2.2.2_basic_image_manipulation_open_CV

September 13, 2021

**

Manipulating Images

**

Estimated time needed: **30** minutes

Objectives

In this lab, you will learn how to manipulate images, OpenCV image Arrays. You will learn how to copy an image to avoid aliasing. We will cover flipping images and cropping images. You will also learn to change pixel images; this will allow you to draw shapes, write text and superimpose images over other images.

Manipulating Images

Copying Images

Fliping Images

Cropping an Image

Changing Specific Image Pixels

Download the images for the lab

[1]:
```
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-CV0101EN-SkillsNetwork/images%20/images_part_1/cat.
↪png -O cat.png
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-CV0101EN-SkillsNetwork/images%20/images_part_1/
↪lenna.png -O lenna.png
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-CV0101EN-SkillsNetwork/images%20/images_part_1/
↪baboon.png -O baboon.png
```

```
--2021-09-13 15:45:18--  https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-CV0101EN-
SkillsNetwork/images%20/images_part_1/cat.png
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)… 169.63.118.104
```

```
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 663451 (648K) [image/png]
Saving to: 'cat.png'

cat.png             100%[===================>] 647.90K  --.-KB/s    in 0.005s

2021-09-13 15:45:18 (137 MB/s) - 'cat.png' saved [663451/663451]

--2021-09-13 15:45:19--  https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-CV0101EN-
SkillsNetwork/images%20/images_part_1/lenna.png
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)… 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 473831 (463K) [image/png]
Saving to: 'lenna.png'

lenna.png           100%[===================>] 462.73K  --.-KB/s    in 0.004s

2021-09-13 15:45:20 (109 MB/s) - 'lenna.png' saved [473831/473831]

--2021-09-13 15:45:20--  https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-CV0101EN-
SkillsNetwork/images%20/images_part_1/baboon.png
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)… 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 637192 (622K) [image/png]
Saving to: 'baboon.png'

baboon.png          100%[===================>] 622.26K  --.-KB/s    in 0.005s

2021-09-13 15:45:21 (125 MB/s) - 'baboon.png' saved [637192/637192]
```

We will be using these imported functions in the lab

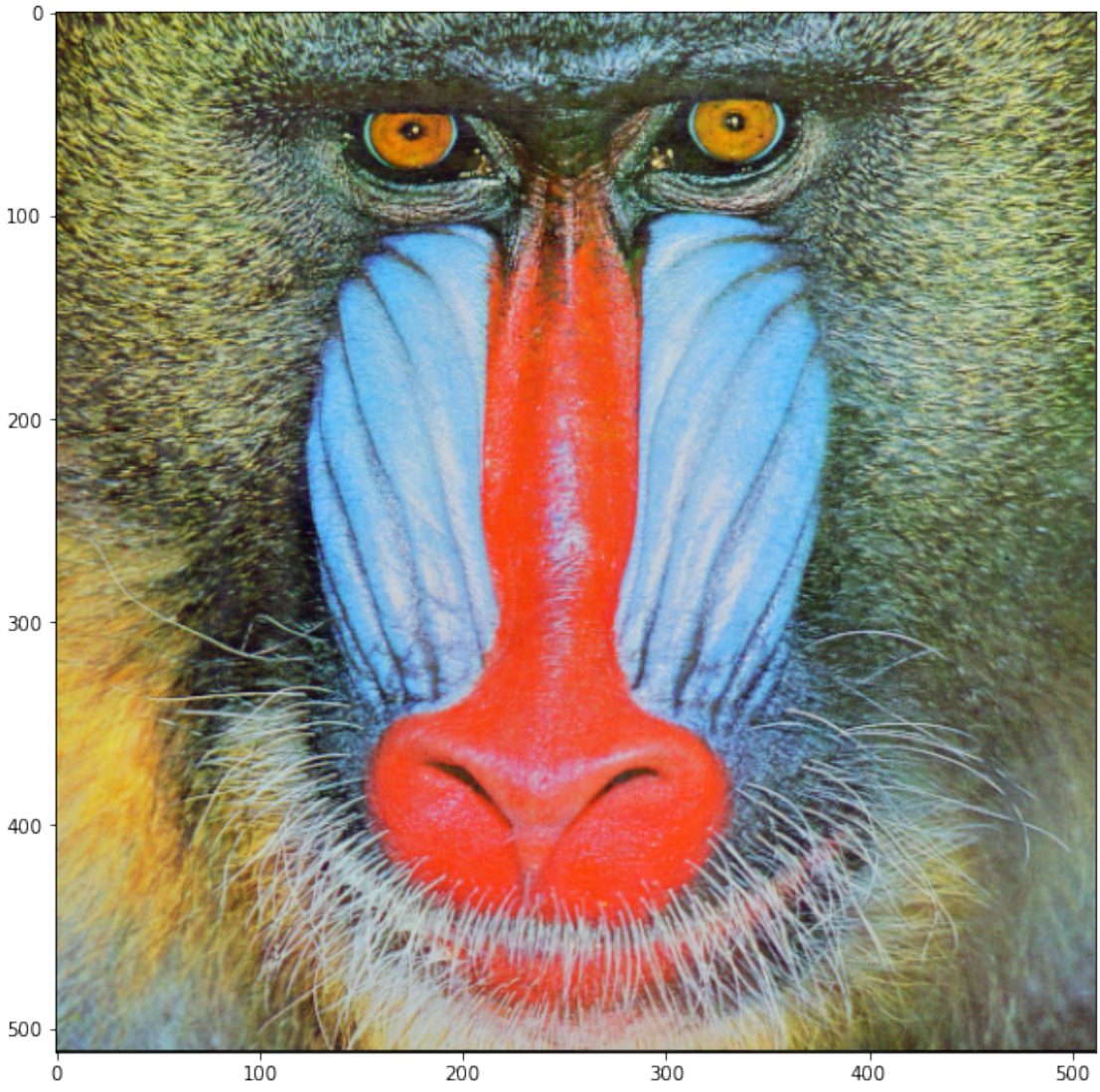
[2]:
```
import matplotlib.pyplot as plt
import cv2
```

```
import numpy as np
```

0.1 Copying Images

If you want to reassign an array to another variable, you should use the `copy` method. If we do not apply the method `copy()`, the variable will point to the same location in memory. Consider the following array:

[3]:
```
baboon = cv2.imread("baboon.png")
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB))
plt.show()
```



If we do not apply the method `copy()`, the new variable will point to the same location in memory:

[4]:
```
A = baboon
```

we use the `id` function to find the object's memory address; we see it is the same as the original array.

[5]:
```
id(A)==id(baboon)
id(A)
```

[5]:
```
139877469432848
```

If we apply the method 'copy(), the memory address is different

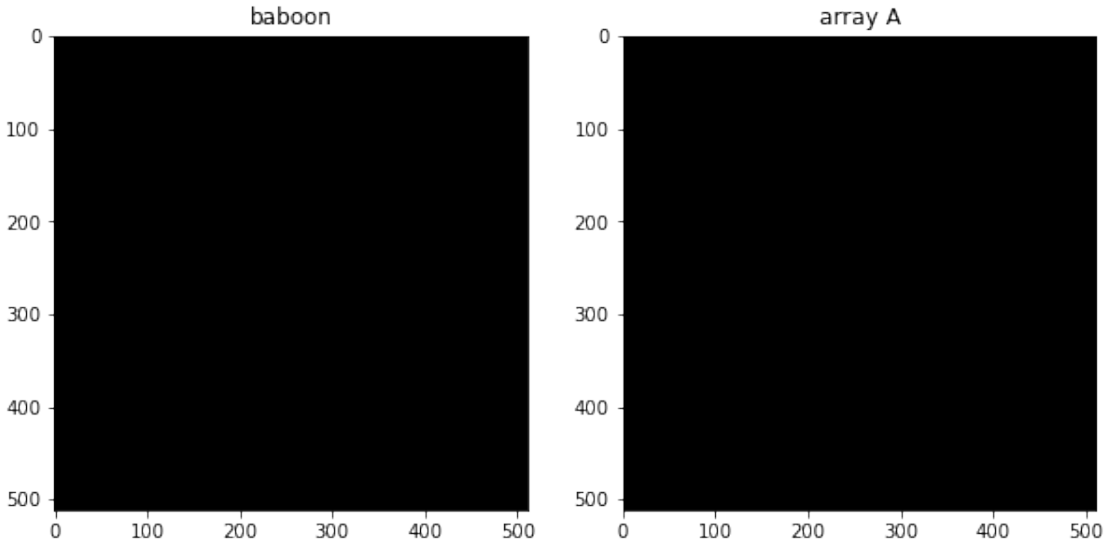
[6]:
```
B = baboon.copy()
id(B)==id(baboon)
```

[6]:
```
False
```

When we do not apply the method copy(), the variable will point to the same location in memory. Consider the array baboon, if we set all its values to zero, then all the values in A will be zero. This is because baboon and A point to the same place in memory, but B will not be affected.

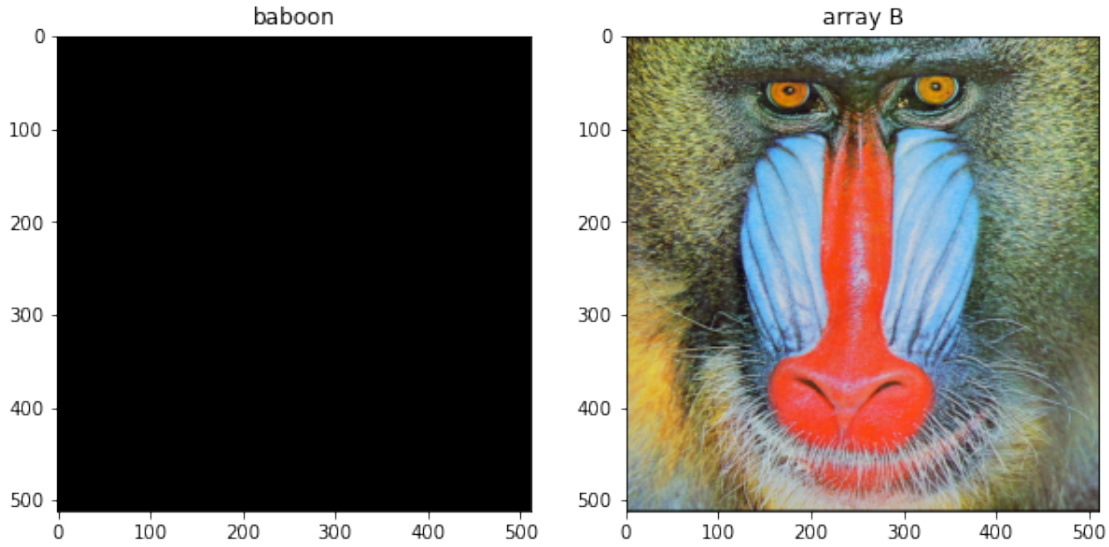
[7]:
```
baboon[:,:,] = 0
```

[8]:
```
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB))
plt.title("baboon")
plt.subplot(122)
plt.imshow(cv2.cvtColor(A, cv2.COLOR_BGR2RGB))
plt.title("array A")
plt.show()
```



We see they are the same, this is called aliasing. Aliasing happens whenever one variable's value is assigned to another variable because variables are just names that store references to values. We can also compare baboon and array B:

[9]:
```
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB))
plt.title("baboon")
plt.subplot(122)
plt.imshow(cv2.cvtColor(B, cv2.COLOR_BGR2RGB))
plt.title("array B")
plt.show()
```

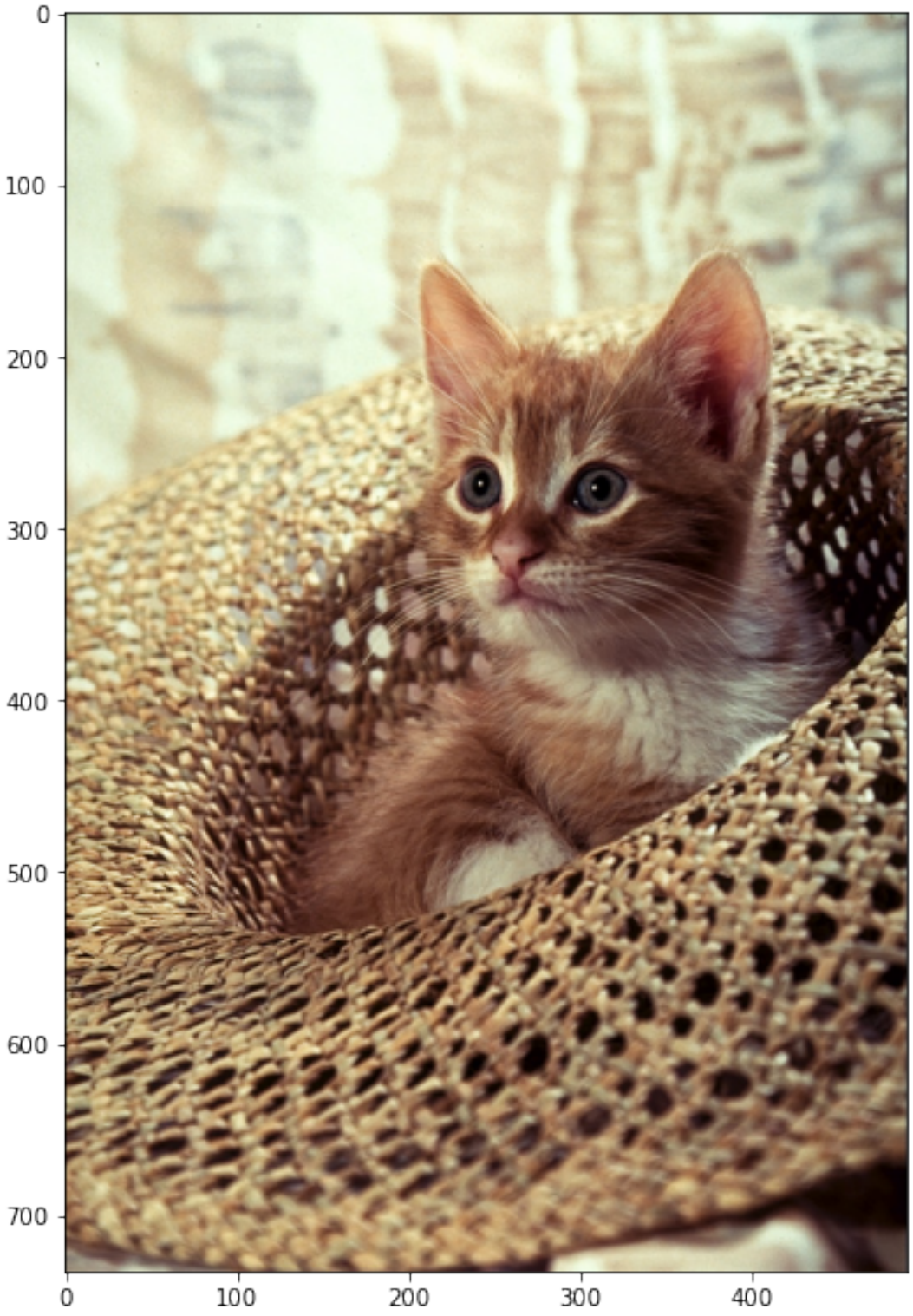


They are different because they used the method copy.

0.2 Fliping Images

Flipping images involves reordering the index of the pixels such that it changes the orientation of the image. Consider the following image:

[10]:
```
image = cv2.imread("cat.png")
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```



We can cast it to an array and find the shape:

[11]:
```
width, height,C=image.shape
print('width, height,C',width, height,C)
```

```
width, height,C 733 490 3
```

Let's Flip i.e rotate it vertically. First, we create an array of equal size of type np.uint8 bit image.

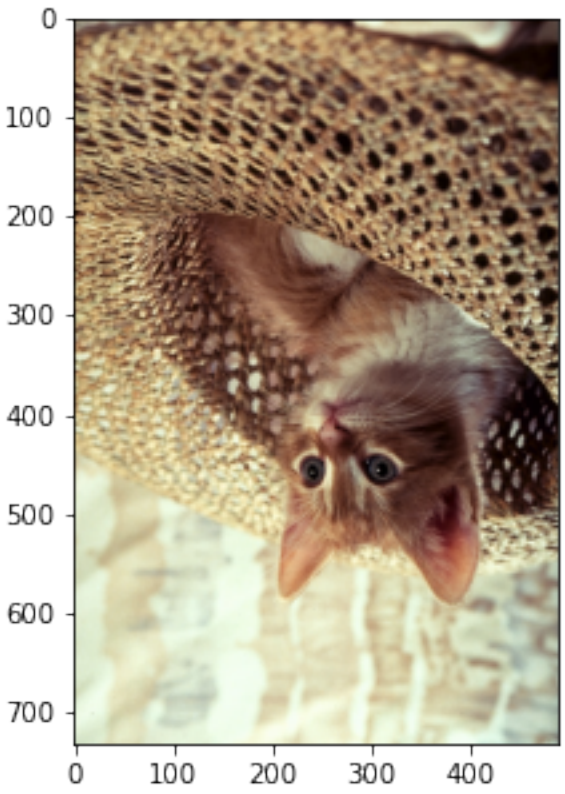
[12]:
```
array_flip = np.zeros((width, height,C),dtype=np.uint8)
```

We assign the first row of pixels of the original array to the new array's last row. We repeat the process for every row, incrementing the row number for the original array and decreasing the new array's row index assigning the pixels accordingly.

[13]:
```
for i,row in enumerate(image):
        array_flip[width-1-i,:,:]=row
```

We plot the results

[14]:
```
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(array_flip, cv2.COLOR_BGR2RGB))
plt.show()
```



OpenCVhas several ways to flip an image, we can use the flip() function; we have the input image

array. The parameter is the flipCode

is the value indicating what kind of flip we would like to perform;

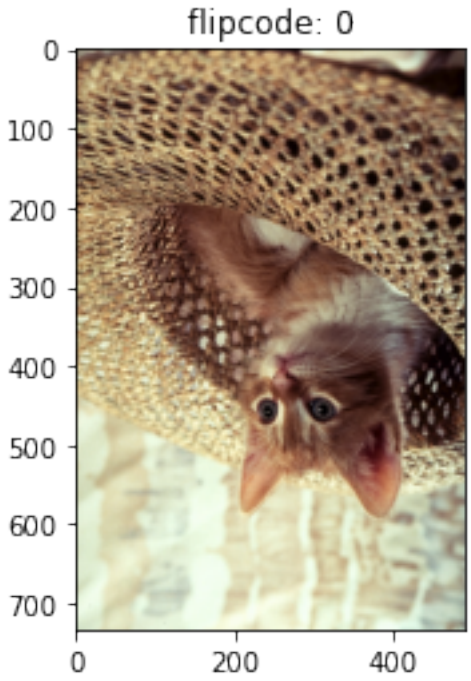
flipcode = 0: flip vertically around the x-axis

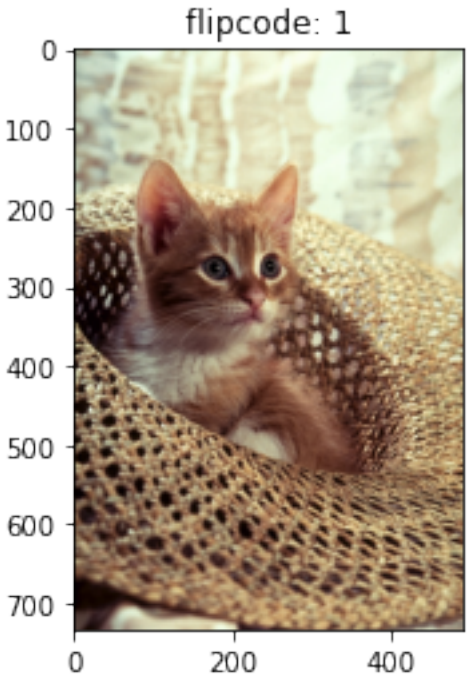
flipcode > 0: flip horizontally around y-axis positive value

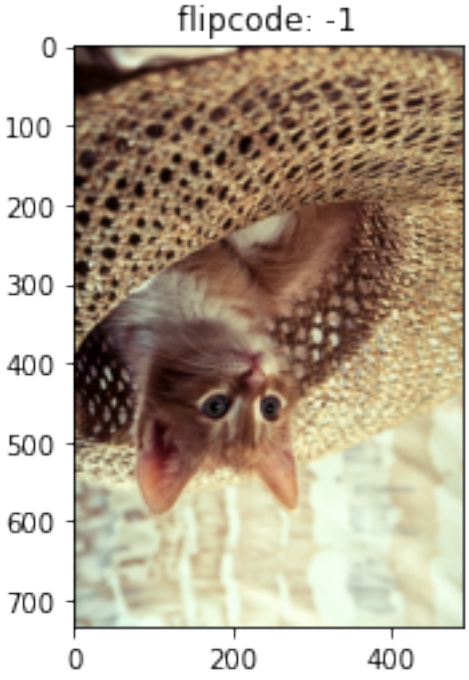
flipcode&#60 0: flip vertically and horizontally, flipping around both axes negative value

Let apply different flipcode's in a loop:

[15]:
```
for flipcode in [0,1,-1]:
    im_flip =  cv2.flip(image,flipcode )
    plt.imshow(cv2.cvtColor(im_flip,cv2.COLOR_BGR2RGB))
    plt.title("flipcode: "+str(flipcode))
    plt.show()
```

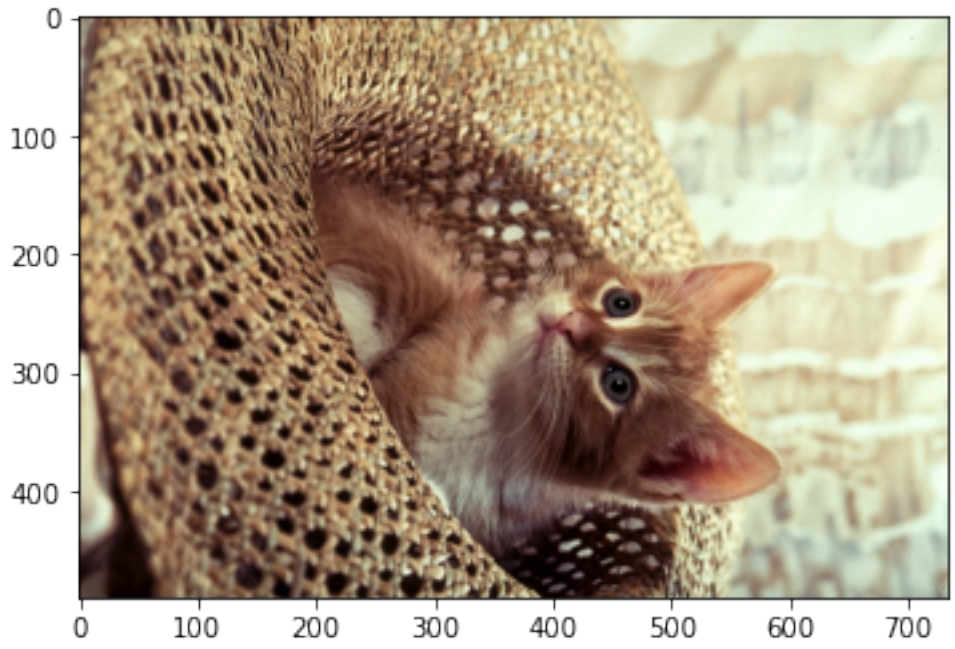






We can also use the rotate() function. The parameter is an integer indicating what kind of flip we would like to perform.

[16]:
```
im_flip = cv2.rotate(image,0)
plt.imshow(cv2.cvtColor(im_flip,cv2.COLOR_BGR2RGB))
plt.show()
```



OpenCV module has built-in attributes the describe the type of flip, the values are just integers. Several are shown in the following dict:

[17]:
```
flip = {"ROTATE_90_CLOCKWISE":cv2.
↪ROTATE_90_CLOCKWISE,"ROTATE_90_COUNTERCLOCKWISE":cv2.
↪ROTATE_90_COUNTERCLOCKWISE,"ROTATE_180":cv2.ROTATE_180}
```

We see the keys are just an integer

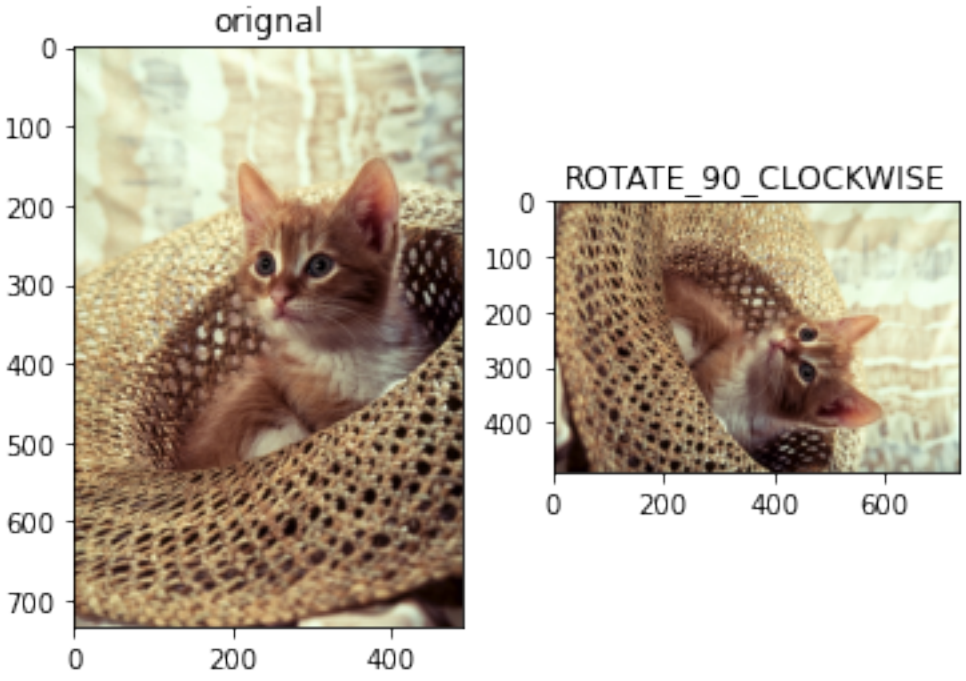
[18]:
```
flip["ROTATE_90_CLOCKWISE"]
```

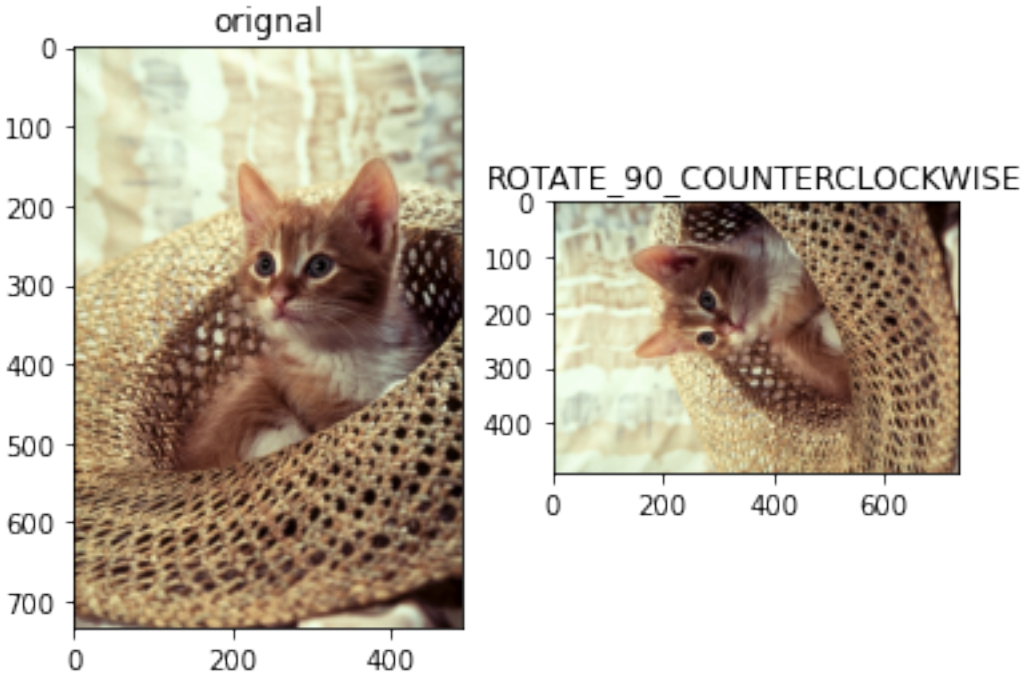
[18]:
```
0
```

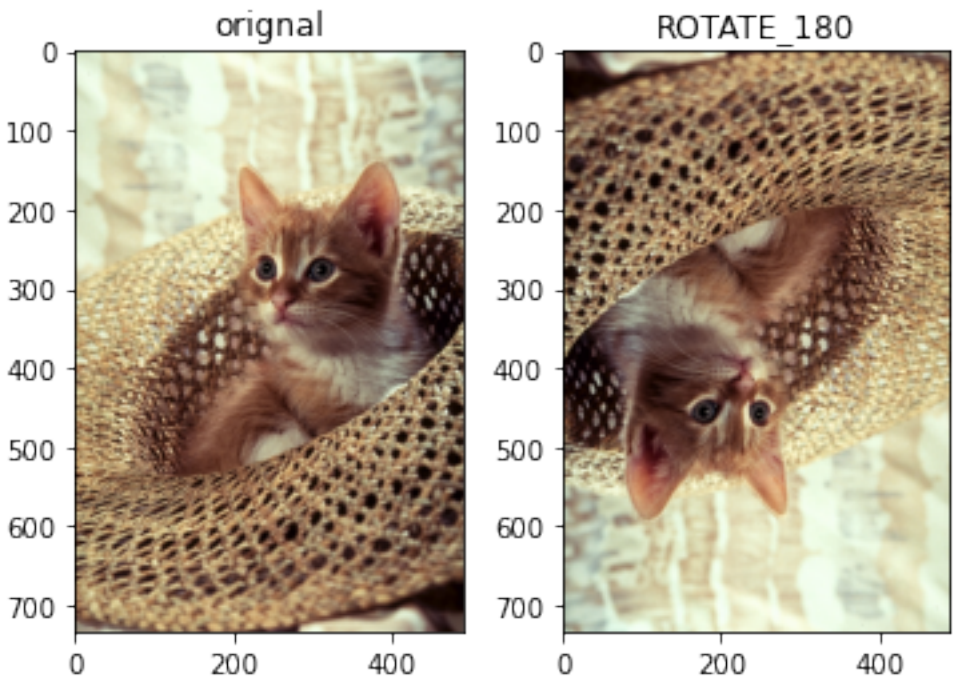
We can plot each of the outputs using the different parameter values

[19]:
```
for key, value in flip.items():
    plt.subplot(1,2,1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("orignal")
    plt.subplot(1,2,2)
    plt.imshow(cv2.cvtColor(cv2.rotate(image,value), cv2.COLOR_BGR2RGB))
    plt.title(key)
```

```
plt.show()
```


orignal
ROTATE_90_CLOCKWISE

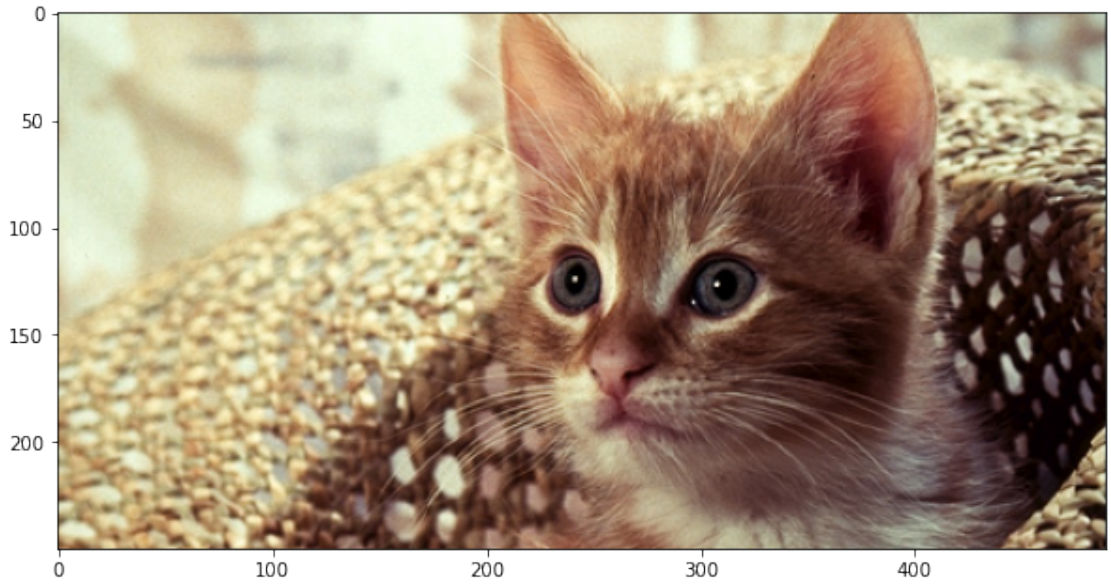

orignal
ROTATE_90_COUNTERCLOCKWISE



0.3 Cropping an Image

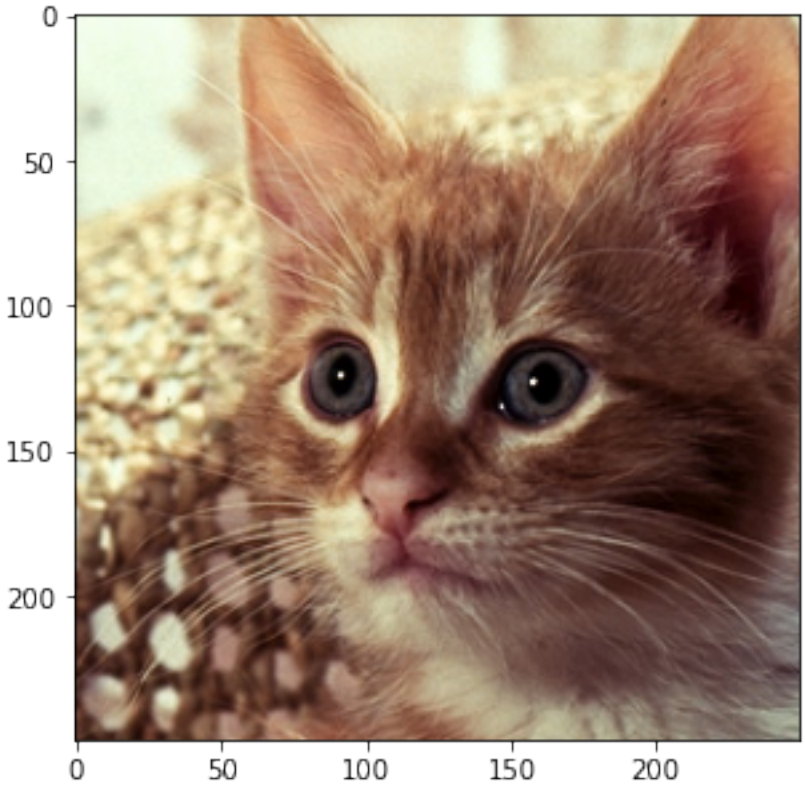
Cropping is "cutting out" the part of the image and throwing out the rest; we can crop using arrays. Let start with a vertical crop; the variable upper is the first row that we would like to include in the image, the variable lower is the last row we would like to include. We then use slicing to obtain the new image.

[20]:
```
upper = 150
lower = 400
crop_top = image[upper: lower,:,:]
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(crop_top, cv2.COLOR_BGR2RGB))
plt.show()
```



consider the array crop_top we can also crop horizontally the variable right is the first column that we would like to include in the image, the variable left is the last column we would like to include in the image.

[21]:
```
left = 150
right = 400
crop_horizontal = crop_top[: ,left:right,:]
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(crop_horizontal, cv2.COLOR_BGR2RGB))
plt.show()
```



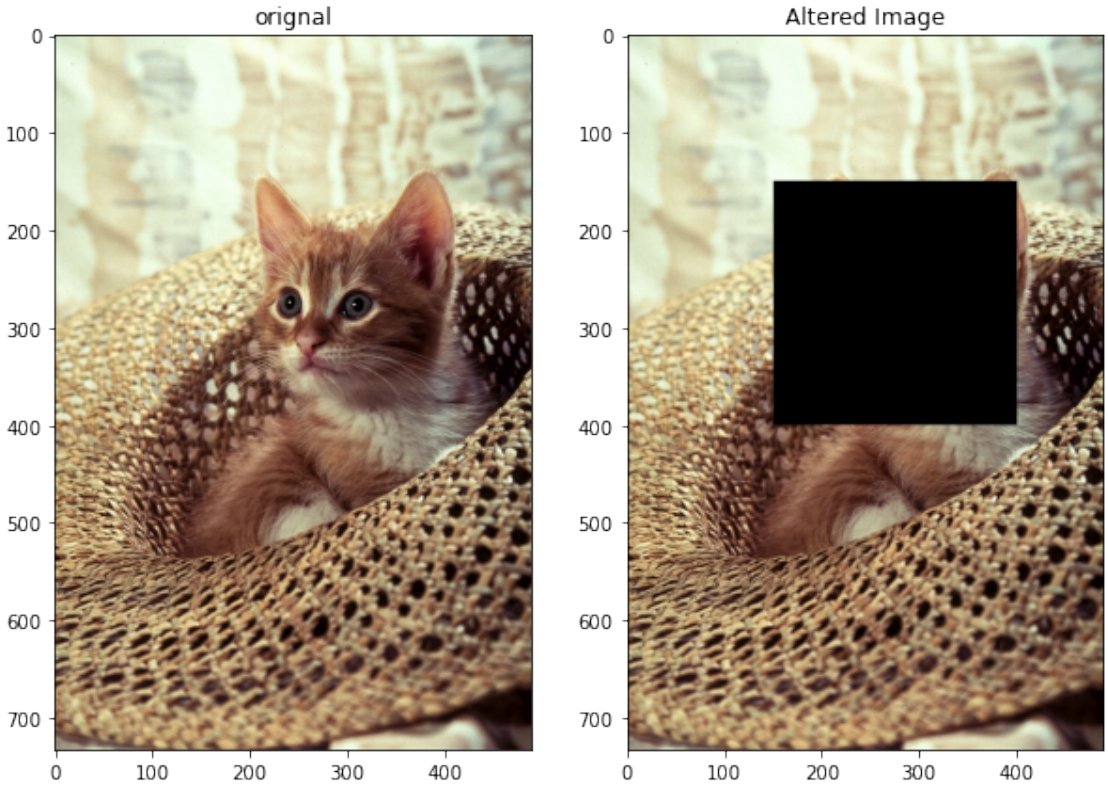
0.4 Changing Specific Image Pixels

We can change specific image pixels using array indexing; for example, we can set all the channels in the original image we cropped to zero :

[22]:
```
array_sq = np.copy(image)
array_sq[upper:lower,left:right,:] = 0
```

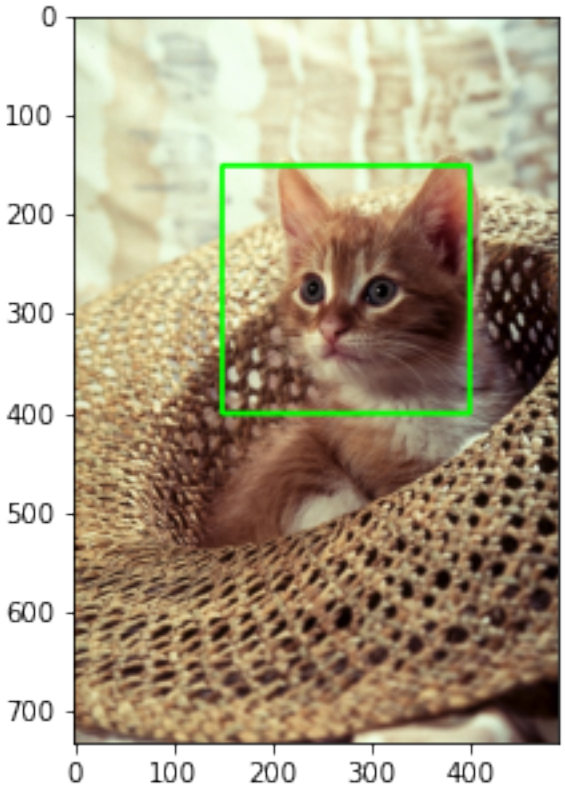
We can compare the results to the new image.

[23]:
```
plt.figure(figsize=(10,10))
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
plt.title("orignal")
plt.subplot(1,2,2)
plt.imshow(cv2.cvtColor(array_sq,cv2.COLOR_BGR2RGB))
plt.title("Altered Image")
plt.show()
```



We can also create shapes and OpenCV, we can use the method rectangle. The parameter pt1 is the top-left coordinate of the rectangle: (left,top) or (x_0, y_0), pt2 is the bottom right coordinate(right,lower) or (x_1, y_1). The parameter color is a tuple representing the intensity of each channel (blue, green, red). Finally, we have the line thickness.

[24]:
```
start_point, end_point = (left, upper),(right, lower)
image_draw = np.copy(image)
cv2.rectangle(image_draw, pt1=start_point, pt2=end_point, color=(0, 255, 0),␣
 ↪thickness=3)
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(image_draw, cv2.COLOR_BGR2RGB))
plt.show()
```



We can overlay text on an image using the function putText with the following parameter values:

img: Image array

text: Text string to be overlayed

org: Bottom-left corner of the text string in the image

fontFace: tye type of font

fontScale: Font scale

color: Text color

thickness: Thickness of the lines used to draw a text

lineType: Line type

[25]:
```
image_draw=cv2.
 ↪putText(img=image,text='Stuff',org=(10,500),color=(255,255,255),fontFace=4,fontScale=5,thic
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(image_draw,cv2.COLOR_BGR2RGB))
plt.show()
```


0
100
200
300
400
500
600
700
0
100
200
300
400
stuff

0.4.1 Question-4:

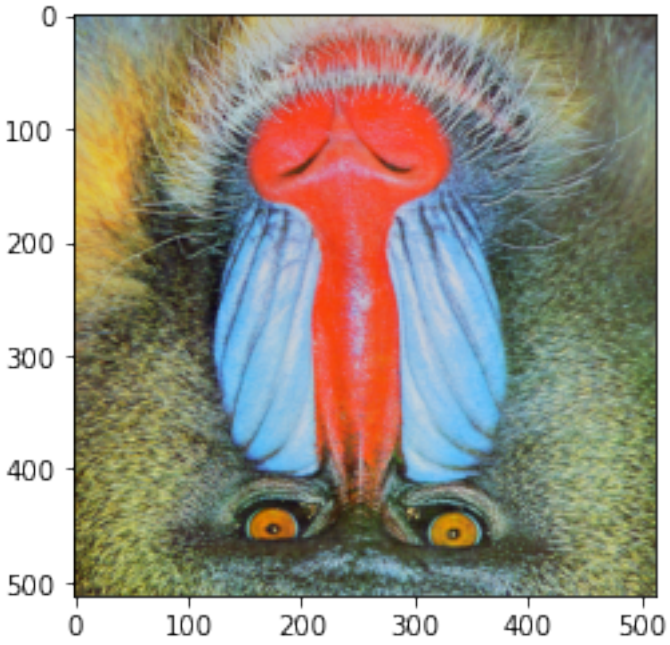
Use the image baboon.png from this lab or take any image you like.

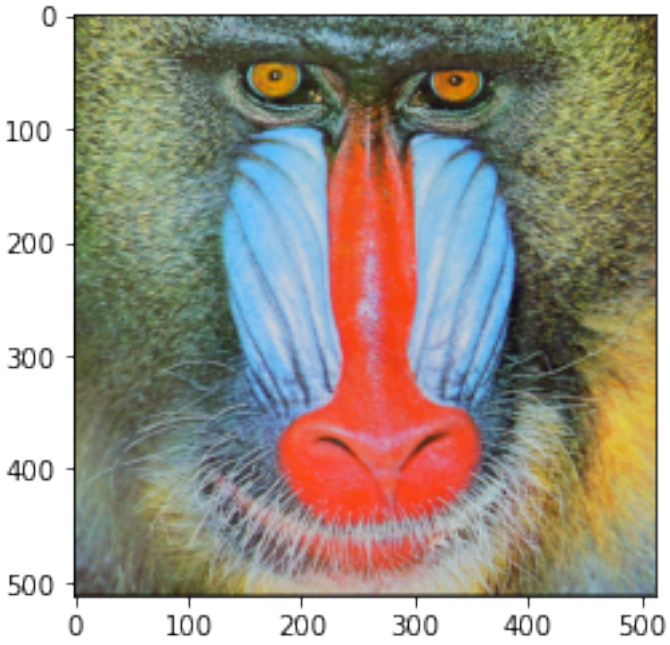
Open the image and create a OpenCV Image object called `im`, convert the image from BGR format to RGB format, flip `im` vertically around the x-axis and create an image called `im_flip`, mirror `im` by flipping it horizontally around the y-axis and create an image called `im_mirror`, finally plot both images

[26]:
```
# write your code here
im = cv2.imread("baboon.png")

im_flip =  cv2.flip(im, 0)
plt.imshow(cv2.cvtColor(im_flip, cv2.COLOR_BGR2RGB))
plt.show()

im_mirror =  cv2.flip(im, 1)
plt.imshow(cv2.cvtColor(im_mirror, cv2.COLOR_BGR2RGB))
plt.show()
```





Double-click **here** for a hint.

Double-click **here** for the solution.

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Nayef Abou Tayoun has a master of management in artificial intelligence degree, focusing on using machine learning and computer vision.

1 References

[1] Images were taken from: https://homepages.cae.wisc.edu/~ece533/images/

[2] Pillow Docs

[3] Open CV

[4] Gonzalez, Rafael C., and Richard E. Woods. "Digital image processing." (2017).

Change Log

Date (YYYY-MM-DD)

Version

Changed By

Change Description	
2020-07-20	
0.2	
Azim	
Modified Multiple Areas	
2020-07-17	
0.1	
Azim	
Created Lab Template	
2021-03-06	
0.3	
Nayef	
Modified some codes	