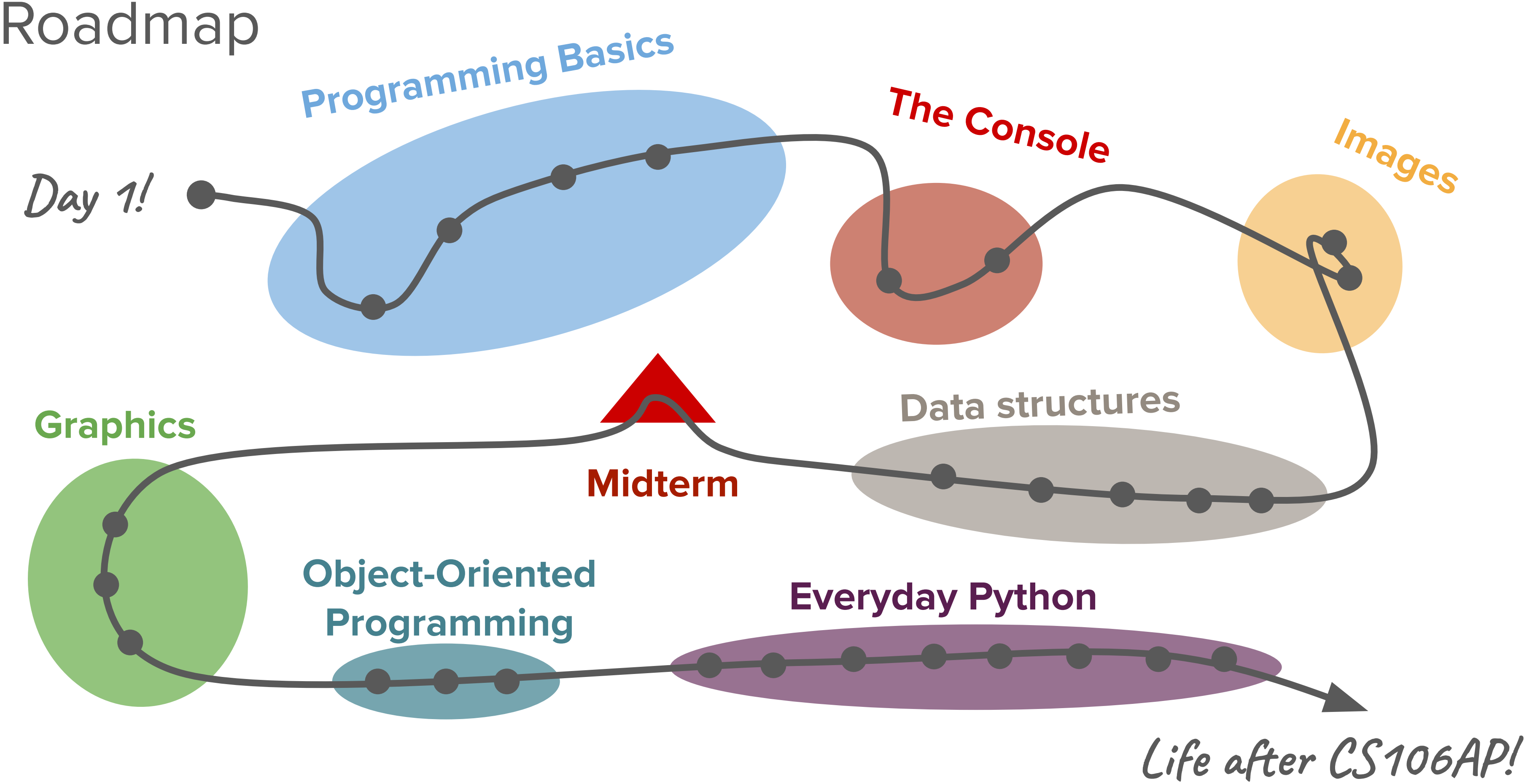
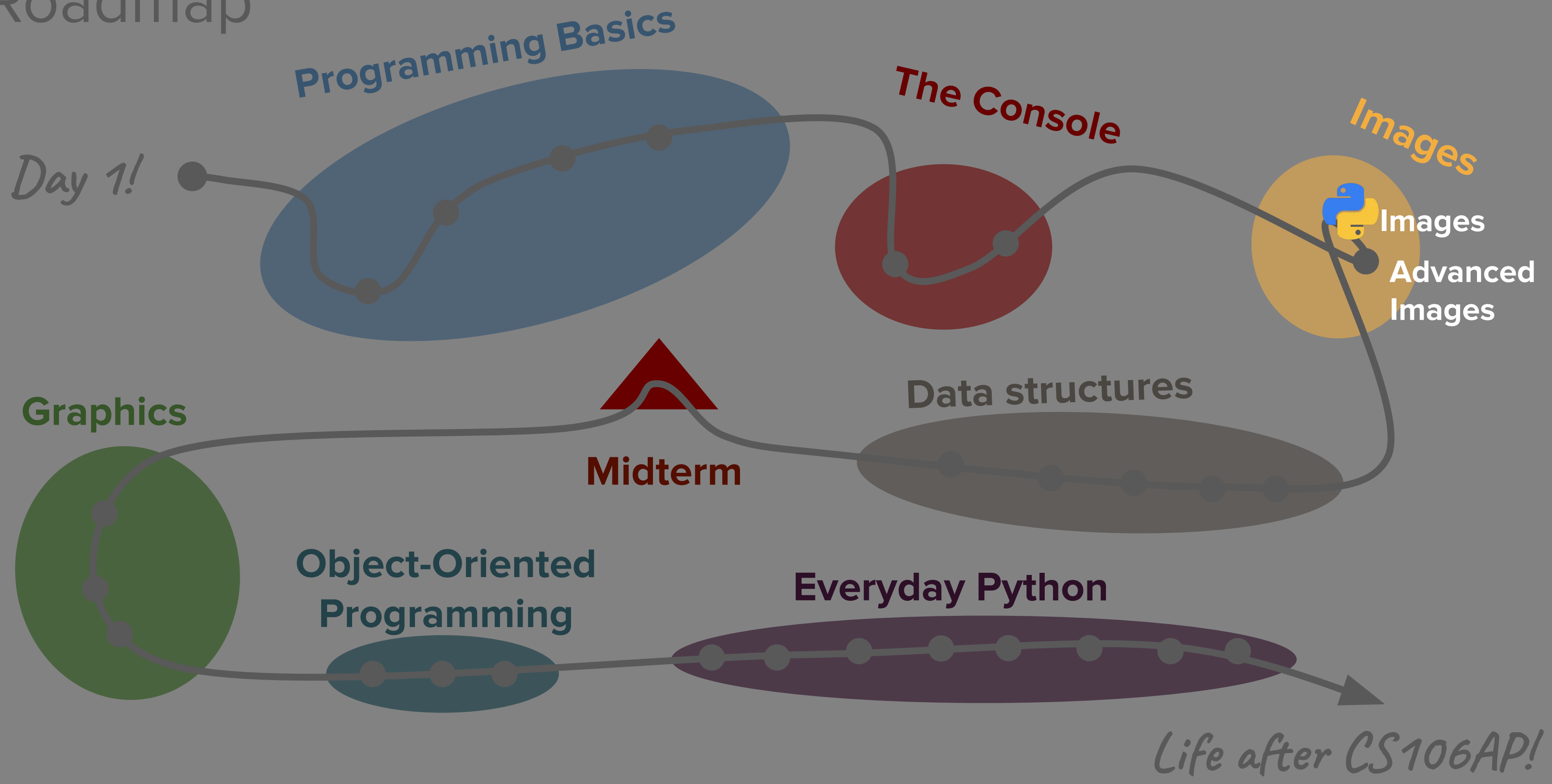
Images (Part 1)

CS106AP Lecture 8

Roadmap


Day 1!
Programming Basics
The Console
Images
Data structures
Midterm
Graphics
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Roadmap



Day 1!

Programming Basics

The Console

Images

Images

Advanced Images

Graphics

Midterm

Data structures

Object-Oriented Programming

Everyday Python

Life after CS106AP!

Today's questions

How do computers store images?

How can we manipulate images through code?

Today's topics

1. Review
2. Introduction to images
3. Another for loop
4. What's next?

Review

Console programs

Definition

Console program

A program that solicits input from a user via an interactive terminal (console) and does something interesting with that input

Console program summary

- Use **`input(prompt)`** to read in information from the user.
  - Make sure to convert the data to the correct type (from the **`string`** data type)!

- Use **`print()`** to display information for the user.
  - Make sure to convert the data to the correct type (from the **`int/float`** data type)

- Use a **`while`** loop to enable multiple runs of your program.

Strings

Definition

string

A data type that represents a sequence of characters

Characters can be letters, digits, symbols (&, !, ~), etc.

Definition

string

A data type that represents a sequence of characters

String fundamentals

- String literals are any string of characters enclosed in single (‘’) or double quotes (“”)

- Each character in the string is associated with an **index**

- Strings can be combined with the + operator in a process called **concatenation**

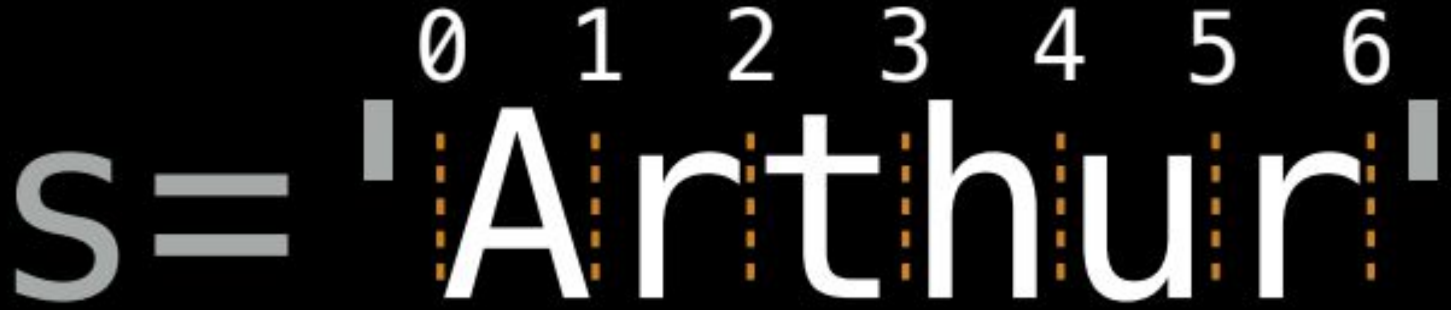
- Strings are **immutable**

Indexing and slicing

- Length
  - The **length** of a string is the number of characters it contains
  - We can use the Python function **`len()`** to evaluate the length of a string

```
len(‘banana’) → 6
len(‘’) → 0
len(‘CS106AP rocks my socks’) → 22
```

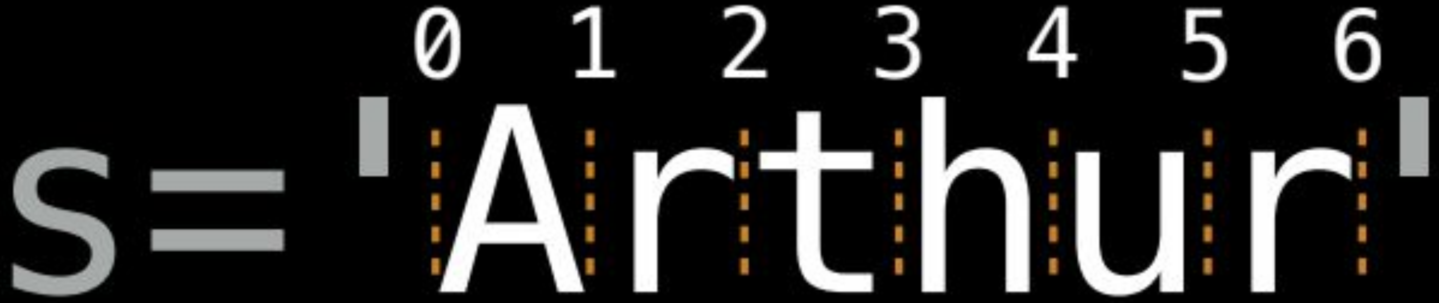
Indexing



```
  0 1 2 3 4 5 6
s='Arthur'
```

Slides courtesy of Sam Redmond, CS41

Indexing

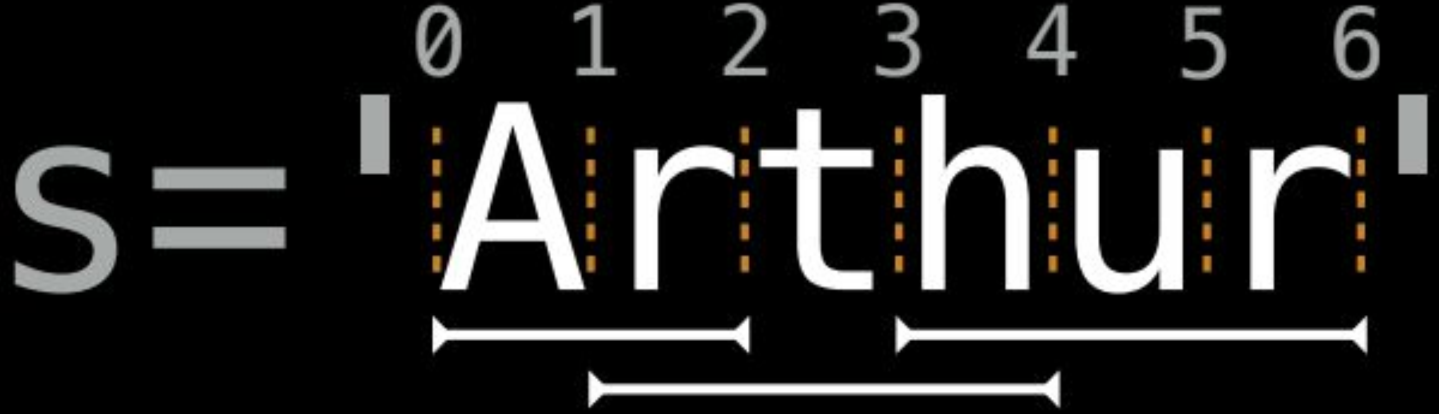


```
 0 1 2 3 4 5 6
s='Arthur'
```

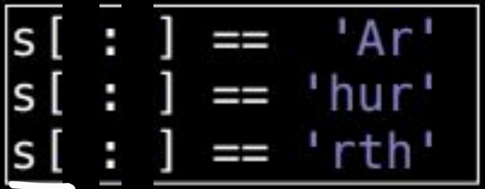
```
s[0] == 'A'
s[1] == 'r'
s[4] == 'u'
s[6] # Bad!
```

Slides courtesy of Sam Redmond, CS41

Slicing



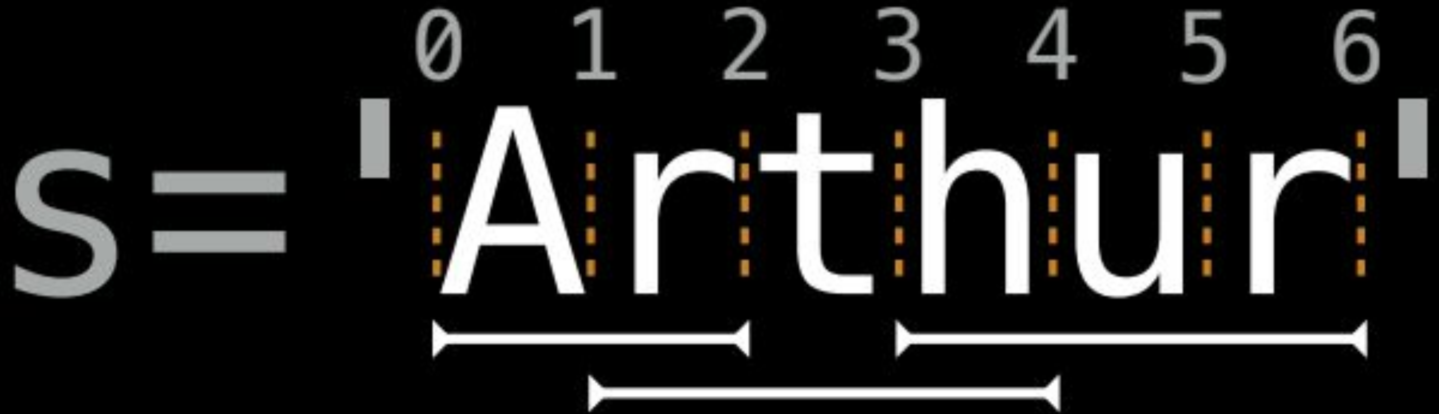
What are the correct indices?



```
s[ : ] ==  'Ar'
s[ : ] == 'hur'
s[ : ] == 'rth'
```

Slides courtesy of Sam Redmond, CS41

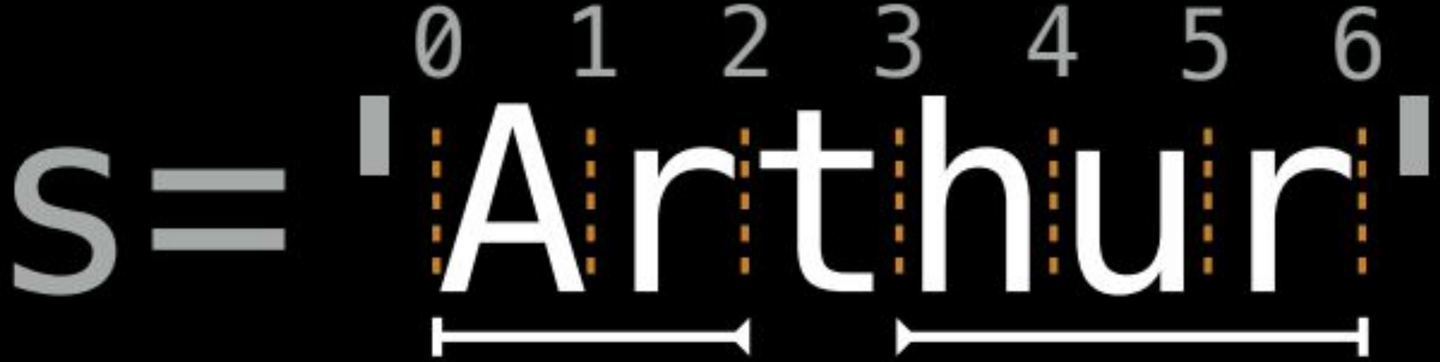
Slicing



```
s[0:2] ==  'Ar'
s[3:6] == 'hur'
s[1:4] == 'rth'
```

Slides courtesy of Sam Redmond, CS41

Strings



Implicitly starts at 0

Implicitly ends at the end

```
s[:2] ==  'Ar'
s[3:] == 'hur'
```

Slides courtesy of Sam Redmond, CS41

String functions

- All follow the **`noun.verb()`** syntax we’ve seen before
- **`str.isupper(), str.islower()`**
- **`str.isalpha(), str.isdigit()`**
- **`str.upper(), str.lower()`**

String functions

- All follow the **`noun.verb()`** syntax we've seen before
- **`str.isupper(), str.islower()`**
- **`str.isalpha(), str.isdigit()`**

Return True or False

- **`str.upper(), str.lower()`**

String functions

- All follow the **`noun.verb()`** syntax we've seen before
- **`str.isupper(), str.islower()`**
- **`str.isalpha(), str.isdigit()`**
- **`str.upper(), str.lower()`** } *Return updated string*

String functions

- All follow the **`noun.verb()`** syntax we've seen before
- **`str.isupper()`, `str.islower()`**
- **`str.isalpha()`, `str.isdigit()`**
- **`str.upper()`, `str.lower()`** } *Return updated string*

Remember: *Original string is unchanged because of immutability!*

Type conversion

- **Important note**: ‘123’ is a **`string`** and 123 is an **`int`**

- In order to convert between data types, we can use built-in Python functions: **`str()`**, **`int()`**, **`float()`**

```
int(‘123’) == 123
float(‘24.7’) == 24.7
str(12345) == ‘12345’
str(20.19) == ‘20.19’
```

Doctests

Doctests

- Python has a great testing framework called **doctests**
  - For each function in your program, write doctests that specify an output for a given input
  - You can (and should) have multiple doctests per function

Doctests

- Python has a great testing framework called **doctests**

- PyCharm supports doctests by allowing you to easily run them in the editor
    - Put doctests in function header comments using `>>>`

Doctests

- Python has a great testing framework called **doctests**

- PyCharm supports doctests by allowing you to easily run them in the editor
    - Put doctests in function header comments using `>>>`

```
def add(a, b):
    “““
    >>> add(2, 4)
    6
    ”””
    ...
```

Doctests

- Python has a great testing framework called **doctests**

- PyCharm supports doctests by allowing you to easily run them in the editor
  - Put doctests in function header comments using `>>>`

```
def add(a, b):
    “““
    >>> add(2, 4)
    6
    ”””
    ...
```

Call the function and specify any arguments if needed.

Doctests

- Python has a great testing framework called **doctests**

- PyCharm supports doctests by allowing you to easily run them in the editor
  - Put doctests in function header comments using `>>>`

```
def add(a, b):
    “““
    >>> add(2, 4)
    6
    ”””
    ...
```

Put the expected output directly after the test.

Testing strategies

- Write tests that cover a wide variety of use cases for your function!

- Consider:
  - Basic use cases
  - Edge cases

Definition

edge case

Uses of your function/program that represent extreme situations

EliminationNation.py

[demo]

Takeaways

- Common pattern: processing all characters in a string

```
for i in range(len(s)):
    current_char = s[i]
    # Use current_char
```

Takeaways

- Common pattern: processing all characters in a string

- Common pattern: building up a new string

```
new_string = ‘’
for i in range(len(s)):
    new_string += s[i]
```

Takeaways

- Common pattern: processing all characters in a string

- Common pattern: building up a new string

```
new_string = ‘’
for i in range(len(s)):
    if __________:
        new_string += s[i]
```

Takeaways

- Common pattern: processing all characters in a string

- Common pattern: building up a new string

```
new_string = ‘’
for i in range(len(s)):
    if ___________:
        new_string += s[i]
```

Select only certain characters - think of this as a filter!

Takeaways

- Common pattern: processing all characters in a string

- Common pattern: building up a new string

- Write doctests for every function!
  - Cover a range of usage patterns for your function
  - Write them before writing the actual function code
  - Run them often as you make changes

Takeaways

- Common pattern: processing all characters in a string

- Common pattern: building up a new string

- Write doctests for every function!
  - Cover a range of usage patterns for your function
  - Write them before writing the actual function code
  - Run them often as you make changes

What is an image?

What is an image?

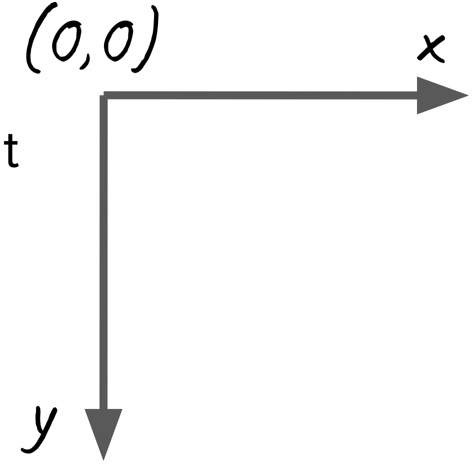
- An image is made up of square **pixels**

[Preview demo]

What is an image?

- An image is made up of square **pixels**

- Each pixel has x and y **coordinates** depending on its location in the image
  - The origin (0, 0) is at the upper left
  - y increases going down, x increases going right



What is an image?

- An image is made up of square **pixels**

- Each pixel has x and y **coordinates** depending on its location in the image

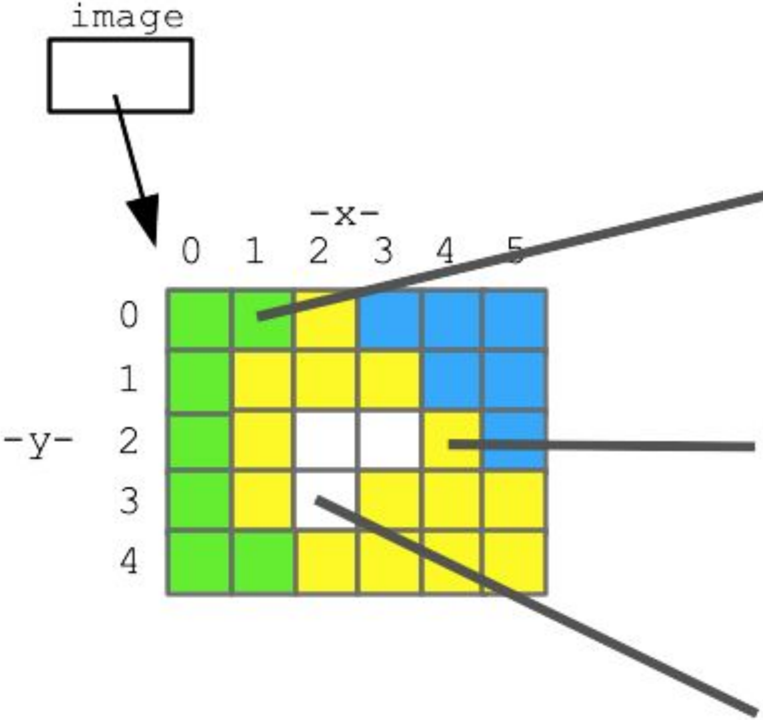
- Each pixel has a single color, encoded as three **RGB** numbers
    - R = red; G = green; B = blue
    - Each value represents a brightness for that color (red, green, or blue)
    - You can use these three colors to make ***any*** color!

What is an image?

- An image is made up of square **pixels**
- Each pixel has x and y **coordinates** depending on its location in the image
- Each pixel has a single color, encoded as three **RGB** numbers

[RGB explorer demo]

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

Image courtesy of Nick Parlante

How can we manipulate images with code?

SimpleImage module

SimpleImage module

- Import the module

```
from simpleimage import SimpleImage
```

SimpleImage module

- Import the module

```
from simpleimage import SimpleImage
```

NOTE: The module may not work for you yet due to some installation requirements!

(We've posted a SimpleImage reference handout on the course website with instructions on how to set up and use the module.)

SimpleImage module

- Import the module

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

```
image = SimpleImage(filename)
```

SimpleImage module

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

```
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!

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!

How do we access pixels?

For each loops

For each loops

A new type of for loop!

Recall: For `range()` loops

```
for i in range(end_index):
   # assumes 0 is the start index
   do_something()

for i in range(start_index, end_index):
   # end_index is not inclusive!
   # recall: range(4,7) -> 4,5,6
   do_something()
```

For each loops

```
for item in collection:
   # Do something with item
```

For each loops

```
image = SimpleImage(‘flower.jpg’)

for pixel in image:

   # Do something with pixel
```

For each loops

```
image = SimpleImage(‘flower.jpg’)

for pixel in image:

    # Do something with pixel
```

Like the **`i`** *in for range() loops,* **`pixel`** *is a variable that gets updated with each loop iteration*

For each loops

```
image = SimpleImage(‘flower.jpg’)

for pixel in image:

    # Do something with pixel
```

`pixel` *gets assigned to each pixel object in the image in turn*

For each loops

```
image = SimpleImage(‘flower.jpg’)

for pixel in image:

    # Do something with pixel
```

This code gets repeated once
for each pixel in the image

Let’s make Photoshop!

[demo]

Summary

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

Summary

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

- Edit a pixel by updating its **properties:**
    - **`pixel.x`**, **`pixel.y`** ➔ coordinates
    - **`pixel.red`**, **`pixel.green`**, **`pixel.blue`** ➔ RGB values
        - A higher R, G, or B value means a greater amount of that color

Summary

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

- Edit a pixel by updating its **properties:**
  - **`pixel.x`**, **`pixel.y`** → coordinates
  - **`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

Summary

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

- Edit a pixel by updating its **properties:**
    - **`pixel.x`**, **`pixel.y`** → coordinates
    - **`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

Think/Pair/Share

How would you darken only the bottom right quadrant?

Grayscale algorithm

[demo]

Grayscale algorithm

- You get the color “gray” in a pixel when its red, green, and blue values are all the same.

- To keep grayscale an image, average the red, green, and blue values for a given pixel and re-assign each RGB value in that pixel to the average.

curb_repair()

[demo]

Greenscreen algorithm

[demo]

Greenscreen (or bluescreen) algorithm

- This is how green-screening in movies works!

```
for pixel in image:
```

Loop over all pixels in the image

Greenscreen (or bluescreen) algorithm

- This is how green-screening in movies works!

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

Average the RGB values for the pixel

Greenscreen (or bluescreen) algorithm

- This is how green-screening in movies works!

```
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 (or bluescreen) algorithm

- This is how green-screening in movies works!

```
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 (or bluescreen) algorithm

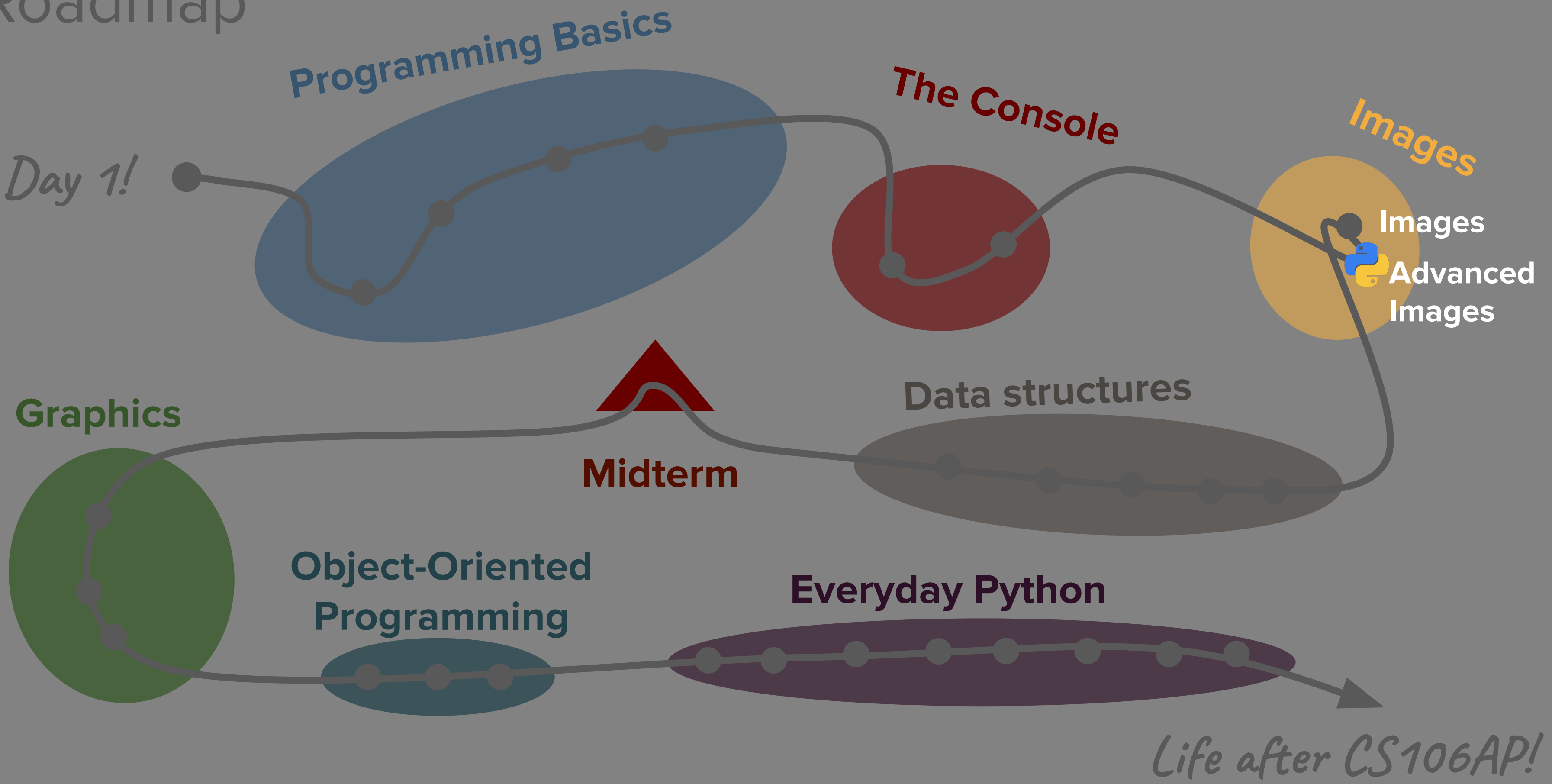
- This is how green-screening in movies works!

```
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!

What’s next?

Roadmap


Day 1!
Programming Basics
The Console
Images
Images
Advanced Images
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Advanced images

- More Photoshop functionality
- Practice with image coordinates
- More complex control flow
- Assignment 2 bluescreen contest

HOMEWORK: Read the SimpleImage reference and install everything!

Advanced images

- More Photoshop functionality
- Practice with image coordinates
- More complex control flow
- Assignment 2 bluescreen contest

This could be YOU!