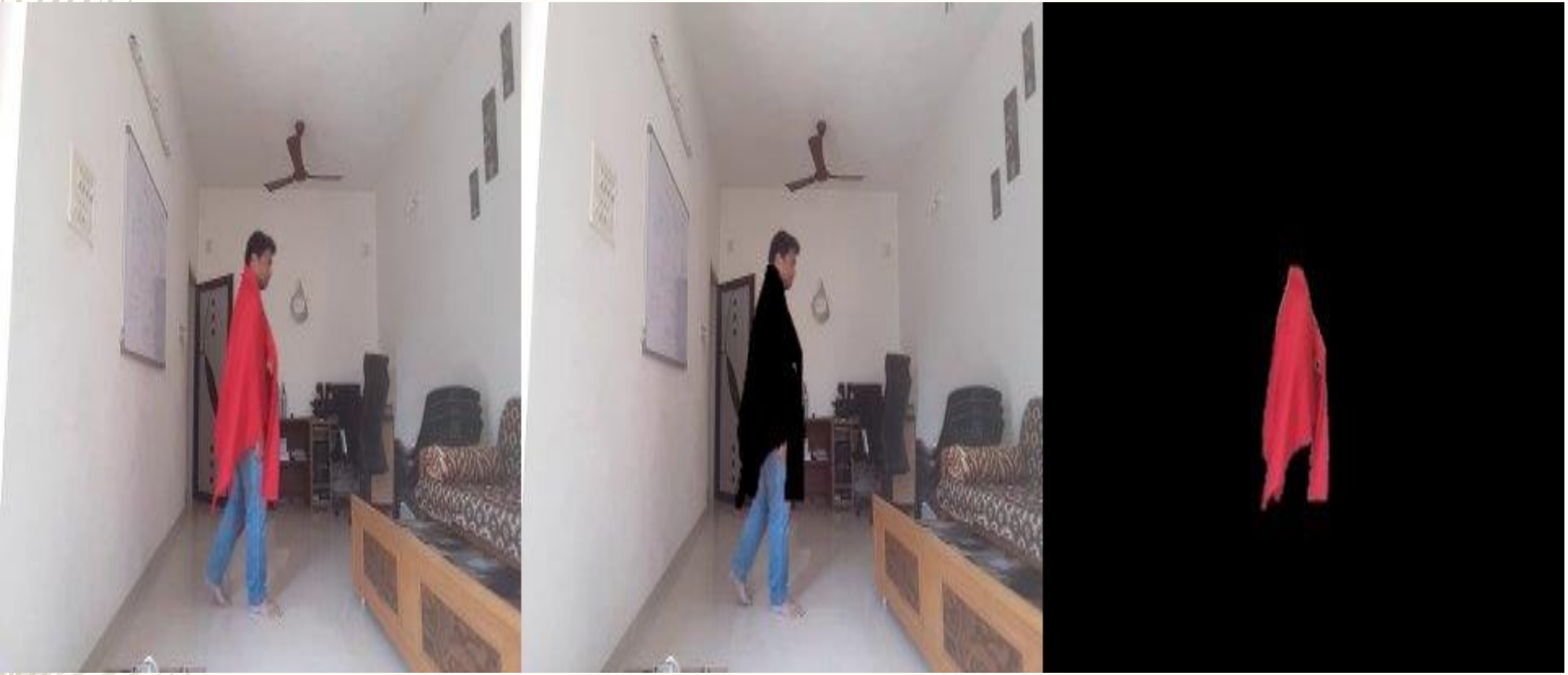
Invisibility Cloak Using Python

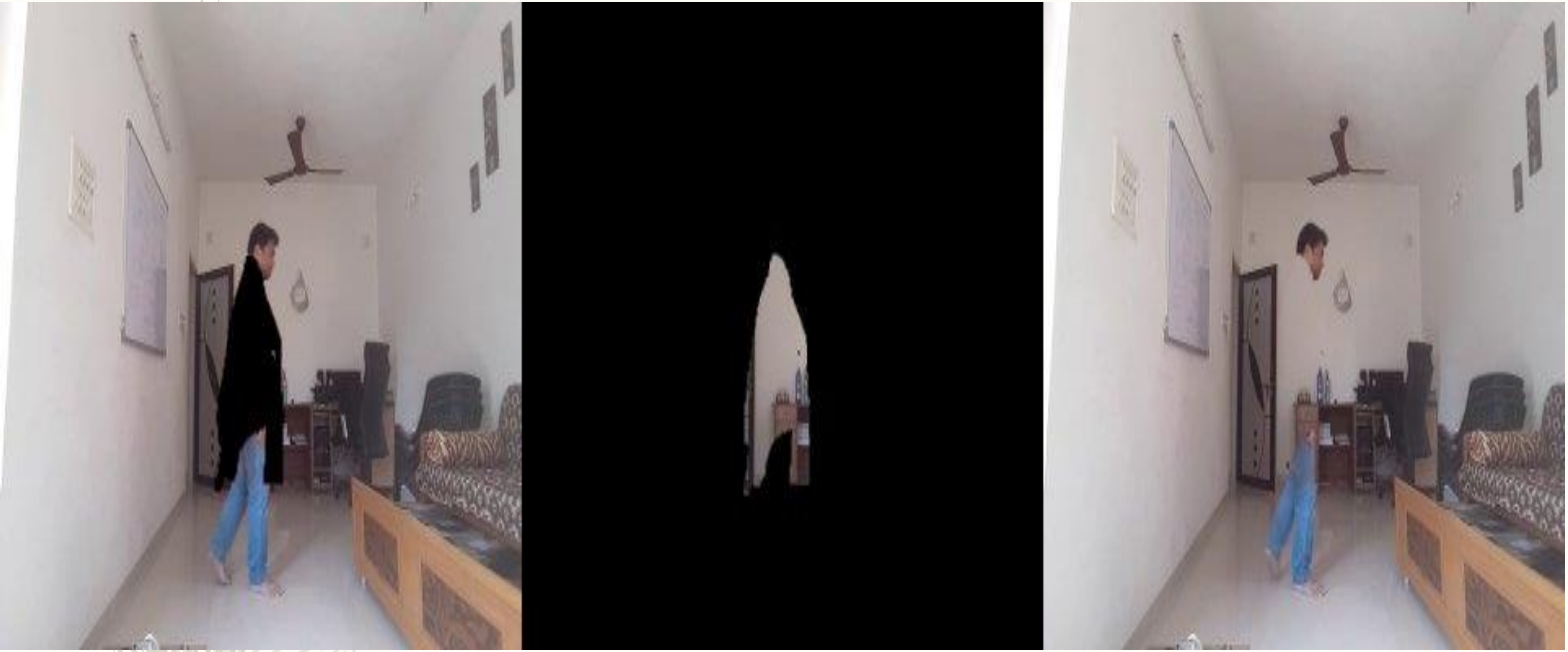
What Libraries do we need

- OpenCV
- Numpy

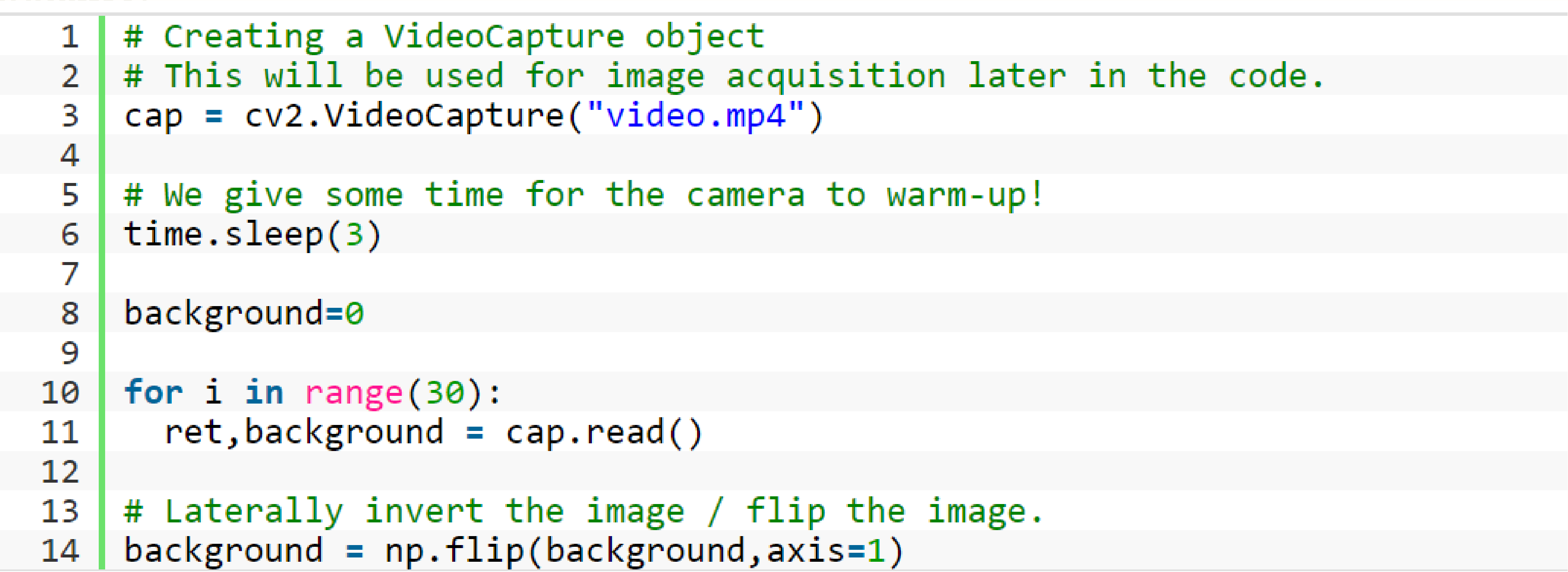
The steps

- Capture and store the background frame.
- Detect the colored cloth using color detection algorithm.
- Segment out the colored cloth by generating a mask.
- Generate the final augmented output to create the magical effect



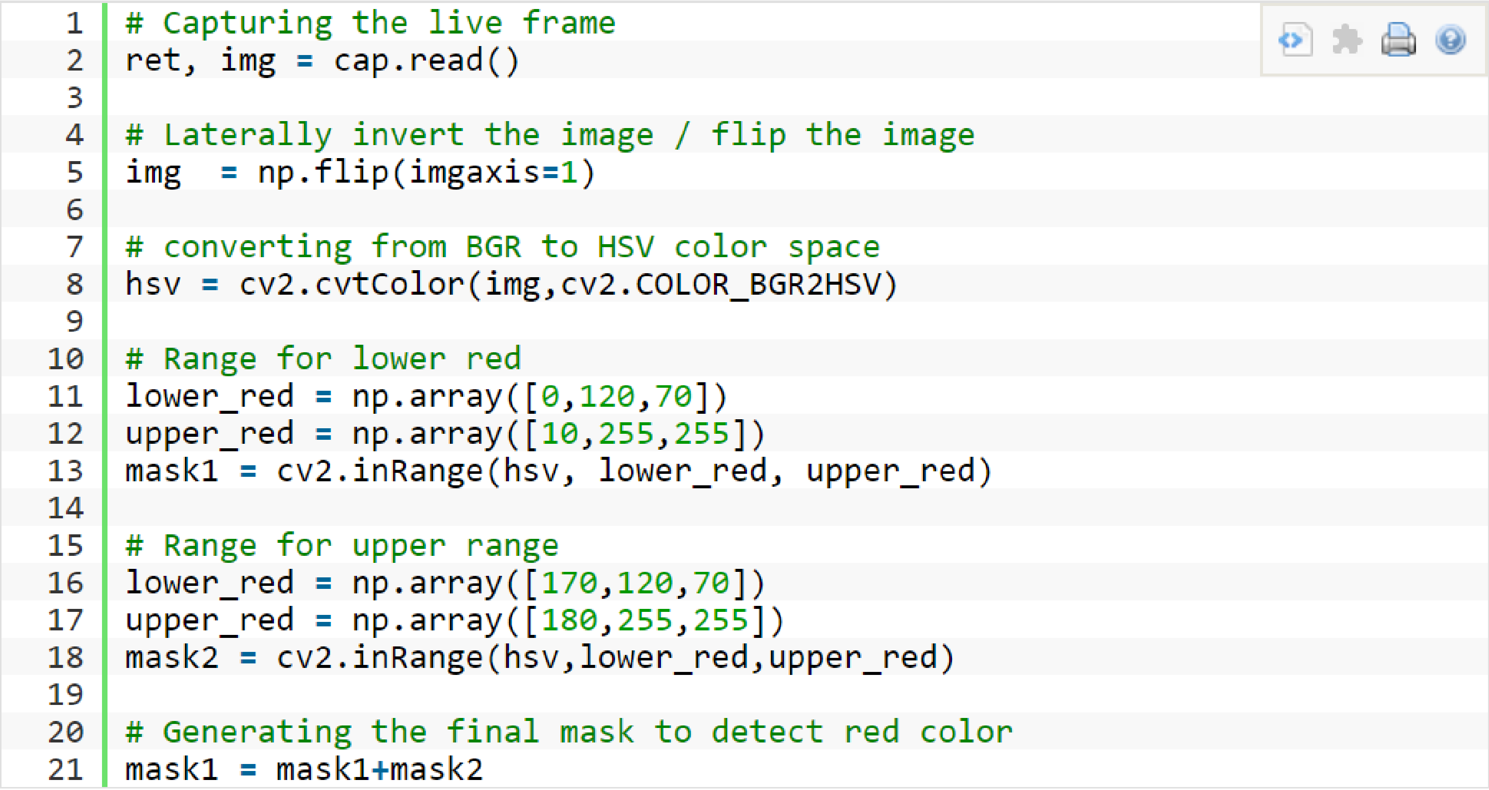


Step 1: Capture and store a background frame



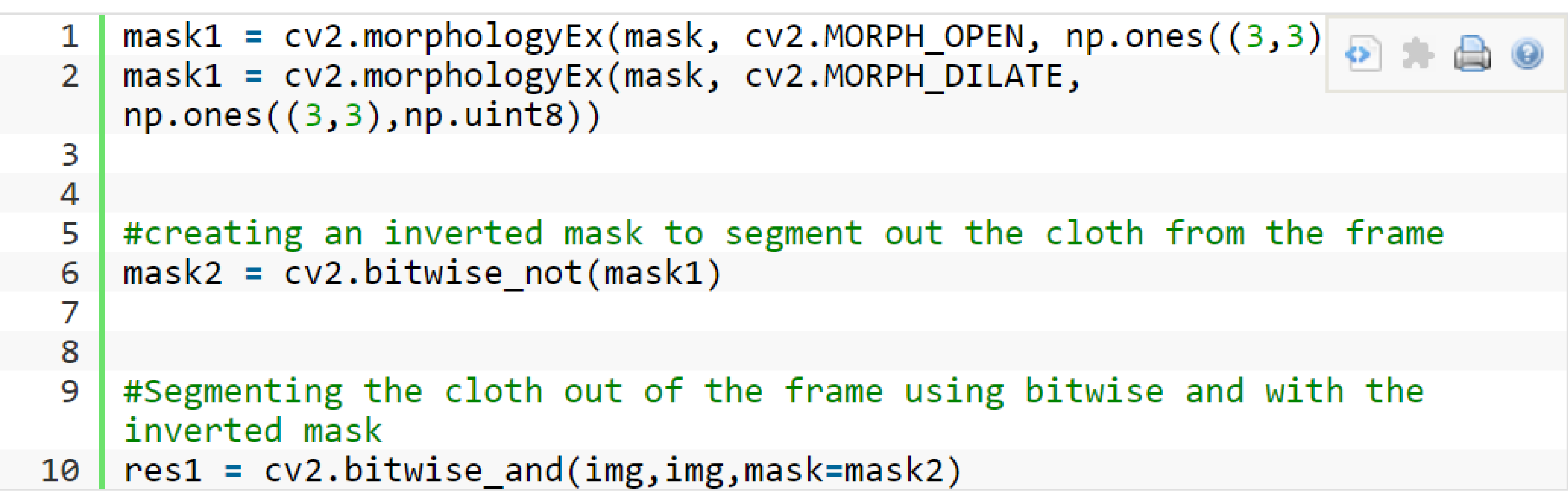
```
# Creating a VideoCapture object
# This will be used for image acquisition later in the code.
cap = cv2.VideoCapture("video.mp4")

# We give some time for the camera to warm-up!
time.sleep(3)

background=0

for i in range(30):
  ret,background = cap.read()

# Laterally invert the image / flip the image.
background = np.flip(background,axis=1)
```

Step 2: Color detection



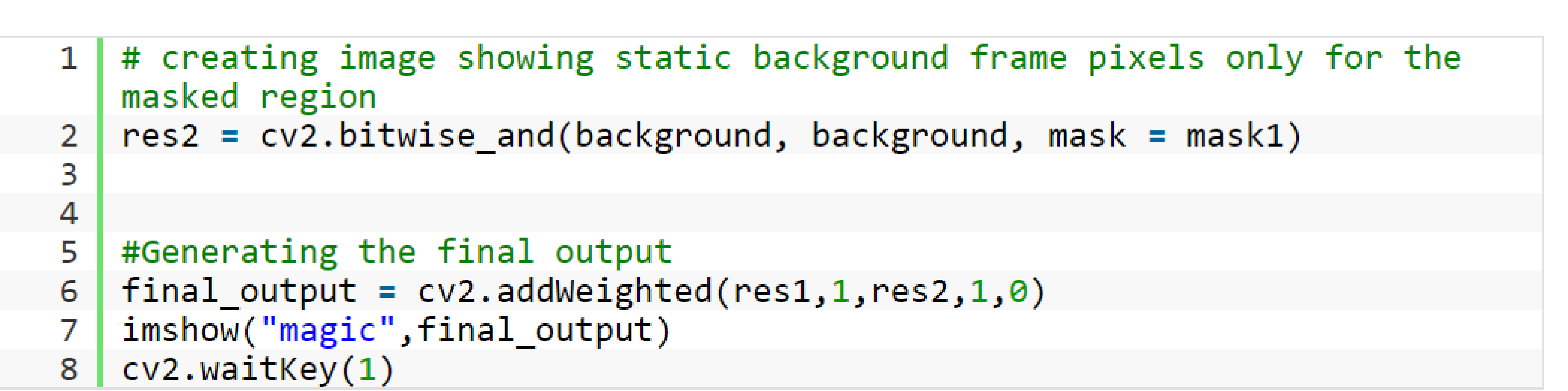
```python
# Capturing the live frame
ret, img = cap.read()

# Laterally invert the image / flip the image
img  = np.flip(imgaxis=1)

# converting from BGR to HSV color space
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

# Range for lower red
lower_red = np.array([0,120,70])
upper_red = np.array([10,255,255])
mask1 = cv2.inRange(hsv, lower_red, upper_red)

# Range for upper range
lower_red = np.array([170,120,70])
upper_red = np.array([180,255,255])
mask2 = cv2.inRange(hsv,lower_red,upper_red)

# Generating the final mask to detect red color
mask1 = mask1+mask2
```

Step 3: Segmenting out the detected colored cloth



```
mask1 = cv2.morphologyEx(mask, cv2.MORPH_OPEN, np.ones((3,3)
mask1 = cv2.morphologyEx(mask, cv2.MORPH_DILATE,
np.ones((3,3),np.uint8))


#creating an inverted mask to segment out the cloth from the frame
mask2 = cv2.bitwise_not(mask1)


#Segmenting the cloth out of the frame using bitwise and with the
inverted mask
res1 = cv2.bitwise_and(img,img,mask=mask2)
```

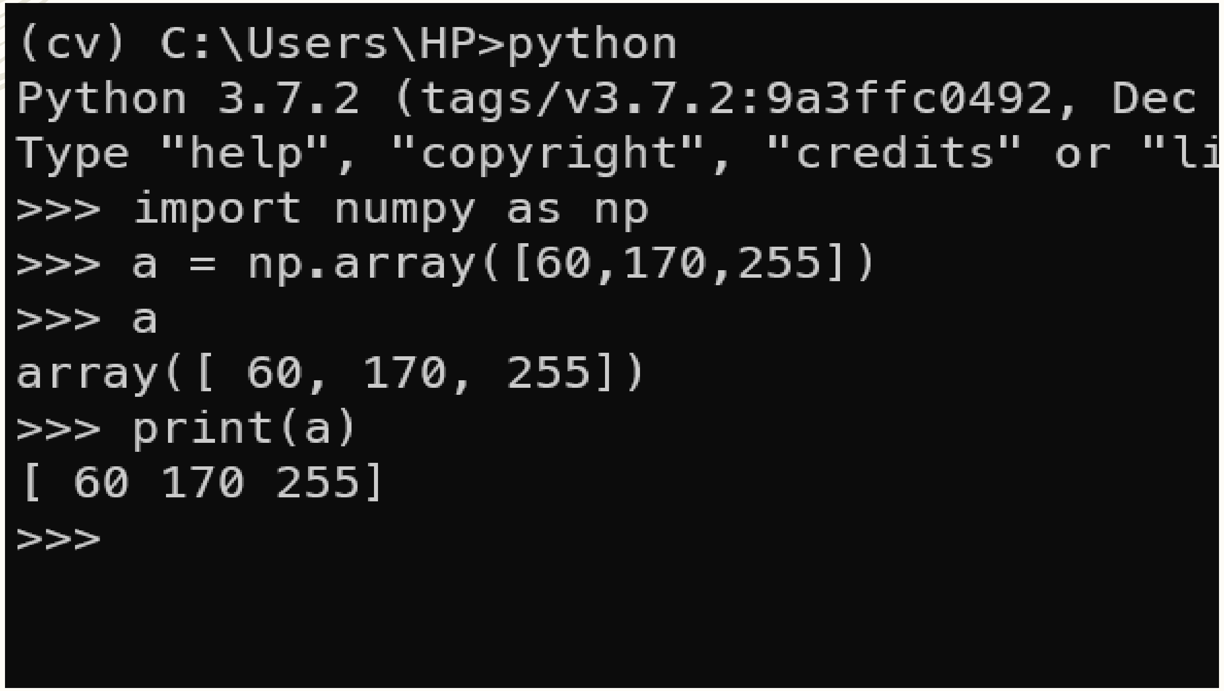
Step 4: Generating the final augmented output.



```
# creating image showing static background frame pixels only for the masked region
res2 = cv2.bitwise_and(background, background, mask = mask1)


#Generating the final output
final_output = cv2.addWeighted(res1,1,res2,1,0)
imshow("magic",final_output)
cv2.waitKey(1)
```

numpy.array([])

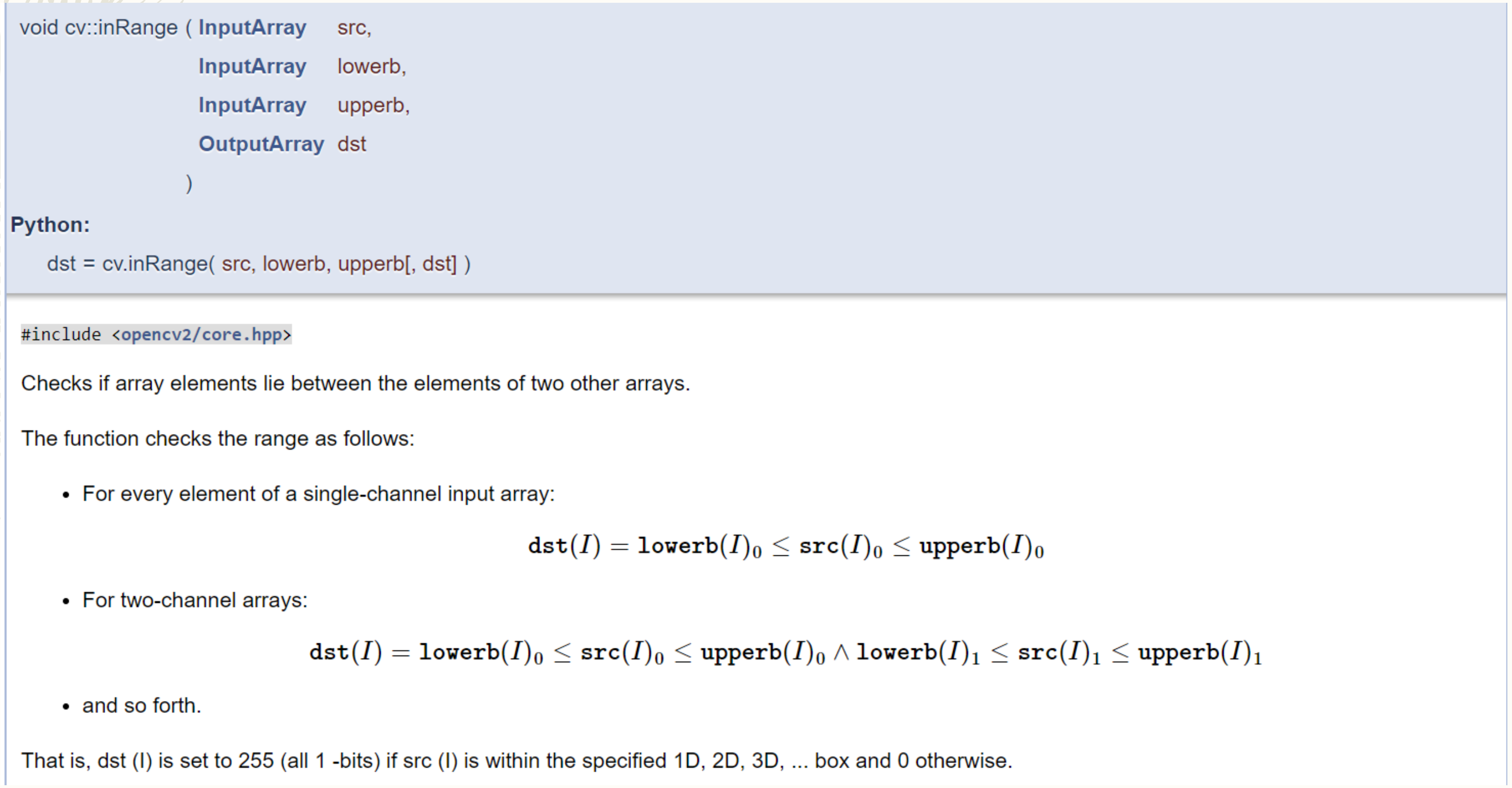
– This will create an array with elements listed as argument.



```
(cv) C:\Users\HP>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec
Type "help", "copyright", "credits" or "li
>>> import numpy as np
>>> a = np.array([60,170,255])
>>> a
array([ 60, 170, 255])
>>> print(a)
[ 60 170 255]
>>>
```

cv2.inRange(src,array1,array2)

– This checks the elements which are in the src array having values between array1 and array2 values.



```
void cv::inRange ( InputArray   src,
                   InputArray   lowerb,
                   InputArray   upperb,
                   OutputArray  dst
                 )
```

Python:

```
    dst = cv.inRange( src, lowerb, upperb[, dst] )
```

`#include <opencv2/core.hpp>`

Checks if array elements lie between the elements of two other arrays.

The function checks the range as follows:

- For every element of a single-channel input array:

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0$$

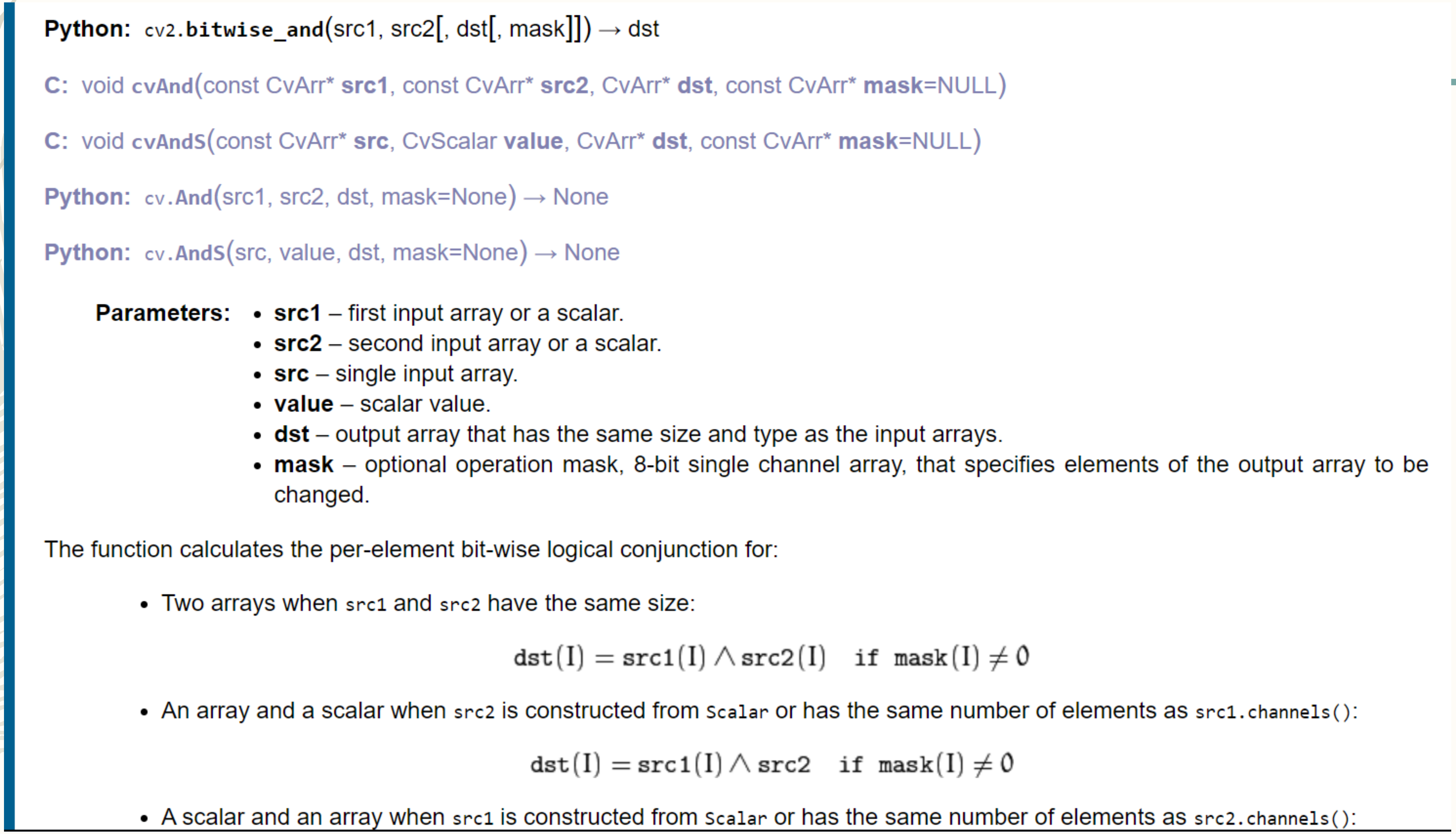
- For two-channel arrays:

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0 \land \texttt{lowerb}(I)_1 \leq \texttt{src}(I)_1 \leq \texttt{upperb}(I)_1$$

- and so forth.

That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

bitwiseAnd(src1,src2,mask)



Python: `cv2.bitwise_and(src1, src2[, dst[, mask]]) → dst`

C: void **cvAnd**(const CvArr* **src1**, const CvArr* **src2**, CvArr* **dst**, const CvArr* **mask**=NULL)

C: void **cvAndS**(const CvArr* **src**, CvScalar **value**, CvArr* **dst**, const CvArr* **mask**=NULL)

Python: `cv.And(src1, src2, dst, mask=None) → None`

Python: `cv.AndS(src, value, dst, mask=None) → None`

Parameters:

- **src1** – first input array or a scalar.
- **src2** – second input array or a scalar.
- **src** – single input array.
- **value** – scalar value.
- **dst** – output array that has the same size and type as the input arrays.
- **mask** – optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.

The function calculates the per-element bit-wise logical conjunction for:

- Two arrays when `src1` and `src2` have the same size:

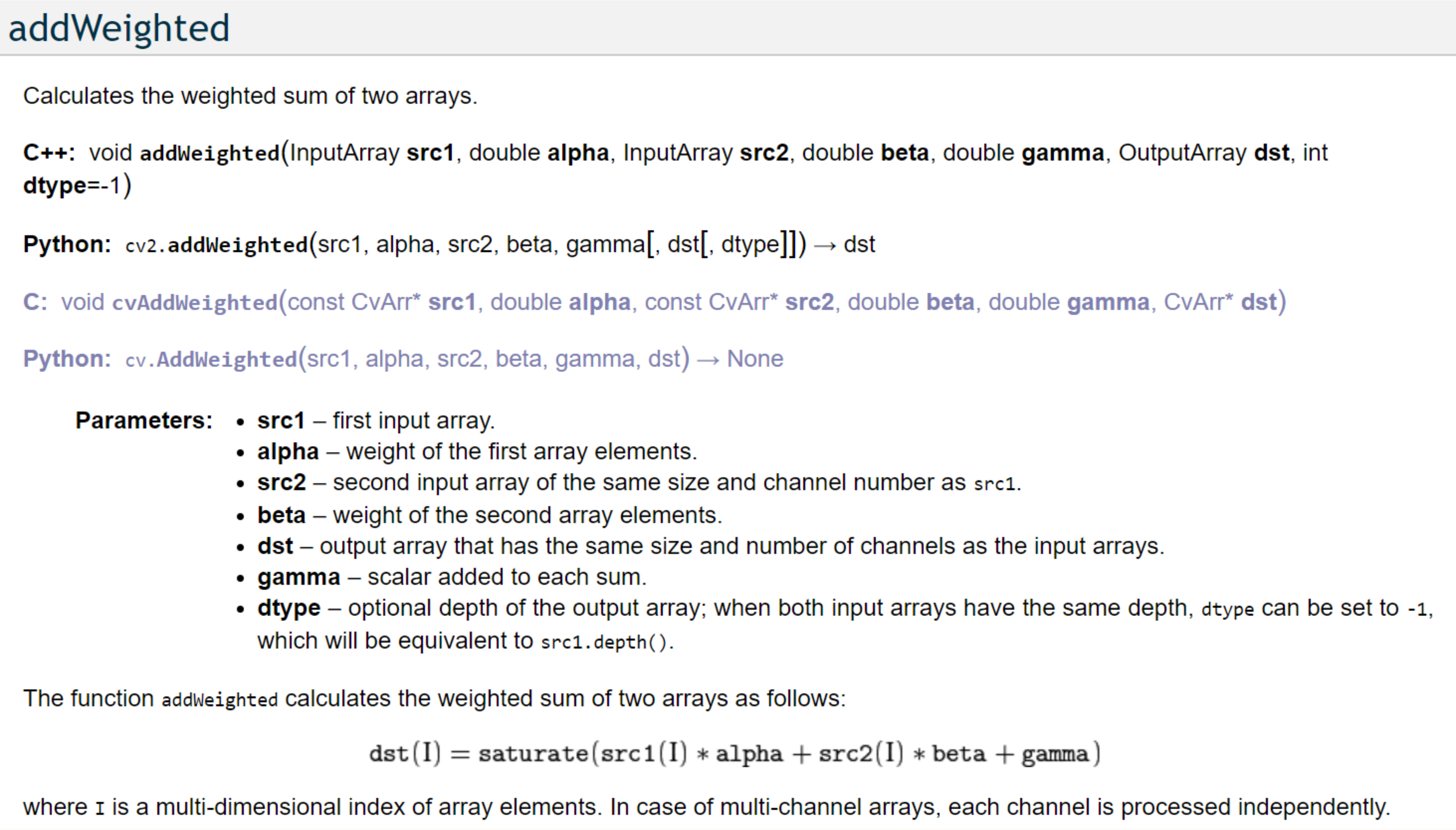
$$\texttt{dst}(I) = \texttt{src1}(I) \wedge \texttt{src2}(I) \quad \texttt{if} \;\; \texttt{mask}(I) \ne 0$$

- An array and a scalar when `src2` is constructed from `Scalar` or has the same number of elements as `src1.channels()`:

$$\texttt{dst}(I) = \texttt{src1}(I) \wedge \texttt{src2} \quad \texttt{if} \;\; \texttt{mask}(I) \ne 0$$

- A scalar and an array when `src1` is constructed from `Scalar` or has the same number of elements as `src2.channels()`:

addWeighted()



addWeighted

Calculates the weighted sum of two arrays.

C++: void `addWeighted`(InputArray **src1**, double **alpha**, InputArray **src2**, double **beta**, double **gamma**, OutputArray **dst**, int **dtype**=-1)

Python: `cv2.addWeighted`(src1, alpha, src2, beta, gamma[, dst[, dtype]]) → dst

C: void `cvAddWeighted`(const CvArr* **src1**, double **alpha**, const CvArr* **src2**, double **beta**, double **gamma**, CvArr* **dst**)

Python: `cv.AddWeighted`(src1, alpha, src2, beta, gamma, dst) → None

Parameters:

- **src1** – first input array.
- **alpha** – weight of the first array elements.
- **src2** – second input array of the same size and channel number as `src1`.
- **beta** – weight of the second array elements.
- **dst** – output array that has the same size and number of channels as the input arrays.
- **gamma** – scalar added to each sum.
- **dtype** – optional depth of the output array; when both input arrays have the same depth, `dtype` can be set to `-1`, which will be equivalent to `src1.depth()`.

The function `addWeighted` calculates the weighted sum of two arrays as follows:

$$\texttt{dst}(I) = \texttt{saturate}(\texttt{src1}(I) * \texttt{alpha} + \texttt{src2}(I) * \texttt{beta} + \texttt{gamma})$$

where `I` is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.

References

- https://www.learnopencv.com/invisibility-cloak-using-color-detection-and-segmentation-with-opencv/