



# COMP2720: Automating Tools for New Media

Programming picture  
manipulations: Using loops



# Learning objectives for this lecture

- How to use the JES MediaTools to check RGB values
- How to get and set colours for pixels
- Using loops to manipulate all pixels in a picture
- Tracing/stepping through programs

# We can change pixels directly...

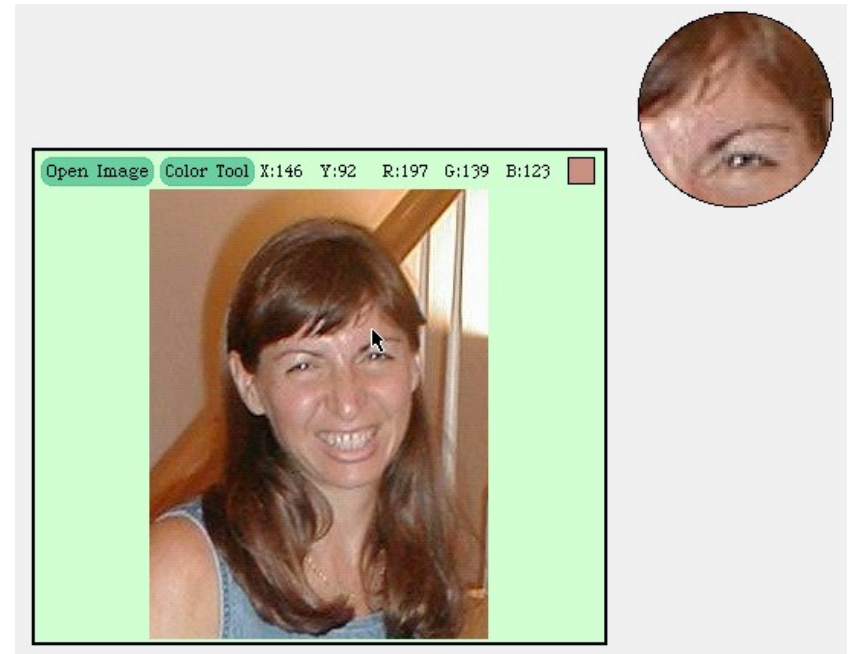
```
>>> file="/Users/guzdial/mediasources/barbara.jpg"  
>>> pict=makePicture(file)  
>>> show(pict)  
>>> setColor(getPixel(pict,10,100),yellow)  
>>> setColor(getPixel(pict,11,100),yellow)  
>>> setColor(getPixel(pict,12,100),yellow)  
>>> setColor(getPixel(pict,13,100),yellow)  
>>> repaint(pict)
```

But that's *really* dull and boring...



# How do you find out what RGB values you have? And where?

- Use the JES MediaTools!



# Use a loop! Our first picture recipe

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

Used like this:

```
>>> file="/Users/guzdial/mediasources/barbara.jpg"  
>>> picture=makePicture(file)  
>>> show(picture)  
>>> decreaseRed(picture)  
>>> repaint(picture)
```





# How loops are written in Python

- `for` is the name of the command
- An *index variable* is used to represent the different values in the loop
- The word `in`
- A function that generates a *sequence*
  - The index variable will be the name for each value in the sequence, each time through the loop
- A colon (":")
- And, another block



# What happens when a loop is executed

- The index variable is set to the next (or first, at the beginning) item in the sequence
- The block is executed
  - The index variable is often used inside the block
- Then execution returns to the `for` statement, where the index variable gets set to the next item in the sequence
- Repeat until the sequence is exhausted.

## getPixels() returns a sequence of pixels

- Each pixel knows its color and its original picture
- When you change the pixels, you change the picture
- So the loop below assigns the index variable `p` to each pixel in the picture `picture`, one at a time.

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

# Let's walk that through slowly...

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```



Here we get a picture object in as input and call it **picture**.

**picture**



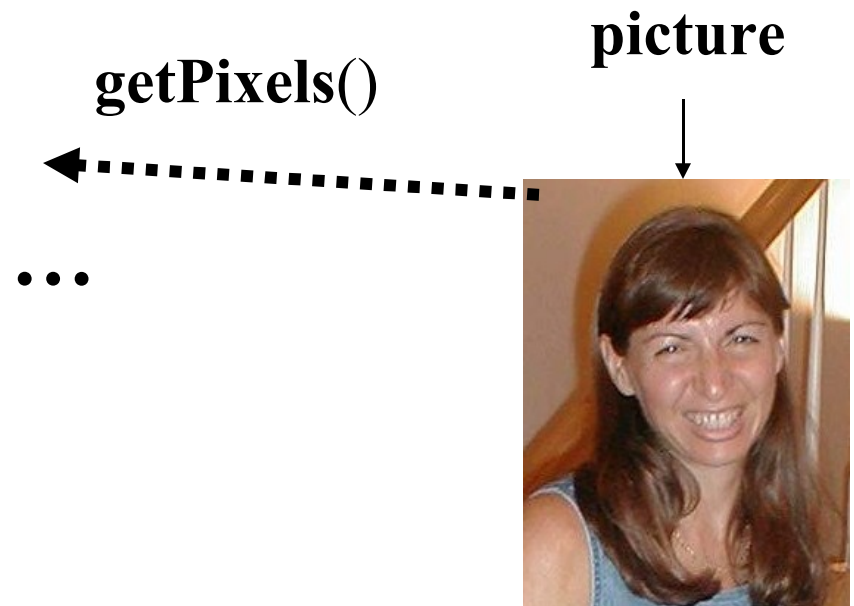
# Now, get the pixels

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

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

Pixel, color r=168 g=131 b=105	Pixel, color r=160 g=131 b=105	Pixel, color r=168 g=132 b=106
--	--	--

**p**



# Get the red value from pixel

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

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



Pixel, color r=168 g=131 b=105	Pixel, color r=160 g=131 b=105	Pixel, color r=168 g=132 b=106
--	--	--

...

**p**

**value = 168**

**picture**



# Now change the pixel

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

Divide the red value of pixel **p** by 2.0 (50% of original) of **value**

Pixel, color <b>r=84</b> g=131 b=105	Pixel, color r=160 g=131 b=105	Pixel, color r=168 g=132 b=106
--	--	--

...

↑  
**p**

**value = 168**

**picture**



# Then move on to the next pixel

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

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

Pixel, color r=84 g=131 b=105	Pixel, color r=160 g=131 b=105	Pixel, color r=168 g=132 b=106	...
---	--	--	-----

↑  
**p**

**value = 168**

**picture**



# Get its red value

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```

← Change **value** to the new red value at the new **p**

Pixel, color r=84 g=131 b=105	Pixel, color r=160 g=131 b=105	Pixel, color r=168 g=132 b=106	...
---	--	--	-----

↑  
**p**

**value = 160**

**picture**



# And change *this* red value

```
def decreaseRed(picture):  
  for p in getPixels(picture):  
    value=getRed(p)  
    setRed(p, value/2.0)
```

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



Pixel, color r=84 g=131 b=105	Pixel, color <b>r=80</b> g=131 b=105	Pixel, color r=168 g=132 b=106
---	--	--

...

**p**

**value = 160**

**picture**



# And eventually, we do all pixels

- We go from this... to this!





# “Tracing/Stepping/Walking through” the program

- What we just did is called “tracing”, or “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 would specifically happen there
  - 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.

## Do we need the variable value?

- Not really: Remember that we can swap names for data or functions that are equivalent.

```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        setRed(p, getRed(p)/2.0)
```

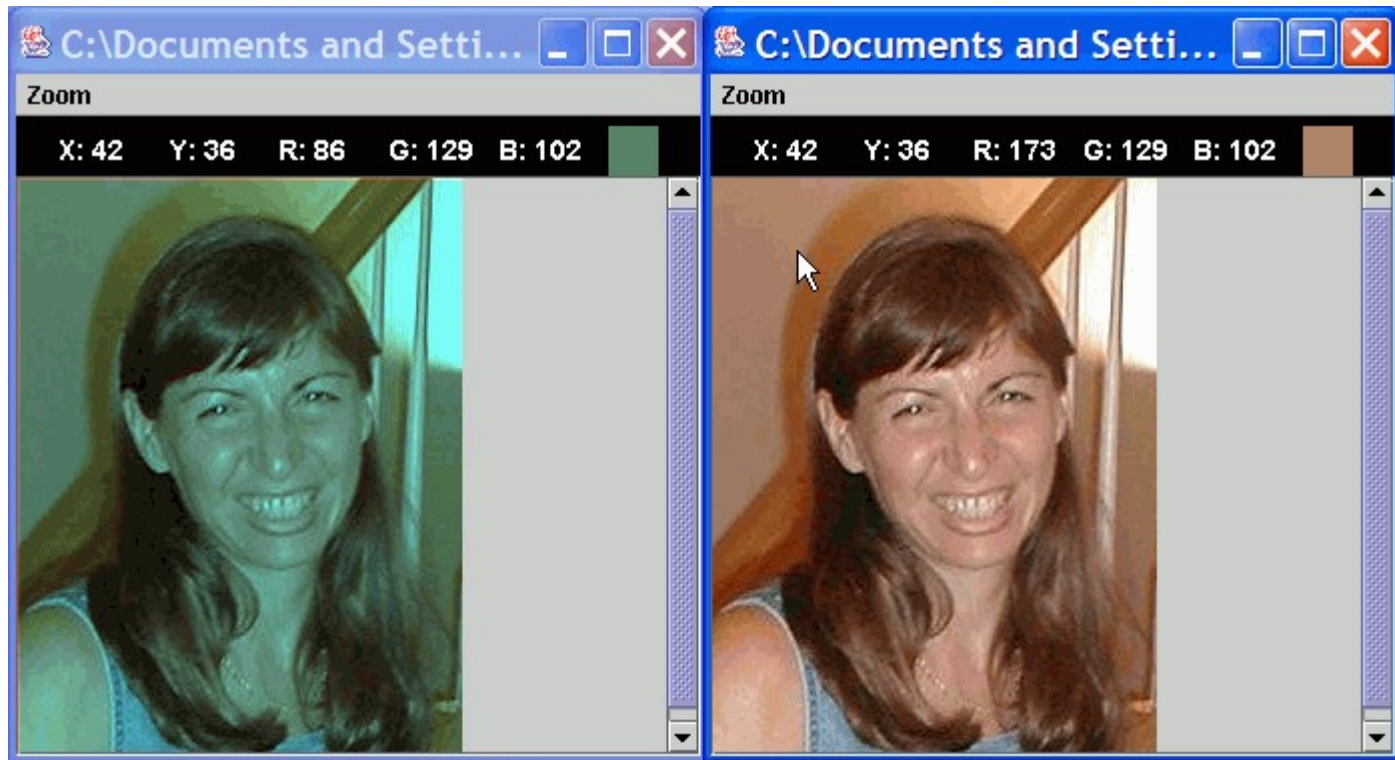


# Did that really work? How can we be sure?

- Sure, the picture *looks* different, but did we actually decrease the amount of red? By as much as we thought?
- Let's check it!

```
>>> file = pickAFile()
>>> print file
C:\Documents and Settings\Mark Guzdial\My
Documents\mediasources\barbara.jpg
>>> pict = makePicture(file)
>>> pixel = getPixel(pict,1,1)
>>> print pixel
Pixel, color=color r=168 g=131 b=105
>>> decreaseRed(pict)
>>> print pixel
Pixel, color=color r=168 g=131 b=105
>>> newPixel = getPixel(pict,1,1)
>>> print newPixel
Pixel, color=color r=84 g=131 b=105
>>> print 168 * 0.5
84.0
```

# Checking it in the MediaTools



# If you make something you like... save it!

- Use `writePictureTo(picture, "filename")`
- Writes the picture out as a JPEG
- Be sure to end your filename as `".jpg"`!
- If you don't specify a full path, the file will be saved in the same directory as JES.

```
>>> writePictureTo(picture, "barbaraLessRed.jpg")
```



## What to do now...

- Read chapter 3 in text book (section 3.1 to 3.3)
- Look at the example program `decreaseRed()` given on COMP2720 Web page