```
Preliminar: Jugando con Colores
          Vamos a crear desde cero un visor de imágenes, que nos permitirá abrir un archivo de imagen y aplicarle ciertos filtros. Sin embargo, vamos a aprender
          primero a manejar colores a partir de tuplas, y luego a construir matrices de pixeles.
          Nuestro primer color
          Los colores digitales se pueden representar de muchas maneras. La más sencilla es usando el formato RGB, que consiste en construir colores a partir de
          tres componentes, Red, Green, y Blue.
          Primero vamos a entender cómo usar tuplas para representar un color usando el formato RGB. Vamos a empezar teniendo tres variables inicializadas en
          cero:
 In [1]: r = 0
          g = 0
          b = 0
          TIP: En python podemos hacer varias asignaciones en una sola línea de la siguiente manera:
 In [2]: r,g,b = 0,0,0
          Podemos usar una tupla para referirnos a esa tripleta de valores:
 In [3]: color = (r,g,b)
          El tipo de esa variable es tuple :
 In [4]: type(color)
 Out[4]: tuple
          Visualizando un Color
          Una imagen digital es finalmente una matriz de pixeles. Sin embargo, antes de ver matrices a fondo, vamos a crear la imagen más sencilla posible, una
          matriz de 1x1 que contiene un solo color.
 In [5]: I = [[color]]
          Para poder mostrar esa imagen vamos a usar una librería, que se llama matplotlib.
 In [6]: import matplotlib.pyplot as plt
          Todos los llamados a métodos de esa librería los haremos usando la variable plt. Para mostrar la imagen toca hacer tres cosas:
            1. Crear una figura usando el método figure. Este método recibe opcionalmente como parámetro el tamaño de la figura (figsize), que es una tupla que
              describe el ancho y el alto en pulgadas.
              plt.figure(figsize = (1,1))
            1. Matplotlib muestra imágenes con los ejes. Vamos a decirle que no queremos pintar los ejes:
             plt.axis('off')
            1. Finalmente, mostramos la matriz usando los dos siguientes métodos:
              plt.imshow(matriz)
              plt.show()
          Vamos a meter lo que acabamos de hacer en una función:
 In [7]: def visualizar_imagen(matriz:list)->None:
              plt.figure(figsize = (1,1))
              plt.axis('off')
              plt.imshow(matriz)
              plt.show()
          Si llamamos a visualizar_imagen vamos a ver una imagen negra:
 In [8]: visualizar_imagen(I)
          Este es el script completo con todo lo anterior. Cópielo en un nuevo archivo de spyder y guárdelo con el nombre visor_imagenes.py en una carpeta llamada
          N4L1:
 In [9]: import matplotlib.pyplot as plt
          import random
          def visualizar_imagen(matriz:list)->None:
               """ Muestra la imagen recibida
                  imagen (list): Matriz de MxN con tuplas (R,G,B) que representan la imagen a visualizar.
              plt.figure(figsize = (1,1))
              plt.axis('off')
              plt.imshow(matriz)
              plt.show()
          def crear_pixel(r:int,g:int,b:int)->list:
               """Crea una imagen de 1 pixel. Recibe los valores de rojo, verde, y azul del pixel."""
              pixel = (r,g,b)
              I = [[pixel]]
              return I
          visualizar_imagen(crear_pixel(0, 0, 0))
          Colores Puros
          Podemos generar cualquier color si cambiamos los valores de \ r , \ g , o \ b . Primero obtengamos un color \ rojo puro:
In [10]: visualizar_imagen(crear_pixel(255,0,0))
          Ahora obtengamos un color verde puro:
In [11]: visualizar_imagen(crear_pixel(0,255,0))
          Y un color azul puro:
In [12]: visualizar_imagen(crear_pixel(0,0,255))
          ¿Por qué 255?
          Porque los valores que puede tomar cada componente de color van de 0 a 255. Si metemos un número mayor, se asumirá que es 255:
In [13]: visualizar_imagen(crear_pixel(0,0,600))
          Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
          Hay otra forma de expresar colores, a partir de float en lugar de int por cada componente. Lo que cambia es el rango de valores, que ya no es de 0-
          255, sino de 0.0 a 1.0:
In [14]: visualizar_imagen(crear_pixel(0.0,0.0,1.0))
          Usted puede usar cualquiera de las dos representaciones.
          Mezclas
          Vamos ahora a mezclar colores puros para obtener cualquier otro color. Por ejemplo, para obtener amarillo mezclamos rojo y azul:
In [15]: visualizar_imagen(crear_pixel(255,255,0))
          El magenta se obtiene mezclando rojo y azul puros:
In [16]: visualizar_imagen(crear_pixel(255,0,255))
          Y el cyan se obtiene mezclando verde y azul puros:
In [17]: visualizar_imagen(crear_pixel(0,255,255))
          Finalmente, el blanco es la mezcla de los tres colores puros:
In [18]: visualizar_imagen(crear_pixel(255,255,255))
          Juegue con diferentes valores de r , g , y b para obtener más colores:
In [19]: visualizar_imagen(crear_pixel(244,128,12))
          Incluso podemos tener una función que siempre que la invoque me dará un color aleatorio:
In [20]: def dar_color_aleatorio()->tuple:
               return (random.randint(0,255), random.randint(0,255), random.randint(0,255))
          Incluso puede usar la función random , en lugar de randint , que me retorna un float entre 0.0 y 1.0:
In [21]: def dar_color_aleatorio()->tuple:
              return (random.random(), random.random(), random.random())
In [22]: | color = dar_color_aleatorio()
          visualizar_imagen([[color]])
          Rampas de color
          Una rampa de color es una transición gradual entre un color y otro. Vamos a hacer ahora una rampa para pasar de negro a rojo.
          Primero modifique visualizar_imagen para que nos muestre una imagen de 10 pulgadas de ancho:
In [23]: def visualizar_imagen(matriz:list)->None:
              plt.figure(figsize = (10,10))
              plt.axis('off')
              plt.imshow(matriz)
              plt.show()
          Nuestra rampa va a empezar en negro, y vamos a ir incrementando el valor de R, de 5 en 5, hasta que llegue al rojo puro.
In [24]: def crear_fila_pixeles():
              I = [[]]
              for i in range(0,256,5):
                  c = (i, 0, 0)
                  I[0].append(c)
              return I
In [25]: F = crear_fila_pixeles()
          visualizar_imagen(F)
          Esta rampa es una matriz de 1 fila por 52 columnas.
          Matrices de color
          Inspirados en lo anterior, podemos crear una función que cree las tres rampas de los colores puros:
In [26]: def crear_rampas():
              I = [[],[],[]]
              for i in range(0,256,5):
                   r = (i, 0, 0)
                   g = (0, i, 0)
                   b = (0, 0, i)
                  I[0].append(r)
                  I[1].append(g)
                  I[2].append(b)
              return I
In [27]: visualizar_imagen(crear_rampas())
          Fíjese que esta imagen es una matriz de 3 filas x 52 columnas.
          Podemos tener una función que arma una matriz de colores aleatorios usando dos ciclos:
In [28]: def crear_matriz_de_pixeles(ancho:int,alto:int)->list:
              I = []
              for i in range(0,alto):
                   for j in range(0, ancho):
                       fila.append(dar_color_aleatorio())
                  I.append(fila)
              return I
In [29]: visualizar_imagen(crear_matriz_de_pixeles(2,2))
In [30]: visualizar_imagen(crear_matriz_de_pixeles(5,5))
In [31]: visualizar_imagen(crear_matriz_de_pixeles(500,200))
          Cargando imágenes
          Para importar imágenes en diferentes formatos (PNG,JPEG,TIFF,etc.) vamos a usar el módulo image de matplotlib :
In [32]: import matplotlib.image as mpimg
          En concreto, usamos la función imread para leer la imagen, y la función tolist para convertirla en una matriz:
In [33]: matriz = mpimg.imread("imagen2.png").tolist()
          Si vemos el primer pixel de la imagen, es una lista de 3 float :
In [34]: print(matriz[0][0])
          [0.4274509847164154, 0.5647059082984924, 0.9176470637321472]
          Para manipular mejor las imágenes -y repasar tuplas- vamos a traducir cada lista de 3 colores a una tupla. Para ello, necesitamos saber las dimensiones de
          la imagen:
In [35]: alto = len(matriz)
          ancho = len(matriz[0])
          Finalmente, recorremos la imagen, pixel a pixel, y traducimos cada lista a tupla:
In [36]: imagen = []
          for i in range(alto):
              fila = []
              for j in range(ancho):
                   tupla = tuple(matriz[i][j])
                   fila.append(tupla)
              imagen.append(fila)
          Esta es la función completa:
In [37]: def cargar_imagen(ruta_imagen: str)-> list:
               """ Carga la imagen que se encuentra en la ruta dada.
                   ruta_imagen (str) Ruta donde se encuentra la imagen a cargar.
              Retorno:
                   list: Matriz de MxN con tuplas (R,G,B).
              matriz = mpimg.imread(ruta_imagen).tolist()
              alto = len(matriz)
              ancho = len(matriz[0])
              imagen = []
              for i in range(alto):
                   fila = []
                   for j in range(ancho):
                       tupla = tuple(matriz[i][j])
                       fila.append(tupla)
                   imagen.append(fila)
              return imagen
          Y este es el resultado de cargar un archivo de imagen:
In [38]: I = cargar_imagen("imagen2.png")
In [39]: visualizar_imagen(I)
          Filtros
          Un filtro no es nada más que un recorrido total sobre la matriz, donde alteramos algunos (o todos) los pixeles.
          Convertir a negativo
          El negativo se calcula cambiando cada componente RGB, tomando el valor absoluto de restarle 1 al componente.
          Para evitar modificar la imagen original, vamos a crear una nueva matriz donde vamos metiendo los pixeles modificados:
In [40]: copia = []
          Para hacer el recorrido total, primero preguntamos a la matriz sus dimensiones:
In [41]: alto = len(imagen)
          ancho = len(imagen[0])
          Finalmente, hacemos el recorrido total, asignando a la copia los colores modificados:
In [42]: for i in range(alto):
              fila = []
              for j in range(ancho):
                  r, g, b = imagen[i][j]
                   nuevo = (abs(r-1), abs(g-1), abs(b-1))
                   fila.append(nuevo)
              copia.append(fila)
          Esta es la función completa:
In [43]: def convertir_negativo(imagen: list) -> list:
               """ Convierte la imagen en negativo.
              El negativo se calcula cambiando cada componente RGB, tomando el valor absoluto de restarle al componente 1.0.
                  imagen (list) Matriz de MxN con tuplas (R,G,B) que representan la imagen a convertir a negativo.
              copia = []
              alto = len(imagen)
              ancho = len(imagen[0])
              for i in range(alto):
                   fila = []
                   for j in range(ancho):
                       r, g, b = imagen[i][j]
                       nuevo = (abs(r-1.0), abs(g-1.0), abs(b-1.0))
                       fila.append(nuevo)
                   copia.append(fila)
              return copia
          Y este es el resultado:
In [44]: visualizar_imagen(convertir_negativo(cargar_imagen("imagen2.png")))
          Con lo aprendido el día de hoy, intente desarrollar los siguientes filtros:
In [45]: def reflejar_imagen(imagen: list)->list:
               """Refleja la imagen.
              Consiste en intercambiar las columnas enteras de la imagen, de las finales a la iniciales.
                   imagen (list) Matriz de MxN con tuplas (R,G,B) que representan la imagen a reflejar.
              copia = []
              # TODO - Completar
              return copia
In [46]: def binarizar_imagen(imagen: list, umbral: float)->list:
               """ Binariza la imagen.
              Consiste en llevar cada pixel de una imagen a negro o blanco.
              Para ello se requiere un umbral: si el promedio de los componentes RGB del pixel está por encima o igual se llev
          a a blanco y si está por debajo se lleva a negro.
              Parámetros:
                   imagen (list) Matriz de MxN con tuplas (R,G,B) que representan la imagen a binarizar.
                   umbral (float) Umbral de la binarización.
              copia = []
              # TODO - Completar
              return copia
In [47]: def convertir_a_grises(imagen: list)->list:
               """ Convierte la imagen a escala de grises.
              Para ello promedia los componentes de cada pixel y crea un nuevo color donde cada componente (RGB) tiene el valo
          r de dicho promedio.
              Parámetros:
                   imagen (list) Matriz de MxN con tuplas (R,G,B) que representan la imagen a convertir a grises.
              copia = []
              # TODO - Completar
              return copia
          Si así lo desea, puede también tener un módulo de consola para probar más fácilmente:
In [50]: # -*- coding: utf-8 -*-
          Ejemplo Nivel 4: Visor de imágenes
          Temas:
          * Matrices
          @author: Cupi2
          # import trabajo as visor_imagenes
          import visor_imagenes
          def imprimir_menu_principal():
               """ Imprime los items del menú principal de la aplicación
              print("\n
                                   Visor de imágenes\n")
              print("(1) Cargar imagen")
              print("(2) Negativo")
              print("(3) Reflejar")
              print("(4) Binarizar")
              print("(5) Escala de grises")
              print("(6) Convolución")
              print("(7) Salir")
          def cargar_imagen() -> list:
               """ Muestra las opciones para cargar una imagen y carga la imagen seleccionada por el usuario.
              ruta = input("Ingrese el nombre del archivo que contiene la imagen: ")
              imagen = visor_imagenes.cargar_imagen(ruta)
              visor_imagenes.visualizar_imagen(imagen)
              return imagen
          def ejecutar_binarizar_imagen(imagen: list) -> list:
               """ Pide al usuario el umbral deseado y binariza la imagen recibida por parámetro.
              Parámetros:
                  imagen (list) Matriz (M,N,3) con la imagen a binarizar.
              umbral = float(input("Ingrese el umbral (valor entre 0 y 1):"))
              print("Calculando imagen...")
              imagen = visor_imagenes.binarizar_imagen(imagen, umbral)
              visor_imagenes.visualizar_imagen(imagen)
              return imagen
          def ejecutar_convolucionar_imagen(imagen: list) -> list:
               """ Aplica la convolución a la imagen recibida por parámetro.
              Parámetros:
                  imagen (list) Matriz (M,N,3) con la imagen a convolucionar.
              print("Calculando imagen...")
              imagen = visor_imagenes.convolucion_imagen(imagen)
               visor_imagenes.visualizar_imagen(imagen)
              return imagen
          def ejecutar_convertir_negativo(imagen: list) -> list:
              print("Calculando imagen...")
              imagen = visor_imagenes.convertir_negativo(imagen)
              visor_imagenes.visualizar_imagen(imagen)
              return imagen
          def ejecutar_reflejar_imagen(imagen: list) -> list:
              print("Calculando imagen...")
              imagen = visor_imagenes.reflejar_imagen(imagen)
              visor_imagenes.visualizar_imagen(imagen)
              return imagen
          def ejecutar_convertir_a_grises(imagen: list) -> list:
              print("Calculando imagen...")
              imagen = visor_imagenes.convertir_a_grises(imagen)
              visor_imagenes.visualizar_imagen(imagen)
              return imagen
          def ejecutar_aplicacion():
              salir = False
              imagen = visor_imagenes.cargar_imagen("imagen2.png")
              visor_imagenes.visualizar_imagen(imagen)
              while not salir:
                   imprimir_menu_principal()
                   opcion = int(input("Ingrese la opción deseada: "))
                   if opcion == 1:
                       imagen = cargar_imagen()
                   elif opcion == 2:
                      imagen = ejecutar_convertir_negativo(imagen)
                   elif opcion == 3:
                       imagen = ejecutar_reflejar_imagen(imagen)
                   elif opcion == 4:
                       imagen = ejecutar_binarizar_imagen(imagen)
                   elif opcion == 5:
                       imagen = ejecutar_convertir_a_grises(imagen)
                   elif opcion == 6:
                       imagen = ejecutar_convolucionar_imagen(imagen)
                   elif opcion == 7:
                       salir = True
                   else:
                       print("El valor ingresado no es válido.")
          ejecutar_aplicacion()
                     Visor de imágenes
          (1) Cargar imagen
          (2) Negativo
          (3) Reflejar
          (4) Binarizar
          (5) Escala de grises
          (6) Convolución
          (7) Salir
          Entrega
```

Suba a Brightspace los archivos que haya desarrollado. La próxima clase trabajaremos con filtros más especializados.