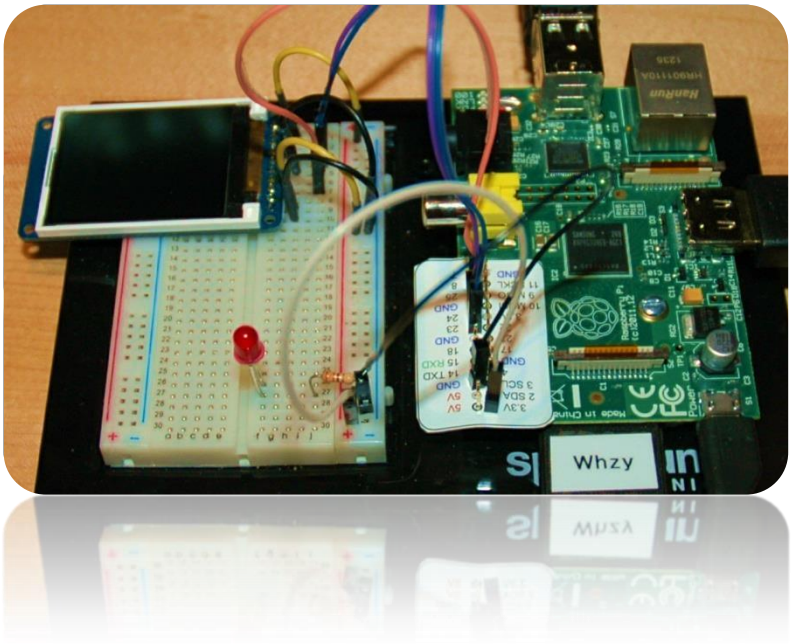


# Add a TFT Display to the Raspberry Pi

**Part 4: Colors, Gamma**

Bruce E. Hall, W8BH

Objective: Control a 160x128 pixel TFT LCD using Python.

## 1) INTRODUCTION

In Part 1, we connected a 1.8" TFT module from Adafruit to the Raspberry Pi. Part 2 added speed using the hardware SPI interface, Part 3 added a simple graphics library, and Part 4 added text. Now let's finish with a discussion of colors and gamma.

## 2) COLOR DICTIONARY

The HTML and CSS specifications define 140 unique, named colors. Colors like "BlanchedAlmond" and "Honeydew". I created a file containing the names and RGB565 values for all 140 colors, called rgb565.py. Here are a few entries in the file:

```
BROWN           = 0xA145
BURLYWOOD       = 0xDDD0
CADETBLUE       = 0x5CF4
CHARTREUSE      = 0x7FE0
CHOCOLATE       = 0xD343
```

Import this file, and all of these colors become immediately available. Python doesn't care much for constants, so all of these entried become global variables. Rather than add 140 variables, you can encapsulate this data in a dictionary. In Python, a dictionary stores data pairs as a key and a value: specify the key, and the dictionary returns the value. Getting a color value would look something like this: value = dict[colorName].

Creating the dictionary from our color file is pretty easy. Each line in the file contains two words, separated by an equals sign. The first word is the color name, and the second is the value. Use the Python 'split' function to split each line into words. By default, it splits according to the space characters. But we can a different split character, like the equals sign. Here is the code:

```
def ColorDictionary(filename):
    dict = {}
    for line in open(filename):          #scan each line of file
        if line.find('=')>0:             #ignore lines without '='
            words = line.split('=')      #split line into two words
            key = words[0].strip()       #first word is the key (colorname)
            value = words[1].strip()     #second word is the RGB value
            dict[key] = int(value,16)    #add pair to dictionary, converting to hex
    return dict
```

It loads the file and iterates over each line in the file, grabbing the color name and value. It adds each item to the dictionary with 'dict[key] = value'. Notice that instead of saving the value as text, it is converted into a number. Python allows base conversion from a hexadecimal string by using 16 as the optional base parameter.

In Part 3 I described how red, green, and blue pixel data is combined into a 16-bit color value. It is handy to be able to convert between the color value and its RGB components. Keeping in mind that there are 5 red bits, 6 green bits, and 5 blue bits, it is just a matter of bit-shifting:

```
def PackColor(red,green,blue):
       "Packs individual red,green,blue color components into 16bit color"
       "16-bit color format is rrrrrggg.gggbbbbb"
       "Max values: Red 31, Green 63, Blue 31"
       r = red & 0x1F
       g = green & 0x3F
       b = blue & 0x1F
       return (r<<11)+(g<<5)+b

def UnpackColor(color):
       "Reduces 16-bit color into component r,g,b values"
       r = color>>11
       g = (color & 0x07E0)>>5
       b = color & 0x001F
       return r,g,b
```

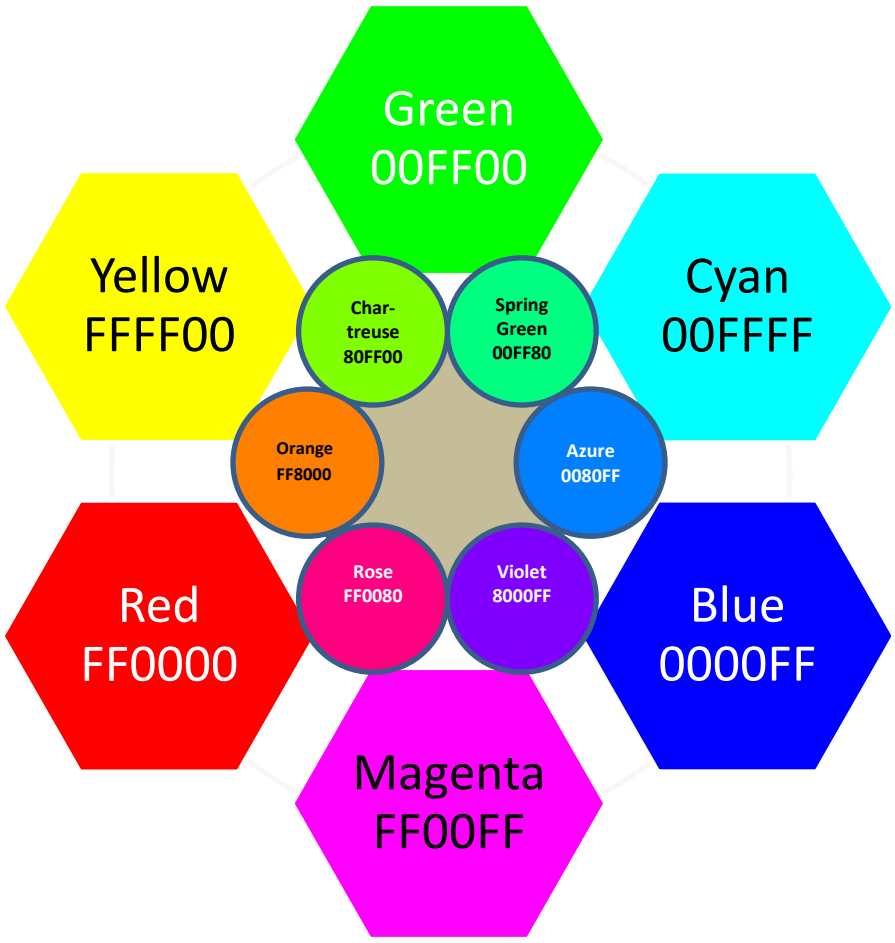
Now we can show a sampling of named colors on screen, with their component data:

```
def ColorTest():
    #load a dictionary with all of the color names & RGB values
    dict = ColorDictionary('rgb565.py')

    #demo with just a few colors
    colorNames = ['AQUA','ORANGE','MAGENTA','SKYBLUE','TOMATO','NAVY']
    for name in colorNames:
        color = dict[name]            #get value from dictionary
        FillRect(0,0,127,127,color)   #display color on LCD

        #display RGB component information at bottom of display
        r,g,b = UnpackColor(color)
        st = "({0:2d},{1:2d},{2:2d}) = {3:4x}  ".format(r,g,b,color)
        PutSt(st,10,140,WHITE)
        st = "{:20s}".format(name)
        PutSt(st,10,150,WHITE)
        time.sleep(1)                 #wait before displaying next color
```

**3) THE RGB COLOR CIRCLE**




There are 12 distinct colors in the RGB color circle, as shown above. These include 3 primary colors, 3 secondary colors, and 6 tertiary colors.

Primary colors are basic colors that are combined to form all of the other colors. With computer displays we use the three additive colors red, green, and blue. In the graphic above, this color triad forms an equilateral triangle, with each color equidistant from the others.

Secondary colors are formed by combining any two adjacent primary colors. For example, the color yellow (FFFF00) is formed by combining red (FF0000) and green (00FF00). Cyan = green + blue, and magenta = blue + red.

Tertiary colors are formed by combining adjacent primary & secondary colors. For example, the color chartreuse (80FF00) is the combination of yellow (FFFF00) & green (00FF00). Since yellow is a mixture of red & green, you could also express chartreuse as an mixture of 1 part red and 2 parts green. Remember that all colors can be expressed as a combination of the primary colors.

Complementary colors are found opposite each other in the circle. These colors sharply contrast each other. For additive colors, magenta (FF00FF) is the complement of green (00FF00). Orange (FF8000) is the complement of azure (0080FF). To calculate a complementary color, subtract each color component values from 0xFF. For example, to calculate the complement of color 0x33cc66, subtract 0xFF-0x33 for red, 0xFF-0xCC for green, and 0xFF-0x66 for blue. The result is 0xCC3399.

Triad colors are any three equidistant colors on the circle. In other words, lines drawn between them form an equilateral triangle. On our 12 color wheel there are four distinct triads: the primary colors, the secondary colors, and two tertiary combinations: (orange, spring green, violet), and (rose, chartreuse, azure). To determine the triad for any color, rotate the red, green, and blue components. For example, rotate 0x33CC66 to the right by 1 byte to get the second color, 0x6633CC. Rotate to the left to get the third color, 0xCC6633. Like the complementary colors, triad colors contrast each other. Use them together sparingly.

The code to determine complementary and triad colors is very compact. The only trick is that green has twice as many values as red and blue in RGB565 colorspace. Notice how TriadColors must take this into account when rotating the subpixels:

```
def Complement(color):
    "Returns color complement"
    r,g,b = UnpackColor(color)
    return PackColor(31-r,63-g,31-b)

def TriadColors(color):
    "Returns two triad colors for given color"
    r,g,b = UnpackColor(color)   #get the rgb components
    tr1 = PackColor(b,r*2,g/2)   #rotate right rgb --> brg
    tr2 = PackColor(g/2,b*2,r)   #rotate left  rgb --> gbr
    return tr1,tr2
```

**4) GAMMA**

Digital image sensors capture light information in a linear fashion: light that doubles in brightness (luminance) will result in a data value twice as large. But our eyes do not work the same way. Human vision requires increasingly more light to be perceived as 'brighter'. This non-linear system allows the human eye to process a much larger range of input. Similarly, we can encode more light information, and improve contrast, by encoding our light input in a non-linear way. This process is called gamma encoding, and is mathematically represented by the power-law equation:

$$Vi = kVo^{\gamma}$$

Where Vi and Vo are the input & output values, k is a constant, and the exponent gamma (γ) is less than one. Most computer images are encoded with a gamma of about 0.45. If images are not gamma encoded, they allocate too many bits to highlights that humans cannot differentiate, and too few bits to details that humans can resolve.

Computer displays must reverse the gamma encoding, so that the luminance of the output image matches the original image. The reverse process, gamma correction or 'decoding', applies the same mathematical formula but with an inverse value (1/0.45 = 2.2) for gamma.

As a simple example for gamma encoding & decoding, let's assume gammas of 0.5 and 2, respectively. Imagine that the raw luminance value for a certain input pixel is 64. Our input device (digital camera) applies the encoding gamma of 0.5, yielding a pixel value in our jpeg file of $64^{0.5} = 8$. To display this pixel, the display device applies the inverse gamma of 2, returning the original luminance value of $8^2 = 64$.

As it turns out, CRT monitors have intrinsic nonlinearity that approximates a gamma of 2.5. They require little or no additional correction. LCD displays, however, have a linear transfer function and must correct for the image file gamma encoding.

Display gamma decoding occurs in the DAC (digital-to-analog converter) of the display driver chip. Our driver contains a gamma table, which maps input pixel values to an output voltage that drives the physical display. The driver has a command, GAMSET, which lets us change the display gamma to one of four preset values: 1, 1.8, 2.2, and 2.5. The non-default value of 2.2 is the standard decoding gamma for computer images. The following two routines will cycle through all four gamma correction levels, immediately applying them to whatever is currently on the display screen:

```
def GamSet(value,text):
    'Set gamma & show text label at bottom of screen'
    'Called by CycleGammas routine'
    FillRect(0,150,127,159,BLACK)     #clear bottom of screen
    PutSt(text,20,152,WHITE)          #put label @ bottom of screen
    Command(GAMSET,value)             #change the gamma here
    time.sleep(2)                     #pause

def CycleGammas(numCycles=3):
    'Cycle display through four different gamma settings'
    for count in range(numCycles):
        GamSet(1,'Gamma 1.0')
        GamSet(2,'Gamma 2.5')
        GamSet(4,'Gamma 2.2')
        GamSet(8,'Gamma 1.8')
```

Put an image on the display and cycle through the gammas. You will find that a gamma of 1.0 creates images that are very bright, but lack contrast. Higher gammas give you darker and more 'contrasty' images. A display gamma of 2.2 generally looks the best.

If you are really adventurous, you can directly manipulate the gamma tables themselves. There are two tables, positive and negative, which each contain 16 values. Each table entry 'tweaks' the voltage output along a section of the gamma correction curve. For my display, the manufacturer provided gamma tables (to Adafruit) that provide the best image transfer function. But if you don't have such a table, using a gamma of 2.2 with GAMSET will usually give good results. [Side note: Why do we need positive and negative tables? Driving an LCD with direct current causes certain electrochemical reactions that shorten LCD life. Therefore they are driven with an AC signal that has positive and negative phases. The positive and negative gamma tables allow correction during both phases of the driving signal.]

I found good references for gamma correction at Wikipedia and CambridgeInColor.

**5) PYTHON SCRIPT for TFT DISPLAY, PART 5:**

The following script combines code from all five TFT articles, and runs a demo that shows text, graphics, color manipulation, and gamma correction. Check out the YouTube video.

```
#!/usr/bin/python

#----------------------------------------------------------------------
#
#  A Python script for controlling the Adafruit 1.8" TFT LCD module
#  from a Raspberry Pi.
#
#   Author :   Bruce E. Hall, W8BH    <bhall66@gmail.com>
#   Date   :   24 Mar 2014
#
#   This module uses the ST7735 controller and SPI data interface
#   PART 5 --- EVERYTHING
#
#   For more information, see w8bh.net
#
#----------------------------------------------------------------------


import cooper14 as font
import font5x7 as font0
import RPi.GPIO as GPIO
import time
import os                       #for popen
import spidev                   #hardware SPI
from random import randint
from math   import sqrt, sin, cos
from struct import unpack_from


#TFT to RPi connections
# PIN TFT          RPi
# 1 backlight    3v3
# 2 MISO        <none>
# 3 CLK            SCLK (GPIO 11)
# 4 MOSI        MOSI (GPIO 10)
# 5 CS-TFT        GND
# 6 CS-CARD        <none>
# 7 D/C            GPIO 25
# 8 RESET        <none>
# 9 VCC            3V3
# 10 GND        GND

DC      = 25
XSIZE   = 128
YSIZE   = 160
XMAX    = XSIZE-1
YMAX    = YSIZE-1
X0      = XSIZE/2
Y0      = YSIZE/2

#Color constants
BLACK   = 0x0000
BLUE    = 0x001F
RED     = 0xF800
GREEN   = 0x0400
LIME    = 0x07E0
CYAN    = 0x07FF
```

```
MAGENTA = 0xF81F
YELLOW  = 0xFFE0
WHITE   = 0xFFFF
PURPLE  = 0x8010
NAVY    = 0x0010
TEAL    = 0x0410
OLIVE   = 0x8400
MAROON  = 0x8000
SILVER  = 0xC618
GRAY    = 0x8410

bColor  = BLACK
fColor  = YELLOW

COLORSET  = [BLACK,BLUE,RED,GREEN,LIME,CYAN,MAGENTA,YELLOW,
             WHITE,PURPLE,NAVY,TEAL,OLIVE,MAROON,SILVER,GRAY]

#TFT display constants
SWRESET = 0x01
SLPIN   = 0x10
SLPOUT  = 0x11
PTLON   = 0x12
NORON   = 0x13
INVOFF  = 0x20
INVON   = 0x21
GAMSET  = 0x26
DISPOFF = 0x28
DISPON  = 0x29
CASET   = 0x2A
RASET   = 0x2B
RAMWR   = 0x2C
RAMRD   = 0x2E
PTLAR   = 0x30
MADCTL  = 0x36
COLMOD  = 0x3A
FRMCT1  = 0xB1
FRMCT2  = 0xB2
FRMCT3  = 0xB3
INVCTR  = 0xB4
DISSET  = 0xB6
PWRCT1  = 0xC0
PWRCT2  = 0xC1
PWRCT3  = 0xC2
PWRCT4  = 0xC3
PWRCT5  = 0xC4
VMCTR1  = 0xC5
PWRCT6  = 0xFC
GAMCTP  = 0xE0
GAMCTN  = 0xE1


#############################################################################
#
#   Low-level routines
#
#

def SetPin(pinNumber,value):
    #sets the GPIO pin to desired value (1=on,0=off)
    GPIO.output(pinNumber,value)

def InitGPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
```

```
    GPIO.setup(DC,GPIO.OUT)

def InitSPI():
    "returns an opened spi connection to device(0,0) in mode 0"
    spiObject = spidev.SpiDev()
    spiObject.open(0,0)
    spiObject.mode = 0
    return spiObject


#############################################################################
#
#   ST7735 TFT controller routines:
#
#

def WriteByte(value):
    "sends an 8-bit value to the display as data"
    SetPin(DC,1)
    spi.writebytes([value])

def WriteWord (value):
    "sends a 16-bit value to the display as data"
    SetPin(DC,1)
    spi.writebytes([value>>8, value&0xFF])

def Command(cmd, *bytes):
    "Sends a command followed by any data it requires"
    SetPin(DC,0)                    #command follows
    spi.writebytes([cmd])           #send the command byte
    if len(bytes)>0:                #is there data to follow command?
        SetPin(DC,1)                #data follows
        spi.writebytes(list(bytes)) #send the data bytes

def OldInitDisplay():
    "Resets & prepares display for active use."
    Command (SWRESET)               #reset TFT controller
    time.sleep(0.2)                 #wait 200mS for controller init
    Command (SLPOUT)                #wake from sleep
    Command (COLMOD, 0x05)          #set color mode to 16 bit
    Command (DISPON)                #turn on display

def InitDisplay (fullReset=False):
    "Resets & prepares display for active use."
    Command (SWRESET)               #reset TFT controller
    time.sleep(0.2)                 #wait 200mS for controller init
    Command (SLPOUT)                #wake from sleep
    Command (COLMOD, 0x05)          #set color mode to 16 bit
    if fullReset:
        Command (FRMCT1, 0x01, 0x2C, 0x2D)
        Command (FRMCT2, 0x01, 0x2C, 0x2D)
        Command (FRMCT3, 0x01, 0x2C, 0x2D, 0x01, 0x2C, 0x2D)
        Command (INVCTR, 0x07)
        Command (PWRCT1, 0xA2, 0x02, 0x84)
        Command (PWRCT2, 0xC5)
        Command (PWRCT3, 0x0A, 0x00)
        Command (PWRCT4, 0x8A, 0x2A)
        Command (PWRCT5, 0x8A, 0xEE)
        Command (VMCTR1, 0x0E)
        Command (MADCTL, 0x00)
        Command (GAMCTP, 0x02, 0x1C, 0x07, 0x12, 0x37, 0x32, 0x29, 0x2D,
                         0x29, 0x25, 0x2B, 0x39, 0x00, 0x01, 0x03, 0x10)
        Command (GAMCTN, 0x03, 0x1D, 0x07, 0x06, 0x2E, 0x2C, 0x29, 0x2D,
                         0x2E, 0x2E, 0x37, 0x3F, 0x00, 0x00, 0x02, 0x10)
```

```
        Command (NORON)
    Command (DISPON)                #turn on display

def SetOrientation(degrees):
    "Set the display orientation to 0,90,180,or 270 degrees"
    if   degrees==90:  arg=0x60
    elif degrees==180: arg=0xC0
    elif degrees==270: arg=0xA0
    else:              arg=0x00
    Command (MADCTL,arg)

def SetAddrWindow(x0,y0,x1,y1):
    "sets a rectangular display window into which pixel data is placed"
    Command (CASET,0,x0,0,x1)       #set column range (x0,x1)
    Command (RASET,0,y0,0,y1)       #set row range (y0,y1)

def WriteBulk (value,reps,count=1):
    "sends a 16-bit pixel word many, many times using hardware SPI"
    "number of writes = reps * count. Value of reps must be <= 2048"
    SetPin(DC,0)                    #command follows
    spi.writebytes([RAMWR])         #issue RAM write command
    SetPin(DC,1)                    #data follows
    valHi = value >> 8              #separate color into two bytes
    valLo = value & 0xFF
    byteArray = [valHi,valLo]*reps  #create buffer of multiple pixels
    for a in range(count):
        spi.writebytes(byteArray)   #send this buffer multiple times

def WritePixels (byteArray):
    "sends pixel data to the TFT"
    SetPin(DC,0)                    #command follows
    spi.writebytes([RAMWR])         #issue RAM write command
    SetPin(DC,1)                    #data follows
    spi.writebytes(byteArray)       #send data to the TFT


#########################################################################
#
#   Graphics routines:
#
#

def DrawPixel(x,y,color):
    "draws a pixel on the TFT display"
    SetAddrWindow(x,y,x,y)
    Command (RAMWR, color>>8, color&0xFF)

def SetPixel(x,y,color):
    "draws a pixel on the TFT display; increases speed by inlining"
    GPIO.output(DC,0)
    spi.writebytes([CASET])
    GPIO.output(DC,1)
    spi.writebytes([0,x,0,x])
    GPIO.output(DC,0)
    spi.writebytes([RASET])
    GPIO.output(DC,1)
    spi.writebytes([0,y,0,y])
    GPIO.output(DC,0)
    spi.writebytes([RAMWR])
    GPIO.output(DC,1)
    spi.writebytes([color>>8, color&0xFF])

def HLine (x0,x1,y,color):
    "draws a horizontal line in given color"
```

```
    width = x1-x0+1
    SetAddrWindow(x0,y,x1,y)
    WriteBulk(color,width)

def VLine (x,y0,y1,color):
    "draws a verticle line in given color"
    height = y1-y0+1
    SetAddrWindow(x,y0,x,y1)
    WriteBulk(color,height)

def Line (x0,y0,x1,y1,color):
    "draws a line in given color"
    if (x0==x1):
        VLine(x0,y0,y1,color)
    elif (y0==y1):
        HLine(x0,x1,y0,color)
    else:
        slope = float(y1-y0)/(x1-x0)
        if (abs(slope)< 1):
            for x in range(x0,x1+1):
                y = (x-x0)*slope + y0
                SetPixel(x,int(y+0.5),color)
        else:
            for y in range(y0,y1+1):
                x = (y-y0)/slope + x0
                SetPixel(int(x+0.5),y,color)

def DrawRect(x0,y0,x1,y1,color):
    "Draws a rectangle in specified color"
    HLine(x0,x1,y0,color)
    HLine(x0,x1,y1,color)
    VLine(x0,y0,y1,color)
    VLine(x1,y0,y1,color)

def FillRect(x0,y0,x1,y1,color):
    "fills rectangle with given color"
    width = x1-x0+1
    height = y1-y0+1
    SetAddrWindow(x0,y0,x1,y1)
    WriteBulk(color,width,height)

def FillScreen(color):
    "Fills entire screen with given color"
    FillRect(0,0,XMAX,YMAX,color)

def ClearScreen():
    "Fills entire screen with black"
    FillScreen(BLACK)              #alternative: bColor for background

def Circle(x0,y0,radius,color=fColor):
    "draws circle at x0,y0 with given radius & color"
    xEnd = int(0.7071*radius)+1
    for x in range(xEnd):
        y = int(sqrt(radius*radius - x*x))
        SetPixel(x0+x,y0+y,color)
        SetPixel(x0+x,y0-y,color)
        SetPixel(x0-x,y0+y,color)
        SetPixel(x0-x,y0-y,color)
        SetPixel(x0+y,y0+x,color)
        SetPixel(x0+y,y0-x,color)
        SetPixel(x0-y,y0+x,color)
        SetPixel(x0-y,y0-x,color)

def FastCircle(x0,y0,radius,color=fColor):
```

```
    "draws circle at x0,y0 with given radius & color"
    "using the 'Midpoint Circle Algoritm'"
    x = radius
    y = 0
    radiusError = 1-x
    while x>=y:
        SetPixel(x0+x,y0+y,color)
        SetPixel(x0+x,y0-y,color)
        SetPixel(x0-x,y0+y,color)
        SetPixel(x0-x,y0-y,color)
        SetPixel(x0+y,y0+x,color)
        SetPixel(x0+y,y0-x,color)
        SetPixel(x0-y,y0+x,color)
        SetPixel(x0-y,y0-x,color)
        y += 1
        if radiusError < 0:
            radiusError += 2*y + 1
        else:
            x -= 1
            radiusError += 2*(y-x+1)


def FillCircle(x0,y0,radius,color):
    "draws filled circle at x,y with given radius & color"
    r2 = radius * radius
    for x in range(radius):
        y = int(sqrt(r2-x*x))
        VLine(x0+x,y0-y,y0+y,color)
        VLine(x0-x,y0-y,y0+y,color)

def Ellipse(x0,y0,width,height,color=fColor):
    "draws an ellipse of given width & height"
    "two-part Bresenham method"
    a = width/2                     #divide by 2, for x-radius
    b = height/2                    #divide by 2, for y-radius
    a2 = a * a
    b2 = b * b

    x=0; y=b
    sigma = 2*b2 + a2*(1-2*b)
    while  b2*x <= a2*y:
        SetPixel(x0+x,y0+y,color)
        SetPixel(x0+x,y0-y,color)
        SetPixel(x0-x,y0+y,color)
        SetPixel(x0-x,y0-y,color)
        if sigma >=0:
            sigma += 4*a2*(1-y)
            y -= 1
        sigma += b2*(4*x + 6)
        x += 1
    x=a; y=0
    sigma = 2*a2 + b2*(1-2*a)
    while a2*y <= b2*x:
        SetPixel(x0+x,y0+y,color)
        SetPixel(x0+x,y0-y,color)
        SetPixel(x0-x,y0+y,color)
        SetPixel(x0-x,y0-y,color)
        if sigma >= 0:
            sigma += 4*b2*(1-x)
            x -= 1
        sigma += a2*(4*y + 6)
        y += 1

def VBar(x,y,width,height,prevHt=0,color=fColor):
```

```
    "draws a vertical bar of given height & color"
    if height>0:
        if prevHt==0:
            FillRect(x,y-height,x+width,y,color)
        elif height>prevHt:
            FillRect(x,y-height,x+width,y-prevHt,color)
        else:
            FillRect(x,y-prevHt,x+width,y-height,bColor)

def FillEllipse(xPos,yPos,width,height, color):
    "draws a filled ellipse of given width & height"
    a    = width/2                  #divide by 2, for x-radius
    b    = height/2                 #divide by 2, for y-radius
    b2   = b * b
    a2   = a * a
    a2b2 = a2 * b2
    x0   = a
    dx   = 0
    HLine(xPos-a,xPos+a,yPos,color)
    for y in range(1,b+1):
        x1 = x0 - (dx-1)
        for x1 in range(x1,0,-1):
            if x1*x1*b2 + y*y*a2 <= a2b2:
                break
        dx=x0-x1
        x0 = x1
        HLine(xPos-x0,xPos+x0,yPos+y,color)
        HLine(xPos-x0,xPos+x0,yPos-y,color)


##########################################################################
#
#   Text routines:
#
#

def GetCharData (ch):
    "Returns array of raster data for a given ASCII character"
    pIndex = ord(ch)-ord(' ')
    lastDescriptor = len(font.descriptor)-1
    charIndex = font.descriptor[pIndex][1]
    if (pIndex >= lastDescriptor):
        return font.rasterData[charIndex:]
    else:
        nextIndex = font.descriptor[pIndex+1][1]
        return font.rasterData[charIndex:nextIndex]

def GetCharWidth(ch):
    "returns the width of a character, in pixels"
    pIndex = ord(ch)-ord(' ')
    return font.descriptor[pIndex][0]

def GetCharHeight(ch):
    "returns the height of a character, in pixels"
    return font.fontInfo[0]

def GetStringWidth(st):
    "returns the width of the text, in pixels"
    width = 0
    for ch in st:
        width += GetCharWidth(ch) + 1
    return width

def PutCh (ch,xPos,yPos,color=fColor):
```

```
    "write ch to X,Y coordinates using ASCII 5x7 font"
    charData = font0.data[ord(ch)-32]
    SetAddrWindow(xPos,yPos,xPos+4,yPos+6)
    SetPin(DC,0)
    spi.writebytes([RAMWR])
    SetPin(DC,1)
    buf = []
    mask = 0x01
    for row in range(7):
        for col in range(5):
            bit = charData[col] & mask
            if (bit==0):
                pixel = bColor
            else:
                pixel = color
            buf.append(pixel>>8)
            buf.append(pixel&0xFF)
        mask <<= 1
    spi.writebytes(buf)

def OldPutChar (ch,xPos,yPos,color=fColor):
    "Writes Ch to X,Y coordinates in current foreground color"
    charData = GetCharData(ch)
    xLen = GetCharWidth(ch)
    numRows = GetCharHeight(ch)
    bytesPerRow = 1+((xLen-1)/8)
    numBytes = numRows*bytesPerRow
    SetAddrWindow(xPos,yPos,xPos+xLen-1,yPos+numRows-1)
    SetPin(DC,0)
    spi.writebytes([RAMWR])
    SetPin(DC,1)
    i= 0
    while (i< numBytes):
        mask = 0x01
        rowBits = 0
        for b in range(bytesPerRow):
            rowBits <<= 8
            mask    <<= 8
            rowBits +=  charData[i]
            i +=  1
        mask >>= 1
        #at this point, all bits for current row should
        #be in rowBits, regardless of number of bytes
        #it takes to represent the row

        rowBuf = []
        for a in range(xLen):
            bit = rowBits & mask
            mask >>= 1
            if (bit==0):
                pixel = bColor
            else:
                pixel = color
            rowBuf.append(pixel>>8)
            rowBuf.append(pixel&0xFF)
        spi.writebytes(rowBuf)

def PutChar (ch,xPos,yPos,color=fColor):
    "Writes Ch to X,Y coordinates in current foreground color"
    charData = GetCharData(ch)
    xLen = GetCharWidth(ch)                    #char width, in pixels
    numRows = GetCharHeight(ch)
    bytesPerRow = 1+((xLen-1)/8)               #char width, in bytes
    numBytes = numRows*bytesPerRow
```

```
    SetAddrWindow(xPos,yPos,xPos+xLen-1,yPos+numRows-1)
    SetPin(DC,0)
    spi.writebytes([RAMWR])                    #pixel data to follow
    SetPin(DC,1)
    index = 0
    buf = []
    for a in range(numRows):                   #row major
        bitNum = 0
        for b in range(bytesPerRow):           #do whole row
            mask = 0x80                        #msb first
            for c in range(8):                 #do all bits in this byte
                if (bitNum<xLen):              #still within char width?
                    bit = charData[index] & mask
                    if (bit==0):               #check the bit
                        pixel = bColor         #0: background color
                    else:
                        pixel = color          #1: foreground color
                    buf.append(pixel>>8)       #add pixel to buffer
                    buf.append(pixel&0xFF)
                    mask >>= 1                 #goto next bit in byte
                bitNum += 1                    #goto next bit in row
            index += 1                         #goto next byte of data
    spi.writebytes(buf)                        #send char data to TFT

def PutString(xPos,yPos,st,color=fColor):
    "Draws string on display at xPos,yPos."
    "Does NOT check to see if it fits!"
    for ch in st:
        width = GetCharWidth(ch)+1
        PutChar(ch,xPos,yPos,color)
        xPos += width

def PutSt(st,x,y,c=fColor):
    "draws string using 5x7 font at pos x,y in color c"
    "Does NOT check to see if it fits!"
    for ch in st:
        PutCh(ch,x,y,c)
        x += 6

def CenterText(st,yPos,color=fColor):
    "Centers the text horizontally at given vertical position"
    xPos = (XSIZE - GetStringWidth(st))/2
    PutString(xPos,yPos,st,color)


#############################################################################
#
#   Testing routines:
#
#

def PrintElapsedTime(function,startTime):
    "Formats an output string showing elapsed time since function start"
    elapsedTime=time.time()-startTime
    print "%15s: %8.3f seconds" % (function,elapsedTime)
    time.sleep(1)

def LabelIt(name):
    "write test name to screen"
    ClearScreen()
    CenterText(name,50)
    CenterText('Test',70)
    time.sleep(1)
    ClearScreen()
```

```
def ColorBars():
    "Fill Screen with 8 color bars"
    for a in range(8):
        FillRect(0,a*20,XMAX,a*20+19,COLORSET[a+1])

def ScreenTest():
    "Measures time required to fill display twice"
    LabelIt('Screen')
    startTime=time.time()
    FillScreen(LIME)
    FillScreen(MAGENTA)
    ColorBars()
    PrintElapsedTime('ScreenTest',startTime)

def RandRect():
    "Returns four integers x0,y0,x1,y1 as screen rect coordinates"
    x1 = randint(1,100)
    y1 = randint(1,150)
    dx = randint(30,80)
    dy = randint(30,80)
    x2 = x1 + dx
    if x2>126:
        x2 = 126
    y2 = y1 + dy
    if y2>158:
        y2 = 158
    return x1,y1,x2,y2

def RandColor():
    "Returns a random color from BGR565 Colorspace"
    index = randint(1,len(COLORSET)-1)
    return COLORSET[index]

def RectTest(numCycles=50):
    "Draws a series of random open rectangles"
    ClearScreen()
    LabelIt('Rectangle')
    startTime = time.time()
    for a in range(numCycles):
        x0,y0,x1,y1 = RandRect()
        DrawRect(x0,y0,x1,y1,RandColor())
    PrintElapsedTime('RectTest',startTime)

def FillRectTest(numCycles=70):
    "draws random filled rectangles on the display"
    startTime=time.time()
    ClearScreen()
    for a in range(numCycles):
        x0,y0,x1,y1=RandRect()
        FillRect(x0,y0,x1,y1,RandColor())
    PrintElapsedTime('FillRect',startTime)

def RectTest2():
    "draw concentric rectangles on the display"
    ClearScreen()
    midX=63; midY=75
    for a in range(5,60,2):
        DrawRect(midX-a,midY-a,midX+a,midY+a,YELLOW)
    time.sleep(1)
    for a in range(60,5,-3):
        DrawRect(midX-a,midY-a,midX+a,midY+a,CYAN)
    time.sleep(1)
    for a in range(5,60,4):
```

```
            DrawRect(midX-a,midY-a,midX+a,midY+a,BLUE)
        time.sleep(1)
        for a in range(60,5,-2):
            DrawRect(midX-a,midY-a,midX+a,midY+a,MAGENTA)
        time.sleep(1)

    def LineTest(numCycles=50):
        "Draw a series of semi-random lines on display"
        ClearScreen()
        LabelIt('Line')
        startTime=time.time()
        for a in range(numCycles):
            Line(10,10,randint(20,126),randint(20,158),YELLOW)
            Line(120,10,randint(2,126),randint(10,158),CYAN)
        PrintElapsedTime('LineTest',startTime)

    def LineTest2():
        "Draw a series of semi-random lines on display"
        ClearScreen()
        LabelIt('Line')
        startTime=time.time()
        for a in range(0,159,5):
            Line(0,0,127,a,YELLOW)
        for a in range(126,0,-5):
            Line(0,0,a,159,YELLOW)
        for a in range(159,0,-5):
            Line(0,a,127,159,CYAN)
        for a in range(0,127,5):
            Line(a,0,127,159,CYAN)
        PrintElapsedTime('LineMoire',startTime)


    def PixelTest(color=BLACK, numPixels=5000):
        "Writes random pixels to the screen"
        ClearScreen()
        LabelIt('Pixel')
        startTime = time.time()
        for i in range(numPixels):
            xPos = randint(0,XMAX)
            yPos = randint(0,YMAX)
            DrawPixel(xPos,yPos,LIME)
        PrintElapsedTime('PixelTest',startTime)

    def FastPixelTest(color=BLACK, numPixels=5000):
        "Writes random pixels to the screen"
        ClearScreen()
        LabelIt('Pixel')
        startTime = time.time()
        for i in range(numPixels):
            xPos = randint(0,XMAX)
            yPos = randint(0,YMAX)
            SetPixel(xPos,yPos,YELLOW)
        PrintElapsedTime('FastPixelTest',startTime)

    def CircleMoireTest():
        "Draws a series of concentric circles"
        ClearScreen()
        startTime = time.time()
        for radius in range(6,60,2):
            Circle(X0,Y0,radius,YELLOW)
        PrintElapsedTime('CircleMoire',startTime)

    def CircleTest(numCycles=20):
        "draw a series of random circles"
```

```
    title = 'Circle'
    LabelIt(title)
    startTime = time.time()
    for a in range(numCycles):
        x = randint(30,90)
        y = randint(30,130)
        radius = randint(10,40)
        Circle(x,y,radius,RandColor())
    PrintElapsedTime(title,startTime)

def FastCircleTest(numCycles=20):
    "draw a series of random circles"
    title = 'Fast Circle'
    LabelIt(title)
    startTime = time.time()
    for a in range(numCycles):
        x = randint(30,90)
        y = randint(30,130)
        radius = randint(10,40)
        FastCircle(x,y,radius,RandColor())
    PrintElapsedTime(title,startTime)

def FillCircleTest(numCycles=20):
    "draw a series of random filled circles"
    ClearScreen()
    startTime = time.time()
    for a in range(numCycles):
        x = randint(30,90)
        y = randint(30,130)
        radius = randint(10,40)
        FillCircle(x,y,radius,RandColor())
    PrintElapsedTime('FillCircleTest',startTime)

def EllipseTest():
    title = 'Ellipse'
    LabelIt(title)
    startTime = time.time()
    for height in range(10,150,10):
        Ellipse(X0,Y0,60,height,CYAN)
    for width in range(10,120,10):
        Ellipse(X0,Y0,width,60,YELLOW)
    PrintElapsedTime(title,startTime)

def FillEllipseTest():
    title = 'Fill Ellipse'
    startTime = time.time()
    for height in range(155,10,-10):
        width = int(height*0.8)
        FillEllipse(X0,Y0,width,height,RandColor())
    PrintElapsedTime(title,startTime)

def VBarTest():
    title = 'VBar'
    LabelIt(title)
    startTime = time.time()
    data = [0,0,0,0]
    for cycle in range(10):
        for col in range(4):
            xPos = 15 + 25*col
            newValue = randint(30,150)
            VBar(xPos,150,15,newValue,data[col],GREEN)
            data[col] = newValue
        time.sleep(1)
    PrintElapsedTime(title,startTime)
```

```
def WaveTest():
    "similate wave motion with animated vertical bar graph"
    title = 'Wave'
    LabelIt(title)
    startTime = time.time()
    data = [0,0,0,0]
    for steps in range(40):
        for col in range(4):
            xPos = 15 + 25*col
            if col==3:
                radians = 6.28*(steps%12)/12
                newValue = int(40*sin(radians)+70)
            else:
                newValue = data[col+1]
            VBar(xPos,150,15,newValue,data[col],CYAN)
            data[col] = newValue
        time.sleep(0.2)
    PrintElapsedTime(title,startTime)

def SineTest():
    "draw a series of sin waves on screen"
    title = 'Sine Graph'
    LabelIt(title)
    startTime = time.time()
    VLine(0,0,150,CYAN)
    HLine(0,120,75,CYAN)
    for step in range(0,120,10):
        for pixels in range(360):
            degrees = (pixels + step)%360
            radians = 6.28*degrees/360.0
            val1 = sin(radians)
            val2 = cos(radians)
            x  = int(pixels/3)
            y1 = int(75*val1+75)
            y2 = int(75*val2+75)
            SetPixel(x,y1,YELLOW)
            SetPixel(x,y2,RED)
    PrintElapsedTime(title,startTime)
    time.sleep(1)

def RunGraphicsTests():
    "run a series of graphics test routines & time them"
    ScreenTest()                    #fill entire screen with color
    RectTest(40)                    #draw rectangles
    FillRectTest(25)                #draw filled rectangles
    RectTest2()                     #draw concentric squares
    FastPixelTest()                 #draw 5000 random pixels
    LineTest2()                     #draw series of lines
    CircleTest()                    #draw random circles
    CircleMoireTest()               #draw concentric circles
    FillEllipseTest()               #draw filled ellipses
    WaveTest()                      #draw animated wave
    SineTest()                      #draw y=sin(x) graphs

def PortaitChars():
    "Writes 420 characters (5x7) to screen in Portrait Mode"
    #font is 5x7 with 1 pixel spacing
    #so character width = 6 pixels, height = 8 pixels
    #display width = 128 pixels, so 21 char per row (21x6 = 126)
    #display ht = 160 pixels, so 20 rows (20x8 = 160)
    #total number of characters = 21 x 20 = 420

    CHARS_PER_ROW = 21
```

```
    FillRect(0,0,XMAX,YMAX,bColor)  #clear screen
    for i in range(420):
        x= i % CHARS_PER_ROW
        y= i / CHARS_PER_ROW
        ascii = (i % 96)+32
        PutCh(chr(ascii),x*6,y*8)
    time.sleep(1)

def LandscapeChars():
    "Writes 416 characters (5x7) to screen, landscape mode"
    #font is 5x7 with 1 pixel spacing
    #so character width = 6 pixels, height = 8 pixels
    #display width = 160 pixels, so 26 char per row (26x6 = 156)
    #display ht = 128 pixels, so 16 rows (16x8 = 128)
    #total number of characters = 26 x 16 = 416

    CHARS_PER_ROW = 26
    FillRect(0,0,YMAX,XMAX,bColor)  #clear screen
    for i in range(416):
        x= i % CHARS_PER_ROW
        y= i / CHARS_PER_ROW
        ascii = (i % 96)+32
        PutCh(chr(ascii),x*6,y*8,CYAN)
    time.sleep(1)

def LargeFontTest():
    "Writes 90 characters (11x17) to the screen"
    title = 'Large Font'
    LabelIt(title)
    startTime = time.time()
    for i in range(90):
        x= i % 10
        y= i / 10
        ascii = (i % 96)+32
        PutChar(chr(ascii),x*12,y*18,LIME)
    PrintElapsedTime(title,startTime)

def SmallFontTest():
    "Writes 1000 random 5x7 characters to the screen"
    title = 'Small Font'
    LabelIt(title)
    startTime = time.time()
    for i in range(1000):
        x= randint(0,20)
        y= randint(0,19)
        color = RandColor()
        ascii = (i % 96)+32
        PutCh(chr(ascii),x*6,y*8,color)
    PrintElapsedTime(title,startTime)

def OrientationTest():
    "Write 5x7 characters at 0,90,180,270 deg orientations"
    title = 'Orientation'
    startTime = time.time()
    LabelIt(title)
    PortaitChars()
    SetOrientation(90)              #display-top on right
    LandscapeChars()
    SetOrientation(180)             #upside-down
    PortaitChars()
    SetOrientation(270)             #display-top on left
    LandscapeChars()
    SetOrientation(0)               #return to 0 deg.
    PrintElapsedTime(title,startTime)
```

```
def GetTempCPU():
    "Returns CPU temp in degrees F"
    tPath = '/sys/class/thermal/thermal_zone0/temp'
    tFile = open(tPath)
    temp = tFile.read()
    tFile.close()
    return (float(temp)*0.0018 + 32)

def Run(cmd):
    "Runs a system (bash) command"
    return os.popen(cmd).read()

def GetIPAddr():
    "Returns IP address as a string"
    cmd = "ifconfig | awk '/192/ {print $2}'"
    res = Run(cmd).replace('addr:','')
    return res.replace('\n','')

def InfoTest():
    "Show Time on screen"
    title = 'Info'
    LabelIt(title)
    startTime = time.time()         #keep track of test duration
    PutString(0,0,'IP addr',WHITE)
    PutString(0,20,GetIPAddr())
    PutString(0,60,'CPU temp',WHITE)
    temp = GetTempCPU()
    PutString(0,80,'{:5.1f} deg F'.format(temp))
    tStr = time.strftime("%I:%M:%S  ")
    PutString(0,120,'Time',WHITE)
    PutString(0,140,tStr)
    PrintElapsedTime(title,startTime)

def RunTextTests():
    LargeFontTest()
    SmallFontTest()
    OrientationTest()
    InfoTest()

######################################################################
#
#   Color routines:
#
#

def PackColor(red,green,blue):
    "Packs individual red,green,blue color components into 16bit color"
    "16-bit color format is rrrrrggg.gggbbbbb"
    "Max values: Red 31, Green 63, Blue 31"
    r = red & 0x1F
    g = green & 0x3F
    b = blue & 0x1F
    return (r<<11)+(g<<5)+b

def UnpackColor(color):
    "Reduces 16-bit color into component r,g,b values"
    r = color>>11
    g = (color & 0x07E0)>>5
    b = color & 0x001F
    return r,g,b

def Brightness(color):
    "returns brightness of color pixel: 0=black,99=white"
```

```
    r,g,b = UnpackColor(color)
    return int(0.94*r + 0.94*g + 0.31*b)

def Complement(color):
    "Returns color complement"
    r,g,b = UnpackColor(color)
    return PackColor(31-r,63-g,31-b)

def TriadColors(color):
    "Returns two triad colors for given color"
    r,g,b = UnpackColor(color)
    tr1 = PackColor(b,r*2,g/2)   #rgb --> brg
    tr2 = PackColor(g/2,b*2,r)   #rgb --> gbr
    return tr1,tr2

def TriadDisplay(c1):
    "Displays color triad & their compliments"
    c2,c3 = TriadColors(c1)
    comp1 = Complement(c1)
    comp2 = Complement(c2)
    comp3 = Complement(c3)
    dy = 40
    FillRect(1,0+dy,42,41+dy,c2)
    #FillRect(43,0+dy,84,41+dy,c1)
    FillRect(85,0+dy,126,41+dy,c3)
    FillRect(1,42+dy,42,83+dy,comp2)
    FillRect(43,42+dy,84,83+dy,comp1)
    FillRect(85,42+dy,126,83+dy,comp3)

def ColorDictionary(filename):
    'loads color dictionary from a file'
    #the file contains a list of named colors, one per line
    #each line is of the format COLORNAME = 0xFFFF
    dict = {}
    for line in open(filename):         #scan each line of file
        if line.find('=')>0:            #ignore lines without '='
            words = line.split('=')     #split line into two words
            key = words[0].strip()      #first word is the key (colorname)
            value = words[1].strip()    #second word is the RGB value
            dict[key] = int(value,16)   #add pair to dictionary, converting to hex
    return dict

def ColorInputTest():
    'Ask user of R,G,B compononents (0-255); Display color on screen'
    'To quit, enter value of >255 for red'
    ClearScreen()
    while True:
        r = eval(raw_input('red? '))
        if r>255: break
        g = eval(raw_input('green? '))
        b = eval(raw_input('blue? '))
        r>>=3; g>>=2; b>>=3
        color = PackColor(r,g,b)
        FillRect(0,0,127,128,color)
        st = "({0:2d},{1:2d},{2:2d}) = {3:4x}  ".format(r,g,b,color)
        PutSt(st,10,140,WHITE)

def ColorTest():
    "Loads a color dictionary; displays sample colors w/ complements"
    global bColor
    LabelIt('Color')

    #load a dictionary with all of the color names & RGB values
    dict = ColorDictionary('rgb565.py')
```

```
    #demo with just a few colors
    colorNames = ['AQUA','ORANGE','MAGENTA','SKYBLUE','TOMATO','NAVY']
    for name in colorNames:
        color = dict[name]
        CUTOFF = 65     #65 for gamma corrected, 30 for uncorrected
        bColor = color

        #display color and it's name
        FillRect(0,0,XSIZE,XSIZE,color) #set background to color
        length = len(name)
        while GetStringWidth(name[0:length])>XSIZE:
            length -= 1
        abbrev = name[0:length]         #trunc name to fit on screen
        b  = Brightness(color)          #get brightness of this color
        if Brightness(color)<CUTOFF:
            c1 = WHITE                  #dark colors get white text
        else:
            c1 = BLACK                  #light colors get dark text
        CenterText(abbrev,20,c1)        #display color name
        CenterText(str(b),40,c1)        #display brightness
        bColor = BLACK                  #restore background color

        #display RGB component information at bottom of display
        r,g,b = UnpackColor(color)
        st = "({0:2d},{1:2d},{2:2d}) = {3:4x}  ".format(r,g,b,color)
        PutSt(st,10,140,WHITE)
        st = "{:20s}".format(name)
        PutSt(st,10,150,WHITE)
        time.sleep(1)

        #display triad colors & their complements
        TriadDisplay(color)
        time.sleep(1)


##########################################################################
#
#   BMP Routines  -  Read .bmp Graphics File Format
#

INT4  = '<l'                    #get 4-byte word from unpacking routine
INT2  = '<h'                    #get 2-byte word from unpacking routine

def DrawBMP(fileName, displayHeader=False):
    ClearScreen()

    #read entire file into list variable 'data'
    data = open(fileName,'rb').read()
    signature = data[0:2]
    if signature != 'BM':
        print 'Not a bitmap file'
        return

    #Get information from BMP header
    fileSize  = unpack_from(INT4,data,02)[0]
    offset    = unpack_from(INT4,data,10)[0]
    width     = unpack_from(INT4,data,18)[0]
    height    = unpack_from(INT4,data,22)[0]
    numPlanes = unpack_from(INT2,data,26)[0]
    colorBits = unpack_from(INT2,data,28)[0]
    compType  = unpack_from(INT4,data,30)[0]
    imageSize = unpack_from(INT4,data,34)[0]
    horizRes  = unpack_from(INT4,data,38)[0]
```

```
    vertRes   = unpack_from(INT4,data,42)[0]

    #optionally display header information
    if displayHeader:
        print 'Filename    ',fileName
        print 'Signature   ',signature
        print 'File Size   ',fileSize
        print 'Offset      ',offset
        print 'Width       ',width
        print 'Height      ',height
        print 'Planes      ',numPlanes
        print 'ColorBits   ',colorBits
        print 'Compression ',compType
        print 'ImageSize   ',imageSize
        print 'Horiz Res   ',horizRes
        print 'Vert Res    ',vertRes

    if (numPlanes!=1) or (colorBits!=24) or (compType!=0):
        print 'Unsupported BMP format'
        return
    rowSize = (width*3 + 3) & ~3

    #send pixel data from file to TFT
    SetAddrWindow(0,0,width-1,height-1)
    SetPin(DC,0)
    spi.writebytes([RAMWR])           #pixel data to follow
    SetPin(DC,1)

    for row in range(height):         #do one row at a time
        buf = []                      #buffer for row of pixels
        index = (height-row-1)*rowSize + offset
        for col in range(width):      #for each pixel in this row:
            b = ord(data[index])      #get blue subpixel (0-255)
            g = ord(data[index+1])    #get green subpixel (0-255)
            r = ord(data[index+2])    #get red subpixel (0-255)
            pixel = PackColor(r>>3,g>>2,b>>3)  #pack into RGB565 format
            buf.append(pixel>>8)      #add pixel to buffer (MSB, LSB)
            buf.append(pixel&0xFF)
            index += 3                #go to next pixel in data stream
        spi.writebytes(buf)           #send row of pixel data to TFT


##############################################################################
#
#   Gamma routines:
#
#

def SetGamma():
    'Sets Gamma table with manufacturer recommended values'
    'These values are very close to a Gamma of 2.2'
    Command (GAMCTP, 0x02, 0x1C, 0x07, 0x12, 0x37, 0x32, 0x29, 0x2D,
                     0x29, 0x25, 0x2B, 0x39, 0x00, 0x01, 0x03, 0x10)
    Command (GAMCTN, 0x03, 0x1D, 0x07, 0x06, 0x2E, 0x2C, 0x29, 0x2D,
                     0x2E, 0x2E, 0x37, 0x3F, 0x00, 0x00, 0x02, 0x10)

def GamSet(value,text):
    'Set gamma & show text label at bottom of screen'
    'Called by CycleGammas routine'
    FillRect(0,150,127,159,BLACK)
    PutSt(text,20,152,WHITE)
    Command(GAMSET,value)
    time.sleep(2)
```

```
def CycleGammas(numCycles=3):
    'Cycle display through four different gamma settings'
    for count in range(numCycles):
        GamSet(1,'Gamma 1.0')
        GamSet(2,'Gamma 2.5')
        GamSet(4,'Gamma 2.2')
        GamSet(8,'Gamma 1.8')

def GradientFill():
    "Draw a vertical color gradient fill on TFT, bright to dark green"
    ClearScreen()
    for i in range(64):
        FillRect(0,2*i,127,2*i+1,PackColor(0,63-i,0))
    DrawRect(0,0,127,128,GREEN)

def BandedFill():
    "Draws Black-to-white bars, left to right, in 10 equal steps"
    ClearScreen()
    for i in range(10):
        FillRect(12*i,0,12*i+11,128,PackColor(i*3,i*6,i*3))
    DrawRect(0,0,120,128,GRAY)

    #this test shows that GamSet sets up the gamma table to reflect
    #the desired gamma value.  If you set your gamma table then call
    #GamSet, your table is wiped out.  If you call GamSet then set your
    #values, the GamSet is wiped out.  There is only one table; not
    #four separate tables for each gamma setting.
    #Adafruits gamma settings is very close to Gamset(2).

def RunGammaTests():
    "Display gradients & images with different gamma curves"
    ClearScreen()
    LabelIt('Gamma')
    BandedFill()
    CycleGammas(1)
    GradientFill()
    CycleGammas(1)
    DrawBMP('parrot.bmp')
    CycleGammas(1)
    time.sleep(1)



########################################################################
#
#   Main Program
#

print "Adafruit 1.8 TFT display demo with hardware SPI"
spi = InitSPI()                     #initialize SPI interface
InitGPIO()                          #initialize GPIO interface
InitDisplay(True)                   #initialize TFT controller
RunTextTests()                      #run suite of text tests
RunGraphicsTests()                  #run suite of graphic tests
ColorTest()                         #display named colors & complements
RunGammaTests()                     #run suite of gamma tests
DrawBMP('sarah.bmp')                #end with my daughter!
spi.close()                         #close down SPI interface
print "Done."


#   END  ###############################################################
```