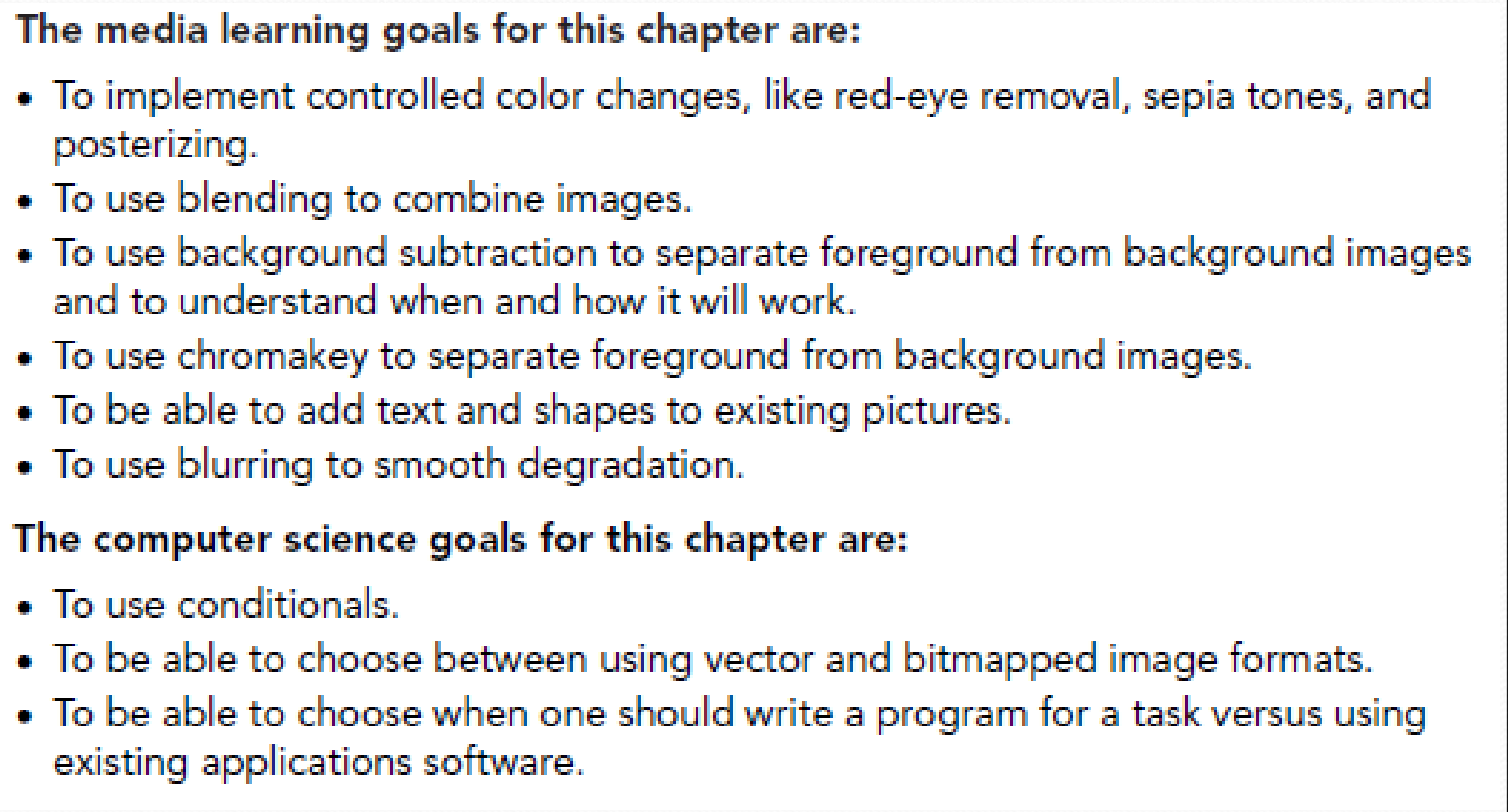
Introduction to Computing and Programming in Python:

A Multimedia Approach

Chapter 5: Advanced Picture Techniques

Chapter Objectives



The media learning goals for this chapter are:

- To implement controlled color changes, like red-eye removal, sepia tones, and posterizing.
- To use blending to combine images.
- To use background subtraction to separate foreground from background images and to understand when and how it will work.
- To use chromakey to separate foreground from background images.
- To be able to add text and shapes to existing pictures.
- To use blurring to smooth degradation.

The computer science goals for this chapter are:

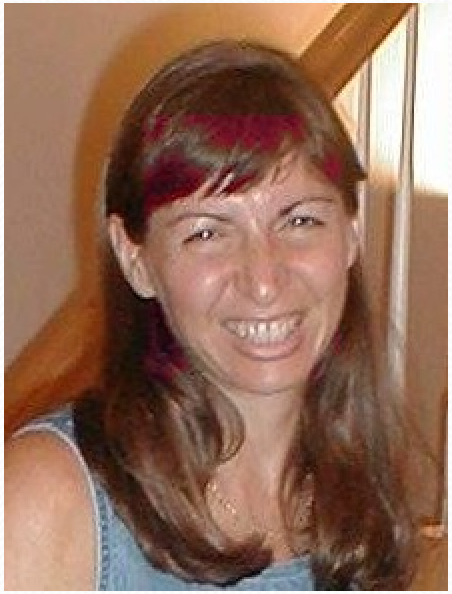
- To use conditionals.
- To be able to choose between using vector and bitmapped image formats.
- To be able to choose when one should write a program for a task versus using existing applications software.

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…

Replacing colors in a range

Get the range using MediaTools



```
def turnRedInRange():
  brown = makeColor(57,16,8)
  file=“/Users/guzdial/mediasources/barbara.jpg"
  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, same color we want to change, same file, make a picture

```
def turnRedInRange():
  brown = makeColor(57,16,8)
  file=“/Users/guzdial/mediasources/barbara.jpg"
  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

- I used MediaTools to find the rectangle where most of the hair is that I want to change

```
def turnRedInRange():
 brown = makeColor(57,16,8)
 file=“/Users/guzdial/mediasources/barbara.jpg“
 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)
```

Same thing as last time (could raise threshold now)

- Then we’re looking for a close-match on hair color, and increasing the redness

```
def turnRedInRange():
  brown = makeColor(57,16,8)
  file=“/Users/guzdial/mediasources/barbara.jpg“
  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)
```

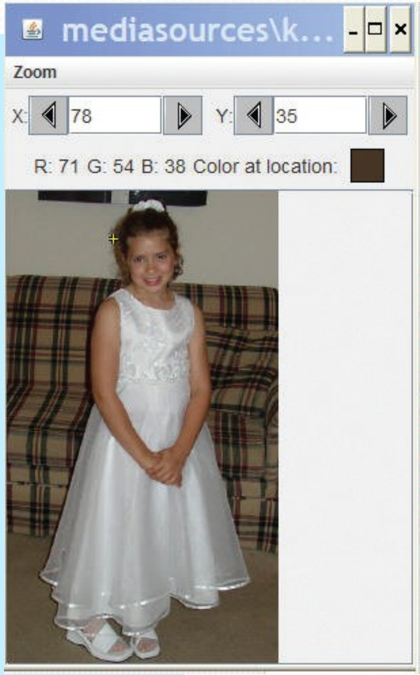
Could we do this without nested loops?

- Yes, but complicated IF

```
def turnRedInRange2():
  brown = makeColor(57,16,8)
  file=“/Users/guzdial/mediasources/barbara.jpg“
  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
```

Working on Katie's Hair

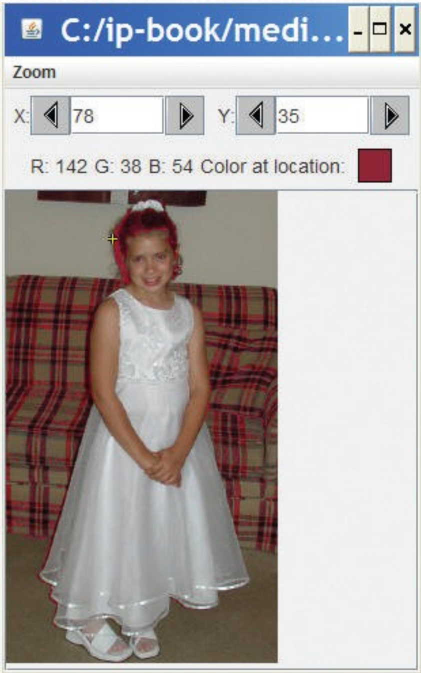
```
def turnRed():
  brown = makeColor(42,25,15)
  file="C:/ip-book/mediasources/katieFancy.jpg"
  picture=makePicture(file)
  for px in getPixels(picture):
    color = getColor(px)
    if distance(color,brown)<50.0:
      redness=int(getRed(px)*2)
      blueness=getBlue(px)
      greenness=getGreen(px)
      setColor(px,makeColor(redness,blueness,greenness))
  show(picture)
  return(picture)
```



mediasources\k...

X: 78 Y: 35

R: 71 G: 54 B: 38 Color at location:



C:/ip-book/medi...

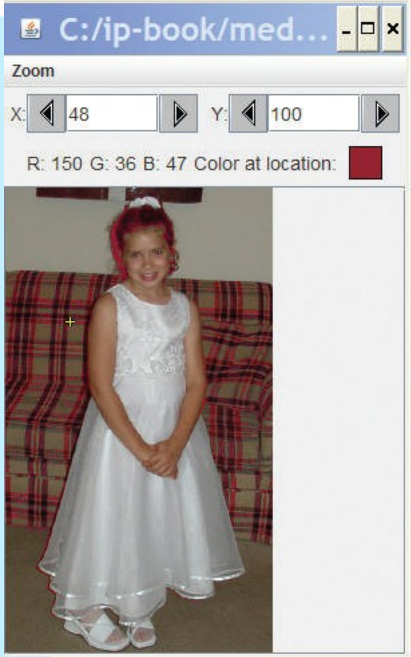
X: 78 Y: 35

R: 142 G: 38 B: 54 Color at location:

This version doubles all "close" reds. Notice the couch.

Working on Katie’s hair, in a range

```
def turnRedInRange():
  brown = makeColor(42,25,15)
  file="C:/ip-book/mediasources/katieFancy.jpg"
  picture=makePicture(file)
  for x in range(63,125):
    for y in range(6,76):
      px=getPixel(picture,x,y)
      color = getColor(px)
      if distance(color,brown)<50.0:
        redness=int(getRed(px)*2)
        blueness=getBlue(px)
        greenness=getGreen(px)
        setColor(px,makeColor(redness,blueness,greenness))
  show(picture)
  return(picture)
```

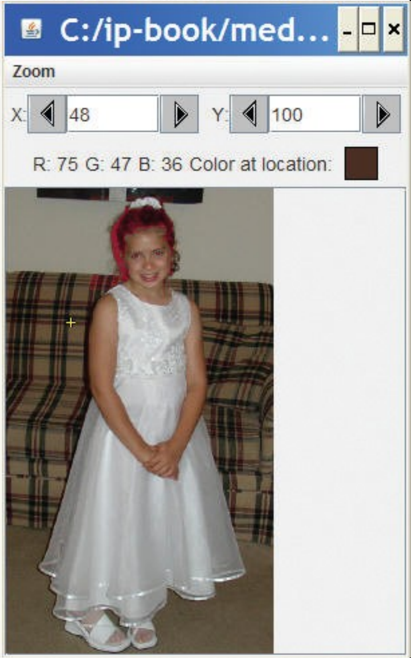


C:/ip-book/med...

Zoom

X: 48 Y: 100

R: 150 G: 36 B: 47 Color at location: 🟥



C:/ip-book/med...

Zoom

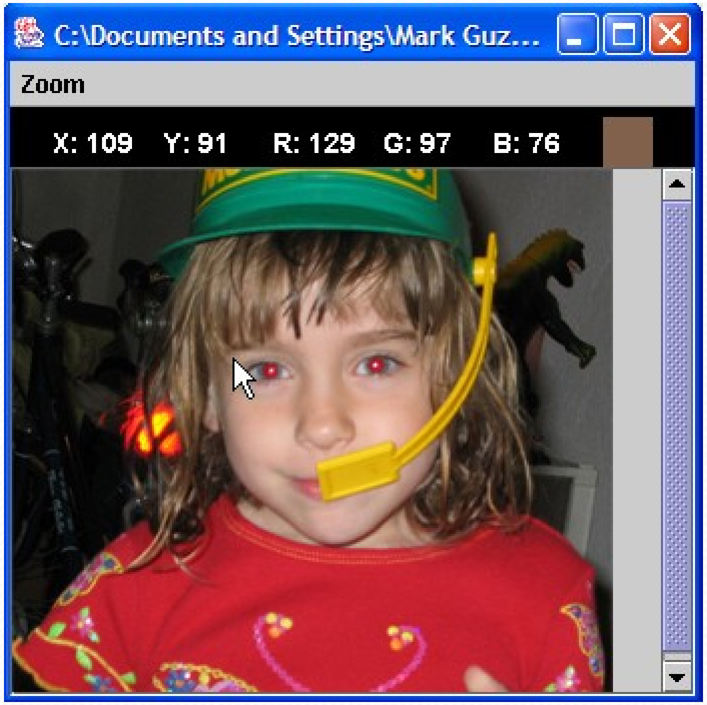
X: 48 Y: 100

R: 75 G: 47 B: 36 Color at location: 🟫

Left is one we did with all “close” browns. Right is same, but only in rect around head.

Removing "Red Eye"

- When the flash of the camera catches the eye just right (especially with light colored eyes), we get bounce back from the back of the retina.
- This results in "red eye"
- We can replace the "red" with a color of our choosing.
- First, we figure out *where* the eyes are (x,y) using MediaTools


C:\Documents and Settings\Mark Guz...
Zoom
X: 109 Y: 91 R: 129 G: 97 B: 76

Removing Red Eye

```
def removeRedEye(pic,startX,startY,endX,endY,replacementcolor):
  red = makeColor(255,0,0)
  for x in range(startX,endX):
    for y in range(startY,endY):
      currentPixel = getPixel(pic,x,y)
      if (distance(red,getColor(currentPixel)) < 165):
        setColor(currentPixel,replacementcolor)
```

Why use a range? Because we don’t want to replace her red dress!

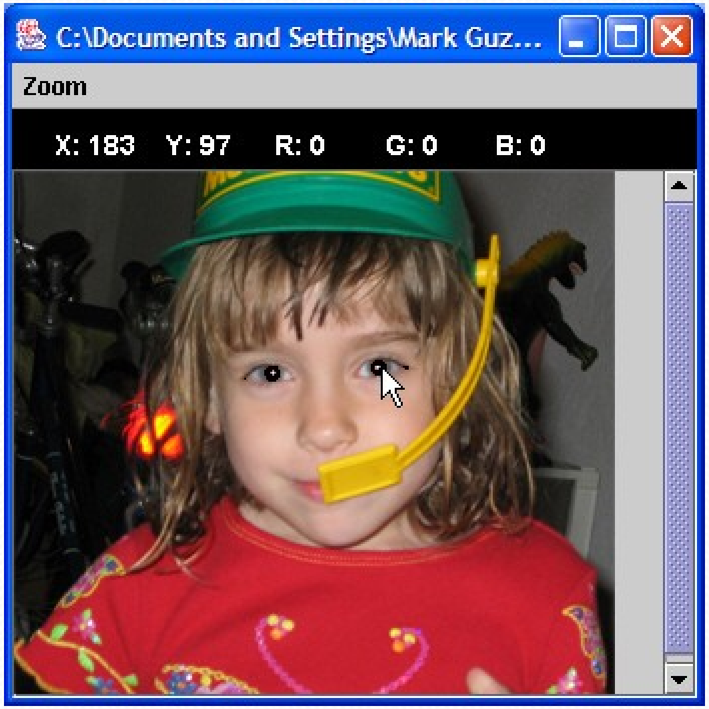
What we’re doing here:

- **Within the rectangle of pixels (startX,startY) to (endX, endY)**
- **Find pixels close to red, then replace them with a new color**

“Fixing” it: Changing red to black

```
removeRedEye(jenny, 109,
  91, 202, 107,
  makeColor(0,0,0))
```

- Jenny’s eyes are actually not black—could fix that
- Eye are also not mono-color
  - A better function would handle *gradations* of red and replace with *gradations* of the right eye color

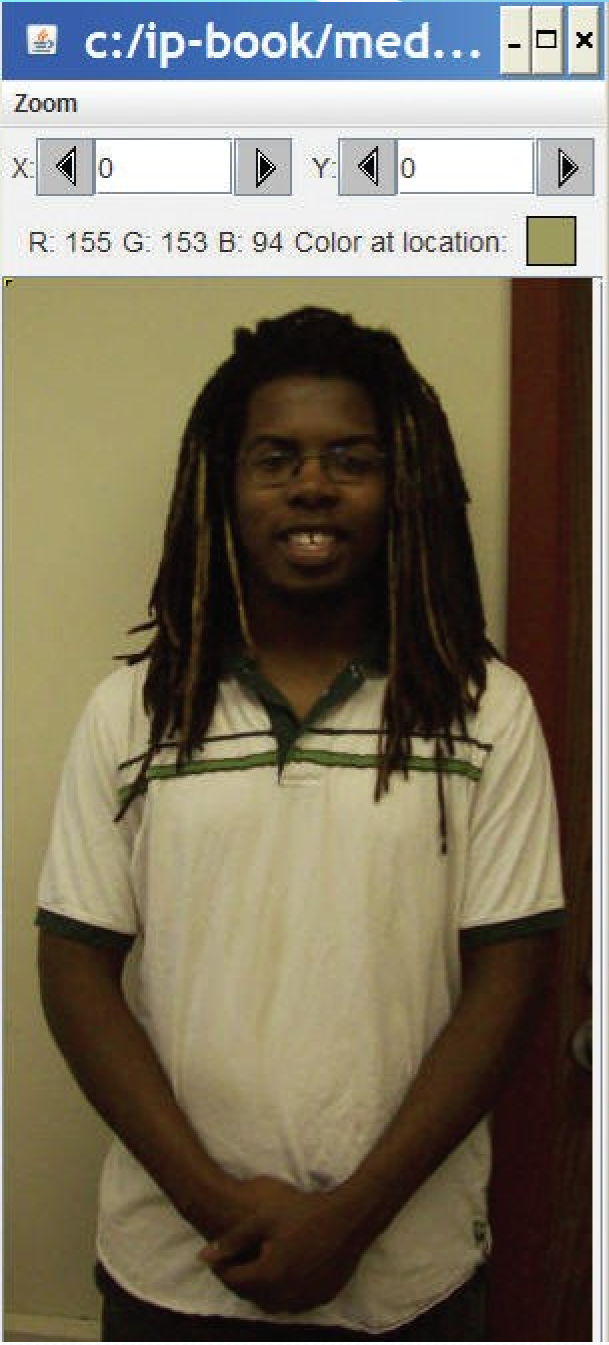


X: 183 Y: 97 R: 0 G: 0 B: 0

Replacing colors using IF

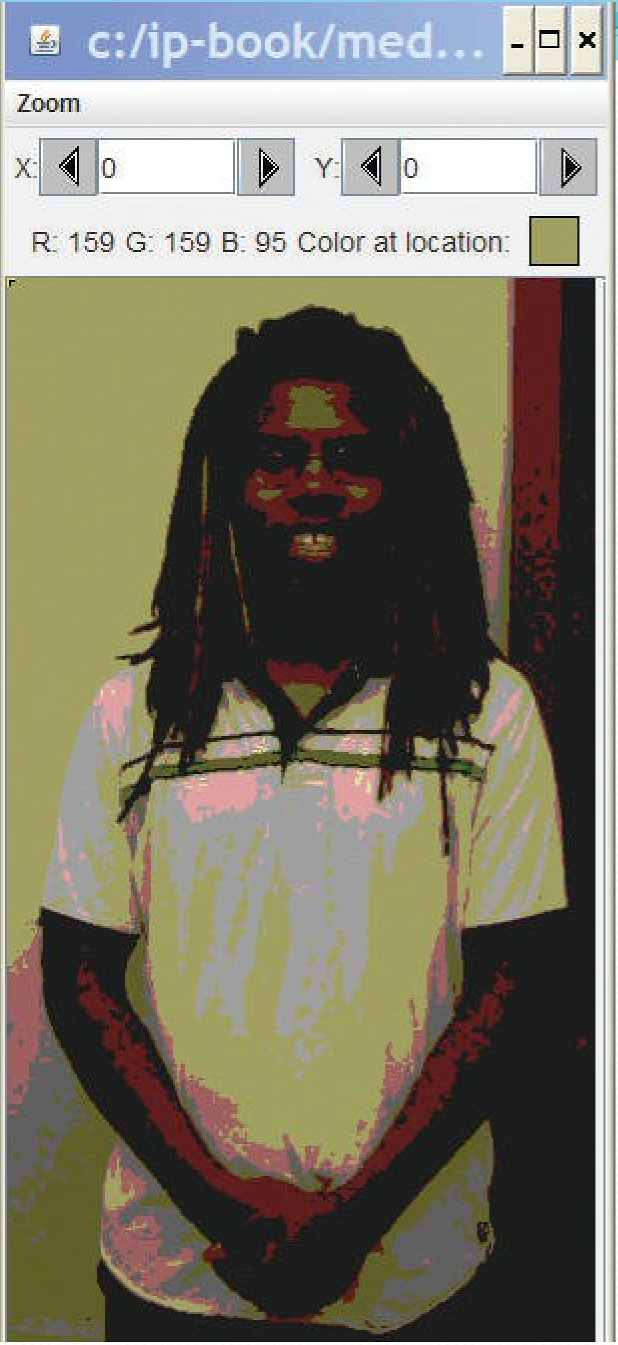
- We don’t have to do one-to-one changes or replacements of color
- We can use **if** to decide if we want to make a change.
  - We could look for a range of colors, or one specific color.
  - We could use an operation (like multiplication) to set the new color, or we can set it to a specific value.
- It all depends on the effect that we want.

Posterizing: Reducing range of colors



X: 0 Y: 0

R: 155 G: 153 B: 94 Color at location:



X: 0 Y: 0

R: 159 G: 159 B: 95 Color at location:

Posterizing: How we do it

- We look for a *range* of colors, then map them to a *single* color.
  - If red is between 63 and 128, set it to 95
  - If green is less than 64, set it to 31
  - …
- It requires a lot of **if** statements, but it's really pretty simple.
- The end result is that a *bunch* of different colors, get set to a *few* colors.

Posterizing function

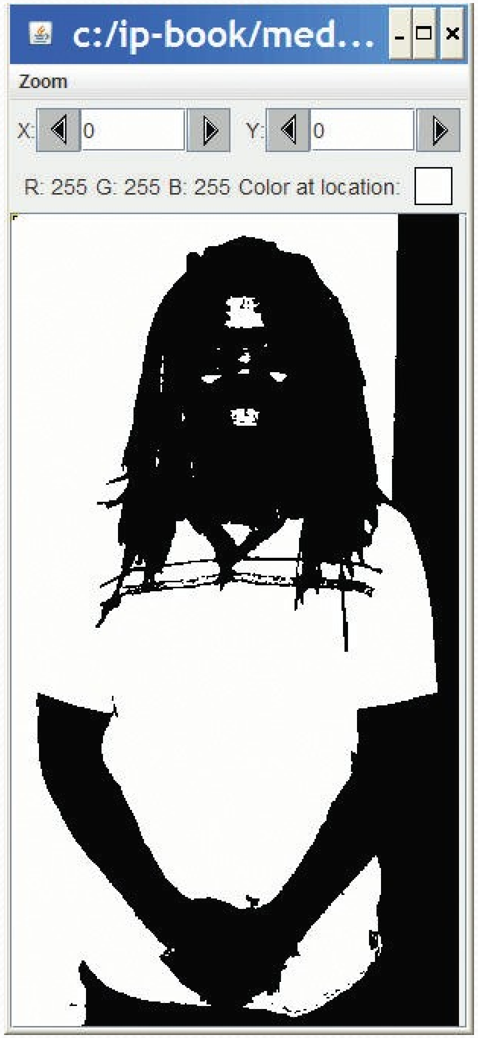
```
def posterize(picture):
  #loop through the pixels
  for p in getPixels(picture):
    #get the RGB values
    red = getRed(p)
    green = getGreen(p)
    blue = getBlue(p)

    #check and set red values
    if(red < 64):
      setRed(p, 31)
    if(red > 63 and red < 128):
      setRed(p, 95)
    if(red > 127 and red < 192):
      setRed(p, 159)
    if(red > 191 and red < 256):
      setRed(p, 223)

    #check and set green values
    if(green < 64):
      setGreen(p, 31)
    if(green > 63 and green < 128):
      setGreen(p, 95)
    if(green > 127 and green < 192):
      setGreen(p, 159)
    if(green > 191 and green < 256):
      setGreen(p, 223)

    #check and set blue values
    if(blue < 64):
      setBlue(p, 31)
    if(blue > 63 and blue < 128):
      setBlue(p, 95)
    if(blue > 127 and blue < 192):
      setBlue(p, 159)
    if(blue > 191 and blue < 256):
      setBlue(p, 223)
```

What’s with this “#” stuff?

- Any line that starts with a “#” is *ignored* by Python.
- This allows you to insert *comments*: Notes to yourself (or another programmer) that explains what’s going on here.
  - When programs get longer, there are lots of pieces to them, and it’s hard to figure out what each piece does.
  - Comments can help.

Posterizing to b/w levels

```
def grayPosterize(pic):
  for p in getPixels(pic):
    r = getRed(p)
    g = getGreen(p)
    b = getBlue(p)
    luminance = (r+g+b)/3
    if luminance < 64:
      setColor(p,black)
    if luminance >= 64:
      setColor(p,white)
```



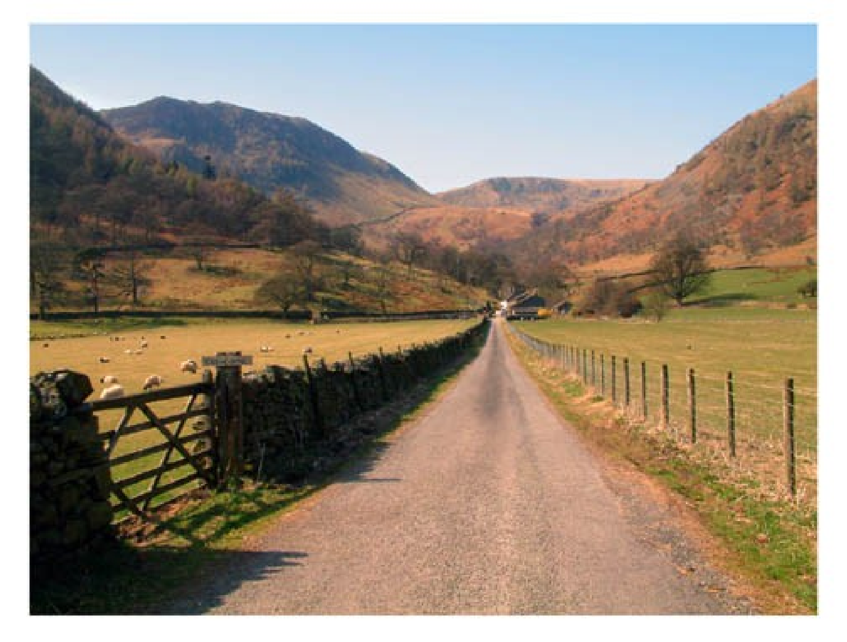
R: 255 G: 255 B: 255 Color at location:

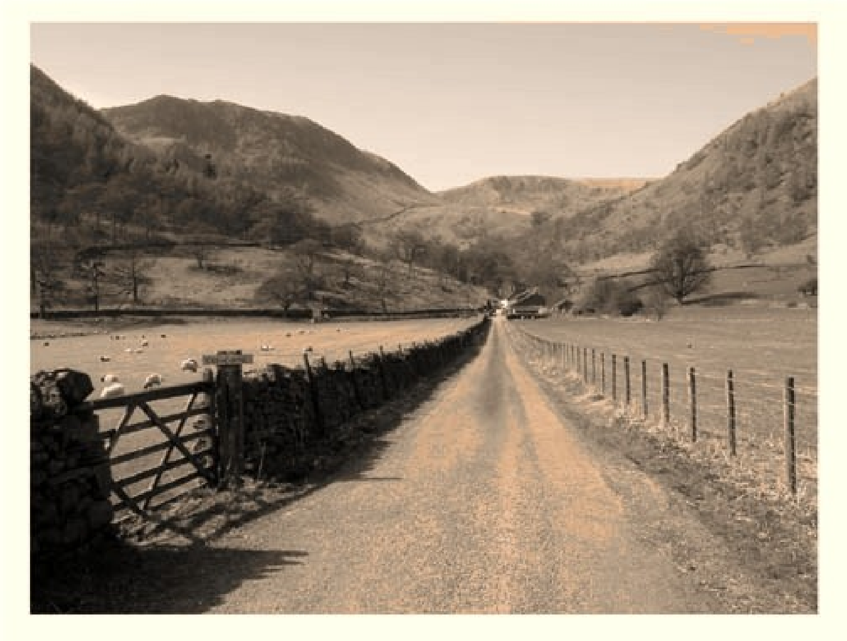
We check luminance on each pixel.
If it's low enough, it's black, and Otherwise, it's white

Generating sepia-toned prints

- Pictures that are *sepia-toned* have a yellowish tint to them that we associate with older pictures.
- It's not directly a matter of simply increasing the yellow in the picture, because it's not a one-to-one correspondence.
  - Instead, colors in different ranges get mapped to other colors.
  - We can create such a mapping using IF

Example of sepia-toned prints





Here’s how we do it

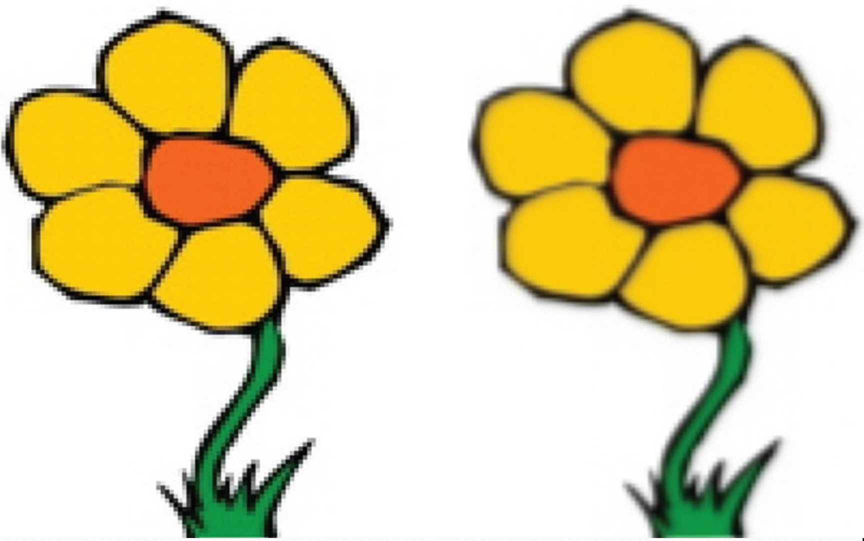
```
def sepiaTint(picture):
  #Convert image to greyscale
  greyScaleNew(picture)

  #loop through picture to tint pixels
  for p in getPixels(picture):
    red = getRed(p)
    blue = getBlue(p)

    #tint shadows
    if (red < 63):
      red = red*1.1
      blue = blue*0.9
```

```
    #tint midtones
    if (red > 62 and red < 192):
      red = red*1.15
      blue = blue*0.85

    #tint highlights
    if (red > 191):
      red = red*1.08
      if (red > 255):
        red = 255

      blue = blue*0.93

    #set the new color values
    setBlue(p, blue)
    setRed(p, red)
```

What’s going on here?

- First, we’re calling greyScaleNew (the one with weights).
  - It’s perfectly okay to have one function calling another.
- We then manipulate the red (increasing) and the blue (decreasing) channels to bring out more yellows and oranges.
  - Why are we doing the comparisons on the red? Why *not*? After greyscale conversion, all channels are the same!
- Why these values? Trial-and-error: Twiddling the values until it looks the way that you want.

Blurring



- When we scale up pictures (make them bigger), we get sharp lines and boxes: *pixelation*.
- Can reduce that by *purposefully* blurring the image.
  - One simple algorithm: Take the pixels left, right, bottom, and top of yours. Average the colors.

Blurring code

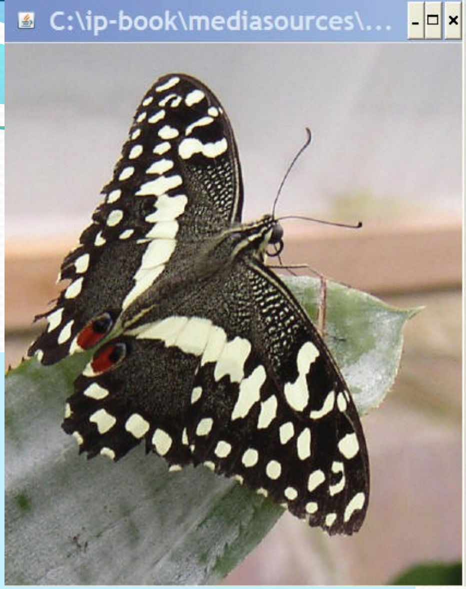
```
def blur(filename):
  source=makePicture(filename)
  target=makePicture(filename)
  for x in range(0, getWidth(source)-1):
    for y in range(0, getHeight(source)-1):
      top = getPixel(source,x,y-1)
      left = getPixel(source,x-1,y)
      bottom = getPixel(source,x,y+1)
      right = getPixel(source,x+1,y)
      center = getPixel(target,x,y)
      newRed=(getRed(top)+ getRed(left)+ getRed(bottom)+getRed(right)+ getRed(center))/5
      newGreen=(getGreen(top)+ getGreen(left)+getGreen(bottom)+getGreen(right)
     +getGreen(center))/5
      newBlue=(getBlue(top)+ getBlue(left)+ getBlue(bottom)+getBlue(right)+ getBlue(center))/5
      setColor(center, makeColor(newRed, newGreen, newBlue))
  return target
```

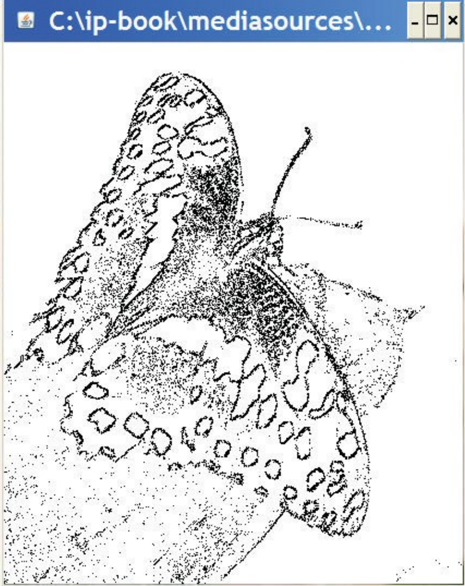
We make two copies of the picture.
We read pixel colors from one, and set them in the other.

Edge Detection

- Blurring is averaging across pixels.
- Edge detection is looking for *differences* between pixels.
  - We draw lines that our eyes see—where the luminance changes.
- If the pixel changes left-to-right, up and down, then we make our pixel black. Else white.

Edge Detection





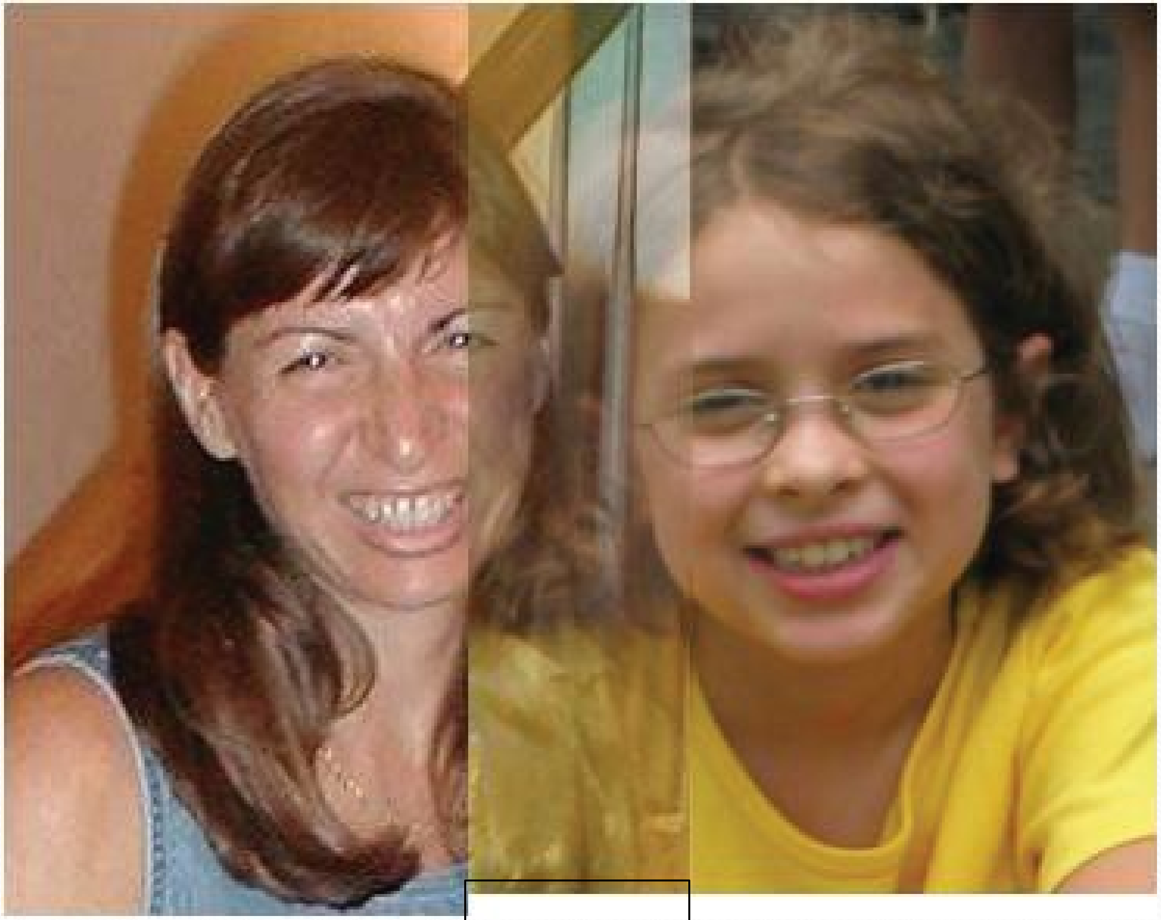
```
def lineDetect(filename):
  orig = makePicture(filename)
  makeBw = makePicture(filename)
  for x in range(0,getWidth(orig)-1):
    for y in range(0,getHeight(orig)-1):
      here=getPixel(makeBw,x,y)
      down=getPixel(orig,x,y+1)
      right=getPixel(orig,x+1,y)
      hereL=(getRed(here)+getGreen(here)+getBlue(here))/3
      downL=(getRed(down)+getGreen(down)+getBlue(down))/3
      rightL=(getRed(right)+getGreen(right)+getBlue(right))/3
      if abs(hereL-downL)>10 and abs(hereL-rightL)>10:
        setColor(here,black)
      if abs(hereL-downL)<=10 and abs(hereL-rightL)<=10:
        setColor(here,white)
  return makeBw
```

Notice the use of absolute value (abs) here. We don’t care which is larger. We care about a large difference.

Blending pictures

- How do we get part of one picture and part of another to blur together, so that we see some of each?
  - It's about making one a bit "transparent."
  - Video cards sometimes support this transparency in hardware, called an *alpha level* to each pixel.
- We do it as a *weighted sum*
  - If it's 50-50, we take 50% of red of picture1's pixels + 50% of red of picture2's pixels, and so on for green and blue, across all overlapping pixels.

Example blended picture



Blended here

Blending code (1 of 3)

```
def blendPictures():
  barb = makePicture(getMediaPath("barbara.jpg"))
  katie = makePicture(getMediaPath("Katie-smaller.jpg"))
  canvas = makePicture(getMediaPath("640x480.jpg"))
  #Copy first 150 columns of Barb
  sourceX=0
  for targetX in range(0,150):
    sourceY=0
    for targetY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY),color)
      sourceY = sourceY + 1
    sourceX = sourceX + 1
```

Straightforward copy of 150 column's of Barb's picture

Blending code (2 of 3)

```
 #Now, grab the rest of Barb and part of Katie
# at 50% Barb and 50% Katie
overlap = getWidth(barb)-150
sourceX=0
for targetX in range(150,getWidth(barb)):
  sourceY=0
  for targetY in range(0,getHeight(katie)):
    bPixel = getPixel(barb,sourceX+150,sourceY)
    kPixel = getPixel(katie,sourceX,sourceY)
    newRed= 0.50*getRed(bPixel)+0.50*getRed(kPixel)
    newGreen=0.50*getGreen(bPixel)+0.50*getGreen(kPixel)
    newBlue = 0.50*getBlue(bPixel)+0.50*getBlue(kPixel)
    color = makeColor(newRed,newGreen,newBlue)
    setColor(getPixel(canvas,targetX,targetY),color)
    sourceY = sourceY + 1
  sourceX = sourceX + 1
```

Here’s the trick. For each pixel, grab 50% of each red, green and blue

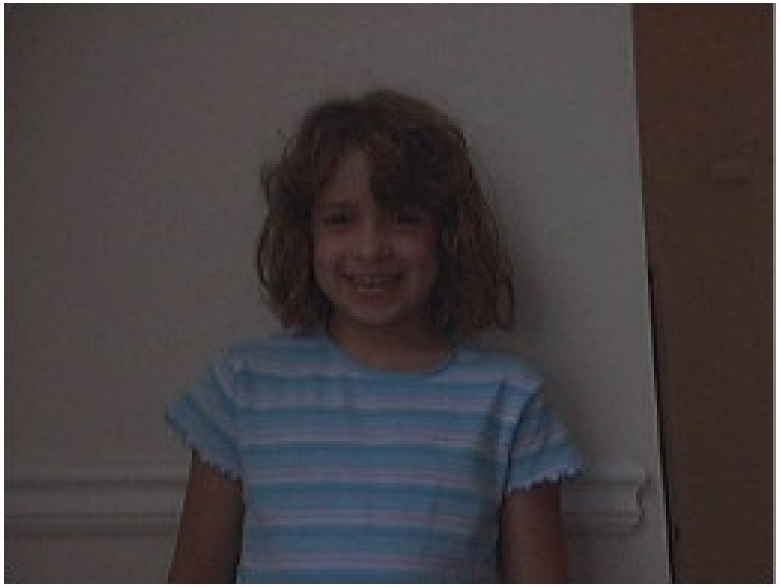
Blending code (3 of 3)

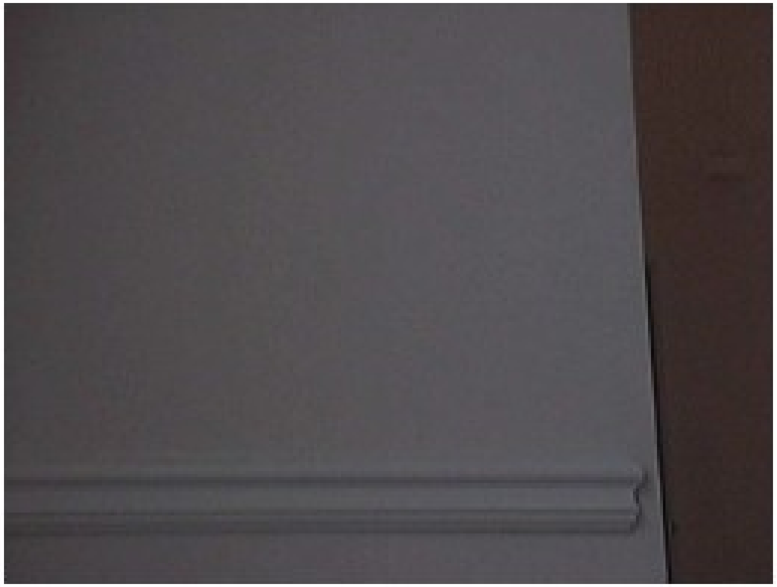
```
# Last columns of Katie
 sourceX=overlap
 for targetX in range(150+overlap,150+getWidth(katie)):
   sourceY=0
   for targetY in range(0,getHeight(katie)):
     color = getColor(getPixel(katie,sourceX,sourceY))
     setColor(getPixel(canvas,targetX,targetY),color)
     sourceY = sourceY + 1
   sourceX = sourceX + 1
 show(canvas)
 return canvas
```

Background subtraction

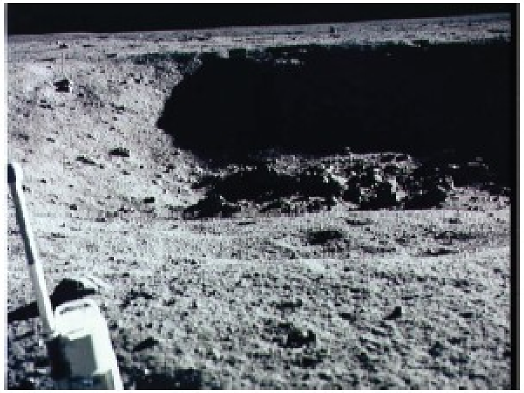
- Let’s say that you have a picture of someone, and a picture of the same place (same background) without the someone there, could you subtract out the background and leave the picture of the person?
- Maybe even change the background?
- Let’s take that as our problem!

Person (Katie) and Background





Let's put
Katie on
the moon!

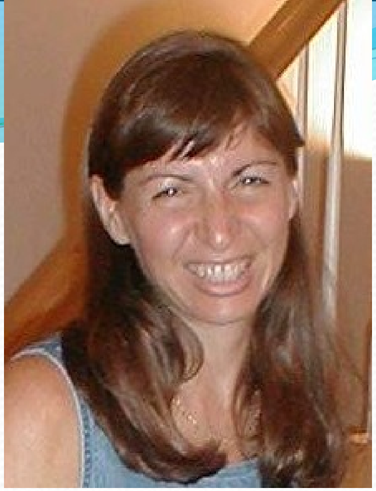


Where do we start?

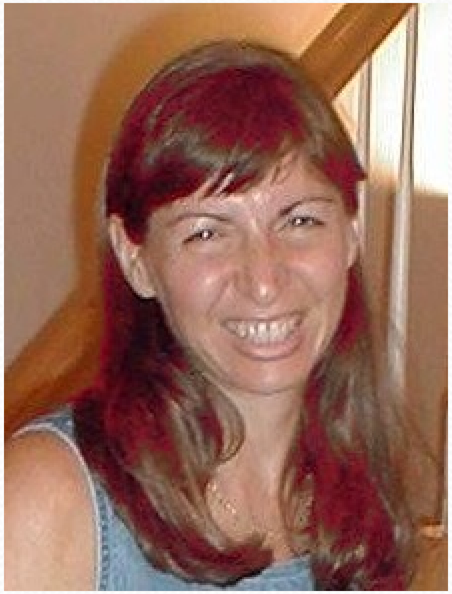
- What we most need to do is to figure out whether the pixel in the Person shot is the same as the in the Background shot.
- Will they be the EXACT same color? Probably not.
- So, we'll need some way of figuring out if two colors are close…

Remember this?

Original:



```
def turnRed():
  brown = makeColor(57,16,8)
  file = r"C:\Documents and Settings\Mark Guzdial\My
Documents\mediasources\barbara.jpg"
  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)
```



Using distance

- So we know that we want to ask:

  ```
  if distance(personColor,bgColor) > someValue
  ```

- And what do we then?
  - We want to grab the color from another background (a new background) at the same point.
  - Do we have any examples of doing that?

Copying Barb to a canvas

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 1
  for sourceX in range(1,getWidth(barb)):
    targetY = 1
    for sourceY in range(1,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Where we are so far:

```
if distance(personColor,bgColor) > someValue:
  bgcolor = getColor(getPixel(newBg,x,y))
  setColor(getPixel(person,x,y), bgcolor)
```

- What else do we need?
  - We need to get all these variables set up
    - We need to input a person picture, a background (background without person), and a new background.
    - We need a loop where x and y are the right values
    - We have to figure out personColor and bgColor

Swap a background using background subtraction

```
def swapbg(person, bg, newbg):
  for x in range(1,getWidth(person)):
    for y in range(1,getHeight(person)):
      personPixel = getPixel(person,x,y)
      bgpx = getPixel(bg,x,y)
      personColor= getColor(personPixel)
      bgColor = getColor(bgpx)
      if distance(personColor,bgColor) > someValue:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(getPixel(person,x,y), bgcolor)
```

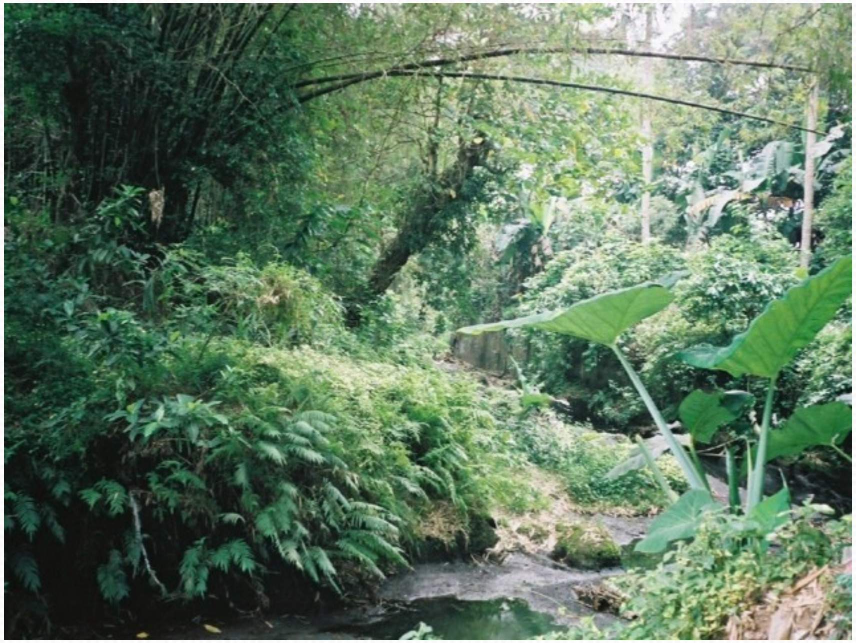
Simplifying a little, and specifying a little

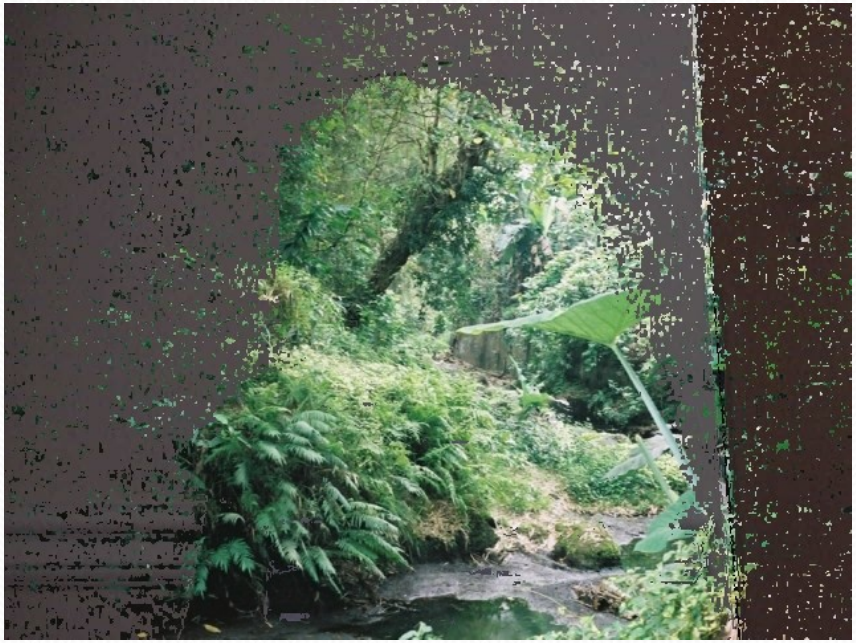
```
def swapbg(person, bg, newbg):
  for x in range(1,getWidth(person)):
    for y in range(1,getHeight(person)):
      personPixel = getPixel(person,x,y)
      bgpx = getPixel(bg,x,y)
      personColor= getColor(personPixel)
      bgColor = getColor(bgpx)
      if distance(personColor,bgColor) > 10:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(personPixel, bgcolor)
```

Specifying a threshold.

Using a variable for the person pixel

Trying it with a jungle background





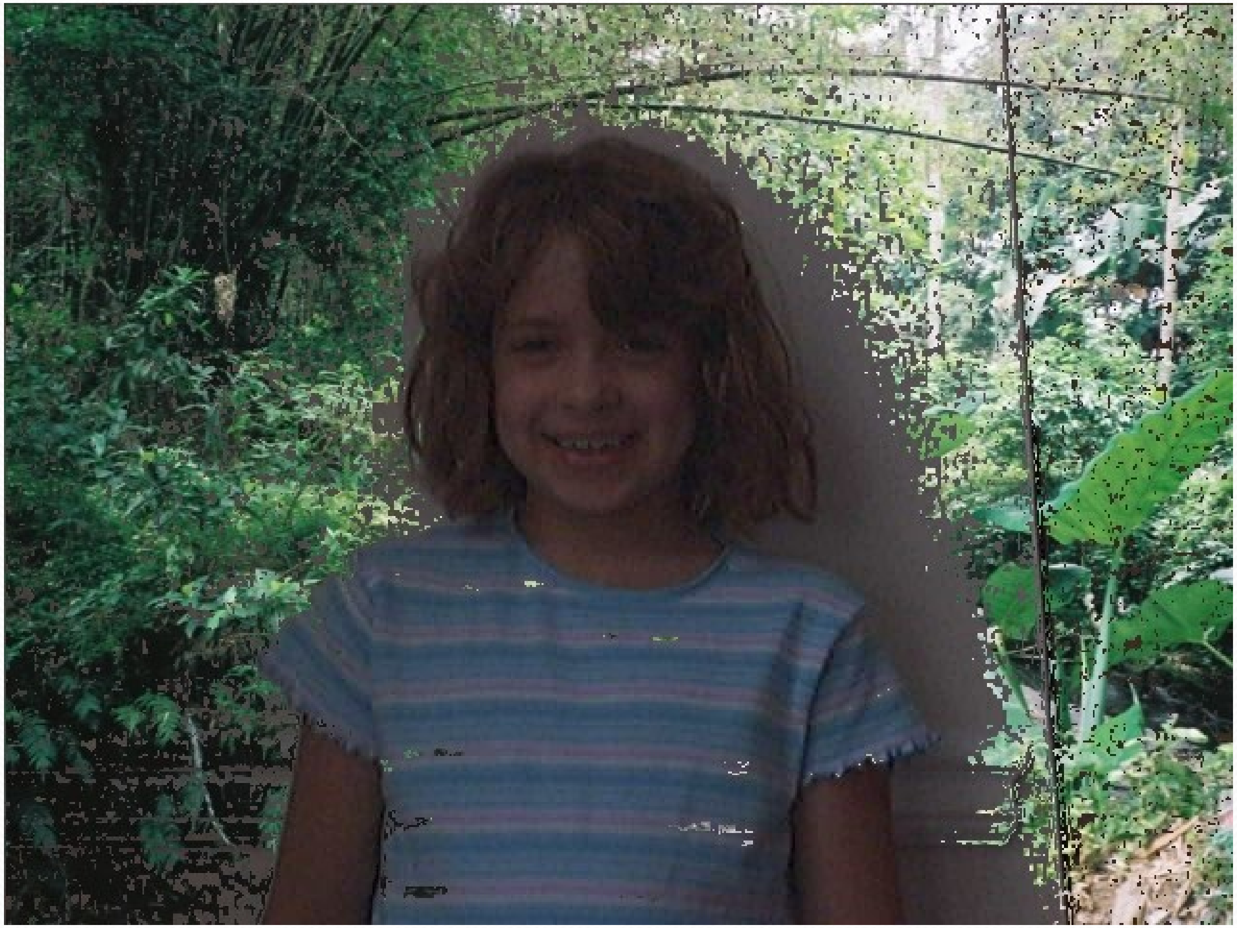
What happened?

- It looks like we reversed the swap
  - If the distance is great, we want to KEEP the pixel.
  - If the distance is small (it's basically the same thing), we want to get the NEW pixel.

Reversing the swap

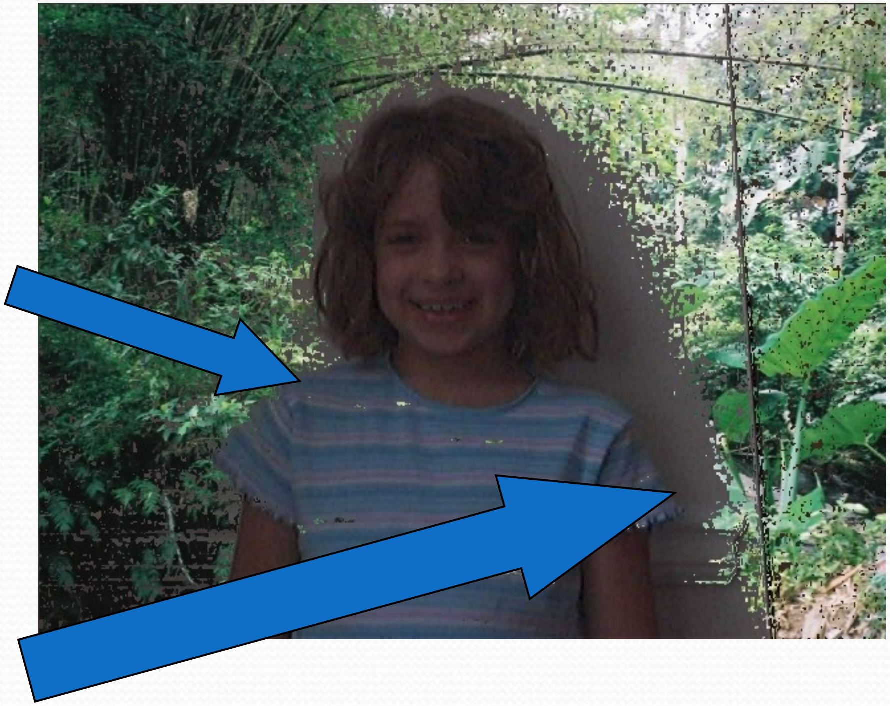
```
def swapbg(person, bg, newbg):
  for x in range(1,getWidth(person)):
    for y in range(1,getHeight(person)):
      personPixel = getPixel(person,x,y)
      bgpx = getPixel(bg,x,y)
      personColor= getColor(personPixel)
      bgColor = getColor(bgpx)
      if distance(personColor,bgColor) < 10:
        bgcolor = getColor(getPixel(newbg,x,y))
        setColor(personPixel, bgcolor)
```

Better!



But why isn’t it alot better?

- We’ve got places where we got pixels swapped that we didn’t want to swap
  - See Katie’s shirt stripes
- We’ve got places where we want pixels swapped, but didn’t get them swapped
  - See where Katie made a shadow



How could we make it better?

- What could we change in the program?
  - We could change the threshold “someValue”
  - If we increase it, we get *fewer* pixels matching
    - That won’t help with the shadow
  - If we decrease it, we get *more* pixels matching
    - That won’t help with the stripe

- What could we change in the pictures?
  - Take them in better light, less shadow
  - Make sure that the person isn’t wearing clothes near the background colors.

Another way: Chromakey

- Have a background of a *known* color
  - Some color that won't be on the person you want to *mask* out
  - Pure green or pure blue is most often used
  - I used my son's blue bedsheet
- This is how the weather people seem to be in front of a map—they're actually in front of a blue sheet.

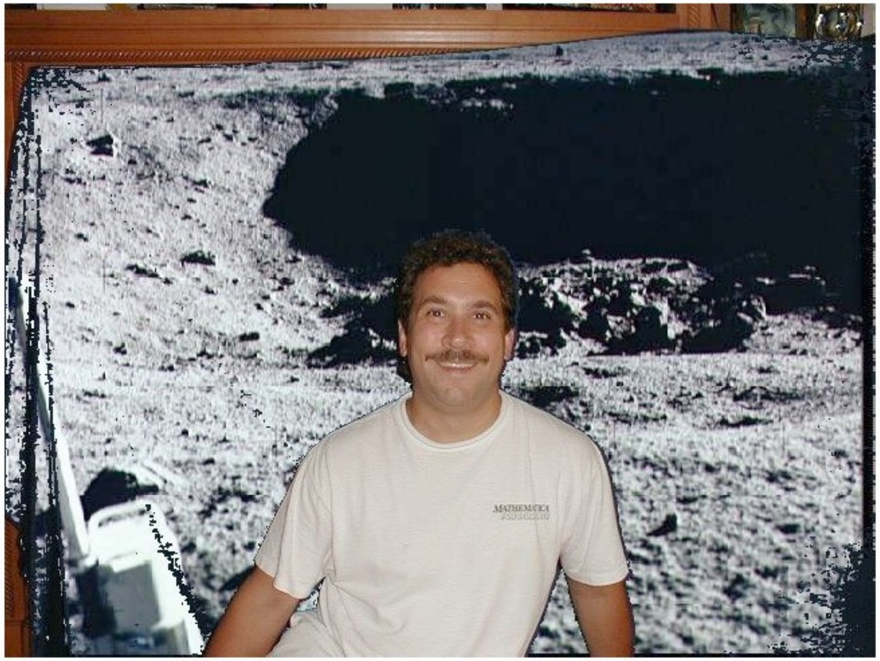
Chromakey recipe

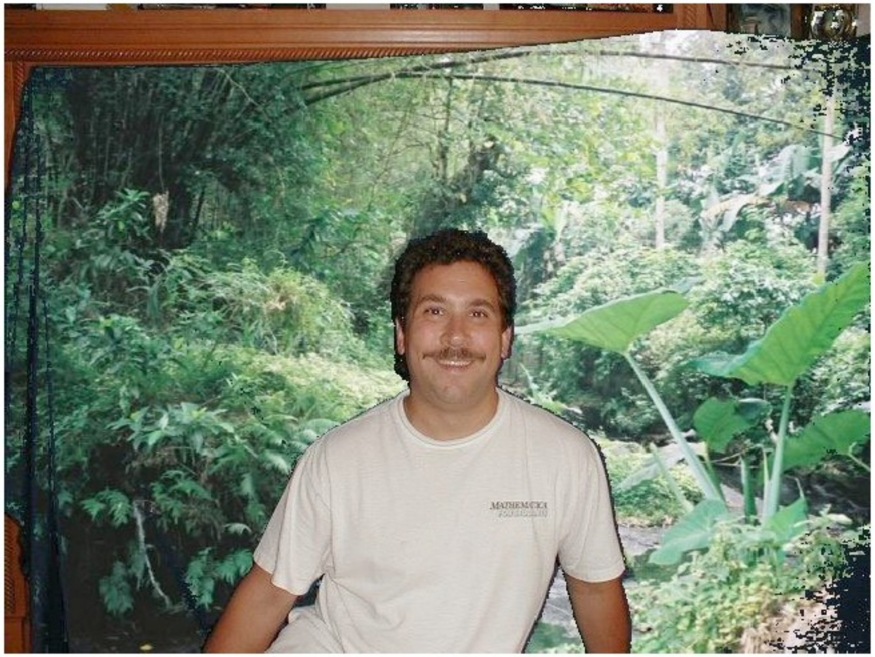
```
def chromakey(source,bg):
  # source should have something in front of blue, bg is the new background
  for x in range(1,getWidth(source)):
    for y in range(1,getHeight(source)):
      p = getPixel(source,x,y)
      # My definition of blue: If the redness + greenness < blueness
      if (getRed(p) + getGreen(p) < getBlue(p)):
      #Then, grab the color at the same spot from the new background
        setColor(p,getColor(getPixel(bg,x,y)))
```

Can also do this with getPixels()

```
def chromakey2(source,bg):
  # source should have something in front of blue,
  # bg is the new background
  for p in getPixels(source):
      # My definition of blue: If the redness + greenness < blueness
      if (getRed(p) + getGreen(p) < getBlue(p)):
      #Then, grab the color at the same spot from the new background
        setColor(p,getColor(getPixel(bg,getX(p),getY(p))))
```

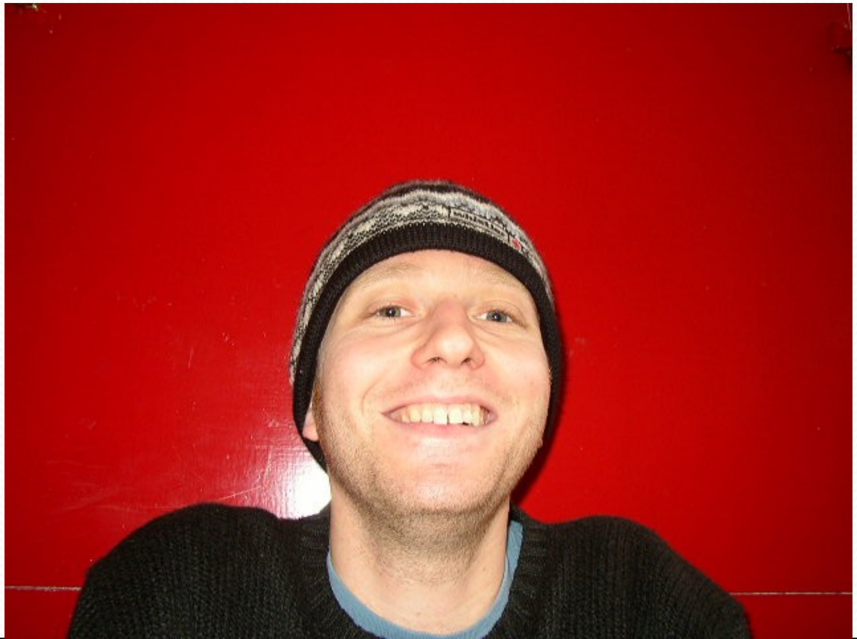
Example results



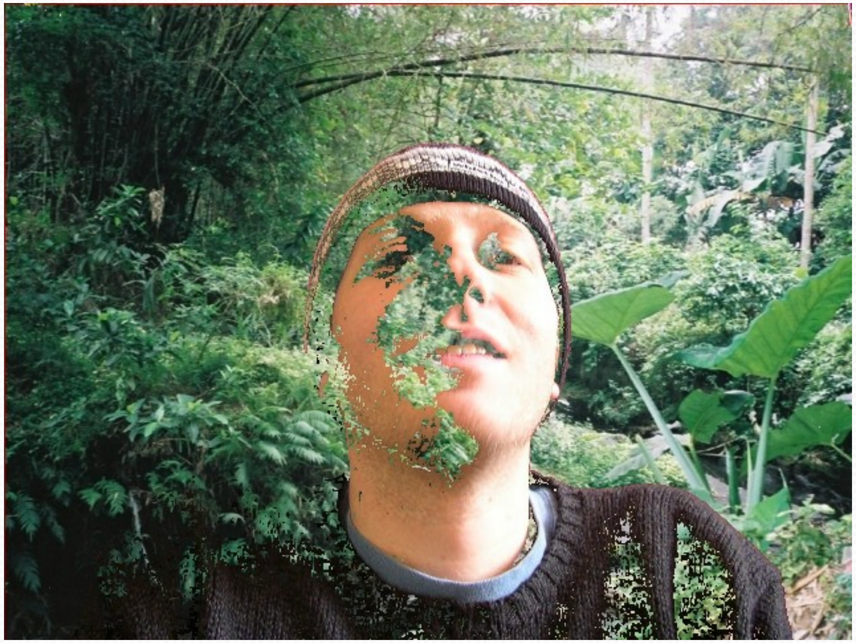


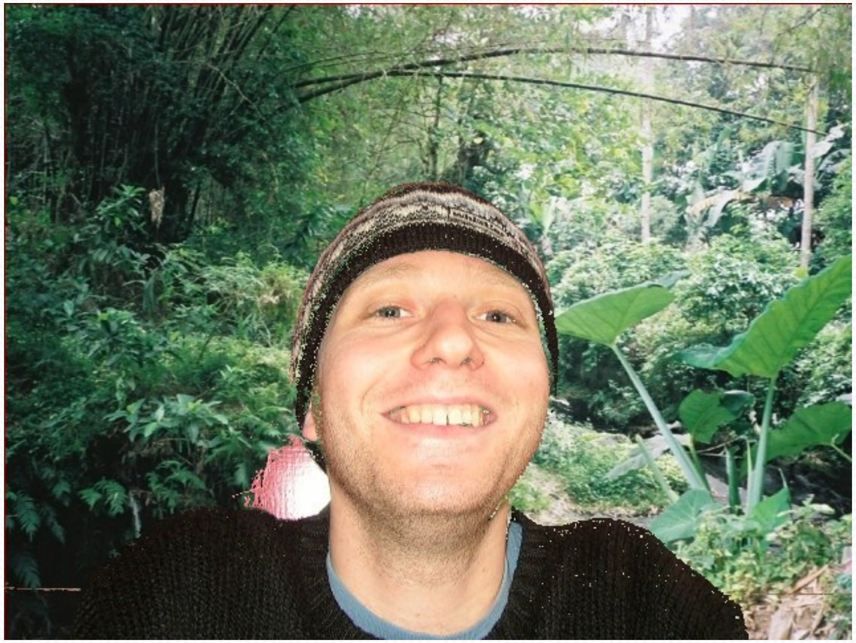
Just trying the obvious thing for Red

```
def chromakey2(source,bg):
  # source should have something in front of red, bg is the new background
  for p in getPixels(source):
    if getRed(p) > (getGreen(p) + getBlue(p)):
    #Then, grab the color at the same spot from the new background
      setColor(p,getColor(getPixel(bg,getX(p),getY(p))))
```



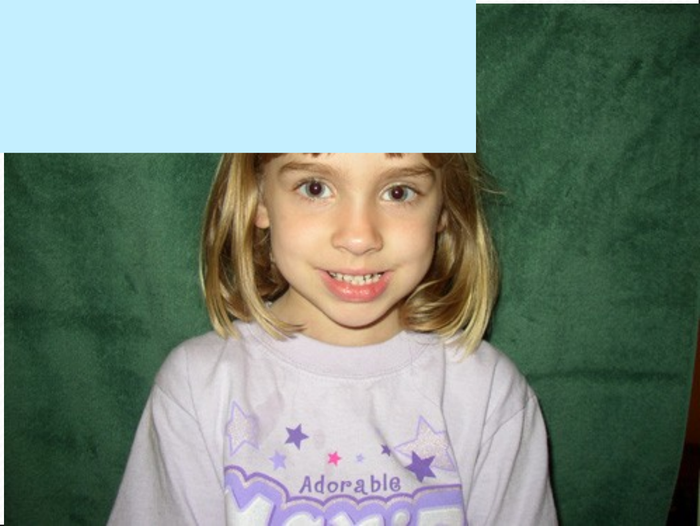
Doesn’t always work as you expect



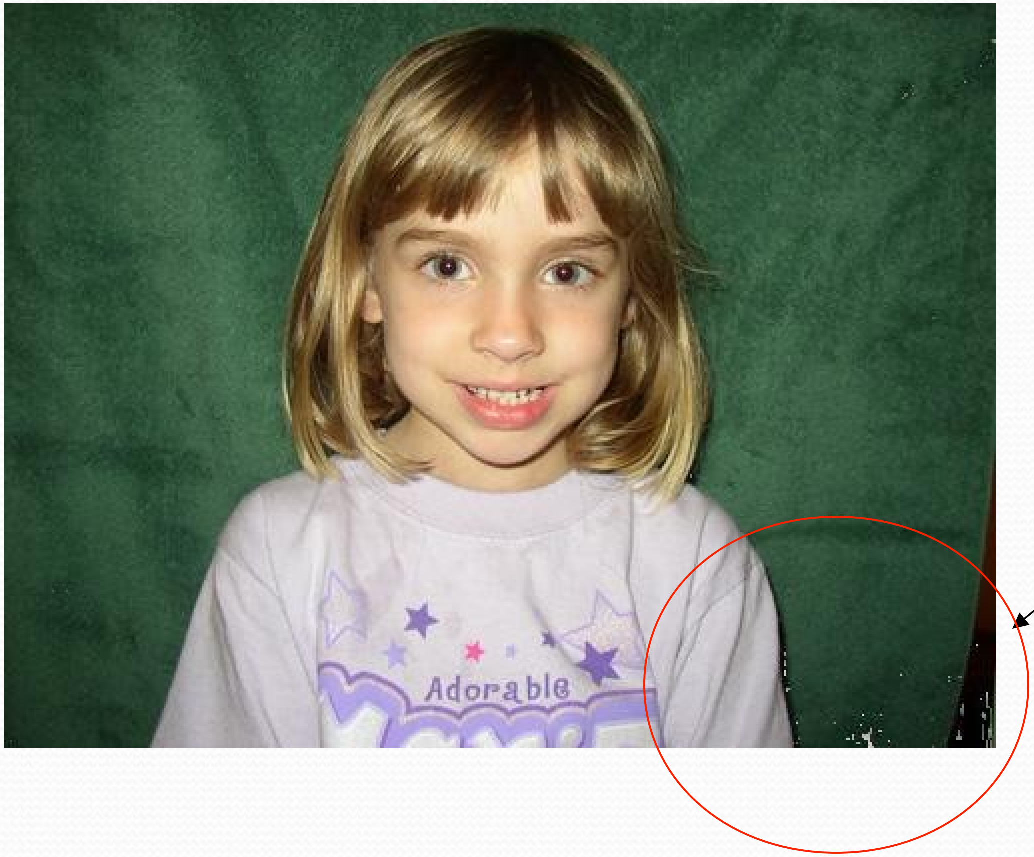


Let’s try that with green

```
def chromakeyGreen(source,bg):
  # source should have something in front of green, bg is the new background
  for x in range(1,getWidth(source)):
    for y in range(1,getHeight(source)):
      p = getPixel(source,x,y)
      # My definition of green: If the greenness > redness + blueness
      if getGreen(p) > getBlue(p) + getRed(p):
      #Then, grab the color at the same spot from the new background
        setColor(p,getColor(getPixel(bg,x,y)))
```



The same definition of green doesn't work



Changes only a few pixels

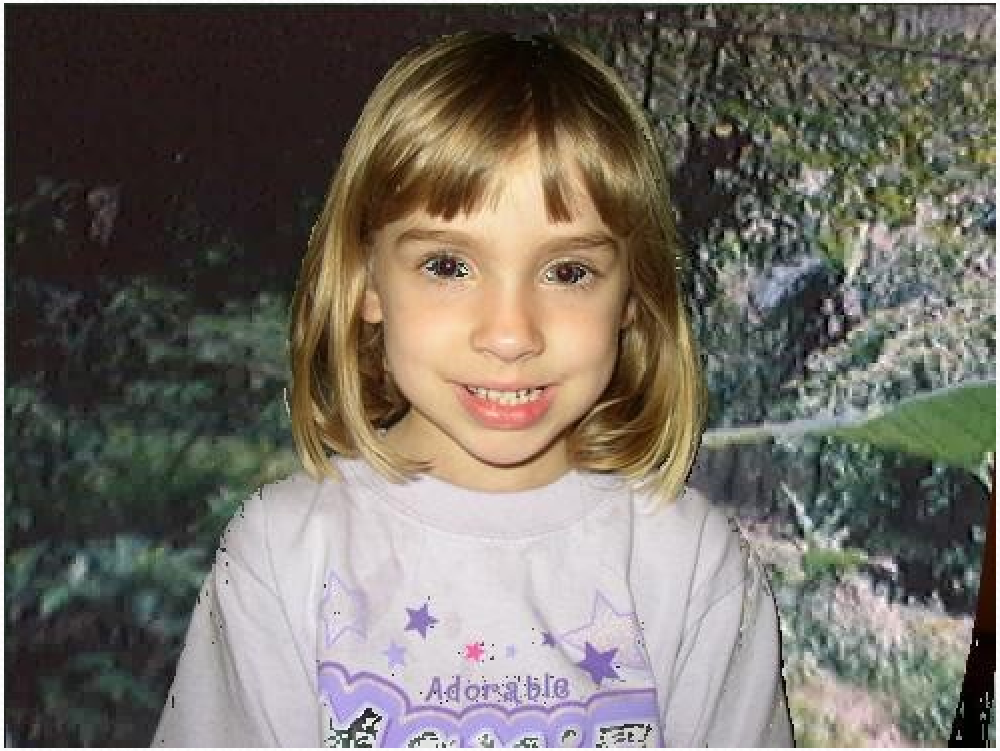
What happened?

- The towel isn’t *just* green
  - The green of the towel has lots of blue and red in it.
- Use MediaTools to figure out a new rule that makes sense.

Tweaking Chromakey

```
def chromakeyGreen(source,bg):
  # source should have something in front of green, bg is the new background
  for x in range(1,getWidth(source)):
    for y in range(1,getHeight(source)):
      p = getPixel(source,x,y)
      # My definition of green: If the greenness > redness AND blueness
      if getGreen(p) > getBlue(p) and getGreen(p) > getRed(p):
      #Then, grab the color at the same spot from the new background
        setColor(p,getColor(getPixel(bg,x,y)))
```

That looks better

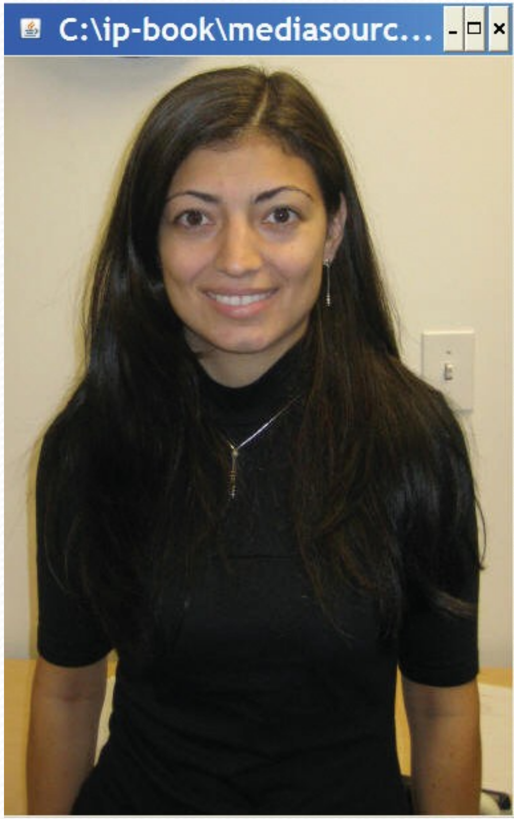


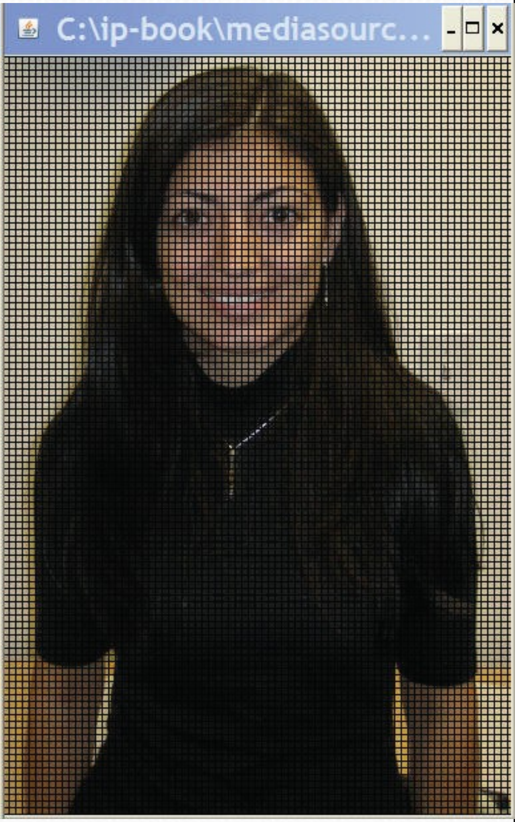
Drawing on images

- Sometimes you want to draw on pictures, to add something to the pictures.
  - Lines
  - Text
  - Circles and boxes.
- We can do that pixel by pixel, setting black and white pixels

Drawing lines on Carolina

```
def lineExample():
  img = makePicture(pickAFile())
  verticalLines(img)
  horizontalLines(img)
  show(img)
  return img

def horizontalLines(src):
  for x in range(0,getHeight(src),5):
    for y in range(0,getWidth(src)):
      setColor(getPixel(src,y,x),black)

def verticalLines(src):
  for x in range(0,getWidth(src),5):
    for y in range(0,getHeight(src)):
      setColor(getPixel(src,x,y),black)
```





We can use the color name “black” – it’s pre-defined for us.

Yes, some colors are already defined

- Colors defined for you already: **black, white, blue, red, green, gray, lightGray, darkGray, yellow, orange, pink, magenta, and cyan**

That’s tedious

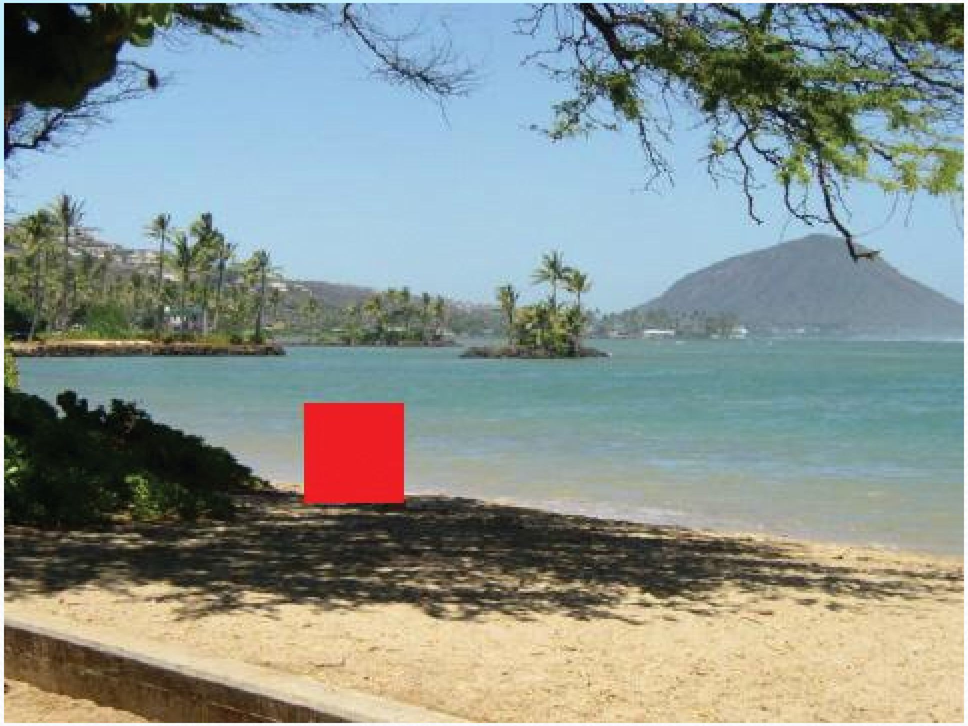
- That’s slow and tedious to set every pixel you want to make lines and text, etc.
- What you really want to do is to think in terms of your desired effect (think about “requirements” and “design”)

New functions

- **addText(pict,x,y,string)** puts the string starting at position (x,y) in the picture
- **addLine(picture,x1,y1,x2,y2)** draws a line from position (x1,y1) to (x2,y2)
- **addRect(pict,x1,y1,w,h)** draws a black rectangle (unfilled) with the upper left hand corner of (x1,y1) and a width of w and height of h
- **addRectFilled(pict,x1,y1,w,h,color)** draws a rectangle filled with the color of your choice with the upper left hand corner of (x1,y1) and a width of w and height of h

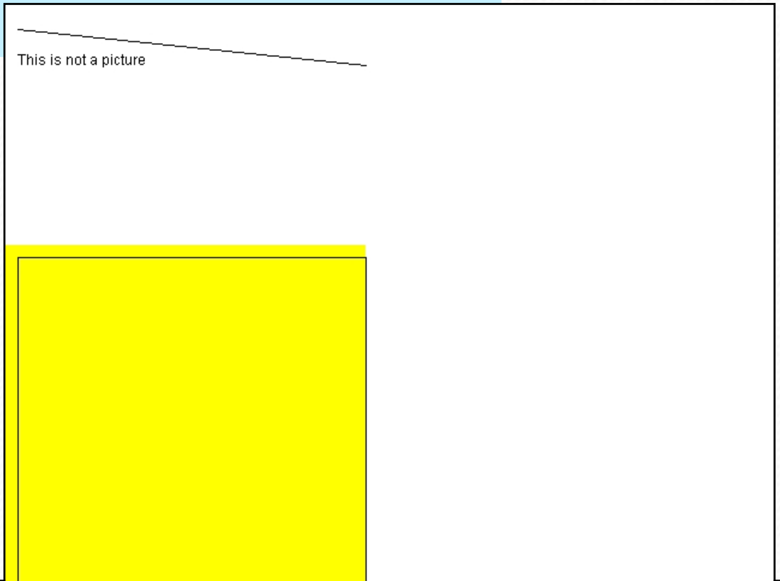
The mysterious red box on the beach

```
def addABox():
  beach = makePicture(getMediaPath("beach-smaller.jpg"))
  addRectFilled(beach,150,150,50,50,red)
  show(beach)
  return beach
```



Example picture

```
def littlepicture():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  addText(canvas,10,50,"This is not a picture")
  addLine(canvas,10,20,300,50)
  addRectFilled(canvas,0,200,300,500,yellow)
  addRect(canvas,10,210,290,490)
  return canvas
```



A thought experiment

- Look at that previous page: Which has a fewer number of bytes?
  - The program that drew the picture
  - The pixels in the picture itself.
- It’s a no-brainer
  - The program is less than 100 characters (100 bytes)
  - The picture is stored on disk at about 15,000 bytes

Vector-based vs. Bitmap Graphical representations

- Vector-based graphical representations are basically *executable programs* that generate the picture on demand.
  - Postscript, Flash, and AutoCAD use vector-based representations
- Bitmap graphical representations (like JPEG, BMP, GIF) store individual pixels or representations of those pixels.
  - JPEG and GIF are actually *compressed* representations

Vector-based representations can be smaller

- Vector-based representations can be much smaller than bit-mapped representations
  - Smaller means faster transmission (Flash and Postscript)
  - If you want all the detail of a complex picture, no, it's not.

But vector-based has more value than that

- Imagine that you’re editing a picture with lines on it.
  - If you edit a bitmap image and extend a line, it’s just more bits.
    - There’s no way to really realize that you’ve *extended* or *shrunk* the line.
  - If you edit a vector-based image, it’s possible to just *change the specification*
    - Change the numbers saying where the line is
    - Then it *really is* the same line
- That’s important when the picture drives the creation of the product, like in automatic cutting machines

How are images compressed?

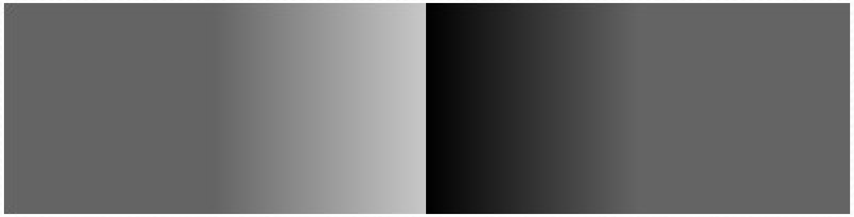
- Sometimes *lossless* using techniques like *run length encoding (RLE)*
  - Instead of this:
    B B Y Y Y Y Y Y Y Y Y B B
  - We could say “9 Y’s” like this:
    B B 9 Y B B
- *Lossy compression* (like JPEG and GIF) loses detail, some of which is invisible to the eye.

When changing the picture means changing a program…

- In a vector-based drawing package, changing the drawing is changing a *program*.
- How could we reach in and change the actual program?
- We can using *string manipulation*
  - The program is just a string of characters
  - We want to manipulate those characters, in order to manipulate the program

Example programmed graphic

- If I did this right, we perceive the left half as lighter than the right half
- In reality, the end quarters are actually the same colors.



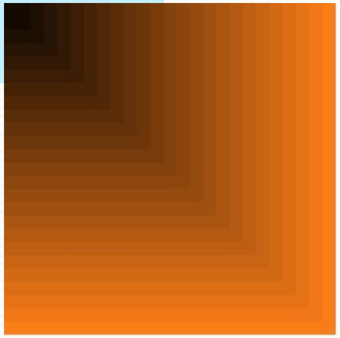
Building a programmed graphic

```
def greyEffect():
  file = getMediaPath("640x480.jpg")
  pic = makePicture(file)
  # First, 100 columns of 100-grey
  grey = makeColor(100,100,100)
  for x in range(1,100):
    for y in range(1,100):
      setColor(getPixel(pic,x,y),grey)
  # Second, 100 columns of increasing greyness
  greyLevel = 100
  for x in range(100,200):
    grey = makeColor(greyLevel, greyLevel,
greyLevel)
    for y in range(1,100):
      setColor(getPixel(pic,x,y),grey)
    greyLevel = greyLevel + 1
```

```
# Third, 100 colums of increasing greyness, from
0
  greyLevel = 0
  for x in range(200,300):
    grey = makeColor(greyLevel, greyLevel,
greyLevel)
    for y in range(1,100):
      setColor(getPixel(pic,x,y),grey)
    greyLevel = greyLevel + 1
  # Finally, 100 columns of 100-grey
  grey = makeColor(100,100,100)
  for x in range(300,400):
    for y in range(1,100):
      setColor(getPixel(pic,x,y),grey)
  return pic
```

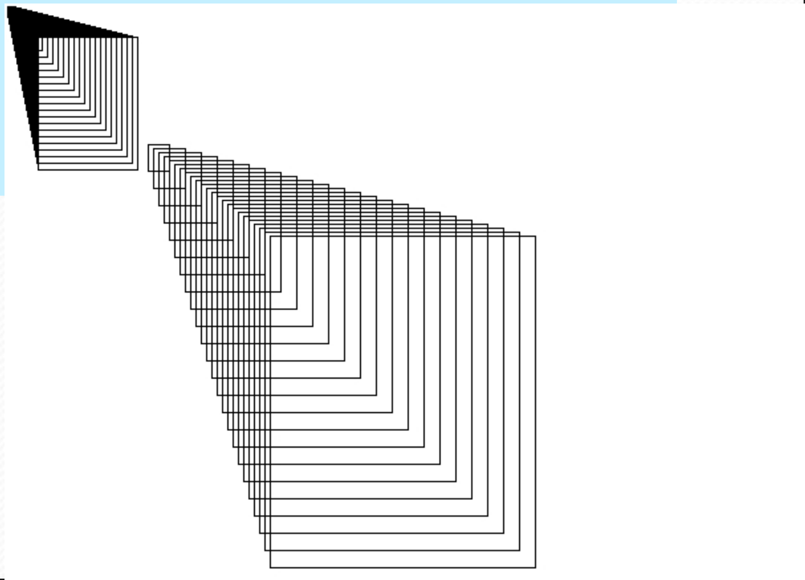
Another Programmed Graphic

```
def coolpic():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  for index in range(25,1,-1):
    color = makeColor(index*10,index*5,index)
    addRectFilled(canvas,0,0,index*10,index*10,color)
  show(canvas)
  return canvas
```



And another

```
def coolpic2():
  canvas=makePicture(getMediaPath("640x480.jpg"))
  for index in range(25,1,-1):
    addRect(canvas,index,index,index*3,index*4)
    addRect(canvas,100+index*4,100+index*3,index*8,index*10)
  show(canvas)
  return canvas
```



Why do we write programs?

- Could we do this in Photoshop? Maybe
  - I’m sure that you can, but you need to know how.
- Could I teach you to do this in Photoshop? Maybe
  - Might take a lot of demonstration
- But this program is an *exact* definition of the process of generating this picture
  - It works for anyone who can run the program, without knowing Photoshop

We write programs to encapsulate and communicate process

- If you can do it by hand, do it.
- If you need to teach someone else to do it, consider a program.
- If you need to explain to lots of people how to do it, definitely use a program.
- If you want lots of people to do it without having to teach them something first, definitely use a program.