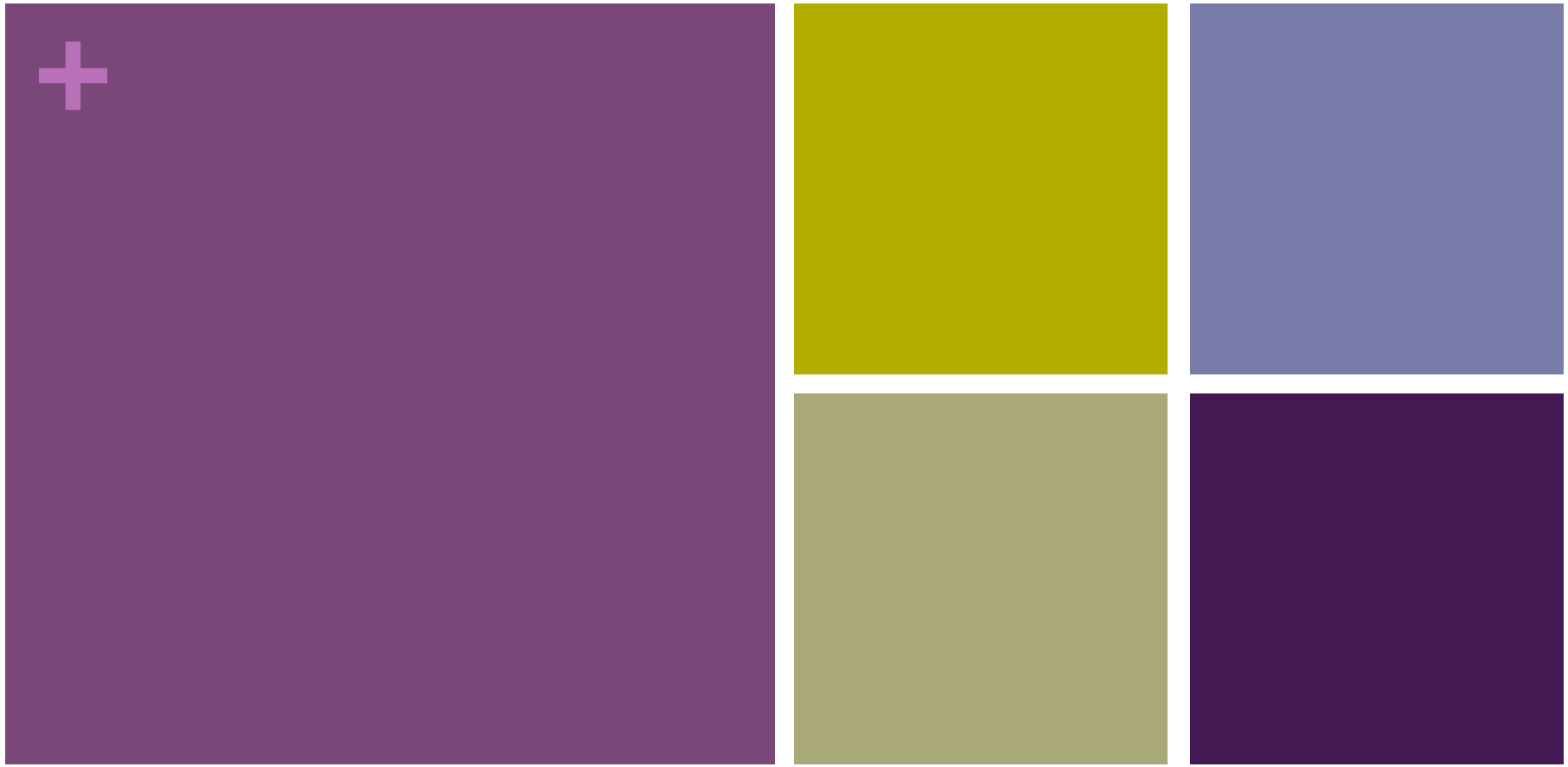


Refactoring, Classes, and Loops

Review

- Display 5 Coins
  - display paragraph became display table
  - added variables, arrays, functions, and loops
  - added file input

Display table of 5 coins

My Coins from the United States:

type	date	condition
Nickel	1875	uncirculated
Quarter	1916	very worn
penny	1946	mint
silver dollar	2015	brilliant uncirculated
Half-dollar	1970	good

Just using function calls to text

```
textSize(width/30);
text("My Coins from the United States:", 10, 10, 0.8*width, width/20);
// plot the header
text("type",50, 10 + width/20, 0.8*width, width/20);
text("date",50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition",50 + 400, 10 + width/20, 0.8*width, width/20);
// plot the first coin
text("Nickel",50, 10 + 2*width/20, 0.8*width, width/20);
text("1875",50 + 200, 10 + 2*width/20, 0.8*width, width/20);
text("uncirculated",50 + 400, 10 + 2*width/20, 0.8*width, width/20);
```

Exercise: Refactor FiveCoins

Step 1: make more general

- variables
- repeated code changes
  - functions
  - variables
  - loops
- Make Arrays of Strings or Array of Coins

Using variables, arrays, and loops

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                  rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,    rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Let's look at the header
[just function calls to text()]

```
// plot the header
text("type",50, 10 + width/20, 0.8*width, width/20);
text("date",50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition",50 + 400, 10 + width/20, 0.8*width, width/20);
```

Let's look at the header
1st repeated variable: rowHeight

```
// plot the header
text("type",50, 10 + width/20, 0.8*width, width/20);
text("date",50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition",50 + 400, 10 + width/20, 0.8*width, width/20);
```

Let's look at the header
2nd repeated variable, rowWidth

```
// plot the header
text("type",50, 10 + width/20, 0.8*width, width/20);
text("date",50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition",50 + 400, 10 + width/20, 0.8*width, width/20);
```

Let's look at the header
3rd repeated variable rowStart

```
// plot the header
text("type",50, 10 + width/20, 0.8*width, width/20);
text("date",50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition",50 + 400, 10 + width/20, 0.8*width, width/20);
```

Let's look at the header
4th repeated variable, left

```
// plot the header
text("type", 50  10 + width/20, 0.8*width, width/20);
text("date", 50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition", 50 + 400, 10 + width/20, 0.8*width, width/20);
```

Let's look at the header
There are 5 other values as well.

```
// plot the header
text("type", 50, 10 + width/20, 0.8*width, width/20);
text("date", 50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition", 50 + 400, 10 + width/20, 0.8*width, width/20);
```

Let's compare the headers

```
// plot the header
text("type", 50, 10 + width/20, 0.8*width, width/20);
text("date", 50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition", 50 + 400, 10 + width/20, 0.8*width, width/20);
```

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                  rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,    rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
```

Let's compare the headers rStart, reduces computation

```
// plot the header
text("type",50, 10 + width/20, 0.8*width, width/20);
text("date",50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition",50 + 400, 10 + width/20, 0.8*width, width/20);
```

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],   left,                  rStart, rowWidth, rowHeight)
text(header[CMID],    left + colTwoShift,    rStart, rowWidth, rowHeight)
text(header[CRIGHT], left + colThreeShift, rStart  rowWidth, rowHeight)
// plot the data
```

Let's compare the headers left

```
// plot the header
text("type", 50, 10 + width/20, 0.8*width, width/20);
text("date", 50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition", 50 + 400, 10 + width/20, 0.8*width, width/20);
```

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT], left,                    rStart, rowWidth, rowHeight);
text(header[CMID],  left + colTwoShift,      rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
```

Let's compare the headers rowWidth

```
// plot the header
text("type", 50, 10 + width/20, 0.8*width, width/20);
text("date", 50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition", 50 + 400, 10 + width/20, 0.8*width, width/20);
```

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                  rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,    rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
```

Let's compare the headers
4 rep. var. & 1 rep. expression

```
// plot the header
text("type", 50, 10 + width/20, 0.8*width, width/20);
text("date", 50 + 200, 10 + width/20, 0.8*width, width/20);
text("condition", 50 + 400, 10 + width/20, 0.8*width, width/20);
```

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT], left,                    rStart, rowWidth, rowHeight)
text(header[CMID],  left + colTwoShift,      rStart, rowWidth, rowHeight)
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight)
// plot the data
```

String literals to array variables & int literals to variables

```
// plot the header
text("type",      0 + width/20, 0.8*width, width/20);
text("date",      200  10 + width/20, 0.8*width, width/20);
text("condition" 50 + 400, 10 + width/20, 0.8*width, width/20);
```

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                    rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift       rStart, rowWidth, rowHeight);
text(header[CRIGHT]  left + colThreeShift,    rStart, rowWidth, rowHeight);
// plot the data
```

Now let's look at the loop

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                  rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,    rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Now let's look at the loop

Starting Position for header is *one* row down.

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                 rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,   rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Now let's look at the loop

Starting Position for header is *one* row down.

Starting Position for first coin is *two* rows down.

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                 rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,   rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift, rStart, rowWidth, rowHeight);
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Now let's look at the loop

Starting Position for header is *one* row down.

Starting Position for first coin is *two* rows down.

```
// plot the header
float rStart = rowStart + rowHeight;
text(header[CLEFT],  left,                  rStart, rowWidth, rowHeight);
text(header[CMID],   left + colTwoShift,    rStart, rowWidth, rowHeight);
text(header[CRIGHT], left + colThreeShift,  rStart, rowWidth, rowHeight);
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Loop uses 3 arrays of Strings.

Arrays of Strings vs. Array of Objects

Arrays of Strings

```
String[] types;
String[] years;
String[] conditions;
```

- Alternatively:
  - `String[][] cells;`

Array of Objects

```
Coin[] coins;

class Coin {
    String type;
    String year;
    String condition;
}
```

Loop through your array(s) to display your coins row by row.

Arrays of Strings vs. Array of Objects

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Array of Objects

```
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(coins[i].type,      left,                 rStart, rowWidth, rowHeight);
  text(coins[i].date,      left + colTwoShift,   rStart, rowWidth, rowHeight);
  text(coins[i].condition, left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Arrays of Strings vs. Array of Objects

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

Array of Objects

```
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(coins[i].type,      left,                rStart, rowWidth, rowHeight);
  text(coins[i].date,      left + colTwoShift,  rStart, rowWidth, rowHeight);
  text(coins[i].condition, left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

What if we wanted this code in the Coin class?

Arrays of Strings vs. 2D Array of Strings

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

2D Array of Strings

```
// plot the data
for (int i = 0; i < coins.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  for (int j = 0; j < coins[i].length; ++j) {
    text(coins[i][j], left + j * colShift, rStart, rowWidth, rowHeight);
  }
}
```

Arrays of Strings vs. 2D Array of Strings

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

2D Array of Strings

```
// plot the data
for (int i = 0; i < coins.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  for (int j = 0; j < coins[i].length; ++j) {
    text(coins[i][j], left + j * colShift, rStart, rowWidth, rowHeight);
  }
}
```

Column value is based based on *j*.

Arrays of Strings vs. 2D Array of Strings

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

2D Array of Strings

```
// plot the data
for (int i = 0; i < coins.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  for (int j = 0; j < coins[i].length; ++j) {
    text(coins[i][j], left + j*colShift, rStart, rowWidth, rowHeight);
  }
}
```

Column value is based based on *j*.

Column Position is based based on *j*.

Arrays of Strings vs. 2D Array of Strings

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

2D Array of Strings

```
// plot the data
for (int i = 0; i < coins.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  for (int j = 0; j < coins[i].length; ++j) {
    text(coins[i][j], left + j * colShift, rStart, rowWidth, rowHeight);
  }
}
```

One value with multiplier for shifting columns instead of two.

Arrays of Strings vs. 2D Array of Strings

Arrays of Strings

```
// plot the data
for (int i = 0; i < dates.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  text(types[i], left, rStart, rowWidth, rowHeight);
  text(dates[i], left + colTwoShift, rStart, rowWidth, rowHeight);
  text(condition[i], left + colThreeShift, rStart, rowWidth, rowHeight);
}
```

2D Array of Strings

```
// plot the data
for (int i = 0; i < coins.length; ++i) {
  rStart = rowStart + (i+2)*rowHeight;
  for (int j = 0; j < coins[i].length; ++j) {
    text(coins[i][j], left + j * colShift, rStart, rowWidth, rowHeight);
  }
}
```

Exercise: Refactor FiveCoins

Step 1: make more general

- variables
- repeated code changes
  - functions
  - variables
  - loops
- Make Arrays of Strings or Array of Coins

Step 2: read a file

- format:
  - comma separated
  - header line first
  - any number of rows after the header line.

Step 2: read a file
Using Arrays of Strings

```
String[] rows = loadStrings("coins.csv");
String[] header = split(rows[0], ",");
String[] types = new String[rows.length - 1];
String[] dates = new String[rows.length - 1];
String[] condition = new String[rows.length - 1];

for (int i = 0; i < types.length; ++i ) {
  String[] col = split(rows[i+1], ",");
  types[i] = col[CLEFT];
  dates[i] = col[CMID];
  condition[i] = col[CRIGHT];
}
```

Step 2: read a file
Using Array of Coins

```
String[] rows = loadStrings("coins.csv");
String[] header = split(rows[0], ",");
Coin[] coins = new Coin[rows.length - 1];

for (int i = 0; i < coins.length; ++i ) {
  String[] col = split(rows[i+1], ",");
  coins[i].type      = col[CLEFT];
  coins[i].dates     = col[CMID];
  coins[i].condition = col[CRIGHT];
}
```

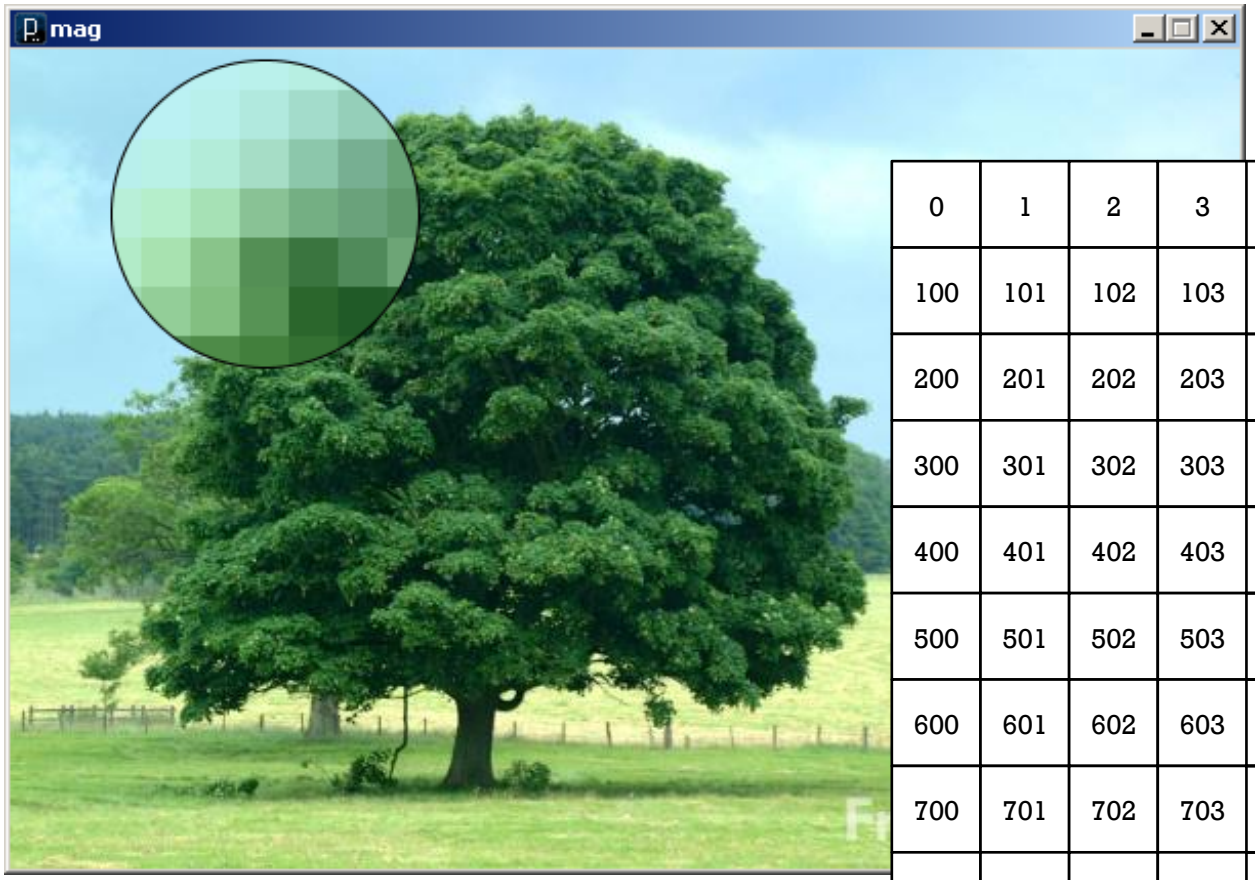
Step 2: read a file
Using 2D Array of Strings

```
String[] rows = loadStrings("coins.csv");
String[] header = split(rows[0], ",");
String[][] coins = new String[rows.length - 1][];

for (int i = 0; i < coins.length; ++i ) {
  coins[i] = split(rows[i+1], ",");
}
```

Let's apply these ideas to Images

- Image Processing
- … computing with and about data,
- … where "data" includes the values and relative locations of the colors that make up an image.

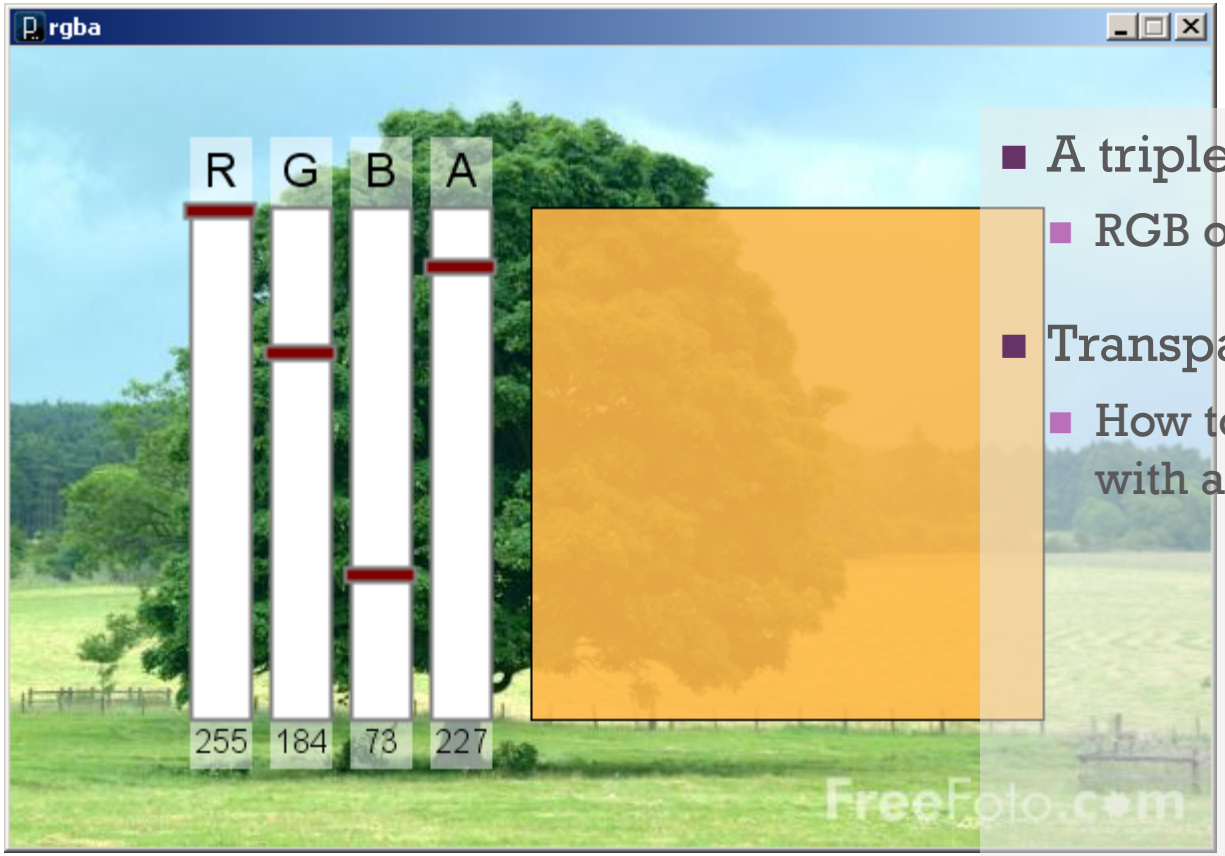
An image is an array of colors



0	1	2	3	…		98	99
100	101	102	103	…		198	199
200	201	202	203	…		298	299
300	301	302	303	…		398	399
400	401	402	403	…		498	499
500	501	502	503	…		598	599
600	601	602	603	…		698	699
700	701	702	703	…		798	799
800	801	802	803	…		898	899
⋮	⋮	⋮	⋮	…		⋮	⋮

Pixel : Picture Element

Color



- A triple of bytes [0, 255]
  - RGB or HSB
- Transparency (alpha)
  - How to blend a new pixel color with an existing pixel color

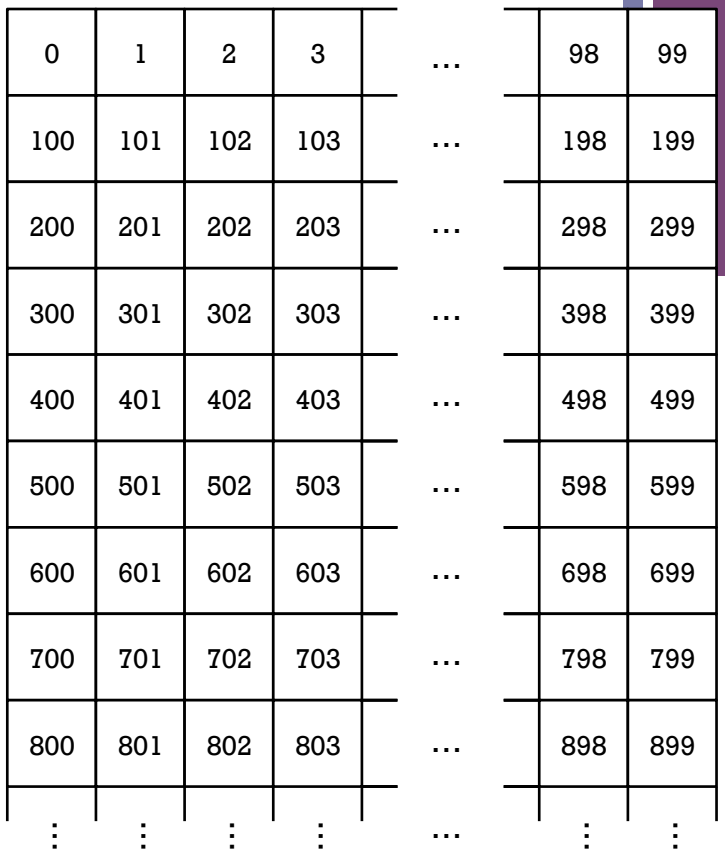
Accessing the pixels of a sketch

- loadPixels()
  - Loads the color data out of the sketch window into a 1D array of colors named pixels[]
  - The pixels[] array can be modified

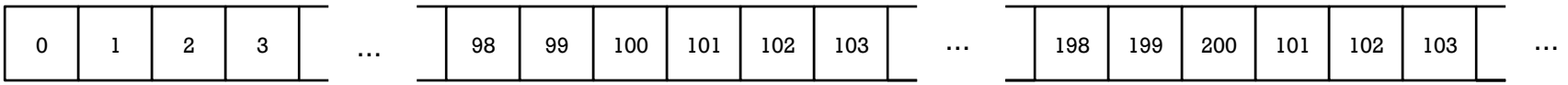
- updatePixels()
  - Copies the color data from the pixels[] array back to the sketch window

A 100-pixel wide image

- First pixel at index 0
- Right-most pixel in first row at index 99
- First pixel of second row at index 100



The pixels[] array is one-dimensional



Your Canvas as an Image

```
// whiteNoise

void setup() {
  size(400, 300);
}

void draw() {
  float b;

  // Load colors into the pixels array
  loadPixels();

  // Fill pixel array with a random
  // grayscale value
  for (int i=0; i<pixels.length; i++) {
    b = random(0, 255);
    pixels[i] = color(b);
  }

  // Update the sketch with pixel data
  updatePixels();
}
```

See also colorNoise.pde

Accessing Pixels as a 2D Array

- Pixels can be accessed as a 2D array using the following formula:

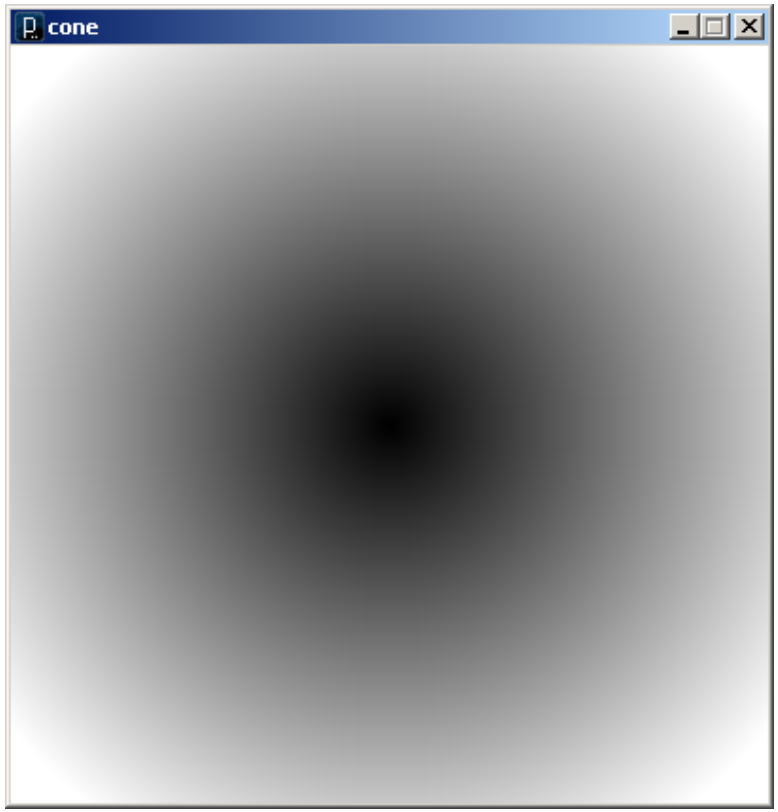
```
index = row * width + column
index = y   * width + x
```

- Using 0-based indices

```
int idx      = width * row + column;
pixels[idx] = color(b);
```

Radial cone

```
// cone
void setup() {
  size(400, 400);

  // Load colors into the pixels array
  loadPixels();

  // Access pixels as a 2D array
  for (int y=0; y<height; y++) {
    for (int x=0; x<width; x++) {

      // Compute distance to center point
      float d =
          dist(x, y, width/2, height/2);

      int idx = width*y + x;

      // Set pixel as distance to center
      pixels[idx] = color(d);
    }
  }

  // Update the sketch with pixel data
  updatePixels();
}
```



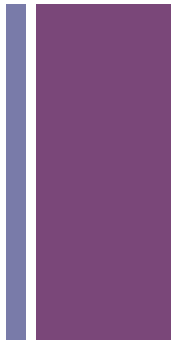
Rendering Images in a Sketch

- Image data can be loaded from a file using loadImage() method, and drawn on a sketch with the image() command

```
PImage img = loadImage("myImage.jpg");
image(img, 0, 0);
```

- The PImage object also permits individual pixel color data to be modified.
  - like the sketch window

PImage



Fields

```
  width       - the width of the image
  height      - the height of the image
  pixels[]    - the image pixel colors
                (after a call to loadPixels())
```

PImage

Methods

loadPixels()

Loads the color data out of the PImage object into a 1D array of colors named `pixels[]`.

updatePixels()

Copies the color data from the pixels[] array back to the PImage object.

Also

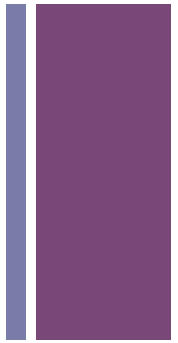
red(color)	extract the red component of from color
blue(color)	extract the green component from a color
green(color)	extract the blue component from a color

PImage

Methods (Cont'd)

get(…)	Reads the color of any pixel or grabs a rectangle of pixels
set(…)	Writes a color to any pixel or writes an image into another
copy(…)	Copies pixels from one part of an image to another
mask(…)	Masks part of the image from displaying
save(…)	Saves the image to a TIFF, TARGA, PNG, or JPEG file
resize(…)	Changes the size of an image to a new width and height
blend(…)	Copies a pixel or rectangle of pixels using different blending modes
filter(…)	Processes the image using one of several algorithms

get(…)



- Get a single pixel (very slow)

```
Color c = img.get(x, y);
```

- Get a rectangular range of pixels

```
PImage img2 = img.get(x, y, w, h);
```

Exam 2 Study Guide Pg. 1 of 5

- Arrays (of any type:
  `boolean, char, int, float, PImage, String, PVector`, or an arbitrary Class)
  - declare
  - instantiate
  - initialize all values
  - get the size
  - assign a value to an arbitrary location
  - get a value from an arbitrary location
  - iterate through any number (1 to array.length) of elements of an array starting from any valid index.
  - compare a value in an array with another value
  - compare two values in an array
  - use built in functions on arrays such as max(), min(), etc.

Exam 2 Study Guide Pg. 2 of 5

- ArrayLists (of any type:
  `boolean, char, int, float, PImage, String, PVector,` or an arbitrary Class)
  - declare
  - instantiate
  - initialize all values
  - get the size
  - assign a value to an arbitrary location
  - get a value from an arbitrary location
  - iterate through any number (1 to arrayList.size()) of elements of an ArrayList starting from any valid index.
  - compare a value in an ArrayList with another value
  - compare two values in an ArrayList

Exam 2 Study Guide Pg. 3 of 5

- String/PVector/PImage/(arbitrary class defined in test)*
  - declare
  - instantiate (using constructor)
  - using keyword `this` *
  - identify*/use/assign fields
  - identify*/call methods
  - compare the values of 2 instances of the class using .equals()
  - use Processing methods and operators on Strings●
  - write the contents of (fill in) a method based on description*

* just for arbitrary defined class.

● just for Strings.

Exam 2 Study Guide Pg. 4 of 5

- loops
  - read a loop and identify
    - the number of times the loop will iterate and/or the reason the loop will stop
    - the values of each variable used in the loop at each iteration and after the loop has ended
    - the output of the loop (if anything is printed or displayed)
    - Ex:

```
int[] test = { 3 , 7, 11 };
int c = 10;
for (int i = 0; i < test.length; ++i) {
  c += i;
  test[i] = test[i] * test[i];
  c *= (i+1);
} // Note: the variables in the loop
  // are c, i, and test
```

Exam 2 Study Guide Pg. 5 of 5

- functions/methods
  - identify name/parameters/return type
  - write a function from a description
- Processing functions (you should feel comfortable with these)
  - print(), println(), loadStrings(), color(), random(), abs(), ceil(), dist(), floor(),pow(),round(),sq(),sqrt(), cos(),sin(),tan()
- Recursion
  - given a base case(s) and a recursive case, write the recursive function. For example: write a recursive function fib, that returns an int
    base cases: fib(1) = 1, fib(2) = 1
    recursive case: fib(n) = fib(n-2) + fib(n-1)
- Be comfortable with material from Exam 1.