Unit -3

Document Prepared by: Obula Raju (M.Tech, CNIS) ✅

Basics of Python:

Data Types

Data Type represent the type of data present inside a variable. In Python we are not required to specify the type explicitly. Based on value provided,the type will be assigned automatically.Hence Python is Dynamically Typed Language.

Python contains the following inbuilt data types

1. int

2. float

3.complex

4.bool

5.str

6.bytes

7.bytearray

8.range

9.list

10.tuple

11.set

12.frozenset

13.dict

14.None

- **Variable** → a name that stores a value in memory.
- No need to declare type explicitly (Python is **dynamically typed**).

```
# Variables
x = 10        # int
```

```
y = 3.14       # float

name = "Giri"   # str

is_active = True  # bool


print(type(x))   # <class 'int'>

print(type(y))   # <class 'float'>
```

Common Data Types:

- `int` → whole numbers (`10`, `-5`)
- `float` → decimal numbers (`3.14`, `-2.7`)
- `str` → text (`"hello"`, `'world'`)
- `bool` → True / False
- `None` → absence of value

2. Data Structures

Python has built-in **collections**:

```
# List (ordered, mutable)
fruits = ["apple", "banana", "cherry"]
fruits.append("mango")

# Tuple (ordered, immutable)
coordinates = (10, 20)

# Set (unordered, unique elements)
unique_numbers = {1, 2, 3, 2}

# Dictionary (key-value pairs)
person = {"name": "Giri", "age": 25}
print(person["name"])  # Giri
```

3. Control Flow Statements

Used to **control execution order**.

- **Loops:**

```
# for loop
for i in range(5):
```

```
    print(i)  # prints 0 to 4

# while loop
count = 0
while count < 5:
    print(count)
    count += 1
```

- **Loop control keywords:**
  - `break` → exits loop
  - `continue` → skips current iteration
  - `pass` → placeholder, does nothing

4. Conditional Statements

Used for **decision-making**.

```
x = 15

if x > 10:
    print("Greater than 10")
elif x == 10:
    print("Equal to 10")
else:
    print("Less than 10")
```

5. Functions

Functions help **reuse code**.

```
# Function definition
def greet(name):
    return f"Hello, {name}!"

# Function call
print(greet("Giri"))
```

- With **default arguments**:

```
def add(a, b=5):
    return a + b

print(add(10))      # 15
print(add(10, 20))  # 30
```

- With **multiple values** (`*args`, `**kwargs`):

```
def show(*args, **kwargs):
    print(args)      # tuple of values
    print(kwargs)    # dict of key-value pairs

show(1, 2, 3, name="Giri", age=25)
```

Scikit Image:

Uploading and Viewing an Image

```
from skimage import io
import skimage
#dir(skimage)
dir(io)
```

```
[ 'imread',
 'imsave',
 'imshow',
 'show',
.....]
```

```
from skimage import io
# Load image (supports jpg, png, tif, etc.)
img = io.imread("images/horses.jpg") #//local side
#Url path
#img =
io.imread("https://assets.bucketlistly.blog/sites/5adf778b6eabcc00190b75b1/assets/6075182186d092000b192cee/best-free-travel-images-image-2.jpg")
#img01=img.T


# Show image
io.imshow(img)
io.show()
```

```
otlib`, `napari`, etc. to visualize images.
  io.imshow(img)
/tmp/ipykernel_105595/1229699752.py:11: FutureWarning: `show` is deprecated
lib`, `napari`, etc. to visualize images.
  io.show()
```

Getting Image Resolution

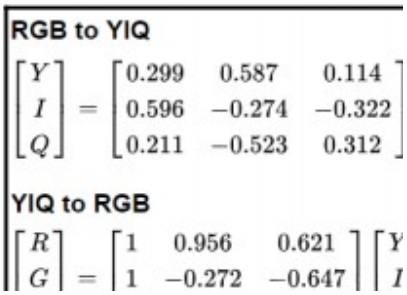
```
print("Image resolution:", img.shape)
# (height, width, channels)
```

Looking at Pixel Values

```
# Access a pixel (row=100, col=150, channel=RGB)
print("Pixel value:", img[100, 150])


# Change a pixel
img[100, 150] = [255, 0, 0]  # make it red
```

Converting Color Space

Some color spaces (channels)
The following represents a few popular channels/color spaces for an image: RGB, HSV, XYZ, YUV, YIQ, YPbPr, YCbCr, and YDbDr. We can use Affine mappings to go from one color space to another. The following matrix represents the linear mapping from the RGB to YIQ color space:



Some color spaces (channels)

The following represents a few popular channels/color spaces for an image: RGB, HSV, XYZ, YUV, YIQ, YPbPr, YCbCr, and YDbDr. We can use Affine mappings to go from one color space to another. The following matrix represents the linear mapping from the RGB to YIQ color space:

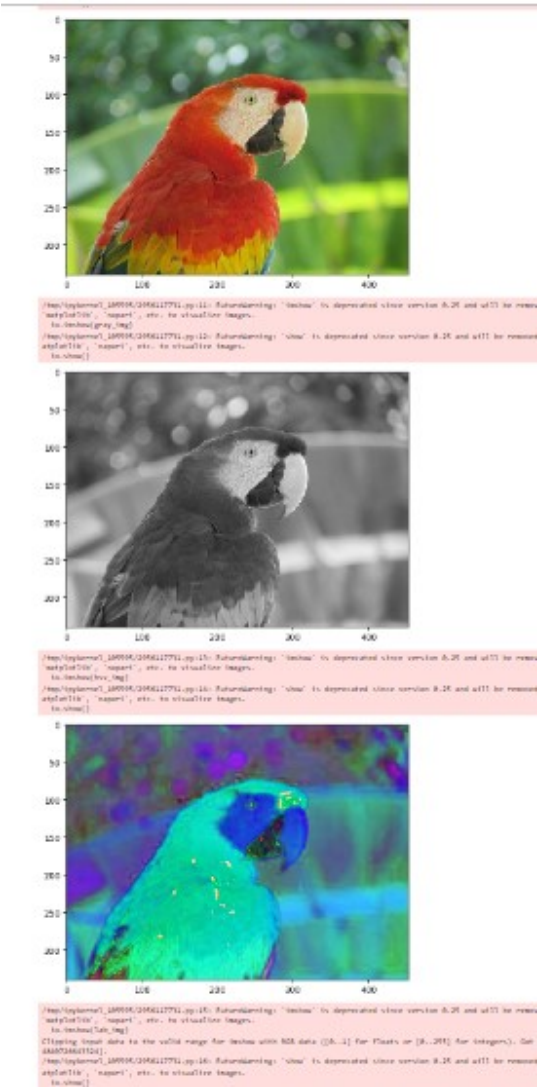
```
from skimage import color
dir(color)
['rgb2gray',
 'rgb2hed',
 'rgb2hsv',
.........
]
```

Converting from one color space into another

We can convert from one color space into another using library functions; for example, the following code converts an RGB color space into an HSV color space image:

```
from skimage import color
img = io.imread("images/parrot.jpg")


gray_img = color.rgb2gray(img)        # RGB → Grayscale
```

```
hsv_img = color.rgb2hsv(img)          # RGB → HSV
lab_img = color.rgb2lab(img)          # RGB → LAB


io.imshow(img)
io.show()
io.imshow(gray_img)
io.show()
io.imshow(hsv_img)
io.show()
io.imshow(lab_img)
io.show()
```



Saving an Image

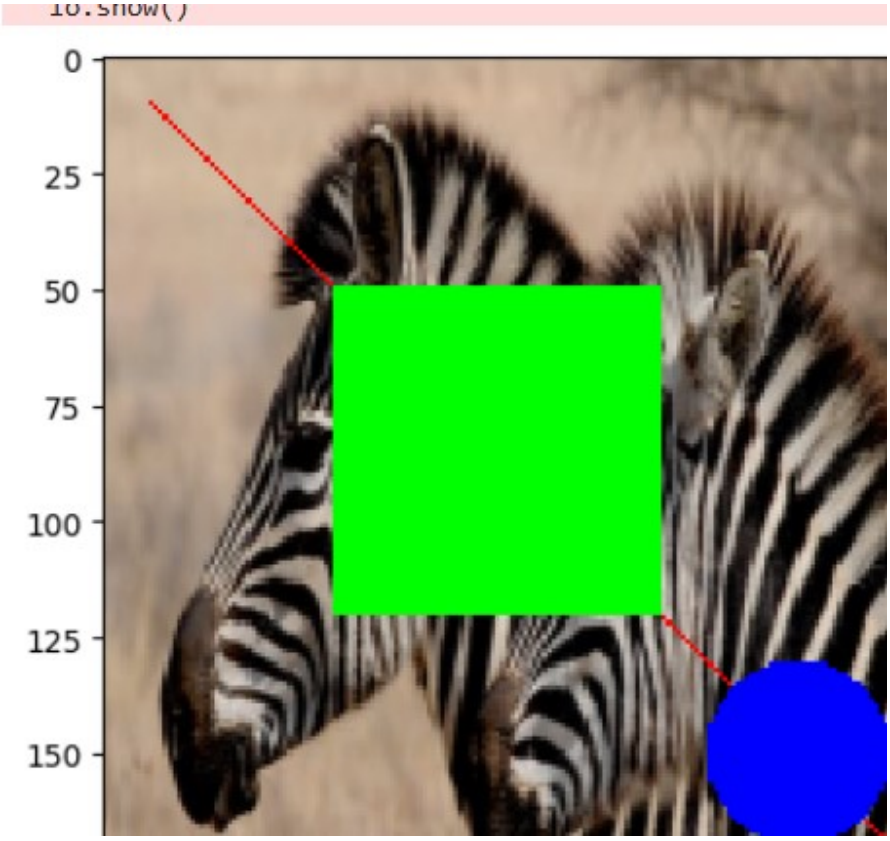
```
import numpy as np
from skimage import io

lab_norm = (lab_img - lab_img.min()) / (lab_img.max() - lab_img.min())  # scale to [0,1]
lab_uint8 = (lab_norm * 255).astype(np.uint8)

io.imsave("lab_img.jpg", lab_uint8)

or

from skimage import exposure, img_as_ubyte, io
# # Rescale lab_img to [0,1] before converting
lab_rescaled = exposure.rescale_intensity(lab_img, in_range='image', out_range=(0,1))
io.imsave("lab_img.jpg", img_as_ubyte(lab_rescaled))
```

Creating Basic Drawings

```
from skimage import draw
dir(draw)

['_bezier_segment',
 'bezier_curve',
 'circle_perimeter',
 'circle_perimeter_aa',
 'disk',
 'ellipse',
 'ellipse_perimeter',
 'ellipsoid',
 'ellipsoid_stats',
 'line',
 'line_aa',
 'line_nd',
 'polygon',
 'polygon2mask',
```

```
 'polygon_perimeter',
 'random_shapes',
 'rectangle',
 'rectangle_perimeter',
 'set_color']
```

```
import numpy as np
from skimage.draw import line, rectangle, disk
from skimage import io, transform

# Load background image
background = io.imread("images/zebras.jpg")   # <-- replace with your image path
background = transform.resize(background, (200, 200),
preserve_range=True).astype(np.uint8)

canvas = background.copy()

# Draw line
rr, cc = line(10, 10, 180, 180)
canvas[rr, cc] = [255, 0, 0]   # Red line

# Draw rectangle
rr, cc = rectangle(start=(50, 50), end=(120, 120))
canvas[rr, cc] = [0, 255, 0]   # Green rectangle

# Draw circle
rr, cc = disk((150, 150), 20)
canvas[rr, cc] = [0, 0, 255]   # Blue circle

# Show result
io.imshow(canvas)
io.show()
```



Doing Gamma Correction

POWER – LAW TRANSFORMATIONS:

There are further two transformation is power law transformations, that include nth power and nth root transformation. These transformations can be given by the expression:

$s=cr^{\gamma}$

This symbol γ is called gamma, due to which this transformation is also known as gamma transformation.
Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity

where c and g are positive constants. Sometimes Eq. (6) is written as $S = C (r +\varepsilon)^{\gamma}$ to account for an offset (that is, a measurable output when the input is zero). Plots of s versus r for various values of γ are shown in
Figure. As in the case of the log transformation, power-law curves with fractional values of γ map a narrow

range of dark input values into a wider range of output values, with the opposite being true for higher values
of input levels. Unlike the log function, however, we notice here a family of possible transformation curves
obtained simply by varying γ.
In Fig that curves generated with values of γ>1 have exactly The opposite effect as those generated
with values of γ<1. Finally, we Note that Eq. (6) reduces to the identity transformation when c=γ=1.

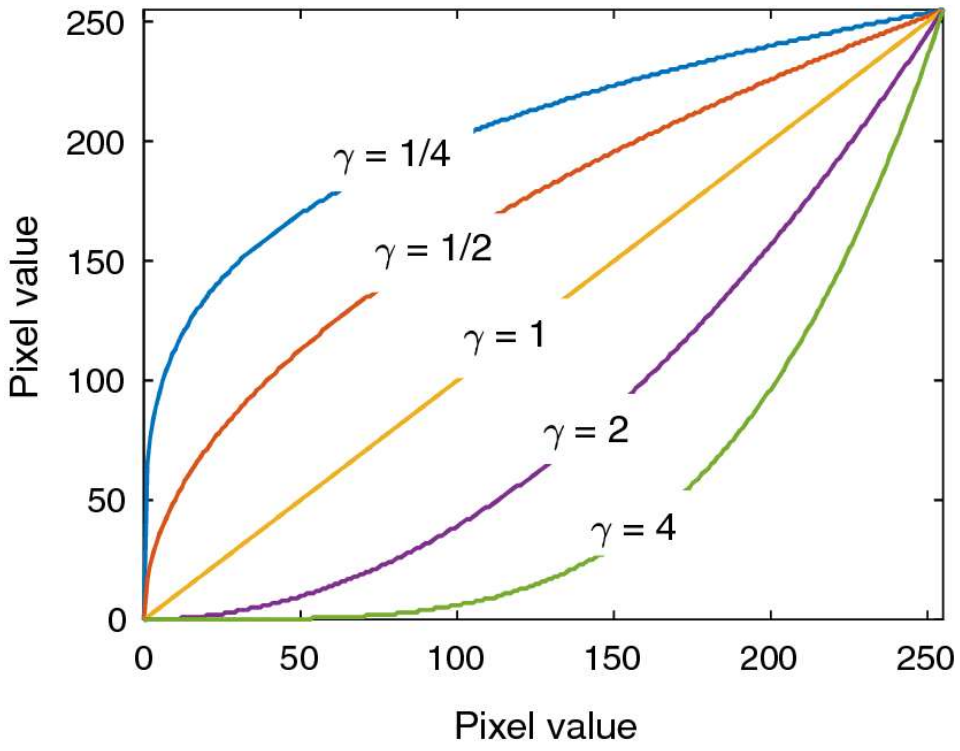
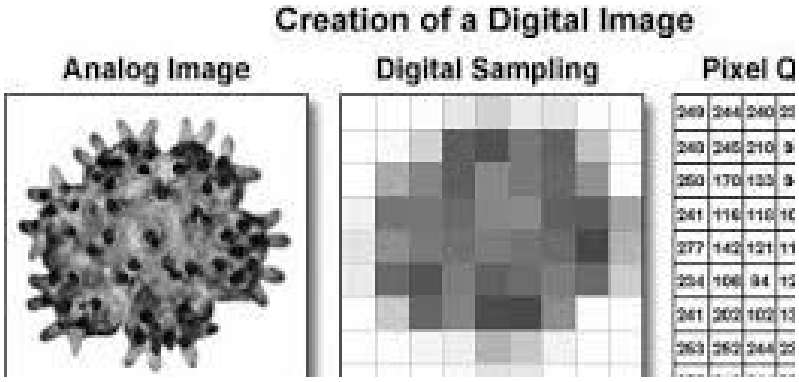


Fig: Plot of the equation $S = cr^{\gamma}$ for various values of γ (c =1 in all cases).
This type of transformation is used for enhancing images for different type of display devices. The
gamma of different display devices is different. For example Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.
Varying gamma (γ) obtains family of possible transformation curves $S = C^* r^{\gamma}$
Here C and γ are positive constants. Plot of S versusr for various values of γ
is γ > 1 compresses dark values
Expands bright values
γ< 1(similar to Logtransformation)
Expands dark values
Compresses bright values
When C = γ = 1 , it reduces to identity transformation .

```
from skimage import exposure


gamma_corrected = exposure.adjust_gamma(img, gamma=1)  # >1 = darker, <1 = brighter
io.imshow(gamma_corrected)
io.show()
```

Rotating, Shifting, and Scaling Images

Representing Digital Images:
The result of sampling and quantization is matrix of real numbers. Assume that an image f(x,y) is sampled so that the resulting digital image has M rows and N Columns. The values of the coordinates (x,y) now become discrete quantities thus the value of the coordinates at orgin become (X,y) =(0,0) The next Coordinates value along the first signify the image along the first row. It does not mean that these are the actual values of physical coordinates when the image was sampled.



$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0, \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1, \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(N \end{bmatrix}$$

Thus the right side of the matrix represents a digital element, pixel or pel. The matrix can be represented in the following form as well. The sampling process may be viewed as partitioning the
xy plane into a grid with the coordinates of the center of each grid being a pair of elements from the Cartesian products Z2 which is the set of all ordered pair of elements (Zi, Zj) with Zi and Zj being integers from Z. Hence f(x,y) is a digital image if gray level (that is, a real number from the
set of real number R) to each distinct pair of coordinates (x,y). This functional assignment is the quantization process. If the gray levels are also integers, Z replaces R, the and a digital image become a 2D function whose coordinates and she amplitude value are integers. Due to processing
storage and hardware consideration, the number gray levels typically is an integer power of2.
L=2k
Then, the number, b, of bites required to store a digital image is b=M *N* k When M=N, the equation become b=N2 *k
When an image can have 2k gray levels, it is referred to as “k- bit”. An image with 256 possible gray levels is called an “8- bit image” (256=28).

Rotating

In linear algebra, a **rotation matrix** is a transformation matrix that is used to perform a rotation in Euclidean space. For example, using the convention below, the matrix

Clock Wise

$$R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

Anti - Clock Wise

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

This rotates column vectors by means of the following matrix multiplication,

Says For ClockWise Rotation

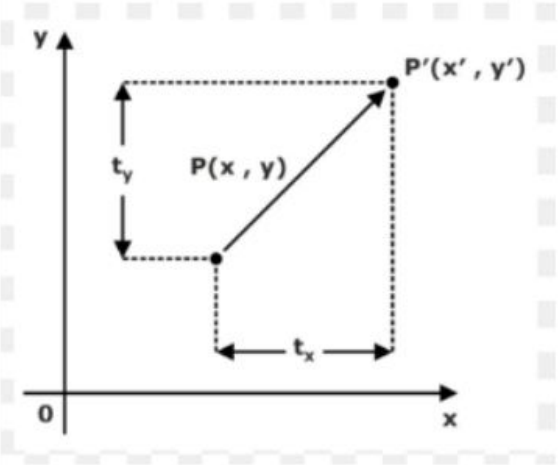
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}.$$

Shift An Image

$$\text{Matrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix}$$

Cal :



$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix}_{2\times 3} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}_{3\times 1}$$

Perform the multiplication:

- First row:

$$1 \times x + 0 \times y + a \times 1 = x + a$$

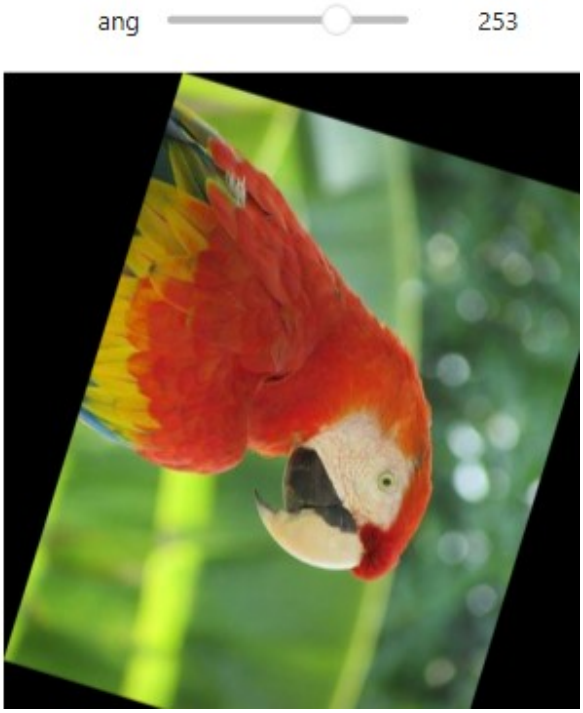
- Second row:

$$0 \times x + 1 \times y + b \times 1 = y + b$$

So the result is:

$$\begin{bmatrix} x + a \\ y + b \end{bmatrix}$$

```python
from skimage import transform

rotated = transform.rotate(img, angle=45)  # rotate 45°
shifted = transform.warp(img, transform.AffineTransform(translation=(50, 30)))
scaled = transform.rescale(img, 0.5, channel_axis=-1)  # 50% size
plt.imshow(rotated)
plt.axis('off')
plt.show()
```



Determining Structural Similarity (SSIM)

```
from skimage.metrics import structural_similarity as ssim
from skimage import io

# Load grayscale images
img1 = io.imread("images/parrot.jpg", as_gray=True)
img2 = io.imread("images/parrot.jpg", as_gray=True)

# Compute SSIM
score, diff = ssim(img1, img2, full=True, data_range=img1.max() - img1.min())

print("SSIM:", score)  # 1.0 = identical
```