Introduction to Programming

My first red-eye removal

+

What most schools don’t teach
https://www.youtube.com/watch?v=nKIu9yen5nc

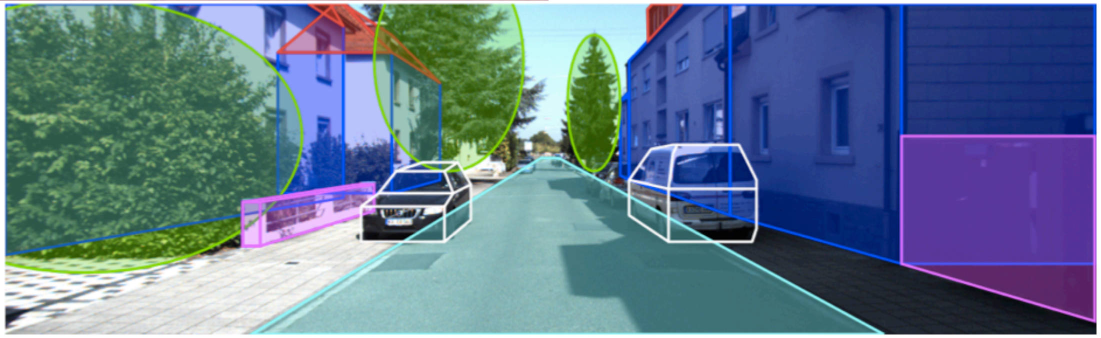
The hour of code
https://www.youtube.com/watch?v=FC5FbmsH4fw

“If 'coding' would be your superpower, what would you do?”

Computer Science

What else do we do in CS here at the University of Adelaide:

“Internet of Things”



And much more…

+
Computer Science

Computing has become pervasive, touching nearly every aspect of our lives.

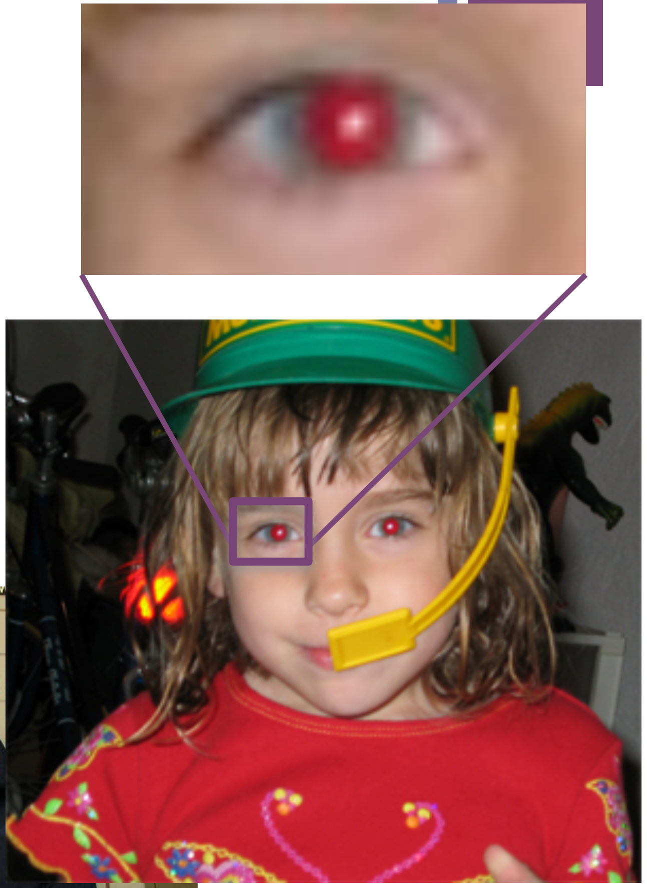
A degree in computer science can open up opportunities for employment in the software development industry, and **also** in many areas including healthcare, business, engineering, medicine, graphics, utilities, and education.

Personally:

- I have been to over 25 countries, have friends and colleagues in Italy, Norway, Germany, UK, China, Japan, USA, …
- I have been active in CS since 2003, and it has been an awesome journey so far!

+

RX1R
SONY
Carl Zeiss
Sonnar 2/35
AF
DMF
MF



Goals

- To understand the use of a matrix of pixels in representing a picture.
- To understand how to change the colour of a pixel in an image.

Our tool

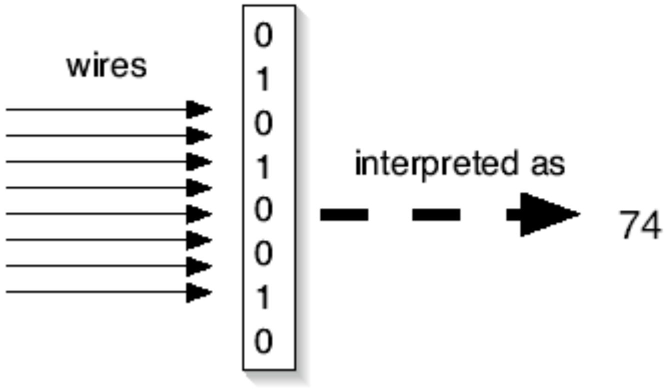
Jython Environment for Students
http://code.google.com/p/mediacomp-jes/

What computers understand

- Everything is 0’s and 1’s
- Computers are exceedingly stupid
  - The only data they understand is 0’s and 1’s
  - They can only do the most simple things with those 0’s and 1’s
    - Move this value here
    - Add, multiply, subtract, divide these values
    - Compare these values, and if one is less than the other, go follow this step rather than that one.
  - Done fast enough, those simple things can be amazing.

Key Concept: Encodings

- We can *interpret* the 0's and 1's in computer memory any way we want.
  - We can treat them as numbers.
  - We can *encode* information in those numbers
- Even the notion that the computer understands numbers is an interpretation
  - We encode the voltages on wires as 0's and 1's,
    eight of these defining a *byte*
  - Which we can, in turn, interpret as a decimal number



We perceive light differently from how it actually is

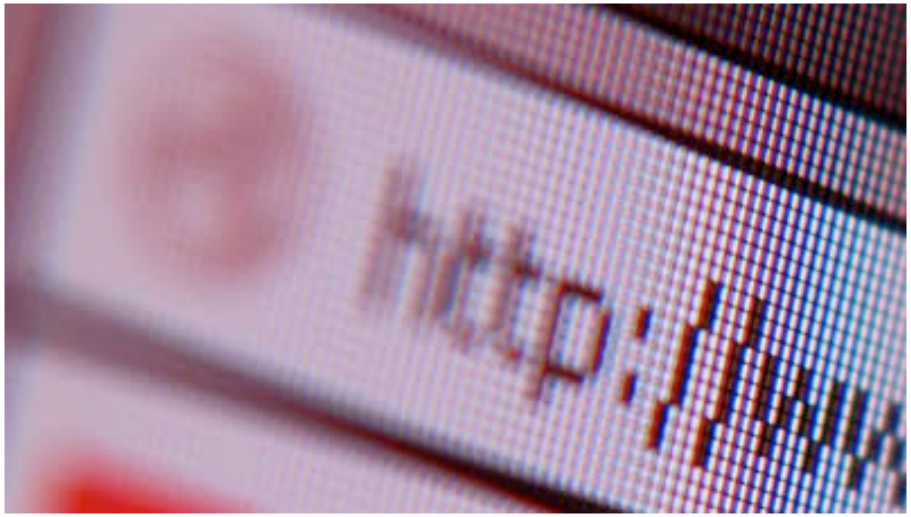
- Colour is continuous
  - Visible light is in the wavelengths between 370 and 730 nanometers
    - That' s 0.00000037 and 0.00000073 meters

- But we *perceive* light with colour sensors that peak around 425 nm (blue), 550 nm (green), and 560 nm (red).
    - Our brain figures out which colour is which by figuring out how much of each kind of sensor is responding
    - Dogs have only two kinds of sensors
      - They *do* see colour. Just *less* colour.

+

Digitising pictures as bunches of little dots

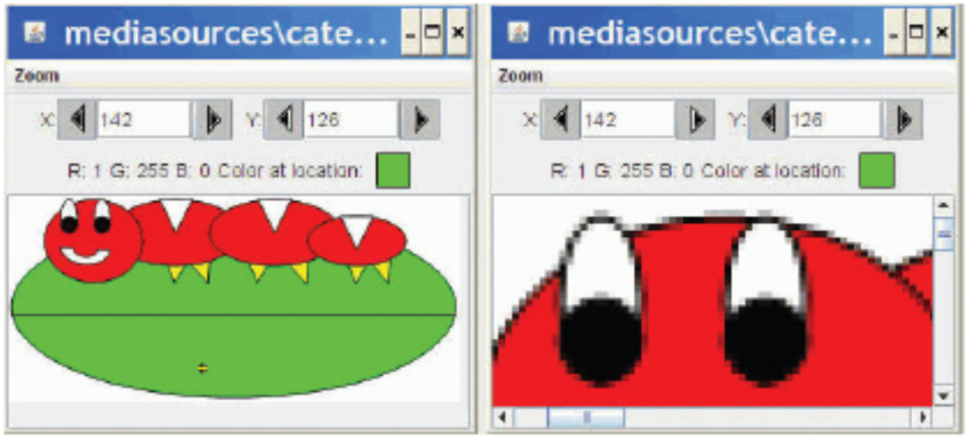
- We digitise pictures into lots of little dots
- Enough dots and it looks like a continuous whole to our eye
  - Our eye has limited resolution
- Each picture element is referred to as a *pixel*



Pixels

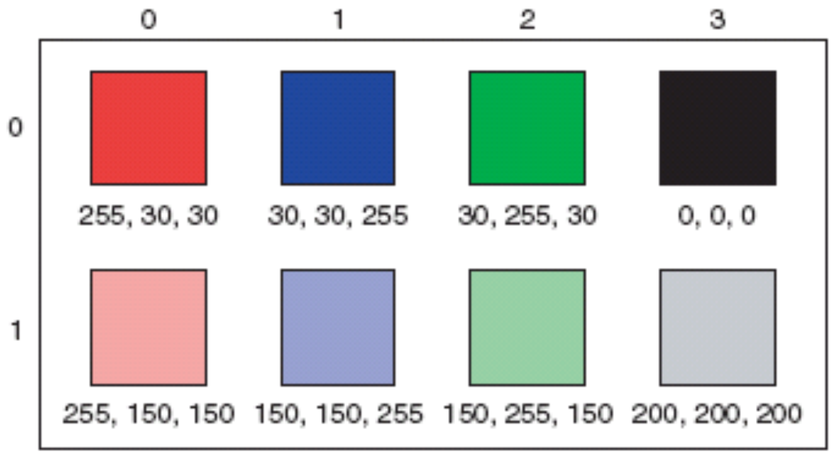
- **Pixels are *picture elements***
  - Each pixel object (in JES) knows its *colour* and *position*
  - A picture is a *matrix* of pixels

```
>>> file = '/users/generic/mathsci/caterpillar.jpg'
>>> pict = makePicture(file)
>>> explore(pict)
```



When we zoom the picture to 500%, we can see individual pixels.

Encoding Colour

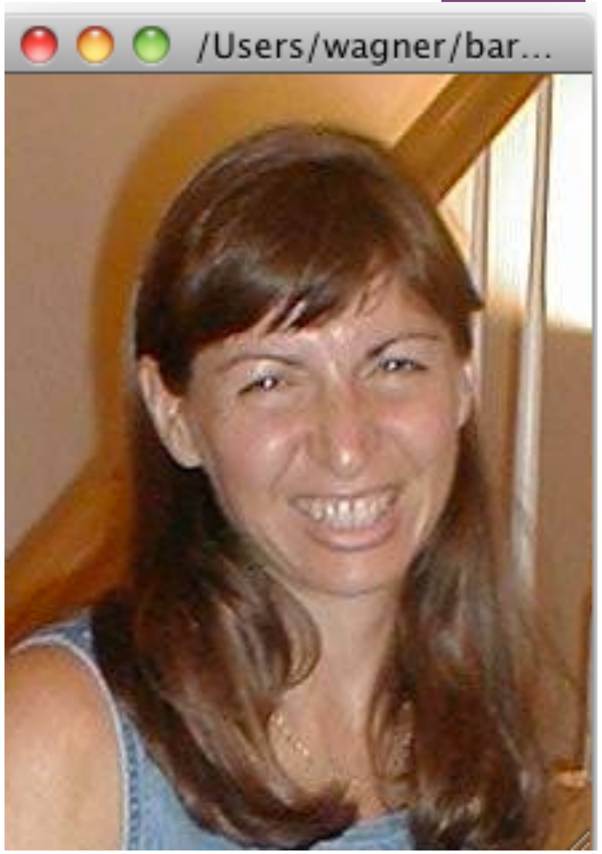


- Each pixel encodes colour at that position in the picture
- Each component colour (red, green, and blue) is encoded as a single byte
- Colours go from (0,0,0) to (255,255,255)
  - If all three components are the same, the colour is in greyscale
    - (200,200,200) at (3,1)
  - (0,0,0) (at position (3,0) in example) is black
  - (255,255,255) is white
  - That's 16,777,216 (2^{24}) possible colors!

[Note: this is only one possible encoding format amongst several]

In JES: showing a picture

```
>>> file = pickAFile()
>>> print(file)
/users/generic/mathsci/barbara.jpg
>>> show(picture)
>>> print(picture)
Picture, filename /users/generic/mathsci/barbara.jpg
height 294 width 222
```



+

Manipulating pixels

getPixel(picture,x,y) gets a single pixel.

getPixels(picture) gets all of them in an array.

```
>>> pixel=getPixel(picture,0,0)
>>> print(pixel)
Pixel, color=color r=168 g=131 b=105
>>> pixels=getPixels(picture)
>>> print(pixels[0])
Pixel, color=color r=168 g=131 b=105
```

+

What can we do with a pixel?

- getRed, getGreen, and getBlue are functions that take a pixel as input and return a value between 0 and 255

- setRed, setGreen, and setBlue are functions that take a pixel as input *and* a value between 0 and 255

Similarly for "colours" (a pixel has a location and a colour):
- setColour, getColour, makeColour, ...

Let us see this in an example...

We can change pixels directly…

```
>>> file='/users/generic/mathsci/barbara.jpg'
>>> pict=makePicture(file)
>>> show(pict)
>>> setColor(getPixel(pict,10,100),yellow)
>>> setColor(getPixel(pict,11,100),yellow)
>>> setColor(getPixel(pict,12,100),yellow)
>>> setColor(getPixel(pict,13,100),yellow)
>>> repaint(pict)
```

But that's *really* dull and boring to change each pixel at a time…
Isn't there a better way?

+

Use a loop!
Our first picture recipe

```
def decreaseRed(picture):
  for p in getPixels(picture):
    value=getRed(p)
    setRed(p,value*0.5)
```

Used like this:

```
>>> file='/users/generic/mathsci/barbara.jpg'
>>> picture=makePicture(file)
>>> show(picture)
>>> decreaseRed(picture)
>>> repaint(picture)
```

Copy & paste from
http://tinyurl.com/unitech2015

Our first picture recipe works for *any* picture

```
def decreaseRed(picture):
  for p in getPixels(picture):
    value=getRed(p)
    setRed(p,value*0.5)
```

Used like this:

```
>>> file='/users/generic/mathsci/katie.jpg'
>>> picture=makePicture(file)
>>> show(picture)
>>> decreaseRed(picture)
>>> repaint(picture)
```

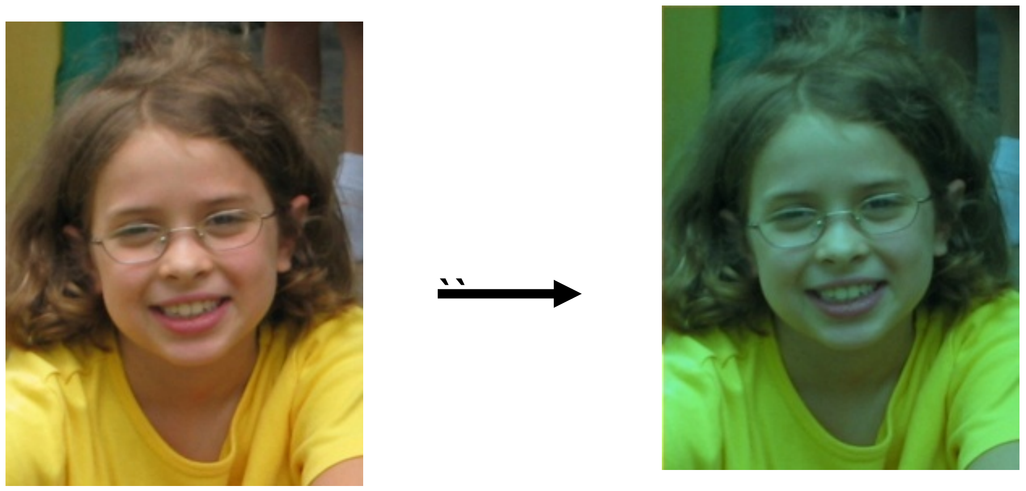
+
How do you make an omelet?

- Something to do with eggs…
- What do you do with each of the eggs?
- And then what do you do?

All useful recipes involve repetition
- Take <u>four</u> eggs and crack <u>them</u>….
- Beat the eggs <u>until</u>…

We need these repetition ("iteration") constructs in computer algorithms, too!

+ Decreasing the red in a picture



- Recipe: To decrease the red
- Ingredients: One picture, name it **pict**
- Step 1: Get all the pixels of **pict**. For each pixel **p** in the set of pixels…
- Step 2: Get the value of the red of pixel **p**, and set it to 50% of its original value

+

Use a for loop!
Our first picture recipe

```
def decreaseRed(picture):
 for p in getPixels(picture):
   value = getRed(p)
   setRed(p, value * 0.5)
```

The loop.

Note the indentation!

How for loops are written

```
def decreaseRed(picture):
  for p in getPixels(picture):
    value = getRed(p)
    setRed(p, value * 0.5)
```

- **for** is the name of the command
- An *index variable* is used to hold each of the different values of a sequence
- The word **in**
- A function that generates a *sequence*
  - **The index variable will be the name for one value in the sequence, each time through the loop (the fact that we use getPixels should suffice here)**
- A colon (":")
- And a *block* (the indented lines of code)

+

What happens when a for loop is executed

- The *index variable* is set to an item in the *sequence*
- The block is executed
  - The variable is often used inside the block
- Then execution *loops* to the **for** statement, where the index variable gets set to the next item in the sequence
- Repeat until every value in the sequence was used.

Let's walk that through slowly…

```
def decreaseRed(picture):
  for p in getPixels(picture):
    originalRed = getRed(p)
    setRed(p, originalRed * 0.5)
```

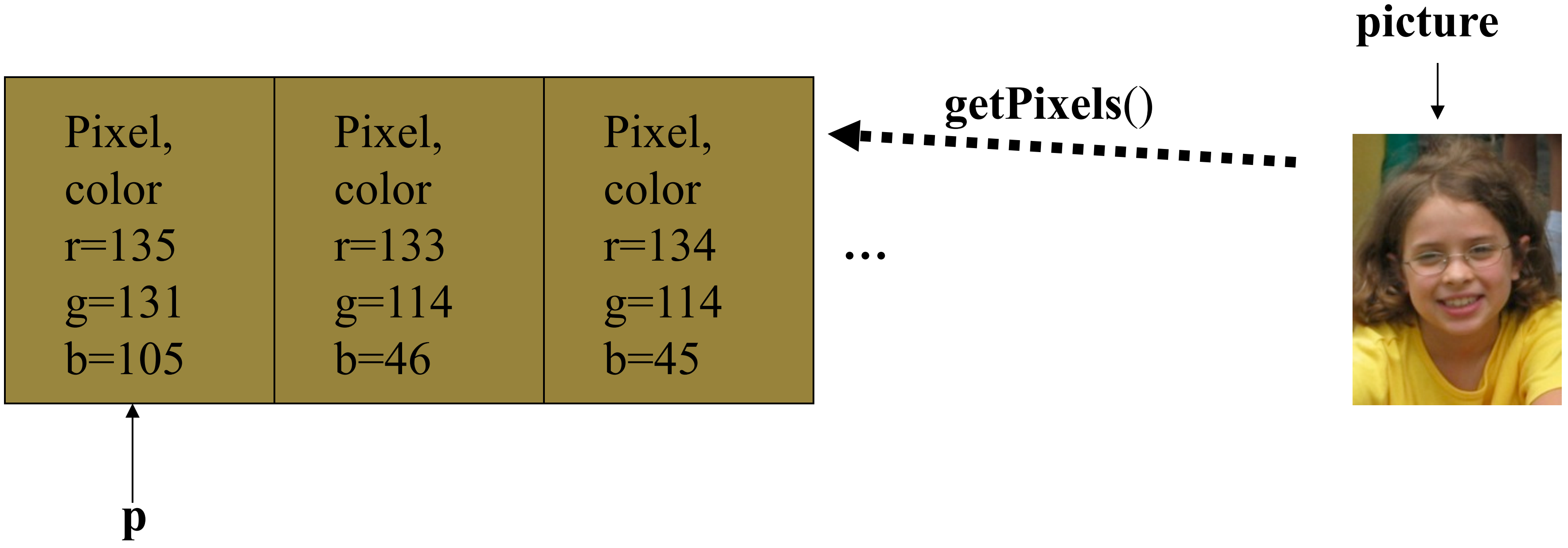
Here we take a picture object in as a parameter to the function and call it **picture**

picture

Now, get the pixels

```
def decreaseRed(picture):
  for p in getPixels(picture):
    originalRed = getRed(p)
    setRed(p, originalRed * 0.5)
```

We get all the pixels from the **picture**, then make **p** be the name of each pixel *one at a time*

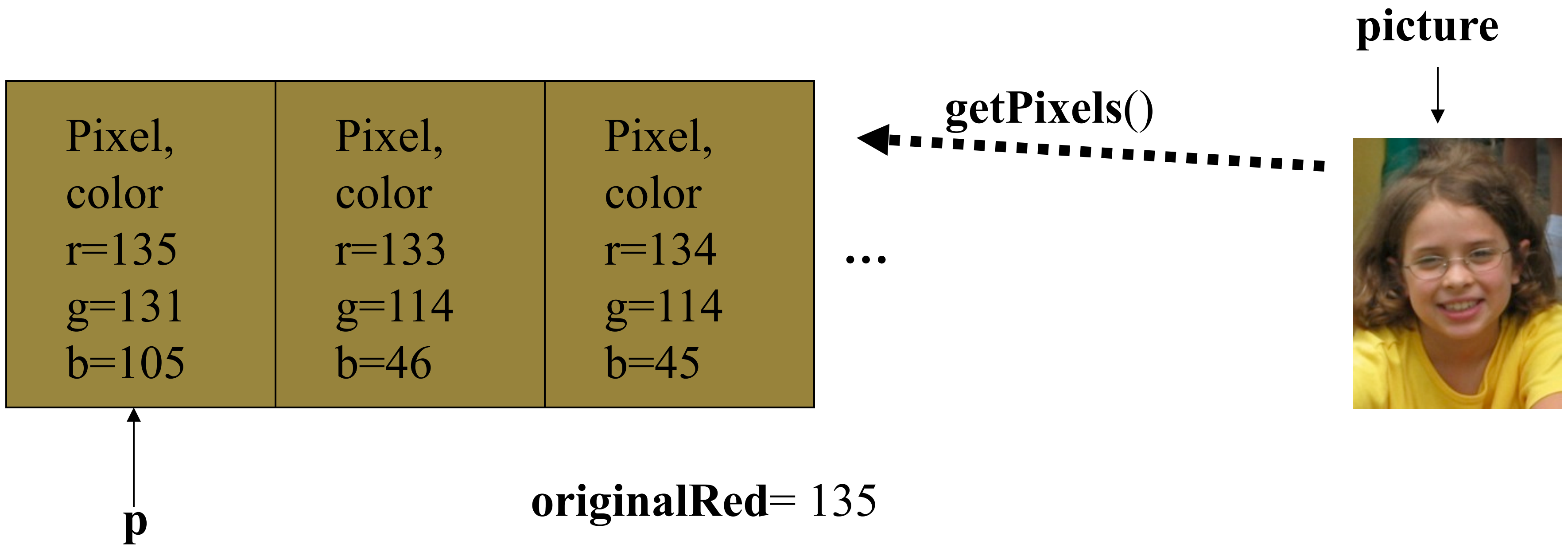


picture
getPixels()
Pixel, color r=135 g=131 b=105
Pixel, color r=133 g=114 b=46
Pixel, color r=134 g=114 b=45
…
p

Now, get the red value from pixel p

```
def decreaseRed(picture):
  for p in getPixels(picture):
    originalRed = getRed(p)
    setRed(p, originalRed * 0.5)
```

We get the red value of pixel **p** and name it **originalRed**

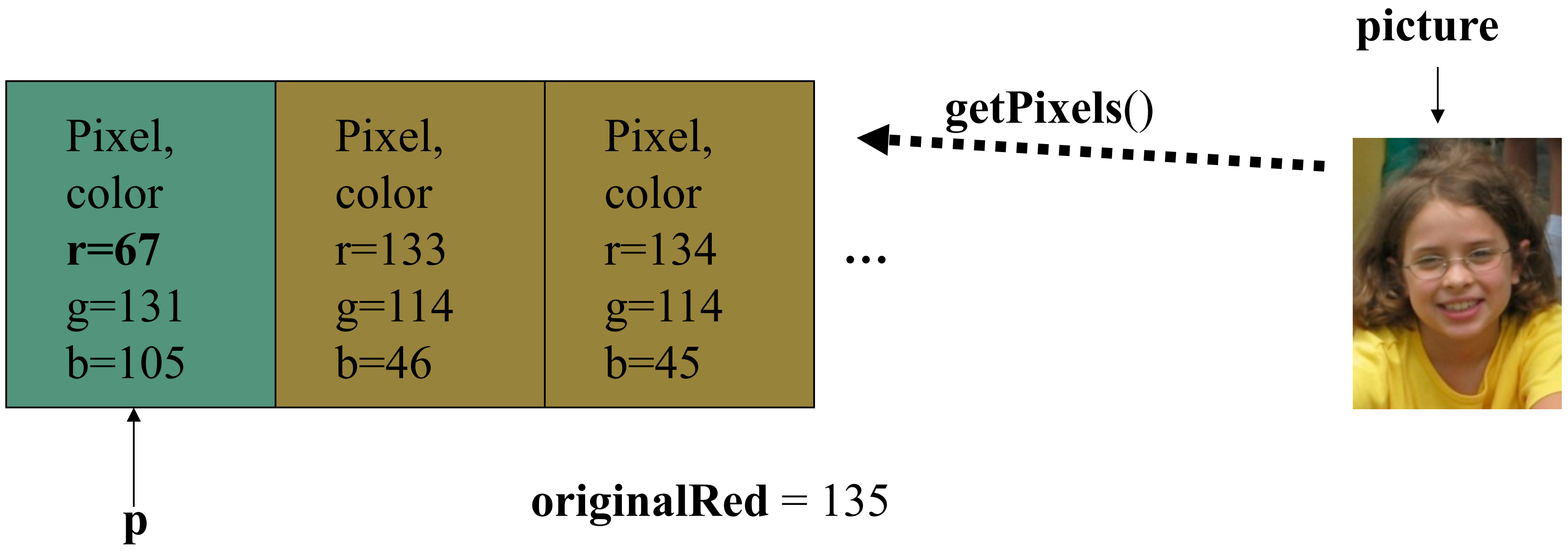


picture
getPixels()
Pixel, color r=135 g=131 b=105
Pixel, color r=133 g=114 b=46
Pixel, color r=134 g=114 b=45
…
p
originalRed= 135

Now change the pixel

```
def decreaseRed(picture):
    for p in getPixels(picture):
        originalRed = getRed(p)
        setRed(p, originalRed * 0.5)
```

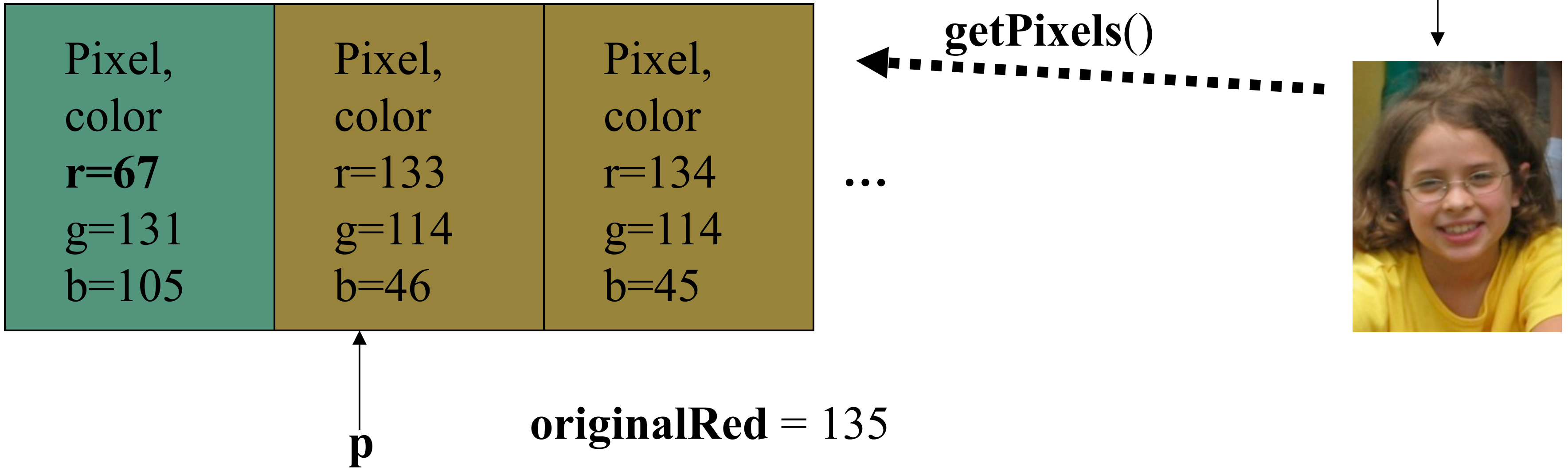
Set the red value of pixel **p** to 0.5 (50%) of **originalRed**


picture
getPixels()
Pixel, color **r=67** g=131 b=105
Pixel, color r=133 g=114 b=46
Pixel, color r=134 g=114 b=45
…
p
originalRed = 135

Then move on to the next pixel

```
def decreaseRed(picture):
  for p in getPixels(picture):
    originalRed = getRed(p)
    setRed(p, originalRed * 0.5)
```

Move on to the next pixel and name *it* **p**

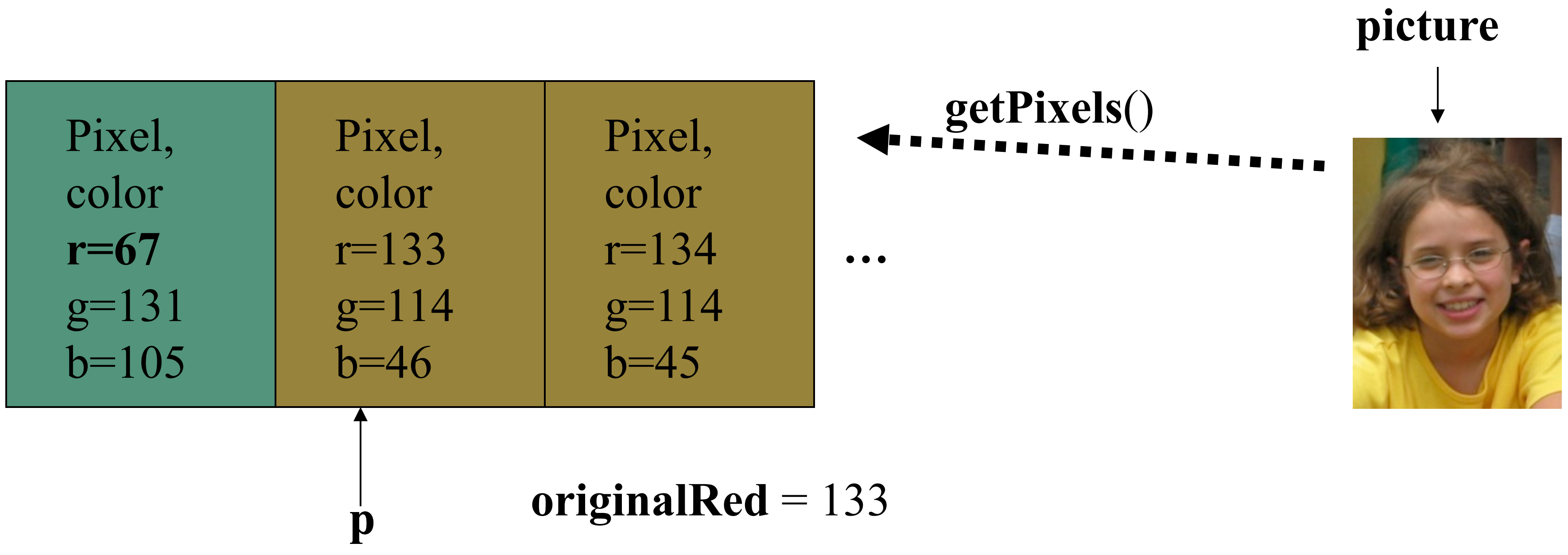


picture
getPixels()
Pixel, color r=67 g=131 b=105
Pixel, color r=133 g=114 b=46
Pixel, color r=134 g=114 b=45
…
p
originalRed = 135

Get its red value

```
def decreaseRed(picture):
    for p in getPixels(picture):
        originalRed = getRed(p)
        setRed(p, originalRed * 0.5)
```

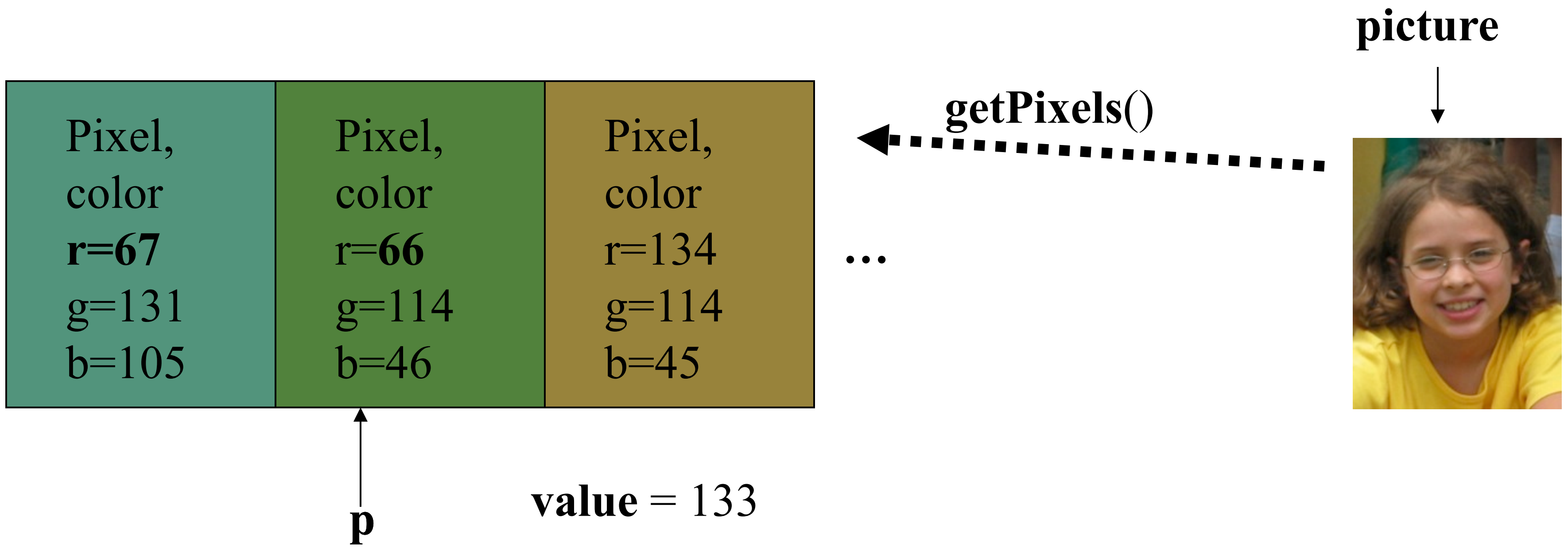
Set **originalRed** to the red value at the new **p**, then change the red at that new pixel.


picture
getPixels()
Pixel, color **r=67** g=131 b=105
Pixel, color r=133 g=114 b=46
Pixel, color r=134 g=114 b=45
…
p
originalRed = 133

And change *this* red value

```
def decreaseRed(picture):
  for p in getPixels(picture):
    originalRed = getRed(p)
    setRed(p, originalRed * 0.5)
```

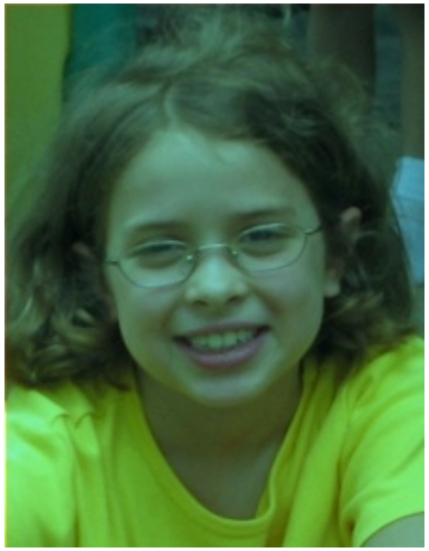
Change the red value at pixel **p** to 50% of value



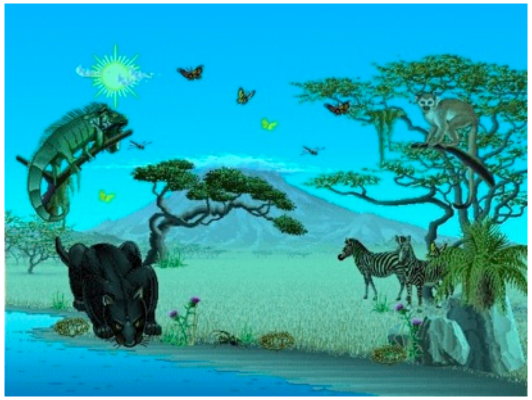
picture
getPixels()
Pixel, color **r=67** g=131 b=105
Pixel, color r=**66** g=114 b=46
Pixel, color r=134 g=114 b=45
…
p
value = 133

And eventually, we do all pixels

- We go from this… to this!



- Or from this… to this!



You can apply this one filter to many different pictures!

"Tracing/Stepping/Walking through" the program

- What we just did is called "stepping" or "walking through" the program
  - You consider each step of the program, in the order that the computer would execute it
  - You consider what *exactly* would happen
  - You write down what values each variable (name) has at each point.
- It's one of the most important *debugging* skills you can have.
  - And *everyone* has to do a *lot* of debugging, especially at first.

Read it as a Recipe

```
def decreaseRed(picture):
  for p in getPixels(picture):
    originalRed = getRed(p)
    setRed(p, originalRed * 0.5)
```

- Recipe: To decrease the red
- Ingredients: One picture, name it **pict**
- Step 1: Get all the pixels of **pict**. For each pixel **p** in the pixels…
- Step 2: Get the value of the red of pixel **p**, and set it to 50% of its original value

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]
>>> print range(3)
[0,1,2]
```

Notice:

- **End value is never included.**
  - **range(0,10) ends at 9.**
- **If you leave out a start value, it' s assumed to be zero.**

+
We can use range to generate index numbers

- We'll do this by working the range from 0 to the height-1, and 0 to the width-1.
  - Using the range function will make it easy to start from 0 and stop before the end value.
- But we'll need more than one loop.
  - Each for loop can only change one variable, and we need two for indexing 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
    - Be sure to watch your blocks (i.e., indentation) carefully!

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

What's going on here?

The first time through the first loop, x starts at 0.

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

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

Now, the inner loop

Next, we set y to 0. We' re now going to process each of the pixels in the first column.

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

+

Process a pixel

With x = 0 and y = 0, we get the upperleftmost pixel and increase its red by 10%

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

+

Next pixel

Next we set y to 1 (next value in the sequence *range (0,getHeight(picture))*

$\longrightarrow$

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

+

Process pixel (0,1)

x is still 0, and now y is 1, so increase the red for pixel (0,1)

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

We continue along this way, with y taking on every value from 0 to the height of the picture (minus 1).

+
Finally, next column

Now that we' re done with the loop for y, we get back to the FOR loop for x.
x takes on the value 1, and we go back to the y loop to process all the pixels in the column x=1.

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      value = getRed(p)
      setRed(p,value*1.1)
```

+

There are many ways…

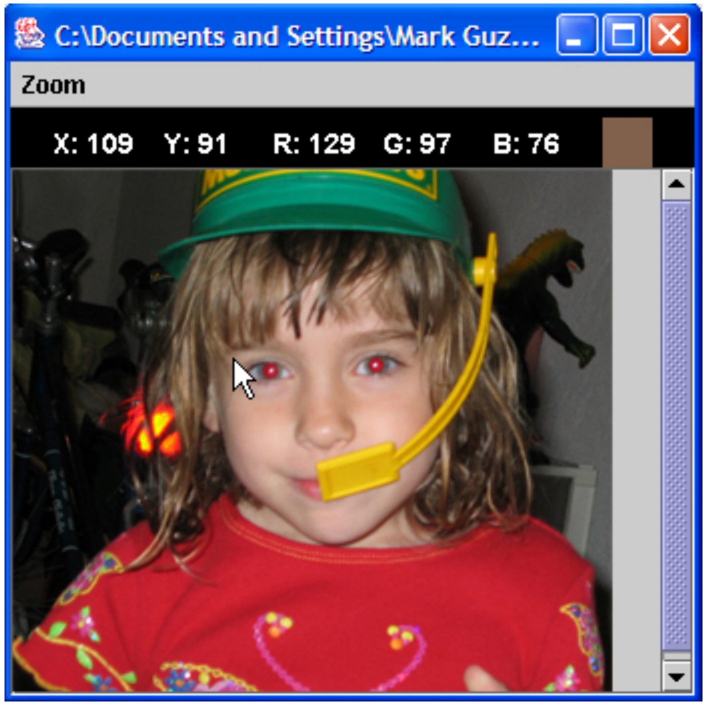
```
def decreaseRed(picture):
    for p in getPixels(picture):
       originalRed = getRed(p)
       setRed(p, originalRed * 0.5)
```

Similarly, instead of using “getPixels” we can use the “range” function:
(remember: pictures are matrices of pixels)

```
def decreaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      p = getPixel(picture,x,y)
      originalRed= getRed(p)
      setRed(p, originalRed*0.5)
```

Removing "Red Eyes"

- 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 eyes"
- We can replace the "red" with a color of our choosing.
- First, we figure out *where* the eyes are (x,y) using the JES MediaTools *(hint: pickAFile and then makePicture)*


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

jenny.jpg

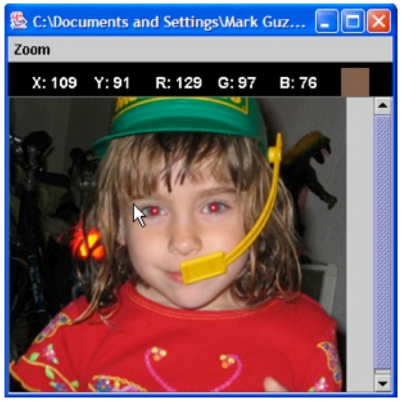
Removing Red Eye

```
def removeRedEye(picture,startX,startY,endX,endY,replacementcolor):
  red = makeColor(255,0,0)
  for x in range(startX,endX):
    for y in range(startY,endY):
      currentPixel = getPixel(picture,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**


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

Distance between colors?

- Sometimes you need to, e.g., when deciding if something is a "close enough" match
- How do we measure distance?
  - Pretend it is the Cartesian coordinate system
  - Distance between two points:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Distance between two colors:

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

Fortunately, the distance function is already implemented (see previous slide)!

What would happen if we just did getPixels() here?

- QUESTION: Why not process all pixels the same to remove redeye?

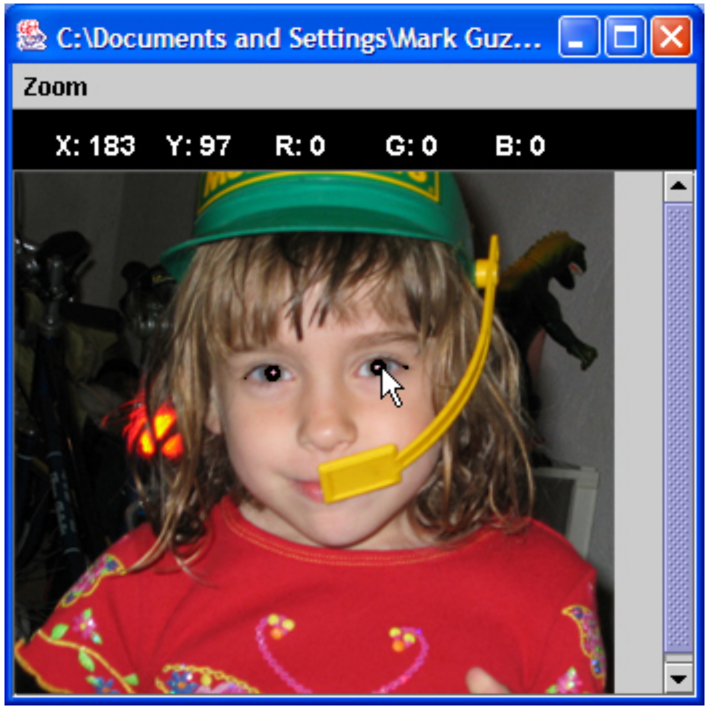
1. We would remove the red in her dress
2. The whole picture would go red
3. The whole picture would go black
4. We would probably miss her eyes

- ANSWER: Just go back a couple of slides ;)

"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


C:\Documents and Settings\Mark Guz...
Zoom
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 – and on how much you have developed your superpower! ☺

+

That's all folks!

- For more information visit
  http://cs.adelaide.edu.au/
- More about Computing and computing Careers, from IEEE
  http://www.trycomputing.org/discover
- NCSS challenge - learn python online at the National CS School, $20 registration, several weeks in August
  https://groklearning.com/challenge/

These slides are available online: http://cs.adelaide.edu.au/~markus/teaching/outreach.html