Assignment 1: Building a Better Contact Sheet

In the lectures for this week you were shown how to make a contact sheet for digital photographers, and how you can take one image and create nine different variants based on the brightness of that image. In this assignment you are going to change the colors of the image, creating variations based on a single photo. There are many complex ways to change a photograph using variations, such as changing a black and white image to either "cool" variants, which have light purple and blues in them, or "warm" variants, which have touches of yellow and may look sepia toned. In this assignment, you'll be just changing the image one color channel at a time

Your assignment is to learn how to take the stub code provided in the lecture (cleaned up below), and generate the following output image:

From the image you can see there are two parameters which are being varied for each sub-image. First, the rows are changed by color channel, where the top is the red channel, the middle is the green channel, and the bottom is the blue channel. Wait, why don't the colors look more red, green, and blue, in that order? Because the change you to be making is the ratio, or intensity, or that channel, in relationship to the other channels. We're going to use three different intensities, 0.1 (reduce the channel a lot), 0.5 (reduce the channel in half), and 0.9 (reduce the channel only a little bit).

For instance, a pixel represented as (200, 100, 50) is a sort of burnt orange color. So the top row of changes would create three alternative pixels, varying the first channel (red). one at (20, 100, 50), one at (100, 100, 50), and one at (180, 100, 50). The next row would vary the second channel (blue), and would create pixels of color values (200, 10, 50), (200, 50, 50) and (200, 90, 50).

Note: A font is included for your usage if you would like! It's located in the file `readonly/fanwood-webfont.ttf`

Need some hints? Use them sparingly, see how much you can get done on your own first! The sample code given in the class has been cleaned up below, you might want to start from that.

In [6]:

```
import PIL
from PIL import Image
from PIL import ImageEnhance
from PIL import ImageFont, ImageDraw

# read image and convert to RGB
image=Image.open("readonly/msi_recruitment.gif")
image=image.convert('RGB')

# build a list of 9 images which have different colors
images=[]
lables=[]
#i defines the channel and j defines the intensity
for i in range(3):
    for j in (0.1,0.5,0.9):
        # split the image into individual bands for RGB
        source = image.split()
        # process the selected band
        mid = source[i].point(lambda x:x*j)
        # paste the processed band back
        source[i].paste(mid)
        # build a new multiband image
        im = Image.merge(image.mode, source)
        #creating label texts list
        lables.append('channel {} intensity {}'.format(i,j))
        #creating images list
        images.append(im)
#specifying font type
font = ImageFont.truetype("readonly/fanwood-webfont.ttf",75)

# create a contact sheet from different color
first_image=images[0]
contact_sheet=PIL.Image.new(first_image.mode, (first_image.width*3,first_image.h
eight*3+3*85))
x=0
y=0
draw = ImageDraw.Draw(contact_sheet)
for i,img in enumerate(images):
    # Lets paste the current image into the contact sheet
    contact_sheet.paste(img, (x, y) )
    #adding the lable images
    draw.text((x,y+first_image.height+5), lables[i], font=font)
    # Now we update our X position. If it is going to be the width of the image,
then we set it to 0
    # and update Y as well to point to the next "line" of the contact sheet.
    if x+first_image.width == contact_sheet.width:
        x=0
        y=y+first_image.height+85
    else:
        x=x+first_image.width

# resize and display the contact sheet
contact_sheet = contact_sheet.resize((int(contact_sheet.width/2),int(contact_she
et.height/2) ))
display(contact_sheet)
```

HINT 1

Check out the `PIL.ImageDraw module` for helpful functions

HINT 2

Did you find the `text()` function of `PIL.ImageDraw` ?

HINT 3

Have you seen the `PIL.ImageFont` module? Try loading the font with a size of 75 or so.

HINT 4

These hints aren't really enough, we should probably generate some more.

In [6]:

```
from PIL import ImageDraw
help(ImageDraw)
```

```
Help on module PIL.ImageDraw in PIL:

NAME
    PIL.ImageDraw

DESCRIPTION
    # The Python Imaging Library
    # $Id$
    #
    # drawing interface operations
    #
    # History:
    # 1996-04-13 fl   Created (experimental)
    # 1996-08-07 fl   Filled polygons, ellipses.
    # 1996-08-13 fl   Added text support
    # 1998-06-28 fl   Handle I and F images
    # 1998-12-29 fl   Added arc; use arc primitive to draw ellipses
    # 1999-01-10 fl   Added shape stuff (experimental)
    # 1999-02-06 fl   Added bitmap support
    # 1999-02-11 fl   Changed all primitives to take options
    # 1999-02-20 fl   Fixed backwards compatibility
    # 2000-10-12 fl   Copy on write, when necessary
    # 2001-02-18 fl   Use default ink for bitmap/text also in fill m
ode
    # 2002-10-24 fl   Added support for CSS-style color strings
    # 2002-12-10 fl   Added experimental support for RGBA-on-RGB dra
wing
    # 2002-12-11 fl   Refactored low-level drawing API (work in prog
ress)
    # 2004-08-26 fl   Made Draw() a factory function, added getdraw
() support
    # 2004-09-04 fl   Added width support to line primitive
    # 2004-09-10 fl   Added font mode handling
    # 2006-06-19 fl   Added font bearing support (getmask2)
    #
    # Copyright (c) 1997-2006 by Secret Labs AB
    # Copyright (c) 1996-2006 by Fredrik Lundh
    #
    # See the README file for information on usage and redistributio
n.
    #

CLASSES
    builtins.object
        ImageDraw

    class ImageDraw(builtins.object)
     |  ImageDraw(im, mode=None)
     |
     |  Methods defined here:
     |
     |  __init__(self, im, mode=None)
     |      Create a drawing instance.
     |
     |      :param im: The image to draw in.
     |      :param mode: Optional mode to use for color values.  For
RGB
     |         images, this argument can be RGB or RGBA (to blend th
e
     |         drawing into the image).  For all other modes, this a
rgument
```

```
      |         must be the same as the image mode.  If omitted, the
mode
      |         defaults to the mode of the image.
      |
      |  arc(self, xy, start, end, fill=None, width=0)
      |      Draw an arc.
      |
      |  bitmap(self, xy, bitmap, fill=None)
      |      Draw a bitmap.
      |
      |  chord(self, xy, start, end, fill=None, outline=None, width=
0)
      |      Draw a chord.
      |
      |  ellipse(self, xy, fill=None, outline=None, width=0)
      |      Draw an ellipse.
      |
      |  getfont(self)
      |      Get the current default font.
      |
      |      :returns: An image font.
      |
      |  line(self, xy, fill=None, width=0, joint=None)
      |      Draw a line, or a connected sequence of line segments.
      |
      |  multiline_text(self, xy, text, fill=None, font=None, anchor=
None, spacing=4, align='left', direction=None, features=None)
      |
      |  multiline_textsize(self, text, font=None, spacing=4, directi
on=None, features=None)
      |
      |  pieslice(self, xy, start, end, fill=None, outline=None, widt
h=0)
      |      Draw a pieslice.
      |
      |  point(self, xy, fill=None)
      |      Draw one or more individual pixels.
      |
      |  polygon(self, xy, fill=None, outline=None)
      |      Draw a polygon.
      |
      |  rectangle(self, xy, fill=None, outline=None, width=0)
      |      Draw a rectangle.
      |
      |  shape(self, shape, fill=None, outline=None)
      |      (Experimental) Draw a shape.
      |
      |  text(self, xy, text, fill=None, font=None, anchor=None, *arg
s, **kwargs)
      |
      |  textsize(self, text, font=None, spacing=4, direction=None, f
eatures=None)
      |      Get the size of a given string, in pixels.
      |
      |  ----------------------------------------------------------
----------
      |  Data descriptors defined here:
      |
      |  __dict__
      |      dictionary for instance variables (if defined)
      |
```

```
      |  __weakref__
      |      list of weak references to the object (if defined)

FUNCTIONS
    Draw(im, mode=None)
        A simple 2D drawing interface for PIL images.

        :param im: The image to draw in.
        :param mode: Optional mode to use for color values.  For RGB
           images, this argument can be RGB or RGBA (to blend the
           drawing into the image).  For all other modes, this argum
ent
           must be the same as the image mode.  If omitted, the mode
           defaults to the mode of the image.

    Outline = outline(...)

    floodfill(image, xy, value, border=None, thresh=0)
        (experimental) Fills a bounded region with a given color.

        :param image: Target image.
        :param xy: Seed position (a 2-item coordinate tuple). See
            :ref:`coordinate-system`.
        :param value: Fill color.
        :param border: Optional border value.  If given, the region
consists of
            pixels with a color different from the border color.  If
not given,
            the region consists of pixels having the same color as t
he seed
            pixel.
        :param thresh: Optional threshold value which specifies a ma
ximum
            tolerable difference of a pixel value from the 'backgrou
nd' in
            order for it to be replaced. Useful for filling regions
of
            non-homogeneous, but similar, colors.

    getdraw(im=None, hints=None)
        (Experimental) A more advanced 2D drawing interface for PIL
images,
        based on the WCK interface.

        :param im: The image to draw in.
        :param hints: An optional list of hints.
        :returns: A (drawing context, drawing resource factory) tupl
e.

FILE
    /opt/conda/lib/python3.7/site-packages/PIL/ImageDraw.py
```

In [ ]: