CS 1124
Media Computation

Steve Harrison

Lecture 3.1 (September 8, 2008)

Today ...

- Review of Project 1
- One more thing about **`distance( color1, color2 )`**
- Ranges, sequences, matrixes, and indicies
  - **range(start, oneMoreThanTheLastOne)**
  - **[] notation**
  - **what can you do with []?**
- Nested loops

Pay close attention:
Today we'll have lots of "gotchas" that can cause problems!

Project 1

- Most people did a pretty good job
  - **Many recognized that negative and grayscale could be done in one loop**
  - **Others recognized that recipe could be made out of other recipes**
- Most common problem:
  - **def makeBandWNegative(file) :**
    - file is the name of the file, not the picture in the file
    - so the next line should be:
  - **picture = makePicture(file)**
    - WHY?

Project 1

- Most people did a pretty good job
  - **Many recognized that negative and grayscale could be done in one loop**

```
def makeBandWNegative(file):
  picture = makePicture(file)
  for p in getPixels(picture):
    newRed = 255-getRed(p)
    newGreen = 255-getGreen(p)
    newBlue = 255-getBlue(p)
    negcolor = makeColor(newRed, newGreen, newBlue)
    setColor(p,negcolor)
    luminance = (getRed(p)+getGreen(p)+getBlue(p))/3.
    setColor(p,makeColor(luminance, luminance, luminance))
  show(picture)
```

Project 1

- Most people did a pretty good job
  - **Many recognized that negative and grayscale could be done in one loop**

```
def makeBandWNegative(file):
  picture = makePicture(file)
  for p in getPixels(picture):
    newRed = 255-getRed(p)
    newGreen = 255-getGreen(p)
    newBlue = 255-getBlue(p)
    negcolor = makeColor(newRed, newGreen, newBlue)
    setColor(p,negcolor)
    luminance = (getRed(p)+getGreen(p)+getBlue(p))/3.
    setColor(p,makeColor(luminance, luminance, luminance))
  show(picture)
```

Project 1

- Most people did a pretty good job
  - **Many recognized that negative and grayscale could be done in one loop**

```
def makeBandWNegative(file):
  picture = makePicture(file)
  for p in getPixels(picture):
    newRed = 255-getRed(p)
    newGreen = 255-getGreen(p)
    newBlue = 255-getBlue(p)
    luminance = (newRed+newGreen+newBlue(p))/3.
    setColor(p,makeColor(luminance, luminance, luminance))
  show(picture)
```

  - **This idea is called "optimizing performance"**

Project 1

- Most people did a pretty good job
  - **Many recognized that negative and grayscale could be done in one loop**
  - **Others recognized that recipe could be made out of other recipes**
- Most common problem:
  - **def makeBandWNegative(file) :**
    - file is the name of the file, not the picture in the file
    - so the next line should be:
  - **picture = makePicture(file)**
    - WHY?

Project 1

- Most people did a pretty good job
  - **Others recognized that recipe could be made out of other recipes**

```
def makeBandWNegative(file):
  picture=makePicture(file)
  negative(picture)
  grayScale(picture)
  show(picture)


def negative(picture):
 for pixelArray in getPixels(picture):
   red = getRed(pixelArray)
   green = getGreen(pixelArray)
   blue = getBlue(pixelArray)
   negColor=makeColor(255-red,255-green,255-blue)
   setColor(pixelArray,negColor)

def grayScale(picture):
```

  - **This idea is called "modularity".**

Project 1

- Most people did a pretty good job
  - **Many recognized that negative and grayscale could be done in one loop**
  - **Others recognized that recipe could be made out of other recipes**
- Most common problem:
  - **def makeBandWNegative(file) :**
    - file is the name of the file, not the picture in the file
    - so the next line should be:
  - **picture = makePicture(file)**
    - WHY?

More about comments

- Some of you did not put in any comments - points off
- Some put # in first column, some followed indent level - both are OK

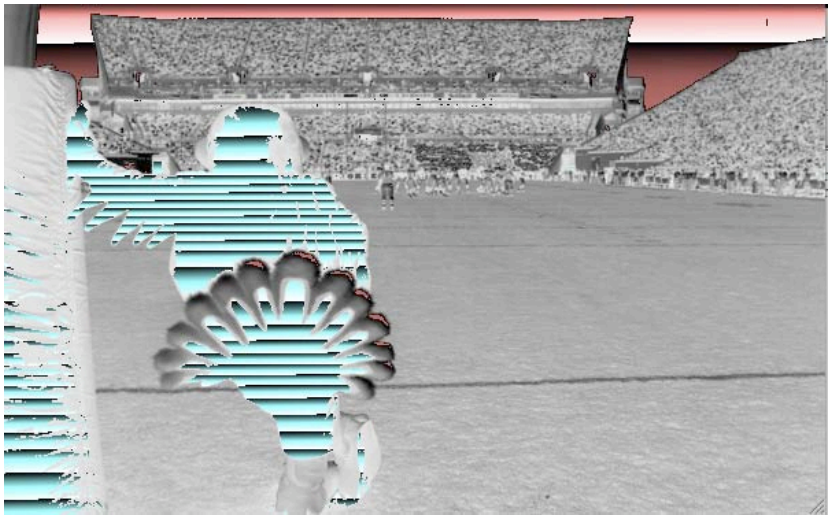
```
def makeBandWNegative(fileName) :
# create a picture to work with from the file
    picture = makePicture(fileName)
    # loop through each pixel in the picture
    for pxl in getPixels(picture)
```

- And some put # following the code on the same line

```
def makeBandWNegative(fileName) :
    picture = makePicture(fileName) # create a picture to work with
    for pxl in getPixels(picture)        # loop through each pixel in picture
```

Little bit extras…

- Most read ahead in the book and tried things
  - **THAT IS REALLY GREAT!**
  - **did comparative showing of picture, weighted grayscale, replaced one color with another, gradient …**



Worried about grade?

- Ways to make up...
  - **extra credit opportunities in group project**

- Post your questions to the project forum
- Come see us -- at least send e-mail

Distance as example of multiple functions

- So in the current recipe:
  -
```
def turnRed(file ):
  brown = makeColor(57, 16, 8)
  ...
  return(picture)


def distance( color1, color2 ):
 redDiff = getRed(color1) - getRed(color2)
 ...
 return(picture)
```

Using distance

```
def turnRed(file ):
   brown = makeColor(57, 16, 8)
   picture = makePicture(file)
   for px in getPixels(picture):
       color = getColor(px)
       if distance(color, brown) < 50.0:
          redness = getRed(px) * 1.5
          setRed(px, redness)
   show(picture)
   return(picture)
```

Original:

Digital makeover:

distance(color1, color2)

- It does the distance calculation:

$$\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$$

```
def distance( color1, color2 ):
    redDiff = getRed(color1) - getRed(color2)
    greenDiff = getGreen(color1) - getGreen(color2)
    blueDiff = getBlue(color1) - getBlue( color2)
    colorDistance = sqrt((redDiff*redDiff)+(greenDiff*greenDiff)+(blueDiff*blueDiff))
    return colorDistance
```

Hey, its purple!

distance(color1, color2)

- No need to define
- It is built in

```
def distance( color1, color2 ):
    redDiff = getRed(color1) - getRed(color2)
    greenDiff = getGreen(color1) - getGreen(color2)
    blueDiff = getBlue(color1) - getBlue( color2)
    colorDistance = sqrt((redDiff*redDiff)+(greenDiff*greenDiff)+(blueDiff*blueDiff))
    return colorDistance
```

Today ...

- Review of Project 1
- One more thing about **`distance( color1, color2 )`**
- Ranges, sequences, matrixes, and indicies
  - **range(start, oneMoreThanTheLastOne)**
  - **[] notation**
  - **what can you do with []?**
- Nested loops

Remember that pixels are in a matrix

- Matrices have two dimensions: A height and a width
- We can reference any element in the matrix with (x,y) or (horizontal, vertical)
  - **We refer to those coordinates as index numbers or indices**
- We sometimes want to know *where* a pixel is, and **getPixels** doesn't let us know that.

Tuning our color replacement

- If you want to get more of Barb's hair, just increasing the threshold doesn't work
  - **Wood behind becomes within the threshold value**
- How could we do it better?
  - **Lower our threshold, but then miss some of the hair**
  - **Work only within a range…**

Introducing the function range

- Range returns a sequence between its first two inputs, possibly using a third input as the increment

```
>>> print range(1, 4)
[1, 2, 3]
>>> print range(-1, 3)
[-1, 0, 1, 2]
>>> print range(1, 10, 2)
[1, 3, 5, 7, 9]
```

Bug Alert:
"range(start, oneMoreThanTheLastOne)"

That thing in [] is a sequence

```
>>> a=[1, 2, 3]
>>> print a
[1, 2, 3]
>>> a = a + 4
An attempt was made to call a
function with a parameter of an
invalid type
>>> a = a + [4]
>>> print a
[1, 2, 3, 4]
>>> a[0]
1
```

We can assign names to sequences, print them, add sequences, and access individual pieces of them.

We can also use **for** loops to process each element of a sequence.

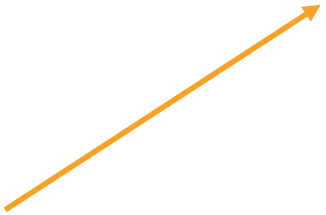
We can use range to generate index numbers

- We'll do this by working the range from 1 to the height, and 1 to the width
- But we'll need more than one loop.
  - **Each for loop can only change one variable, and we need two for a matrix**

Working the pixels by number

- To use **range**, we'll have to use *nested loops*
  - **One to walk the width, the other to walk the height**

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture, x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```



Bug Alert:
Be sure to watch your blocks carefully!

PAY CLOSE ATTENTION !!!!!!

- Look at page 71 in the textbook
- there are very similarly named functions that do different things:
  - **getPixel(picture, x, y) <> getPixels(picture)**
  - **getPixel returns the ONE pixel at that location**
  - **getPixels returns ALL the pixels in a picture**

- A simple rule of thumb is to re-read page 71 & 72 if your recipe is not working as you expect.
- "heuristic" is a term meaning "rule of thumb"

What's going on here?

The first time through the first loop, x is the name for 1.

We'll be processing the first column of pixels in the picture.

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture, x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```

(1, 1)	(2, 1)	(3, 1)	(x, 1)
(1, 1)	(2, 2)	(3, 2)	(x, 2)
(1, 3)	(2, 3)	(3, 3)	(x, 3)
(1, y)	(2, y)	(3, y)	(x, y)

Now, the inner loop

Next, we set y to 1. We're now going to process each of the pixels in column 1.

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture, x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```

(1, 1)	(2, 1)	(3, 1)	(x, 1)
(1, 1)	(2, 2)	(3, 2)	(x, 2)
(1, 3)	(2, 3)	(3, 3)	(x, 3)
(1, y)	(2, y)	(3, y)	(x, y)

Bug Alert:
The higher the y value, the lower it is on the picture.
Corollary heuristic:
"Height" means "down".

Process a pixel

With x = 1 and y = 1, we get the leftmost pixel and increase its red by 10%

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture, x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```

(1, 1)	(2, 1)	(3, 1)	(x, 1)
(1, 1)	(2, 2)	(3, 2)	(x, 2)
(1, 3)	(2, 3)	(3, 3)	(x, 3)
(1, y)	(2, y)	(3, y)	(x, y)

Next pixel

Next we set y to 2 (next value in the sequence *range(1, getHeight(picture) +1)*

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture, x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```

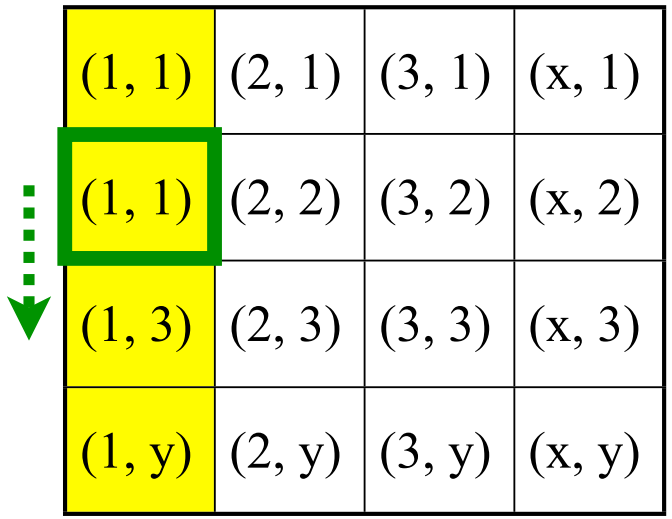
(1, 1)	(2, 1)	(3, 1)	(x, 1)
(1, 1)	(2, 2)	(3, 2)	(x, 2)
(1, 3)	(2, 3)	(3, 3)	(x, 3)
(1, y)	(2, y)	(3, y)	(x, y)

Process pixel (1,2)

x is still 1, and now y is 2, so increase the red for pixel (1,2)

We continue along this way, with y taking on every value from 1 to the height of the picture. (Remember, the range runs to LESS THAN the second parameter.)

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture,  x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```

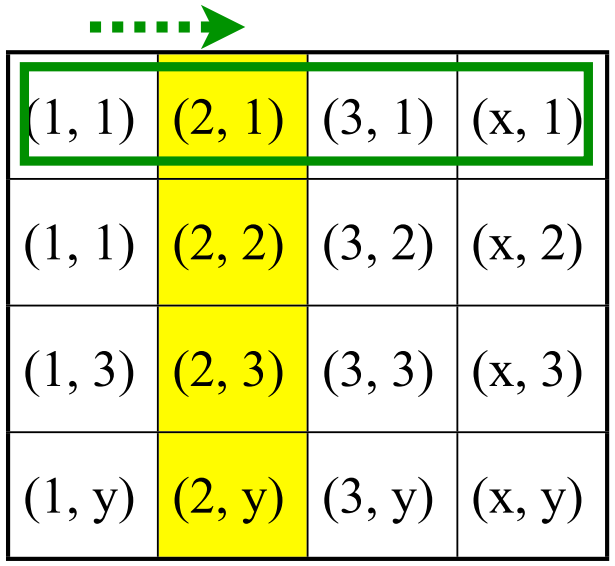


Finally, next column

Now that we're done with the loop for y, we get back to the for loop for x.

x now takes on the value 2, and we go back to the y loop to process all the pixels in the column x=2.

```
def increaseRed2(picture):
  for x in range(1, getWidth(picture)+1):
    for y in range(1, getHeight(picture)+1):
      px = getPixel(picture, x, y)
      value = getRed(px)
      setRed(px, value * 1.1)
```



BTW There are a 0 row and 0 column in a picture

Remember how the sequence example put its first element at position 0 ?

```
>>> a=[1, 2, 3]
>>> a[0]
1
```

JES doesn't use them for picture elements that are displayed.

Some JES functions use it for special, hidden purposes.

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(x, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(x, 1)
(0, 2)	(1, 1)	(2, 2)	(3, 2)	(x, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(x, 3)
(0, y)	(1, y)	(2, y)	(3, y)	(x, y)

Replacing colors in a range

Get the range using MediaTools

```
def turnRedInRange(file):
  brown = makeColor(57, 16, 8)
  picture = makePicture(file)
  for x in range(70, 168):
    for y in range(56, 190):
      px = getPixel(picture, x, y)
      color = getColor(px)
      if distance(color, brown) < 50.0:
        redness = getRed(px) * 1.5
        setRed(px, redness)
  show(picture)
  return(picture)
```

Walking this code

- Like last time:
  - **Don't need input parameters**
  - **same color we want to change**
  - **same file**
- make a picture

```
def turnRedInRange(file):
  brown = makeColor(57, 16, 8)
  picture = makePicture(file)
  for x in range(70, 168):
    for y in range(56, 190):
      px = getPixel(picture, x, y)
      color = getColor(px)
      if distance(color, brown) < 50.0:
        redness = getRed(px) * 1.5
        setRed(px, redness)
  show(picture)
  return(picture)
```

The nested loop

- Used MediaTools to find the rectangle where most of the hair is that we want to change

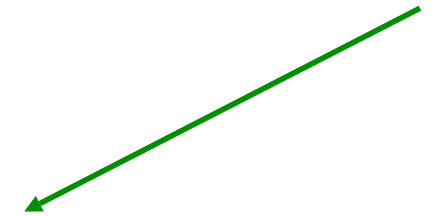
```
def turnRedInRange(file):
  brown = makeColor(57,16,8)
  picture = makePicture(file)
  for x in range(70, 168):
    for y in range(56, 190):
      px = getPixel(picture, x, y)
      color = getColor(px)
      if distance(color, brown) < 50.0:
        redness = getRed(px) * 1.5
        setRed(px, redness)
  show(picture)
  return(picture)
```

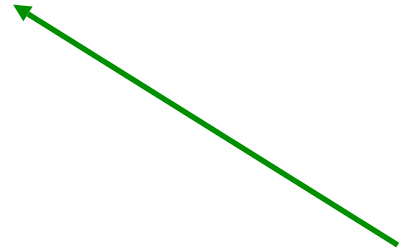
Scanning for brown hair

- We're looking for a close-match on hair color, and increasing the redness

```
picture = makePicture(file)
for x in range(70, 168):
  for y in range(56, 190):
    px = getPixel(picture, x, y)
    color = getColor(px)
    if distance(color, brown) < 50.0:
      redness = getRed(px) * 1.5
      setRed(px, redness)
show(picture)
return(picture)
```

Similar to scanning whole picture





We could raise threshold now. (Why?…)

Could we do this without nested loops?

- Yes, but only with a complicated **if** statement
- Less than optimal performance. Why?
- Moral: Nested loops are common for 2D data

```
def turnRedInRange2(file):
  brown = makeColor(57, 16, 8)
  picture = makePicture(file)
  for p in getPixels(picture):
    x = getX(p)
    y = getY(p)
    if x >= 70 and x < 168:
      if y >=56 and y < 190:
          color = getColor(p)
          if distance(color, brown) < 100.0:
            redness = getRed(p) * 2.0
            setRed(p, redness)
  show(picture)
  return picture
```

Moving pixels across pictures

- We've seen using index variables to track the pixel position we're working with in a picture.
- We can copy *between* pictures, if we keep track of:
  - **The source index variables**
    - Where we're getting the pixels *from*
  - **The target index variables**
    - Where we're putting the pixels *at*
- (Not really copying the pixels: Replicating their color.)

Questions?

Today ...

- Review of Project 1
- One more thing about **`distance( color1, color2 )`**
- Ranges, sequences, matrixes, and indicies
  - **range(start, oneMoreThanTheLastOne)**
  - **[] notation**
  - **what can you do with []?**
- Nested loops

Bug Alerts:

-“range(start, oneMoreThanTheLastOne)”

- Be sure to watch your blocks carefully!

- The higher the y value, the lower it is on the picture.

Coming Attractions

- GTA Matt Schaefer is away this week, so make an appointment with Bobby Beaton <rbeaton@vt.edu>if you have questions or need help.
- For Friday
  - **Project 2 Due**
- For Monday
  - **(re)Read Chapter 4**
  - **Quiz 4 due 10:00 AM**