Week 4 Lecture Recap

Lectures covered this week:

1. [Images (Friday May 7)](#_hmnhl2878x3o)
2. [Lists](#_m8aola4yoowl) (Wednesday May 12)

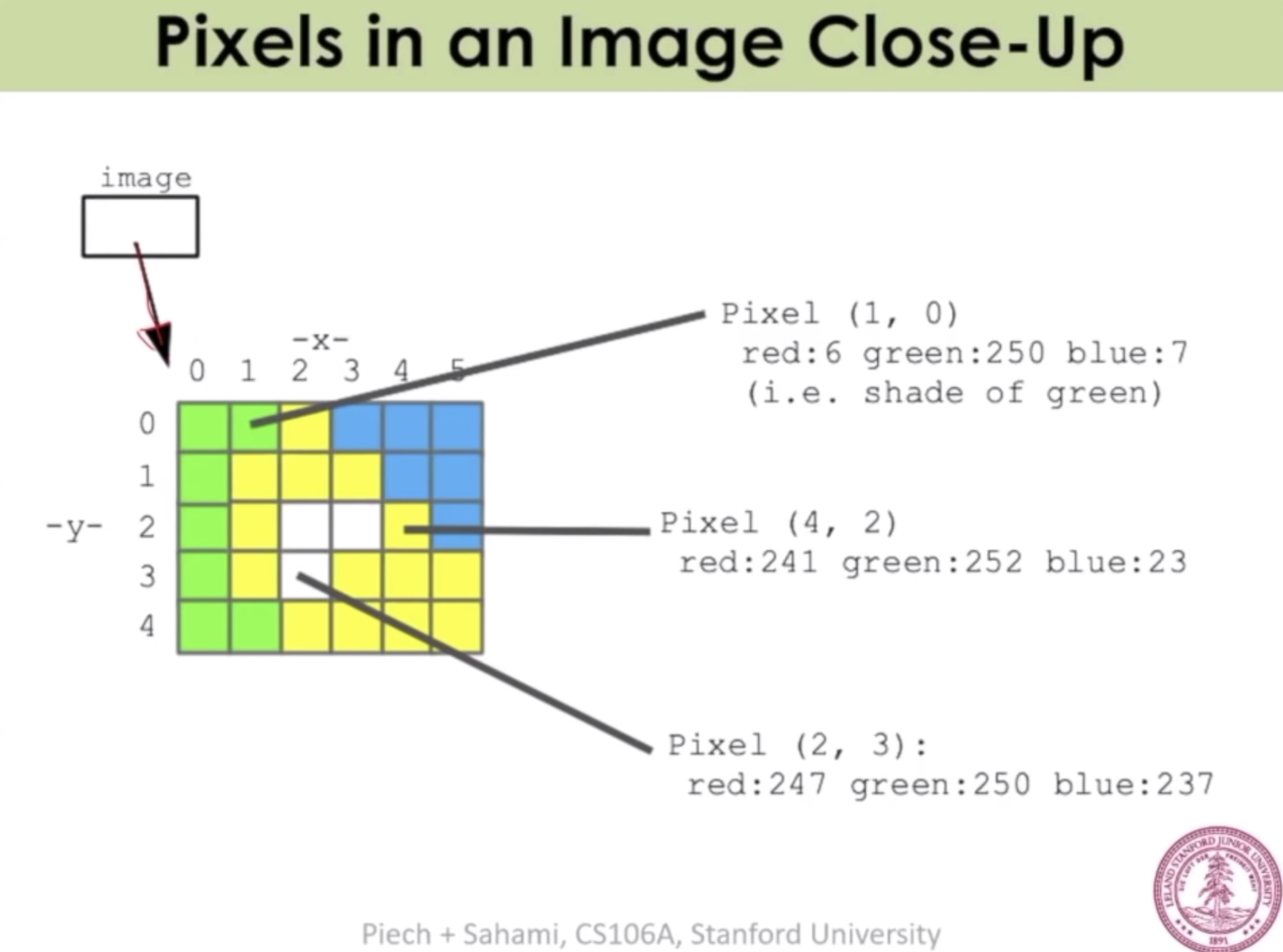
(Because of the diagnostic on Monday, there are only 2 lectures covered)

# Images

Covered in this lecture:

1. [What is an image?](#_impp3rv6tfa1)
2. [SimpleImage Library](#_2gcj4yk670j)
3. [For-Each Loop](#_v42723ppb2j9)
4. [Examples](#_89cp9zev3a07)
5. [Nested Loop vs For-Each Loop](#_9444yxcnewv8)

|  |  |
| --- | --- |
| What is an image?  * Image is made of square pixels   + Example: flower.png * Each pixel has x and y coordinates in the image   + The origin (0,0) is at the upper-left corner   + y increases going down, x increases going right * Each pixel has single color encoded as 3 RGB values   + R = red; G = green; B = blue   + Each value represents brightness for that color (red, green, or blue)   + Can set RGB values to make any color! |  |



## SimpleImage Library

Using SimpleImage Library

* In folders for assignment or lecture on images, there is a file **simpleimage.py** 
  + This is the SimpleImage library
* To use the SimpleImage library in your code, include at the top of your program file:

**from simpleimage import SimpleImage**

* This is importing the SimpleImage module, so that it is accessible in the code you write
  + Similar to when you used import random to use random number generator library

Functions in SimpleImage Library

* Create a SimpleImage object by reading an image from file (jpg, png, gif, etc.) and store it in a variable
  + Note: each SimpleImage object is made up of Pixel objects

**my\_image = SimpleImage(filename)**

* Show the image on your computer.

**my\_image.show()**

Properties of Images and Pixels

* Each SimpleImage image has properties you can access:
  + Can get the width and height of image

image.width, image.height

* Each pixel in an image also has properties:
  + Can get x, y coordinates of a pixel in an image

pixel.x, pixel.y

* + Can get RGB values of a pixel
    - These are just integers between 0 and 255

pixel,red, pixel.green, pixel.blue

* + - Higher R, G, or B values means more of that color in pixel

## For-Each Loop

Accessing Pixels in an Image

* We can use a new kind of loop called a "for-each" loop
* For-each loop:

for item in collection:

# Do something with item

* For-each loop with image

image = SimpleImage("flower.jpg")

for pixel in image:

# The code here gets repeated once for each pixel in image

## Examples

A darker Image:

|  |
| --- |
| def darker(filename):  """  Reads image from file specified by filename.  Makes image darker by halving red, green, blue values  Returns the darker version of image.  """  image = SimpleImage(filename)  for px in image:  px.red = px.red // 2  px.green = px.green // 2  px.blue = px.blue // 2  return image |

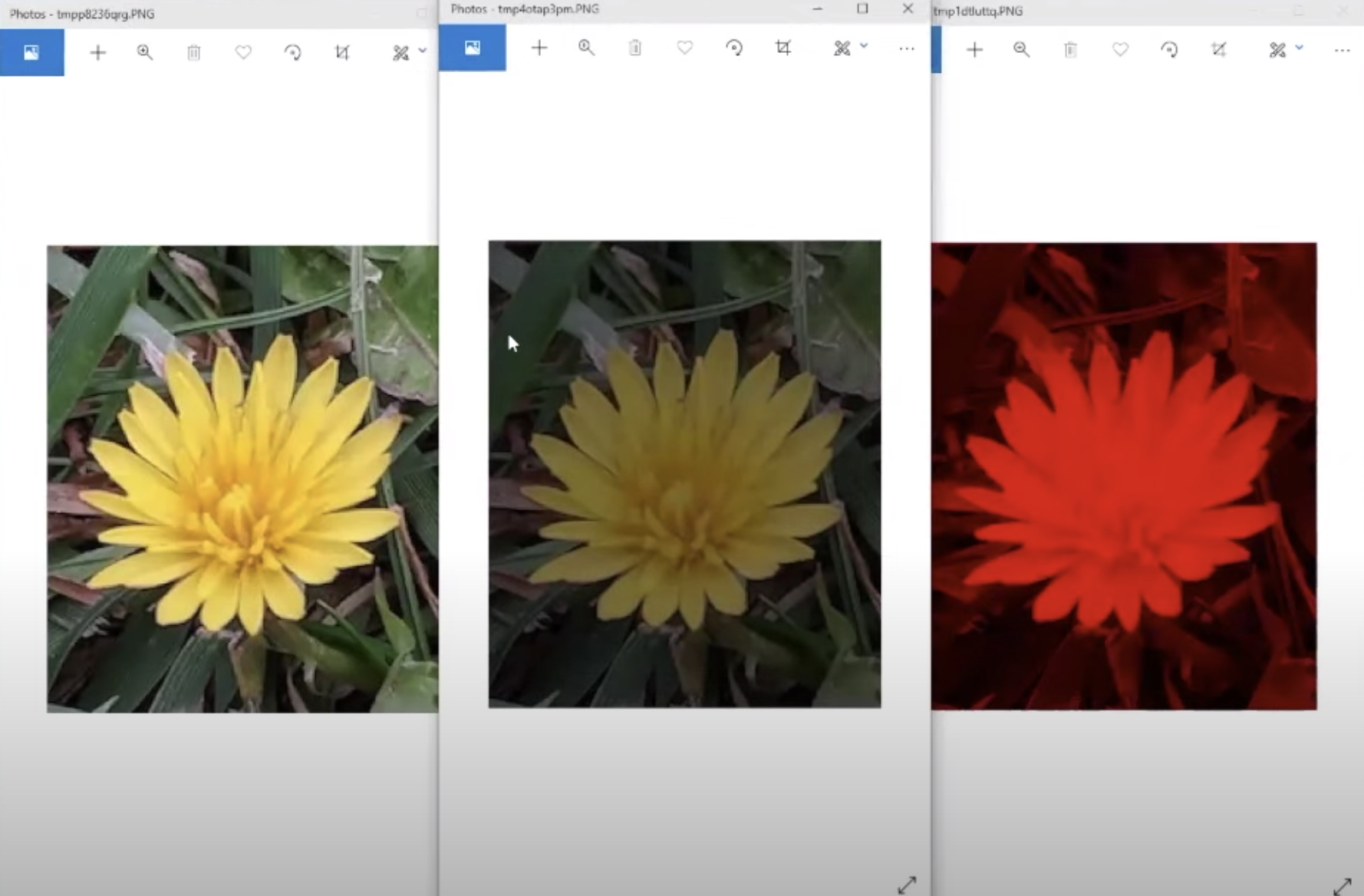
Get Red Channel

|  |
| --- |
| def red\_channel(filename):  """  Reads image from file specified by filename.  Changes the image as follows:  For every pixel, set green and blue values to 0,  yielding the red channel.  Return the changed image.  """  image = SimpleImage(filename)  for px in image:  px.green = 0  px.blue = 0  return image |

Let's take these functions for a spin!

|  |
| --- |
| def main():  """  Run your desired image manipulation functions here.  You should store the return value (image) and then  call .show() to visualize the output of your program.  """    # The name of the image file is called flower.png and it is in the  # images folder, so to access the file we do: 'images/flower.png'    original\_flower = SimpleImage('images/flower.png')  original\_flower.show() # displays the original image  dark\_flower = darker('images/flower.png')  dark\_flower.show()  red\_flower = red\_channel('images/flower.png')  red\_flower.show() |

Our result:



## Nested Loop vs For-Each Loop

What's the difference?

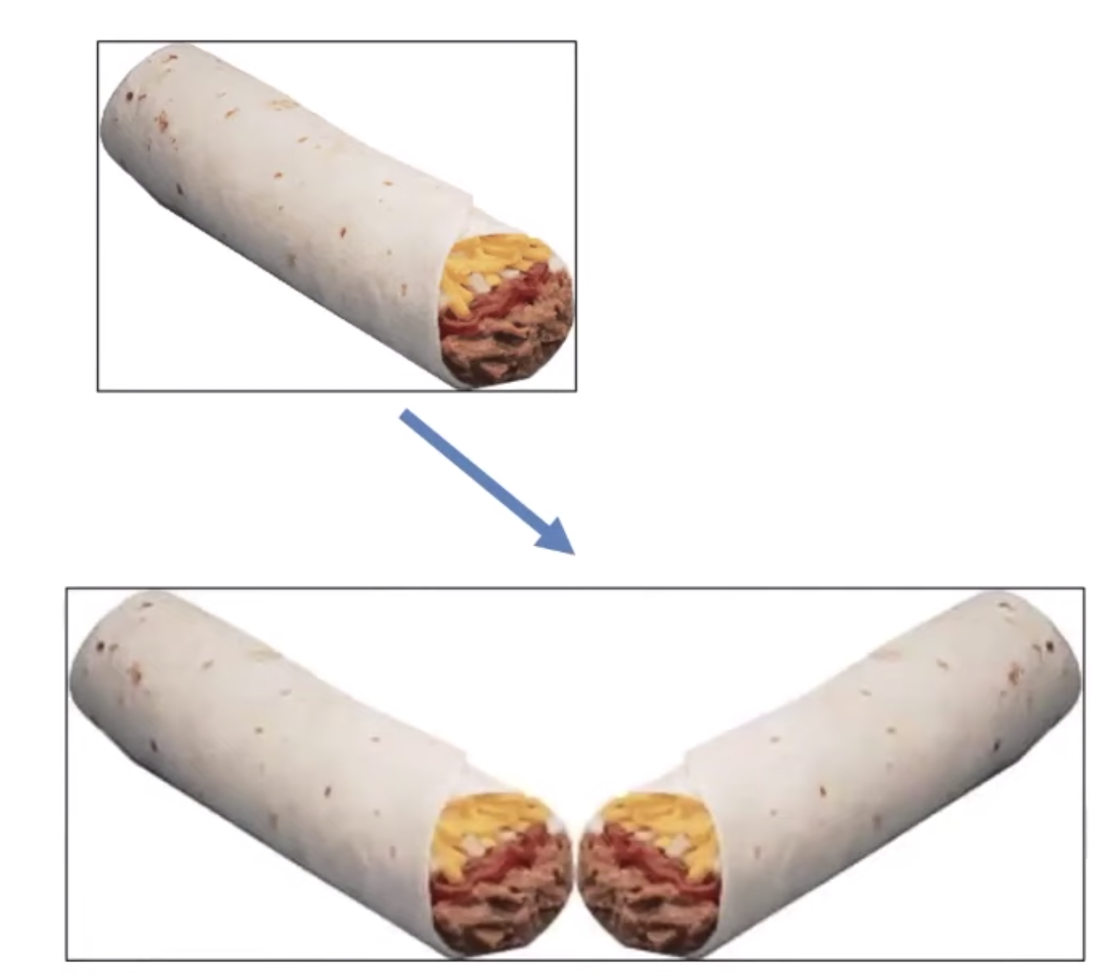
|  |  |
| --- | --- |
| #For-each loop  def darker(filename):  image = SimpleImage(filename)  for px in image:  px.red = px.red // 2  px.green = px.green // 2  px.blue = px.blue // 2  return image | #Nested for-loop  def darker(filename):  image = SimpleImage(filename)  for y in range(image.height):  for x in range(image.width):  px = image.get\_pixel(x, y)  px.red = px.red // 2  px.green = px.green // 2  px.blue = px.blue // 2  return image |

Nothing! Both functions above do the exact same thing: make an image darker.

We only want to use nested for loops if we care about **x** and **y**. (we need this for mirroring images)

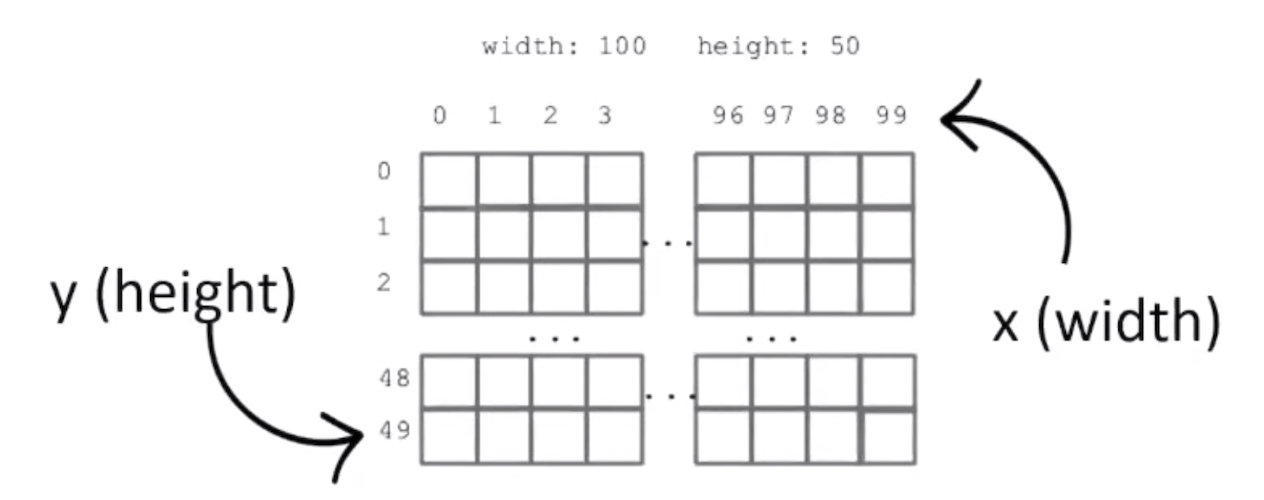
Mirroring an Image

For this program, we are given an image, and we want to return an image that includes the original image and its mirror reflection, like this:



How can we write a program that gives us the mirror image?

We will need access to the **x** and **y** values of the pixels in the original image. In our output image, we want to set the corresponding **x** and **y** coordinates of the original image and the mirror image to be the same color.



We can use a nested for-loop to do this!

Here's how:

|  |
| --- |
| def mirror\_image(filename):  """  Reads image from file specified by filename.  Returns a new image that includes the original image  and its mirror reflection.  """    image = SimpleImage(filename)  width = image.width  height = image.height  # Create new image to contain mirror reflection  mirror = SimpleImage.blank(width \* 2, height)  for y in range(height):  for x in range(width):  pixel = image.get\_pixel(x,y)  mirror.set\_pixel(x, y, pixel)  mirror.set\_pixel((width \* 2) - (x+1), y, pixel)  return mirror |

# Lists

What is a list?

* A **list** is way to keep track of an *ordered* *collection* of items
  + Items in the list are called "elements"
  + Ordered: can refer to elements by their position
  + Collection: list can contain multiple items
* The list dynamically adjusts its size as elements are added or removed

Creating Lists

* Lists start/end with brackets. Elements are separated by commas.

**my\_list = [1, 2, 3]**

**reals = [4.7, -6.0, 0.22, 1.6]**

**strs = ['lots', 'of', 'strings', 'in', 'list']**

**mix = [4, 'hello', -3.2, True, 6]**

**empty\_list = []**

* \*Note: A list with one element is not the same as the element!

Accessing Elements of a List

* Consider the following list:

**letters = ['a', 'b', 'c', 'd', 'e']**

* Can think of it like a series of variables that are indexed
  + Indexes start from 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **'a'** | **'b'** | **'c'** | **'d'** | **'e'** |
| 0 | 1 | 2 | 3 | 4 |

* Access individual elements:

**letters[0]** is **'a'**

**letters[4]** is **'e'**

* Can set individual elements like regular variable:

**letters[0] = 'x'**

* Can get the length of a list with the **len** function:

**len(letters)** is 5

* Can use negative index to work back from end of list

**letters[-1]** is **'e'**

**letters[-2]** is **'d'**

**letters[-5]** is **'a'**

* + For indexes, think of **–x** as same as **len(list)–x**

**letters[-1]** is same as **letters[len(letters)-1]**

* How about this?

**letters[6]**

**IndexError: list index out of range**

What can you do with lists?

* Add elements to end of list with **.append**
* Remove elements from end of list with **.pop**

**num\_list = [1, 2, 3, 4]**

**str\_list = ['Leia', 'Luke', 'Han']**

* Printing lists (here, we show using the console):

**>>> print(num\_list)**

**[1, 2, 3, 4]**

**>>> print(str\_list)**

**['Leia', 'Luke', 'Han']**

* Check to see if list is empty (empty list is like "False")

**if num\_list:**

**print('num\_list is not empty')**

**else:**

**print('num\_list is empty')**

* Check to see if a list contains an element:

**x = 1**

**if x in num\_list:**

**# do something**

* General form of test (evaluates to a Boolean):

***element***  **in** ***list***

* + Returns **True** if ***element*** is a value in ***list***, **False** otherwise
  + Could use as test in a **while** loop too
* Function: ***list*.extend(other\_list)**
  + Adds all element from other list to list that function is called on

**>>> list1 = [1, 2, 3]**

**>>> list2 = [4, 5]**

**>>> list1.extend(list2)**

**>>> list1**

**[1, 2, 3, 4, 5]**

* **append** is not the same as **extend**
  + Append adds a single element, extends merges a list onto another

**>>> list1 = [1, 2, 3]**

**>>> list2 = [4, 5]**

**>>> list1.append(list2)**

**>>> list1**

**[1, 2, 3, [4, 5]]**

* Using **+** operator on lists works like **extend** , but creates a new list. Original lists are unchanged.
* Can use **+=** operator just like **extend**
* Function: ***list*.index(elem)**
  + Returns index of first element in list that matches parameter elem

**>>> alist = ['a', 'b', 'b', 'c']**

**>>> i = alist.index('b')**

**>>> i**

**1**

* + ValueError if you ask for index of an element that isn't in list
* Function: ***list*.insert(index, elem)**
  + Inserts elem at the given index. Shifts all other elements down.

**>>> jedi = ['luke', 'obiwan']**

**>>> jedi.insert(1, 'mehran')**

**>>> jedi**

**['luke', 'mehran', 'obiwan']**

* Function: ***list*.copy()**
  + Returns a copy of the list
* Function: **max(list)**
  + Returns maximal value in the list

**>>> max(reals)**

**8.0**

* Function: **min(list)**
  + Returns minimal value in the list

**>>> min(reals)**

**-3.2**

* Function: **sum(list)**
  + Returns sum of the values in the list

**>>> sum(reals)**

**11.8**

Looping Through List Elements

* General form of for-each loop:

**for *element* in *collection*:**

**# do something with *element***

* ***element*** can be any variable you want to use to refer to items in the ***collection***
  + On each iteration through the loop, ***element*** will be set to be the next item (in order) in the ***collection***
  + Recall, example:

**for elem in str\_list:**

**print(elem)**

* + Lists are collections
  + Images are also collections (of pixels)

When Passed as Parameters

