Manipulating Images

Objectives

Downloading the images for the lab

In [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
```

```
--2023-03-05 10:59:34--  https://cf-courses-data.s3.us.cloud-object-storag
e.appdomain.cloud/IBMDeveloperSkillsNetwork-CV0101EN-SkillsNetwork/images%2
0/images_part_1/cat.png
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-co
urses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (c
f-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 0.01
s

2023-03-05 10:59:34 (52.6 MB/s) - ‘cat.png’ saved [663451/663451]

--2023-03-05 10:59:36--  https://cf-courses-data.s3.us.cloud-object-storag
e.appdomain.cloud/IBMDeveloperSkillsNetwork-CV0101EN-SkillsNetwork/images%2
0/images_part_1/lenna.png
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-co
urses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (c
f-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 0.00
8s

2023-03-05 10:59:36 (57.0 MB/s) - ‘lenna.png’ saved [473831/473831]

--2023-03-05 10:59:37--  https://cf-courses-data.s3.us.cloud-object-storag
e.appdomain.cloud/IBMDeveloperSkillsNetwork-CV0101EN-SkillsNetwork/images%2
0/images_part_1/baboon.png
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-co
urses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (c
f-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 0.01
s

2023-03-05 10:59:38 (41.0 MB/s) - ‘baboon.png’ saved [637192/637192]
```

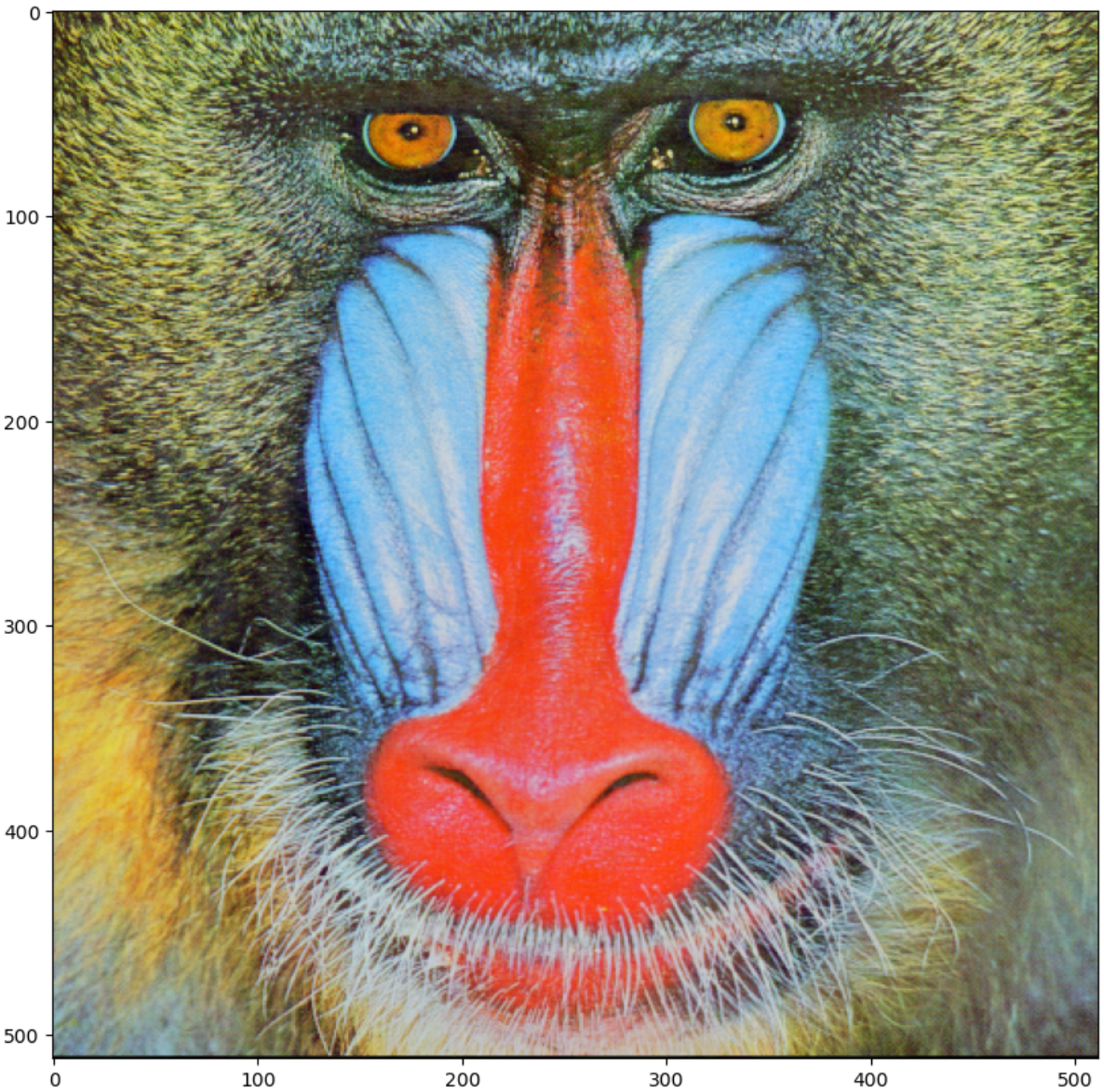
We will be using these imported functions in the lab

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

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:

In [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:

In [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.

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

Out[5]:
```
140207389452496
```

If we apply the method `copy()</coode>, the memory address is different

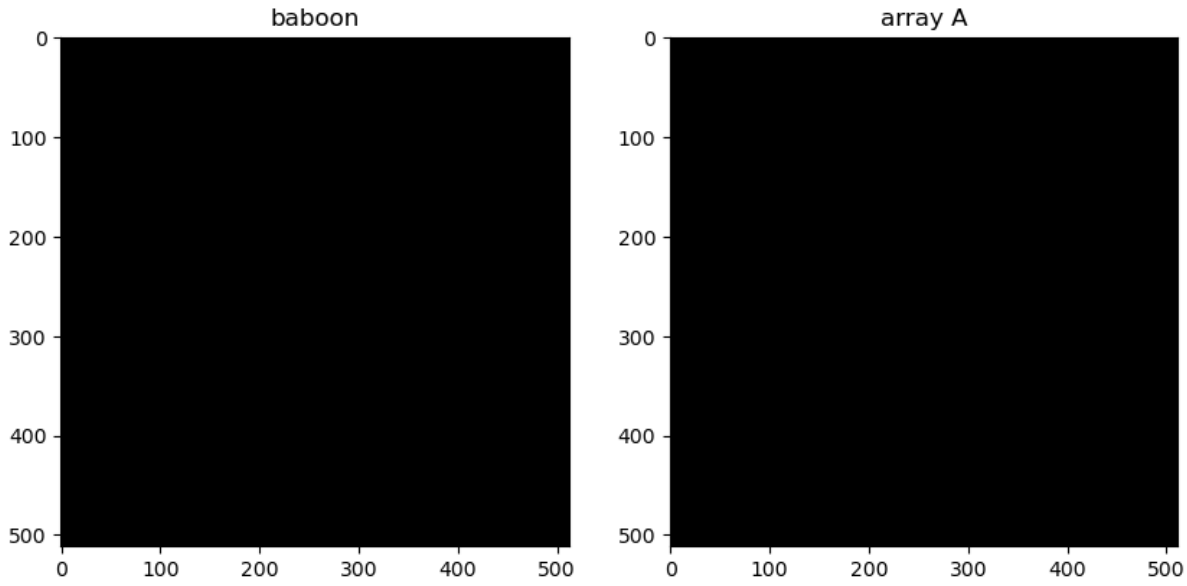
In [6]:
```
B = baboon.copy()
id(B)==id(baboon)
```

Out[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.

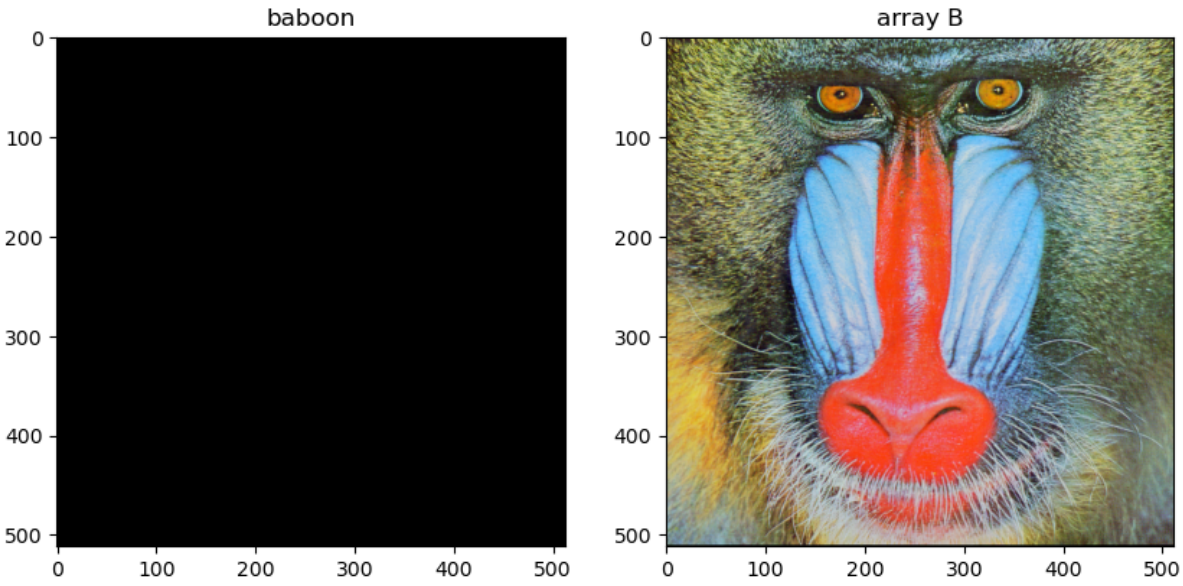
In [7]:
```
baboon[:,:,] = 0
```

In [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`:

In [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()
```

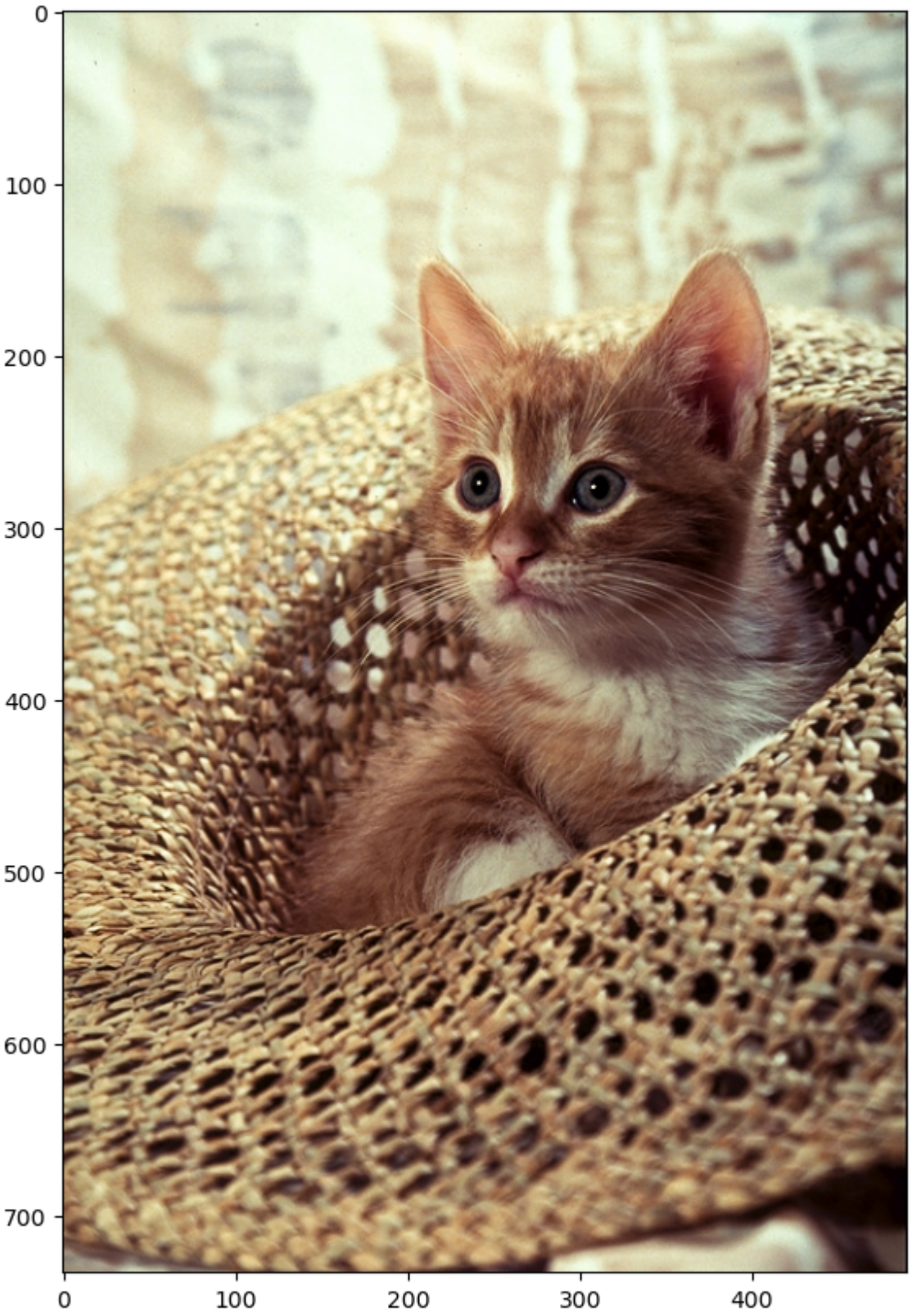


They are different because they used the method copy.

Fliping Images

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

In [11]:
```
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:

In [12]:
```
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.

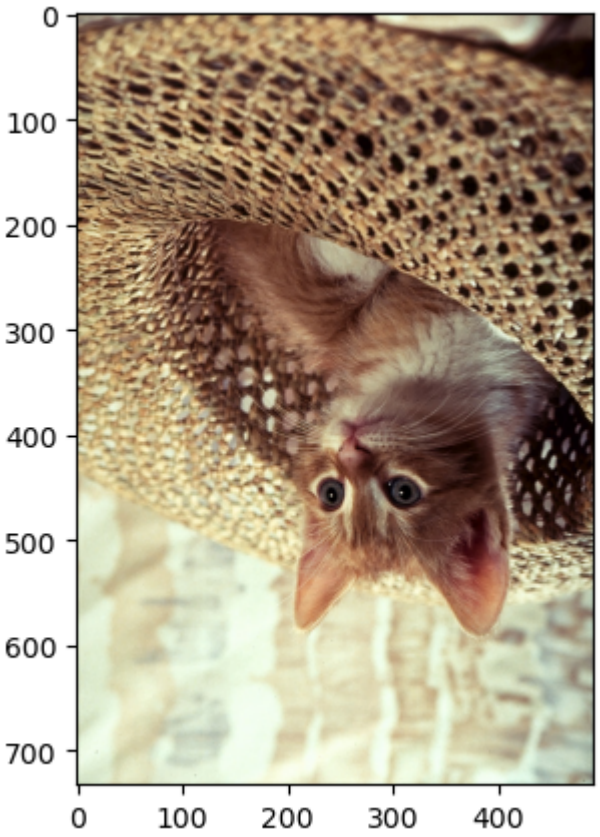
In [13]:
```
array_flip = np.zeros((width, height,C),dtype=np.uint8)
```

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.

In [14]:
```
for i,row in enumerate(image):
        array_flip[width-1-i,:,:]=row
```

We plot the results

In [15]:
```
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(array_flip, cv2.COLOR_BGR2RGB))
plt.show()
```



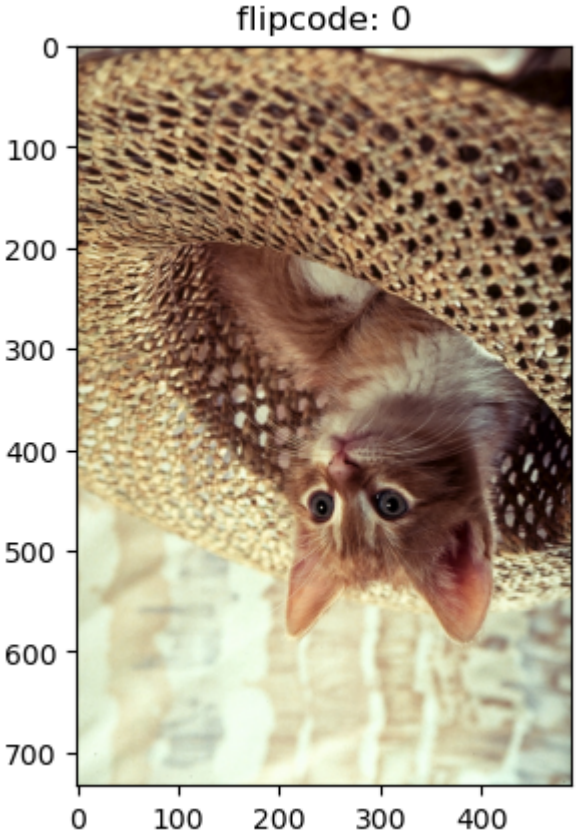
`OpenCV` has several ways to flip an image, we can use the `flip()` function; we have the input image array. The parameter is the `flipCode`

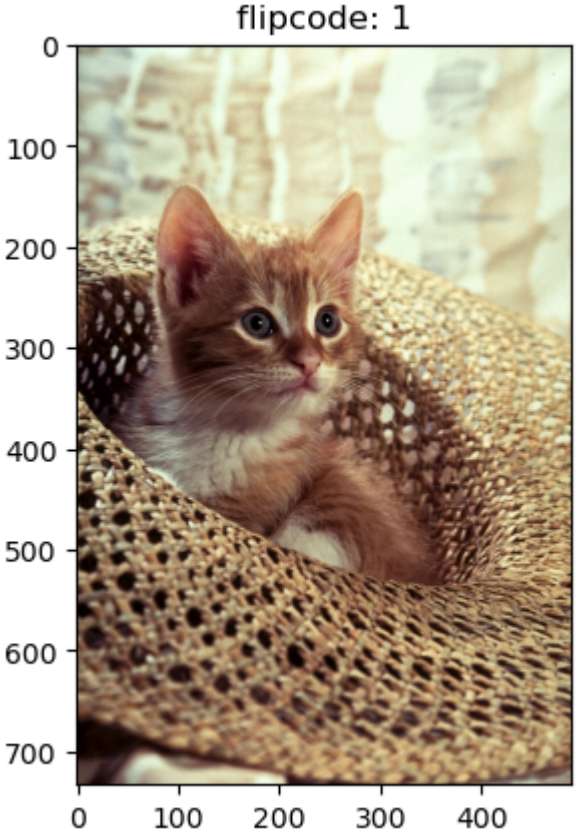
is the value indicating what kind of flip we would like to perform;

- `flipcode` = 0: flip vertically around the x-axis
- `flipcode` > 0: flip horizontally around y-axis positive value
- `flipcode` < 0: flip vertically and horizontally, flipping around both axes negative value

Let apply different `flipcode` 's in a loop:

In [16]:
```
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()
```


flipcode: 0
0
100
200
300
400
500
600
700
0
100
200
300
400


flipcode: 1
0
100
200
300
400
500
600
700
0
100
200
300
400

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

In [17]:
```
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` :

In [18]:
```
flip = {"ROTATE_90_CLOCKWISE":cv2.ROTATE_90_CLOCKWISE,"ROTATE_90_COUNTERCLO
CKWISE":cv2.ROTATE_90_COUNTERCLOCKWISE,"ROTATE_180":cv2.ROTATE_180}
```

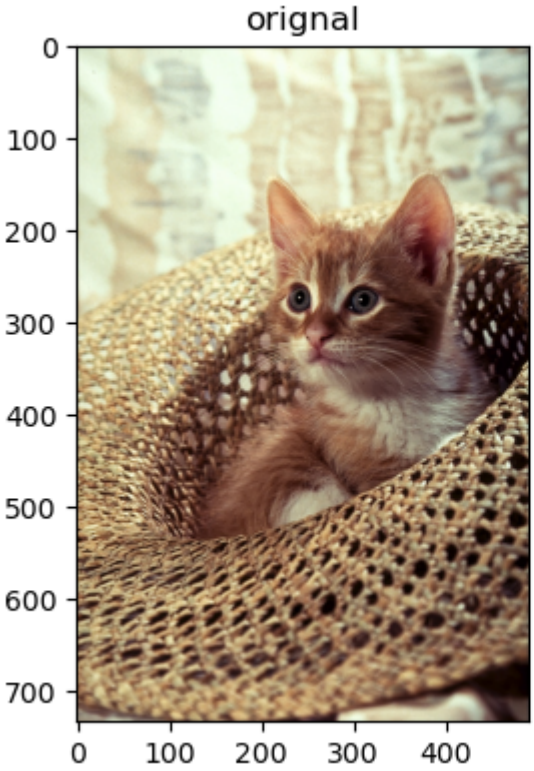
We see the keys are just an integer

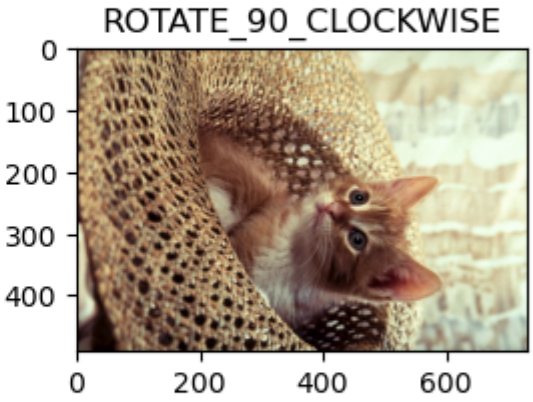
In [19]:
```
flip["ROTATE_90_CLOCKWISE"]
```

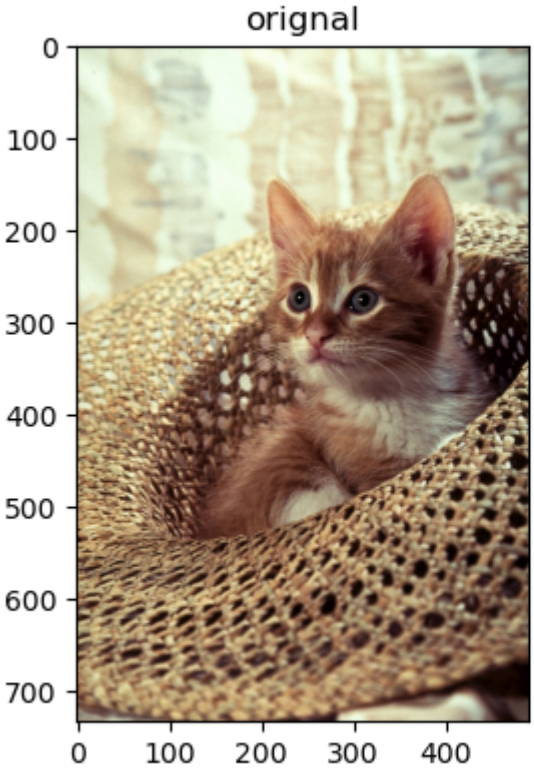
Out[19]: 0

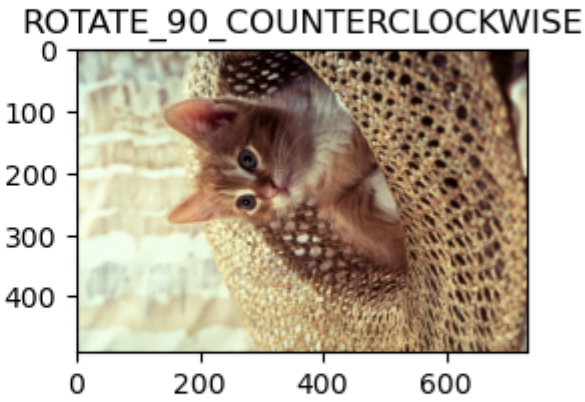
We can plot each of the outputs using the different parameter values

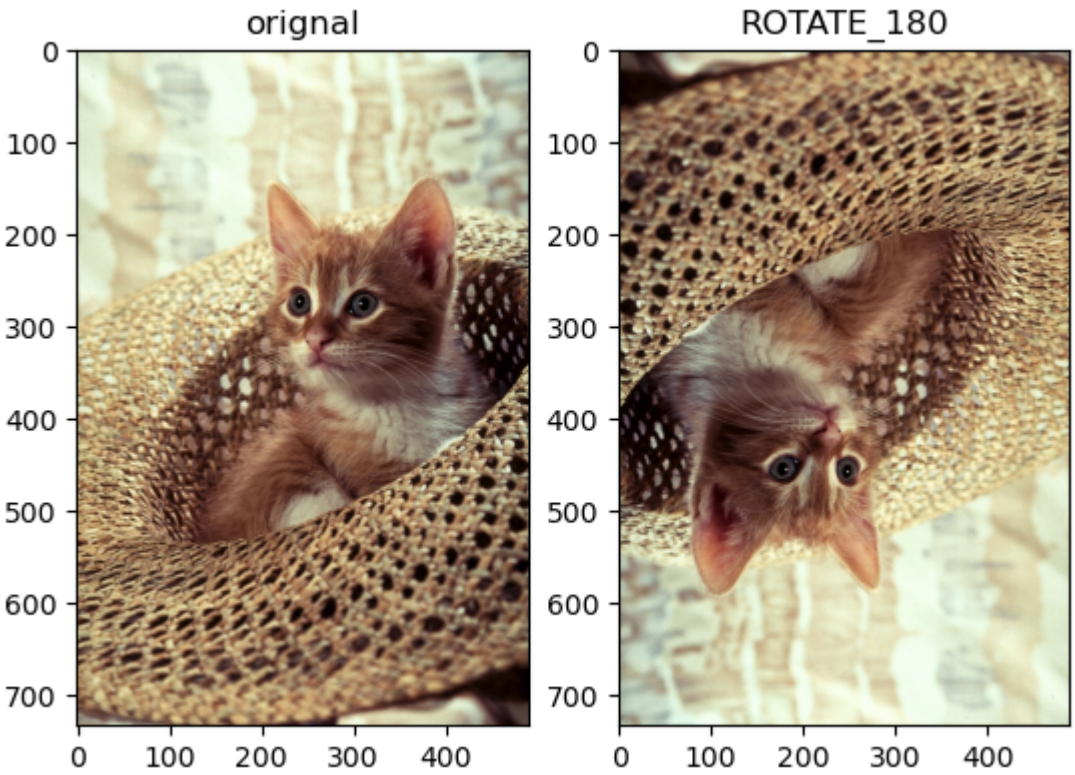
In [20]:
```
for key, value in flip.items():
    plt.subplot(1,2,1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("orignal")
    plt.subplot(1,2,2)
    plt.imshow(cv2.cvtColor(cv2.rotate(image,value), cv2.COLOR_BGR2RGB))
    plt.title(key)
    plt.show()
```


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


ROTATE_90_CLOCKWISE
0
100
200
300
400
0
200
400
600


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

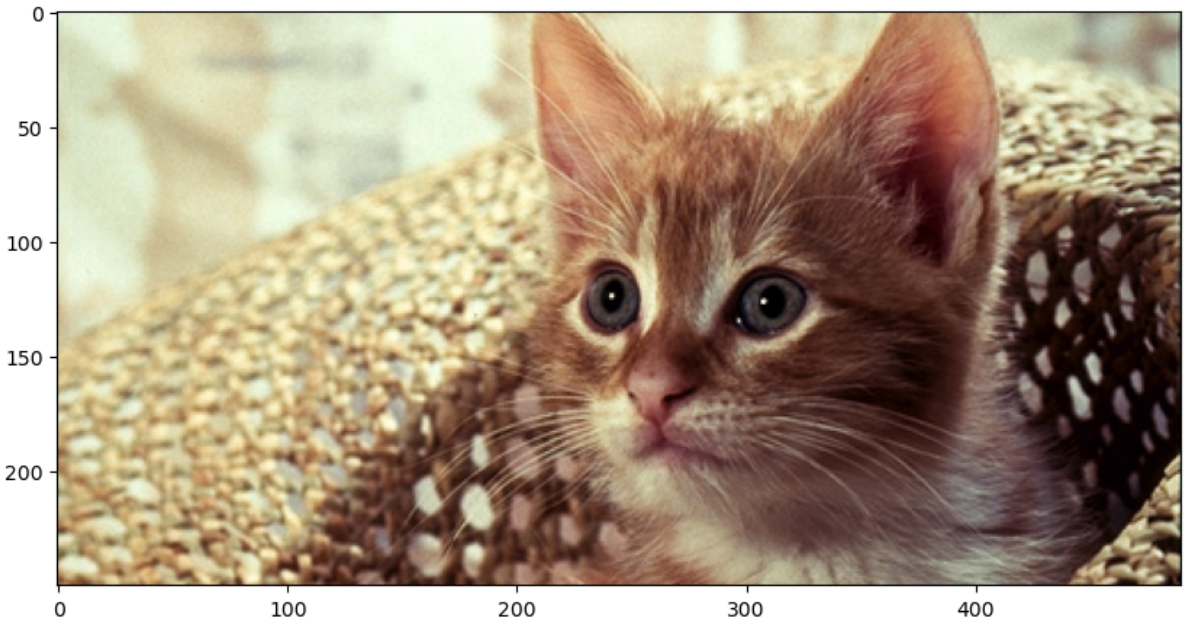

ROTATE_90_COUNTERCLOCKWISE
0
100
200
300
400
0
200
400
600



Cropping an Image

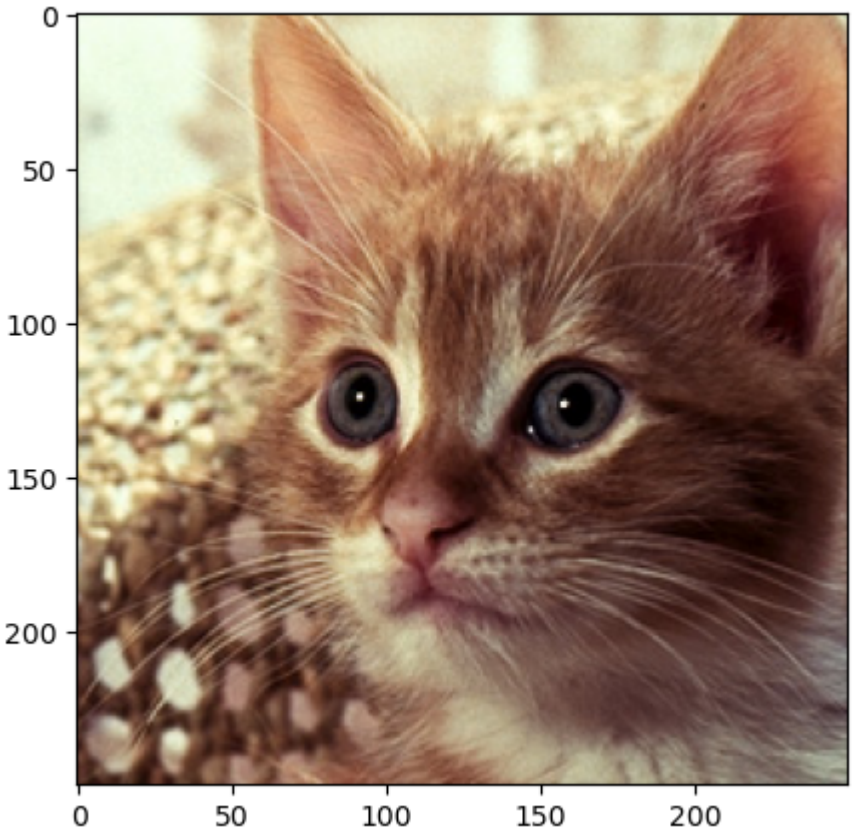
Cropping is "cutting out" the part of the image and throwing out the rest; we can crop using arrays. Let start with a vertical crop; the variable `upper` is the first row that we would like to include in the image, the variable `lower` is the last row we would like to include. We then use slicing to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

In [21]:
```
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.

In [22]:
```
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()
```



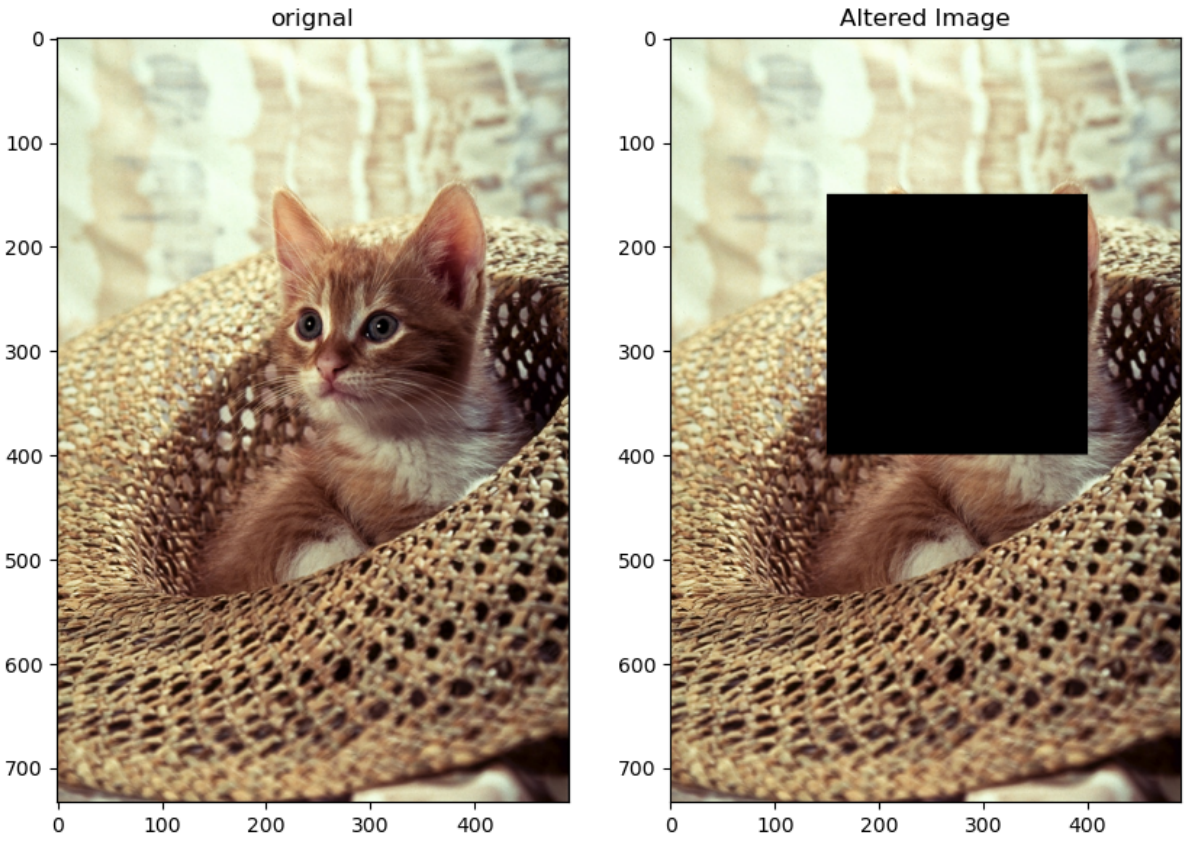
Changing Specific Image Pixels

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 :

In [23]:
```
array_sq = np.copy(image)
array_sq[upper:lower,left:right,:] = 0
```

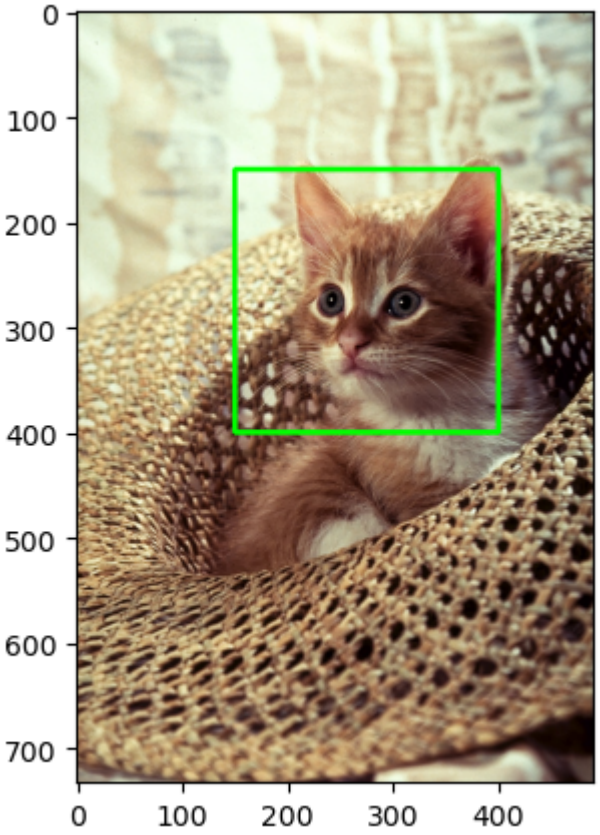
We can compare the results to the new image.

In [24]:
```
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.

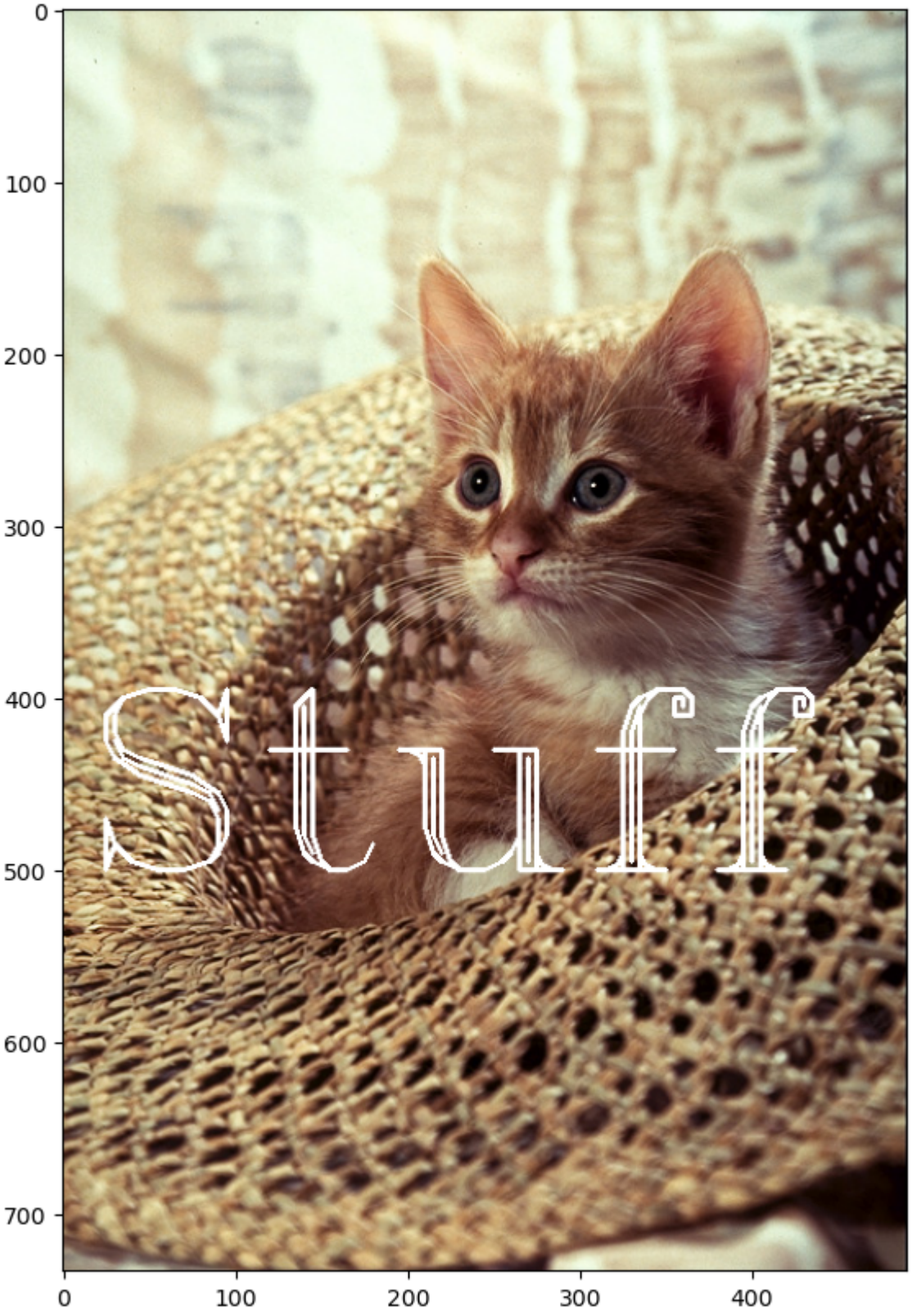
In [25]:
```
start_point, end_point = (left, upper),(right, lower)
image_draw = np.copy(image)
cv2.rectangle(image_draw, pt1=start_point, pt2=end_point, color=(0, 255,
0), thickness=3)
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(image_draw, cv2.COLOR_BGR2RGB))
plt.show()
```



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

In [26]:
```
image_draw=cv2.putText(img=image,text='Stuff',org=(10,500),color=(255,255,255),fontFace=4,fontScale=5,thickness=2)
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(image_draw,cv2.COLOR_BGR2RGB))
plt.show()
```

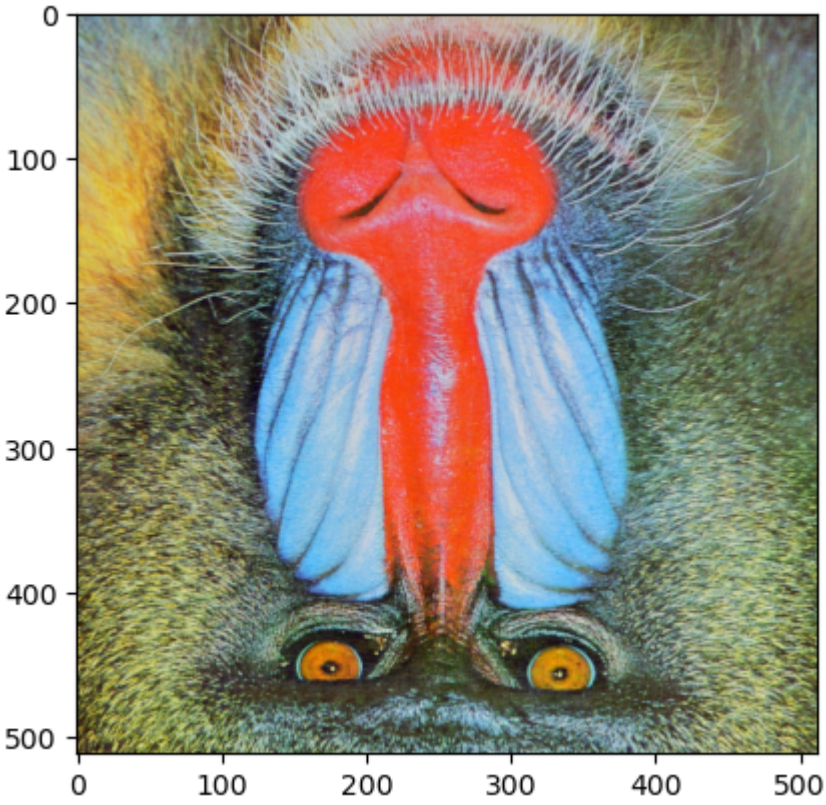


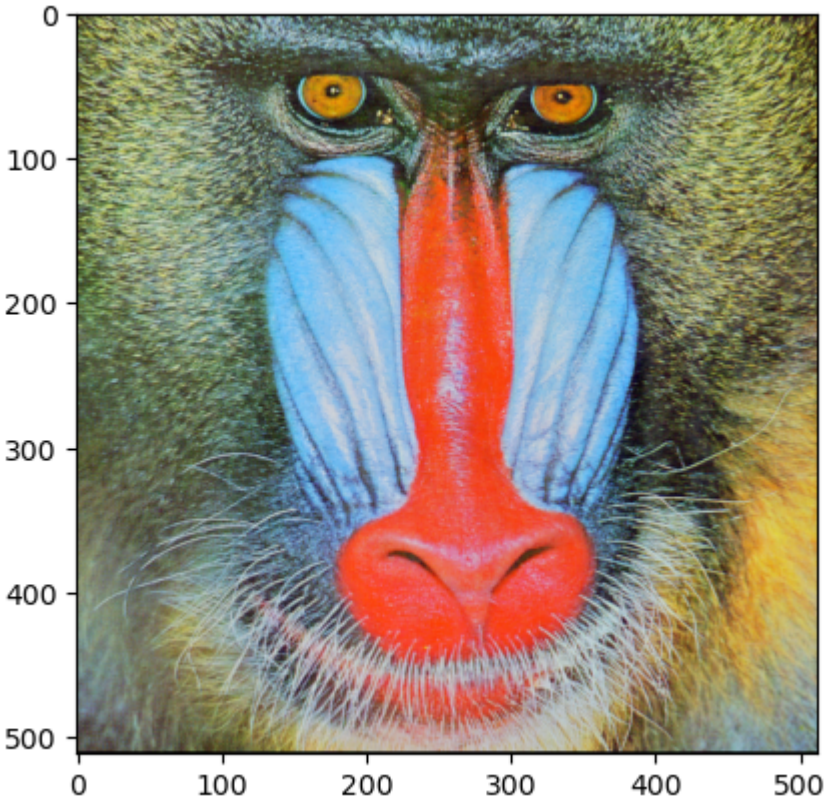
Appendix: Converting the image from BGR format to RGB format

In [3]:
```
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()
```


0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500