```

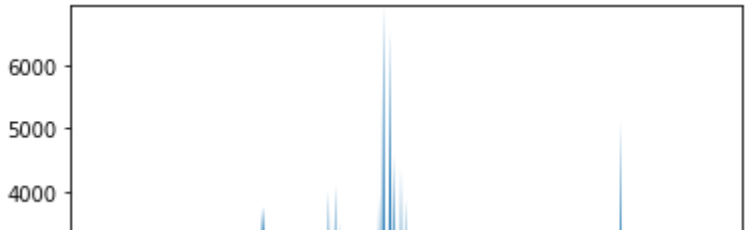


Réalisez maintenant l'effet inverse. Les valeurs inférieures à n doivent être ramenées à 0 et les autres doivent être réduites de n.

In [25]:
```
def assombrir(im,n):
    # a compléter
    im -= n
    im[im < 0] = 0
    return im
```

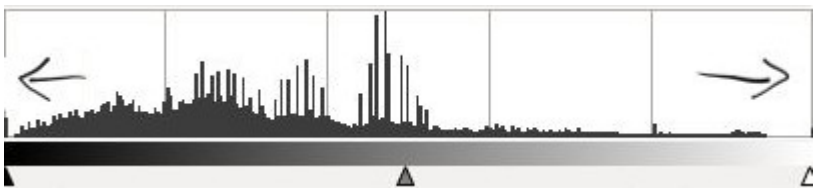
In [26]:
```
n = 30
im2 = assombrir(im,n)
plt.imshow(im2,cmap='gray',vmin=0,vmax=255)
plt.figure()
hist(im2)
plt.show()
```





5.4 Étirement d'histogramme (comparaison numpy/python seul)

L'objectif de cet exercice est d'étirer l'histogramme d'une image, en appliquant aux niveaux de gris des pixels une transformation affine telle que le plus petit niveau de gris aura, pour valeur 0 et le plus grand 255, après transformation des niveaux de gris.

Étirement d'histogramme: 

Nous utiliserons pour cela deux méthodes et comparerons l'efficacité de ces deux méthodes.

Mais auparavant, vous commencerez par charger l'image face_gris.png (fournie sur ecampus) en mémoire dans la variable `im` et vous l'afficherez à l'écran. La lecture d'une image sur disque peut se faire au moyen de la fonction `plt.imread` de la librairie *matplotlib*. L'affichage peut se faire à l'aide de la fonction `plt.imshow(im,cmap='gray',vmin=0,vmax=255)` de matplotlib. L'option `cmap='gray'` permet de préciser que l'image est en noir et blanc. `vmin = 0` et `vmax = 255` permet de préciser que les pixels sont définis par des valeurs allant de 0 (noir) à 255 (blanc).

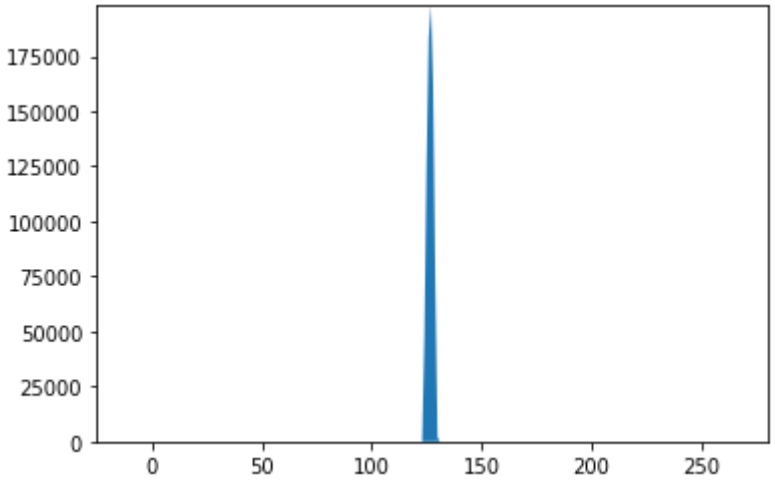
In [27]:
```
im = (plt.imread('face_gris.png')*256).astype('int') # lit l'image et ramè

plt.title('Image face_gris.png')
plt.imshow(im,vmin=0,vmax=255,cmap='gray')
plt.show()
```

Première méthode: utilisation de boucles

Visualisez l'histogramme de cette image. Que pouvez-vous dire sur la répartition des

différents niveaux de gris ?

In [28]:
```
hist(im)
plt.show()
```



À l'aide d'instruction `if` et `for` (en utilisant que du code python 3 de base) écrivez une fonction qui trouve la valeur minimale et maximale des pixels de l'image. Nous les noterons p_{min} et p_{max} dans la suite du sujet.

In [29]:
```
def min_max_image(im):
    # a faire
    pmin = 255
    pmax = 0
    for i in range(len(im)):
        for j in range(len(im[i])):
            if im[i, j] > pmax:
                pmax = im[i, j]
            elif im[i,j] < pmin:
                pmin = im[i, j]
    return pmin, pmax
```

In [30]:
```
p_min,p_max = min_max_image(im)
print('P_min=',p_min,'P_max=',p_max)
```

```
P_min= 124 P_max= 130
```

À l'aide d'instructions `if` et `for` faites une fonction qui crée une nouvelle image dont les pixels correspondent à l'équation $255\frac{p_{ij}-p_{min}}{p_{max}-p_{min}}$ où p_{ij} est le pixel de la ligne i et de la colonne j. Cette opération consiste à ramener les valeurs effectives des pixels entre 0 et 255 et forcer ainsi l'utilisation de toute la plage de valeur possible.

Pour cette question vous calculerez les nouveaux pixels, pixel par pixel, à l'aide de l'image d'origine. Pour simplifier l'initialisation, vous pouvez commencer avec une image noir avec l'instruction: im2 = np.zeros(im.shape).

In [31]:
```
def rehaussementHistogramme(im,p_min,p_max):
    # a faire
    new_im = im
    denominateur = p_max - p_min
    for i in range(len(new_im)):
        for j in range(len(new_im[i])):
            new_im[i, j] = 255 * (im[i, j] - p_min) / denominateur
    return new_im
```

In [32]:
```
%%time
# remarquez bien le temps d'execution de cette approche.
p_min,p_max = min_max_image(im)
im2 = rehaussementHistogramme(im,p_min,p_max)
```

```
CPU times: user 1.21 s, sys: 0 ns, total: 1.21 s
Wall time: 1.21 s
```

In [33]:
```
plt.imshow(im2,cmap='gray',vmin=0,vmax=255)
plt.show()
```

5.5 Seconde méthode : utilisation de numpy

Utilisez maintenant les fonctions numpy: `np.min` et `np.max` pour calculer la valeur minimale et maximale des pixels de l'image. Effectuez le même traitement que pour la méthode précédente, mais cette fois-ci sans boucle, en traitement l'ensemble de l'image en même temps grâce aux opérations mathématiques sur les tableaux numpy.

In [34]:
```
%%time
im3 = 255 * (im - np.min(im)) / (np.max(im) - np.min(im))
```

```
CPU times: user 8.33 ms, sys: 4.05 ms, total: 12.4 ms
Wall time: 9.81 ms
```

Comparez les images obtenues avec les deux approches. Que pouvez-vous dire des temps d'exécutions dans les deux cas ?

In [35]:
```
plt.imshow(im3,cmap='gray',vmin=0,vmax=255)
plt.show()
```

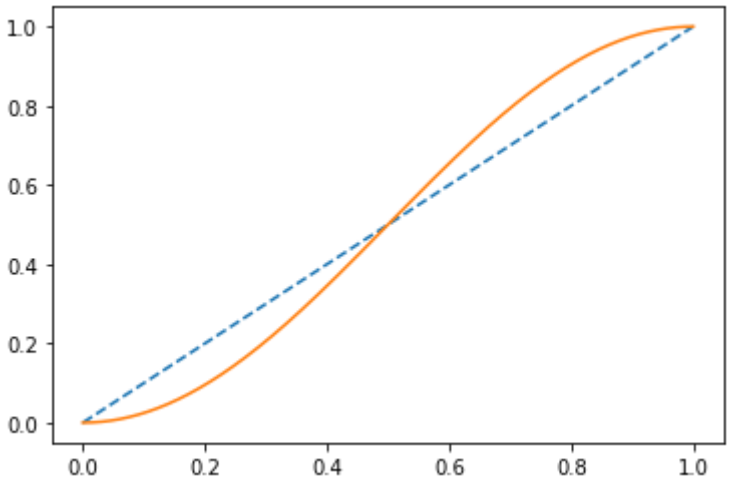
5.6 - Rehaussement de contraste

Pour rehausser les contrastes d'une image, il faut accentuer l'écart entre les tons clairs et les tons foncés. Pour cela, il faut assombrir davantage les pixels sombres et éclaircir les pixels clairs.

Afin d'obtenir ce résultat, nous allons appliquer aux pixels de l'image une fonction qui augmente les valeurs déjà fortes et diminue les valeurs plus faibles. Généralement on prend, pour faire cette opération, une courbe en forme de "S".

Affichez sur la même figure les fonctions $y = x$ et $y = 0.5\cos((1-x)\pi) + 0.5$ entre 0 et 1. Vous utiliserez `np.pi` pour avoir la valeur de π.

In [36]:
```
x3 = np.arange(0, 1, 1e-6)
y3 = 0.5 * np.cos((1-x3) * np.pi) + 0.5
plt.plot(x3, x3, "--")
plt.plot(x3, y3)

plt.show()
```



Reprenez la première images vue dans ce TP et divisez toutes ces valeurs par 255.0 pour avoir des valeurs comprises entre 0 et 1.

In [37]:
```
im = sc.misc.ascent()
im4 = im / 255.0
```

Prenez l'image de la question précédente et appliquez lui la courbe en S vu dans la première question de cette partie.

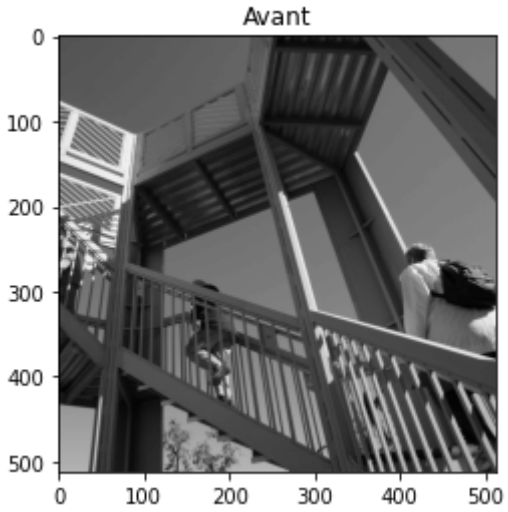
In [38]:
```
im4 = 0.5 * np.cos((1-im4) * np.pi) + 0.5
```

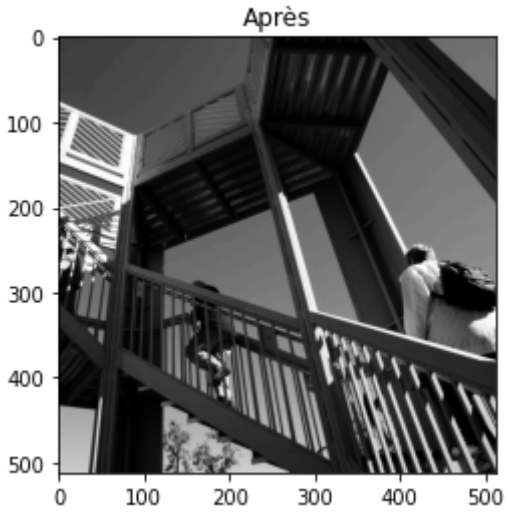
Repassez toutes les valeurs entre 0 et 255 en multipliant l'image de la question précédente par 255.

In [39]:
```
im4 *= 255
```

Visualisez l'effet obtenu sur l'image et sur son histogramme.

In [40]:
```
plt.figure(1)
plt.imshow(im, cmap='gray', vmin = 0, vmax = 255)
plt.title("Avant")

plt.figure(2)
plt.imshow(im4, cmap='gray', vmin = 0, vmax = 255)
plt.title("Après")

plt.show()
```

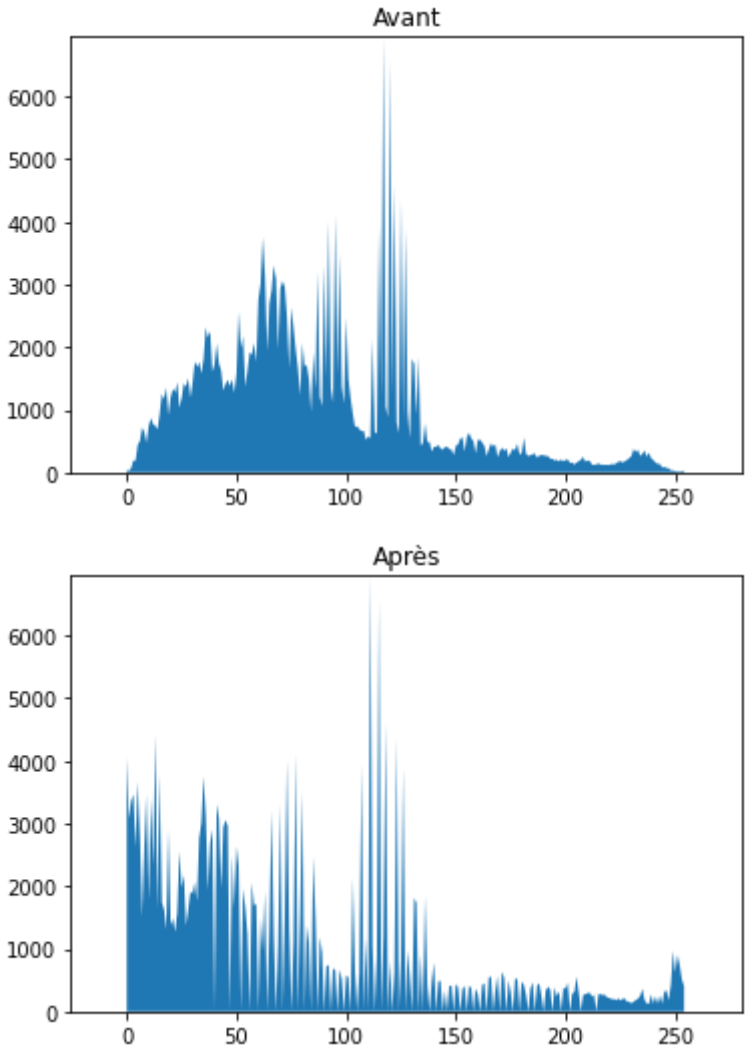




Comparez les histogrammes avant et après. Que pouvez-vous dire sur l'effet sur les tons clairs, les tons foncés et les tons moyens ?

In [41]:

```
plt.figure(1)
hist(im)
plt.title("Avant")

plt.figure(2)
hist(im4)
plt.title("Après")

plt.show()
```



In [ ]: