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, pictures, 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: Create an appropriately named function which prints the ASCII values of the letters in your name. My name is Pam, so my function would look like this:

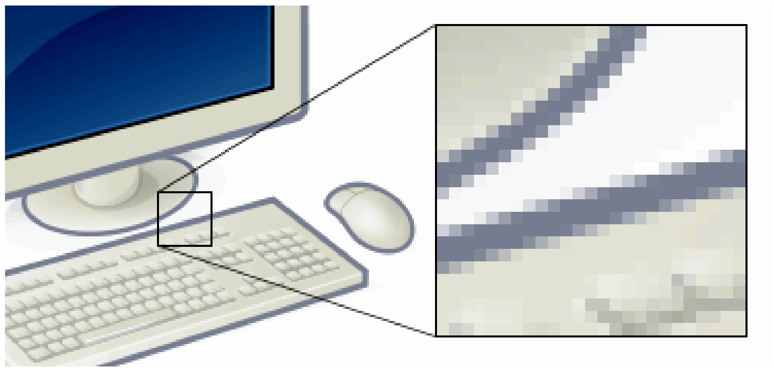
```
def nameToASCII( ):
        print(ord(‘P’))
        print(ord(‘a’))
        print(ord(‘m’))
```

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 pictures.

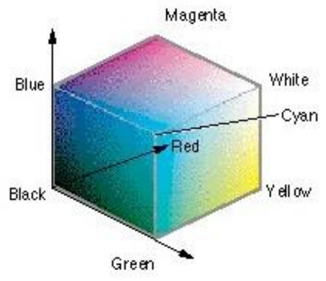
To begin with, we represent a picture as a grid of tiny pixels (picture elements).

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.

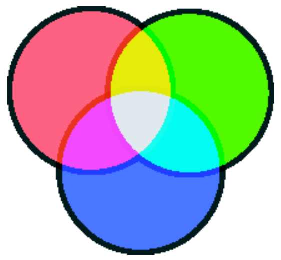


.

Each pixel encodes color for that spot in the picture. 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).

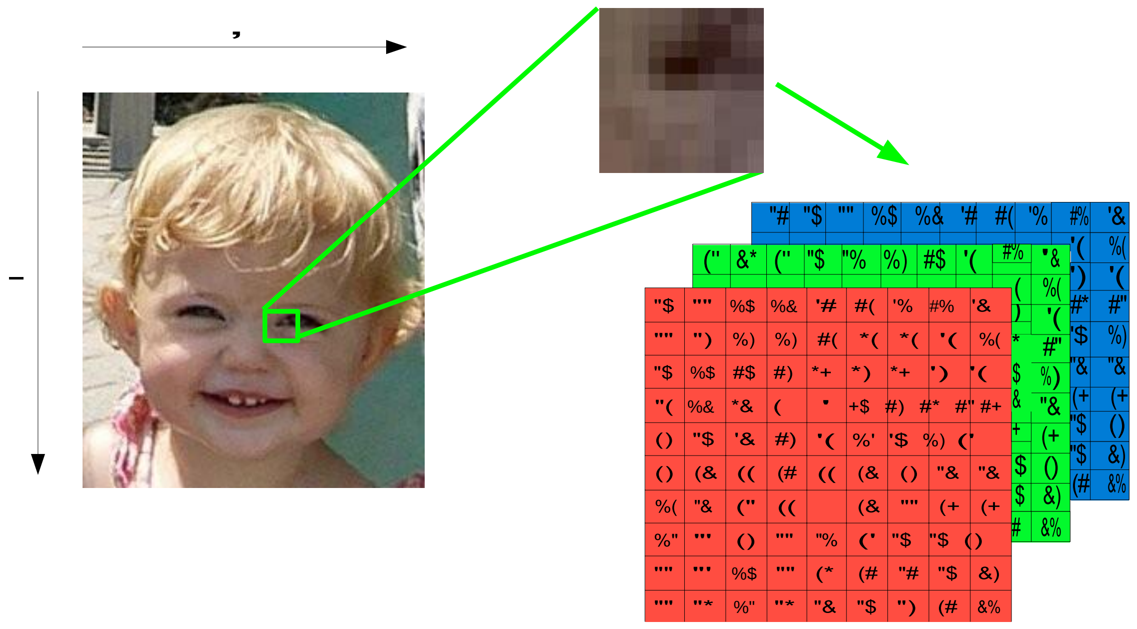


In RGB, each color has three component colors: the amount of red, the amount of green, and the amount of blue. In a color image, each pixel is typically represented with three bytes (a byte is 8 bits, or a sequence of 8 0s and 1s) of storage, one for each of the red, green, and blue components. This gives us 24 bits of storage per pixel, and 16,777,216 possible different colors.



A small image from a digital camera may have 1280 x 1024 = 1,310,70 pixels. This would have 1,310,720 x 24 = 31,457,280 bits of storage, which is about 31,457,280 / 8 = (about) 4 MB.

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



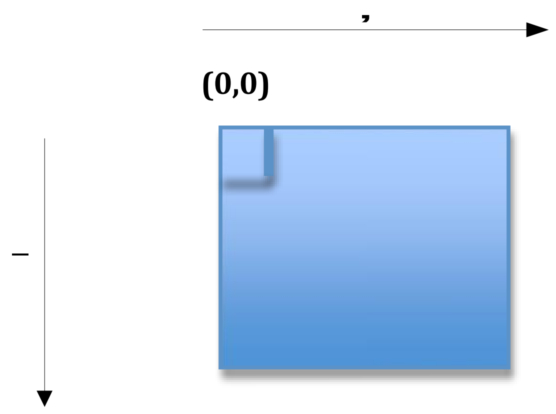
This example shows an image with a small section enlarged, in which the individual pixels and their RGB values can be seen.

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.

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

Pixels and jes4py

We said a picture is represented as a grid of pixels. So, when programming, if we are going to manipulate the colors in a picture, we need to have a way to work with each pixel. The jes4py library provides a pixel data type for us that has an *x*- and *y*-position, and R, G, and B values for 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 the jes4py library, there are a number of functions defined to let us work with pictures. To find the width and height of a picture, we will use the `getWidth` and `getHeight` functions, respectively. These two functions both require a picture object as a parameter. To get a pixel from a particular picture, we will use the `getPixel` function. This function requires a picture, an *x*-value, and a *y*-value as parameters.

Example: The following code demonstrates how to use the `getWidth`, `getHeight`, and `getPixel` functions. Suppose that the image chosen has dimensions 640 x 480.

```
myFile = pickAFile( )
myPict = makePicture(myFile)
w = getWdith(myPict)
print(w)
h = getHeight(myPict)
print(h)
px1 = getPixel(myPict, 0, 0)
px2 = getPixel(myPict, 10, 15)
```

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 picture?

Once we have a pixel, the jes4py library has functions to return the RGB color values of the pixel (`getRed`, `getGreen`, `getBlue`), and functions to set the color values of the pixel (`setRed`, `setGreen`, `setBlue`).

Example: The following code gets a pixel from a picture and prints out its red, green, and blue values:

```
myFile = pickAFile( )
myPict = makePicture(myFile)
px = getPixel(myPict, 10, 15)
r = getRed(px)
print(r)
g = getGreen(px)
print(g)
b = getBlue(px)
print(b)
```

Exercise 8: What are the minimum and maximum values that `getRed`, `getGreen`, and `getBlue` can return?

Example: The following code gets a pixel from a picture and turns it white:

```
myFile = pickAFile( )
myPict = makePicture(myFile)
px = getPixel(myPict, 10, 15)
setRed(px, 255)
setGreen(px, 255)
setBlue(px, 255)
```

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

Drawing on Pictures

The jes4py library provides a number of functions that will draw basic lines and shapes on top of a picture. Some of these include:

- `addLine(picture, x1, y1, x2, y2, color)`
- `addRect(picture, x, y, w, h, color)`
- `addRectFilled(picture, x, y, w, h, color)`
- `addText(picture, x, y, string, color)`
- `addOval(picture, x1, y1, w, h, color)`
- `addOvalFilled(picture, x1, y1, w, h, color)`
- `addArc(picture, x, y, w, h, startAngle, arcAngle, color)`
- `addArcFilled(picture, x, y, w, h, startAngle, arcAngle, color)`

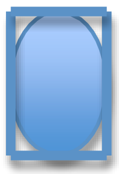
How do these functions work? The `addline` function takes 6 parameters – a picture, the *x*- and *y*- coordinates of one end of the line, the *x*- and *y*- coordinates of the other end of the line, and the color you'd like the line to be. (jes4py has some predefined colors that you may use, such as `red`, `green`, `blue`, `yellow`, `orange`, `magenta`, `white`, and `black`.) It will then draw the line between these two points on the specified picture. It does not return anything, but instead, modifies the original picture.

The `addRect` function also takes 6 parameters – a picture, *x*- and *y*- coordinates, a width, a height, and a color. It then draws a rectangle starting at the specified *x*- and *y*- coordinates that is the specified width, height and color. The top left corner of the rectangle is the pixel at (x, y).

The `addRectFilled` function does the same thing as `addRect`, except that it fills in the rectangle to make it a solid figure.

The `addText` function takes 5 parameters – a picture, *x*- and *y*-coordinates, some text, and a color. The text to be displayed must be specified within quotation marks. The x- and y-coordinates specify a *baseline* for the text string, similar to the line on a piece of lined paper. The function will then add the text to the specified picture.

The addOval and addOvalFilled functions are similar to the addRect and addRectFilled, except that the *x*- and *y*-coordinates are for the top left corner of the bounding box of the oval. (See the diagram for what a bounding box is.



The addArc function draws part of a circle, and the addArcFilled function draws part of a circle and fills it in. You can find more detailed descriptions of the parameters for these functions in Visual Code. Select a function you want to know more about, right click, and select **Go to Definition.** There are several other drawing functions also described in the **media.py** file that pops up.

WARNING: DO NOT MODIFY **media.py**!!!

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

```
pict = makeEmptyPicture(200, 300)
```

This code would create an empty *white* picture with dimensions 200 x 300. To get a blank picture of a different color, a third parameter can be added to the makeEmptyPicture function, as in:

```
pict = makeEmptyPicture(200, 300, red)
```

In the next mini-lab, we will experiment with drawing on pictures and will gain additional practice writing and using functions.

Mini-lab: Drawing Pictures