Images + PyCharm
by Ecy!
Housekeeping

- Assignment 1, Bit is due this Friday, July 7th at 11:59 pm
  - with Grace Period until Saturday, July 8th at 11:59 pm
- Second section happening this week
- Coding can be difficult, but rewarding, keep at it!
Note on Style

- **Style is an important part of CS106A**
  - Descriptive variable names
  - Decomposition can be super duper useful
  - Write inline and function header comments
  - Have good formatting (spacing)

*Style guidelines linked here!*
Today

- **Recap Images**
  - Image functions, pixels
  - Double for-loop
  - Code demo

- **Look at New Functionality**
  - How do we make new, blank images?
  - How do we make two pixels the same?

- **Intro PyCharm**
  - How will we use PyCharm?
  - How can we run a program in PyCharm?
  - How can make a "bluescreen"?
Image Recap
Images
Images are made of pixels that we can loop over with their x, y coordinates. We can load image files into variables using Simple Image.

Pixels
Pixels have red, green, and blue attributes. We can grab a pixel at x, y in an image with the get_pixel() function.

Double For Loops
Double (or nested) for loops get us all possible x, y combos and thus, all possible coordinates. Thus, we can access every single pixel.
Images

Images are made of pixels that we can loop over with their x, y coordinates.

We can load image files into variables using Simple Image.

```python
# we can now treat the image like a variable
image = SimpleImage('tree.jpg')
```
Pixels

Pixels have red, green, and blue attributes. We can grab a pixel at $x, y$ in an image with the `get_pixel()` function. Often we either store or change a pixel's value.

```
# we can now treat the image like a variable
image = SimpleImage('tree.jpg')
# we have access to the pixel now!
pixel = image.get_pixel(4, 0)
```
We can store and change a pixel's value.

# we can now treat the image like a variable
image = SimpleImage('tree.jpg')
# we have access to the pixel now!
pixel = image.get_pixel(4, 0)
# we can change the pixel's values
pixel.red = 255
pixel.green = 189
pixel.blue = 89
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
```
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(0, image.height):  # is 4
    for x in range(0, image.width):  # is 5
```
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
```
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height): # is 4
    for x in range(image.width): # is 5
        pixel = image.get_pixel(x,y)
```

![Diagram of pixel grid and image with various colors]
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height): # is 4
    for x in range(image.width): # is 5
        pixel = image.get_pixel(x,y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height): # is 4
    for x in range(image.width): # is 5
        pixel = image.get_pixel(x,y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height): # is 4
    for x in range(image.width): # is 5
        pixel = image.get_pixel(x,y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
        pixel.red = pixel.red * 0.5
        pixel.green = pixel.green * 0.5
        pixel.blue = pixel.blue * 0.5
    return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
    return image
```
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x,y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
pixel.red = pixel.red*0.5
pixel.green = pixel.green*0.5
pixel.blue = pixel.blue*0.5
return image
```
Double For Loop

What if we wanted to go through all of the pixels and half their color?

image = SimpleImage('tree.jpg')

for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x,y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5

return image
Double For Loop

What if we wanted to go through all of the pixels and half their color?

```python
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
return image
```
What if we wanted to go through all of the pixels and half their color?

```
image = SimpleImage('tree.jpg')
for y in range(image.height):  # is 4
    for x in range(image.width):  # is 5
        pixel = image.get_pixel(x, y)
        pixel.red = pixel.red*0.5
        pixel.green = pixel.green*0.5
        pixel.blue = pixel.blue*0.5
    return image
```
Double For Loop

What if we wanted to go through all of the pixels and half their color?

Darker Nested
**Image Functions**

- `image = SimpleImage(filename)`
- `width = image.width`
- `height = image.height`
- `pixel = image.get_pixel(x, y)`

**Pixel Attributes and Functionality**

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255 # set pixel to exact color`
New Functionality
Image Functions

- `image = SimpleImage(filename)`
- `out = SimpleImage.blank(width, height)`
- `width = image.width`
- `height = image.height`
- `pixel = image.get_pixel(x, y)`

Pixel Attributes and Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255 # set pixel to exact color`
- `pixel_out.red = pixel.red # assuming pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`
New Image Functions

- `image = SimpleImage(filename)`
- `out = SimpleImage.blank(width, height)`

# Create a blank image of custom width, height
out = SimpleImage.blank(5, 4)
New Image Functions

- `image = SimpleImage(filename)`
- `out = SimpleImage.blank(width, height)`

# Create a blank image of custom width, height
out = SimpleImage.blank(5, 4)
New Image Functions

- `image = SimpleImage(filename)`
- `out = SimpleImage.blank(width, height)`

```python
image = SimpleImage('tree.jpg')
width = image.width
height = image.height
# Create a blank image twice as wide as the OG
out = SimpleImage.blank(width*2, height)
```

![original image](tree.jpg)

New image twice as wide and BLANK
New Pixel Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255`
- `pixel_out.red = pixel.red` # assuming `pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`

# GOAL: Set one pixel to another pixel's value
New Pixel Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255`
- `pixel_out.red = pixel.red`  # assuming `pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`

# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
```
# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
out = SimpleImage.blank(10, 4)  # create out

# pixel manipulation
pixel.red, pixel.blue, pixel.green
pixel.red = 255

# New Pixel Functionality
pixel_out.red = pixel.red  # assuming pixel_out
pixel_out.green = pixel.green
pixel_out.blue = pixel.blue
```

# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
out = SimpleImage.blank(10, 4)  # create out

# pixel manipulation
pixel.red, pixel.blue, pixel.green
pixel.red = 255

# New Pixel Functionality
pixel_out.red = pixel.red  # assuming pixel_out
pixel_out.green = pixel.green
pixel_out.blue = pixel.blue
```
# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')
out = SimpleImage.blank(10, 4)
pixel = image.get_pixel(4,0)

# New Pixel Functionality
pixel.red, pixel.blue, pixel.green
pixel.red = 255
pixel_out.red = pixel.red  # assuming pixel_out
pixel_out.green = pixel.green
pixel_out.blue = pixel.blue
```

# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
out = SimpleImage.blank(10, 4)  # create out
pixel = image.get_pixel(4,0)  # get original pixel
```
# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
out = SimpleImage.blank(10, 4)  # create out
pixel = image.get_pixel(4, 0)  # get original pixel
pixel_out = out.get_pixel(0, 0)  # get blank pixel
```

```python
pixel.red, pixel.blue, pixel.green
pixel.red = 255
pixel_out.red = pixel.red  # assuming pixel_out
pixel_out.green = pixel.green
pixel_out.blue = pixel.blue
```

New Pixel Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255`
- `pixel_out.red = pixel.red` # assuming `pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`

# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
out = SimpleImage.blank(10, 4)  # create out
pixel = image.get_pixel(4, 0)  # get original pixel
pixel_out = out.get_pixel(0, 0)  # get blank pixel
```
# GOAL: Set one pixel to another pixel's value

```python
image = SimpleImage('tree.jpg')  # get OG image
out = SimpleImage.blank(10, 4)  # create out
pixel = image.get_pixel(4, 0)  # get original pixel
pixel_out = out.get_pixel(0, 0)  # get blank pixel

# Make pixel values
pixel.red = 255
pixel_out.red = pixel.red  # assuming pixel_out
pixel_out.green = pixel.green
pixel_out.blue = pixel.blue
```

New Pixel Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255`
- `pixel_out.red = pixel.red`  # assuming `pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`
New Pixel Functionality

- \texttt{pixel.red, pixel.blue, pixel.green}
- \texttt{pixel.red = 255}
- \texttt{pixel\_out.red = pixel.red} \# assuming \texttt{pixel\_out}
- \texttt{pixel\_out.green = pixel.green}
- \texttt{pixel\_out.blue = pixel.blue}

# GOAL: Set one pixel to another pixel's value
# set blank pixel to our original pixel's value
\texttt{pixel\_out.red = pixel.red}
\texttt{pixel\_out.green = pixel.green}
\texttt{pixel\_out.blue = pixel.blue}
New Pixel Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255`
- `pixel_out.red = pixel.red` # assuming `pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`

# GOAL: Set one pixel to another pixel's value
# set blank pixel to our original pixel's value

```python
pixel_out.red = pixel.red
pixel_out.green = pixel.green
pixel_out.blue = pixel.blue
```
New Pixel Functionality

- `pixel.red`, `pixel.blue`, `pixel.green`
- `pixel.red = 255`
- `pixel_out.red = pixel.red`  # assuming `pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`

# GOAL: Set one pixel to another pixel's value
# Achieved :)

```python
# GOAL: Set one pixel to another pixel's value
# Achieved :)!
```
Creating an Out Image

New image that's now doubled
New image that now has margins
New image that's now flipped

Editing the Same Image
General Steps: Creating an Out Image

Step 1: Create a new blank image based on original image

Step 2: If margins, loop over new image to create

Step 3: Loop over the original image and find corresponding pixel(s) in the new image

Step 4: Set corresponding new pixel values to old ones

\[(x, y) \rightarrow (x + 1, y)\]
Aqua stripe problem
Step 1: Create a new blank image based on original image

```python
def aqua_stripe(filename):
    image = SimpleImage(filename)
```
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)

Step 2: If margins, loop over and create in the new image
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)

Step 2: If margins, loop over and create in the new image
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            # only some has aqua
            if y in range(4, 8):
                for x in range(10):
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            # only some has aqua
            pixel_out = out.get_pixel(x, y)

Step 2: If margins, loop over and create in the new image
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            # only some has aqua
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            # only some has aqua
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0

Step 2: If margins, loop over and create in the new image

def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            # only some has aqua
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0

Step 2: If margins, loop over and create in the new image
def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):  # only some has aqua
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0

Step 3: Loop over the original image to find corresponding pixel(s) in the new image
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td></td>
</tr>
<tr>
<td>5, 10</td>
<td></td>
</tr>
<tr>
<td>49, 0</td>
<td></td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>10, 0</td>
</tr>
<tr>
<td>5, 10</td>
<td></td>
</tr>
<tr>
<td>49, 0</td>
<td></td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>10, 0</td>
</tr>
<tr>
<td>5, 10</td>
<td></td>
</tr>
<tr>
<td>49, 0</td>
<td></td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>10, 0</td>
</tr>
<tr>
<td>5, 10</td>
<td>15, 10</td>
</tr>
<tr>
<td>49, 0</td>
<td></td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>10, 0</td>
</tr>
<tr>
<td>5, 10</td>
<td>15, 10</td>
</tr>
<tr>
<td>49, 0</td>
<td></td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image.

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>10, 0</td>
</tr>
<tr>
<td>5, 10</td>
<td>15, 10</td>
</tr>
<tr>
<td>49, 0</td>
<td>59, 0</td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

Let's trace the journey of a single pixel!

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>10, 0</td>
</tr>
<tr>
<td>5, 10</td>
<td>15, 10</td>
</tr>
<tr>
<td>49, 0</td>
<td>59, 0</td>
</tr>
<tr>
<td>x, y</td>
<td>x + 10, y</td>
</tr>
</tbody>
</table>
Step 3: Loop over the original image to find corresponding pixel(s) in the new image

def aqua_stripe(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    # creates BLANK image with proper dimensions
    out = SimpleImage.blank(width + 10, height)
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10)
def aqua_stripe(filename):
    ...
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
Step 4: Set corresponding new pixel values to old ones

```python
def aqua_stripe(filename):
    ...
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
```
Step 4: Set corresponding new pixel values to old ones

```python
def aqua_stripe(filename):
    ...
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
```
def aqua_strip(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
def aqua_strip(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
def aqua_strip(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    out = SimpleImage.blank(width + 10, height)

    for y in range(out.height):
        for x in range(10):
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0

    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue

    return out
def aqua_strip(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
def aqua_strip(filename):
    image = SimpleImage(filename)
    width = image.width
    height = image.height
    out = SimpleImage.blank(width + 10, height)
    for y in range(out.height):
        for x in range(10):
            pixel_out = out.get_pixel(x, y)
            pixel_out.red = 0
    for y in range(height):
        for x in range(width):
            pixel = image.get_pixel(x, y)
            pixel_out = out.get_pixel(x + 10, y)
            pixel_out.red = pixel.red
            pixel_out.green = pixel.green
            pixel_out.blue = pixel.blue
    return out
A Quick Note on Image Flipping

Flipping vertically (y)

Flipping horizontally (x)
A Quick Note on Image Flipping

Flipping Horizontally

image
- 0,0
- 0,2
- 1,2
- 2,0
- 3,0
- 4,2

out
- 4,0
- 4,2
- 3,2
- 2,0
- 1,0
- 0,2
A Quick Note on Image Flipping

Flipping Horizontally

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>4,0</td>
</tr>
<tr>
<td>0,2</td>
<td>4,2</td>
</tr>
<tr>
<td>1,2</td>
<td>3,2</td>
</tr>
<tr>
<td>2,0</td>
<td>2,0</td>
</tr>
<tr>
<td>3,0</td>
<td>1,0</td>
</tr>
<tr>
<td>4,2</td>
<td>0,2</td>
</tr>
<tr>
<td>x, y</td>
<td>?,?</td>
</tr>
</tbody>
</table>
A Quick Note on Image Flipping

Flipping Horizontally

<table>
<thead>
<tr>
<th>image</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>4,0</td>
</tr>
<tr>
<td>0,2</td>
<td>4,2</td>
</tr>
<tr>
<td>1,2</td>
<td>3,2</td>
</tr>
<tr>
<td>2,0</td>
<td>2,0</td>
</tr>
<tr>
<td>3,0</td>
<td>1,0</td>
</tr>
<tr>
<td>4,2</td>
<td>0,2</td>
</tr>
<tr>
<td>x, y</td>
<td>?, y</td>
</tr>
</tbody>
</table>
A Quick Note on Image Flipping

Flipping Horizontally

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>K</td>
<td>L</td>
<td>O</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

image

<table>
<thead>
<tr>
<th></th>
<th>0,0</th>
<th>0,2</th>
<th>1,2</th>
<th>2,0</th>
<th>3,0</th>
<th>4,2</th>
<th>x, y</th>
</tr>
</thead>
</table>

out

|   | 4,0 | 4,2 | 3,2 | 2,0 | 1,0 | 0,2 | ?, y |

edge coordinate - x
Flipping Horizontally

image

0,0
0,2
1,2
2,0
3,0
4,2
x, y

out

4,0
4,2
3,2
2,0
1,0
0,2
?, y

effect coordinate - x
(out.width - 1) - x
A Quick Note on Image Flipping

Flipping Horizontally

image

out

image: (0,0)
out: (4,0)
A Quick Note on Image Flipping

Flipping Horizontally

image: (1,0)
out: (3,0)
A Quick Note on Image Flipping

Flipping Horizontally

image: (2,0)
out: (2,0)
A Quick Note on Image Flipping

Flipping Horizontally

image: (3,0)  out: (1,0)
A Quick Note on Image Flipping

Flipping Horizontally

image: (4,0)
out: (0,0)
A Quick Note on Image Flipping

Flipping Horizontally

image: (4,3)
out: (0,3)
Image Functions

- `image = SimpleImage(filename)`
- `out = SimpleImage.blank(width, height)`
- `width = image.width`
- `height = image.height`
- `pixel = image.get_pixel(x, y)`

Pixel Attributes and Functionality

- `pixel.red, pixel.blue, pixel.green`
- `pixel.red = 255 # set pixel to color`
- `pixel_out.red = pixel.red # assuming pixel_out`
- `pixel_out.green = pixel.green`
- `pixel_out.blue = pixel.blue`

Image Problems

- Old Image Filtering
- New Image Creation
PyCharm!
Tour of PyCharm

- Files
- Command Line
- Running the Code in our File
- Adding inputs from terminal
- Bluescreen example
Greenscreen Explanation

An image with a greenscreen + A cool background = An image with a cool background

Photo Credits: TLDump, YouTube and Unsplash
Greenscreen Explanation

An image with a greenscreen

front

+ A cool background

back

= An image with a cool background

front

#if front has certain amount of green (supergreen)
#replace with back(ground) image

Photo Credits: TLDump, YouTube and Unsplash
Greenscreen Explanation

An image with a greenscreen
front

A cool background
back

An image with a cool background
front

#if front pixel is "supergreen"
#replace with back pixel

Photo Credits: TLDump, YouTube and Unsplash
Bluescreen Explanation

An image with a bluescreen
front

+ A cool background
back

= An image with a cool background
front

#if front pixel is "superblue"
#replace with back pixel

Photo Credits: Unsplash
What might "superblue" look like?
Bluescreen Explanation

What might "superblue" look like?

Significantly bluer than average, relative to other colors

pixel average: \( \frac{\text{pixel.red} + \text{pixel.blue} + \text{pixel.green}}{3} \)
Bluescreen Explanation

What might "superblue" look like?

Significantly bluer than average, relative to other colors

The higher our number, the less blue gets replaced (i.e. it is more selective/more "bluey")
Bluescreen Algorithm

Given we have the two filenames of front and back, how might we create a "bluescreen", filtering for pixels that have an abnormally high amount of blue?

Pseudocode of Algorithm

# Front image has special color in back
# Back image is special background

# Loop through front image
    #if front pixel is "superblue"
        #replace with back pixel

# return front image

This is called a Chroma Key!
Let's Code it Up!
(Download zip from website!)
Solution

```python
front = SimpleImage(front_filename)
back = SimpleImage(back_filename)

for y in range(front.height):
    for x in range(front.width):
        pixel = front.get_pixel(x, y)
        # if front images have more than (weighted) average blue
        avg = (pixel.red + pixel.blue + pixel.green)//3

        # lower average threshold = easier to get rid of blue
        if pixel.blue > avg*0.9: # can manipulate weight
            back_pixel = back.get_pixel(x,y)
            pixel.red = back_pixel.red
            pixel.green = back_pixel.green
            pixel.blue = back_pixel.blue # BACK replaces FRONT

return front # front has been modified
```
Recap

Today, we talked about and learned how to...

**Images and Pixels**
- create a new, blank image of custom dimensions
- create an out image from an original image
- copying pixel values over
- code aqua stripe (mirror1, mirror2) examples
- make an out image from original image

**PyCharm**
- venture beyond the experimental server into PyCharm
- how to use the command line
- command line + args
- bluescreen image example
THE END