CS Kickstart – Day 4

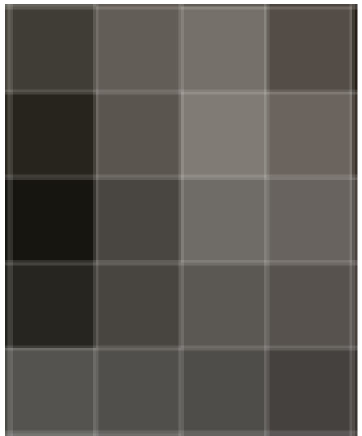
Additional Collage Techniques and Side Projects

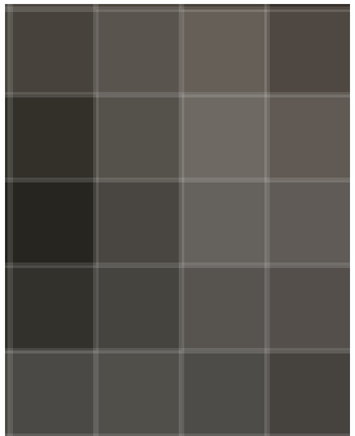
New Picture Effects

- Blurring
- Sketch effect
- Blending pictures
- Adding shapes and text

Blurring

* Can be used to reduce pixelization in scaled up images or for artistic effect
* Blur an image by averaging nearby pixels
* Blurring is close to the opposite of sketch effect

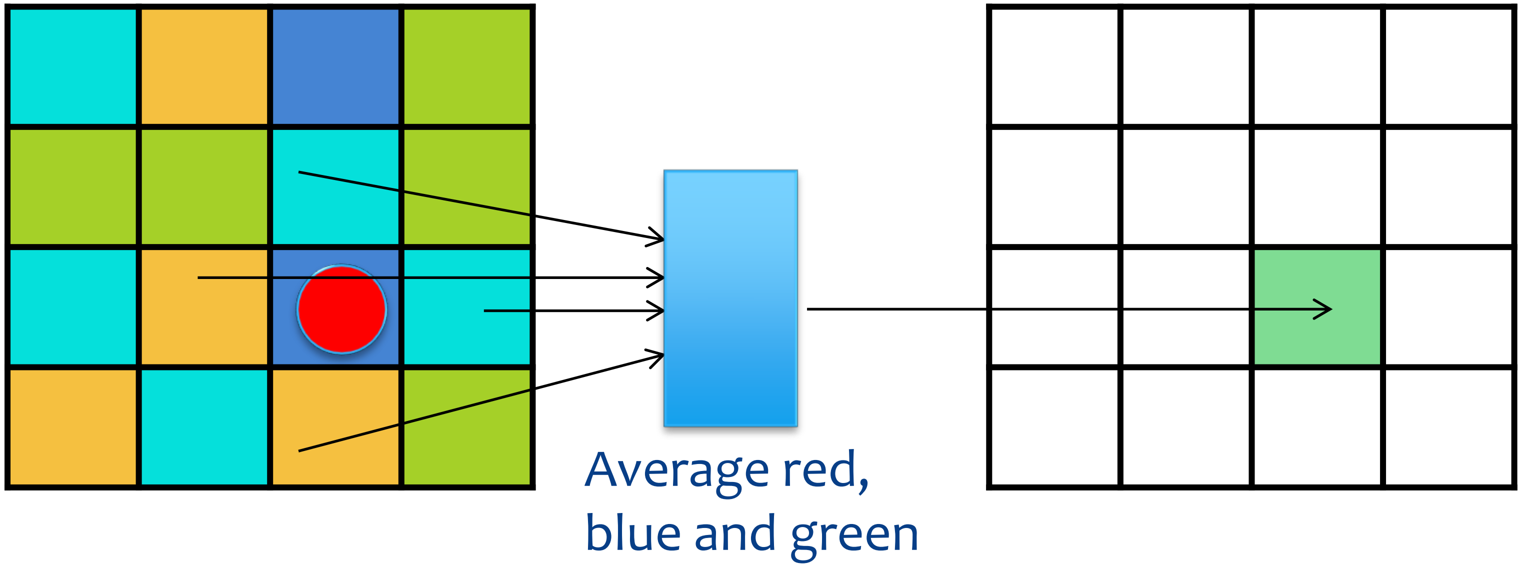




Simple Blurring Algorithm

For each pixel in the area you want to blur

set the pixel equal to the average of the surrounding pixels



Blurring Psuedocode

```
def blur(pic):
    set newPic to an empty picture
    for x in range(1, getWidth(source)-1):
        for y in range(1, getHeight(source)-1):
            1) get red for top, bottom, L, R, center pixels of pic
            2) set red of center pixel on newPic as the average
            of the pixels you found in step 1
            repeat 1, 2 for blue and green
```

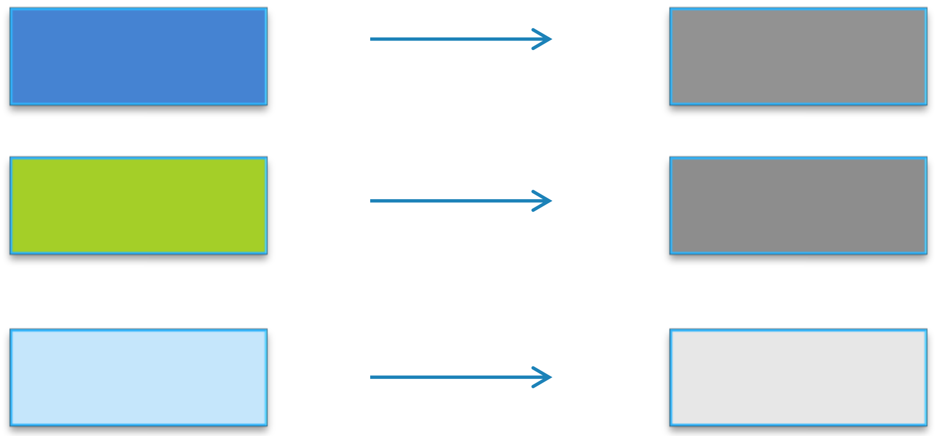
Sample Results

Edge Detection

- Blurring is averaging across pixels.
- Edge detection is looking for *differences* between pixels.
- We define *differences* as changes in *luminescence*
- If the pixel changes left-to-right, up and down, then we make our pixel black. Else white.

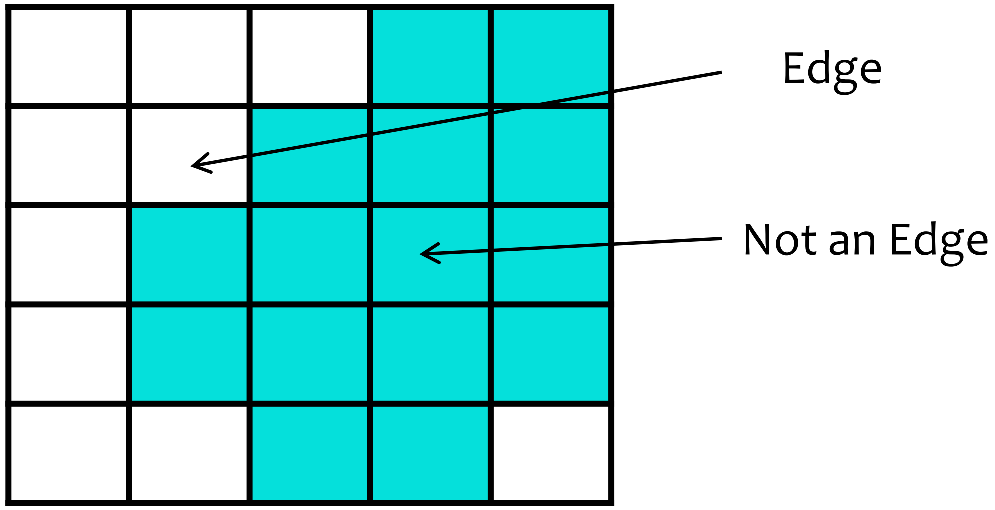
Review: Luminescence

* Luminescence = average of red, blue and green values



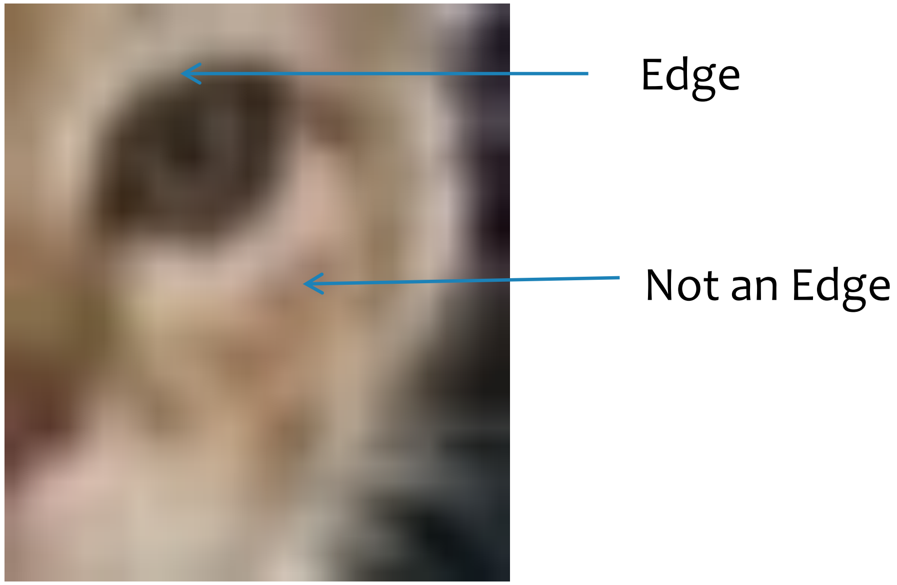
Color Luminescence

How to detect edges for sketch



A pixel is considered to be on an edge if the pixels underneath it and to the right have significantly different luminance values

How to detect edges for sketch



A pixel is considered to be on an edge if the pixels underneath it and to the right have significantly different luminance values

Sketch Effect Psuedocode

```
def sketch(origPic):
    set newPic to be an empty picture
    for x in range(1, getWidth(origPic)-1):
        for y in range(1, getHeight(origPic) – 1):
            set lumRight to the luminance of the pixel to the right of (x,y)
            set lumDown to the luminance of the pixel under (x,y)
            set lumThis to the luminance of pixel at (x,y)
```

Sketch Effect Psuedocode

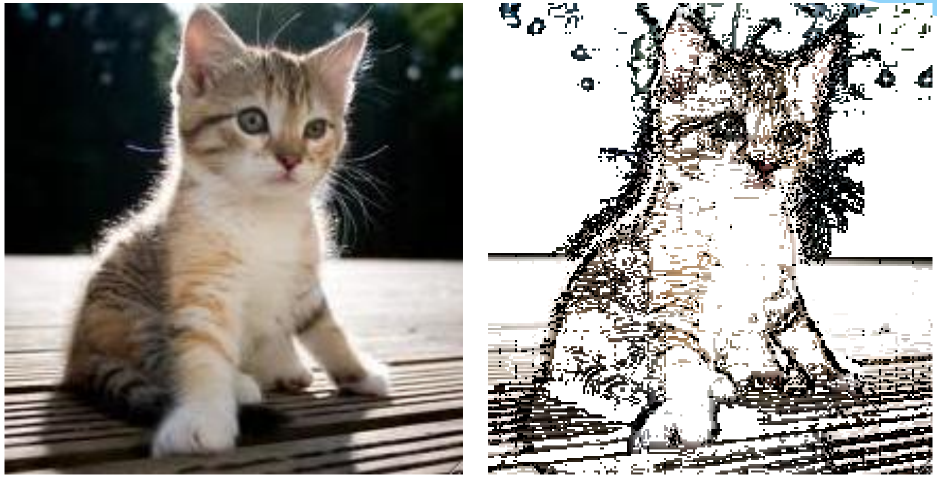
```
def sketch(origPic):
    set newPic to be an empty picture
    for x in range(1, getWidth(origPic)-1):
        for y in range(1, getHeight(origPic) – 1):
            set lumRight to the luminance of the pixel to the right of (x,y)
            set lumDown to the luminance of the pixel under (x,y)
            set lumThis to the luminance of pixel at (x,y)
```

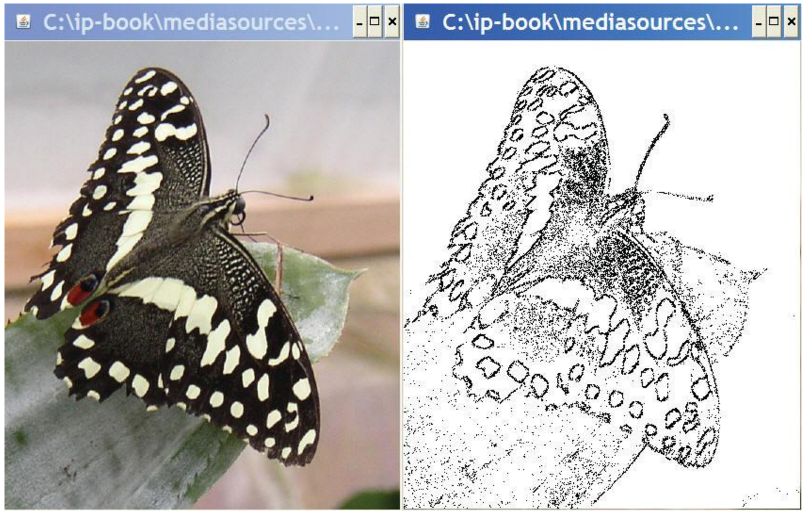
Luminance = (green + red + blue)/3

Sketch Effect Psuedocode

```
def sketch(origPic):
    set newPic to be an empty picture
    for x in range(1, getWidth(origPic)-1):
        for y in range(1, getHeight(origPic) – 1):
            set lumRight to the luminance of the pixel to the right of (x,y)
            set lumDown to the luminance of the pixel under (x,y)
            set lumThis to the luminance of pixel at (x,y)
            if lumThis is significantly different from lumDown & lumRight:
                pixel (x,y) in newPic is black
            else:
                pixel (x,y) in newPic is white
```

Sample Results

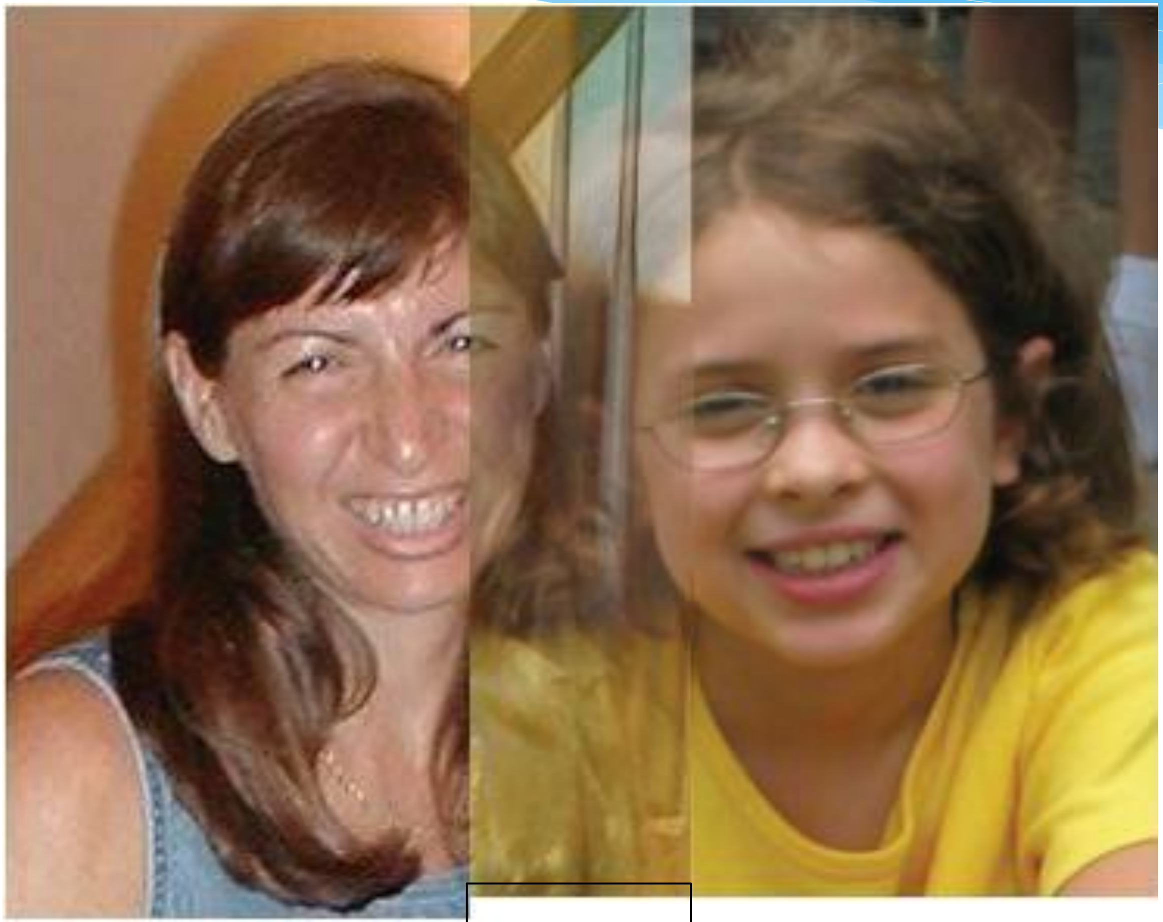




Blending pictures

- How do we make pictures blend together?
- Make pixels “transparent”
- Do this as a “weighted sum” of each color in each pixel
  - If it’s 50-50, we take 50% of red of picture1’s pixels + 50% of red of picture2’s pixels (etc)
  - Can weight sums to make one picture seem more transparent than the other with 80-20 weighting

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("640x4
 #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
```

Drawing on images

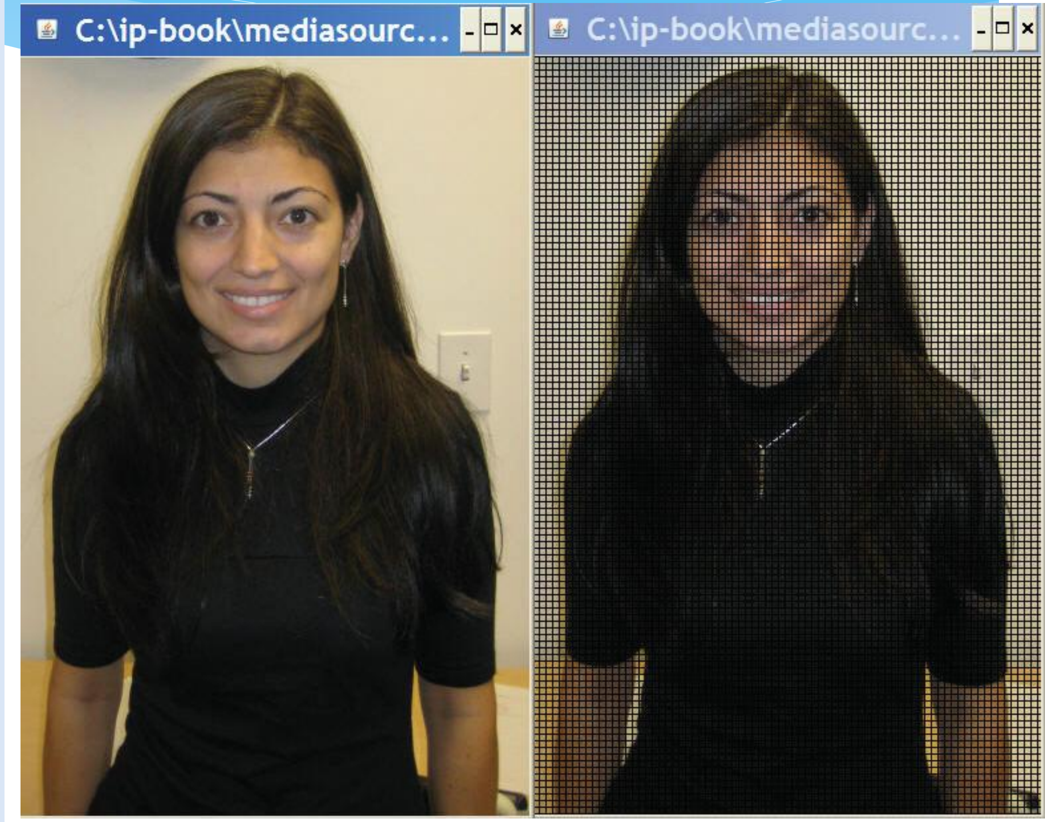
- Sometimes you want to draw on pictures, to add something to the pictures.
  - Lines
  - Text
  - Circles and boxes.
- We can do that pixel by pixel, setting black and white pixels

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

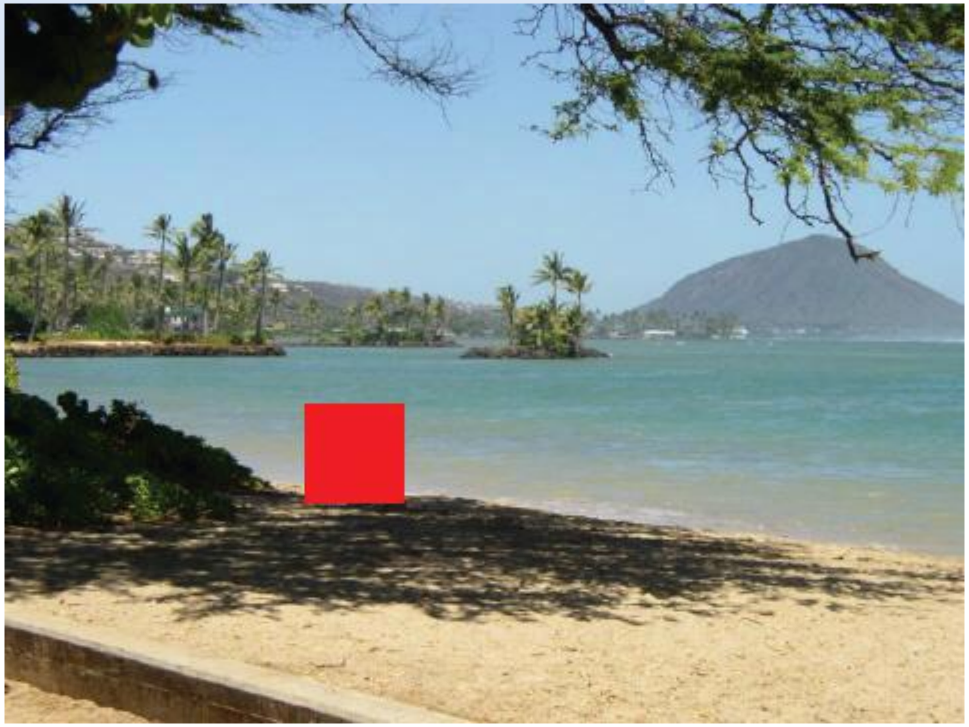
- That’s slow and tedious to set every pixel you want to make lines and text, etc.
- What you really want to do is to think in terms of your desired effect (think about “requirements” and “design”)

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

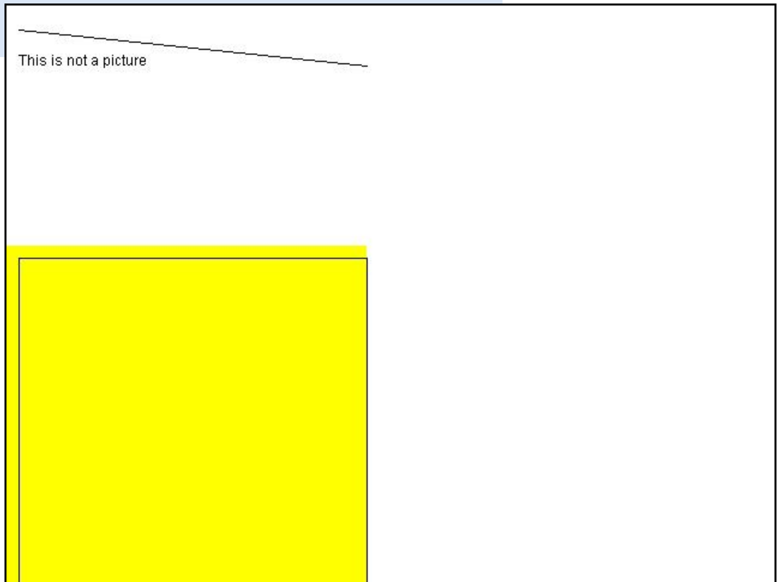
The mysterious red box on the beach

```
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?

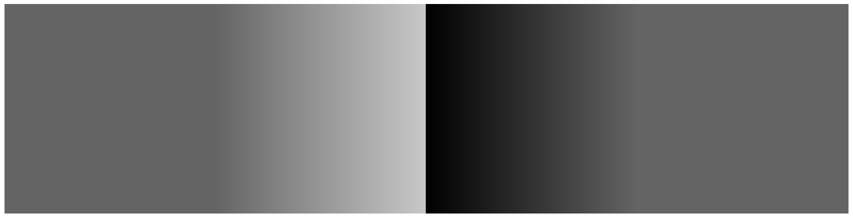
- Sometimes *lossless* using techniques like *run length encoding (RLE)*
  - Instead of this:
    B B Y Y Y Y Y Y Y Y Y B B
  - We could say “9 Y’s” like this:
    B B 9 Y B B
- *Lossy compression* (like JPEG and GIF) loses detail, some of which is invisible to the eye.

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.

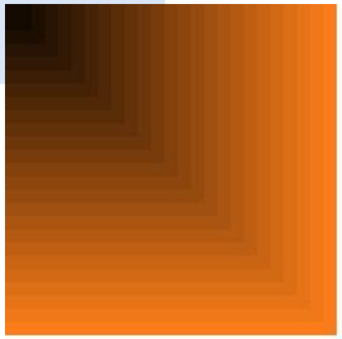


Building a programmed graphic

```
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 greynes
  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
```