



# Lab: Day 1

## Media Computation



### What you'll do:

Changing colors

Negatives

Grayscale

Making sunsets

### What you'll learn:

Jython Basics

Calling Functions

Assigning Variables

For loops

## Exercise 4: Changing Colors

Recall the structure of a for loop...

for item in items:

do something with item      *# indentation!!!*

### Exercise 4(a): `increaseRed`

Write the function, `increaseRed`, which should take in a picture and increase the red in the picture. Did anything interesting happen?

Here's an example in pseudocode:

```
def increaseRed(somePicture):  
    for each pixel in the picture  
        - get the red value and assign it to a variable  
        - set the red value to be slightly higher by multiplying by some constant  
    show the updated picture
```

### Handy built-in functions that you'll find helpful:

\* You can find these by looking at the "JES Functions" drop-down menu in the bar at the top of your computer. They can be found under the functions for pictures and pixels!

- `getPixels(picture)`
- `makeColor(redValue, greenValue, blueValue)`
- `getColor(pixel)`
- `setColor(pixel, newColor)`
- `getRed(pixel)`
- `setRed(pixel, redness)`

### Exercise 4(b): `clearBlue`

Here's an example in pseudocode:

```
def clearBlue(somePicture):  
    for each pixel in the picture  
        - set the value of blue to 0      # why don't we need to save the original blue?  
    show the updated picture
```

## Exercise 5: negative



### Strategy:

- Let's think it through
  - R, G, B go from 0 to 255
  - Let's say Red is 10. That's very light 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.

### Exercise 5(a): negative

- Define a new function named `negative` that takes a picture as input and makes that picture negative.  
(Go to the next page if you are stuck)

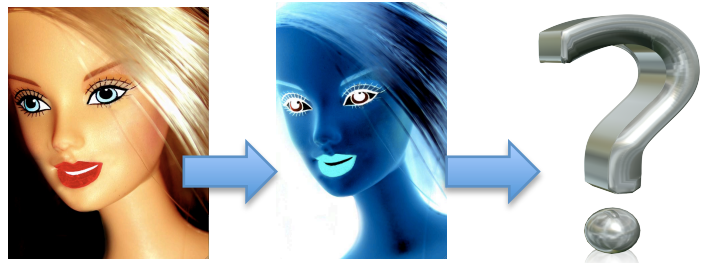
## Exercise 5(a): negative

Here's an example in pseudocode:

```
def negative(somePicture):  
    for each pixel in the picture  
        - find the red, green and blue values of the pixel and save those values to variables  
        - make a new color with the new values being 255 – the old values  
        - set the color of the picture to be the new color  
    show the updated picture
```

Handy built-in functions that you'll find helpful:

```
getPixels(picture)  
makeColor(redValue, greenValue, blueValue)  
getColor(pixel)  
setColor(pixel, newColor)  
getRed(pixel)  
getGreen(pixel)  
getBlue(pixel)  
setRed(pixel, redness)  
setBlue(pixel, blueness)  
setGreen(pixel, greenness)
```



## Exercise 5(b): double negative

“A double negative occurs when two forms of a negation are used in the same clause...canceling one another and producing an affirmative sense” ~Wikipedia

No! Not *that* “double negative!” That’s what you get for quoting such a reliable source. 😊

## Concept:

Actually, the idea that a double negative results in a positive isn’t that far off from the purpose of this exercise. Calling negative once should give us the opposite picture. If we call it again, should it not give us the original picture?

## Experiment:

Try calling the function twice. Once to get the negative of our original picture, and then again on that negative to get the original back! Does it work?

## Exercise 6: grayscale

“When you photograph people in color you photograph their clothes. But when you photograph people in B&W, you photograph their souls!” ~Ted Grant



### The Big Picture:

- We know that if red=green=blue, we get gray
  - 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 many ways, but one way that works reasonably well is dirt simple—simply take the average:

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

### Exercise 6(a): grayscale

- Define a new function named `grayscale` that takes a picture as input and converts that picture to black and white!

(use the next page for reference if you are stuck)

## Exercise 6(a) (cont): grayscale

Here's an example in pseudocode:

```
def grayscale(somePicture):  
    for each pixel in the picture  
        - find the sum of the red, blue and green values of that pixel and assign  
          that to a variable  
        - divide that by three and assign that to some variable representing the  
intensity  
        - set the color of the picture to be the new color  
    show the updated picture
```

Recall the structure of a for loop...

```
for item in items:  
    do something with item
```

Handy built-in functions that you'll find helpful:

```
getPixels(picture)  
makeColor(redValue, greenValue, blueValue)  
getColor(pixel)  
setColor(pixel, newColor)  
getRed(pixel)  
getGreen(pixel)  
getBlue(pixel)  
setRed(pixel, redness)  
setBlue(pixel, blueness)  
setGreen(pixel, greenness)
```

- After you've fully created your function, load your program and call this function at the bottom of your program by typing:

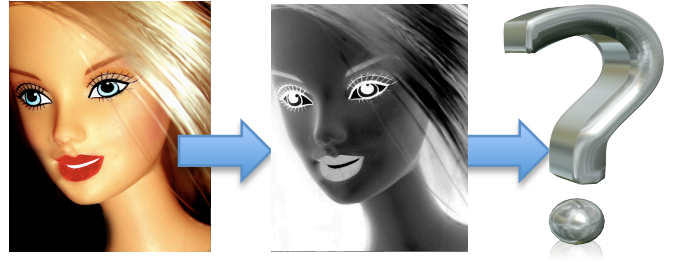
```
> grayscale(somePicture)
```

- Then show that picture!

```
> show(somePicture)
```

You should now see your picture in black and white!

## Exercise 6(b): Double grayscale



### Concept:

With our negative function, we were able to get our original picture back by calling the function twice!

### Experiment:

Try this with grayscale! What happens? Try to explain why to your partner!

## Exercise 6(c): Building a better grayscale!

### Concept:

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 of each

### Experiment:

Rewrite grayscale to set each level (red, green, blue) to the `getRed()` of the pixel. Compare this to the result of setting each level to the `getBlue()` of the pixel. How do they compare to the original grayscale? Make a guess before you try it!

We will show you our version of grayscale towards the end of lab, but try for yourself!



## Exercise 7: makeSunset

We are now going to combine some of the ideas we have practiced so far, by taking a normal beach picture and generating a new picture that looks like a sunset.

Strategy: A sunset picture is more red.



- Find a picture online with some normal beach (not already with a sunset). Save the photo into your [Name] folder.
- Define a new function named `makeSunset` that takes a picture as input.
- **The Big Picture:** We want to use the idea of looping through the pixels in a picture (just like the previous exercises) to create a photo that looks more like a sunset. A sunset has more red coloring!
- Question: Does our sunset function generate a believable sunset image when we increase the redness by a certain factor? Try this:

Alternative Strategy: As the sun sets, less blue and green is visible, which makes things look more red.