Image Manipulation: Pixel Traversal
Digital Images

- Bits are binary (0 or 1)
- Pixels are composed of bits
  - Bits-per-pixel determine the range of color
- Images are composed of pixels
Image Buffers

- Screen pixel data is stored in an array
- This array (or image buffer) allows us to access per-pixel information
Images in Processing

- Image buffers are stored in the PImage data type
- PImage allows for loading and displaying image data
- Some image manipulation:
  - Size
  - Position
  - Opacity
  - Tint
- To display: `image(PImage img, float x, float y, float width, float height)`
Loading and Displaying Images

PImage img;

void setup() {
    size(100, 100);
    img = loadImage("foo.png");
}

void draw() {
    image(img, 0, 0);  //Note: we must load an image before displaying it!
}
Fitting Processing Window to Image Size

```java
void setup() {
    surface.setResizable(true);
    img = loadImage("foo.png");
    surface.setSize(img.width, img.height);
}
```
Changing Individual Pixels

- `loadPixels()` and `updatePixels()` should be called before and after pixel manipulation respectively.

  - `loadPixels()` allows us to read from the pixel data.
  - `updatePixels()` writes any changes back to the pixel data.
  - Calls not necessary for every Operating System, but may not work without them.

- `PImage.pixels` array stores each pixel as a color.

  - Access the color of the pixel at index in `PImage img`:
    ```java
    color c = img.pixels[index];
    ```
Consider...

❖ How can we access every pixel in an image?
❖ How can we access every pixel by its (x, y) value?
❖ Hint: remember this layout!

How the pixels look:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

How the pixels are stored:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
Accessing by (x, y) Coordinates

- We will perform a *stride* into the 1D array to know which row (e.g. *y value*) we are currently looking for.
- Once we get to the correct row, we can use the *column* (e.g. *x value*) to find the final placement of the index into the 1D array.
- To do this, we must know the image width.
  - row * imageWidth = index of row in the 1D array
  - Then add in the column value
Traversing an Image Buffer

//access img’s pixels

img.loadPixels();

for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++) {
        //access pixel at index and set c to its value
        int index = x + y * img.width;
        color c = img.pixels[index];
    }
}

//update any modifications to img’s pixels

img.updatePixels();
Tint

- `tint()` modifies the color of the displayed images
- `noTint()` disables `tint()` modifications

```java
void draw() {
    tint(0, 153, 204);
    image(img, 0, 0);
    noTint();
    image(img, 50, 50);
}
```
Hands-on: Creating Tint

Today’s activity:

1. Re-create Processing’s `tint` functionality using a method you create (i.e. **do not use** the existing tint function)

   1. This method can take RBG/color data like the Processing `tint` method does

   2. You may want to make your `tint` method to be “per image” rather than “per screen” — to do this, your method should also have an argument for the `PImage` you will tint