Images (Part 2)

CS106AP Lecture 9
Life after CS106AP!

Day 1!

Roadmap

Programming Basics

The Console

Images

Graphics

Data structures

Midterm

Object-Oriented Programming

Everyday Python

Life after CS106AP!
Life after CS106AP!

Day 1!

Graphics

Object-Oriented Programming

Midterm

Programming Basics

The Console

Data structures

Images

Everyday Python

Life after CS106AP!
Today’s questions

How can we manipulate images beyond changing color?

What does it mean for code to be “readable”?
Today’s topics

1. Review
2. Advanced image manipulation
3. An exercise in style
4. Nested for loops (more image manipulation)
5. What’s next?
A note about Karel...
In code, there are many ways to solve problems!

- Sometimes, certain solutions may be more “efficient” than others...
  - Developing this intuition will come with lots of practice.
In code, there are many ways to solve problems!

- Sometimes, certain solutions may be more “efficient” than others...
  - Developing this intuition will come with lots of practice.

- But many times, there’s no difference.
  - Creativity in problem-solving is great!
In code, there are many ways to solve problems!

- Sometimes, certain solutions may be more “efficient” than others...
  - Developing this intuition will come with lots of practice.

- But many times, there’s no difference.
  - Creativity in problem-solving is great!

This is why we start the quarter with Karel!
CheckerboardKarel

Ask your neighbors: How did you see the algorithm?
CheckerboardKarel

[demo]
Review
What is an image?

Pixel (1, 0):
red: 6  green: 250  blue: 7
(i.e. shade of green)

Pixel (4, 2):
red: 241  green: 252  blue: 23

Pixel (2, 3):
red: 247  green: 250  blue: 237
SimpleImage module
SimpleImage module

- Import the module

```python
from simpleimage import SimpleImage
```
SimpleImage module

- Import the module

- Create a SimpleImage object and store it in a variable
  - Each SimpleImage object is made up of Pixel objects

```python
image = SimpleImage(filename)
```
SimpleImage module

- Import the module
- Create a SimpleImage object and store it in a variable
- Show the image on your computer

```python
image.show()
```
SimpleImage module

- Import the module
- Create a SimpleImage object and store it in a variable
- Show the image on your computer

- Idea: We manipulate images by editing their pixels!
For each loops
For each loops

```python
for item in collection:
    # Do something with item
```
For each loops

    image = SimpleImage('flower.jpg')

    for pixel in image:
        # Do something with pixel
Summary

- Use a **for each loop** to loop over all pixels in an image

- Edit a pixel by accessing and updating its **properties**:
  - `pixel.x`, `pixel.y` ➔ coordinates (can’t be changed)
  - `pixel.red`, `pixel.green`, `pixel.blue` ➔ RGB values
    - A higher R, G, or B value means a greater amount of that color

- Each `SimpleImage` also has properties:
  - `image.width` ➔ maximum x value
  - `image.height` ➔ maximum y value
Let’s make Photoshop!
What we’ve done so far

- Isolated an RGB channel (red)
What we’ve done so far

- Isolated an RGB channel (red)

- Darkened an image
  - Modified only a particular half/quadrant in an image
What we’ve done so far

- Isolated an RGB channel (red)
- Darkened an image
  - Modified only a particular half/quadrant in an image

How would you write an if statement that only selects pixels in the upper right quadrant of an image?
How would you write an if statement that only selects pixels in the upper right quadrant of an image?

```python
if (pixel.x >= image.width // 2 and pixel.y < image.height // 2):
```
What we’ve done so far

- Isolated an RGB channel (red)
- Darkened an image
  - Modified only a particular half/quadrant in an image
- Grayscaled an image
  - Grayscaled only pixels of a particular color
What we’ve done so far

- Isolated an RGB channel (red)
- Darkened an image
  - Modified only a particular half/quadrant in an image
- Grayscaled an image
  - Grayscaled only pixels of a particular color

*We’ve only manipulated color!*
How can we manipulate images beyond changing color?
Greenscreen algorithm

[demo]
Redscreen

Greenscreen algorithm

[demo]
Greenscreen algorithm

- This is how green-screening in movies works!

```python
for pixel in image:
```

*Loop over all pixels in the image*
Greenscreen algorithm

- This is how green-screening in movies works!

```python
for pixel in image:
    average = (pixel.red + pixel.green + pixel.blue) // 3
```

*Average the RGB values for the pixel*
Greenscreen algorithm

- This is how green-screening in movies works!

```python
for pixel in image:
    average = (pixel.red + pixel.green + pixel.blue) // 3
    if pixel.red >= average * 1.6:
```

*Filter for pixels whose red value is above the average times some “hurdle factor” (i.e. find “red-enough” pixels!)*
Greenscreen algorithm

- This is how green-screening in movies works!

```python
for pixel in image:
    average = (pixel.red + pixel.green + pixel.blue) // 3
    if pixel.red >= average * 1.6:
        # the key line:
        pixel_back = back.get_pixel(pixel.x, pixel.y)
```

Get the corresponding pixel from the “background” image
Greenscreen algorithm

- This is how green-screening in movies works!

```python
for pixel in image:
    average = (pixel.red + pixel.green + pixel.blue) // 3
    if pixel.red >= average * 1.6:
        # the key line:
        pixel_back = back.get_pixel(pixel.x, pixel.y)
        pixel.red = pixel_back.red
        pixel.green = pixel_back.green
        pixel.blue = pixel_back.blue
```

Set the RGB values accordingly to “replace” the pixel!
An exercise in style
(a quick break from images)
An exercise in style
(a quick break from images)

Readability!
Takeaways

● “Readable” code can be more easily understood by anyone
  ○ Through good naming conventions and use of whitespace, it has a narrative/tells a story!

● Variables help us “divide-and-conquer” within a function - breaking steps down into understandable pieces

● Expressions can also be broken down into components to increase readability
  ○ More lines of code does not necessarily mean worse!
How can we manipulate images beyond just altering color?
mirror()
What if we care about the pixels’ x,y coordinates?
Nested for loops

for i in range(end_index_1):
    for j in range(end_index_2):
        # Do something
Nested for loops

```python
for i in range(end_index_1):
    for j in range(end_index_2):
        # Do something
        Repeats \textit{i} \times j \text{ times}!
```
Nested for loops

```python
image = SimpleImage(filename)
for y in range(image.height):
    for x in range(image.width):
        # Do something with pixel at x, y
```
Iterates over all pixels and gives us access to \texttt{x} and \texttt{y}
Nested for loops

```python
image = SimpleImage(filename)
for y in range(image.height):
    for x in range(image.width):
        # Do something with pixel at x,y
```

*Iterate over the rows*
Nested for loops

```
image = SimpleImage(filename)
for y in range(image.height):
    for x in range(image.width):
        # Do something with pixel at x,y
```

*Iterate over the columns*
Nested for loops

```python
image = SimpleImage(filename)
for y in range(image.height):
    for x in range(image.width):
        pixel = image.get_pixel(x, y)
        # Do something with pixel
```

Gets the pixel at x,y
Image coordinate system

width: 100  height: 50

y (height)  x (width)
mirror()
def darker(filename):
    img = SimpleImage(filename)
    for px in img:
        px.red = px.red // 2
        px.green = px.green // 2
        px.blue = px.blue // 2
    return img
What's the difference?

def darker(filename):
    img = SimpleImage(filename)
    for px in img:
        px.red = px.red // 2
        px.green = px.green // 2
        px.blue = px.blue // 2
    return img

def darker(filename):
    img = SimpleImage(filename)
    for y in range(img.height):
        for x in range(img.width):
            px = img.get_pixel(x, y)
            px.red = px.red // 2
            px.green = px.green // 2
            px.blue = px.blue // 2
    return img

Nothing!
def darker(filename):
    img = SimpleImage(filename)
    for px in img:
        px.red = px.red // 2
        px.green = px.green // 2
        px.blue = px.blue // 2
    return img

For `darker()`, we prefer this code.
def darker(filename):
    img = SimpleImage(filename)
    for px in img:
        px.red = px.red // 2
        px.green = px.green // 2
        px.blue = px.blue // 2
    return img

We only want to use nested for loops if we care about \( x \) and \( y \)
Now you try it!

shrink()
flip_horizontal()
Summary

- Use nested for range() loops to manipulate pixels when we care about x,y
  - Use `image.get_pixel(x, y)` to get the pixel at the specific coordinates
Summary

- Use nested for `range()` loops to manipulate pixels when we care about x,y

- Common pattern: Swapping two variables
  - Use a temporary ("temp") variable to store the old value

```
x1 = 3
x2 = 4
temp = x1
x1 = x2
x2 = temp
```
Summary

- Use nested for range() loops to manipulate pixels when we care about x, y
- Common pattern: Swapping two variables

- `SimpleImage.blank(new_width, new_height)` allows us to create a new, empty image of a specific size
  - Then we can loop over its pixels to set their RGB
Summary

- Use nested for range() loops to manipulate pixels when we care about x,y
- Common pattern: Swapping two variables
- `SimpleImage.blank(new_width, new_height)` allows us to create a new, empty image of a specific size
What’s next?
Life after CS106AP!

Day 1!

Roadmap

Programming Basics

The Console

Images

Data structures

Midterm

Graphics

Object-Oriented Programming

Everyday Python

Life after CS106AP!