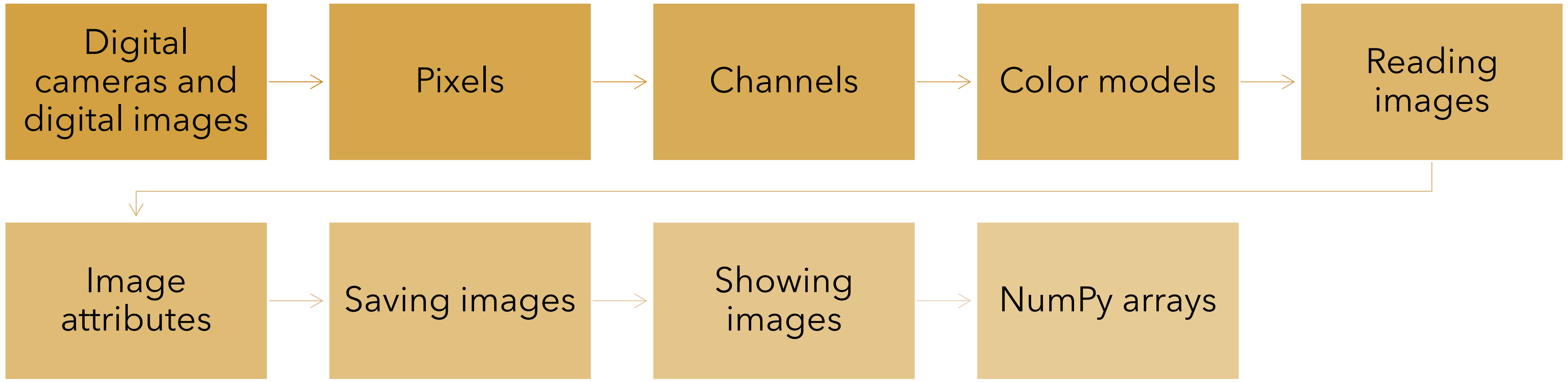
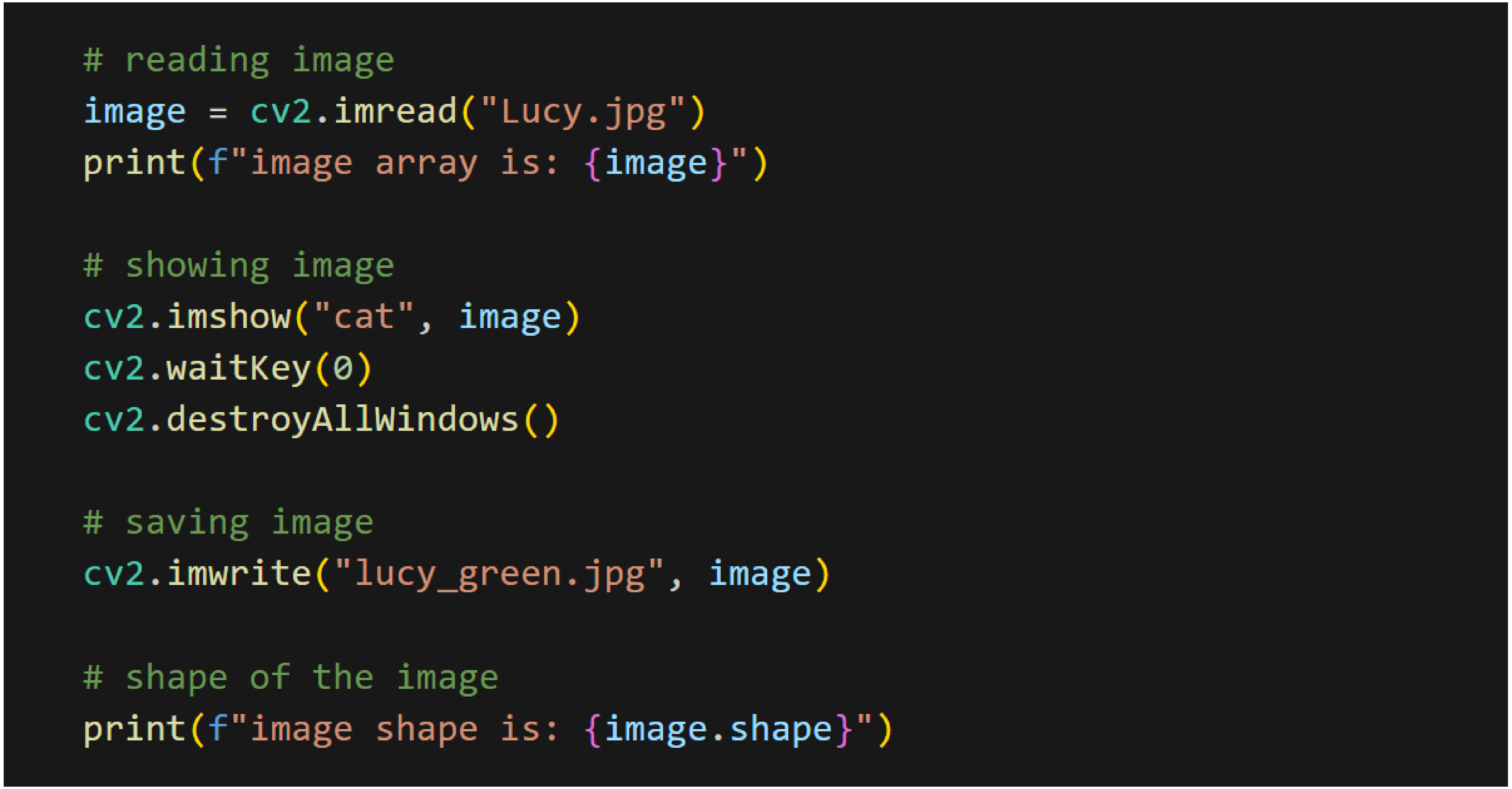
# Computer Vision

CVI620

Session 3
01/2025

## Overview


Digital cameras and digital images
Pixels
Channels
Color models
Reading images
Image attributes
Saving images
Showing images
NumPy arrays



```python
# reading image
image = cv2.imread("Lucy.jpg")
print(f"image array is: {image}")

# showing image
cv2.imshow("cat", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# saving image
cv2.imwrite("lucy_green.jpg", image)

# shape of the image
print(f"image shape is: {image.shape}")
```

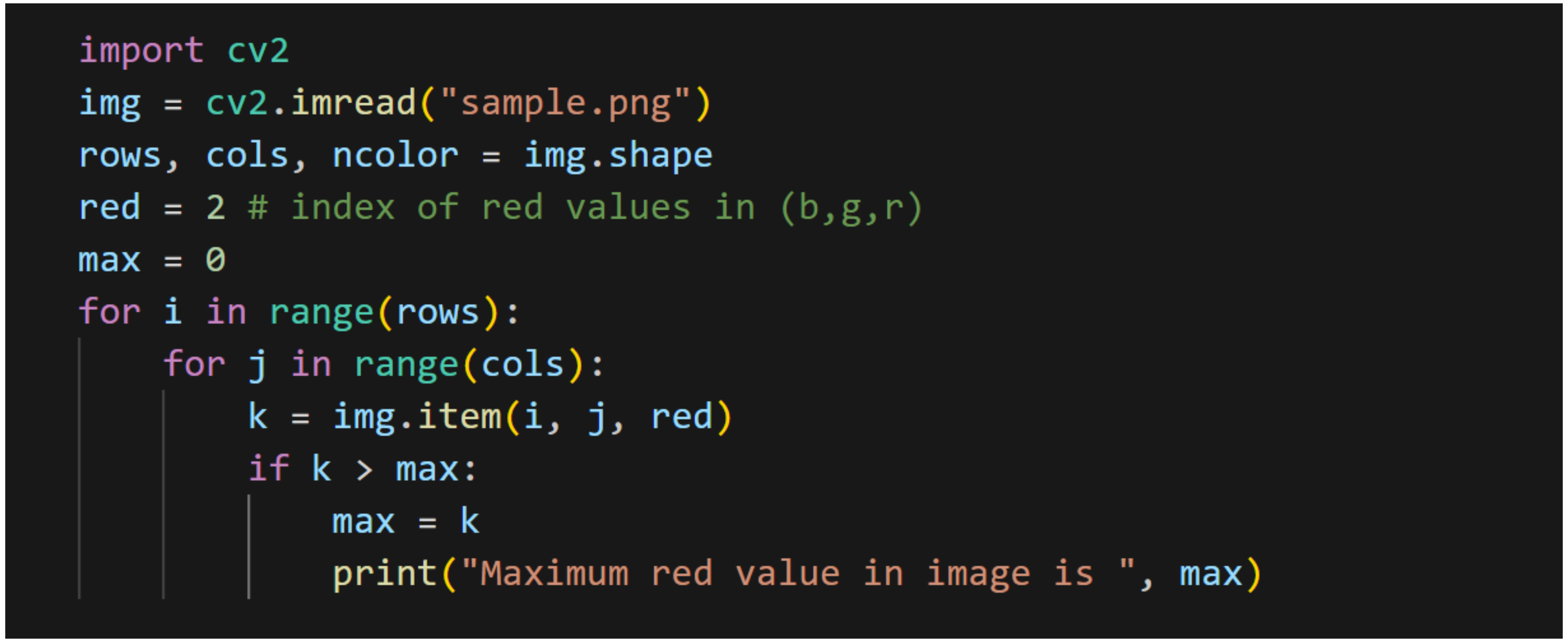
## Q & A?

- Python questions
- NumPy questions
- OpenCV questions
- non configuration issues
- Don’t have an idea!

# Min and Max

- Dynamic Range Adjustment: Normalizing pixel intensity for consistent brightness and contrast
- Thresholding: Setting thresholds for binary segmentation or feature detection.
- Error Detection: Identifying overexposed (max) or underexposed (min) areas in images.
- Feature Extraction: Detecting edges or regions based on intensity differences.
- Quality Control: Ensuring uniform intensity distribution in medical or industrial imaging.

## Finding Max Pixel Value



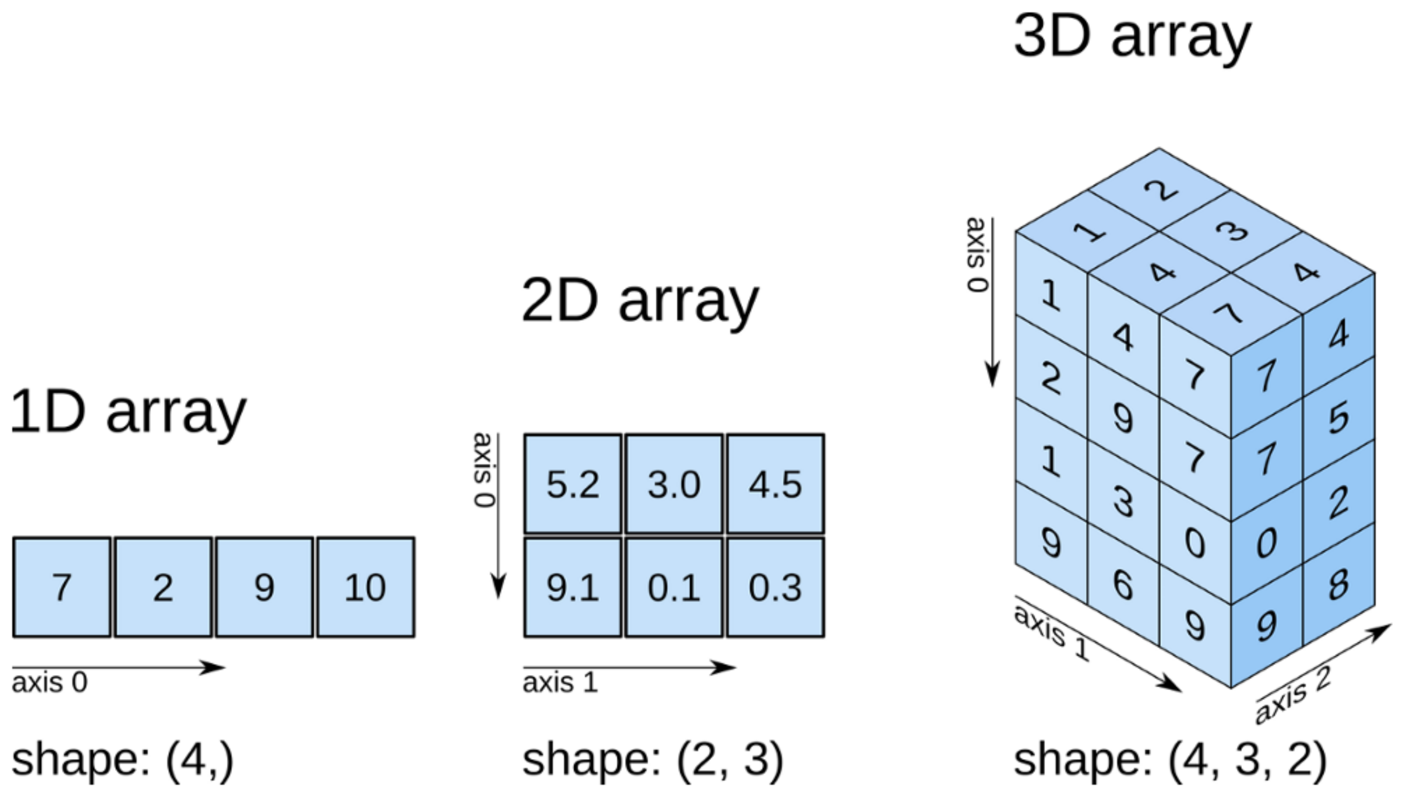
```python
import cv2
img = cv2.imread("sample.png")
rows, cols, ncolor = img.shape
red = 2 # index of red values in (b,g,r)
max = 0
for i in range(rows):
    for j in range(cols):
        k = img.item(i, j, red)
        if k > max:
            max = k
            print("Maximum red value in image is ", max)
```

## Finding Max Red Value

- What is a more efficient code to find the maximum red channel value, as we did in the previous slide?

# Numpy Coordinates

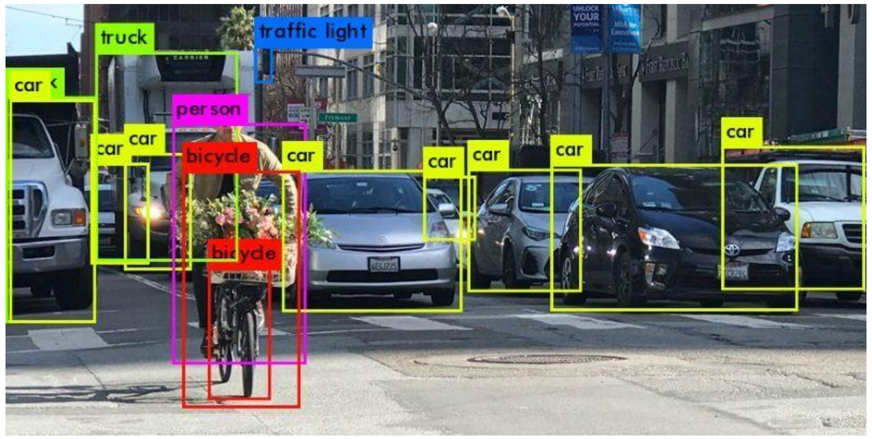
vector, matrix, tensor



## Region of Interest (ROI)

- A specific part of an image for focused processing
- Enhance efficiency by processing only the necessary area
- Common in object detection, image cropping, and analysis

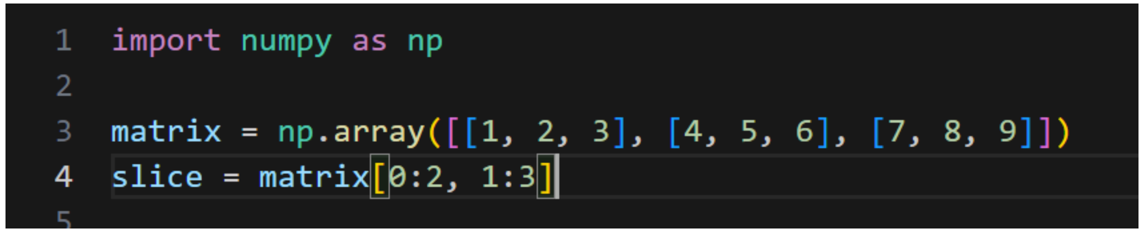
- uint8



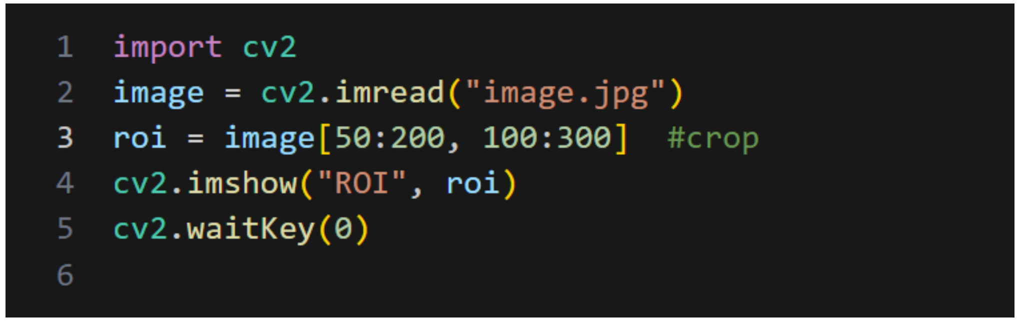
## Slicing

- Extracting a portion of a NumPy array

```
array[start:stop:step]
```



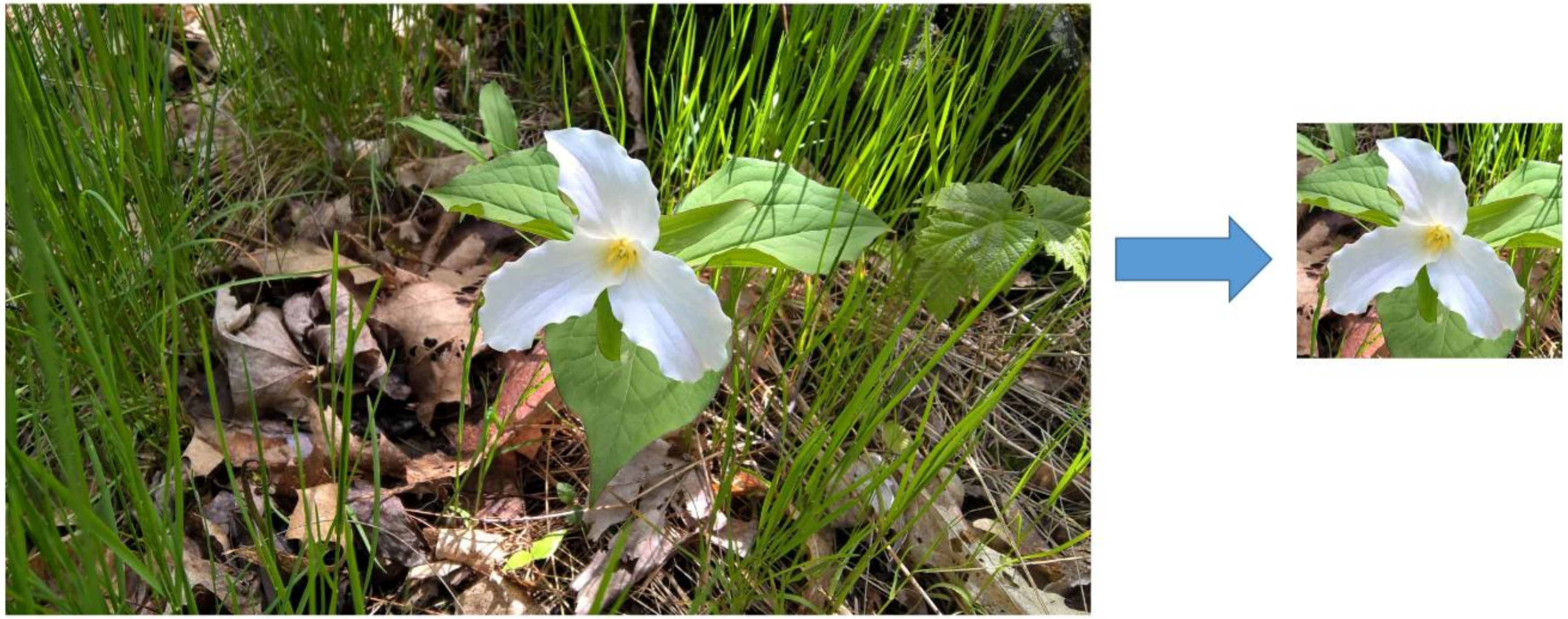
```python
import numpy as np

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
slice = matrix[0:2, 1:3]
```



```python
import cv2
image = cv2.imread("image.jpg")
roi = image[50:200, 100:300]  #crop
cv2.imshow("ROI", roi)
cv2.waitKey(0)
```

## Cropping

```
cropped = img[411:1560, 1700:3000]
```

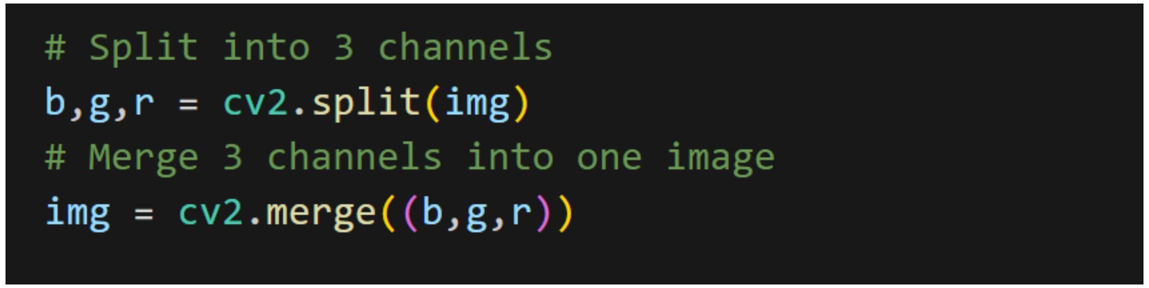


## Exercise

- Copy an ROI from your image, and paste it in another region

# Split and Merge

- Color Channel Processing: Isolating specific channels for analysis or filtering (enhancing the red channel in an image)
- Grayscale Conversion: Extracting a single channel to create a custom grayscale image
- Channel-Based Feature Extraction: Using specific channels to detect features like edges or color patterns
- Custom Image Composition: Replacing or modifying one channel and merging back to create new images.
- Color Space Conversion: Manipulating individual channels in spaces like HSV, YUV, etc.
- Image Masking: Applying operations to specific channels and merging them back for the final result.



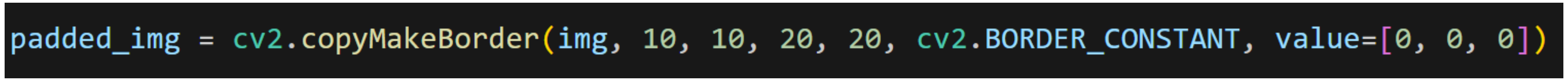
```
# Split into 3 channels
b,g,r = cv2.split(img)
# Merge 3 channels into one image
img = cv2.merge((b,g,r))
```

## Padding

- Preserve Dimensions: Maintain spatial size during convolutions in CNNs
- Data Augmentation: Enable cropping, shifting, or rotation without losing content
- Standardization: Resize images to uniform dimensions
- Object Detection: Keep bounding boxes within image boundaries
- Edge Processing: Avoid truncation of features at image borders
- Image Alignment: Align images of different sizes for stitching or other operations

## Padding

- src: It is the source image
- top: It is the border width in number of pixels in top direction
- bottom: It is the border width in number of pixels in bottom direction
- left: It is the border width in number of pixels in left direction
- right: It is the border width in number of pixels in right direction
- borderType: It depicts what kind of border to be added. It is defined by flags like cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc
- value: It is an optional parameter which depicts color of border if border type is cv2.BORDER_CONSTANT.



```
padded_img = cv2.copyMakeBorder(img, 10, 10, 20, 20, cv2.BORDER_CONSTANT, value=[0, 0, 0])
```

## BorderType

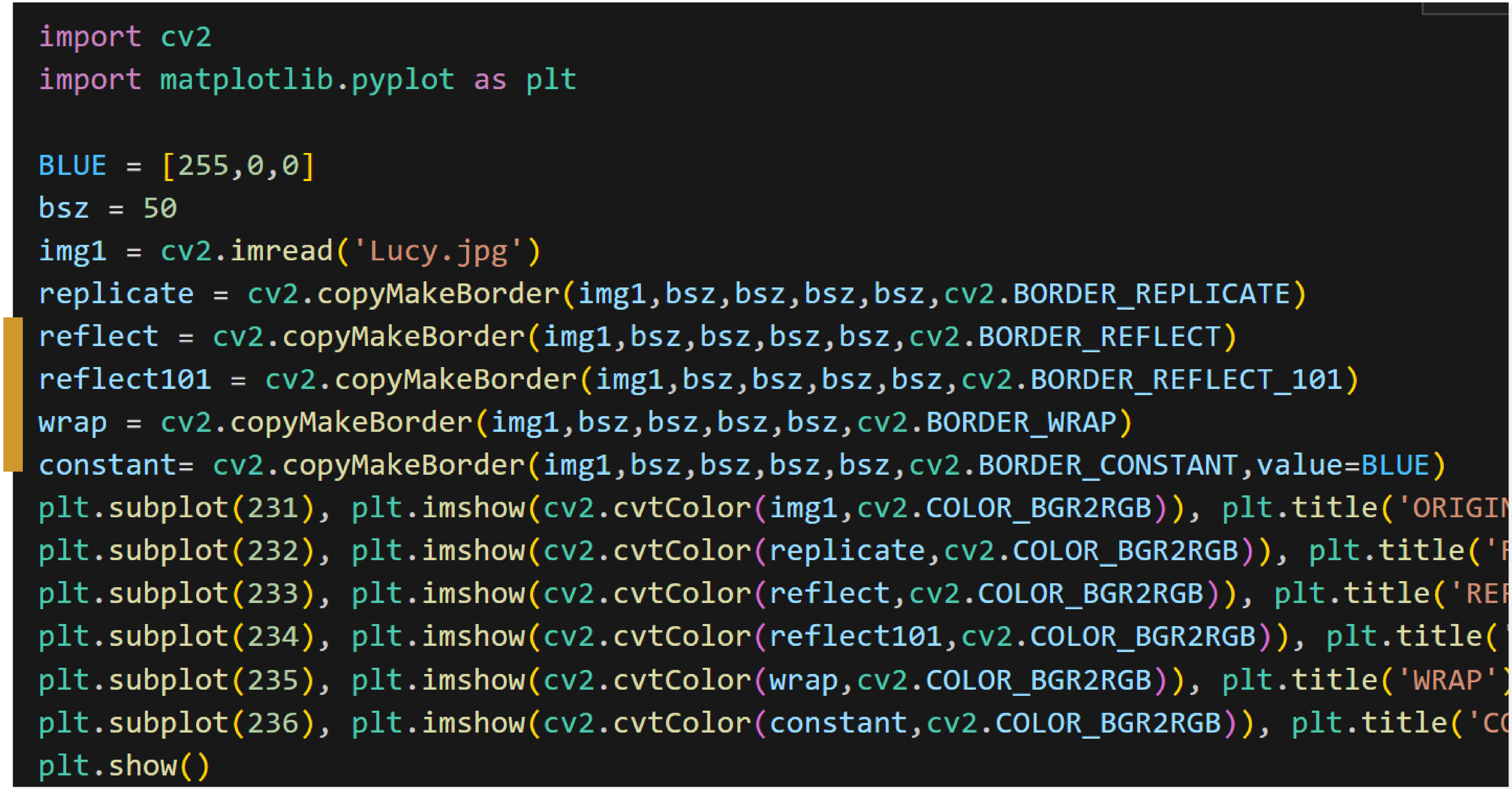
cv2.BORDER_CONSTANT: It adds a constant colored border. The value should be given as a keyword argument

cv2.BORDER_REFLECT: The border will be mirror reflection of the border elements. Suppose, if image contains letters “abcdefg” then output will be “gfedcba|abcdefg|gfedcba“.

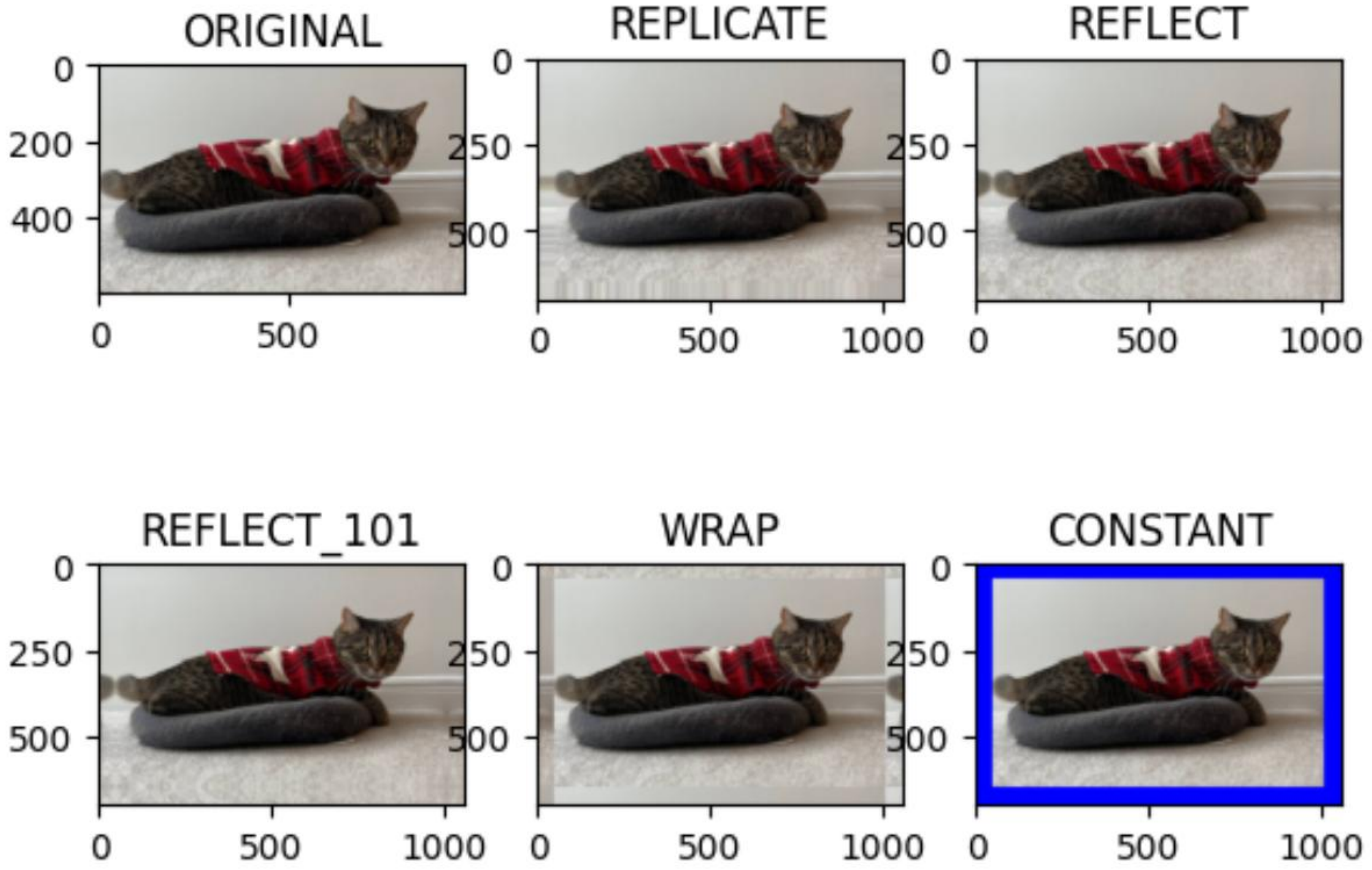
cv2.BORDER_REFLECT_101 or cv2.BORDER_DEFAULT: It does the same works as reflect

cv2.BORDER_REFLECT but with slight change. Suppose, if image contains letters “abcdefgh” then output will be “gfedcb|abcdefgh|gfedcba“.

cv2.BORDER_REPLICATE: It replicates the last element. Suppose, if image contains letters “abcdefgh” then output will be “aaaaa|abcdefgh|hhhhh“.



```
import cv2
import matplotlib.pyplot as plt

BLUE = [255,0,0]
bsz = 50
img1 = cv2.imread('Lucy.jpg')
replicate = cv2.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv2.BORDER_REPLICATE)
reflect = cv2.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv2.BORDER_REFLECT)
reflect101 = cv2.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv2.BORDER_REFLECT_101)
wrap = cv2.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv2.BORDER_WRAP)
constant= cv2.copyMakeBorder(img1,bsz,bsz,bsz,bsz,cv2.BORDER_CONSTANT,value=BLUE)
plt.subplot(231), plt.imshow(cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)), plt.title('ORIGIN
plt.subplot(232), plt.imshow(cv2.cvtColor(replicate,cv2.COLOR_BGR2RGB)), plt.title('R
plt.subplot(233), plt.imshow(cv2.cvtColor(reflect,cv2.COLOR_BGR2RGB)), plt.title('REF
plt.subplot(234), plt.imshow(cv2.cvtColor(reflect101,cv2.COLOR_BGR2RGB)), plt.title('
plt.subplot(235), plt.imshow(cv2.cvtColor(wrap,cv2.COLOR_BGR2RGB)), plt.title('WRAP')
plt.subplot(236), plt.imshow(cv2.cvtColor(constant,cv2.COLOR_BGR2RGB)), plt.title('CO
plt.show()
```


ORIGINAL
REPLICATE
REFLECT
REFLECT_101
WRAP
CONSTANT
0
200
400
250
500
1000

## Convert Channels

Changing the color format or extracting individual color channels.

```
cv2.cvtColor(image, conversion_code)
```

cv2.COLOR_BGR2GRAY – Convert to grayscale
cv2.COLOR_BGR2RGB – Convert to RGB
cv2.COLOR_BGR2HSV – Convert to HSV

```
b, g, r = cv2.split(image)  # Blue, Green, Red channels
```

```
merged_image = cv2.merge([b, g, r])
```

## Image Formats

- File types that store visual information in various ways.
- For communicating with Hardware. Like a "protocol".
- They determine how images are saved, compressed, and displayed.

- To balance quality and file size for different use cases.
- Specific formats cater to unique needs like transparency, animation, or high-resolution printing
- Optimize images for web, print, or storage purposes

- Storage formats (e.g., BMP, JPEG, PNG) compress and encode pixel data for efficient use.

## Format Examples

JPEG (JPG):

- Best for: Photographs and complex images.
- Features: Lossy compression (small size, lower quality at high compression).
- No transparency support.

PNG:

- Best for: Graphics, logos, and transparent images.
- Features: Lossless compression and transparency support.

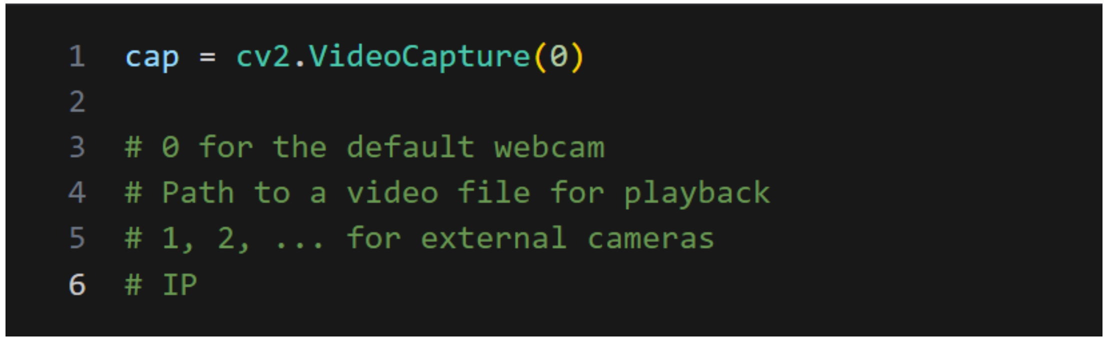
TIFF:

- Best for: High-quality printing and professional imaging.
- Features: Lossless or lossy compression, supports layers, and large file sizes.

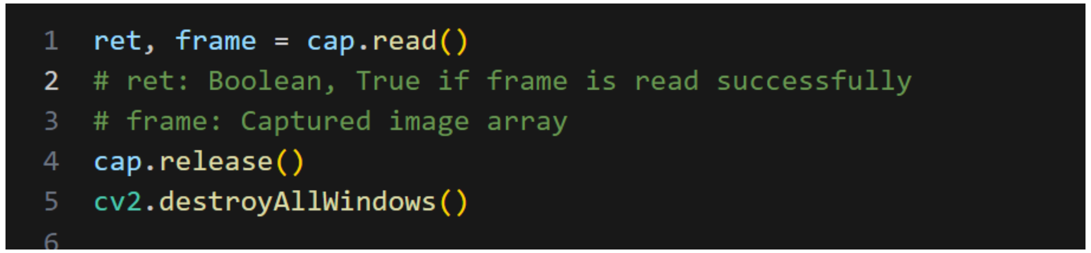
## Video

- Videos are sequences of images
- A class to capture video streams from:
- Webcam
- Video files (e.g., .mp4, .avi)
- IP cameras or other sources.

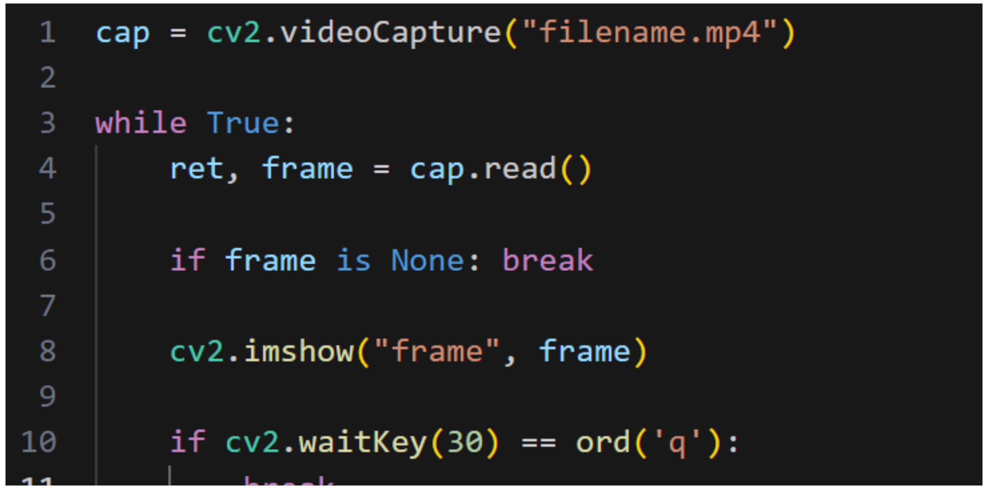
- object being created



```python
cap = cv2.VideoCapture(0)

# 0 for the default webcam
# Path to a video file for playback
# 1, 2, ... for external cameras
# IP
```

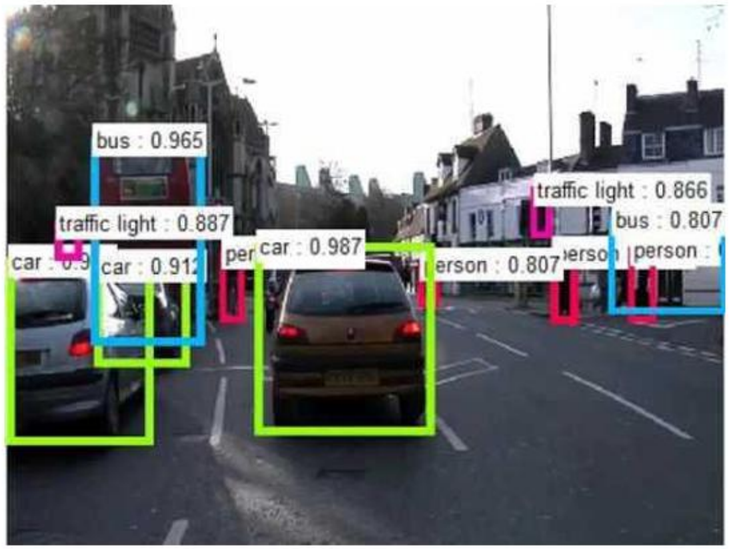


```python
ret, frame = cap.read()
# ret: Boolean, True if frame is read successfully
# frame: Captured image array
cap.release()
cv2.destroyAllWindows()
```



```python
cap = cv2.videoCapture("filename.mp4")

while True:
    ret, frame = cap.read()

    if frame is None: break

    cv2.imshow("frame", frame)

    if cv2.waitKey(30) == ord('q'):
```

# Drawing Shapes

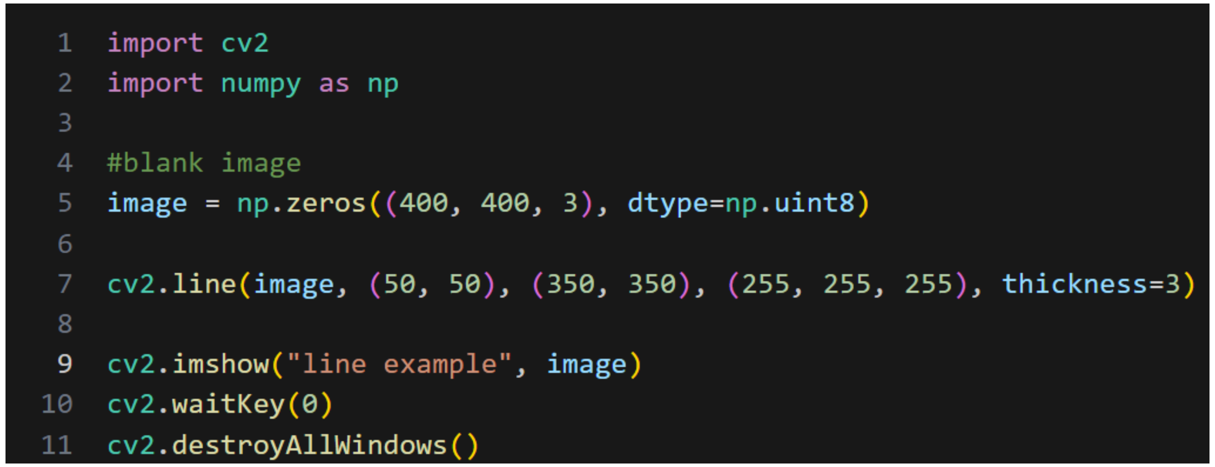


## Line

- draw a straight line on an image

cv2.line(image, pt1, pt2, color, thickness)

image: Input image where the line will be drawn
pt1: starting point (x1, y1) (W, H)
pt2: ending point (x2, y2)
color: line color in BGR format (e.g., (255, 0, 0) for blue)
thickness: line thickness (integer)



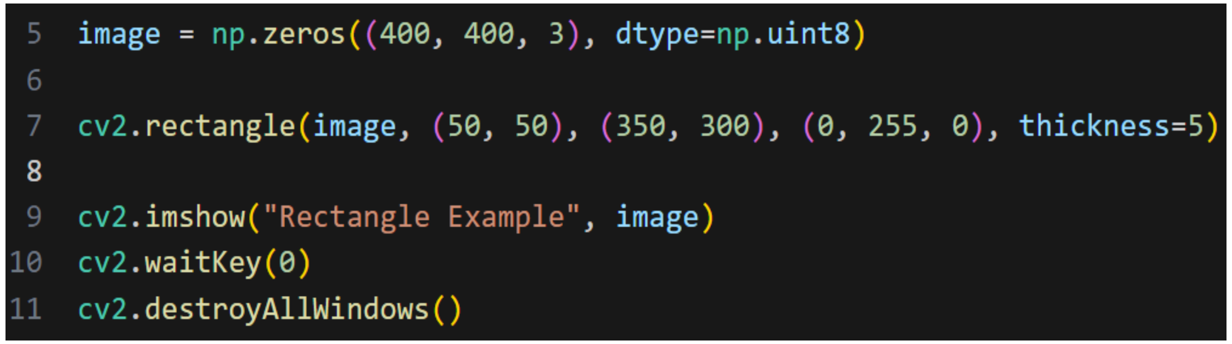
```python
import cv2
import numpy as np

#blank image
image = np.zeros((400, 400, 3), dtype=np.uint8)

cv2.line(image, (50, 50), (350, 350), (255, 255, 255), thickness=3)

cv2.imshow("line example", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Rectangle

- Draw rectangle on an image

`cv2.rectangle(image, pt1, pt2, color, thickness)`

image: Input image where the rectangle will be drawn.
pt1: Top-left corner (x1, y1).
pt2: Bottom-right corner (x2, y2).
color: Rectangle color in BGR format (e.g., (0, 255, 0) for green).
thickness: Border thickness (integer). Use -1 to fill the rectangle.



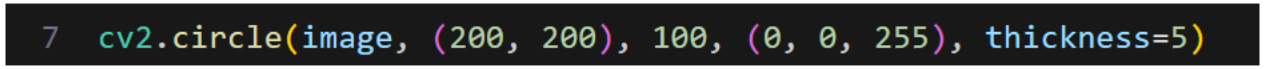
```python
image = np.zeros((400, 400, 3), dtype=np.uint8)

cv2.rectangle(image, (50, 50), (350, 300), (0, 255, 0), thickness=5)

cv2.imshow("Rectangle Example", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Circle

- Draw circle

cv2.circle(image, center, radius, color, thickness)

image: input image where the circle will be drawn
center: center of the circle (x, y)
radius: radius of the circle (integer)
color: circle color in BGR format (e.g., (0, 0, 255) for red)
thickness: circle thickness (integer). Use -1 for a filled circle



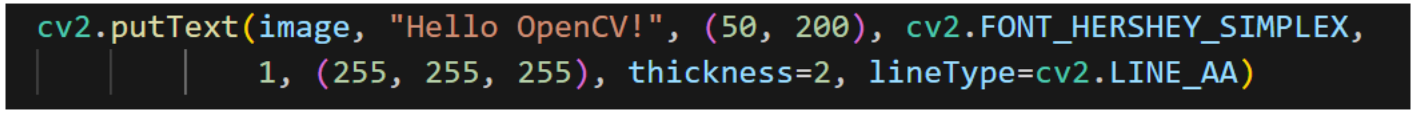
```python
cv2.circle(image, (200, 200), 100, (0, 0, 255), thickness=5)
```

# Text

- Add text on an image

cv2.putText(image, text, org, font, font_scale, color, thickness, line_type)

image: Input image where text will be added.
text: The string to display.
org: Bottom-left corner of the text (x, y).
font: Font type (e.g., cv2.FONT_HERSHEY_SIMPLEX).
font_scale: Scale factor for font size.
color: Text color in BGR format (e.g., (255, 255, 255) for white).
thickness: Thickness of the text stroke.
line_type: Type of line for the text (e.g., cv2.LINE_AA).

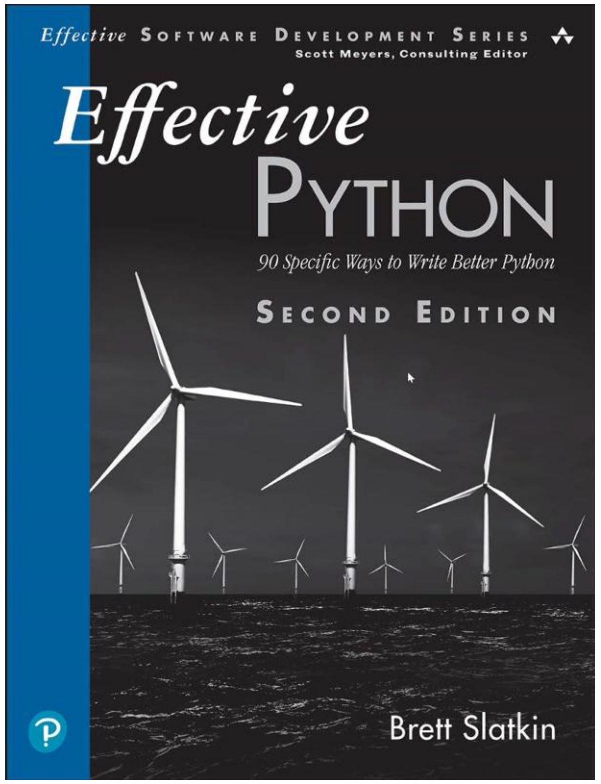


```
cv2.putText(image, "Hello OpenCV!", (50, 200), cv2.FONT_HERSHEY_SIMPLEX,
            1, (255, 255, 255), thickness=2, lineType=cv2.LINE_AA)
```

## More shapes

- cv2.line()
- cv2.rectangle()
- cv2.circle()
- cv2.ellipse()
- cv2.polylines()
- cv2.fillPoly()
- cv2.putText()
- cv2.arrowedLine()
- cv2.drawMarker()

# PEP8 Standard


Effective Software Development Series
Scott Meyers, Consulting Editor
Effective
Python
90 Specific Ways to Write Better Python
Second Edition
Brett Slatkin