Image Manipulation

Institute for Personal Robots in Education (IPRE)

CS 1 with Robots

Taking / Saving / Loading a Picture

```
p = takePicture()
show(p)
savePicture(p, "class.gif")

p2 = loadPicture("class.gif")
```

```
print getWidth(p)
```

256

```
print getHeight(p)
```

192

Robot Pictures

- Robot pictures are:
  - 256 pixels wide by 192 pixels high
- Each pixel is made up of 3 colors:
  - Red, Green, and Blue
  - Colors range in value from 0 – 255
  - Plus an “Alpha” value we won't use (for now).
- When you print a picture or a pixel, Python gives you some textual information about the item, but does not give a visual representation.
- The show(picture) method will display the picture on the screen.

Colors (R,G,B)

- Different colors can be made by mixing different values of the primary (Red,Green,Blue) color lights. In computer output, colors are made by adding lights of different colors, and color combinations are additive. For example:
  - (0, 0, 0) – Black
  - (255, 255, 255) – White
  - (255, 0, 0) – Red
  - (255, 255,0) – Yellow
  - (255, 0, 255) – Magenta / fuchsia (bright purple)
  - (0, 255, 255) – Cyan (bright teal)

Accessing a specific pixel: getPixel(picture, x,y)

```
print p
```

<Picture instance (256 x 192)>

```
pix = getPixel(p, 50,50)
print pix
```

<Pixel instance (r=153, g=255, b=255, a=255) at (50, 50)>

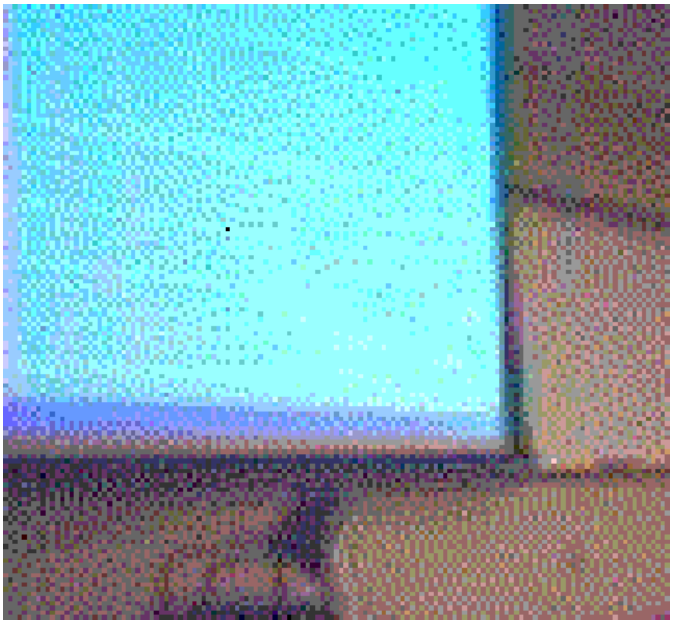
```
setRed(pix,0)
setGreen(pix,0)
setBlue(pix,0)

print pix
```

<Pixel instance (r=0, g=0, b=0, a=255) at (50, 50)>

```
show(p)
```


Zoomed In View: One Black Pixel



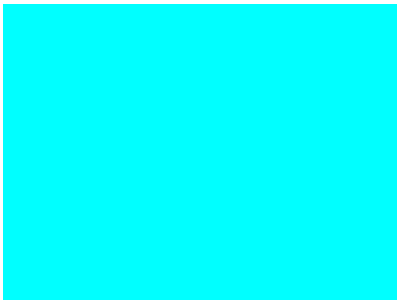
Looping through all pixels: getPixels(picture)

```
print p

for pix in getPixels(p):
   setRed(pix,0)
   setBlue(pix,255)
   setGreen(pix,255)




show(p)
```



Looping through all pixels: getPixels(picture)

```
p = loadPicture("class.gif")

for pix in getPixels(p):
   setRed(pix,255)
```

But what if you only change the red part of the pixels?

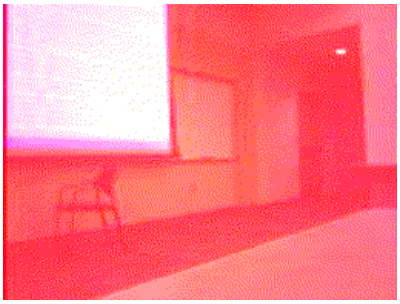
Looping through all pixels: getPixels(picture)

```
p = loadPicture("class.gif")

for pix in getPixels(p):
   setRed(pix,255)

show(p)
```

The Green and Blue parts of the pixels retain their original information!



Pixel Functions

- Getting a specific pixel:
  - getPixel(picture,x,y)
- Getting all pixels:
  - getPixels(picture)
- Getting information about pixel color:
  - getRed(pixel)
  - getGreen(pixel)
  - getBlue(pixel)
- ...and about pixel location:
  - getX(pixel)
  - getY(pixel)
- Changing color information:
  - setRed(pixel, value)
  - setGreen(pixel,value)
  - setBlue(pixel,value)

How to change more than 1 pixel, but less than all?

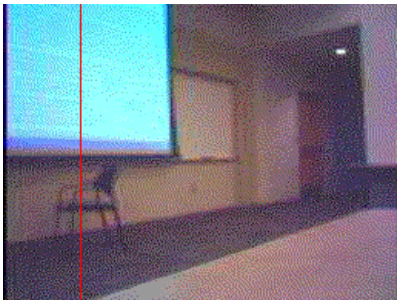
```
p = loadPicture("class.gif")

for pix in getPixels(p):
   x = getX(pix)
   if(x == 50):
      setRed(pix,255)
      setGreen(pix,0)
      setBlue(pix,0)



show(p)
```

You can check each pixel to see if you want to change it!



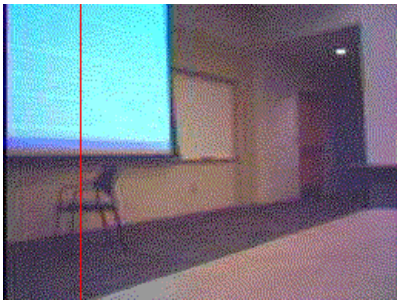
Drawing a line, quickly!

- A (robot) picture has 256 x 192 pixels = 49,152 pixels
- A horizontal line has 256 pixels, and a vertical line has 192 pixels.
- Code on the previous slide looked at all 49,152 pixels and decided if it should color them red based upon the value of the X coordinate.
- That's 49,152 calls to getX and 49,152 IF statement checks!
- No wonder it takes a second to finish!

Drawing a line, quickly!

```
p = loadPicture("class.gif")
pHeight = getHeight(p)
for y in range(pHeight):
   pix = getPixel(p,50,y)
   setRed(pix,255)
   setGreen(pix,0)
   setBlue(pix,0)
show(p)
```

Only work on the pixels you want to change!



Only work with the pixels we want to change!

- The quick line drawing code only works with 192 pixels!
- That's only 192 calls to getPixel, and 192 cycles through the loop!
- It's no surprise that this code runs faster!
- How would we change it to draw a horizontal line?

Drawing a horizontal line!

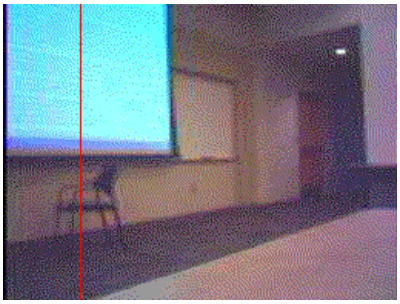
```
p = loadPicture("class.gif")

pHeight = getHeight(p)

for y in range(pHeight):
   pix = getPixel(p,50,y)
   setRed(pix,255)
   setGreen(pix,0)
   setBlue(pix,0)

show(p)
```

Change all the code that deals with height, and make it work with width instead!



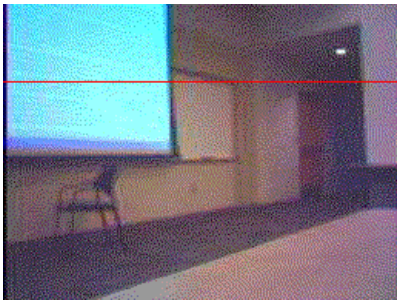
Drawing a horizontal line

```
p = loadPicture("class.gif")

pWidth = getWidth(p)

for x in range(pWidth):
   pix = getPixel(p,x,50)
   setRed(pix,255)
   setGreen(pix,0)
   setBlue(pix,0)

show(p)
```



Making it more general...

```
p = loadPicture("class.gif")

pWidth = getWidth(p)

for x in range(pWidth):
   pix = getPixel(p,x,50)
   setRed(pix,255)
   setGreen(pix,0)
   setBlue(pix,0)




show(p)
```

Let's turn this code into a function!
What do we want to control when drawing a horizontal line?
What parameters do we want to send into the function?

H-line function

```
def drawHLine(pic, y, r,g,b):
   pWidth = getWidth(pic)

   for x in range(pWidth):
      pix = getPixel(pic,x,y)
      setRed(pix,r)
      setGreen(pix,g)
      setBlue(pix,b)
```

```
p = loadPicture("class.gif")
drawHLine(p, 20, 0, 255,0)
show(p)
```

Moving the H-Line

- We have a function that draws a line.
- Lets animate the line by drawing it multiple times!
- Our procedure:
  - Draw the line at the top of the picture
  - Show the picture
  - Draw the line one pixel lower
  - Show the picture again
  - Repeat until the line is at the bottom of the picture!

Moving Horizontal Line

```
p = loadPicture("class.gif")

y = 0 #Start at the top!

while(y < getHeight(p) ):
   drawHLine(p, y, 255,0,0)
   show(p)
   wait(0.05)
   y = y + 1
```

Moving Horizontal Line

- Why didn't that work?
- How can we fix it?

Saving overwritten data

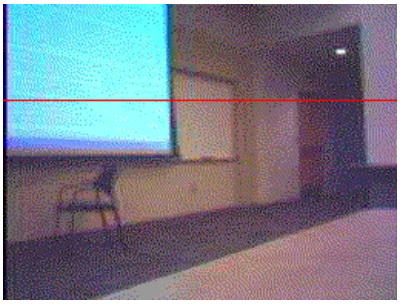
- Our drawHLine() function overwrites pixel data. But, it doesn't restore pixel data!
  - To fix it, we could save the pixel data from each line, and then restore it before we draw the next line.
- Or, we could be lazy, and just save a copy of the entire picture!
  - By drawing our line on a new copy of the picture each time, we ensure that only the line we are drawing now will be red!

Moving Horizontal Line (working on a picture copy)

```
p = loadPicture("class.gif")
p2 = copyPicture(p)

y = 0 #Start at the top!

while(y < getHeight(p) ):
   drawHLine(p2, y, 255,0,0)
   show(p2)
   wait(0.05)
   y = y + 1
   p2 = copyPicture(p)
```



Speeding things up?

- Ok, that works, but it was kinda slow...
  - Did we wait too long? What happens if we change the wait time to zero?
- How could we speed things up?
  - Buy a faster computer!
- We could cheat....
  - By only drawing the line every five lines, it would *appear* to go down the image faster.

Moving Horizontal Line (every 5th line)

```
p = loadPicture("class.gif")
p2 = copyPicture(p)

y = 0 #Start at the top!

while(y < getHeight(p) ):
   if (y % 5 == 0):
      drawHLine(p2, y, 255,0,0)
      show(p2)
      wait(0.05)
      p2 = copyPicture(p)
   y = y + 1
```


What else can we do?

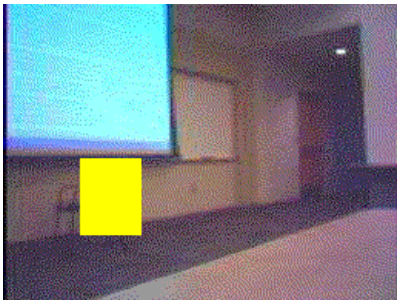
- Draw a rectangle!
- Copy a rectangle from one picture to another!

Draw a rectangle!

```
p = loadPicture("class.gif")

for x in range(50,90):
   for y in range(100,150):
      pix = getPixel(p,x,y)
      setRed(pix,255)
      setGreen(pix,255)
      setBlue(pix,0)

show(p)
```



Double (Nested) for loops!

- You just saw something that is very useful when working with two dimensional data.
- By using two for loops, one nested inside of the other, you can iterate two variables (e.g. x & y) over all values within a rectangle!

```
for x in range(startX,endX):
    for y in range(startY,endY):
        doSomethingAt( x, y)
```

Copy a rectangle!

```
p = loadPicture("class.gif")
p2 = loadPicture("p3.jpg")
show(p); show(p2)


for x in range(30,100):
   for y in range(60,170):
      pix = getPixel(p,x,y)
      pix2 = getPixel(p2,x,y)
      setRed(pix, getRed(pix2) )
      setGreen(pix,getGreen(pix2) )
      setBlue(pix,getBlue(pix2) )



show(p)
```