2.2.1_basic_image_manipulation_PIL

September 13, 2021

**

Manipulating Images

**

Estimated time needed: **60** minutes

Objectives

In this lab, you will learn how to manipulate images, both as arrays and PIL image objects. 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 15:34:12--  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 15:34:12 (139 MB/s) - 'cat.png' saved [663451/663451]

--2021-09-13 15:34:13--  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 0.003s

2021-09-13 15:34:14 (136 MB/s) - 'lenna.png' saved [473831/473831]

--2021-09-13 15:34:15--  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 0.004s

2021-09-13 15:34:15 (161 MB/s) - 'baboon.png' saved [637192/637192]
```

We will be using these imported functions in the lab:

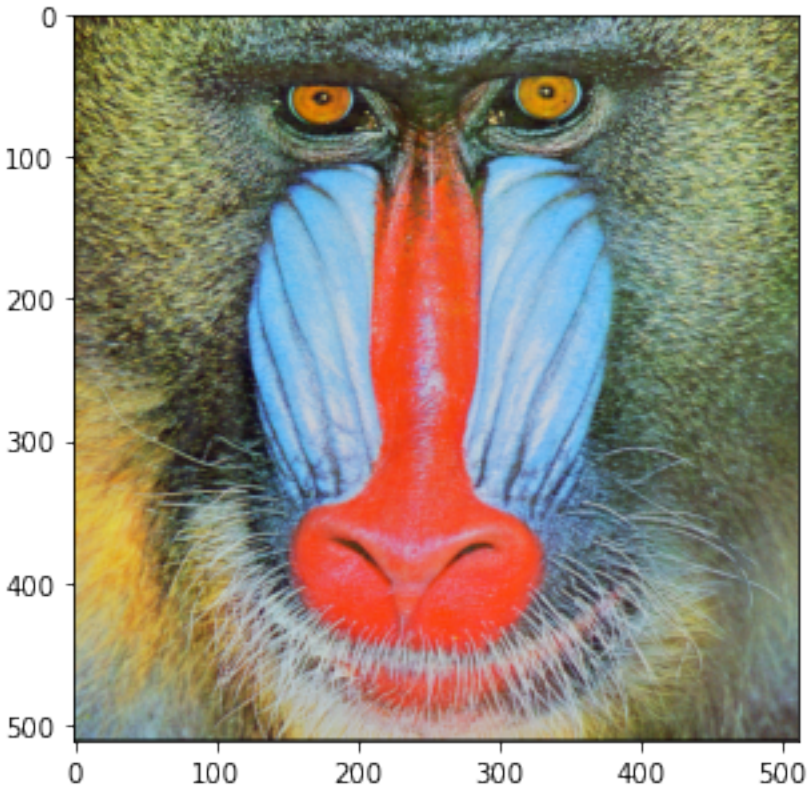
[2]:
```
import matplotlib.pyplot as plt
from PIL import Image
```

```
import numpy as np
```

0.1 Copying Images

If you want to reassign an array to another variable, you should use the `copy` method.

[3]:
```
baboon = np.array(Image.open('baboon.png'))
plt.figure(figsize=(5,5))
plt.imshow(baboon )
plt.show()
```



If we do not apply the method `copy()`, the two variables would point to the same location in memory:

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

We use the `id` function to find a variable's memory address; we see the objects in the memory space which the variables `A` and `baboon` point to are the same.

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

[5]: `True`

However, if we apply method `copy()`, their memory addresses are different.

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

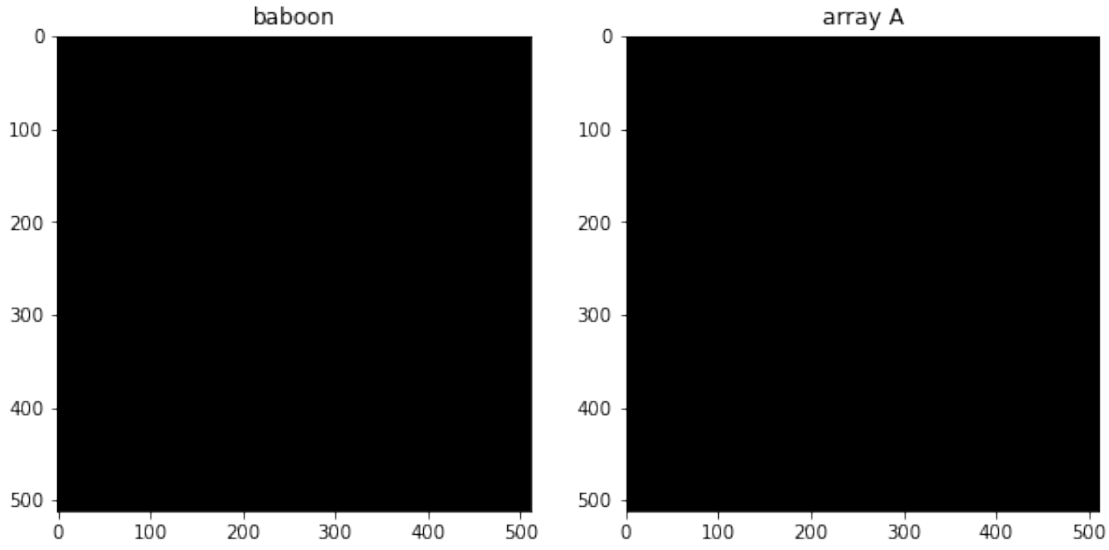
[6]:
```
False
```

There may be unexpected behaviours when two variables point to the same object in memory. Consider the array `baboon`. If we set all its entries to zero, all entires in `A` will become zero as well. However, as `baboon` and `B` points to different objects, the values in `B` will not be affected.

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

We can compare the variables baboon and array A:

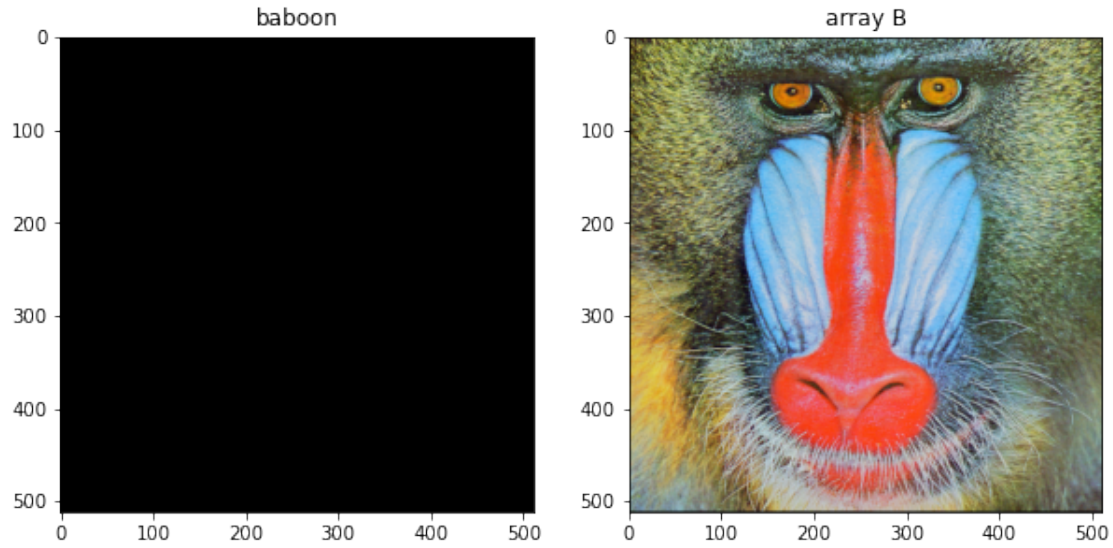
[8]:
```
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(baboon)
plt.title("baboon")
plt.subplot(122)
plt.imshow(A)
plt.title("array A")
plt.show()
```



We see they are the same. This behaviour is called aliasing. Aliasing happens whenever one variable's value is assigned to another variable. Variables are references to values on the memory.

We can also compare baboon and array B:

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

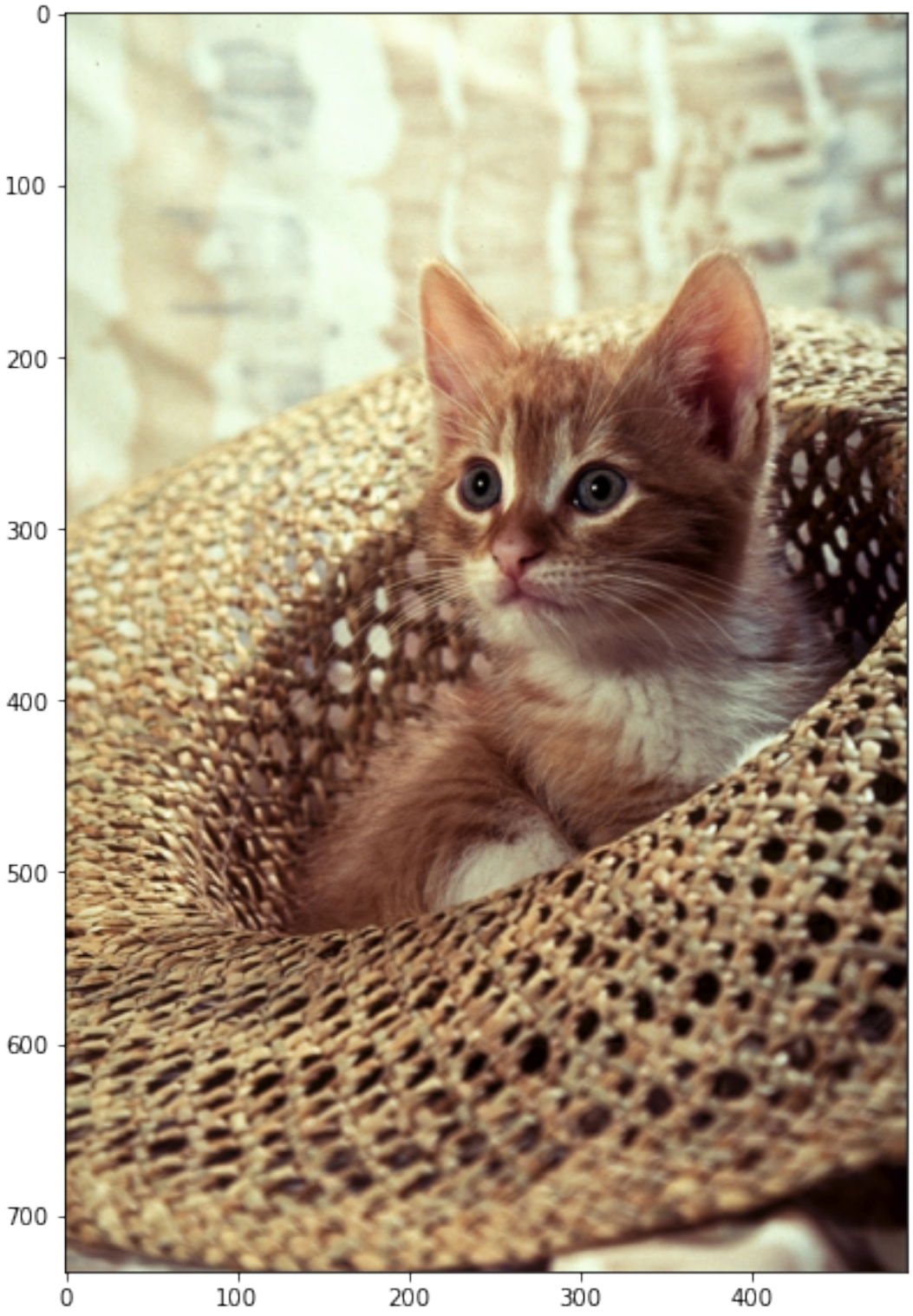


If a PIL function does not return a new image, the same principle applies. We will go over some examples later on.

0.2 Flipping Images

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

[10]:
```
image = Image.open("cat.png")
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.show()
```



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

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

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

Let's flip, i.e rotate vertically, the image. Let's try the traditional approach. First, we create an array of the same size with datatype np.uint8.

[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 from the original array and decreasing the new array's row index to assign the pixels accordingly. After excecuting the for loop below, `array_flip` will become the flipped image.

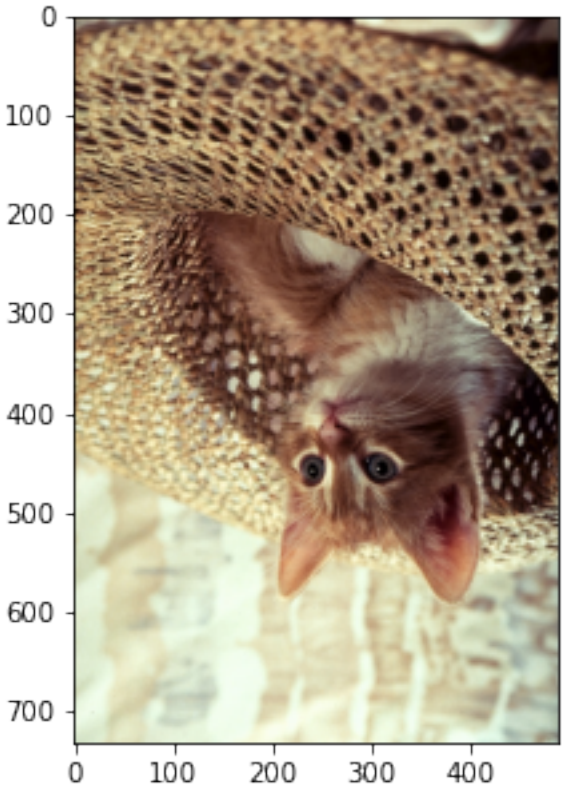
[13]:
```
for i,row in enumerate(array):
    array_flip[width - 1 - i, :, :] = row
```

PIL has several ways to flip an image, for example, we can use the ImageOps module:

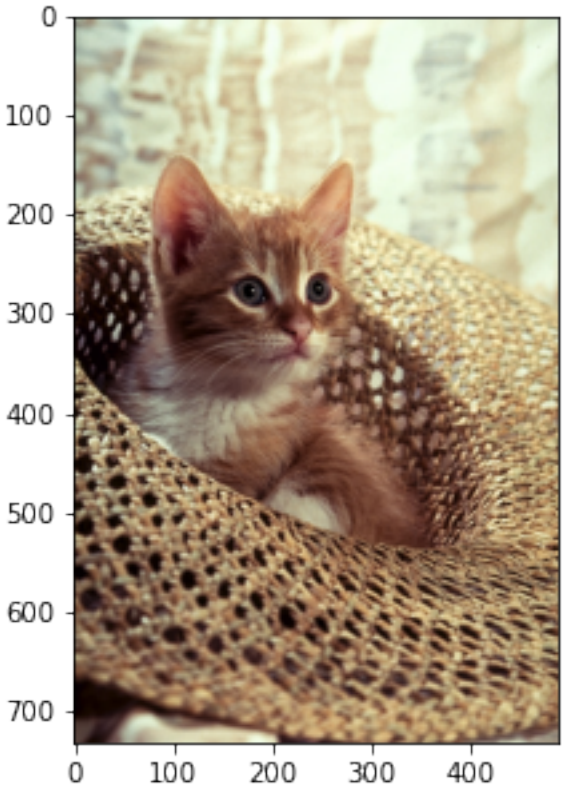
[14]:
```
from PIL import ImageOps
```

The `flip()` method of `ImageOps` module flips the image. The `mirror()` method will mirror the image:

[15]:
```
im_flip = ImageOps.flip(image)
plt.figure(figsize=(5,5))
plt.imshow(im_flip)
plt.show()
```

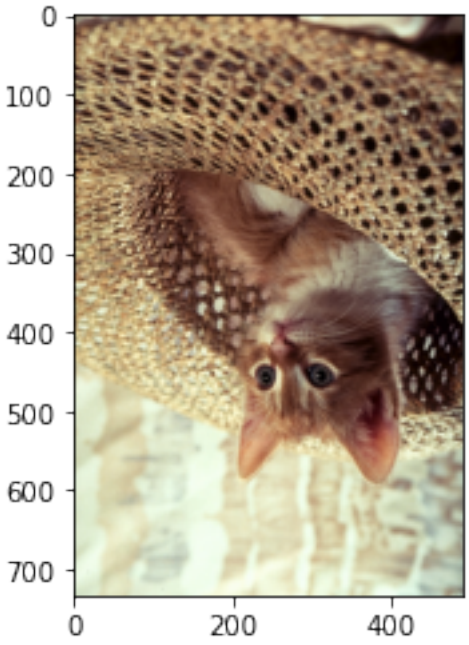


[16]:
```
im_mirror = ImageOps.mirror(image)
plt.figure(figsize=(5,5))
plt.imshow(im_mirror)
plt.show()
```



We can use the transpose() method; the parameter is an integer indicating what type of transposition we would like to perform. For example, we can flip the image using a value of `1`.

[17]:
```
im_flip = image.transpose(1)
plt.imshow(im_flip)
plt.show()
```



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

[18]:
```
flip = {"FLIP_LEFT_RIGHT": Image.FLIP_LEFT_RIGHT,
        "FLIP_TOP_BOTTOM": Image.FLIP_TOP_BOTTOM,
        "ROTATE_90": Image.ROTATE_90,
        "ROTATE_180": Image.ROTATE_180,
        "ROTATE_270": Image.ROTATE_270,
        "TRANSPOSE": Image.TRANSPOSE,
        "TRANSVERSE": Image.TRANSVERSE}
```

We see the values are integers.

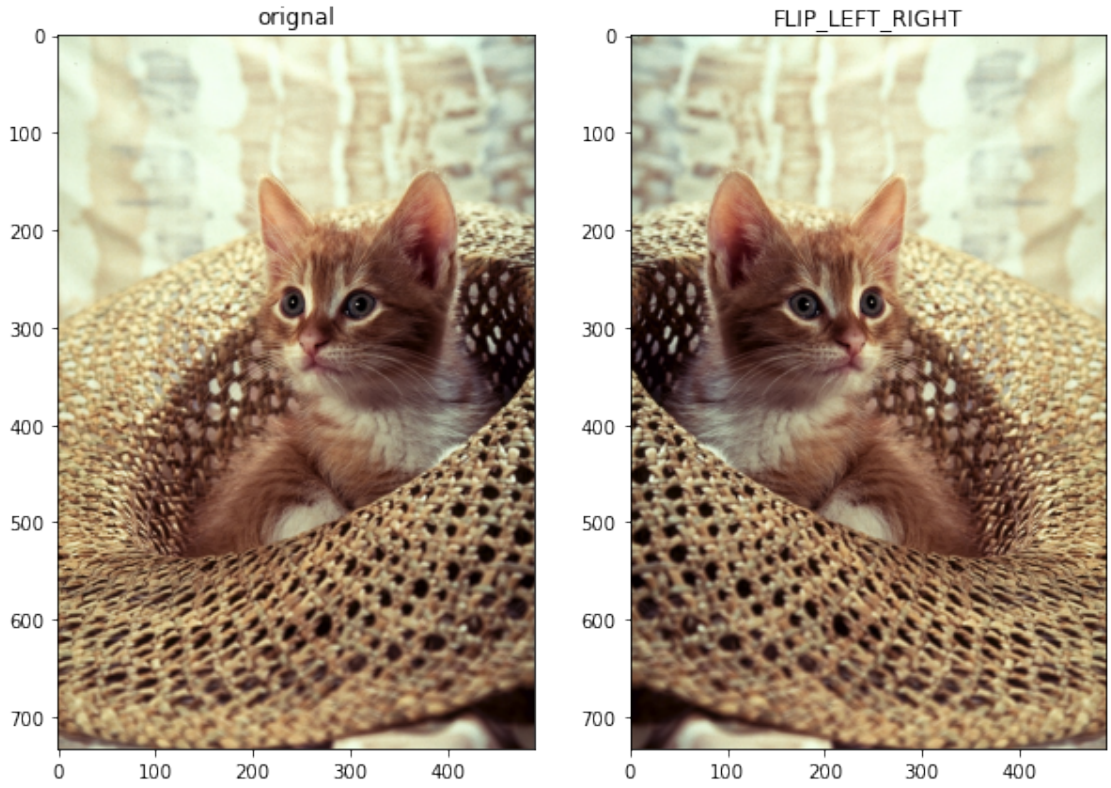
[19]:
```
flip["FLIP_LEFT_RIGHT"]
```

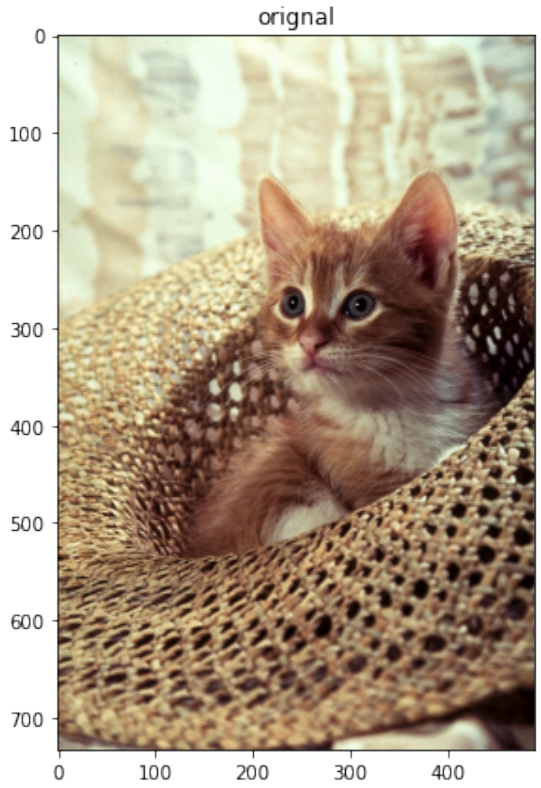
[19]:
```
0
```

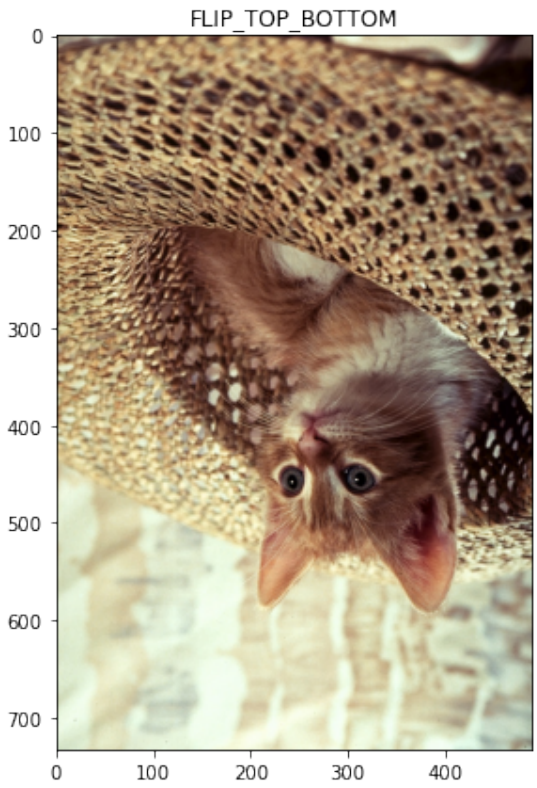
We can plot each of the outputs using the different parameter values:

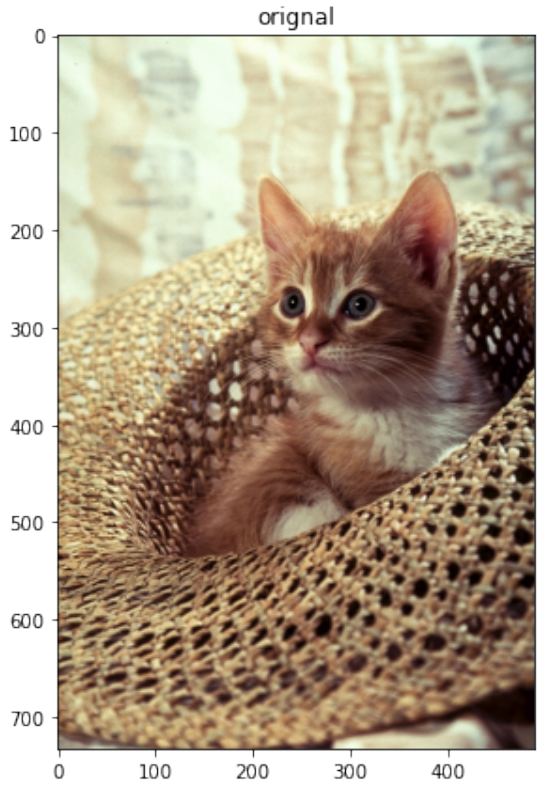
[20]:
```
for key, values in flip.items():
    plt.figure(figsize=(10,10))
    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.title("orignal")
    plt.subplot(1,2,2)
    plt.imshow(image.transpose(values))
```

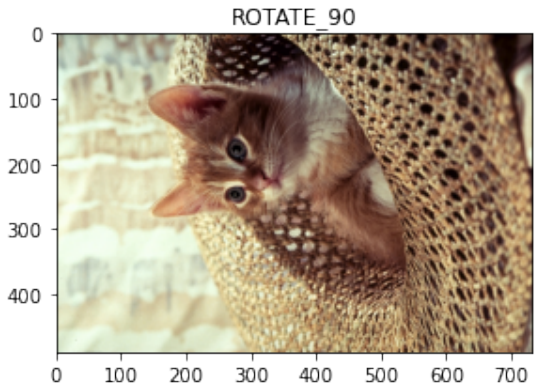
```
    plt.title(key)
    plt.show()
```

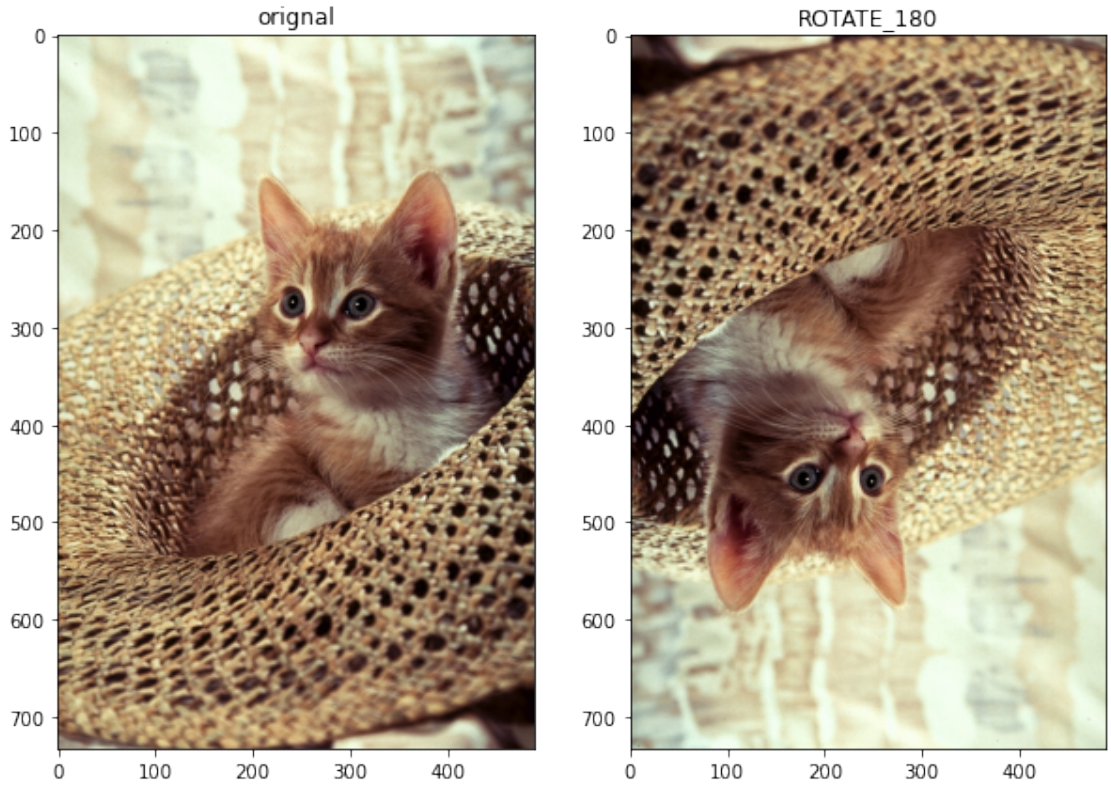


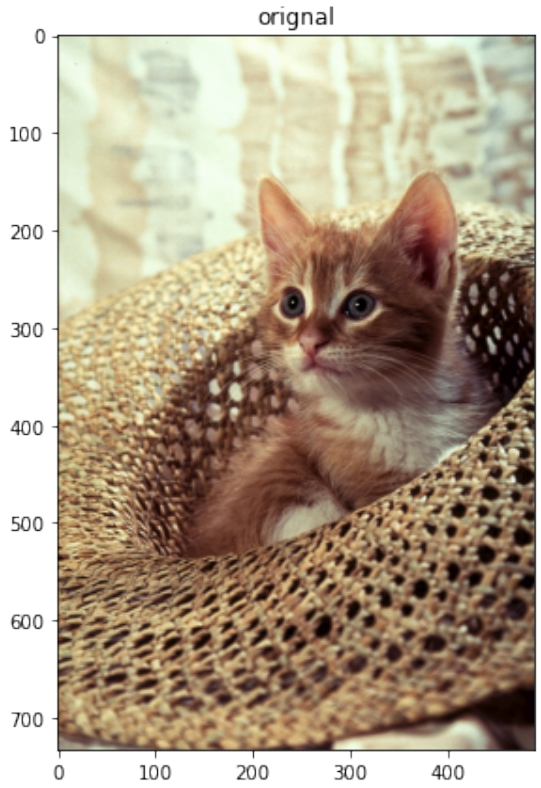

orignal
0
100
200
300
400
500
600
700
0
100
200
300
400

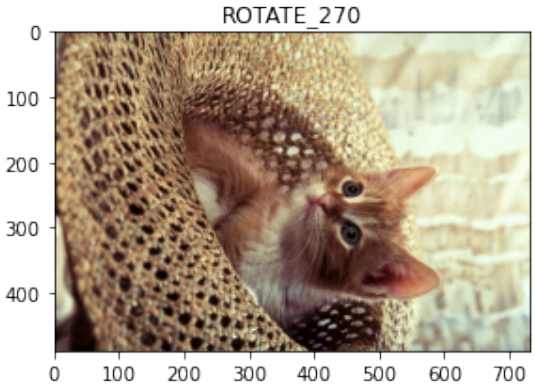

FLIP_TOP_BOTTOM
0
100
200
300
400
500
600
700
0
100
200
300
400

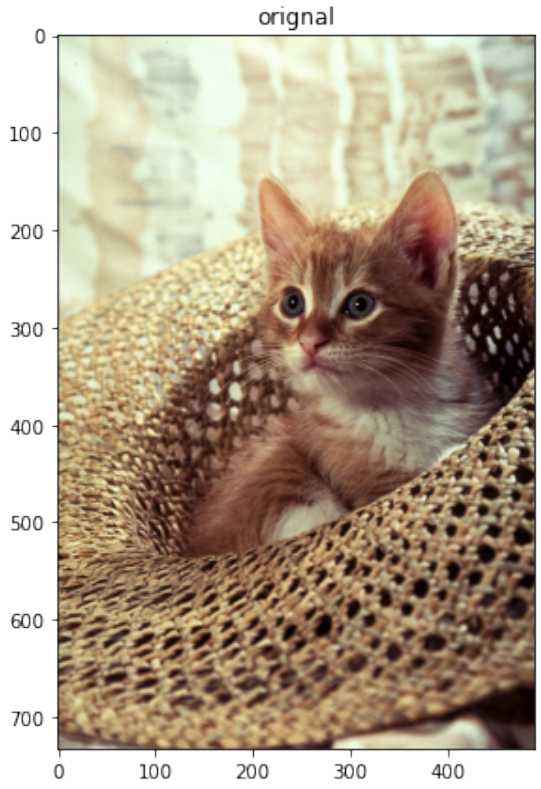

orignal
0
100
200
300
400
500
600
700
0
100
200
300
400

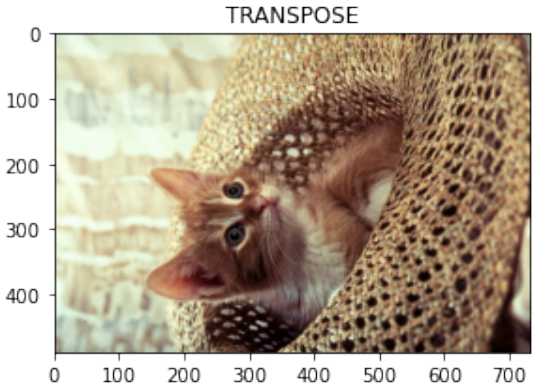

ROTATE_90
0
100
200
300
400
0
100
200
300
400
500
600
700

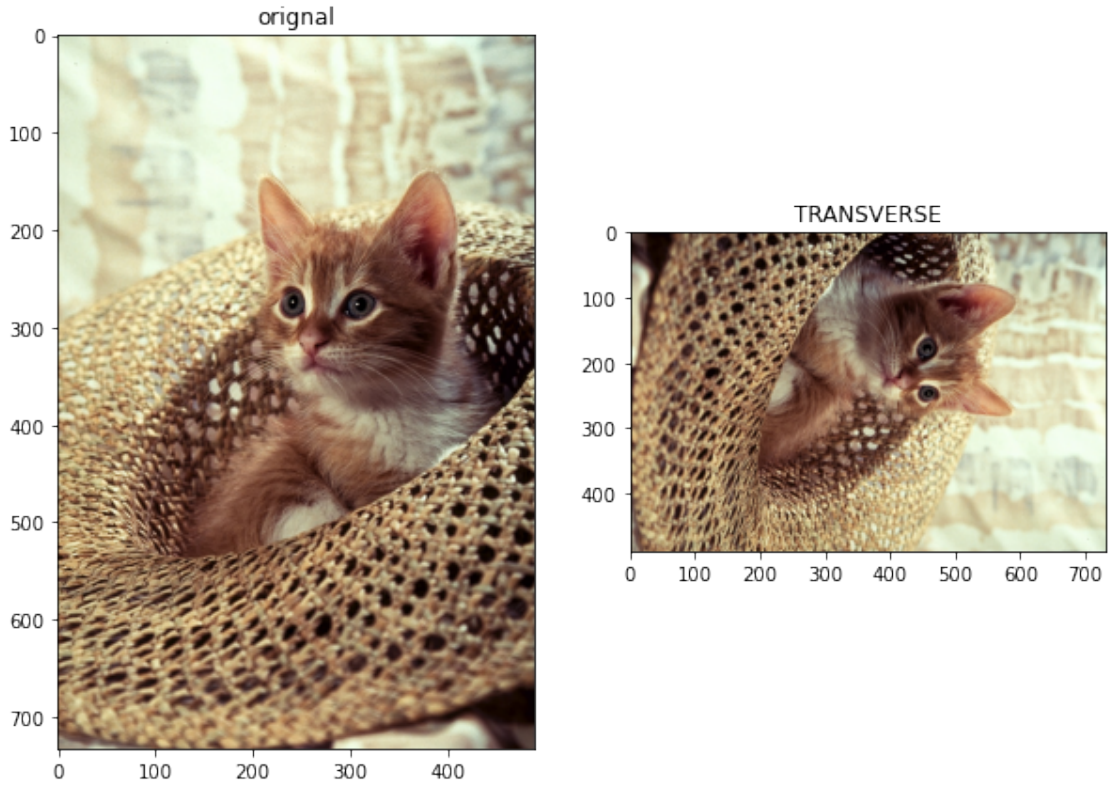

orignal
ROTATE_180
0
100
200
300
400
500
600
700


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


ROTATE_270
0
100
200
300
400
0
100
200
300
400
500
600
700


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


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

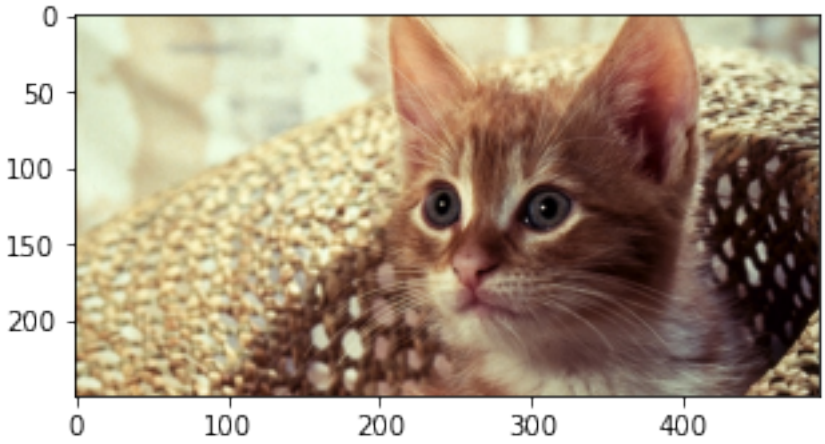


0.3 Cropping an Image

Cropping is the act of "cutting out" a part of an image and throwing out the rest. We can perform cropping using array slicing.

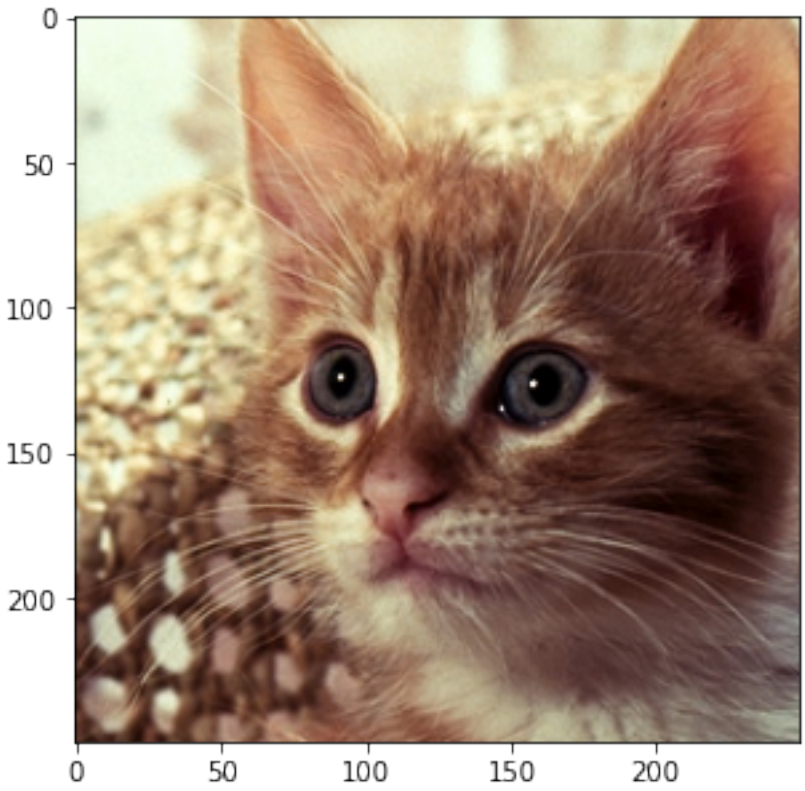
Let start with a vertical crop: the variable `upper` is the index of the first row that we would like to include in the image, the variable `lower` is the index of the last row we would like to include. We then perform array slicing to obtain the new image.

[21]:
```
upper = 150
lower = 400
crop_top = array[upper: lower,:,:]
plt.figure(figsize=(5,5))
plt.imshow(crop_top)
plt.show()
```



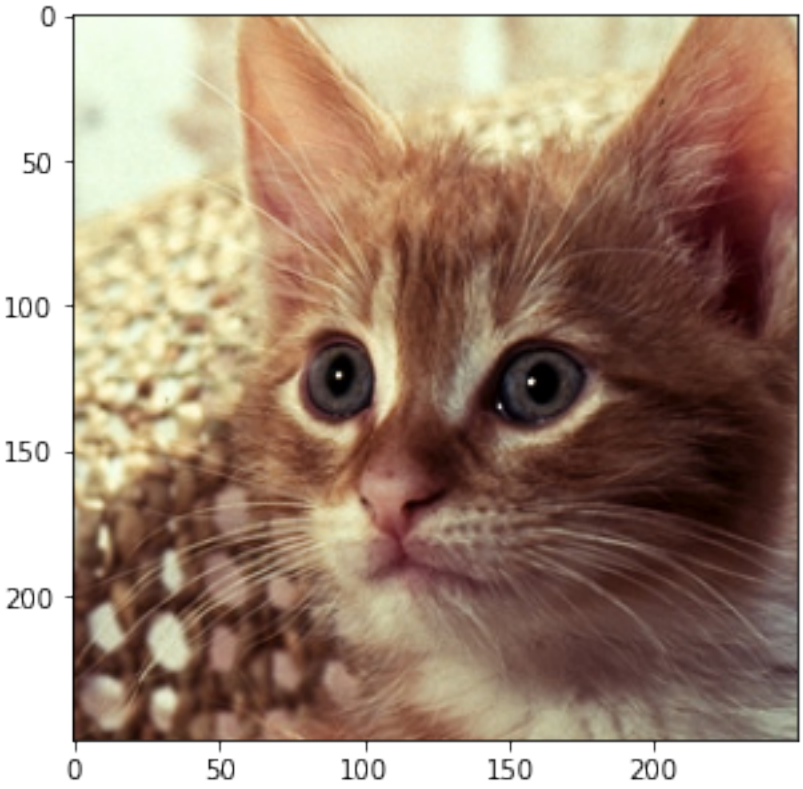
Consider the array `crop_top`: we can also crop horizontally. The variable `right` is the index of the first column that we would like to include in the image and the variable `left` is the index of the last column we would like to include in the image.

[22]:
```
left = 150
right = 400
crop_horizontal = crop_top[: ,left:right,:]
plt.figure(figsize=(5,5))
plt.imshow(crop_horizontal)
plt.show()
```



You can crop the PIL image using the crop() method, using the parameters from above Set the cropping area with box=(left, upper, right, lower).

[23]:
```
image = Image.open("cat.png")
crop_image = image.crop((left, upper, right, lower))
plt.figure(figsize=(5,5))
plt.imshow(crop_image)
plt.show()
```



We can also flip the new image:

[24]:
```
crop_image = crop_image.transpose(Image.FLIP_LEFT_RIGHT)
crop_image
```

[24]:

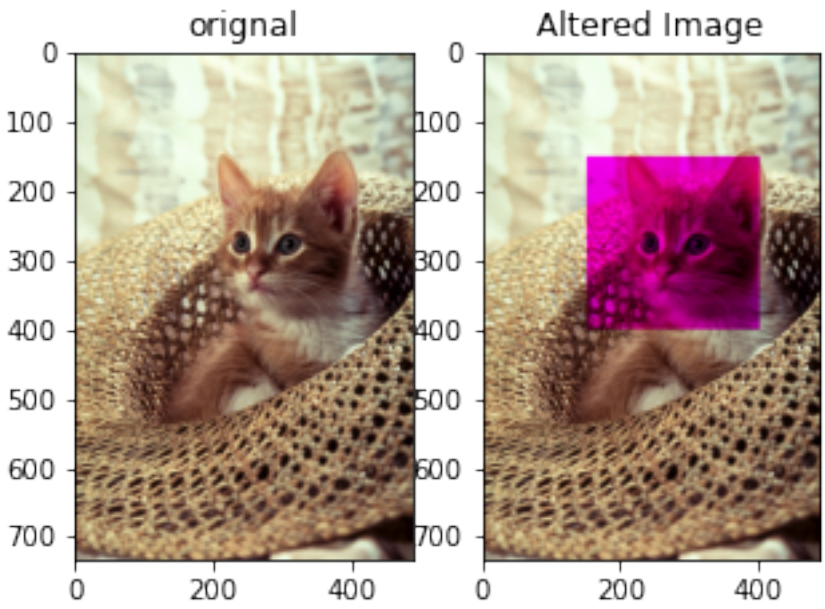
0.4 Changing Specific Image Pixels

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

[25]:
```
array_sq = np.copy(array)
array_sq[upper:lower, left:right, 1:2] = 0
```

We can compare the results with the new image.

[26]:
```
plt.figure(figsize=(5,5))
plt.subplot(1,2,1)
plt.imshow(array)
plt.title("orignal")
plt.subplot(1,2,2)
plt.imshow(array_sq)
plt.title("Altered Image")
plt.show()
```



We can also use the ImageDraw module from PIL library, which provides simple 2D graphics for Image objects

[27]:
```
from PIL import ImageDraw
```

We will copy the image object:

[28]:
```
image_draw = image.copy()
```

The draw constructor creates an object that can be used to draw in the given image. The input im is the image we would like to draw in.

[29]:
```
image_fn = ImageDraw.Draw(im=image_draw)
```

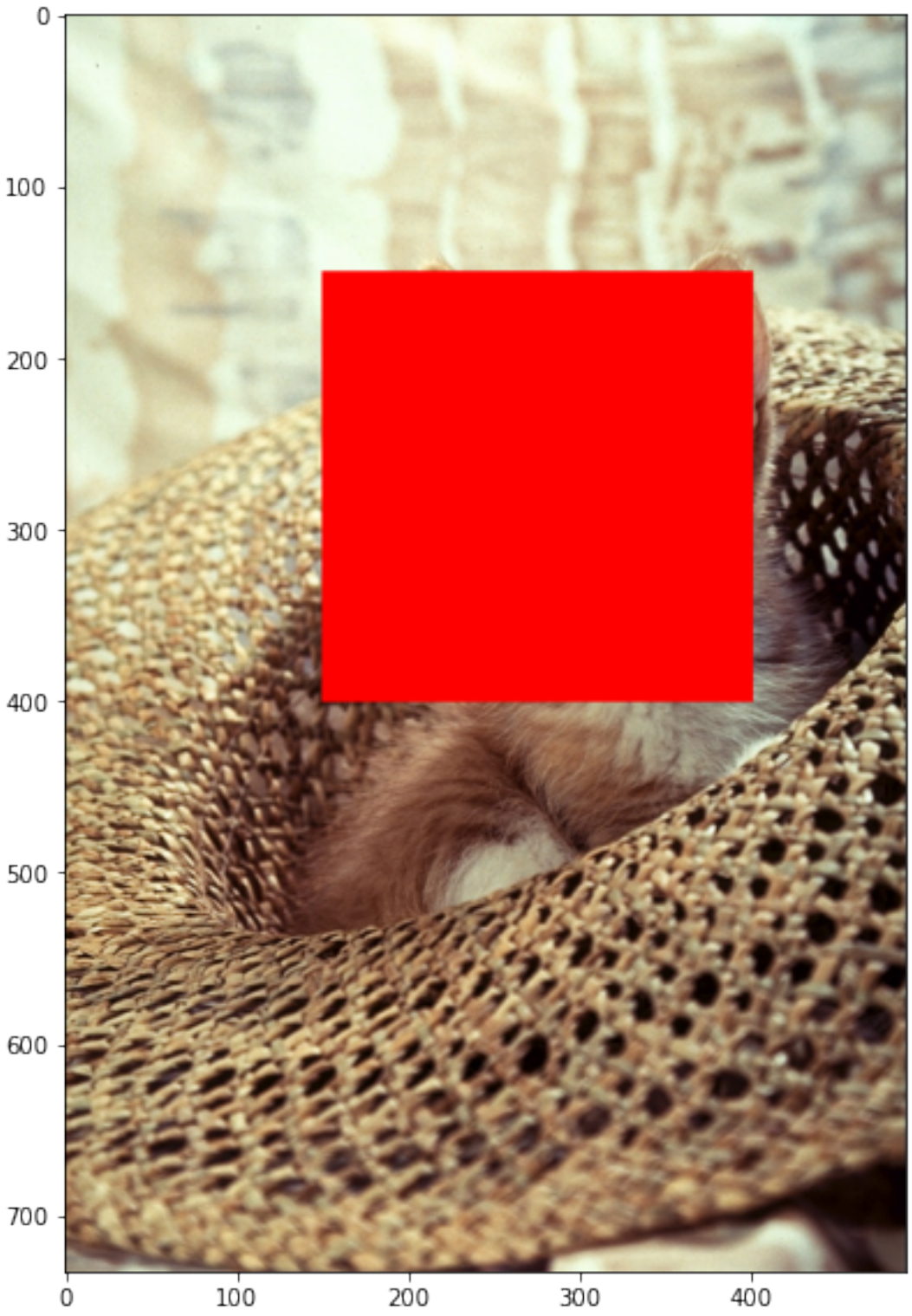
Whatever method we apply to the object image_fn, will change the image object image_draw.

We can draw a rectangle using the rectangle function, two important parameters include: xy -- the coordinates bounding box and fill -- Color of the rectangle.

[30]:
```
shape = [left, upper, right, lower]
image_fn.rectangle(xy=shape,fill="red")
```

We can plot the image.

[31]:
```
plt.figure(figsize=(10,10))
plt.imshow(image_draw)
plt.show()
```



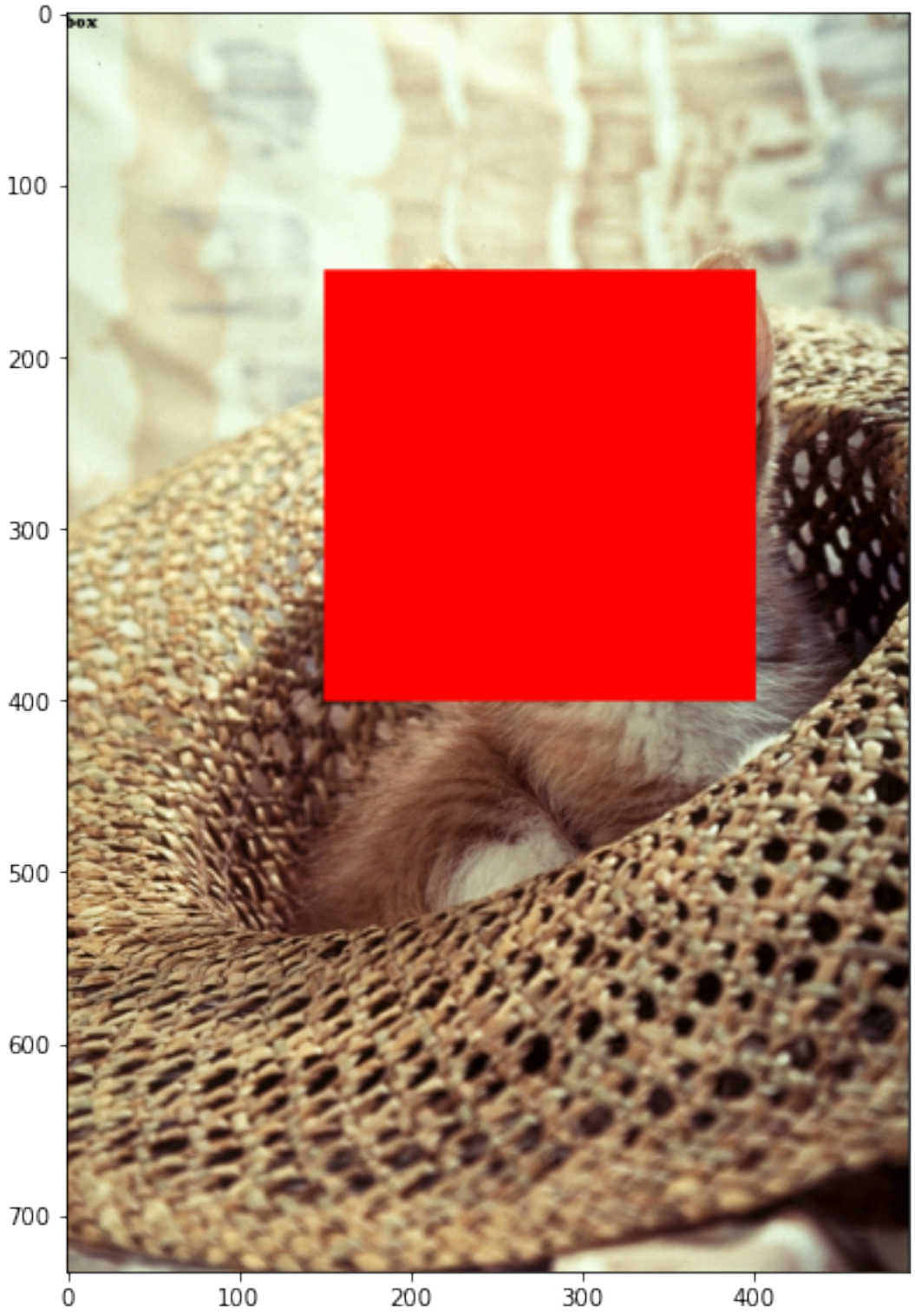
There are other shapes we can use. For example, we can also overlay text on an image: we use the ImageFont module to obtain bitmap fonts.

[32]:
```
from PIL import ImageFont
```

We use the text method to place the text on the image. The parameters include xy (the top-left anchor coordinates of the text), the parameter text (the text to be drawn), and fill (the color to use for the text).

[33]:
```
image_fn.text(xy=(0,0),text="box",fill=(0,0,0))
```

[34]:
```
plt.figure(figsize=(10,10))
plt.imshow(image_draw)
plt.show()
```

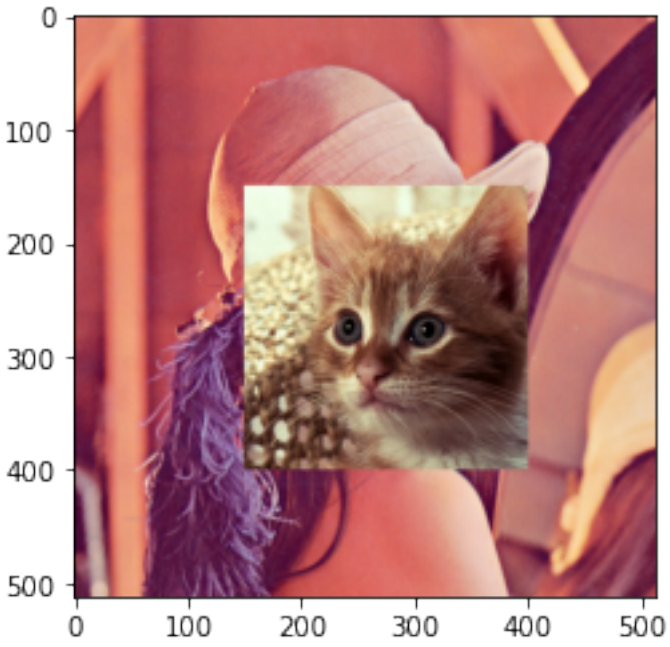


We can overlay or paste one image over another by reassigning the pixel for one array to the next. Consider the following image array:

[35]:
```
image_lenna = Image.open("lenna.png")
array_lenna = np.array(image_lenna)
```

We can reassign the pixel values as follows:

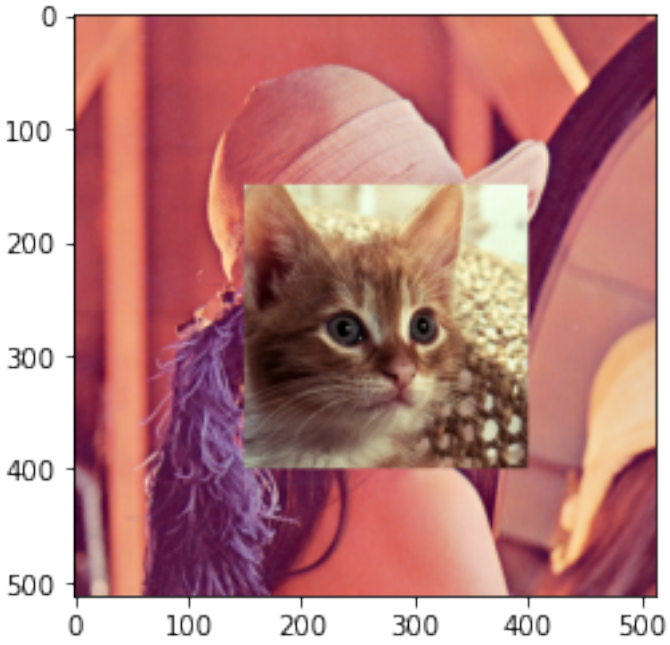
[36]:
```
array_lenna[upper:lower,left:right,:]=array[upper:lower,left:right,:]
plt.imshow(array_lenna)
plt.show()
```



In the same manner, the paste() method allows you to overlay one image over another, with the input being the image you would like to overlay. The parameter box is the left and upper coordinate of the image:

[37]:
```
image_lenna.paste(crop_image, box=(left,upper))
```

[38]:
```
plt.imshow(image_lenna)
plt.show()
```



We can see the method copy() applies to some PIL objects. We create two image objects, we set new_image to the image, and we use the method copy() for the copy_image object.

[39]:
```
image = Image.open("cat.png")
new_image=image
copy_image=image.copy()
```

Like the array, we see that the same memory address relationship exists. For example, if we don't use the method copy(), the image object has the same memory address as the original PIL image object.

[40]:
```
id(image)==id(new_image)
```

[40]: True

If we use the method copy(), the address is different:

[41]:
```
id(image)==id(copy_image)
```

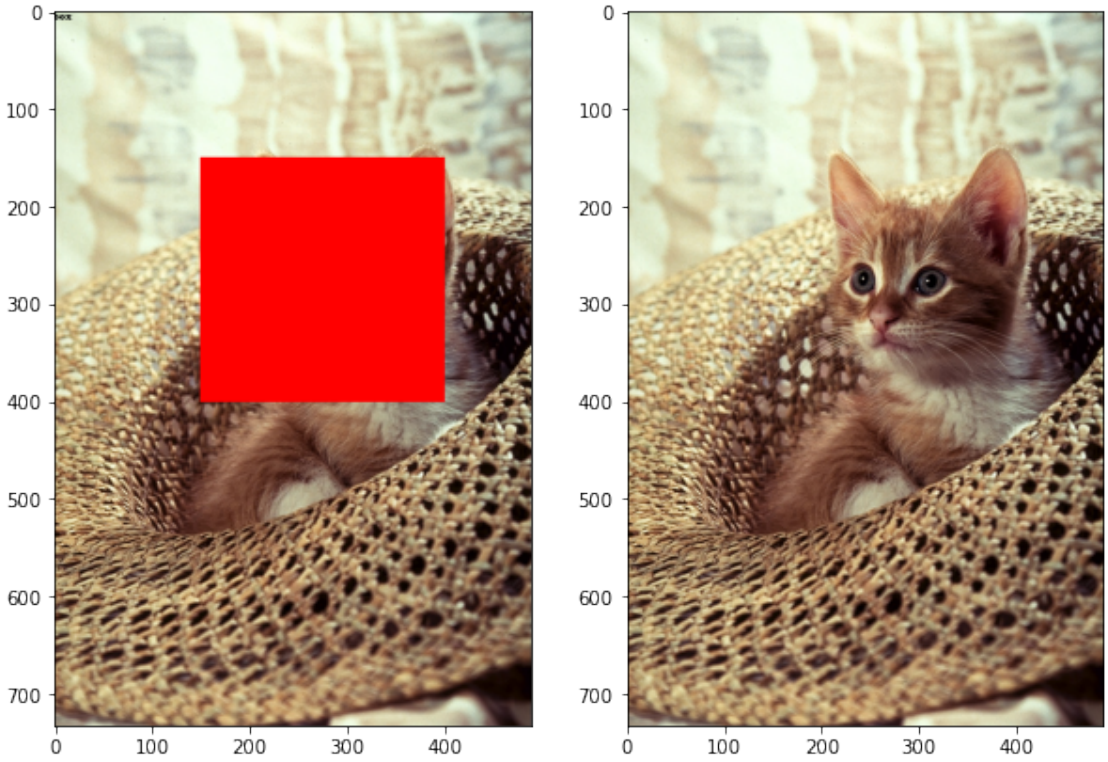
[41]: False

If we change the object image, new_image will change, but copy_image will remain the same:

[42]:
```
image_fn= ImageDraw.Draw(im=image)
image_fn.text(xy=(0,0),text="box",fill=(0,0,0))
```

```
image_fn.rectangle(xy=shape,fill="red")
```

[43]:
```
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(new_image)
plt.subplot(122)
plt.imshow(copy_image)
plt.show()
```



0.4.1 Question 1:

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

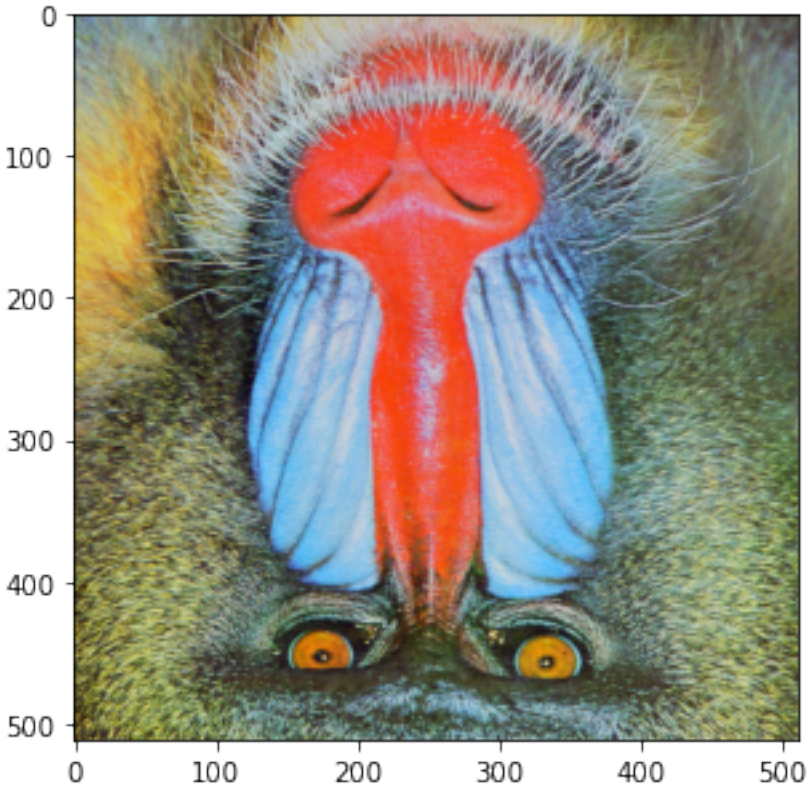
Open the image and create a PIL Image object called im, flip im and create an image called im_flip. Mirror im and create an image called im_mirror. Finally, plot both images.

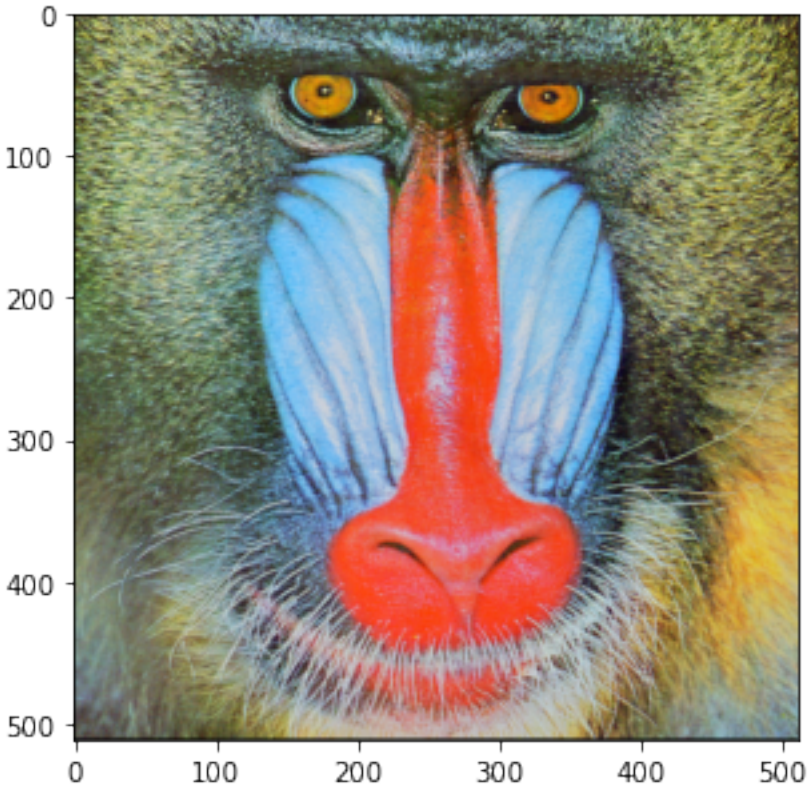
[45]:
```
# write your script here
im = Image.open("baboon.png")

im_flip = ImageOps.flip(im)
plt.figure(figsize=(5,5))
plt.imshow(im_flip)
```

```
plt.show()

im_mirror = ImageOps.mirror(im)
plt.figure(figsize=(5,5))
plt.imshow(im_mirror)
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