ANEXO 2. INTRODUCCIÓN A PYTHON Y OPENCV

En este capítulo se realizará una introducción al lenguaje de programación Python, incluyendo variables, operadores, condicionales, iteraciones, algunos tipos de datos y funciones.

Como primer acercamiento al lenguaje, se visualiza en pantalla el mensaje "*Hola mundo*" mediante la función `print()`, y se define como argumento de la función la cadena de caracteres.

```
print("Hola mundo")
```

Indentación: Algo importante a tener en cuenta en Python es el uso obligatorio de indentación para definir el alcance de un determinado entorno. Por ejemplo, las instrucciones asociadas a un ciclo `for` deberán tener un nivel de indentación respecto a esta función, así:

```
for i in range(3): # Imprimir de 0-2
  print(i)
```

Variables

En Python, no es necesario asignar el tipo de una variable cuando se utiliza por primera vez. Los principales tipos de variables son *integer, float, string, boolean, NoneType*. En cualquier caso, siempre es posible verificar el tipo de una variable usando `type()`, o el estado de una variable lógica utilizando `bool()`. Para convertir entre tipos de datos, es posible utilizar funciones como `str()`, `int()` o `float()`. Es posible también ingresar cadenas texto mediante la función `input()`.

```
# Integer
num1 = 2021

# float
num2 = 20.21

```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
# String
mensaje = "Teleco UMNG"

# Boolean
var_logica = False
#Al utilizar True, False (la primera letra deberá
estar en mayúscula)

# None
num3 = None
```

```
# Verificar contenido y tipo de una variable
print(num1)
type(num1)

# Prueba de variable lógica
bool("valor") # True
bool("") # False
```

```
# Ingreso de datos por usuario
name = input("Enter your name: ") # name es de tip
o String
print("El nombre ingresado es: " + name)
```

Operadores

Python dispone de operadores aritméticos, de comparación y lógicos como cualquier lenguaje de programación. Los principales operadores aritméticos incluidos son (entre paréntesis se especifica el operador):

> Suma (+), resta (-), multiplicación (*), división (/), módulo de la división (%), parte entera de la división (//), asignación (=), suma y asignación (+=), resta y asignación (-=), multiplicación y asignación (*=), división y asignación (/=).

Como operadores de comparación, se tiene comprobación de igualdad (==), de diferencia (!=), mayor que (>), menor que (<), mayor o igual (>=) y menor o igual (<=).

Prohibida su reproducción o difusión total o parcial | © 2023 |

En cuanto a operadores lógicos, se cuenta con operadores booleanos tipo `and`, `or` y `not`.

El operador de concatenación de cadenas de texto en python es "+", como lo muestra la siguiente línea de código:

```
var = 5.0
print("Obtendré un " + str(var) + " en Computer Vi
sion")
```

Condicionales

El uso de condicionales en Python (`if`, `else`, `elif`) tiene la versatilidad de la mayoría de lenguajes de programación, tal como se relaciona a continuación:

Uso básico:

```
var = True
if var:
  print("Condición verdadera")
else:
  print("No cumple la condición")
```

Evaluación de una única condición:

```
a = 5
b = 10

if a == 5:
  print('Ok')
```

Evaluación de diferentes condiciones:

```
if var_mes == 4:
  print('Estamos en febrero')
elif var_mes == 6:
  print('Estamos en junio')
elif var_mes == 9:
  print('Estamos en septiembre')
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
elif var_mes == 11:
  print(' Estamos en noviembre')
else:
  print('Este mes no es de 30 días')
```

Evaluación utilizando comparación y operadores lógicos :

```
# AND
if a == 1 and b == 2:
  print('Los dos números son enteros positivos
menores a 3')

# OR
if a == 0 or a == 1:
  print('El factorial del número es 1')

# NOT
if not a == 0:
  print('El número es diferente de cero')
```

Existe también la posibilidad de evaluar un ocndicional en una sola línea:

```
# única condición
if var1 >= var2: print("La primera variable es
mayor o igual a la segunda")

# If + Else
print('A vale
cero ') if a == 0 else print('A es diferente a
cero')
```

Para el uso de if anidados, es importante tener en cuenta la indentación:

```
num = 123
if num > 0:
  print("El número es positivo")
  if num > 5:
    print("El número es positivo mayor a 5")
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
  if num > 55:
    print("El número es positivo mayor a 55")
else:
  print("El número es negativo o cero")
```

Iteraciones (loops)

Uso de ciclo `for` para iterar sobre elementos de un arreglo, o una cadena de texto:

```
programa = "Teleco"
for char in programa:
  print(char) # Imprime cada carácter de la cadena
de texto

vigencia = ['2', '0', '2', '1']
for ele in vigencia: # Imprime cada elemento del a
rreglo
  print(ele)
```

Es posible también generar una secuencia de números utilizando `range`. Aquí se especifica el valor de inicio (opcional, si no se especifica se utiliza 0), el valor final (obligatorio) y el valor de incremento (opcional, si no se especifica se utiliza 1).

```
# Ciclo for y Range para visualizar los números
impares entre 1 y 10
for i in range(1, 10, 2):
  print("Número " + str(i))
else:
  print("Fin")
```

Además, los valores generados por `range` pueden ser utilizados para indexar los elementos en un arreglo, por ejemplo:

```
arr = ['2', '0', '2', '1']
for index in range(0, len(arr)):
  print(arr[index])
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

Al igual que en los condicionales, es posible anidar ciclos `for` (con un manejo adecuado de indentación):

```
# Ciclos For anidados
for i in range(5):
  for j in range(10):
    print("Conteo interno" + str(j))
  print("Conteo externo..." + str(i))
```

Por su parte, para el uso de la sentencia `while`, es necesario realizar el incremento de la variable utilizada en la condición:

```
i = 2
while i <= 10:
  print(i)
  i += 2
```

Hay que tener en cuenta que tanto `for` como `while` admiten el uso de `break`, `continue` y `else`:

```
while i <= 25:
  print(i)
  i += 1
  if i == 18:
    break
else:
  print('Completado')
```

Contenedores de datos

Listas: las listas son un tipo de arreglo ordenado en Python, cuyos elementos pueden ser de diferente tipo. Por ejemplo, la siguiente lista contiene valores enteros, cadenas de texto y valores booleanos especificados entre corchetes [ ]:

```
lista = [1, 5, "xyz", True, "Telec0", 2021]
```


Prohibida su reproducción o difusión total o parcial | © 2023 | Pág. 142

Al ser un arreglo ordenado, es posible acceder, adicionar o borrar elementos mediante índices. Aquí hay que tener en cuenta que el índice del primer elemento es 0, el segundo tendrá un índice 1, y así sucesivamente. Es posible utilizar números negativos como índices, ante lo cual el índice -1 corresponde al último elemento de la lista, el -2 al penúltimo, y así sucesivamente. Por su parte, es posible rangos del arreglo mediante el operador dos puntos (:). Al utilizar este operador como índice único, se seleccionará el arreglo completo, mientras que si acompaña de valores que lo precedan o que lo sucedan, estos harán las veces de inicio y fin de rango, respectivamente.

```
# Acceso a los elementos de una lista por índices
lista[0]
lista[-1]
lista[-2]
lista[2:4] # inicio : fin-1
lista[:3] # 0... fin-1
lista[3:] # inicio... fin
lista[:] # Arreglo completo
```

Para cambiar un valor de la lista, bastará con asignar el nuevo valor en la posición o índice adecuado:

```
#Cambiar un valor de una lista especificando su 
posición
lista[4] = "Teleco"
```

Para agregar elementos a una lista, es posible utilizar tanto el método `append()`, como el método `insert()`. El primero de ellos agrega el nuevo elemento al final de la lista, mientras que el segundo inserta el objeto antes del índice especificado.

```
lista.append("UMNG")
lista
lista.insert(4, "Computer vision") # Insertar en u
na posición específica
lista
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

En cualquier caso, es posible obtener el número de elementos de un contenedor, utilizando `len()`.

```
len(lista)
```

Para eliminar objetos de una lista, basta con utilizar la sentencia `del`, el contenedor y la posición a eliminar. Para eliminar y devolver un elemento específico (posición) de una lista, el método `pop()` es el indicado. Sin embargo, si en este método no se especifica el índice, por defecto se eliminará el último elemento. Además, es posible especificar elementos a eliminar de una lista, de acuerdo con su valor. Para ello se utiliza el método `remove()` y el valor que debe tener el elemento a eliminar (el comando eliminará el primer elemento que encuentre en la lista con dicho valor).

```
# Borrar un elemento específico
del lista[4]    # opción 1
lista

# Eliminar el último elemento de la lista
elemento_eliminado = lista.pop()
print(elemento_eliminado)
print(lista)

# Eliminar elementos por su valor (el primer eleme
nto que encuentre)
lista.remove(5)
lista
```

Dado que una lista contiene elementos en posiciones específicas, es posible cambiar los elementos de posición, de acuerdo con un criterio de ordenamiento. Para esto, el método `sort()`, permite ordenar de manera ascendente (predeterminado) o descendente (especificando el parámetro `reverse=True`). Además, cuenta con la posibilidad de suministrar una función que especifique los criterios para ordenar los elementos.

```
# Ordenar (sort es una operación que no se puede d
eshacer)
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
letras = ["u", "m", "n", "g"]
letras.sort()
print(letras)

letras.sort(reverse=True)
print(letras)
```

Es importante tener en cuenta que esta operación no es reversible, por lo cual se deberá realizar una duplicación de la variable si se desea conservar el orden original. Para solamente devolver una nueva lista con todos los elementos en orden ascendente (sin cambiar la lista original), se puede utilizar la función `sorted()`:

```
# Imprimir datos ordenados sin cambiar el orden de
 los datos originales
letras = ["u", "m", "n", "g"]
print(sorted(letras))
print(alpha)
```

De forma complementaria, el método `reverse()`, permite ordenar los elementos de un arreglo en sentido inverso. Es decir, el último elemento de la lista se ubicará en primera posición, el penúltimo en segunda posición, etc.

```
# Invertir un arreglo
numeros = [10, 1, 5]
numeros.reverse()
print(numeros)
```

Para duplicar una lista, es importante tener en cuenta el método para realizarlo, dado que en algunos casos puede existir referenciación entre las dos variables, y un cambio en una de ellas puede verse reflejado en la otra variable, como en los siguientes ejemplos:

```
# Copia referenciado
letras = ["u", "m", "n", "g"]
letras2 = letras # Cualquier cambio en la variable
 original o en la segunda afecta la variable enlaz
ada
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
letras[1]="eme"
letras2[3]="gege"
print(letras)
print(letras2)
```

```
['u', 'eme', 'n', 'gege']
['u', 'eme', 'n', 'gege']
```

En el anterior código, un cambio en cualquiera de las dos variables posterior a la copia de la variable ocasionará que el contenido de las dos variables se vea alterado.

Para evitar esta situación, se puede realizar una copia independiente de todos los elementos de la lista, mediante el operador dos puntos [:]. De esta forma, al realizar un cambio en cualquiera de las dos variables posterior a su duplicación, solo afectará a la variable sobre la cual se realizó dicha operación:

```
# Copia de un arreglo no referenciado
letras = ["u", "m", "n", "g"]
letras1 = letras[:] # Copia independiente. Cualqui
er cambio en la variable original o en la segunda 
NO afecta la variable enlazada
letras[1]="eme"
letras1[3]="gege"
print(letras)
print(letras1)
```

```
['u', 'eme', 'n', 'g']
['u', 'm', 'n', 'gege']
```

Otra alternativa para crear una copia independiente de una lista es utilizar el método `copy()`, el cual devuelve una copia superficial de la lista.

```
# Referenciado 2
letras = ["u", "m", "n", "g"]
letras3 = letras.copy() # Cualquier cambio en la v
ariable original o en la segunda NO afecta la vari
able enlazada
letras[1]="eme"
letras3[3]="gege"
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
print(letras)
print(letras3)
```

```
['u', 'eme', 'n', 'g']
['u', 'm', 'n', 'gege']
```

Tuplas: este tipo de contenedor es similar a una lista, con la diferencia de que nos es posible cambiarlas posterior a su creación. Es decir, no es posible agregar elementos ni actualizar su contenido. Por ejemplo, la siguiente tupla contiene valores enteros, cadenas de texto y valores booleanos especificados entre paréntesis ( ):

```
tupla = [1, 5, "xyz", True, "Telec0", 2021]
```

El acceso a los elementos de la tupla se puede realizar de manera similar a los de una lista:

```
print(tupla)
print(tupla[3])
print(len(tupla))
```

```
(1, 5, 'xyz', True, 'Telec0', 2021)
True
6
```

Aunque es posible crear tuplas vacías, la adición o modificación de sus valores generará una excepción:

```
tupla[0] = "Diego"
```

```
TypeError: 'tuple' object does not support item assignment
```

```
tupla_blanco = ()   # Tupla vacía
tupla_blanco[0] = "Diego"
```

```
TypeError: 'tuple' object does not support item assignment
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

Conjuntos: este tipo de contenedor corresponde a una colección no ordenada de elementos, es decir que no existen elementos duplicados dentro del conjunto y que a su vez pueden ser de diferente tipo. Por ejemplo, la creación del siguiente conjunto con valores enteros y caracteres especificados entre llaves { }, devolverá un solo elemento por tipo:

```
conjunto = {'u', 'm', 'n', 'g', 2, 0, 2, 2}
print(conjunto)
```

```
{0, 2, 'm', 'n', 'g', 'u'}
```

Al igual que en las listas, es posible iterar sobre los elementos del conjunto (teniendo en cuenta las particularidades de este contenedor). Asimismo, el número de elementos del conjunto se puede obtener de forma similar a una lista o a una tupla:

```
# Acceder a elementos en un conjunto (a través de
un ciclo)
for elemento in conjunto:
  print(elemento)

# Longitud de un conjunto
len(conjunto)
```

```
0
2
m
n
g
u
```

Para adicionar elementos a una conjunto, es posible utilizar el étodo add() (para un elemento) o el método update para más de un elemento:

```
# Agregar 1 elemento
conjunto.add('a')
print(conjunto)

# Agregar más de un elemento
conjunto.update(['b', 'c', 'd'])
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
print(conjunto)
```

```
{0, 2, 'm', 'n', 'g', 'a', 'u'}
{0, 2, 'd', 'm', 'n', 'g', 'a', 'c', 'b', 'u'}
```

Para eliminar un elemento de la lista, Python dispone del método `remove()` y del método `discard()`, siendo más recomendable este último.

```
# Eliminar un elemento de la lista
conjunto.remove('a')
print(conjunto)

conjunto.discard('b') # Opción preferida
print(conjunto)
```

```
{0, 2, 'd', 'm', 'n', 'g', 'c', 'b', 'u'}
{0, 2, 'd', 'm', 'n', 'g', 'c', 'u'}
```

Diccionarios: tipo de contenedor en Python que corresponde a una colección desordenada que puede actualizarse. La forma de definir un diccionario en Python requiere definir para cada elemento una llave y su respectivo valor ():

```
diccionario = {
  "key" : "value"
}
```

Por ejemplo, el siguiente diccionario contiene cadenas de texto para tres elementos:

```
empleado = {
    "nombre": "Diego",
    "apellido": "Renza",
    "cargo": "Docente"
  }
```

El acceso a los valores del diccionario puede realizarse indexando la llave del elemento (entre corchetes []):

```
# Acceder a valores
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
empleado["nombre"]       # Opción 1
```

```
'Diego'
```

Sin embargo, al realizarlo de esta forma se genera un error cuando no existe la llave especificada. En su lugar, puede ser más adecuado utilizar el método `get()`:

```
empleado.get("nombre")
```

Al igual que en los casos anteriores, el número de elementos de la lista se obtiene con la función `len()`. Sin embargo, para los diccionarios es posible listas independientes para las llaves y para los valores:

```
# Obtener todas las llaves y valores de un diccion
ario (en una lista)
llaves = empleado.keys()
print(llaves)
# Valores
valores = empleado.values()
print(valores)
```

```
dict_keys(['nombre', 'apellido', 'cargo'])
dict_values(['Diego', 'Renza', 'Docente'])
```

Asimismo, es posible iterar sobre los elementos de un diccionario utilizando el método `items()` con la diferencia de que en este caso se tendrán variables tanto para la llave como para su valor:

```
for key, value in empleado.items():
  print(key + " corresponde a " + value)
```

```
nombre corresponde a Diego
apellido corresponde a Renza
cargo corresponde a Docente
```

Al utilizar el método `enumerate()` se tendrán variables tanto para el índice de la llave como para su valor:

```
for index, value in enumerate(empleado):
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

```
  print("El índice " + str(index) + " del dicciona
rio corresponde a la llave " + value)
```

```
El índice 0 del diccionario corresponde a la llave nombre
El índice 1 del diccionario corresponde a la llave apellido
El índice 2 del diccionario corresponde a la llave cargo
```

Funciones

La creación de funciones en Python involucra la utilización del comando `def`, el nombre de la función y especificar los argumentos necesarios para la ejecución de la función. Al crear la función, es preciso utilizar la debida indentación y a diferencia de otros lenguajes, no es obligatorio especificar el valor retornado por la función (en caso tal se utilizará `return` y el valor a retornar por la función). Para ejecutar la función, basta con especificar los argumentos de la misma como una tupla. En el siguiente código de ejemplo, se crea una función para la suma de una lista de números, donde se ha utilizado * para especificar que el número de argumentos de la función es variable:

```
def suma2(*num):
  print(sum(num))

suma2(1,3,7,9,15)
```

```
35
```

Al ejecutar una función en Python, es posible también ingresar los argumentos en desorden, con la condición de especificar el nombre y valor de cada argumento al llamar la función. Asimismo, es posible establecer el valor predeterminado de un argumento.

```
def visua_empleado(nombre, apellido, cargo='Docent
e'):
  print(nombre + " - " + apellido + " - " + cargo)

visua_empleado(apellido='Renza', nombre='Diego')
```

```
Diego - Renza - Docente
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

INTRODUCCIÓN A OPENCV

En esta sección del libro, se utilizarán algunas librerías y módulos para el manejo y visualización de datos e imágenes. En primera instancia, se importarán los módulos de extensión para Python NumPy y Matplotlib. NumPy (Numeric Python) está orientado a acelerar la ejecución de operaciones con estructuras de datos, con base en representación de matrices y arreglos multidimensionales. En el segundo caso, este permite activar el soporte interactivo de gráficas con `matplotlib` en cualquier momento de una sesión.

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
from PIL import Image
```

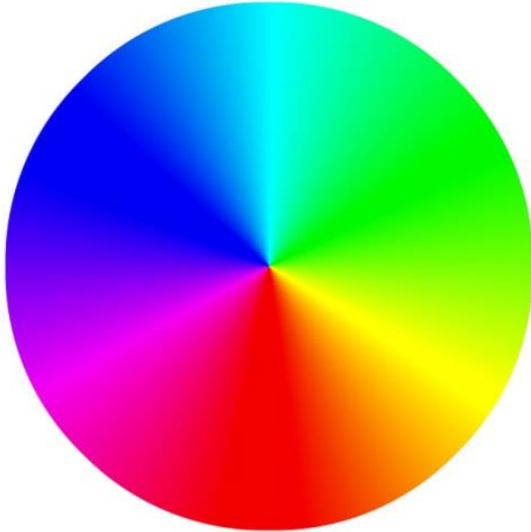
Para el manejo de imágenes en Python, en primera instancia se utilizará la librería Pillow (PIL), en particular con el módulo *Image*. Al utilizar este módulo se dispone de una clase para representar imágenes, además de la posibilidad de realizar operaciones sobre ellas (ej. carga desde archivo, creación de nuevas imágenes, etc.).

https://pillow.readthedocs.io/en/stable/reference/Image.html

Para abrir una imagen desde un archivo con PIL/Image se puede utilizar el método open, especificando la ruta del archivo (de no existir el archivo en la ruta especificada, se generará un error). Para visualizarla, bastará con llamar a la variable que la contiene:

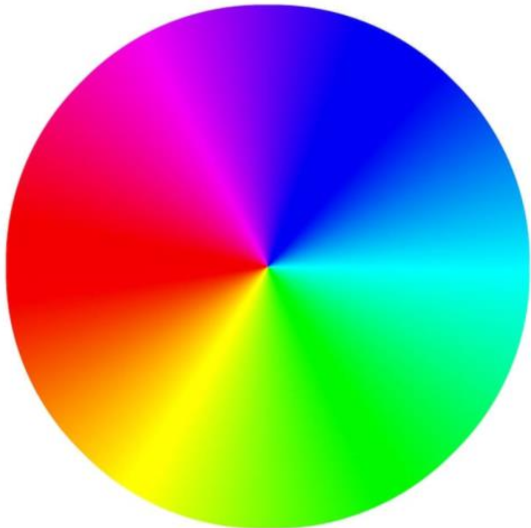
```
img = Image.open('images/Test.jpg')
img
```


Prohibida su reproducción o difusión total o parcial | © 2023 |



Como ejemplo de otro tipo de operaciones que se pueden realizar con PIL/Image, está la rotación en un ángulo específico. Para más información de esta librería, se remite al lector a la documentación oficial: https://pillow.readthedocs.io/en/stable/reference/Image.html.

```
img.rotate(-90)
```



Una vez leída la imagen, es posible comprobar el tipo de objeto al cual corresponde, que para el caso del ejemplo es una imagen tipo JPEG en formato PIL.

```
type(img)
```

```
PIL.JpegImagePlugin.JpegImageFile
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

Dado que la imagen está en formato PIL, para su procesamiento puede ser conveniente representarla como un arreglo tridimensional, utilizando el módulo NumPy. Para esto, es posible utilizar el método `asarray` sobre la variable de la imagen. Esto devolverá una variable tipo NumPy (arreglo multidimensional):

```
img_array = np.asarray(img)
type(img_array)
```

```
numpy.ndarray
```

Para obtener las dimensiones del arreglo, el método `shape` devuelve en este caso el número de filas, columnas y canales de la imagen:

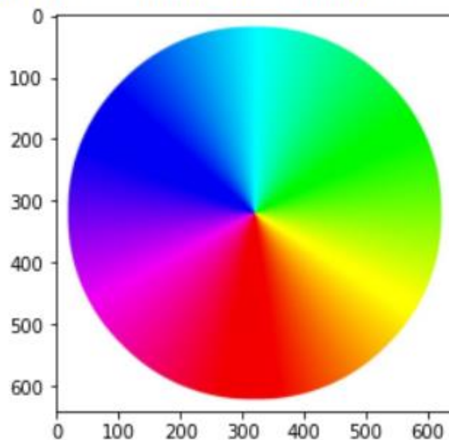
```
img_array.shape
```

```
(640, 640, 3)
```

Para visualizar la imagen en formato NumPy, una alternativa es utilizar Matplolib:

```
plt.imshow(img_array)
```

```
<matplotlib.image.AxesImage at 0x7fefcdfe4a20>
```



Esta imagen de ejemplo corresponde a una imagen a color (dado que tiene 3 canales), y el valor de cada píxel en cada canal está almacenado como un entero de 8 bits sin signo (*unsigned integer* de 8 bits, uint8). Es decir, el valor de intensidad de cada píxel en un canal dado oscila entre 0 (mínima intensidad o negro) y 255 (máxima intensidad o blanco).

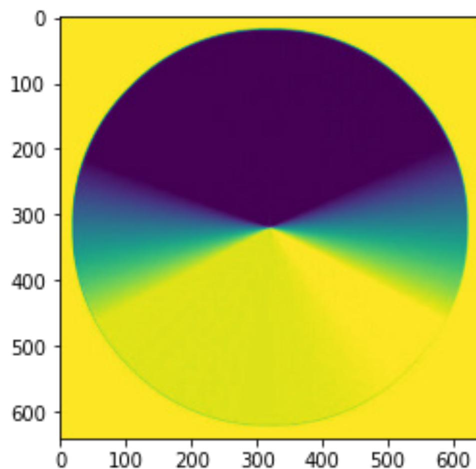
En el caso de los canales para una imagen a color, el espacio de color o representación comúnmente usada es RGB. En este espacio de color, el primer canal o canal 0 representa las intensidades en la longitud de onda del rojo, el

Prohibida su reproducción o difusión total o parcial | © 2023 |

segundo canal o canal 1 representa el verde y el tercer canal o canal 2 representa las intensidades del color azul. Al tener la imagen a color, cada píxel de la imagen tendrá 24 bits, es decir 8 bits/pixel/canal.

Para visualizar Para visualizar uno de los canales de la imagen, se realizará una copia de la variable que contiene el arreglo de la imagen, y en la función `imshow` se especificarán los índices respectivos a nivel de filas, columnas y canales. En este caso se mostrarán todas las filas, todas las columnas y solo el canal rojo:

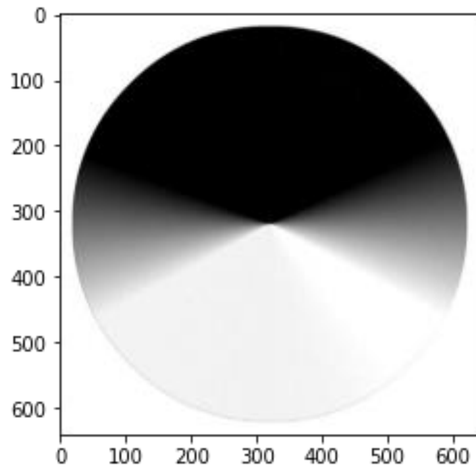
```
img_test = img_array.copy()

plt.imshow(img_test[:, :, 0])
```



En la imagen anterior se aprecian dos aspectos fundamentales. El primero tiene que ver con las intensidades de mayor valor (las que están en color más claro, amarillo en este caso). Se puede observar que las zonas de la imagen de color rojo son las que tienen mayor intensidad. Por otra parte, aunque se especificó que `imshow` mostrara solo un canal de la imagen, la visualización se realizó como una imagen a color; esto se debe al mapa de color predeterminado de *matplolib*. Para asegurarse que la imagen se muestre en escala de grises, es necesario especificar el argumento `cmap` en gris.

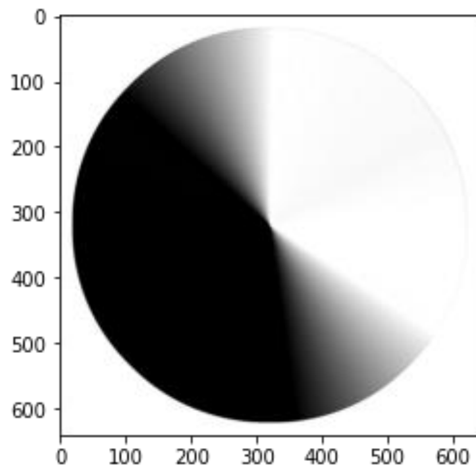
```
plt.imshow(img_test[:, :, 0], cmap ='gray')
```


Prohibida su reproducción o difusión total o parcial | © 2023 |



De forma similar es posible mostrar el canal verde de la imagen en escala de grises:

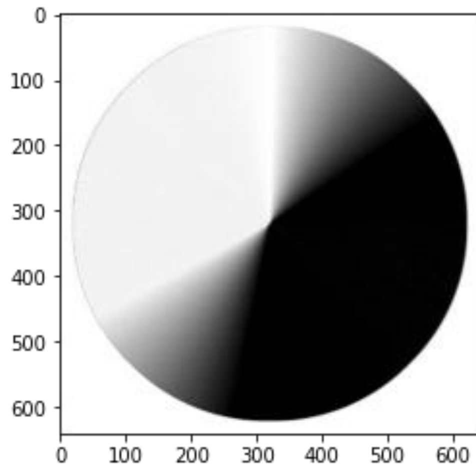
```
plt.imshow(img_test[:, :, 1], cmap ='gray')
```



Y el canal azul también en escala de grises:

```
plt.imshow(img_test[:, :, 2], cmap ='gray')
```

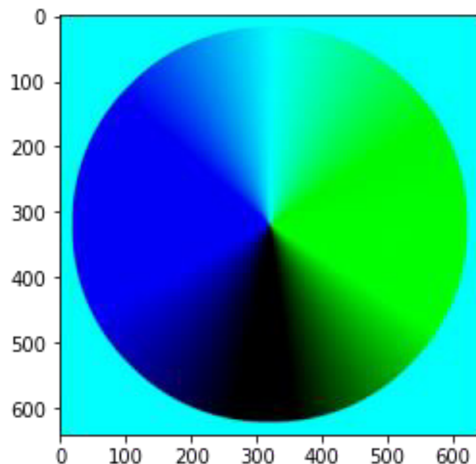

Prohibida su reproducción o difusión total o parcial | © 2023 |



Se invita al lector a analizar las imágenes anteriores, comparando las zonas de mayor intensidad respecto al canal visualizado.

De forma complementaria, y a manera de prueba, se eliminará un canal determinado, estableciendo todos los valores de píxel del canal en cero. De esta forma, al visualizar la imagen las tonalidades relacionadas con el color eliminado no se mostrarán.

```
img_test = img_array.copy()
img_test[:,:,0]=0
plt.imshow(img_test)
```



Al comparar la imagen anterior con la original, se percibe que la zona que antes era roja ahora se muestra oscura (en color negro), dado que las intensidades de píxel de dicho canal se establecieron en cero. Asimismo, el fondo de la imagen que antes era blanco (el cual se forma con el valor máximo en cada canal, es decir Rojo=255, Verde=255 y azul=255), ahora se muestra en color cian, color

Prohibida su reproducción o difusión total o parcial | © 2023 |

que se obtiene al mezclar verde y azul (sin ningún aporte del canal rojo). Utilizando este ejemplo, se invita al lector a analizar la imagen eliminando el color verde, y luego eliminando solo el color azul.

OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de software de visión por ordenador y aprendizaje automático de código abierto[36], que está disponible a través de una licencia BSD. Esta librería dispone de más de 2500 algoritmos para tareas de visión por computador orientadas principalmente a aplicaciones en tiempo real como detección de rostros, identificación de objetos, clasificación, seguimiento y rastreo de objetos, modelos 3D, o fusión de datos,

OpenCV tiene interfaces en diferentes lenguajes como C++, Python, Java y MATLAB por lo cual es compatible con diversos sistemas operativos, a saber, Windows, Linux, Android y Mac OS. Para su utilización en Python, al igual que otras librerías es necesario importarla con el comando `import cv2`:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
```

Una vez importada, la librería dispone de métodos para operación con imágenes. Por ejemplo, para cargar una imagen bastará con utilizar el método `imread` y especificar la ruta y el nombre de archivo de la imagen. Es importante verificar que la ruta y el archivo existan, dado que OpenCV no mostrará ningún error o excepción si el archivo no existe.

```
img = cv2.imread('logo_umng.jpg')
```

La imagen leída con OpenCV se almacena como un arreglo multidimensional tipo NumPy, lo que se puede corroborar con el comando `type`:

```
type(img)
```

[36] https://opencv.org

Prohibida su reproducción o difusión total o parcial | © 2023 |

```
numpy.ndarray
```

Dado que es un arreglo multidimensional, es pertinente comprobar las dimensiones de la imagen a través del método `shape`:

```
img.shape
```

```
(625, 516, 3)
```

De esta forma, el archivo del ejemplo corresponde a una imagen de 322500 píxeles, estructurados en 625 filas y 516 columnas, y es una imagen a color, dado que tiene 3 canales.

Para visualizar la imagen, es posible utilizar el comando `imshow` de matplolib. Este comando muestra una imagen a color (RGB), desplegando en el primer canal el componente rojo (R), en el segundo canal el componente verde (G), y en el tercer canal el componente azul (B). El resultado para la imagen del ejemplo se muestra a continuación:

```
plt.imshow(img)
```

Al realizar este proceso, lo que se observa es que la imagen mostrada en pantalla se presenta en falso color. Esto sucede porque al trabajar con OpenCV, la variable que contiene el arreglo de la imagen se estructura ubicando en el primer canal el componente azul, en el segundo canal el componente verde y en el tercer canal el componente rojo, es decir con una distribución BGR. Para solventar esta situación, OpenCV permite convertir la imagen hacia RGB, y así poder utilizar un visualizador estándar como matplolib:

Prohibida su reproducción o difusión total o parcial | © 2023 |

```
img_fix = cv2.cvtColor(img,
                       cv2.COLOR_BGR2RGB)
plt.imshow(img_fix)
```

OpenCV también permite realizar diversas operaciones de procesamiento de imagen, desde el mismo momento de su lectura. Por ejemplo, es posible leer una imagen a color en escala de grises, lo que arrojará que nuestro arreglo multidimensional solo tenga un canal (es decir solo está conformado por filas y columnas):

```
img_gray = cv2.imread('logo_umng.jpg',
                      cv2.IMREAD_GRAYSCALE)
img_gray.shape
```

```
(625, 516)
```

```
plt.imshow(img_gray, cmap = 'gray')
```

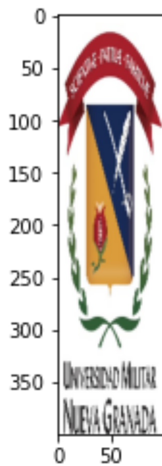
Prohibida su reproducción o difusión total o parcial | © 2023 |

Algunos ejemplos de operaciones comunes en procesamiento de imagen disponibles en OpenCV se muestran a continuación:

Redimensionado de imagen:

Opción 1: especificar la variable de la imagen a operar y las dimensiones deseadas:

```
img_new = cv2.resize(img_fix,
                    (100,400))
plt.imshow(img_new)
```



Opción 2: especificar la relación de la dimensión de la imagen tanto en alto como en ancho:

```
width_ratio = 2
height_ratio = 2
img2 = cv2.resize(img_fix,
                 (0,0),
                 img_fix,
                 width_ratio,
                 height_ratio)
plt.imshow(img2)
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

Reflejo de imagen

Con respecto al eje X (eje 0):

```
img_3 = cv2.flip(img_fix, 0)
plt.imshow(img_3)
```

Con respecto al eje Y (eje 1):

```
img_3 = cv2.flip(img_fix, 1)
plt.imshow(img_3)
```


Prohibida su reproducción o difusión total o parcial | © 2023 |

Con respecto a los dos ejes:

```
img_3 = cv2.flip(img_fix, -1)
plt.imshow(img_3)
```

Referencias

Brooks, R. A. (1979). The ACRONYM model-based vision system. *Proceedings of the 6th international joint conference on Artificial intelligence*, *1*, págs. 105-113.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence, 6*, 679-698.

Chan, T. F. (2001). Active contours without edges. *IEEE Transactions on image processing, 10(2)*, 266-277.

Chollet, F. (2021). *Deep learning with Python.* Simon and Schuster.

Prohibida su reproducción o difusión total o parcial | © 2023 |