

TI Image

Transformations

After converting an image to grayscale in the first activity, let's look at some *transformational* image processing that can be done using the **ti_image** module. Transformations include effects such as flipping, mirroring and rotating the entire image.

Introduction: In the "Getting Started..." activity you downloaded a working TI-Nspire document found at [Image Processing Activities.tns](#). Make another copy of the startup code (pythonimage.py) in the document to work on this project, too.

1. To work with your own image, after inserting an image into your TI-Nspire document (on a **Notes** app), it will appear like this. Name the image by right-clicking (press **[ctrl]-[menu]** on the handheld) the image and selecting 'Name Image'. Use that name in your Python programs.



2. To **flip** an image (vertically) or **mirror** an image (horizontally), get the value of each pixel and place it in a new (opposite) position in a new image. Let's flip an image first.

Use the 'generic' image processing code provided in the document to load and show the image.

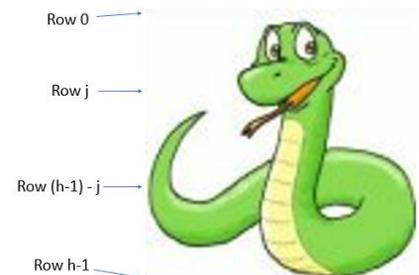
Use the same nested **for** loops for the rows and columns.

```

1.1 1.2 1.3 *image pr...hon RAD
pythonimage0.py 18/41
=====
W,H = get_screen_dim()
img=load_image("python0")
w,h=img.w, img.h
img.show_image((W-w)/2,(H-h)/2)
=====
# Flip...
img2=new_image(w,h,(0,0,0))
for j in range(h):
    for i in range(w):
        |
    
```

3. 'Flip' means to transfer pixels between top and bottom. The top row is row 0 and the bottom row is row (h-1). Python counting always starts with 0.

In general, row **j** from the top of the old image moves to **row (h-1) - j** from the bottom in the new image.





4. We can perform this transformation in just one or two statements: get the value of pixel (i, j) from the original image and place it in position (i, (h-1) – j) in the new image:

```
c = img.get_pixel(i, j)
img2.set_pixel(i, (h - 1) - j, c)
```

These two statements can be combined into just one statement:

```
img2.set_pixel(i, (h-1)-j, img.get_pixel(i, j))
```

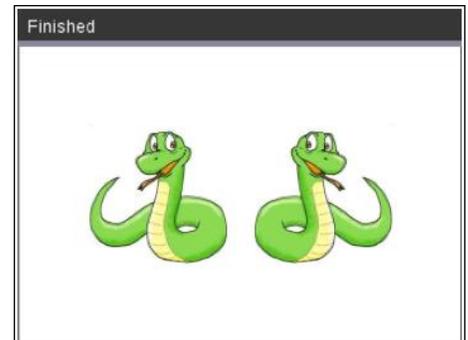
```
1.1 1.2 1.3 *image pr... hon RAD 32/43
*pythonimage0.py
# Flip...
img2=new_image(w,h,(0,0,0))
for j in range(h):
  for i in range(w):
    img2.set_pixel(i, (h-1)-j, img.get_pixel(i, j))
img2.show_image((W-w)/2,(H-h)/2)
```



5. To **mirror** the original image (left <-> right), perform a similar operation on the columns instead of the rows.

To show both images at once just change the positions in

```
img.show_image( , )
```



6. **Challenge 1: Efficiency**

Perform the transformation (flip or mirror) using only *one* image variable, not two. Note that this *does* require modifying the *original* image, but only within the program, not the actual image in the Notes app of the document. Would the program perform faster?





10 MOC: Python Modules

TI-NSPIRE™ CX II PYTHON

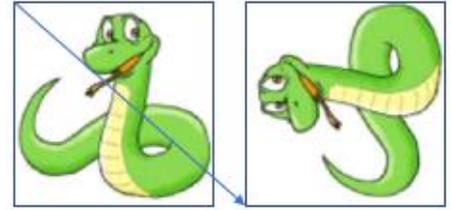
TI IMAGE: IMAGE TRANSFORMATIONS

7. Challenge 2: Rotate

Use the same image processing technique to rotate the image.

Hint: row **i** from the top of the original image moves to **column i** (from the left) in the new image working from *bottom* to *top*.

$$\text{Pixel } \begin{matrix} \text{row} & \text{col} \\ \mathbf{(i, j)} \end{matrix} \rightarrow \begin{matrix} \text{row} & \text{col} \\ \mathbf{(j, (h - 1) - i)} \end{matrix}.$$



This will rotate the image **90 degrees counter-clockwise**. You can also rotate 180 degrees (that's not the same as flipping!) or 90 degrees clockwise.



Teacher Tip: About the Challenges.

Challenge 1: To flip (top to bottom) or mirror (left to right) using only one image variable, process just *one half* of the image, *swapping* pixels in the first half with the corresponding pixel in the other half. It requires **temporarily** storing the color of the first pixel.

Sample code:

```
# mirror with only one image var...
W, H = get_screen_dim()
img=load_image("python0")
w,h=img.w, img.h
for j in range(h):
    for i in range(w//2):
        temp=img.get_pixel(i, j)
        img.set_pixel(i, j, img.get_pixel(w-1-i,j))
        img.set_pixel(w-1-i, j, temp)
img.show_image((W-w)/2, (H-h)/2)
```

The program will *not* run significantly faster. Most of the processing time is used by the **get_** and **set_** functions. Actual performance will vary depending on your hardware and software. To test speed performance of any code on your system, add a **time** statement supplied with **ti_system**:

```
# Mirror...
from ti_system import *
#
img2=new_image(w,h,(0,0,0))
t0=get_time_ms() # found on [menu] > More Modules > ti_system
for j in range(h):
    for i in range(w):
        img2.set_pixel(w-1-i,j,img.get_pixel(i,j))
print(get_time_ms() - t0)
```

Challenge 2: Rotations

Sample code using the same nested **for** loops as the other transformations:

```
# left 90...
img2.set_pixel(j, (h-1)-i,img.get_pixel(i,j))

# right 90...
(i, j) → ((h-1)-j, i)

# left 180...
(i, j) → ((w-1)-i, (h-1)-j)
```