



COMP2720: Automating Tools for New Media

More on using loops for pictures

Use a loop! Our first picture recipe

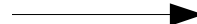
```
def decreaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value/2.0)
```



Used like this:

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

Once we make it work for one picture, it will work for any picture



Think about what we just did

- Did we change the program at all?
- Did it work for all the different examples?
- What was the input variable `picture` each time, then?
 - It was the *value* of whatever picture we provided as input!

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

Read it as a *recipe*

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

- Recipe: *To decrease the red*
- Ingredients: *One picture, name it picture*
- Step 1: *Get all the pixels of picture. 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*

Let's use something with known red to manipulate:
Santa Claus (who used to be green until CocaCola
started using him as a commercial tool)



What if you decrease Santa's red again and again and again...?

```
>>> file=pickAFile()
>>> pic=makePicture(file)
>>> decreaseRed(pic)
>>> show(pic)
(that's the first one)
>>> decreaseRed(pic)
>>> repaint(pic)
(that's the second)
```



Increasing red

```
def increaseRed(picture):  
    for p in getPixels(picture):  
        value=getRed(p)  
        setRed(p, value*1.2)
```



What happened here?!?

Remember that the limit for redness is 255.

If you go beyond 255, all kinds of weird things can happen...

How does `increaseRed()` differ from `decreaseRed()`?

- Well, it does increase rather than decrease red, but other than that...
 - It takes the same input
 - It can also work for any picture
 - It's a specification of a process that'll work for any picture.
 - There's nothing specific to any picture here.



**Practical programs =
parameterised processes**

Clearing blue

```
def clearBlue(picture):  
    for p in getPixels(picture):  
        setBlue(p, 0)
```

Again, this will work for any picture.

Try stepping through this one yourself!



Can we combine these? Why not!

- How do we turn this beach scene into a sunset?
- What happens at sunset?
 - At first, we tried to increase the red, but that made things like red specks in the sand REALLY prominent.
 - That can't be how it really works
 - New Theory: As the sun sets, less blue and green is visible, which makes things look more red.



A sunset-generation function

```
def makeSunset(picture):  
    for p in getPixels(picture):  
        value=getBlue(p)  
        setBlue(p, value*0.7)  
        value=getGreen(p)  
        setGreen(p, value*0.7)
```



Creating the negative of a picture

- Let's think it through
 - Red, green and blue (RGB) go from 0 to 255.
 - Let's say Red is 10. That's just a little bit of red.
 - What's the opposite? LOTS of Red!
 - The negative of that would be 245: $255-10$
- So, for each pixel, if we *negate* each color component in creating a new color, we negate the whole picture.

Recipe for creating a negative

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



Original, negative, double negative



**(This gives us a quick way to test our function:
Call it twice and see if the result is equivalent
to the original)**

Converting to greyscale

- We know that if red=green=blue, we get grey
 - But what value do we set all three to?
- What we need is a value representing the darkness of the color, the *luminance*
- There are lots of ways of getting it, but one way that works reasonably well is dirt simple — simply take the average:

$$\frac{(red+green+blue)}{3}$$

Converting to greyscale

```
def greyScale(picture):  
    for p in getPixels(picture):  
        intensity = (getRed(p)+getGreen(p)+getBlue(p))/3  
        setColor(p, makeColor(intensity, intensity, intensity))
```



**Shouldn't that
be "grayscale"?**

Why can't we get back again?

- Converting to greyscale is different from computing a negative.
 - A negative transformation retains information.
- With greyscale, we've lost information
 - We no longer know what the ratios are between the reds, the greens, and the blues
 - We no longer know any particular value.

Media compressions are one kind of transformation.
Some are **lossless** (like negative);
Others are **lossy** (like greyscale), like JPEG

But that's not really the best greyscale

- In reality, we don't perceive red, green, and blue as equal in their amount of luminance: How bright (or non-bright) something is.
 - We tend to see blue as "darker" and red as "brighter", even if, physically, the same amount of light is coming off from each
- *PhotoShop's* greyscale is very nice: Very similar to the way that our eye sees it.
 - Black & White TV's are also pretty good

Building a better greyscale

- We'll *weight* red, green, and blue based on how light we perceive them to be, based on laboratory experiments.

```
def greyScaleNew(picture):  
  for px in getPixels(picture):  
    newRed = getRed(px) * 0.299  
    newGreen = getGreen(px) * 0.587  
    newBlue = getBlue(px) * 0.114  
    luminance = newRed+newGreen+newBlue  
    setColor(px, makeColor(luminance,luminance,luminance))
```

Comparing the two greyscale:
Average on left, weighted on right



Let's use a black cat to compare



Average on left, weighted on right





What to do now...

- Finish reading chapter 3 by the end of this week!
- Prepare for lab 1 (next week): Read lab sheet on COMP2720 Web page (available soon)
- Read assignment 1 and portfolio specifications (available later this week)