**Outline**

Develop an understanding of how images and graphics are drawn and stored in a computer. Learn about the RGB colour space. Apply Python concepts related to lists and loops.

**Objectives**

* tbd

**Materials**

* tbd

**Level 1: Pixels & RGB**

1. Create a new Repl for Python with Turtle.
   1. Copy and paste “Sample Program #1” from the listing at the end of this module.
   2. Run the program and examine the Turtle output
2. Colours can be specified by using a combination of three numbers. These three numbers together define a “Pixel” point in a graphic image.
   1. What position is the number that controls the amount of red (r) in the pixel?

The first one would be r.

* 1. What position is the number that controls the amount of green (g) in the pixel?

The second one will be green.

* 1. What position is the number that controls the amount of blue (b) in the pixel?  
     The third one will be blue.

1. Colour number values can range from 0 to 255.
   1. What happens when the colour value is less than 255?

It turns out to appear darker.

* 1. What happens when the colour value is close to 0?

There is almost no type of color it looks black but still isn’t as long it is not 0

1. Other shades of colours can be created using a combination of r,g,b number values.
   1. Create a pixel containing a shade of the colour orange.

redColor = (255,69,0)

* 1. Create a pixel containing a shade of the colour yellow.

redColor = (255,255,0)

* 1. Create a pixel containing a shade of your favorite colour.

Favorite color magenta

redColor = (255,0,255)

1. Black, white, and shades of grey are created using combinations of equal r,g,b number values.
   1. Create a completely white pixel.

redColor = (255,255,255)

* 1. Create a completely black pixel.

redColor = (0,0,0)

* 1. Create a pixel containing a shade of middle grey.

redColor = (124,124,124)

**Level 2: Images Using Pixels**

1. Download the image “Resoultion\_284x177.jpg” from Topic B folder in the class repository.
   1. Open the image in a program like Paint or Photoshop.
   2. What is the size of this image? How many pixels does it contain?

The size of the image is 284 pixels across and 177 pixels going vertical. The amount of pixels in the picture are 50,268.

* 1. Describe how the image looks (e.g. Can you see the pixels?)

If looking at it not zoomed in the pixels are not visible as there are so much.

* 1. Zoom in the view to enlarge the image
  2. Describe how the image looks (e.g. Can you see the pixels?)

Now the pixels are visible as small squares because I am zoomed into the image a lot making it easier to see the squares.

1. Download the image “Resoultion\_16x16.jpg” from Topic B folder in the class repository.
   1. Open the image in a program like Paint or Photoshop.
   2. What is the size of this image? How many pixels does it contain?

The image contains 16 pixels horizontally and vertically. There are 256 pixels

* 1. Describe how the image looks (e.g. Can you see the pixels?)

You can somewhat see the pixels as there are not a lot.

* 1. Zoom in the view to enlarge the image

e. Describe how the image looks (e.g. Can you see the pixels?)  
Now you can easily see the pixels more easier than before.

1. Create a new Repl for Python with Turtle.
2. Copy and paste “Sample Program #2” from the listing at the end of this module.
3. Run the program and examine the Turtle output
4. Compare the program output to the “Resoultion\_16x16.jpg” image in question #2 above.  
   They are similar when it comes to color scheme but the picture has more pixels than my python turtle as mine is 8 by 8 and the picture is 16 by 16.
5. Explain how the program code in lines 52 to 58 works. (i.e. The main program code.)
6. How the program prints out pixels to produce and 8 by 8 resolution image.

The program knows that it is an 8 by 8 resolution image because in on line 8 it clearly says each row contains 8 pixels. Also, for line 53 and 54 the code says for row in range 8 and for column in range 8 meaning it is telling the program to make the circles (pixels) 8 pixels in a row and 8 pixels in a column

1. How the program decides which colour information to use for each pixel.

The program decides this by looking at the RGB value per pixel. For example the first pixel has an RGB value of 15,15, and 5 meaning the colour is going to be dark and close to black and green but leaning more towards dark forest green.

It prints out an 8 by 8 as it is instructed 64 pixels and the code explains

1. Explain the purpose of the code in lines 12 to 21
2. How this code is related to the pixels produced by the main program.

It is related as it is one of the main things needed when producing a resolution image. The lines include provide the RGB which makes the color of the shape. Without it, there would be just a black or white image and this is the purpose to express and show the image.

1. The RGB value of the 19th pixel in the image

The RGB value of the 19th pixel is (28,28,12)

1. The RGB value of the pixel in the 5th column on the 4th row.

It would be the 29th pixel and the RGB value is (154,140,22)

1. Modify the main program to print the image at a resolution of 12 by 4 pixels.
2. Show your modified image to Mr. Nestor.
3. Explain your changes to the program code below.

The image is a lot more long vertically also the image is a lot more less recognizable as it is not equal and looks distorted. In order to change it to make it 12 by 4 is to change line 53 and 54. For line 53 instead of for row in range (8) it should be 12 and for line 54 it should be for column in range (4) instead of 8.

1. Modify the main program to print the image upside-down (i.e. pixels in reverse order).
2. Show your modified image to Mr. Nestor.
3. (47,93,23),(48,91,26),(32,76,17),(29,66,13),(66,103,36),(92,125,53), (57,109,33),(49,102,29),(32,80,12),(39,83,12),(28,87,3),(100,134,82), (216,204,146),(223,208,144),(71,105,42),(35,86,13),(42,98,17), (58,103,28),(41,99,17),(120,153,92),(255,225,140),(224,213,156),(63,104,39),(50,95,13),(43,85,18),(52,86,22),(44,91,15),(109,134,66),(213,195,123),(98,120,49),(48,94,18),(42,88,15),(19,22,5),(9,12,1),(28,43,10),(154,140,22),(156,139,26),(24,35,12),(35,47,15),(34,59,10),(16,11,3),(11,11,4),(42,42,14),(205,202,21),(191,192,18),(28,28,12),(16,13,8),(25,19,5),(21,28,10),(24,25,8),(11,8,6),(42,47,7),(58,65,11),(7,7,6),(20,18,9),(21,19,17),(25,42,17),(32,54,21),(33,52,22),(32,33,16),(23,21,10),(8,10,3),(13,13,6),(15,15,5)
4. Explain your changes to the program code below.

The changes to the program code is it is opposite. As well as the image looks the same but backwards it also looks the same if you turn your head around or flip the screen which is another cool aspect.

1. Modify the main program to print the image at a resolution of 12 by 4 pixels.
2. Show your modified image to Mr. Nestor.

[

**Level 3: Your Custom Image**

1. Use and modify the sample pixel program code to create your own custom image.
   1. Create a larger resolution image than provided in the sample.
   2. Make sure the image is recognizable (or a clear pattern).
   3. Show your image to Mr. Nestor.

My picture is a tiger here It is the size of the image I made was 10 by 8



1. List and explain your modified image code below.

My image was simple I used the same formatting Mr.Nestor did but changed the rows and columns to more pixels as well as changed the RGB to match my picture the code is below.

(218,234,94),(235,255,90),(154,151,61),(148,118,79),(201,176,95),(161,142,85),(149,122,64),(142,116,73),(209,212,129),(229,243,128),(232,242,121),(222,248,66),(211,218,54),(202,137,52),(160,119,67),(66,37,29),(64,27,15),(166,113,62),(228,234,115),(217,238,107),(255,255,217),(249,251,183),(255,255,158),(207,177,116),(187,184,175),(153,107,51),(144,106,65),(117,91,62),(200,204,82),(232,250,120),(214,229,116),(233,245,148),(237,231,210),(160,147,141),(180,146,91),(246,166,45),(146,84,16),(54,29,20),(151,145,88),(249,255,147),(198,216,99),(215,239,95),(216,217,194),(148,144,157),(238,197,126),(220,152,53),(132,71,4),(75,50,32),(151,146,132),(237,245,149),(220,230,130),(220,236,108),(242,233,208),(214,205,205),(214,203,185),(197,179,168),(147,119,82),(159,146,113),(181,163,128),(145,103,38),(221,243,113),(201,190,97),(221,188,165),(239,228,209),(166,150,127),(186,183,172),(130,122,100),(108,88,50),(102,61,18),(122,59,8),(229,241,126),(163,135,93),(202,183,165),(238,225,190),(151,126,82),(125,97,53),(92,55,14),(68,19,0),(82,32,0),(123,72,16)

**SAMPLE PROGRAM #1**

import turtle

myPen = turtle.Turtle()

# These variables track the position of the turtle pen

posX = 0

posY = 0

# This user defined function draws a single image pixel

def drawPixel(rgb) :

global posX

myPen.down()

myPen.color(rgb)

myPen.begin\_fill()

myPen.circle(8)

myPen.end\_fill()

myPen.up()

myPen.forward(18)

posX = posX + 18

# THE MAIN PROGRAM CODE STARTS HERE

#

redColor = (255,0,0)

drawPixel(redColor)

drawPixel((128,0,0))

greenColor = (0,255,0)

drawPixel(greenColor)

drawPixel((0,128,0))

blueColor = (0,0,266)

drawPixel(blueColor)

drawPixel((0,0,128))

**SAMPLE PROGRAM #2**

import turtle

myPen = turtle.Turtle()

# These variables track the position of the turtle pen

posX = 0

posY = 0

# These variables define the image information.

# Each pixel in the image has a (r,g,b) value

# The complete image is simply a list of pixels

pixelAddress = 0

pixelMemory = [

(15,15,5),(13,13,6),(8,10,3),(23,21,10),(32,33,16),(33,52,22),(32,54,21),(25,42,17),

(21,19,17),(20,18,9),(7,7,6),(58,65,11),(42,47,7),(11,8,6),(24,25,8),(21,28,10),

(25,19,5),(16,13,8),(28,28,12),(191,192,18),(205,202,21),(42,42,14),(11,11,4),(16,11,3),

(34,59,10),(35,47,15),(24,35,12),(156,139,26),(154,140,22),(28,43,10),(9,12,1),(19,22,5),

(42,88,15),(48,94,18),(98,120,49),(213,195,123),(109,134,66),(44,91,15),(52,86,22),(43,85,18),

(50,95,13),(63,104,39),(224,213,156),(255,225,140),(120,153,92),(41,99,17),(58,103,28),(42,98,17),

(35,86,13),(71,105,42),(223,208,144),(216,204,146),(100,134,82),(28,87,3),(39,83,12),(32,80,12),

(49,102,29),(57,109,33),(92,125,53),(66,103,36),(29,66,13),(32,76,17),(48,91,26),(47,93,23)

]

# This user defined function draws a single image pixel

def drawPixel(rgb) :

global posX

myPen.down()

myPen.color(rgb)

myPen.begin\_fill()

myPen.circle(8)

myPen.end\_fill()

myPen.up()

myPen.forward(18)

posX = posX + 18

# This user defined function starts a new row of pixels

def newRow() :

global posX

global posY

myPen.up()

myPen.left(180)

myPen.forward(posX)

myPen.left(90)

myPen.forward(18)

myPen.left(90)

myPen.down()

posX = 0

posY = posY + 18

# THE MAIN PROGRAM CODE STARTS HERE

#

# Draw eight rows of the image.

# Each row contains eight pixels

for row in range (8) :

for column in range(8) :

drawPixel(pixelMemory[pixelAddress])

pixelAddress += 1

newRow()

(47,93,23),(48,91,26),(32,76,17),(29,66,13),(66,103,36),(92,125,53), (57,109,33),(49,102,29),(32,80,12),(39,83,12),(28,87,3),(100,134,82), (216,204,146),(223,208,144),(71,105,42),(35,86,13),(42,98,17), (58,103,28),(41,99,17),(120,153,92),(255,225,140),(224,213,156),(63,104,39),(50,95,13),(43,85,18),(52,86,22),(44,91,15),(109,134,66),(213,195,123),(98,120,49),(48,94,18),(42,88,15),(19,22,5),(9,12,1),(28,43,10),(154,140,22),(156,139,26),(24,35,12),(35,47,15),(34,59,10),(16,11,3),(11,11,4),(42,42,14),(205,202,21),(191,192,18),(28,28,12),(16,13,8),(25,19,5),(21,28,10),(24,25,8),(11,8,6),(42,47,7),(58,65,11),(7,7,6),(20,18,9),(21,19,17),(25,42,17),(32,54,21),(33,52,22),(32,33,16),(23,21,10),(8,10,3),(13,13,6),(15,15,5)