Computer Vision

CVI620

Session 4

Overview

- Matplotlib
- Shallow/Deep Copy
- Min Max
- ROI, Slicing, Cropping
- Split, Merge
- Image attributes

Day	Date	Agenda/Topic	Reading(s)	Due
Tue	5/6	- Welcome and course overview - Introduction to Computer Vision and Imaging Systems: What is Computer Vision, Applications in real-world systems, etc. - Roadmap of the field - Human vision and Cameras - Installing prerequisites and system configurations. VSCode, Python, Virtualenv, NumPy, Pandas, OpenCV, matplotlib, ipykernel, Git.	- Install prerequisites and configurations	
Thu	5/8	- Digital Cameras and Images - Pixels, resolution, image size and shape - Color models: Binary, Grayscale, RGB, HSV, etc.		
Tue	5/13	- Introduction to NumPy library and arrays - Introduction to matplotlib - Introduction to OpenCV: reading, displaying, and saving images. - Image Formats: PNG, JPEG (JPG), TIFF, etc. - Image Coordinates	Lab 1	May 13
Thu	5/15	- Basic image operations: slicing, crop, split, merge, min & max - Basic image operations: rotate, padding, color model conversion - Drawing on images - PEP8 standard	Lab 2	May 15

Agenda

Padding

Drawing on images: line, rectangle, circle

Annotations

Video, FPS

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

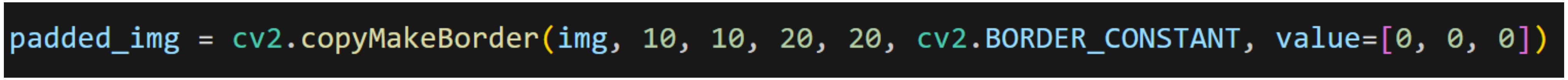
Object Detection: Keep bounding boxes within image boundaries

Edge Processing: Avoid truncation of features at image borders

Image Alignment: Align images of different sizes for stitching or other operations

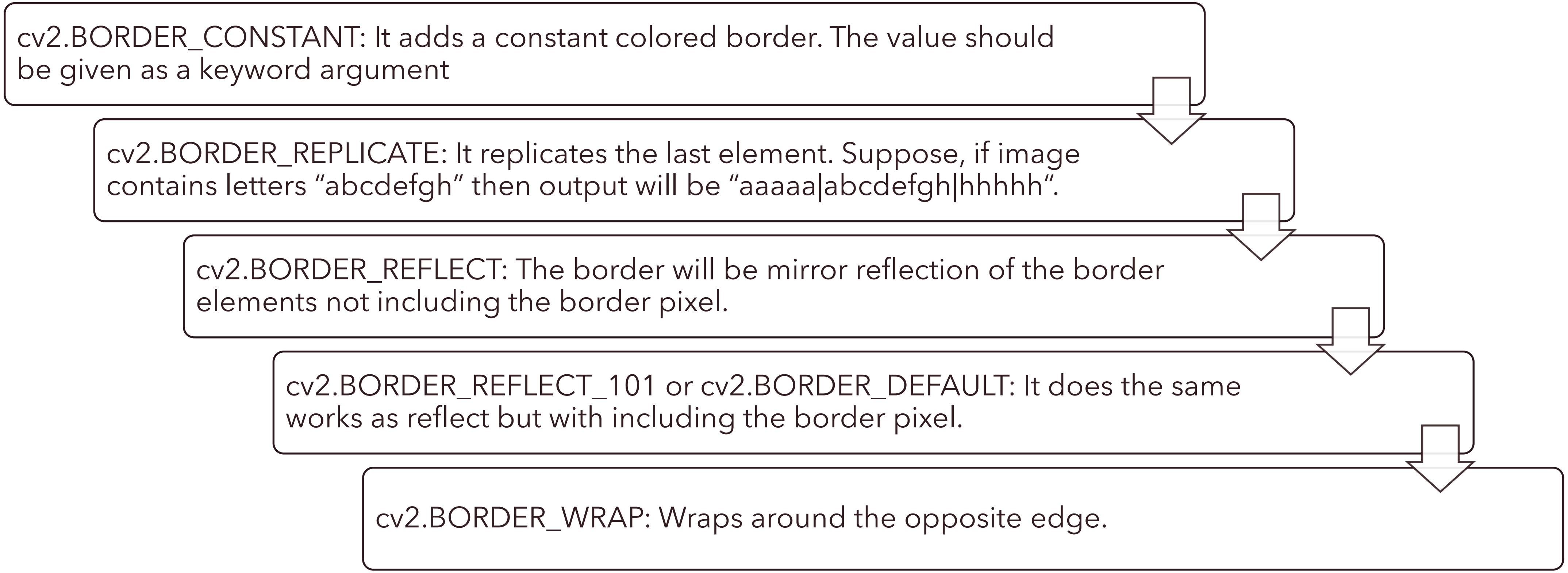
Padding

- 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



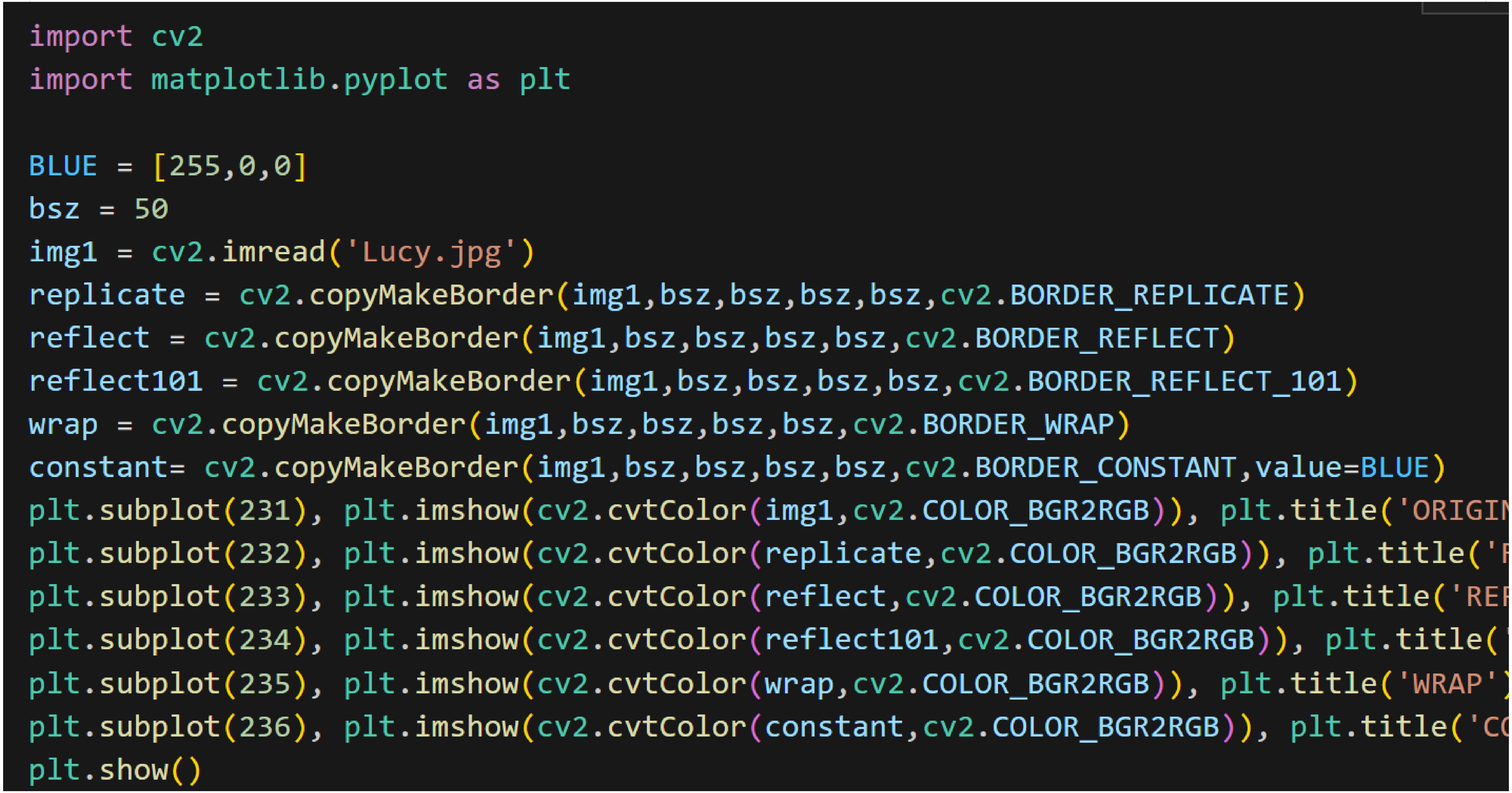
cv2.BORDER_CONSTANT: It adds a constant colored border. The value should be given as a keyword argument

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

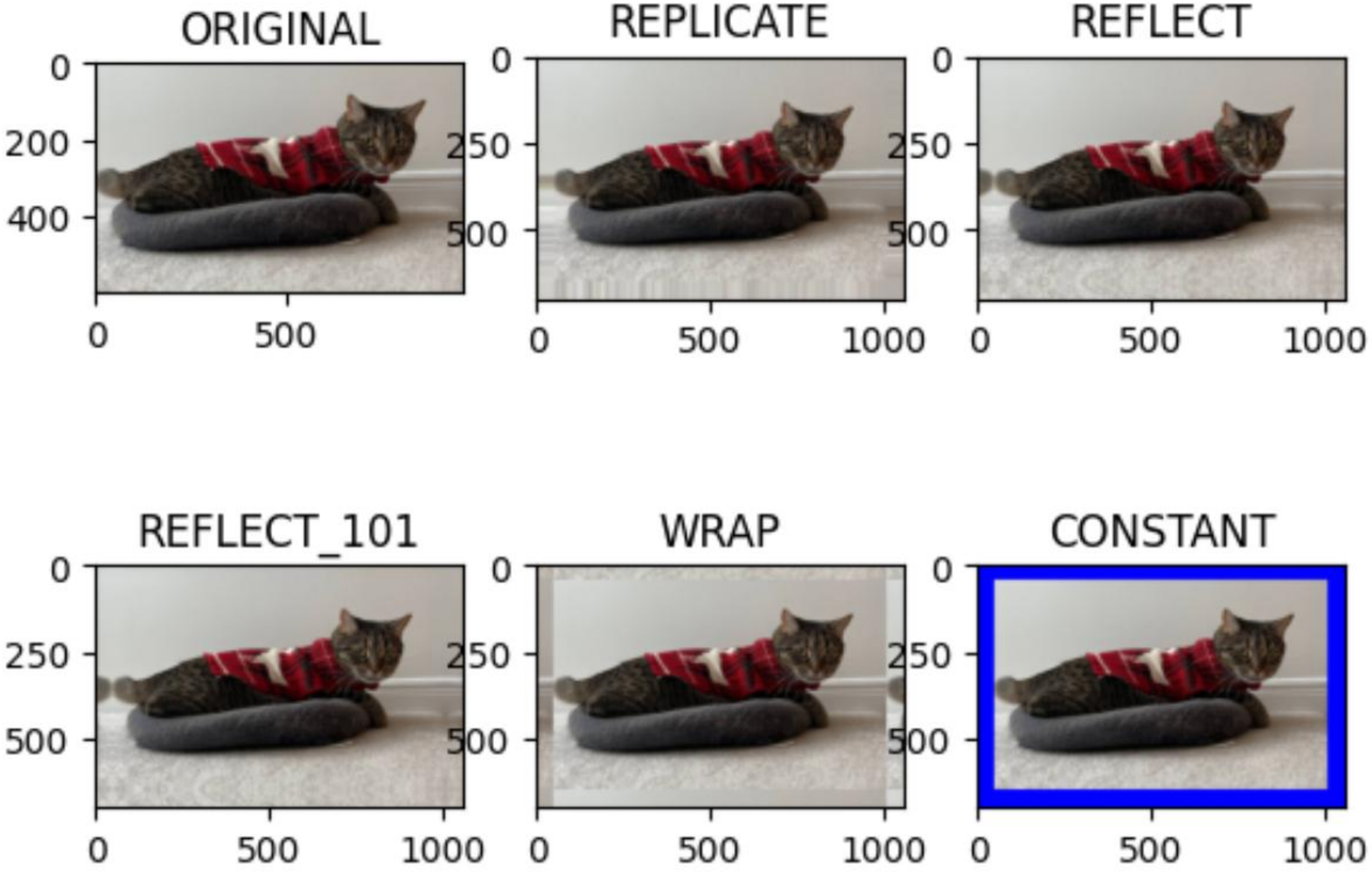
cv2.BORDER_REFLECT: The border will be mirror reflection of the border elements not including the border pixel.

cv2.BORDER_REFLECT_101 or cv2.BORDER_DEFAULT: It does the same works as reflect but with including the border pixel.

cv2.BORDER_WRAP: Wraps around the opposite edge.



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

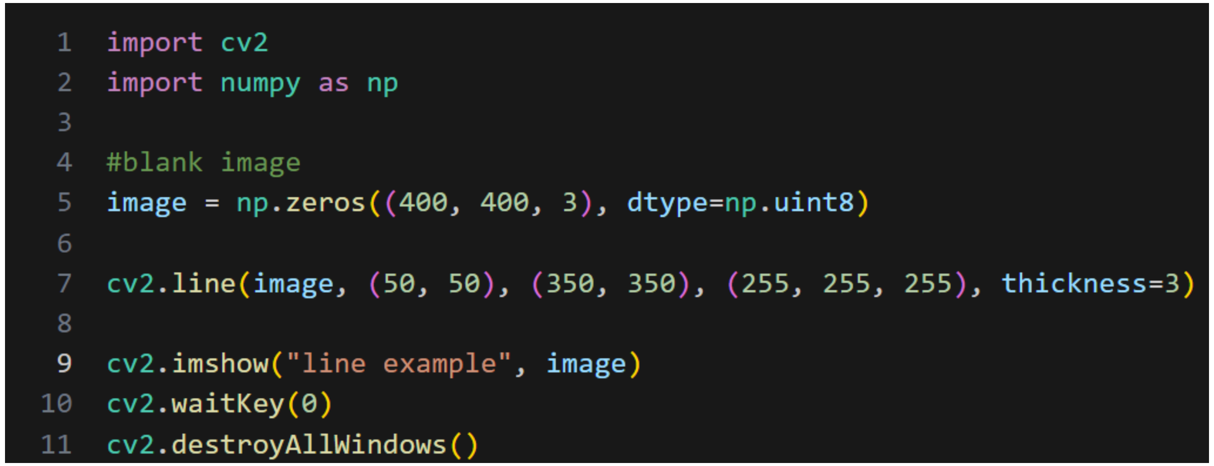
Drawing Shapes

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)



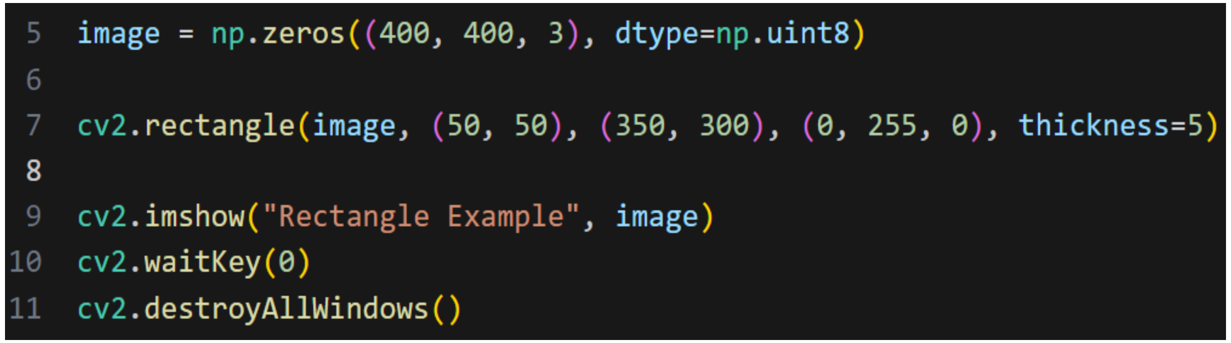
```
import cv2
import numpy as np

#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.



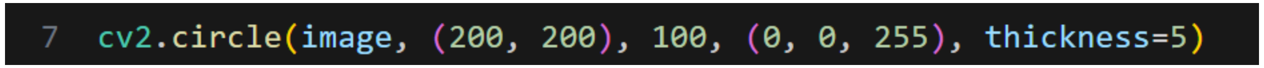
```python
image = np.zeros((400, 400, 3), dtype=np.uint8)

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



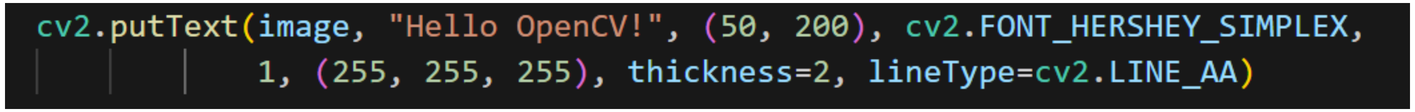
```
cv2.circle(image, (200, 200), 100, (0, 0, 255), thickness=5)
```

Text

- Add text on an image

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

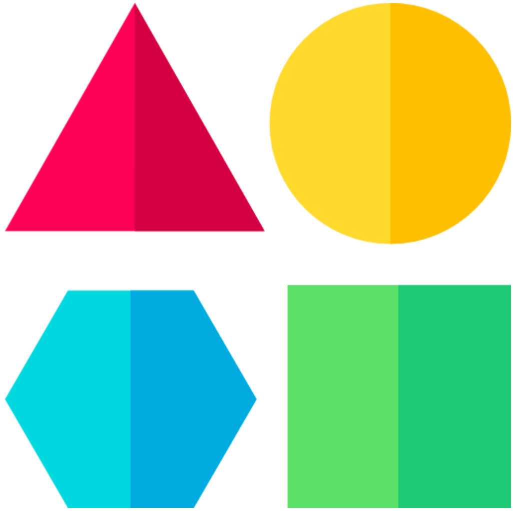
- image: Input image where text will be added.
- text: The string to display.
- org: Bottom-left corner of the text (x, y).
- font: Font type (e.g., cv2.FONT_HERSHEY_SIMPLEX).
- font_scale: Scale factor for font size.
- color: Text color in BGR format (e.g., (255, 255, 255) for white).
- thickness: Thickness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

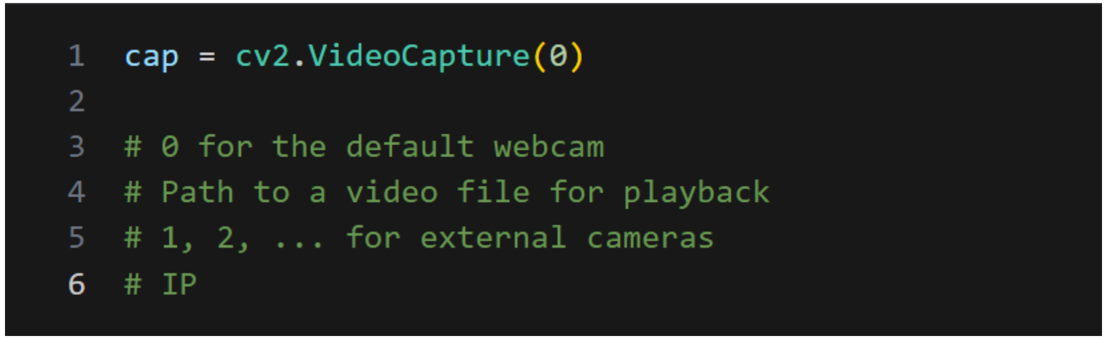
- cv2.line()
- cv2.rectangle()
- cv2.circle()
- cv2.ellipse()
- cv2.polylines()
- cv2.fillPoly()
- cv2.putText()
- cv2.arrowedLine()
- cv2.drawMarker()



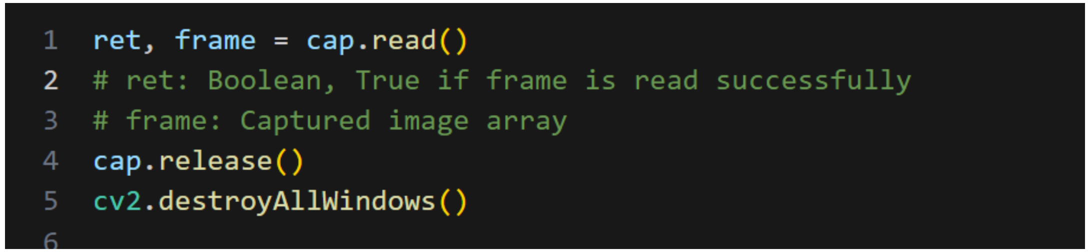
Video

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

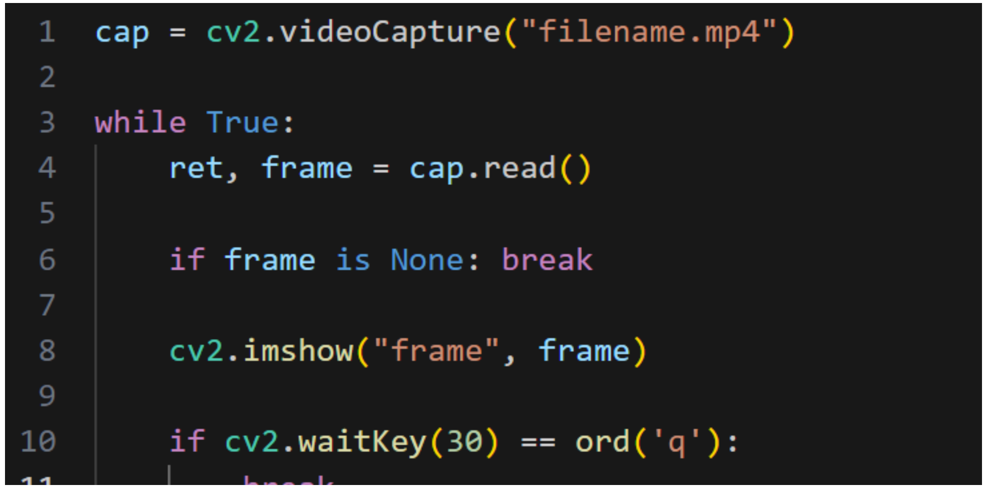
- object being created
- waitKey for speed



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

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

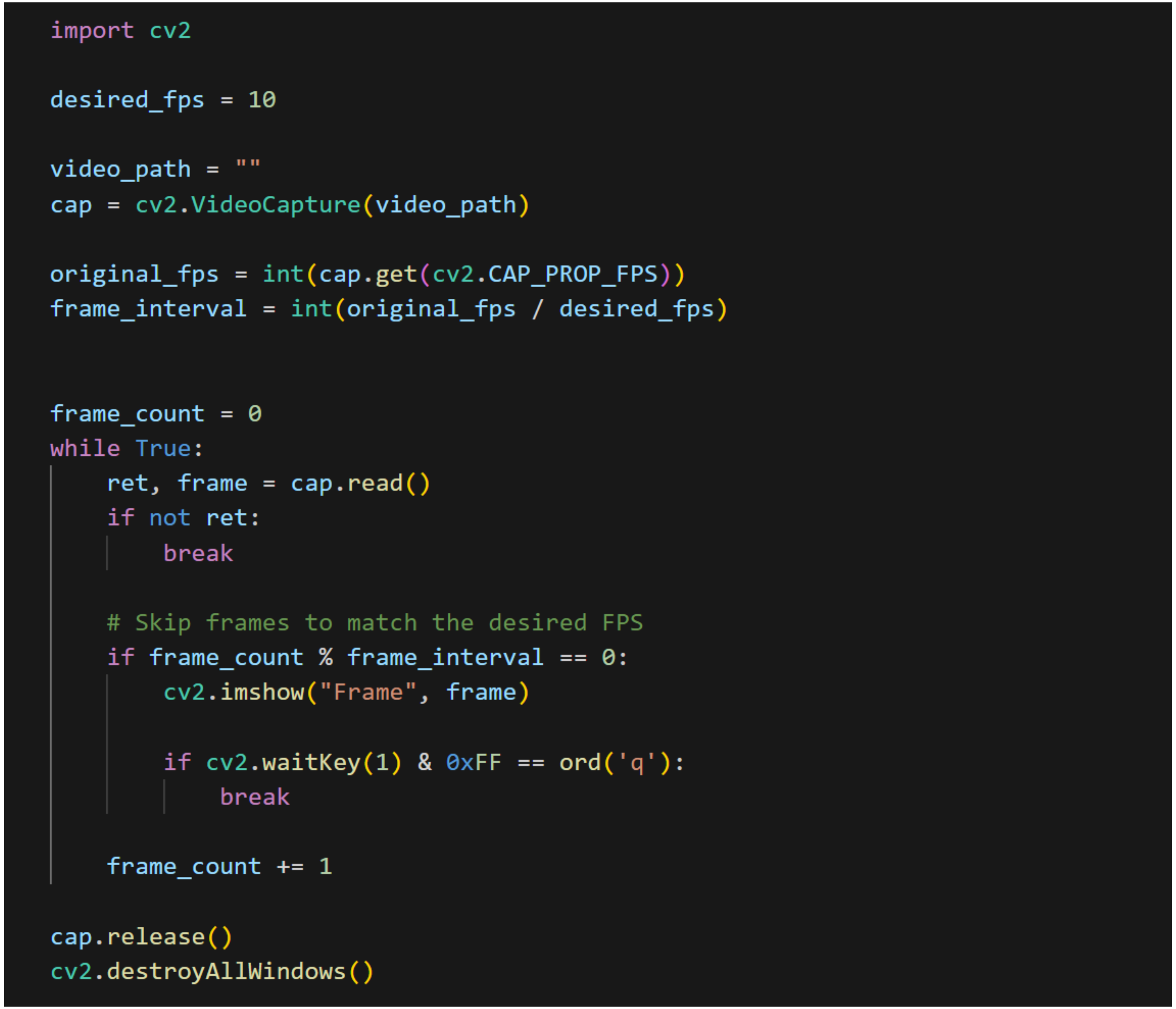


```python
ret, frame = cap.read()
# ret: Boolean, True if frame is read successfully
# frame: Captured image array
cap.release()
cv2.destroyAllWindows()
```



```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'):
        break
```

FPS



```python
import cv2

desired_fps = 10

video_path = ""
cap = cv2.VideoCapture(video_path)

original_fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_interval = int(original_fps / desired_fps)


frame_count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Skip frames to match the desired FPS
    if frame_count % frame_interval == 0:
        cv2.imshow("Frame", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    frame_count += 1

cap.release()
cv2.destroyAllWindows()
```