You Know More Than You Think… ;)

Digital Images in Python
While Loops

CS 8: Introduction to Computer Science, Winter 2018
Lecture #13

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- Homework #7 is due **ON MONDAY 3/12**
- Lab #5 due **ON FRIDAY 3/9 (EXTENDED)**
- Remaining on the calendar… ***This supersedes anything on the syllabus***

DATE	TOPIC	ASSIGNED	DUE
Mon. 3/5	File I/O ; Formats for Outputs	Hw #7 Lab #5	Hw #6 Lab #5
Wed. 3/7	Digital Images ; While-Loops		
Mon. 3/12	Recursive Functions	Hw #8 Lab #6	Hw #7, Hw #8 Lab #6 Proj #2
Wed. 3/14	Review for the Final Exam		

Lecture Outline

Chapter 6

- Digital Images on Computers
- Indexed Color Schemes
- The **cImage** Module

- While-Loops

Starting Chapter 6

Digital Images on Computers

- Two types of images: **raster** vs. **vector**

- **Raster** (a.k.a "bit-map") images
  - Most picture formats from photos, paint/shop programs
  - Typically **JPEG** (**.jpg**, **.jpeg**) types
  - Made of a finite number of **pixels** (or **dots**)
    - Quality of picture is measured in **dots per inch** (**dpi**)
    - Close-ups look blurry or "pixelated"
  - The higher the resolution, the more pixels are needed
    - More pixels mean larger file sizes to store the image
  - Raster images are a great choice for photographic pictures

JPEG Example with Different Quality Settings

Digital Images on Computers

Vector (a.k.a "object-based") images

- Most picture formats that come from drawing programs
- Typically **SVG** (**.svg**) types
- **Not** pixel representation – uses mathematical formulae to represent shapes
  - Close-ups or pull-backs look smooth and clean
- Resolution is always good
  - File size is constant (usually small)
- Great for **logos**, **simple representations** of real objects
- Isn't very good for exact photographic representations

Examples of Raster vs Vector

Raster (bit-map)

Vector

Same Examples (zoomed in)

Raster (bit-map)

Shows “pixilation”

Vector

Shows perfect reproduction

Indexed Colors in Images

- Colors on a monitor are represented by the **RGB scheme**
  - **256** variations on **each of** Red, Green, and Blue palates
  - Mixing gives a full palate of colors (per projected, not reflected light)
  - Giving you a combination of over **16 million colors**

- Are there **more** than 16 million colors in the real world?

Indexed Colors in Images

Q: Are there more than 16 million colors in the real world?

A: Yes! (well, probably, not that *I* can tell… :\)

A fixed scheme, like RGB, is necessary because:

1. **It puts an upper limit**
   (on colors, on file sizes, on time to render pictures onto a screen, etc…)

2. **It accommodates display technologies**
   (they're really advanced, but they're not limitless in their capabilities!)

3. It is **good enough** for 99.99% of computer (esp. Web) users!

The RGB Scale

- 256 settings for Red ➔ 8 bits (why?)
- 256 settings for Green ➔ 8 bits
- 256 settings for Blue ➔ 8 bits

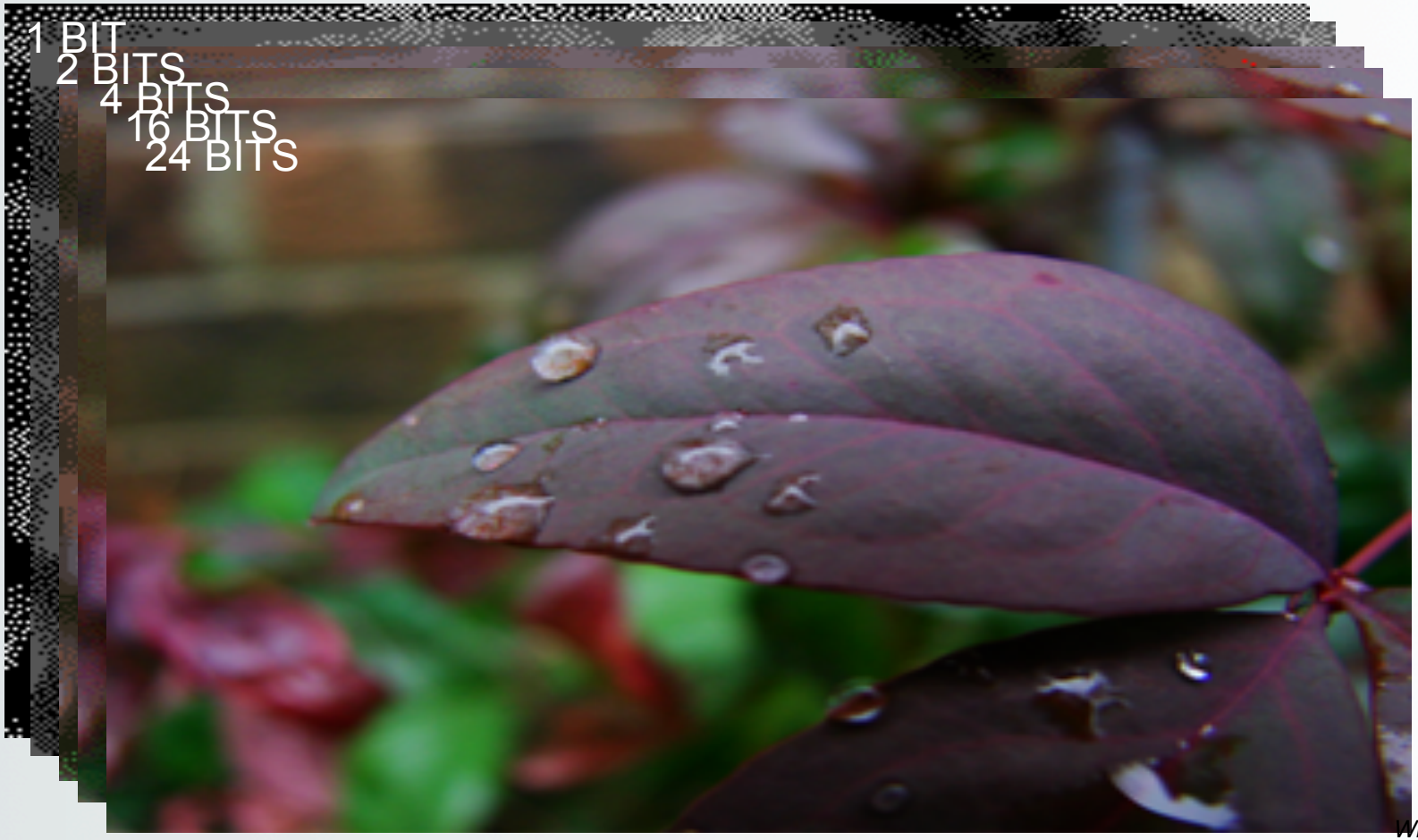
- 1 bit = 2 combinations (0 or 1)
- 2 bits = 4 combinations (00, 01, 10, or 11)
- N bits = 2^N combinations

- RGB has **24 bits** (8 for each R,G,B) to use to define a "color"
  - 2^{24} is approximately 16 million…

Colors	RGB intensities		
	RED	GREEN	BLUE
BLACK	0	0	0
RED	255	0	0
GREEN	0	255	0
BLUE	0	0	255
YELLOW	255	255	0
MAGENTA	255	0	255
CYAN	0	255	255
WHITE	255	255	255

The number of bits used to describe a color pallet

exponentially raises the number of colors used in a computer graphic



Wikipedia.com

Image Processing with the `cImage` Module

- Textbook's `cImage` module processes **raster** data
- Designed to work with `.gif` and `.ppm` formats only
  - Can install a library for `.jpg` format, but not available in lab
- Chapter 6 uses objects of the module's `Pixel`, `FileImage`, `EmptyImage` and `ImageWin` classes

Using cImage

- Import cImage like this:

```
from cImage import *
```

  - This allows you to use cImage methods/functions without having to say "cImage." first
  - Example:
    Instead of:
    `im = cImage.FileImage('x.jpg')`, you could just say:
    `im = FileImage('x.jpg')`

Construct a Window

- To construct a window, use this:

```
title = "My Picture"
width = 300   # units is pixels
height = 300  # units is pixels
myWin = ImageWin(title, width, height)
```

Method Name	Example Use	Explanation
`FileImage(filename)`	`im = FileImage('pic.gif')`	Create an image object from a file named pic.gif.
`getWidth()`	`w = im.getWidth()`	Return the width of the image in pixels.
`getHeight()`	`h = im.getHeight()`	Return the height of the image in pixels.
`getPixel(col, row)`	`p = im.getPixel(150,100)`	Return the `Pixel` from row 100, column 150.
`setPixel(col, row, newp)`	`im.setPixel(150, 100, Pixel(255, 255, 255))`	Set the pixel at row 100, column 150 to be white.
`setPosition(col, row)`	`im.setPosition(20, 20)`	Position the top-left corner of the image at (col, row) in the window.
`draw(imagewin)`	`im.draw(myWin)`	Draw the image `im` in the `myWin` image window. It will default to the upper-left corner.
`save(fileName)`	`im.save(fileName)`	Save the image to a file. Use `gif` or `ppm` as the extension.

Method Name	Example Use	Explanation
`Pixel(r, g, b)`	`p = Pixel(25, 200, 143)`	Create a pixel with 25 red, 200 green, and 143 blue.
`getRed()`	`r = p.getRed()`	Return the red component intensity.
`getGreen()`	`g = p.getGreen()`	Return the green component intensity.
`getBlue()`	`g = p.getBlue()`	Return the blue component intensity.
`setRed()`	`p.setRed(100)`	Set the red component intensity to 100.
`setGreen()`	`p.setGreen(45)`	Set the green component intensity to 45.
`setBlue()`	`p.setBlue(87)`	Set the blue component intensity to 87.

A `Pixel` class

- A way to manage the color of *one pixel*
- A `color` = *amounts* of (`red`, `green`, `blue`)
  - When coded by the RGB color model
  - Range of each part: 0 to 255

```
whitePixel = cImage.Pixel(255,255,255)
blackPixel = cImage.Pixel(0,0,0)
purplePixel = cImage.Pixel(255,0,255)
yellowPixel = cImage.Pixel(255,255,0)
```

The "mixes" don't always work like, say, mixing paints do

- Methods: `getRed()`, `setBlue(value)`, …some others…

Image Classes in `cImage`: `EmptyImage` and `FileImage`

- Create a new (empty) image with dimensions:
  - **Create new**: `img = EmptyImage(cols, rows)`
- Use an existing image to get
  - Or **use existing**: `img = FileImage(filename)  # Careful of where the file is`
- How to manage a set of pixels, organized by rows and columns
  - `x` denotes the column – *leftmost* `x` is 0
  - `y` denotes the row – *topmost* `y` is 0
- Methods:

  `getWidth(), getHeight(), getPixel(x, y),`
  `setPixel(x, y, pixel),  save(filename),`
  … and `draw(window)`

`ImageWin` class

- A window frame that displays itself on-screen

```
window = cImage.ImageWin(title, width, height)
image.draw(window)
```

- Mostly just used to hold (new or existing) images, but also has some methods of its own
  - e.g., `getMouse()` – returns `(x,y)` tuple where mouse is clicked (in window, not necessarily same as image)
  - `exitOnClick()` – closes window and exits program on mouse click

Demo!

```
from cImage import *
im = FileImage('./leo.gif')   # load an existing image
title = "My Friend Leo"
width = 600          # units is pixels
height = 600         # units is pixels
myWin = ImageWin(title, width, height) # Define myWin
im.draw(myWin)       # Draws the image in myWin

im.getWidth()        # Report on the height of the existing image
im.getHeight()       # Report on the width of the existing image

whitePix = Pixel(255,255,255)
im.setPixel(150, 100, Pixel(255,255,255))
for x in range(500):
     im.setPixel(x, 100, whitePix)
     im.setPixel(150, x, whitePix)
```

ImageDemo.py

Negative Images & Grayscale

- Negative images – "flip" each pixel color

```
for row in range(height):
   for col in range(width):
      # get r, g, b from old image here
      negPixel = Pixel(255 - r, 255 - g, 255 - b)
      newImage.setPixel(col, row, negPixel)
```

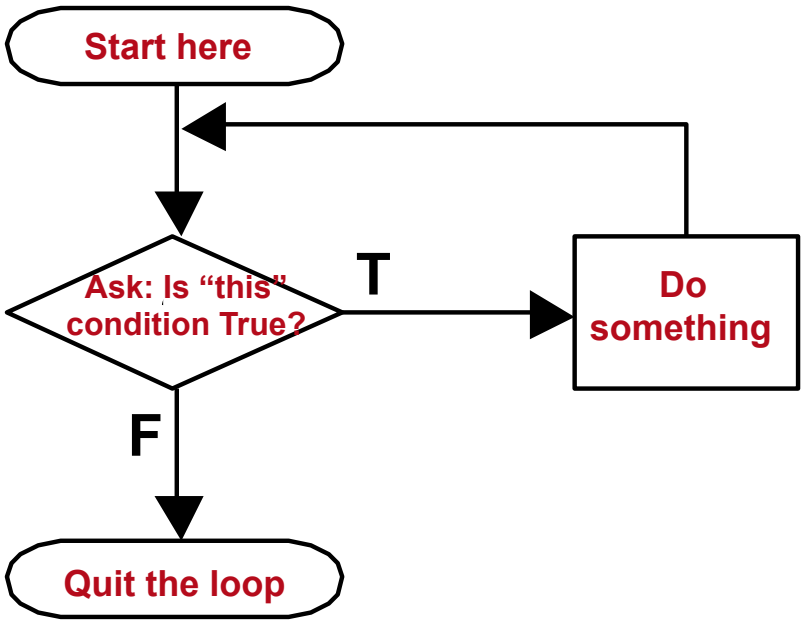
  - **Listings 6.1 and 6.2 in textbook – negimage.py**

- Grayscale similar (Listings 6.3 and 6.4):

```
# ... as above through get r, g, b
      avg = (r + g + b) // 3
      grayPixel = Pixel(avg,avg,avg)
```

  - **Listings 6.3 and 6.4 – grayimage.py**

Flow of an Iteration Structure



EXAMPLE:

```
for x in range(1, 10):
    print (x)
```

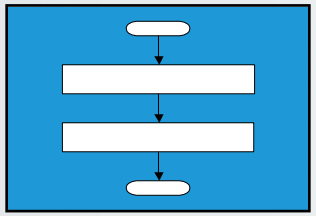
Ask: is 1 <= x < 10?

If True, the do the following:
print x, then make x = x + 1

If False, then quit the loop

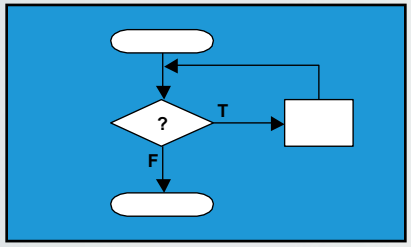
Review: 3 Control Structure Types

Sequence



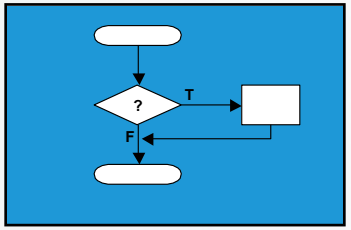
Iteration

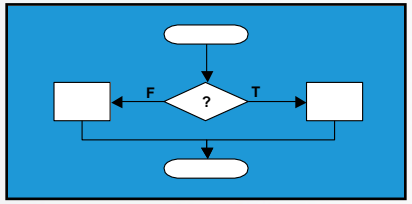
Loops



Selection

If-else statements





Repetition with a `while` loop

- `while` *condition*`:`

  # executes over and over until a condition is ***False***

- Used for indefinite iteration
  - When it isn't possible to predict how many times a loop needs to execute, unlike with for loops

- We use `for` loops for definite iteration
  (e.g., the loop executes exactly **n** times)

Applying `while`

- Can be used for counter-controlled loops:

```
n = 500
counter = 0                     # (1) initialize
while counter < n:              # (2) check condition
    print(counter * counter)
    counter = counter + 1       # (3) change state
```

  - But NOTE that this is a definite loop – easier to use `for loop`

Repetition with a `while` loop

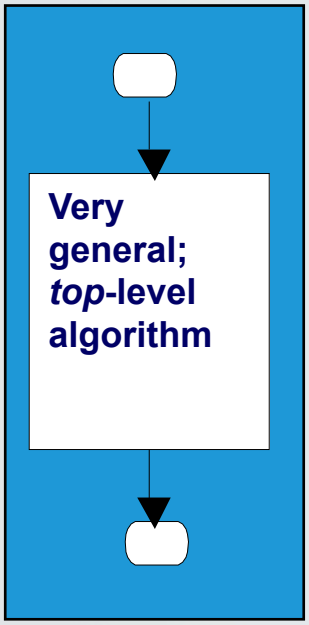
- While loops won't run at all if condition starts out as false
- While loops run forever if condition never becomes false (i.e. if it always stays true)

Applying `while`

- Better application example – unlimited data entry:

```
# (1) initialize
AllGrades = 0
grade = input("enter grade or q to quit: ")
# (2) check condition
while grade != "q":
    # process grade here, then get next one
    AllGrades = AllGrades + int(grades)
    # (3) change states
    grade = input("enter grade or q to quit: ")
# While loop has ended, now do other stuff…
print("You're all done now!")
```

Top-Design of Programs: Step 1



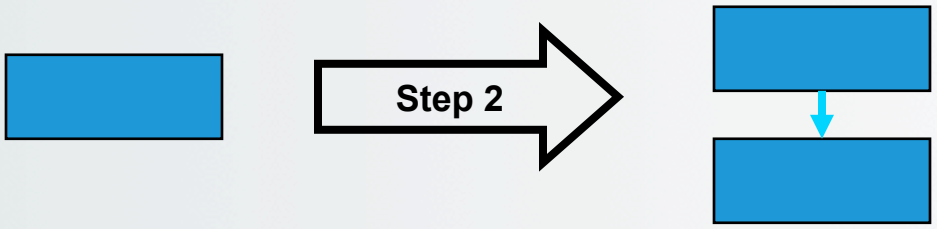
- Think of the simplest flowchart for your problem and think of the “big picture”

Example:

- I want to print all numbers between 1 and 100

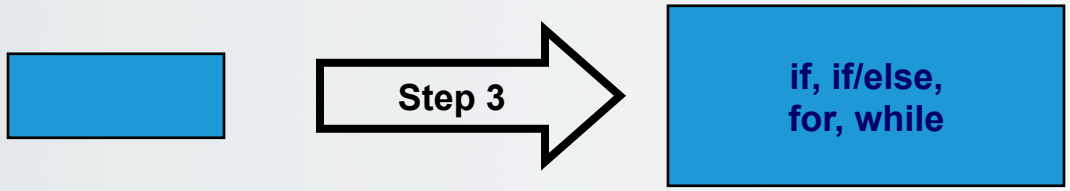
- Notice: just one rectangle in representation

Step 2: Replace Any Rectangle By Two Rectangles In Sequence



- This "stacking rule" can apply repeatedly
- For example:
  1. Get data
  2. Process
  3. Show results

Step 3: Replace Any Rectangle By Any *Control Structure*



- This "nesting rule" also applies repeatedly – each control structure has its own rectangles

- e.g., nest a **`while`** loop in an **`if`** structure:

```
if  n > 0:
   while i < n:
      print(i)
      i = i + 1
```

Step 4: Apply Steps #2 And #3 Repeatedly, And In Any Order

- Stack, nest, stack, nest, nest, stack, …
  gets more and more detailed as one proceeds
  - Think of control structures as building blocks that can be *combined in two ways only*.
- Overall process is known as "top-down design by stepwise refinement"
- Fact: *any algorithm* can be written as a combination of sequence, selection, and iteration structures.

</LECTURE>