Funciones de Matlab para manejar datos de imágenes:

Las imágenes se almacenan en Matlab como matrices de:

- 3 dimensiones (alto x ancho x 3) para imágenes de color real. La última dimensión corresponde a los tres planos de color (R, G y B) de la imagen.
- 2 dimensiones (alto x ancho) con un solo valor por píxel para imágenes en grises. En ese caso debe asignarse una paleta de color (o niveles de gris) a la figura (usando colormap) para visualizarla correctamente.

Algunas funciones relacionadas con la lectura/visualización de imágenes:

- **imformats:** lista los formatos gráficos soportados por imread/imwrite
- **imread:** lee de un fichero gráfico y almacena la imagen en una matriz de MATLAB de tamaño Alto x Ancho x 3 (color) o Alto x Ancho (monocroma).
- **imwrite:** escribe una matriz (imagen) en MATLAB en un fichero (a elegir entre varios formatos) para poder verla con otras aplicaciones.
- **imshow**(im) o **image**(im) visualiza una imagen im en una figura (ventana).
- **colormap**(paleta):establece paleta de color a usar en una ventana. Necesaria en caso de usar imágenes en color indexado o en niveles de gris. Una paleta muy usada es una de 256 niveles de gris, que puede crearse con gray(256).

```
im=imread('faunia.jpg');   % Lee JPG
figure(1); image(im);      % Crea figura (ventana) y visualiza imagen

im2=im(:,end:-1:1,:); % Flip izquierda/derecha
figure(2); image(im2)   % Visualiza nueva imagen en una 2ª ventana
imwrite(im2,'foto_flip.jpg','Quality',50);  % Guardamos nueva imagen (poca calidad)

im=imread('fotobw.jpg'); % Lee foto BW
figure(3); image(im); pause
paleta=gray(256);   % Crea paleta con 256 grises
colormap(paleta);   % Asigna paleta a figura activa
imshow(im);         % Hace los arreglos de la paleta automáticamente
```

En MATLAB disponemos de una Image Toolbox (librería) con muchos programas que hacen más fácil la manipulación y visualización de imágenes. Por ejemplo **imshow**() visualiza una imagen como image(), pero más completa: muestra la imagen con su relación de aspecto (ancho/alto) correcta, usa automáticamente una paleta de gris si detecta que la imagen es monocroma, etc.

Iremos introduciendo las funciones necesarias de MATLAB según aparezcan. En este primer LAB algunas de las que usaremos son:

figure -> crea una nueva ventana
subplot() -> permite ver varias imágenes/gráficos en la misma ventana.
max(im(:)), **min**(im(:)) : máximo y mínimo de los valores de una imagen.
mean2(im), **std2**(im): media/desviación standard de los valores de una imagen 2D
imhist(im) -> visualiza el histograma de una imagen
imresize(im,P) -> cambia tamaño de una imagen por un factor P

Ejercicio 1: Cargar la imagen del fichero 'foto_bw.jpg' y visualizarla con imshow(). Veréis que Matlab crea automáticamente una ventana donde poner la imagen. Haciendo **`>> whos im`** comprobaréis que la imagen cargada es de tipo uint8 (bytes).

El rango posible de valores de un byte es entre 0 y 255. Comprobemos el máximo y el mínimo de esta imagen usando las funciones max() y min(). Si hacéis **`max(im)`** obtendréis un montón de números en vez del máximo esperado. La razón es que muchas funciones de MATLAB, al aplicarlas a una imagen, se aplican a sus columnas por separado. En este caso obtenemos el valor del máximo de cada columna. Para evitarlo hay que hacer **`max(im(:))`.** Con im(:) convertimos toda la imagen en un vector y entonces la función max() nos dará el máximo de TODA la imagen. Volcad valores máximo/mínimo obtenidos, así como el valor medio (función mean).

Para operar con las imágenes usualmente es necesario pasarlas previamente a tipo double, ya que muchas operaciones no se pueden hacer con el tipo de datos bytes. Haced **`im2=double(im),`** lo que convierte la imagen a tipo "double" manteniendo sus valores. Si ahora tratáis de ver la nueva imagen im2 con imshow() veréis que no se ve correctamente. El problema es que imshow (y similares funciones) esperan que la imagen recibida sea:

- De tipo uint8 con valores entre 0 (negro) y 255 (blanco).
- De tipo double pero entonces con valores entre 0 (negro) y 1 (blanco).

En este caso im2 es double pero con valores entre 0 y 255. Para arreglarlo basta hacer **`im2=double(im)/255`**, lo que reescala sus valores entre 0 y 1. Comprobad que la nueva imagen se visualiza correctamente. Tened cuidado de no confundiros y hacer **`im2=double(im/255)`**. Adjuntad la imagen en este caso. ¿Por qué pasa esto?

Como convertir nuestras imágenes de uint8 (rango 0-255) a double (con rango 0-1) es un paso muy habitual antes de cualquier procesado, MATLAB tiene la función **`im2double()`** que convierte a double y rescala al mismo tiempo.

Una vez que la imagen es de tipo double podremos procesarla (hacer operaciones con ella). Por ejemplo, aplicad la función $\sin(\pi \cdot x)$ a los valores de la imagen (sin hacer bucles, trabajando con toda la imagen a la vez). Adjuntad imagen resultado.

En este caso, la imagen procesada también mantiene sus valores en [0,1]. Si tras el procesado la imagen tiene valores fuera de este rango el algoritmo debe indicarnos qué hacer con esos valores. Por ejemplo, podemos simular que añadimos un ruido a la imagen original sumándole un ruido aleatorio **`0.1*randn(size(im)).`** Hacedlo y volcad el máximo y mínimo de la imagen resultante.

Un posible criterio para tratar estos valores fuera de rango es "cortar" a 0 y a 1, esto es, poner a 1 aquellos valores mayores de 1 y a 0 los que sean menores que 0. Indicad qué comandos usaríamos para hacer esto SIN BUCLES (trabajando con la imagen en su conjunto). Visualizar y adjuntad la imagen resultante.

Ejercicio 2: Cargar la imagen del fichero 'faunia.jpg' y visualizarla con imshow(). Es una imagen en color y haciendo **>>whos im** veréis que para MATLAB es una matriz de alto x ancho x 3 (la última dimensión son los 3 planos R,G,B).

Usando subplot(2,2,1), subplot(2,2,2), subplot(2,2,3), ... visualizar en una misma figura la imagen RGB y sus tres planos de color por separado. Para extraer cada plano de color haced im(:,:,1), ... Adjuntad la figura resultante. Observad como el canal R tiene valores altos (blancos) donde predominaba el rojo en la imagen RGB, los blancos de la imagen original presentan valores altos en los tres canales, etc.
Adjuntad la imagen obtenida con imshow(im(:,:,[3 1 2])). ¿Qué estamos haciendo?

El espacio de color RGB es el usado para guardar las imágenes en disco y para visualizarlas (solo hay que mandar cada canal a los canales R, G y B del monitor), pero es posible usar otros espacios de color, dependiendo de la aplicación deseada. En todos ellos tendremos 3 planos (la cantidad de información no cambia) pero el tipo de información de cada plano será diferente según el espacio escogido.

Muchos esquemas de color (HSV, YIQ, ...) separan en un canal la información de la luminancia o intensidad de la imagen (la versión en grises de la imagen en color) y en los otros 2 canales se guarda la información relativa al color.

Por ejemplo, el espacio HSV separa la luminancia de la imagen en el canal V con valores entre 0 (negro) y 1 (blanco) y guarda el color en los canales H (Hue) y S (Saturación). En este esquema la información de color se describe de una forma similar a como lo solemos describir nosotros:

- **Hue (tono):** toma valores entre 0 y 1 indicando el color "básico" de acuerdo a la escala adjunta. Es esta escala 0 equivale a un rojo, 0.33 es un verde, 0.66 un azul y en 1 estamos de vuelta en el rojo (es una escala circular). Colores como el amarillo, cyan o el púrpura ocupan posiciones intermedias.

- **Saturación:** un valor entre 0 y 1 que indica la pureza del color: un valor de 0 corresponde a un gris, un valor bajo es un tono pastel, poco saturado. Valores altos indican colores vivos (saturados). En la figura se muestra la escala de saturación desde 0 (izquierda) a 1 (derecha) para el color rojo:

Convertir la imagen RGB a HSV usando el comando: **hsv=rgb2hsv(im);**
Modificar sus planos HSV según las instrucciones siguientes. En cada caso dad el comando usado y adjuntad la imagen resultante. Recordad partir siempre de la imagen original (no acumuléis los cambios) y volver siempre al espacio RGB para mostrar los resultados.

- Poner a 0.5 todos los valores del canal V (luminancia) lo que deja únicamente la información de color de la imagen.

- Poner a 1 (máximo) todos los valores del canal de la saturación. Debéis ver una imagen con colores exageradamente saturados.

- Poner a 0.33 todos los valores del canal H (tono), dando a todos los píxeles un tono verdoso (con las diferencias indicadas por sus correspondientes valores de saturación y luminancia).

Recordad que aunque las modificaciones se hagan en el nuevo espacio de color, la visualización siempre debe hacerse en RGB, por lo que antes de llamar a imshow() debemos deshacer la transformación y volver al espacio RGB usando hsv2rgb().

Si no deseamos cambiar a otro espacio de color sino solo obtener una versión en grises de una imagen color podemos usar la función rgb2gray(). Esta función calcula una imagen monocroma como el promedio de los tres planos de una imagen en color:

`0.2989*R + 0.5870*G + 0.1140*B`

Observaréis que usamos una media de los tres canales, pero dando más peso al verde y menos al azul. Fórmulas similares (dando más peso al verde y al rojo que al azul) se usan al extraer la luminancia en otros espacios de color. ¿Por qué creéis que se usan este tipo de fórmulas y no simplemente la media (R+G+B)/3?

Ejercicio 3: Una fotografía correctamente expuesta (que no hay que confundir con una buena fotografía) tendría idealmente un valor no muy alejado del punto medio 128. Igualmente su mínimo debería acercarse al 0 y su máximo al 255, para así aprovechar todo el rango disponible. En el ejercicio anterior hallamos la media de una imagen como **`mean(im(:))`**. Para imágenes monocromas (solo 2 dimensiones) también podemos usar la función **`mean2(im)`** que calcula la media de una matriz 2D.

Cargad la imagen 'bw1.png' y obtener su media usando mean2() y comprobad que es igual a la calculada como mean(im(:)). La media nos aporta una idea preliminar del nivel de luminosidad de la imagen, indicando si predominan los tonos oscuros (media baja) o las luces (media alta). La información completa del reparto de los valores de los píxeles de una imagen se obtiene con su histograma, que es un recuento de cuántos píxeles tienen un valor dado. El histograma es una gráfica donde en su eje X se muestra el rango de los posibles valores de los píxeles (0 a 255) y en el eje Y el número de píxeles en la imagen con ese valor. La función de MATLAB imhist(im) visualiza el histograma de una imagen.

Escribir una función function show_hist(im) que reciba una imagen (monocroma) y pinte la imagen en la parte superior (subplot(211)). Para pintar el histograma usar subplot(212) y luego llamar a imhist(im). Después de la llamada a imhist(), añadir **`set(gca,'Xlim',[-5 260]);`** para poder apreciar mejor lo que pasa en los valores extremos, cerca del 0 y del 255. Adjuntar código de vuestra función y la figura

resultante (imagen + histograma) para la imagen cargada. Observaréis que esta es una imagen de bajo contraste, cuyos valores no cubren todo el rango de 0 a 255.

Cargad ahora la imagen 'bw2.png' y obtener su media, que será similar a la de la imagen anterior. Adjuntad la imagen + histograma obtenida con show_hist(). En este caso se trata de una imagen de alto contraste, cuyos valores sí que cubren el rango completo. ¿A qué creéis que es debido los picos a la izquierda y a la derecha del histograma? ¿A qué zonas de la imagen corresponden?

Al igual que la media nos indica el nivel de luminosidad de una imagen, su contraste está asociado con la desviación standard σ (que mide la separación de los valores respecto al valor medio). Las imágenes de alto contraste tienen una distribución de valores o histograma más "ancha" y por lo tanto un valor mayor de σ. Calculad las desviaciones standard σ de nuestras 2 imágenes (usando la función std2).

Ejercicio 4: Hemos visto que en una imagen monocroma su media y su desviación nos indican su nivel de luminosidad y su contraste. Al manejar planos de color, estos valores puede asociarse a aspectos como por ejemplo el tono de la luz (la diferencia entre una imagen cálida, de tonos anaranjados, o fría, con una luz más azulada o blanca). En esta apartado vamos a intentar "transferir" el tono de color de una imagen (referencia) a otra (destino). El proceso sería:

- Cargar la imagen de referencia ("faro.jpg") y la de destino ("playa.jpg").
- Usando rgb2ntsc() convertirlas al espacio de color NTSC. Como es habitual, el 1er plano es la luminancia mientras que el 2º y 3º llevan la info de color.
- Barrer (k=1:3) los tres planos del nuevo espacio de color y en cada paso:

    1) Usando las funciones mean2() y std2() hallar la media y desviación del k-ésimo plano de la imagen de referencia (m,σ) y destino (m_0,σ_0).
    2) Modificar el k-ésimo plano de la imagen destino para que pase a tener media m y varianza σ en vez de sus valores (m_0,σ_0) originales.

  Modificar un plano o imagen con una media m_0 y desviación standard σ_0 para que pase a tener una media m y desviación standard σ es sencillo. Si X es uno de los canales originales con media m_0 y desviación σ_0 basta hacer:

```
X = (X - m₀)/ σ₀    % Pasamos a media cero y desviación standard σ=1
X = (X * σ) + m     % Pasamos a desviación Standard σ y media m
```

Una vez "transferida" la información entre los tres planos devolveremos la imagen modificada al espacio inicial RGB usando ntsc2rgb() para poder visualizarla.

Adjuntad vuestro código y la imagen obtenida. Volcad para cada plano los valores de partida (m_0,σ_0) y los nuevos (m,σ). Usad fprintf con 2 decimales (%.2f).

Sabemos que la media y la desviación standard no capturan ni mucho menos toda la información de una imagen, por lo que no es esperable que este sencillo método nos dé resultados muy buenos. Más adelante veremos métodos que nos permiten transferir más información del histograma de una imagen a otra.