Encodings

We discussed earlier that a program is written in a specific language, which then gets translated into a machine language of 0s and 1s. In a little more detail, this means that everything that is in a program must have a way of being translated into 0s and 1s, including letters, numbers, symbols, images, sounds, etc. If everyone decided on their own way of doing this, it would be very difficult to transport programs from one machine to another. So, there are standard ways of encoding data.

Exercise 1: Do a little research (find another book or look it up on the internet) and find out what ASCII and Unicode are. How are they similar? How are they different?

Exercise 2: Find the ASCII values of the letters in your name.
My name is Pam, so I would type the following commands into a Code cell in Google Colab:

```
print(ord('P'))
print(ord('a'))
print(ord('m'))
```

This would give the following result when the Code cell is run:

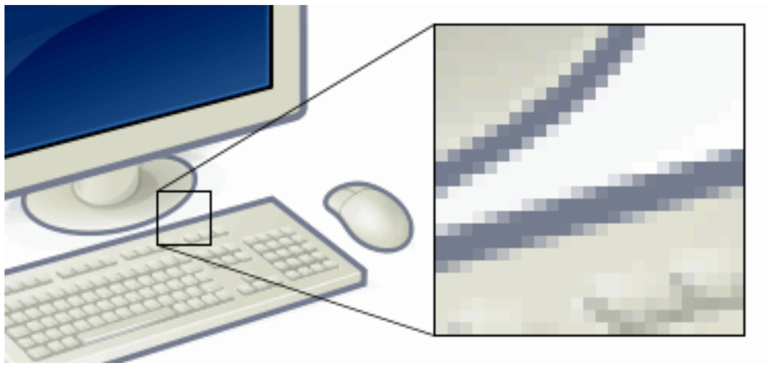
```
80
97
109
```

Follow-up Analysis: What is ord? It is a built-in function in Python that returns the ordinal (*i.e.,* ASCII) value of a character. The print command tells the computer to display the value that was returned from this function.

So we just found that ASCII and Unicode are the standards for encoding characters. Let's now turn our attention to encoding images.

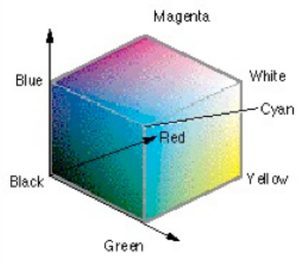
To begin with, we represent an image as a grid of tiny pixels (image elements).

The following example shows an image with a portion greatly enlarged, in which the individual pixels are rendered as little squares and can be easily seen.

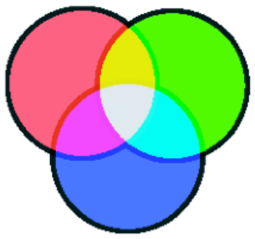


.

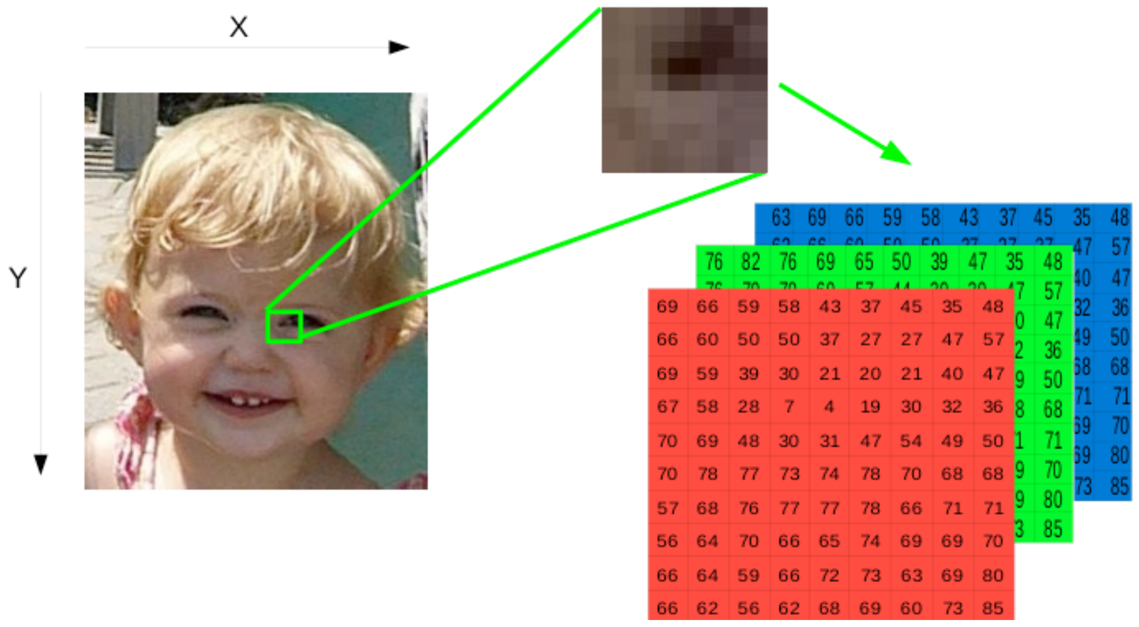
Each pixel encodes color for that spot in the image. There are many encodings for color. Printers use CMYK (**C**yan, **M**agenta, **Y**ellow, and blac**K**). Humans often prefer HSV (Hue, Saturation, and Brightness). We will use the most common for computers: RGB (**R**ed, **G**reen, and **B**lue).



Aside: RGB vs Paint
Paint is an applied medium, where the color seen is the result of reflected light from the "painted surface". This is broken down into the primary colors of red, blue, and yellow as seen on printed material. A specific color can be made by mixing different pigments. RGB are the primary colors referring to projected light and has no relationship to pigments of paint nor the mixing of them.



The following example shows an image, say from a digital camera, with a small section enlarged, in which the individual pixels and their RGB values can be seen.



69	66	59	58	43	37	45	35	48
66	60	50	50	37	27	27	47	57
69	59	39	30	21	20	21	40	47
67	58	28	7	4	19	30	32	36
70	69	48	30	31	47	54	49	50
70	78	77	73	74	78	70	68	68
57	68	76	77	77	78	66	71	71
56	64	70	66	65	74	69	69	70
66	64	59	66	72	73	63	69	80
66	62	56	62	68	69	60	73	85

In RGB, each color has three component colors: the amount of red, the amount of green, and the amount of blue. In a color image, each pixel is typically represented with three bytes of storage, one for each of the red, green, and blue components. This gives us 24 bits of storage per pixel, and 16, 777,216 possible different colors.

Aside: Bits and Bytes

A **bit** is a single 0 or 1. A **byte** is a sequence of 8 bits, such as 10010101 or 01011011. Each bit in a byte has 2 possibilities – either a 0 or a 1, so there are 2^8 = 256 different values represented by a byte. A pixel uses 3 bytes of storage, which allows $256 * 256 * 256 = 16{,}777{,}216$ (or 2^{24}) distinct colors.

A small image from a digital camera may have 1280 x 1024 = 1,310,70 pixels. This would have 1,310,720 x 3 = 3,932,160 bytes of storage, which is about 4 MB.

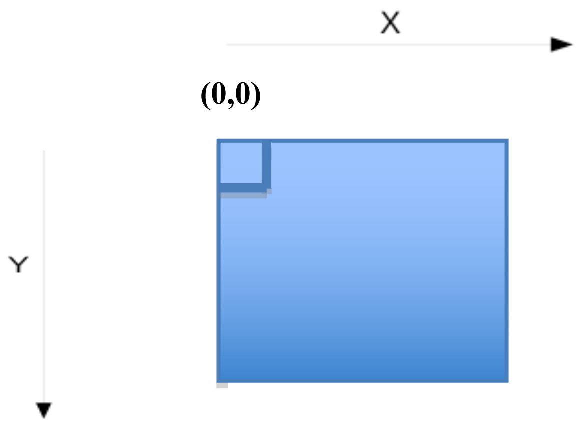
Exercise 3: A pixel in a grayscale image (black and white) is typically represented with one byte of storage. How many possible colors are there?

Exercise 4: What are the RGB values for white? What are the RGB values for black?

Images, Pixels, and Python

We said an image is represented as a grid of pixels. So, in Python, if we are going to manipulate the colors in an image, we need to have a way to work with each pixel within an image. The Python Imaging Library (PIL) provides the foundation for image processing in Python. As part of PIL, the Image module provides a class to represent an image. Images are represented with a Cartesian pixel coordinate system, with (0,0) in the upper left corner. (Note that the coordinates refer to the implied pixel corners; the center of a pixel addressed as (0, 0) actually lies at (0.5, 0.5).) The PIL library also stores values for red, green, and blue with each pixel.

The *x*- and *y*-values are similar to what you would see on a coordinate plane in math, except that the position (0, 0) is at the top left corner, and then the *x*- and *y*-values get larger as you go to the right and down, respectively.



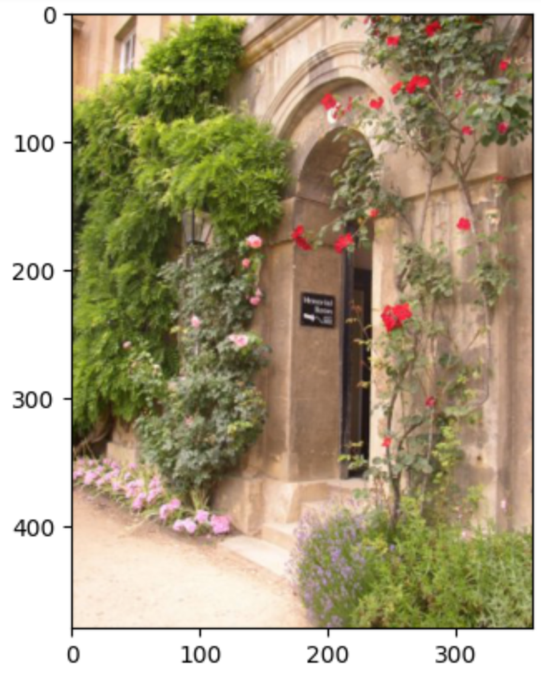
Exercise 5: If an image had dimensions 640 x 480, what would the x- and y-values of the pixel in top right corner be? The bottom left corner? The bottom right corner?

In a previous activity, we saw two different ways to show an image: 1) Just type the name of the image and 2) use the `show` function on the image. There is another way to show an image as a figure. By plotting the image, we are able to see the coordinate axis. The following statements will import the plotting library, choose an image, and will show it with an axis.

```
from matplotlib import pyplot as plt
myImage= Image.open("/drive/My Drive/ColabNotebooks/arch.jpg")
plt.imshow(myImage)
```

This code gives the following output:

```
<matplotlib.image.AxesImage at 0x7f19109c8160>
```



In PIL, there are several attributes (properties) and functions already defined to let us work with images. To find the width and height of an image, we will use the `width` and `height` attributes, respectively. To get a pixel from a particular image, we will use the `getpixel` function. This function requires a tuple consisting of an *x*-value and a *y*-value as its parameter and returns a tuple consisting of the three values for red, green, and blue for that pixel.

Example: The following code segment demonstrates how to use the `width` and `height` attributes, along with the `getpixel` function. Suppose that the chosen image has dimensions 360 x 480.

```
myImage= Image.open("/drive/My Drive/ColabNotebooks/arch.jpg")
myImage.show()
w = myImage.width
h = myImage.height
print("width:",w)
print("height:", h)
px1 = myImage.getpixel((0,0))
px2 = myImage.getpixel((10, 15))
print("pixel1:", px1)
print("pixel2:", px2)
```

The output of this code block looks like the following:

```
width: 360
height: 480
pixel1: (214, 166, 130)
pixel2: (211, 165, 142)
```

Exercise 6: Describe which pixels the variables `px1` and `px2` are storing.

Exercise 7: How would we get the pixel in the middle of the image?

In the code segment of the previous example, we saw that the `getpixel` function returns the red, green, and blue values of a pixel as a tuple. In order to get those individual values from a pixel, we can use three variables, say, `pxred`, `pxgreen`, and `pxblue`. In the following code segment, we get the RGB values from the pixel at (10, 15) and print them out.

```
pxred, pxgreen, pxblue = myImage.getpixel((10,15))
print('Red value', pxred)
print('Green value', pxgreen)
print('Blue value', pxblue)
```

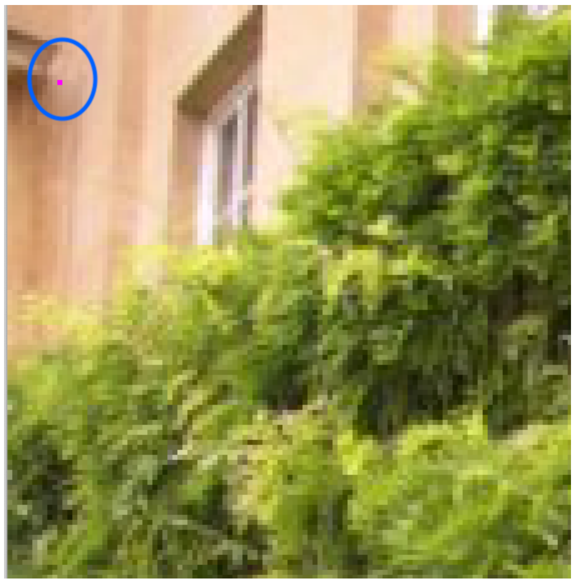
Exercise 8: What are the minimum and maximum values of red, green, and blue that `getpixel` that can return?

To set the color of a pixel, we will use the `putpixel` function. This function takes two parameters. The first parameter is a tuple with the x- and y- coordinates, and the second parameter is a tuple with the red, green, and blue values to set for the pixel.

Example: The following code sets the color of a pixel to purple:

```
myImage.putpixel((10,15),(255,0,255))
myImage.show()
```

Remember, pixels are very small, so it is sometimes hard to see that one pixel has changed color. A zoomed in portion of the results from this code would give us something like this:



Exercise 9: Write code to get a pixel from an image and turn it black.

Drawing on Images

The ImageDraw module from PIL provides a number of functions that will draw basic lines and shapes on top of an image. Some of these include:

- `line([x1, y1, x2, y2], fill, width)`
- `rectangle([x1, y1, x2, y2], fill, outline, width)`
- `ellipse([x1, y1, x2, y2], fill, outline, width)`
- `point([x1, y1, x2, y2, x3, y3, …], fill)`
- `text((x, y), string, fill)`

How do these functions work? The `line` function takes 3 parameters – a tuple containing the *x*- and *y*- coordinates of one end of the line and the *x*- and *y*- coordinates of the other end of the line, the color you'd like the line to be, and the width of the line. It will then draw the line between these two points on the specified image. It does not return anything, but instead, modifies the original image.

The `rectangle` function takes 4 parameters – a tuple containing the *x*- and *y*- coordinates of the upper left corner and the *x*- and *y*- coordinates of the lower right corner, the (optional) color to fill in the rectangle, the (optional) color for the outline, and the (optional) width of the outline. It then draws a rectangle with the top left corner and the bottom right corner and colors as specified. If you want just the outline of the rectangle you would not include a fill color. If you do not want an outline then you do not include the outline parameter.

The `ellipse` function works similarly to the `rectangle` function. The *x*- and *y*- coordinates are of the bounding box around the ellipse.

The `point` function takes a set of *x*- and *y*-coordinates and draws points of the specified color at those coordinates.

The `text` function takes 3 parameters – the x- and *y*- coordinates to place the text, some text, and a color. The text to be displayed must be specified within quotation marks. The *x*- and *y*- coordinates specify the top left corner for the text string. The function will then add the text to the specified image.

There are additional drawing functions also described in the ImageDraw module.

To draw on an image, we have two options – we can draw on an empty image (*i.e.*, a blank image), or we can draw on an existing image. To draw on a blank image, we must first make a blank image by using the `new` function:

```
myImage = Image.new('RGB',(100,150))
```

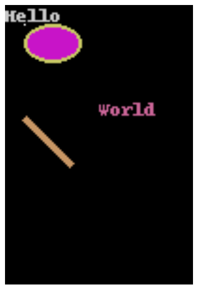
This code would create an empty *black* image with dimensions 100 x 150. To get a blank image of a different color, a third parameter, representing the RGB value to be applied to all of the pixels, can be added to the new function, as in:

```
myImage = Image.new('RGB',(100,150),(75,100,150))
```

Example: The following code segment draws a line, an ellipse, and some text on a blank image.

```
myImage = Image.new('RGB',(100,150))
d =ImageDraw.Draw(myImage)
d.text((50,50),"World",fill=(200,100,150))
d.ellipse([10,10,40,30],fill=(200,20,200),outline=(200,200,100),width=2)
d.line([35,85,10,60],fill=(200,150,100),width=3)
d.text((0,0),'Hello',fill=(200,200,200))
myImage.show()
```

The resulting image:



In the next activity, we will experiment with drawing on images and will practice calling functions with different parameters.

Activity: **Drawing Shapes on Images by Using Functions**