Lab 05- Lab 06

Image and color basics Continued

Discussion is based on F.S. Hill Chapter 02,03,04,10

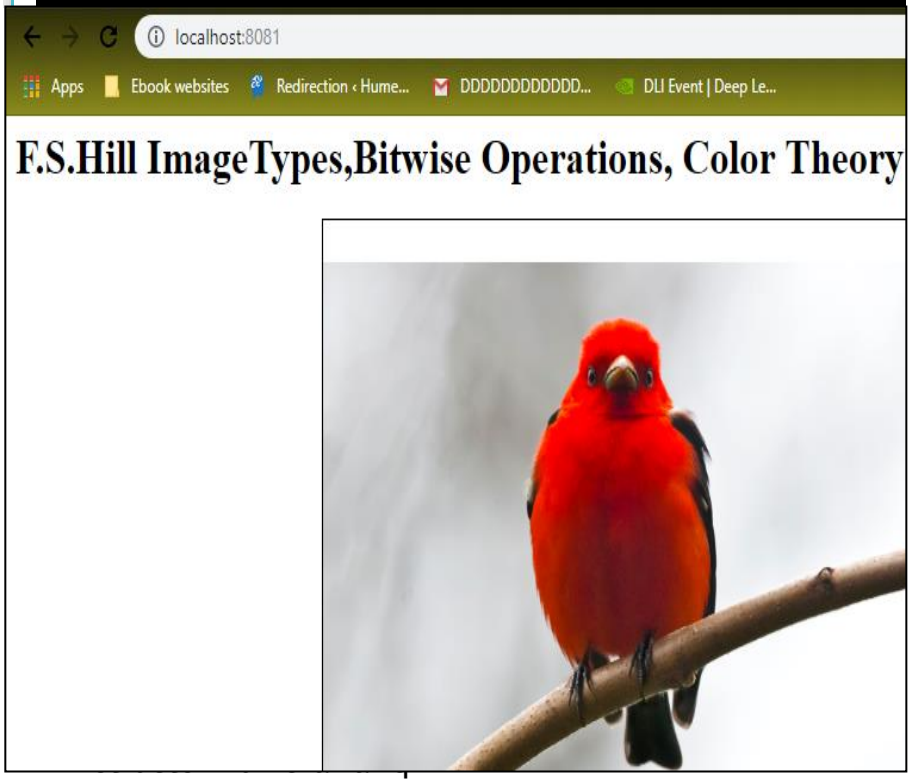
Lab 05-Lab 06 Tasks

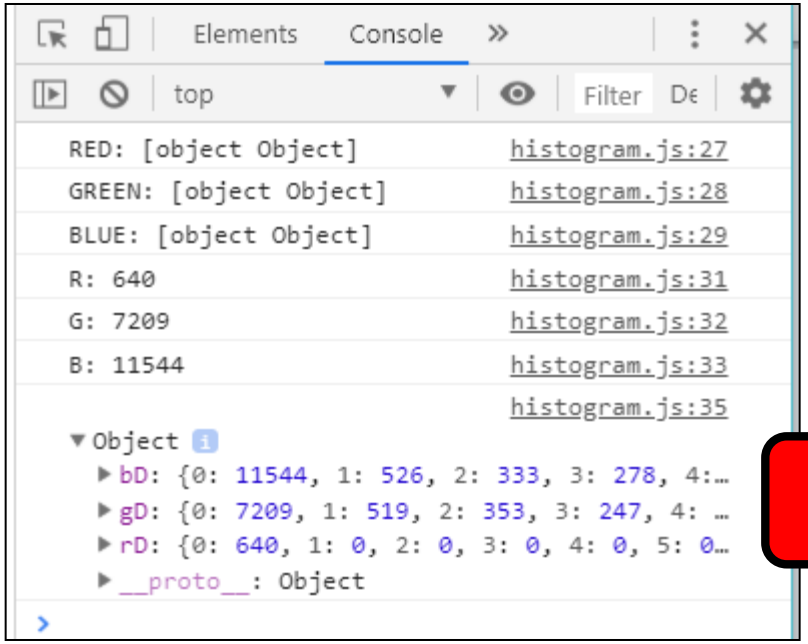
https://medium.com/tech-tajawal/javascript-classes-under-the-hood-6b26d2667677

1. Pixel Counting for histogram(Chap 10)
2. Drawing X and Y axis using moveTo() lineTo() (Chap 3)
3. Working with classes in pure java script
   class Point, class Square, class Rectangle (Chap 3)
4. Tweening / In-Between/ lerp (Chap 4)
5. Generating Random Number between ‘a’ and ‘b’
6. Draw Point as Square and Rectangle (Chap 3)
7. Drawing shapes with and without boundary (chap 3)
8. Mapping of Pixel count to rectangle height
9. Writing text on canvas

Recall the success of running static web site

```
D:\CGLab0304>node server.js
Server started at http://localhost:8081
Press CTRL + C to shutdown
```





1

The **getImageData**() function returns retrieve a set of pixel **data** from the canvas. The ImageData object represents a rectangle area of information and holds every pixel inside that rectangle. Every pixel in an ImageData object has four-element-array-like value, the RGBA values.

Problem 1: Pixels Counting for histogram

ImageData object. It returns the pixels in a special way in order to make them easy to manipulate. If you have, say, a 100×100 pixel canvas, it contains a total of 10,000 pixels. The ImageData array for it will then have 40,000 elements, because the pixels are broken up by component and listed sequentially. Each group of four elements in the ImageData array represent the red, green, blue, and alpha channels for that pixel. To loop through the pixels, just increment your counter by 4 every time, like I do here. Each channel, then, is an integer between 0 and 255.

```
  const iD=ctx.getImageData(0, 0, cv.width, cv.height).data; // image data

 for (var i=0; i<256; i++) { rD[i]=0; gD[i]=0; bD[i]=0; }

  for (var i=0; i<iD.length; i+=4) { // image parsing or splitting
    rD[iD[i]]++;
    gD[iD[i+1]]++;
    bD[iD[i+2]]++;
  }

  histogram({rD, gD,bD});  // passing dictionary to function
}
```

2

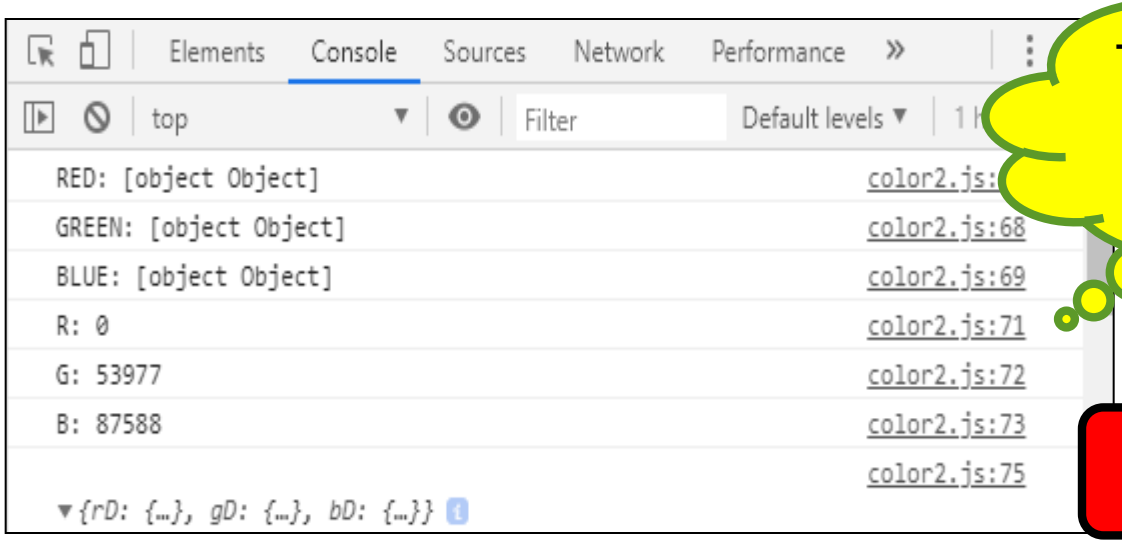
3

Attempt to Retrieve information from passed dictionary object

```
function histogram(data) { //Note function receive whole dictionary {rD, gD,bD}

    // checking that image data passed and manipulated succesfully
    // To view in your browser's JavaScript console
     console.log("RED: " + data.rD); // print as string
     console.log("GREEN: " + data.gD);
     console.log("BLUE: " + data.bD);

     console.log("R: " + data.rD[0]);// [0,1,2......255]
     console.log("G: " + data.gD[0]);// [0,1,2......255]
     console.log("B: " + data.bD[0]);// [0,1,2......255]

     console.log(data); // print object details
}
```

4

Recall passing object



This is output when I didn't resize image

5

Observing arrays

Red[0……255] Green[0……255] Blue[0……255]

```
▼{rD: {…}, gD: {…}, bD: {…}}
  ▼bD:
      0: 87588
      1: 4752
      2: 2584
      3: 2357
      4: 1718
      5: 1920
      6: 1940
      7: 2104
      8: 2434
      9: 2566
      10: 2274
      11: 2527
      12: 2413
      13: 3160
      14: 4282
      15: 5932
      16: 4519
      17: 4258
      18: 3886
      19: 5887
      20: 6517
      21: 6288
      22: 6396
      23: 8857
      24: 9468
```

```
      245: 40681
      246: 48191
      247: 52973
      248: 53587
      249: 37079
      250: 28052
      251: 33599
      252: 33456
      253: 46685
      254: 60950
      255: 272457
    ▶__proto__: Object
  ▶gD: {0: 53977, 1: 3913, 2: 2244, 3: 2026, 4: 985, 5: 978, 6: 1771, 7:…
  ▶rD: {0: 0, 1: 0, 2: 7, 3: 6, 4: 6, 5: 25, 6: 63, 7: 96, 8: 178, 9: 29…
  ▼__proto__:
    ▶constructor: ƒ Object()
    ▶hasOwnProperty: ƒ hasOwnProperty()
    ▶isPrototypeOf: ƒ isPrototypeOf()
    ▶propertyIsEnumerable: ƒ propertyIsEnumerable()
    ▶toLocaleString: ƒ toLocaleString()
    ▶toString: ƒ toString()
    ▶valueOf: ƒ valueOf()
    ▶__defineGetter__: ƒ __defineGetter__()
    ▶__defineSetter__: ƒ __defineSetter__()
    ▶__lookupGetter__: ƒ __lookupGetter__()
    ▶__lookupSetter__: ƒ __lookupSetter__()
    ▶get __proto__: ƒ __proto__()
    ▶set __proto__: ƒ __proto__()
```

6

Red[0……255] Green[0……255] Blue[0……255]

```
▼{rD: {…}, gD: {…}, bD: {…}}
  ▼bD:
      0: 87588
      1: 4752
      2: 2584
      3: 2357
      4: 1718
      5: 1920
      6: 1940
      7: 2104
      8: 2434
      9: 2566
      10: 2274
      11: 2527
      12: 2413
      13: 3160
      14: 4282
      15: 5932
      16: 4519
      17: 4258
      18: 3886
      19: 5887
      20: 6517
      21: 6288
      22: 6396
      23: 8857
      24: 9468
```

```
      245: 40681
      246: 48191
      247: 52973
      248: 53587
      249: 37079
      250: 28052
      251: 33599
      252: 33456
      253: 46685
      254: 60950
      255: 272457
    ▶__proto__: Object
  ▶gD: {0: 53977, 1: 3913, 2: 2244, 3: 2026, 4: 985, 5: 978, 6: 1771, 7:…
  ▶rD: {0: 0, 1: 0, 2: 7, 3: 6, 4: 6, 5: 25, 6: 63, 7: 96, 8: 178, 9: 29…
  ▼__proto__:
    ▶constructor: ƒ Object()
    ▶hasOwnProperty: ƒ hasOwnProperty()
    ▶isPrototypeOf: ƒ isPrototypeOf()
    ▶propertyIsEnumerable: ƒ propertyIsEnumerable()
    ▶toLocaleString: ƒ toLocaleString()
    ▶toString: ƒ toString()
    ▶valueOf: ƒ valueOf()
    ▶__defineGetter__: ƒ __defineGetter__()
    ▶__defineSetter__: ƒ __defineSetter__()
    ▶__lookupGetter__: ƒ __lookupGetter__()
    ▶__lookupSetter__: ƒ __lookupSetter__()
    ▶get __proto__: ƒ __proto__()
    ▶set __proto__: ƒ __proto__()
```

Problem 2: Drawing histogram from arrays step I : Find length of dictionary objects

```
function histogram(data) { //Note function receive whole dictionary {rD, gD,bD}

   // checking that image data passed and manipulated succesfully
   // To view in your browser's JavaScript console
    console.log("RED: " + data.rD); // print as string
    console.log("GREEN: " + data.gD);
    console.log("BLUE: " + data.bD);

    console.log("R: " + data.rD[0]);// [0,1,2......255]
    console.log("G: " + data.gD[0]);// [0,1,2......255]
    console.log("B: " + data.bD[0]);// [0,1,2......255]

    console.log(data); // print object details
    //console.log(data.rD[0].length); // lenght property is not defined for dictionaries

    console.log(size_dict(data.rD))  // size of red dictionary
    console.log(size_dict(data.gD)); // size of green dictionary
    console.log(size_dict(data.bD)); // size of blue dictionary

}
```

Length of dictionary objects

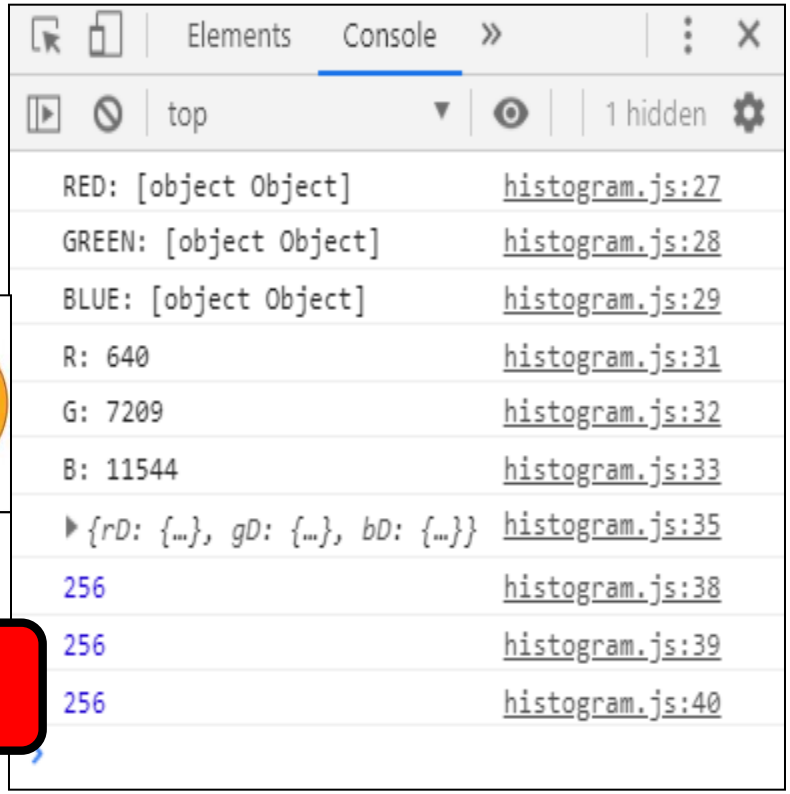
```
function size_dict(d){c=0; for (i in d) ++c; return c}
```

1

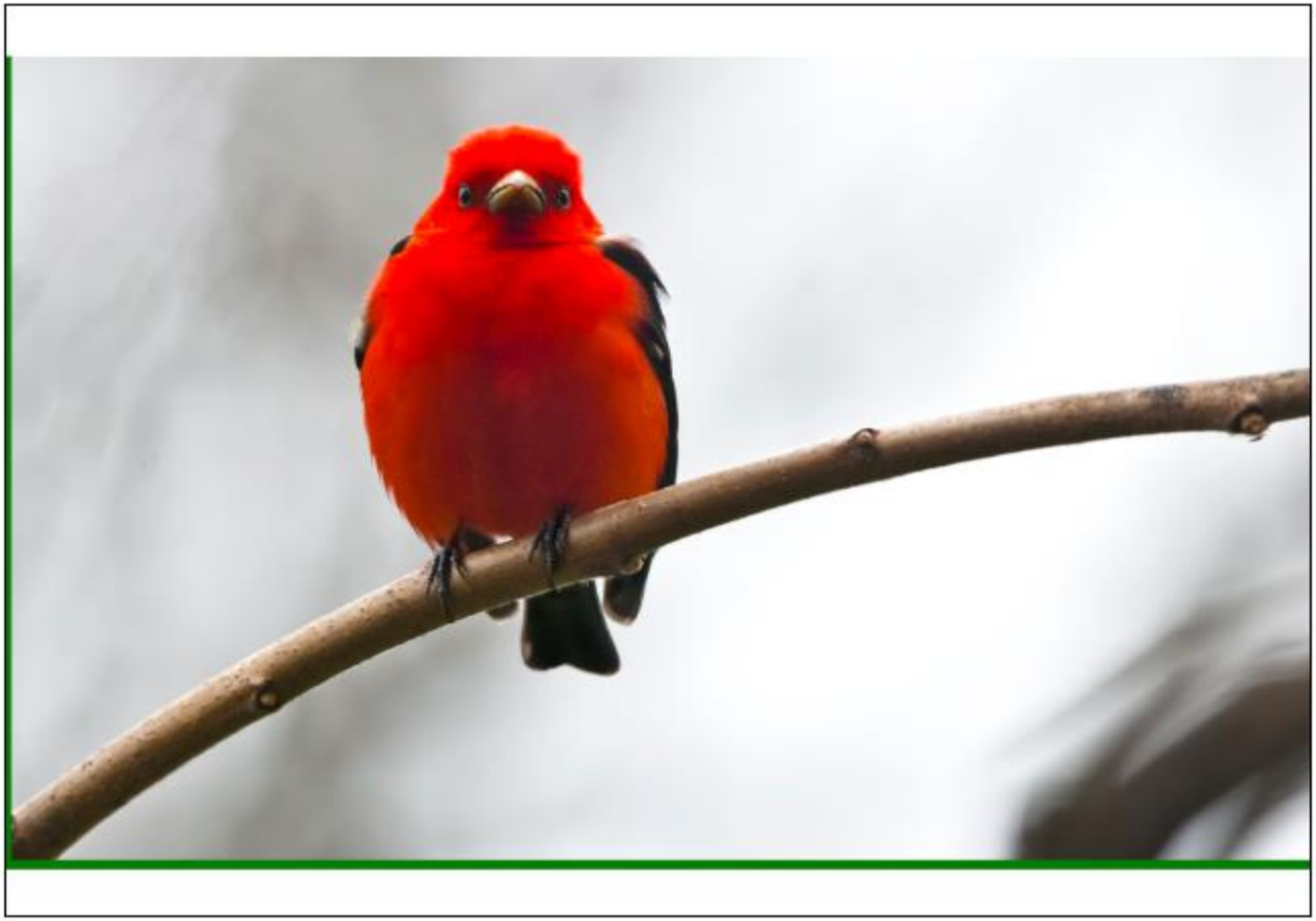
Length property is not defined for dictionary objects so I wrote my own function

```
console.log(size_dict(data.rD))  // size of red dictionary
console.log(size_dict(data.gD)); // size of green dictionary
console.log(size_dict(data.bD)); // size of blue dictionary
```

2

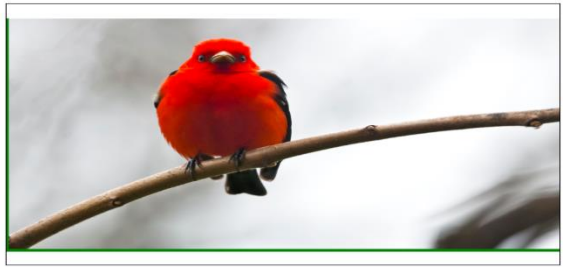


Draw x-axis and y-axis, see code on next slide



Code to draw x-axis and y-axis in function histogram( ). Study moveTo() and lineTo() from chap 3. The code is in continuation of Lab 03-04

```
// draw line for x-axis

ctx.lineWidth = "5";
ctx.strokeStyle = "green"; // Green path
ctx.beginPath();
ctx.moveTo(xStart,yStart+renderableHeight);   // bottom-left
ctx.lineTo(xStart+renderableWidth,yStart+renderableHeight)//bottom-right
/** render */
ctx.stroke();

// draw line for y-axis
ctx.beginPath();
ctx.moveTo(xStart,yStart+renderableHeight);   // bottom-left
ctx.lineTo(xStart,yStart)//top-right
/** render */
ctx.stroke();
```



Problem 4: Generating Points between 0 and 255

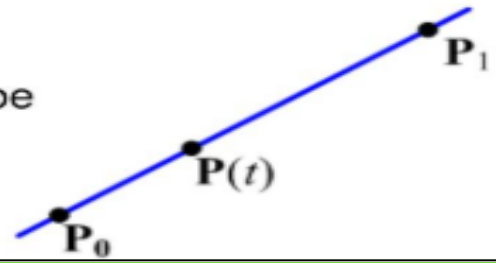
- General linear interpolation between two mathematical elements P_0 and P_1 takes the form:

$$P(t) = P_0(1-t) + P_1 t$$

- Typically t is in the range [0,1] - note that:

$$P(0) = P_0,\ P(1) = P_1 \text{ and } P(1/2) = (P_0 + P_1)/2$$

- The parameter t selects a element $P(t)$ in between the start and end elements P_0 and P_1
- If $\mathbf{P}_0$ & $\mathbf{P}_1$ are points, then $\mathbf{P}(t)$ will be on the line between them:
- Hence *linear* interpolation



Requirement: Chop x-axis into 255 intervals

L(t) = A + (B-A) t

In component form, it can be written as follows:

x(t) = A.x + (B.x-A.x)t ; y(t) = A.y + (B.y-A.y)t

Code Class Point otherwise we need to write code for x and y separately and repeatedly

index.html | server.js | init.txt | histogram.js | Point.js | Vehicle.js

1

```javascript
class Point {
    constructor(x,y) {
        this.x = x;
        this.y = y;

    } // end constructor

    getXY() {
        return this.x + " " + this.y;
    } // end getXY()

    drawPoint (ctx) {
         var square = new Square(new Point(320, 240), 50, '#177b4b')
         square.draw(ctx)
   } // end drawPoint()
}// end class Point
```

2

```html
<!--<script  src="scripts/Excercise.js"> </script> -->
<script type="text/javascript"  src="/js/global.js"> </script>
<script type="text/javascript" src="/js/color2.js"> </script>
<script type="text/javascript" src="/js/Vehicle.js"> </script>
<script type="text/javascript" src="/js/Point.js"> </script>
<script type="text/javascript" src="/js/histogram.js"> </script>
```

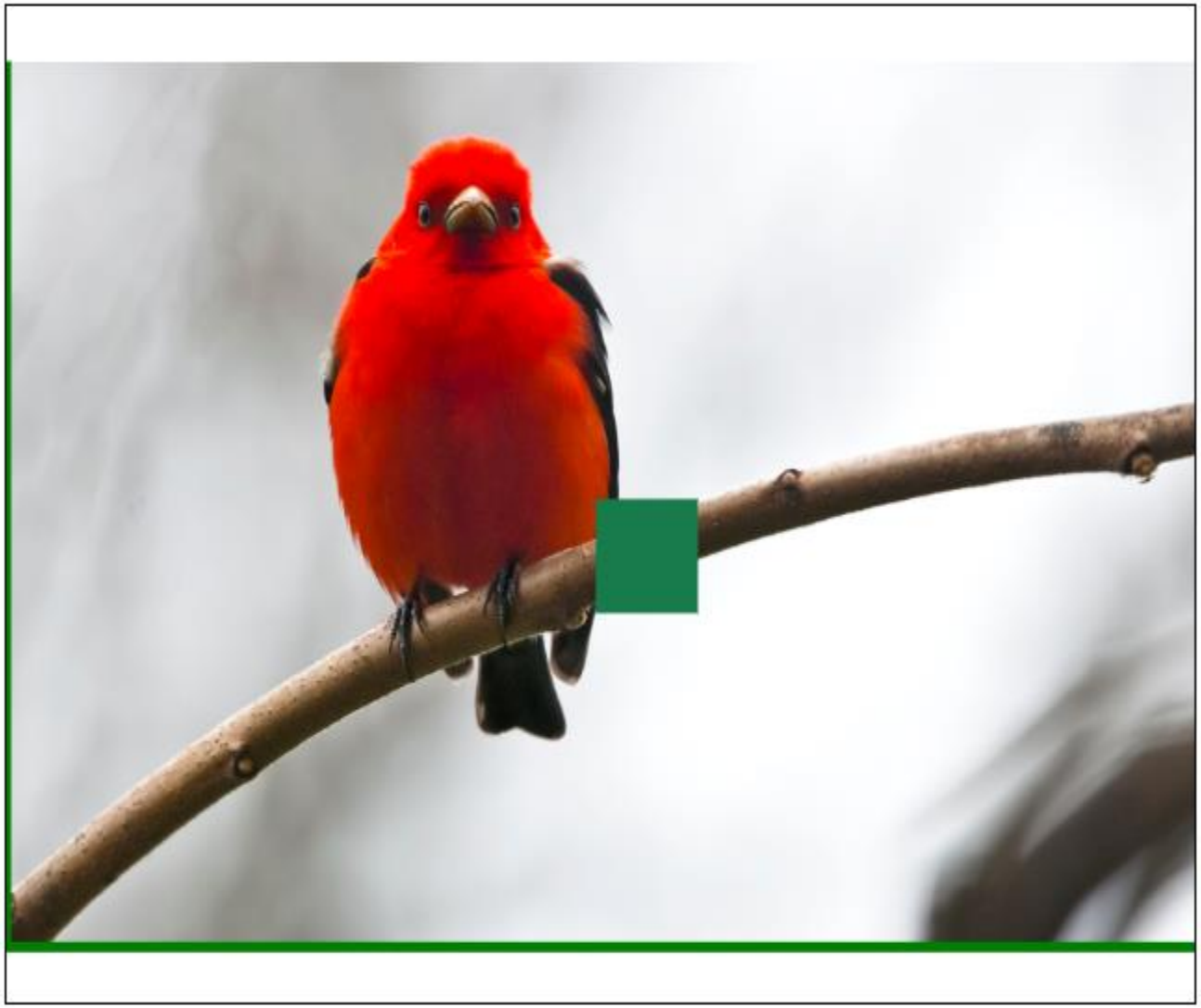
class Square to draw point

```
class Square {
  constructor (origin, length, colour) {
    this.origin = origin
    this.length = length
    this.colour = colour
  }

  draw (ctx) {
    // Draw clockwise
    ctx.beginPath()
    ctx.moveTo(this.origin.x, this.origin.y)
    ctx.lineTo(this.origin.x + this.length, this.origin.y)
    ctx.lineTo(this.origin.x + this.length, this.origin.y + this.length)
    ctx.lineTo(this.origin.x, this.origin.y + this.length)
    ctx.lineTo(this.origin.x, this.origin.y)
    // We want the border color and the fill color to match
    ctx.strokeStyle = this.colour;
    ctx.fillStyle = this.colour;
    // Color the border and the body
    ctx.stroke();
    ctx.fill();
    ctx.closePath();
  }
}//end class square
```

3

Output of using class Point and class Square

Note: Vehicle class is just my testing for class concept in java script



```
R: 640                                   histogram.js:31
G: 7209                                  histogram.js:32
B: 11544                                 histogram.js:33
▶ {rD: {…}, gD: {…}, bD: {…}}            histogram.js:35
256                                      histogram.js:38
256                                      histogram.js:39
256                                      histogram.js:40
▶ Point {x: 320, y: 240}                 histogram.js:61
                                         histogram.js:64
▶ Vehicle {make: "Toyota", model: "Corolla", c
olor: "Black"}
```

Testing class Point inside histogram(…..)

```
// Testing Objects in Java script
A = new Point (320,240); // center of canvas
console.log(A);
A.drawPoint(ctx); // Test Successful

// Testing public accces to class Point attributes
xx = A.x;
yy = A.y;
console.log(xx);
console.log(yy);
```

4

```
                                   histogram.js:35
▶{rD: {…}, gD: {…}, bD: {…}}
256                                histogram.js:38
256                                histogram.js:39
256                                histogram.js:40
▶Point {x: 320, y: 240}            histogram.js:61
320                                histogram.js:67
240                                histogram.js:68
                                   histogram.js:71
```

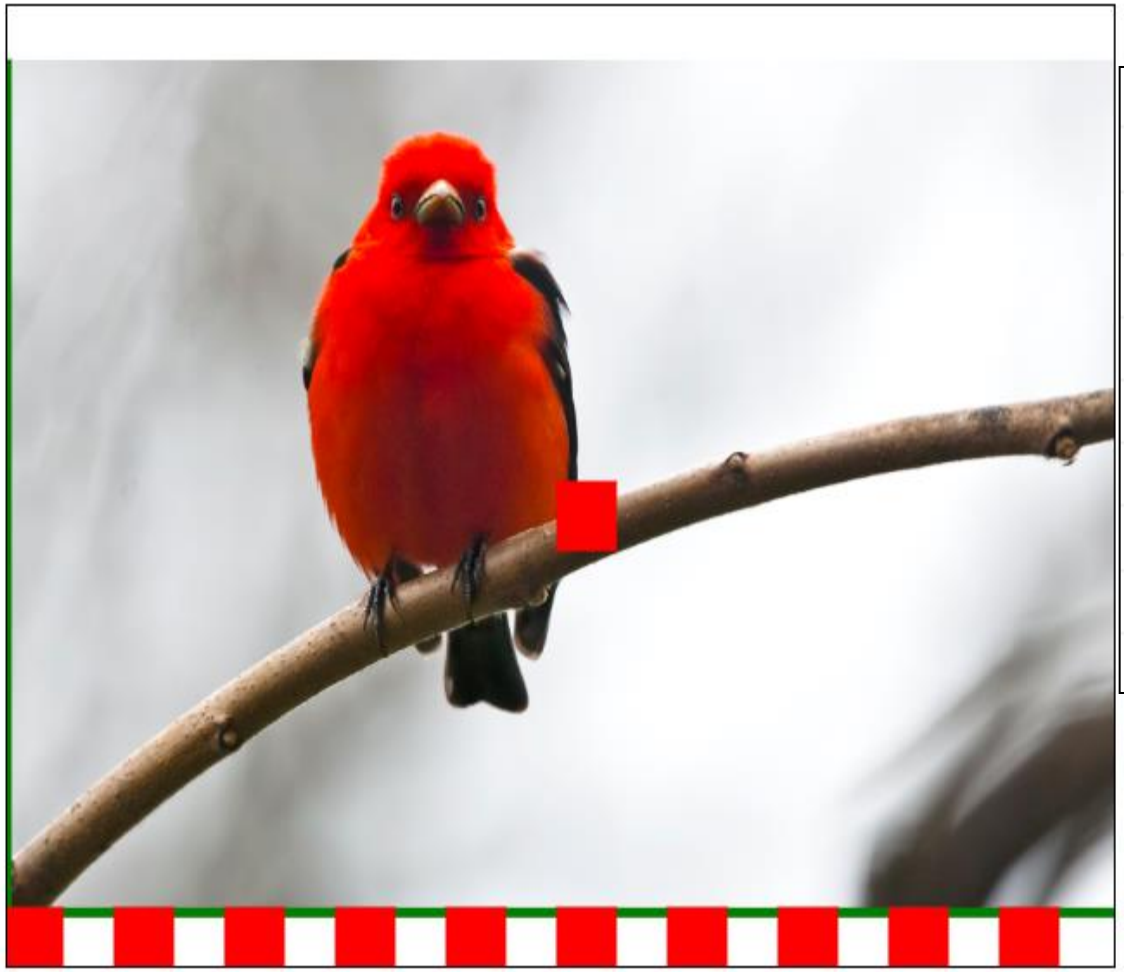
Now start working on Lerp()
See chap 4 & lectures: L(t) = A + (B-A) t

```
// Start working for lerp i.e. linear interpolation
source = new Point (xStart,yStart+renderableHeight); // bottom-right
destination = new Point (xStart+renderableWidth,yStart+renderableHeight); // bottom-left

numPoints = 10;
tMin=0.0; tMax=1.0;  delT = (tMax-tMin)/10;
var t = tMin;

let lerpX={}, lerpY={};   //instantiate the dictionaries

// note discrete  loop vs. loop on continuous variable
for(var i=0; i<numPoints; i++)   // L(t) = source+ (destination-source) * t
{
   lerpX[i] = Math.round((source.x + (destination.x-source.x) * t));
   lerpY[i] = Math.round((source.y + (destination.y-source.y) * t));
   t += delT;
  //lerpArray[i].x = (source.x + (destination.x-source.x) * t); // Array not workin
  //lerpArray[i].y = (source.y + (destination.y-source.y) * t  //
  tween = new Point (lerpX[i],lerpY[i]); // center of canvas

  tween.drawPoint(ctx);
  console.log(tween)
}
```

5

Output of $L(t) = A + (B-A)\ t$

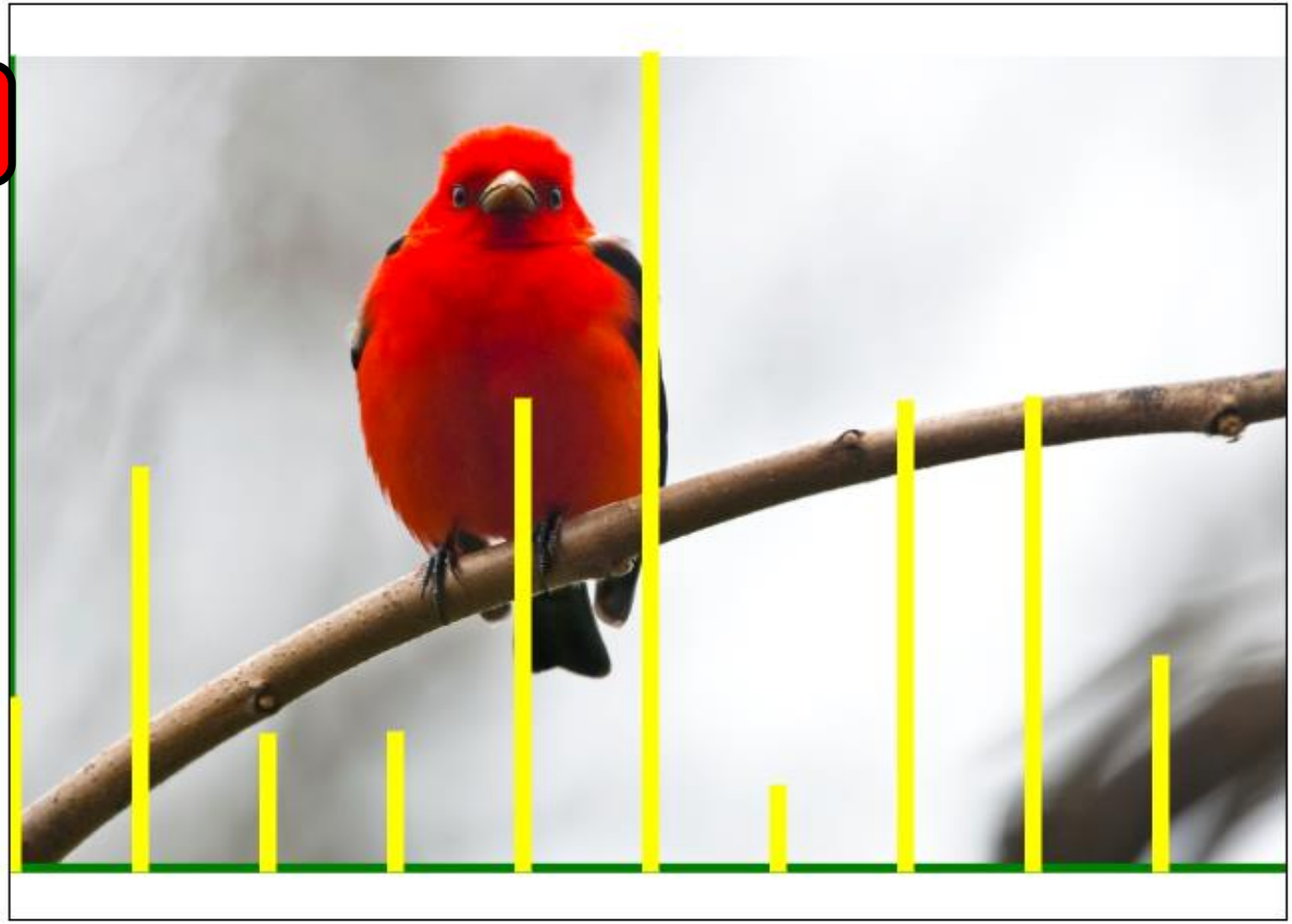
6



```
▸ Point {x: 0, y: 453}      histogram.js:103
▸ Point {x: 64, y: 453}     histogram.js:103
▸ Point {x: 128, y: 453}    histogram.js:103
▸ Point {x: 192, y: 453}    histogram.js:103
▸ Point {x: 256, y: 453}    histogram.js:103
▸ Point {x: 320, y: 453}    histogram.js:103
▸ Point {x: 384, y: 453}    histogram.js:103
▸ Point {x: 448, y: 453}    histogram.js:103
▸ Point {x: 512, y: 453}    histogram.js:103
▸ Point {x: 576, y: 453}    histogram.js:103
```

Try to draw rectangle with random height at point calculated through lerp i.e. modify class Point

7



Changes made to draw function of class Point

```
drawPoint (ctx) {
    //var square = new Square(new Point(320, 240), 30, '#177b4b')
    //var square = new Square(new Point(this.x, this.y), 10, "red")
    //square.draw(ctx)

    var a =0, b = 255;
    var h = Math.floor(a + Math.random() * (b - a));
    var f= true;
    console.log(h);

    var rect = new Rectangle(new Point (this.x-3,this.y),
                             new Point (this.x+3,this.y),
                             new Point (this.x+3,this.y-h),
                             new Point (this.x-3,this.y-h)
                             ,"yellow",f);
    var rect2 = new Rectangle(new Point (this.x-1,this.y),
                             new Point (this.x+1,this.y),
                             new Point (this.x+1,this.y-h),
                             new Point (this.x-1,this.y-h)
                             ,"black",!f);
    rect.draw(ctx);
    //rect2.draw(ctx);


  } // end drawPoint()
}// end class Point
```

7

Class Rect which we use to draw Point

```
class Rectangle {
  constructor (left, right, bottom, top,colour,flag) {
    this.left = left
    this.right = right
    this.bottom = bottom
    this.top = top
    this.colour=colour;
    this.flag = flag;

  }
  draw (ctx) { // draw rectangle using context
     // Draw anti-clockwise
    ctx.beginPath()
    ctx.moveTo(this.left.x, this.left.y)
    ctx.lineTo(this.right.x , this.right.y)
    ctx.lineTo(this.right.x , this.top.y)
    ctx.lineTo(this.left.x, this.top.y)
    //ctx.lineTo(this.left.x, this.left.y)
    // We want the border color and the fill color to match
    ctx.strokeStyle = this.colour;
    ctx.fillStyle = this.colour;
    // Color the border and the body
    ctx.stroke();
    if(this.flag) ctx.fill();
    ctx.closePath();
  }// end draw

  }// end class Rectangle
```

7

Problem 5: Mapping Pixel count to height of rectangle instead of using random as before

Requirement:

1) *Generate equal spaced point between start and end on x-axis using $L(t) = A + (B-A)\ t$*
2) *Read Image pixes(r,g,b,a) in matrix or Flat Format*
3) *Count pixels*
4) *rectHeight = normalized count * scaling fac*
5) *Draw rectangle positioned at L(t) with rectHeight and specified color*

At last successful in generating histogram with my own logic and code effort.

RGB is visible now…….
see code changes on next slides

Code changes in class Point

1

```
class Point {
    //constructor(x,y) {
     //    this.x = x;
     //    this.y = y;

    //} // end constructor

     constructor(x,y,h,colour) {
        this.x = x;
        this.y = y;
        this.h=h;
        this.colour = colour;

    } // end constructor
```

2

```
drawPoint (ctx) {
    //var square = new Square(new Point(320, 240), 30, '#177b4b')
    //var square = new Square(new Point(this.x, this.y), 10, "red")
    //square.draw(ctx)

    //var a =0, b = 255;
    //var h = Math.floor(a + Math.random() * (b - a));
    var f= true;
    console.log(this.h);

    var rect = new Rectangle(new Point (this.x-3,this.y),
                             new Point (this.x+3,this.y),
                             new Point (this.x+3,this.y-this.h),
                             new Point (this.x-3,this.y-this.h)
                             ,this.colour,f);
    var rect2 = new Rectangle(new Point (this.x-1,this.y),
                             new Point (this.x+1,this.y),
                             new Point (this.x+1,this.y-this.h),
                             new Point (this.x-1,this.y-this.h)
                             ,"black",!f);
    rect.draw(ctx);
    rect2.draw(ctx);
```

Code changes in function histogram(…)

```
// Start working for mapping of rgba to rectangle height for drawing

    maxRed = Math.max(data.rD[0]);
    maxGreen = Math.max(data.gD[0]);
    maxBlue = Math.max(data.bD[0]);

    source = new Point (xStart,yStart+renderableHeight,0,"green"); // bottom-right
    destination = new Point (xStart+renderableWidth,yStart+renderableHeight,0,"green"); // bottom-left

   numPoints = 255;
   tMin=0.0; tMax=1.0;  delT = (tMax-tMin)/255;
   var t = tMin;

   let lerpX={}, lerpY={}, redHeight={};   //instantiate the dictionaries
```

3

Code changes in function histogram(…)

4

```
// note discrete  loop vs. loop on continuous variable
for(var i=0; i<numPoints; i++)    // L(t) = source+ (destination-source) * t
{
   lerpX[i] = Math.round((source.x + (destination.x-source.x) * t));
   lerpY[i] = Math.round((source.y + (destination.y-source.y) * t));

   redHeight = (data.rD[i]/maxRed) * 100; // normalize and scale
   greenHeight = (data.gD[i]/maxGreen) * 100; // normalize and scale
   blueHeight = (data.bD[i]/maxBlue) * 100; // normalize and scale

   t += delT;

   tweenRed = new Point (lerpX[i],lerpY[i],redHeight,"red"); // center of canvas
   tweenGreen = new Point (lerpX[i],lerpY[i],greenHeight,"green"); // center of canvas
   tweenBlue = new Point (lerpX[i],lerpY[i],blueHeight,"blue"); // center of canvas

  tweenRed.drawPoint(ctx);
  tweenGreen.drawPoint(ctx);
  tweenBlue.drawPoint(ctx);

  //console.log(tween);
}
```

Time to Sleep finally……

6

Next Lab 07-Lab08

Chap 10 Raster Tool for Images
Image averaging, lerp, blend, bitwise,
Transformation on images