Multidimensional Arrays

Jerry Cain
CS 106AJ
November 5, 2018
slides courtesy of Eric Roberts

Multidimensional Arrays

- Because the elements of an array can be of any JavaScript type, those elements can themselves be arrays. Arrays of arrays are called ***multidimensional arrays***.
- In JavaScript, you can initialize a multidimensional array by using nested brackets in the initial value specification. For example, the following declaration creates a 3×3 array whose values form a magic square:

```
let magic = [ [ 2, 9, 4 ], [ 7, 5, 3 ], [ 6, 1, 8 ] ];
```

- This declaration creates a two-dimensional array conceptually organized like this:

2	9	4
7	5	3
6	1	8

`magic[0][0]`	`magic[0][1]`	`magic[0][2]`
`magic[1][0]`	`magic[1][1]`	`magic[1][2]`
`magic[2][0]`	`magic[2][1]`	`magic[2][2]`

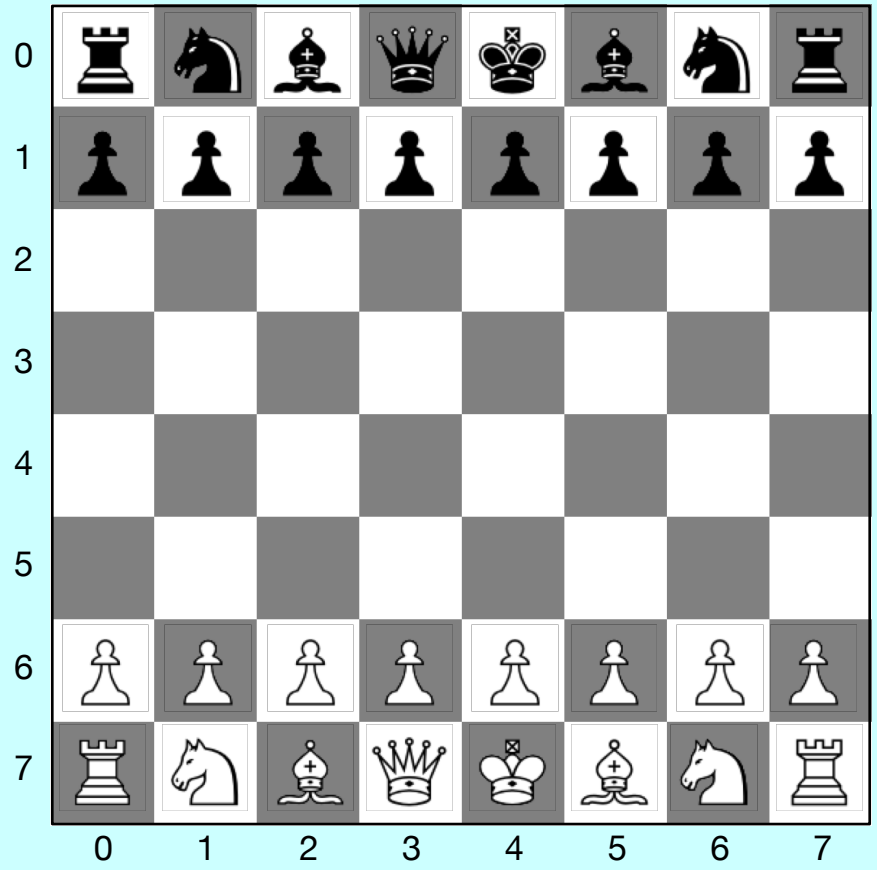
Example: Initialize a Multiplication Table

- The following constant definition initializes a multiplication table for the digits 0 to 9 like this:

```
const MULTIPLICATION_TABLE = [
    [  0,  0,  0,  0,  0,  0,  0,  0,  0,  0  ],
    [  0,  1,  2,  3,  4,  5,  6,  7,  8,  9  ],
    [  0,  2,  4,  6,  8, 10, 12, 14, 16, 18  ],
    [  0,  3,  6,  9, 12, 15, 18, 21, 24, 27  ],
    [  0,  4,  8, 12, 16, 20, 24, 28, 32, 36  ],
    [  0,  5, 10, 15, 20, 25, 30, 35, 40, 45  ],
    [  0,  6, 12, 18, 24, 30, 36, 42, 48, 56  ],
    [  0,  7, 14, 21, 28, 35, 42, 49, 56, 63  ],
    [  0,  8, 16, 24, 32, 40, 48, 56, 64, 72  ],
    [  0,  9, 18, 27, 36, 45, 54, 63, 72, 81  ]
];
```

- The product of the single-digit integers **`x`** and **`y`** appears in **`MULTIPLICATION_TABLE[x][y]`**.

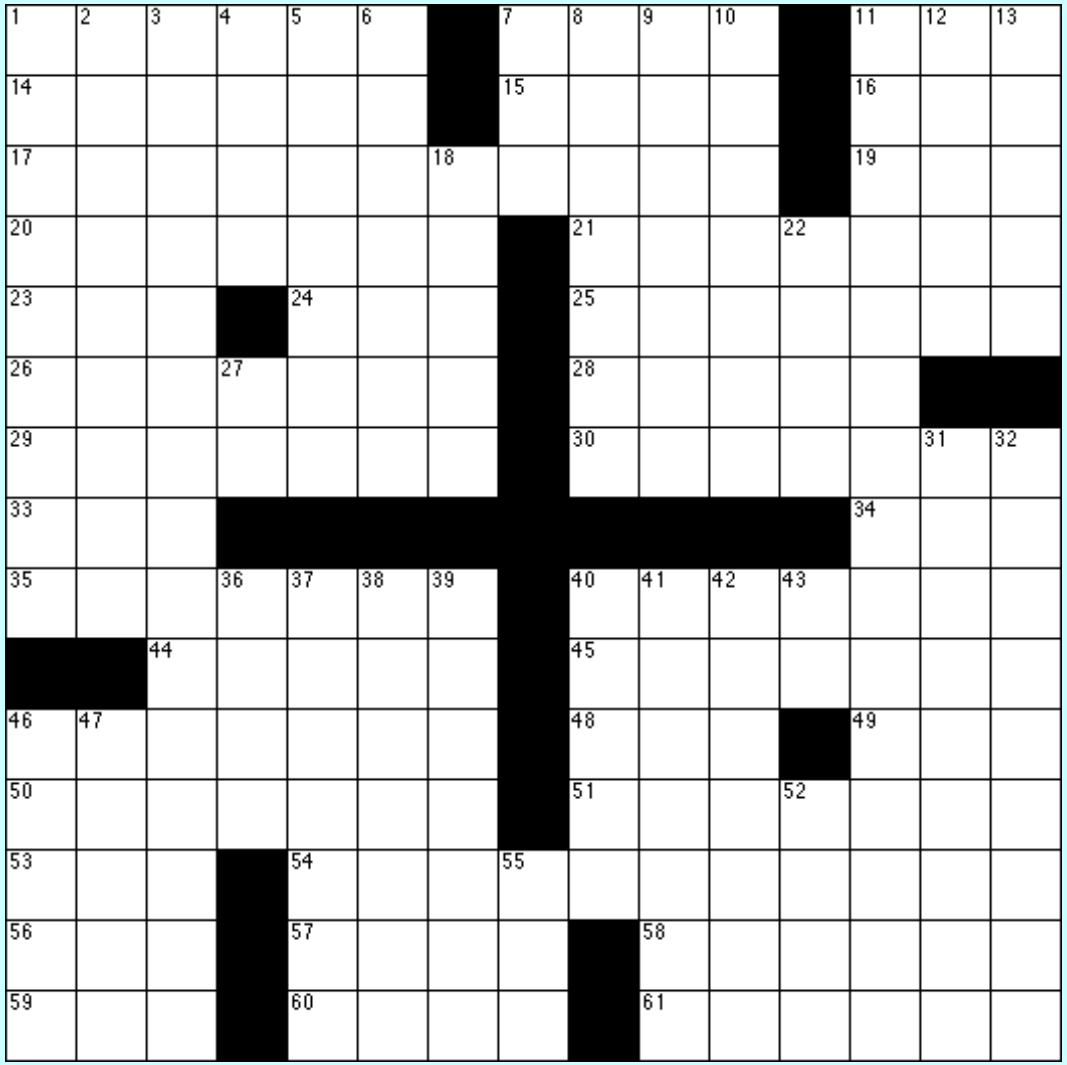
Exercise: Initialize a Chess Board



Exercise: Crossword Numbering

Write a program to number the squares in a crossword grid if they appear at the beginning of a word running either across or down.

- What types would you use to represent the data in this grid?
- How would you represent black squares?
- What rules can you supply to determine if a square is numbered?



The `GImage` Class

- The **`GImage`** class is used to model an image from a file. The **`GImage`** function itself has the form:

```
GImage(filename, x, y)
```

  where *filename* is the name of a file containing a stored image and *x* and *y* are the coordinates of the upper left corner of the image.

- Because **`GImage`** objects are read from the file system, you need to add a **`"load"`** event handler (without arguments) that executes once the image object has been configured from the file, as with:

```
let image = GImage(filename, x, y);
let callback = function() {
   code that further manipulates the loaded image
};
image.addEventListener("load", callback);
```

Images and Copyrights

- Most images that you find on the web are protected by copyright under international law.
- Before you use a copyrighted image, you should make sure that you have the necessary permissions. For images that appear of the web, the hosting site often specifies what rules apply for the use of that image. For example, images from the **`www.nasa.gov`** site can be used freely as long as you include the following citation identifying the source:

  Courtesy NASA/JPL-Caltech

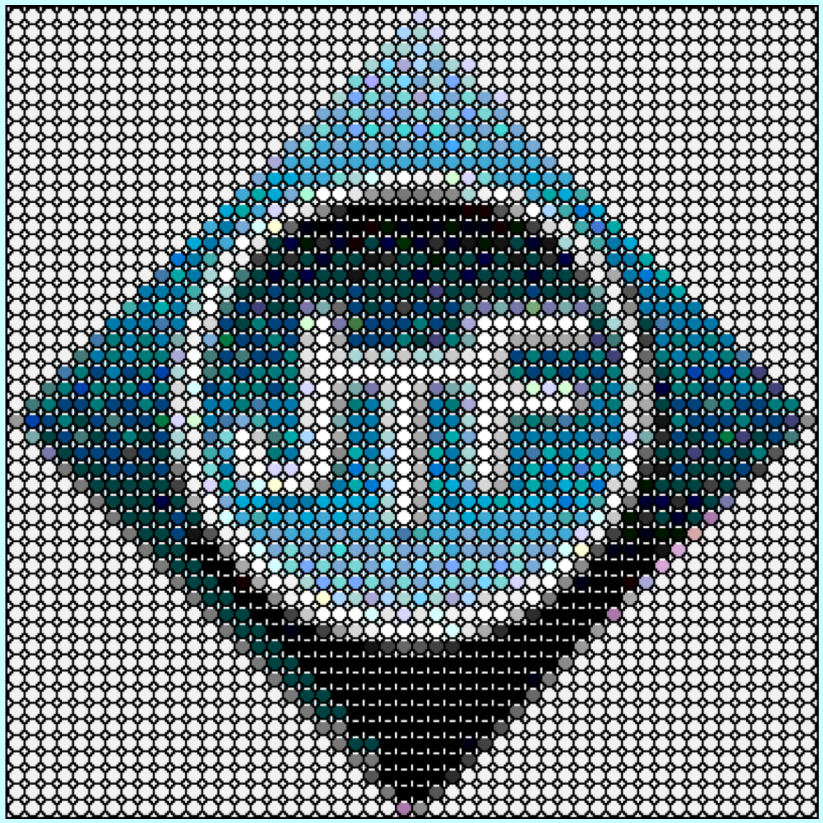
- In some cases, noncommercial use of an image may fall under the "fair use" doctrine, which allows some uses of proprietary material. Even in those cases, however, academic integrity and common courtesy both demand that you cite the source of any material that you have obtained from others.

Example of the `GImage` Class

```
function EarthImage() {
   let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   let image = GImage("EarthImage.png");
   image.addEventListener("load", function() {
      image.scale(GWINDOW_WIDTH / image.getWidth());
      gw.add(image, 0, 0);
      addCitation(gw, "Courtesy NASA/JPL-Caltech ");
   });
}
```

Multidimensional Arrays and Images

- One of the best examples of multidimensional arrays is an image, which is logically a two-dimensional array of pixels.
- Consider, for example, the logo for the Java Task Force at the top right. That logo is actually an array of pixels as shown in the expanded diagram at the bottom.
- The `GImage` class allows you to convert the data for the image into a two-dimensional array of pixel values. Once you have this array, you can work with the data to change the image.



Pixel Arrays

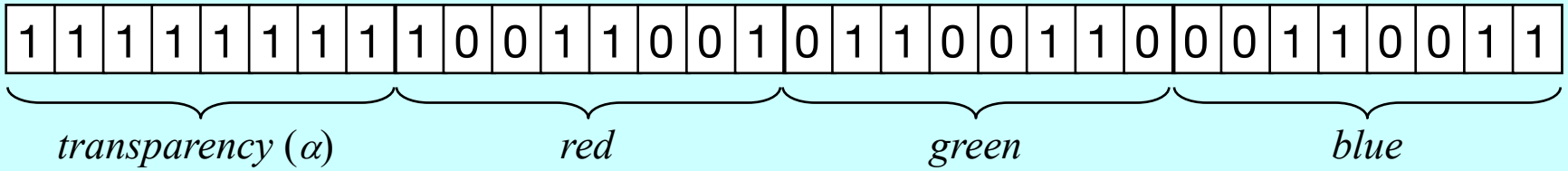
- If you have a **`GImage`** object, you can obtain the underlying pixel array by calling the **`getPixelArray`** method, which returns a two-dimensional array of numbers.
- For example, if you wanted to get the pixels from the image file **`JTFLogo.png`**, you could do so with the following code:

```
let logo = GImage("JTFLogo.png");
logo.addEventEventListener("load", function () {
   let pixels = logo.getPixelArray();
   code that manipulates those pixels and constructs a new image
});
```

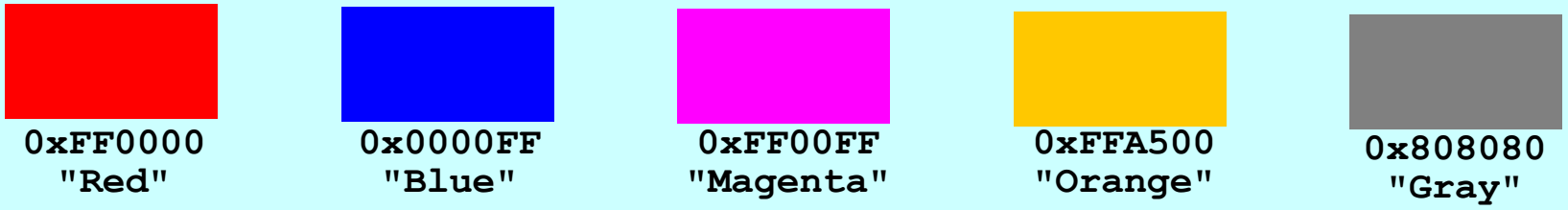
- The first index in a pixel array selects a row in the image, beginning at the top. The height of the image is therefore given by the expression **`pixels.length`**.
- The second index in a pixel array selects an individual pixel within a row, from left to right. You can use the expression **`pixels[0].length`** to determine the width of the image.

Pixel Values

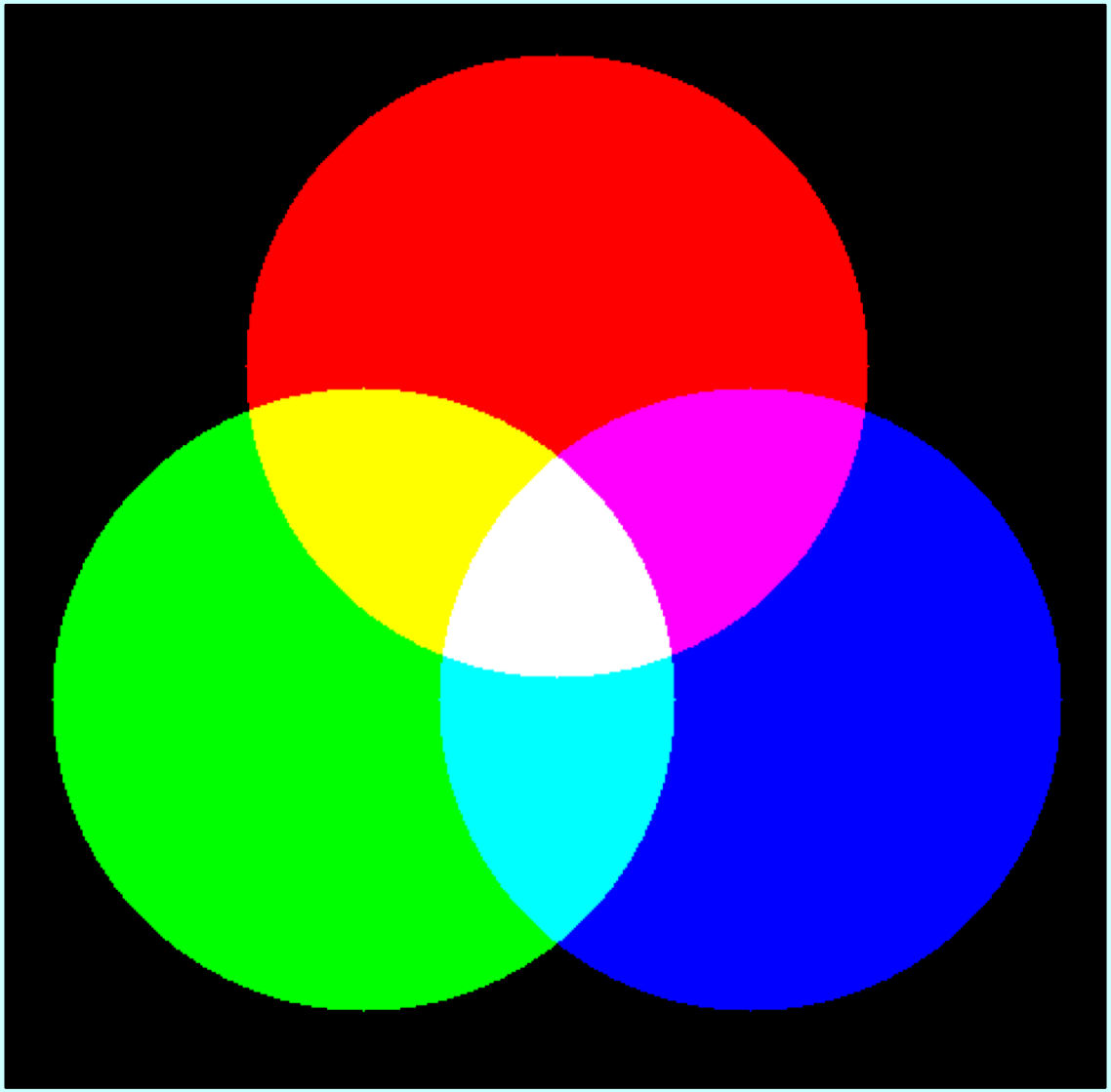
- Each individual element in a pixel array is an integer in which the 32 bits are interpreted as follows:



- The first byte of the pixel value specifies the ***transparency*** of the color, which is described in more detail on a later slide.
- The next three bytes indicate the amount of red, green, and blue in the pixel, in which each value varies from 0 to 255. Together, these three bytes form the ***RGB*** value of the color, which is typically expressed using six hexadecimal digits, as in the following examples:



Combining Colors of Light



Transparency

- The first byte of the pixel value specifies the transparency of the color, which indicates how much of the background shows through. This value is often denoted using the Greek letter alpha (α).
- Transparency values vary from 0 to 255. The value 0 is used to indicate a completely transparent color in which only the background appears. The value 255 indicates an opaque color that completely obscures the background. The standard color constants all have alpha values of 255.

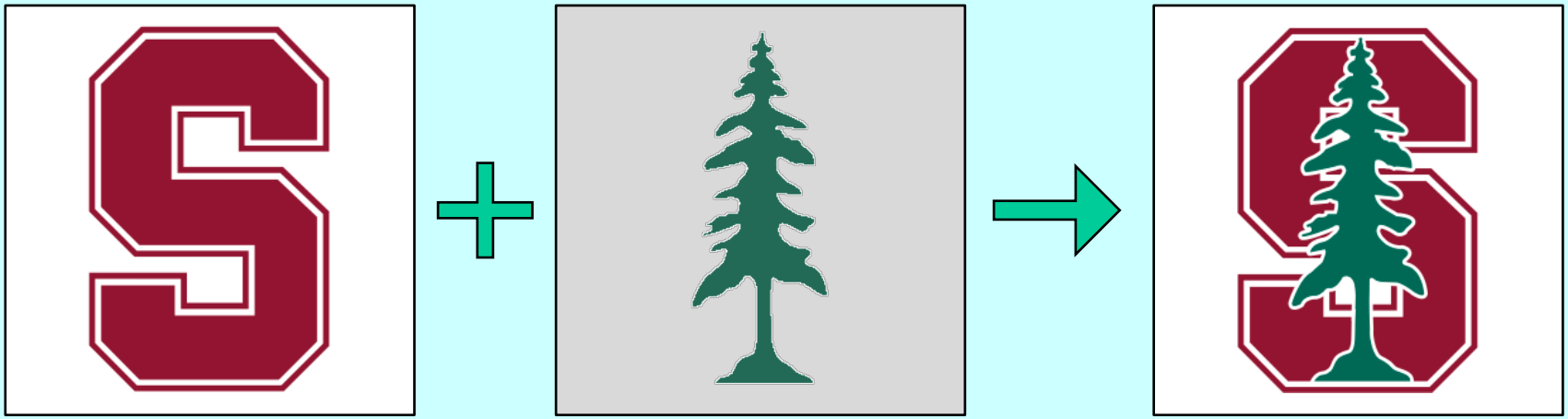


Image Manipulation

- You can use the facilities of the **`GImage`** class to manipulate images by executing the following steps:
  1. Read an existing image from a file into a **`GImage`** object.
  2. Call **`getPixelArray`** to get the pixels.
  3. Write the code to manipulate the pixel values in the array.
  4. Call the **`GImage`** function to create a new image. Instead of supplying a filename, pass in the transformed pixel array instead.
- The program on the next slide shows how you can apply this technique to flip an image vertically. The general strategy for inverting the image is simply to reverse the elements of the pixel array.

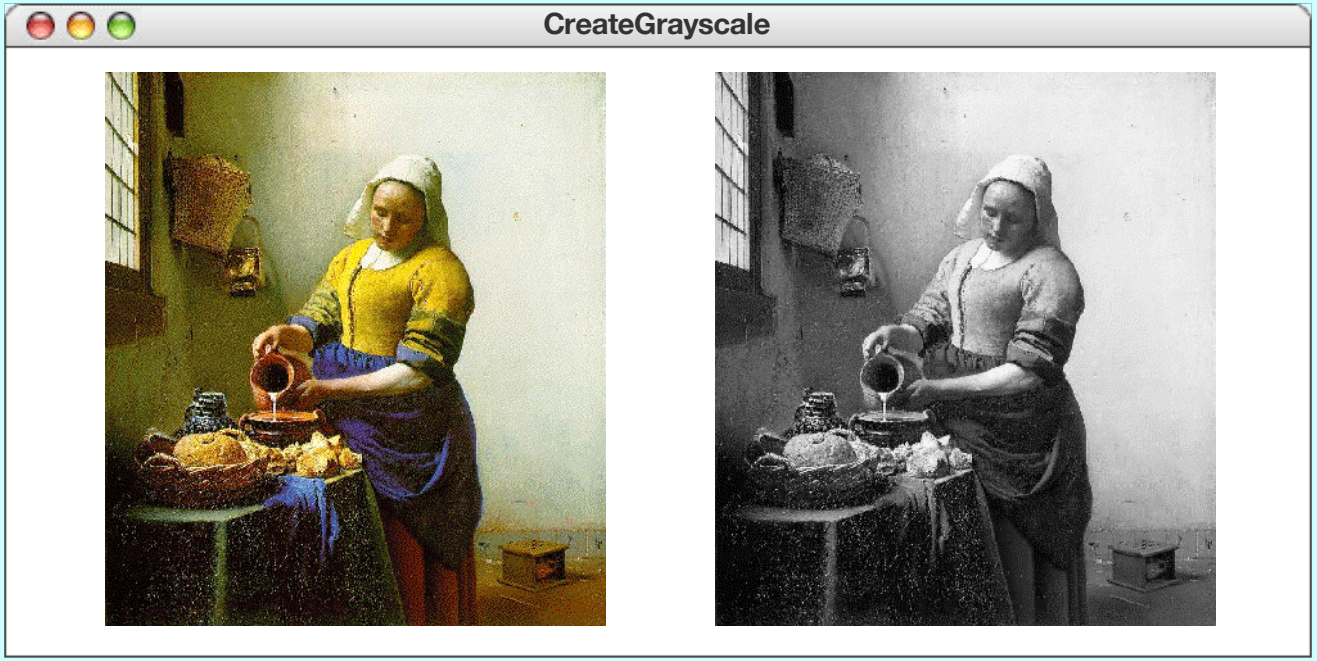
The `flipVertical` Function

```
/*
 * Creates a new image which consists of the bits in the
 * original flipped vertically around the center line.
 */

function flipVertical(image) {
   let array = image.getPixelArray();
   array.reverse();
   return GImage(array);
}
```

Creating a Grayscale Image

- As an illustration of how to use the bitwise operators to manipulate colors in an image, the text implements a method called **`createGrayscaleImage`** that converts a color image into a black-and-white image, as shown in the sample run at the bottom of this slide.
- The code to implement this method appears on the next slide.



The `createGrayscaleImage` Function

```
/* Creates a grayscale version of the original image */

function createGrayscaleImage(image) {
   let array = image.getPixelArray();
   let height = array.length;
   let width = array[0].length;
   for (let i = 0; i < height; i++) {
      for (let j = 0; j < width; j++) {
         let gray = luminance(array[i][j]);
         array[i][j] = GImage.createRGBPixel(gray, gray, gray);
      }
   }
   return GImage(array);
}

/* Calculates the luminance of a pixel using the NTSC formula */

function luminance(pixel) {
   let r = GImage.getRed(pixel);
   let g = GImage.getGreen(pixel);
   let b = GImage.getBlue(pixel);
   return Math.round(0.299 * r + 0.587 * g + 0.114 * b);
}
```

The End