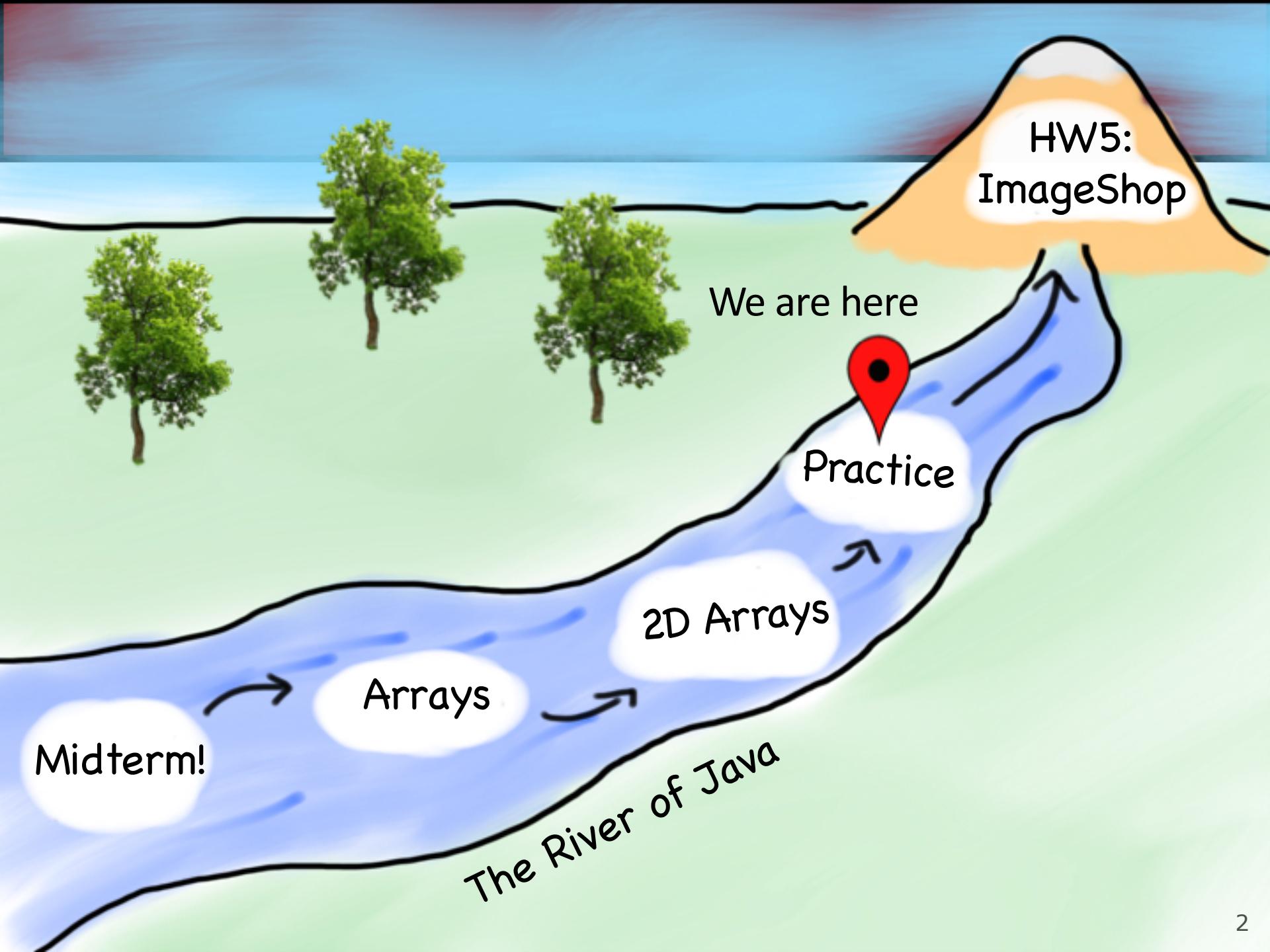


CS 106A, Lecture 18

Practice with 1D and 2D Arrays



Plan for Today

- Recap: 2D Arrays and Images
- Practice: Shrink
- Practice: Cryptogram
- Practice: Tic-Tac-Toe

Plan for Today

- Recap: 2D Arrays and Images
- Practice: Shrink
- Practice: Cryptogram
- Practice: Tic-Tac-Toe

The Matrix



Image used under "fair use" for educational purposes.

Source: <https://www.themarysue.com/decoding-the-transgender-matrix-the-matrix-as-a-transgender-coming-out-story/>

2D Arrays ("Matrices")



WELCOME TO
THE MATRIX!!!!!!

2D Arrays

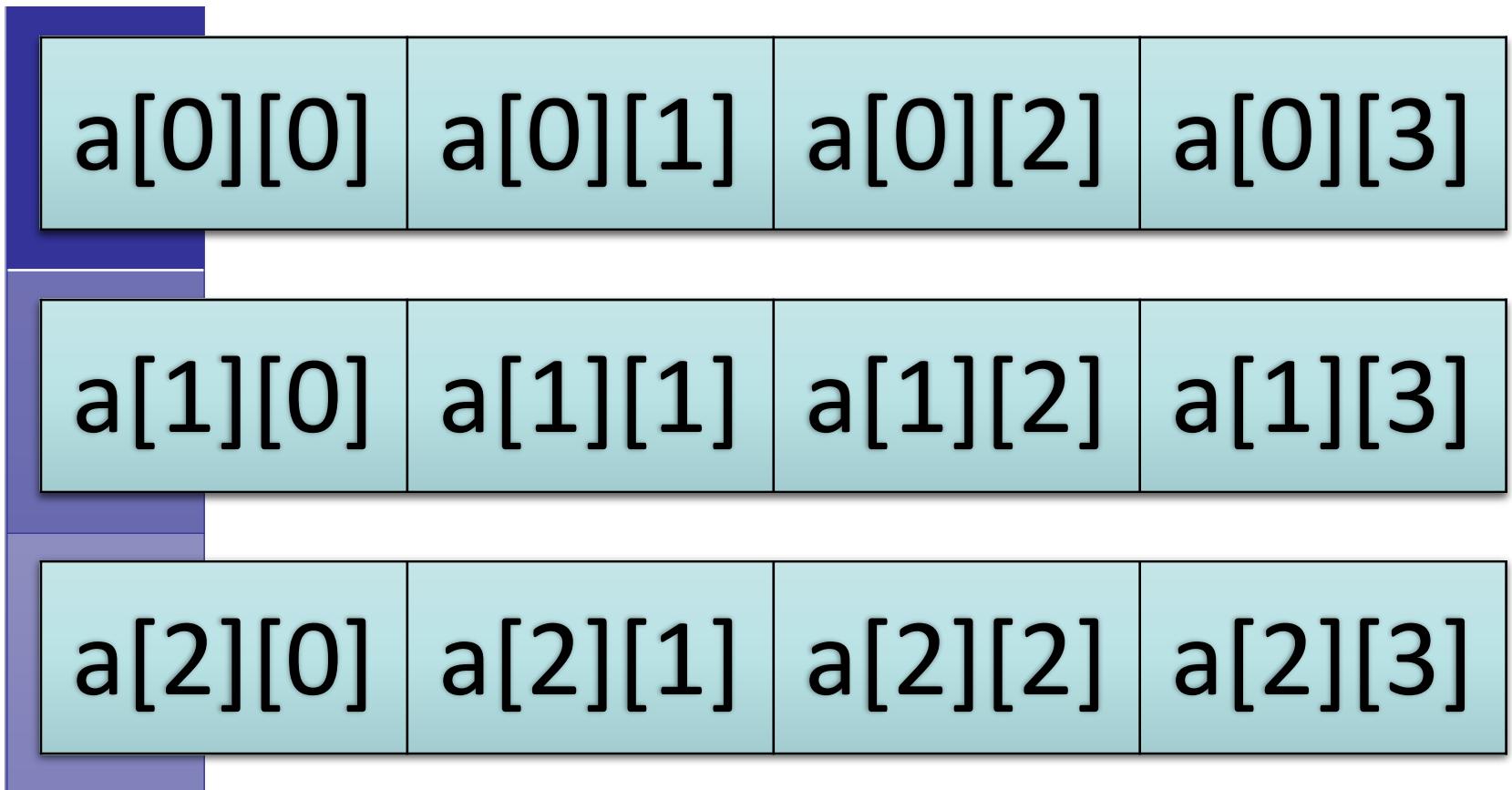
```
type[ ][] name = new type[rows][columns];
```

```
int[][] a = new int[3][5];
```

	0	1	2	3	4
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

2D Arrays = Arrays of Arrays!

```
int[][] a = new int[3][4];  
int[] firstRow = a[0];
```



Summary: 2D Arrays

- Make a new 2D array

```
type[][] name = new type[rows][columns];
```

- Get and set values using bracket notation

```
name[row][col]           // get elem at row,col
```

```
name[row][col] = value; // set elem at row,col
```

- Get the number of rows and columns

```
arr.Length    // # rows
```

```
arr[0].Length // # columns
```

- Iterate over a 2D array using a double for-loop

```
for (int row = 0; row < arr.length; row++) {  
    for (int col = 0; col < arr[0].length; col++) {  
        // do something with arr[row][col];  
    }  
}
```

Limitations of 2D Arrays

- Unlike 1D arrays, you *cannot compare 2D arrays with `Arrays.equals`.* You must use `Arrays.deepEquals`.

```
int[][] a1 = ...
int[][] a2 = ...
if (Arrays.deepEquals(a1, a2)) { ... }
```

- A 2D array does not know how to print itself:

```
int[][] a = new int[rows][cols];
println(a);                                // [[I@8cf420
println(Arrays.toString(a));    // [[I@6b3f44, [I@32c2a8]...
// [[0, 1, 2, 3, 4], [1, 2, ...
println(Arrays.deepToString(a));
```

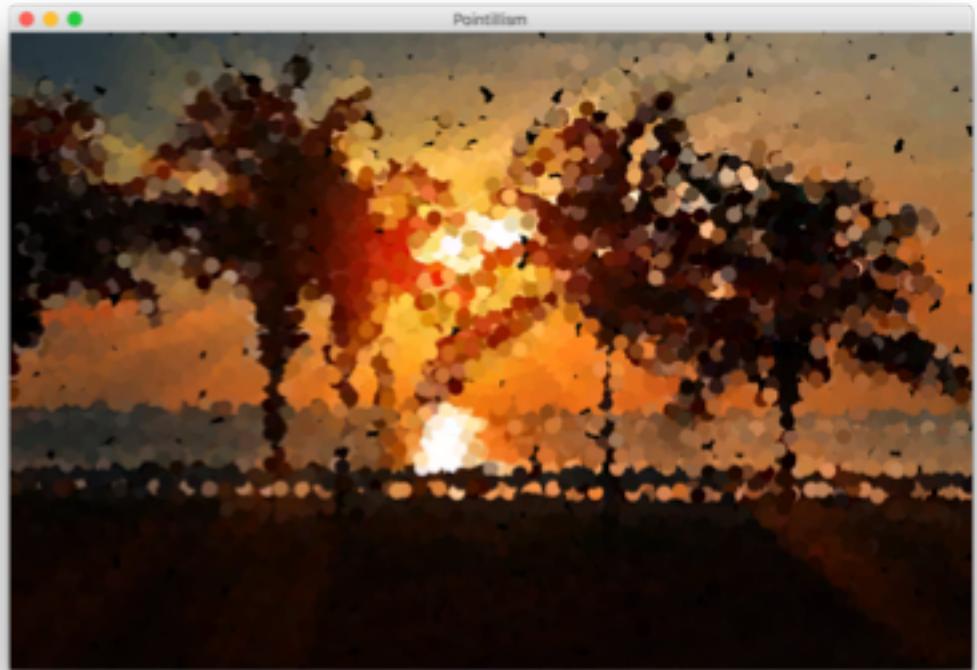
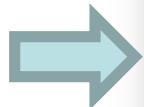
Images

Images are just grids (2D arrays) of pixels! Pixels are just integer values that have red, green, and blue components (each between 0 and 255).



Example: Pointillism

Pointillism is an art style where many small dots of color are combined to make a larger image.



Red, Green and Blue in one int?

Images *encode* the R, G, and B values (between 0 and 255) of a pixel into a single integer. You can convert between this **pixel value** and the individual **RGB values**.

```
int[][] pixels = image.getPixelArray();
int px = pixels[0][0];
int red = GImage.getRed(px);
int green = GImage.getGreen(px);
int blue = GImage.getBlue(px);
```

Creating New Pixels

Images *encode* the R, G, and B values (between 0 and 255) of a pixel into a single integer. You can convert between this **pixel value** and the individual **RGB values**.

You can also create pixels with your own RGB values.

```
int r = ...  
int g = ...  
int b = ...  
int pixel = GImage.createRGBPixel(r, g, b);
```

Images as 2D Arrays

We can get a GImage as a 2D array of pixels, and modify it any way we want. Then, we can create a new GImage with the modified pixels.

```
GImage img = new GImage("res/snowman.jpg");
int[][] pixels = img.getPixelArray();
... // (modify pixels)
img.setPixelArray(pixels); // update image

// or make a new GImage
GImage newImg = new GImage(pixels);
```

Modifying Image Pixels

- There are many cool image algorithms based around modifying individual pixels in an image: grayscale, brighten, normalize, remove red-eye...

grayscale



zoom



GImage Pixel Methods

```
GImage img = new GImage("res/daisy.jpg");
```

Method name	Description
<code>img.getPixelArray()</code>	returns pixels as 2D array of ints, where each int in the array contains all 3 of Red, Green, and Blue merged into a single integer
<code>img.setPixelArray(array);</code>	updates pixels using the given 2D array of ints
<code>GImage.createRGBPixel(r, g, b)</code>	returns an int that merges the given amounts of red, green and blue (each 0-255)
<code>GImage.getRed(px)</code> <code>GImage.getGreen(px)</code> <code>GImage.getBlue(px)</code>	returns the redness, greenness, or blueness of the given pixel as an integer from 0-255

Recap: Modifying Pixels

- **Extract** pixel RGB colors with `GImage.getRed/Blue/Green`.

```
int red    = GImage.getRed(pixels[0][0]);      // 0-255  
int green = GImage.getGreen(pixels[0][0]);     // 0-255  
int blue   = GImage.getBlue(pixels[0][0]);     // 0-255
```

- **Modify** the color components for a given pixel.

```
red = 0;    // remove redness
```

- **Combine** the RGB back together into a single `int`.

```
pixels[0][0] = GImage.createRGBPixel(red, green, blue);
```

- **Update** the image with your modified pixels when finished.

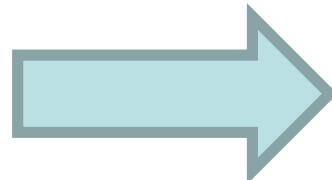
```
image.setPixelArray(pixels);
```

Plan for Today

- Recap: 2D Arrays and Images
- Practice: Shrink
- Practice: Cryptogram
- Practice: Tic-Tac-Toe

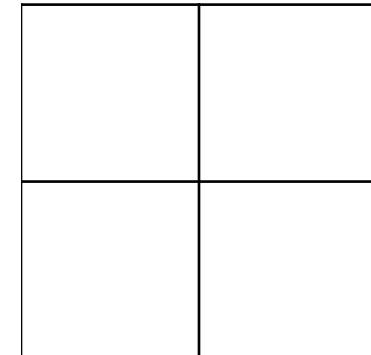
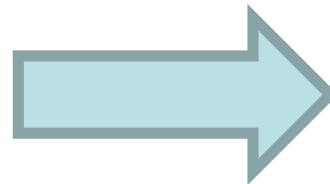
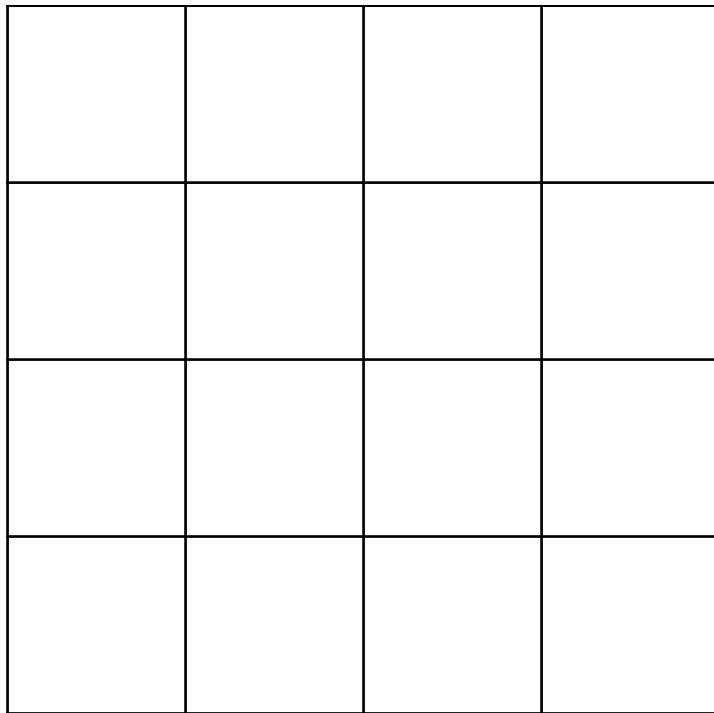
Shrink

Let's write a program that can *shrink* an image to $\frac{1}{2}$ its original size.



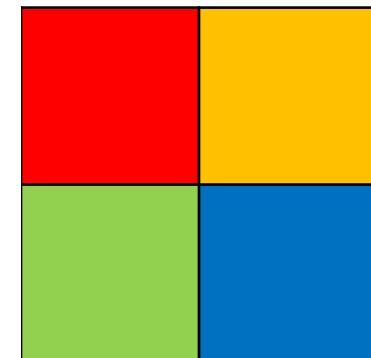
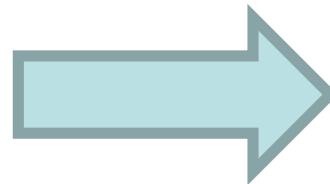
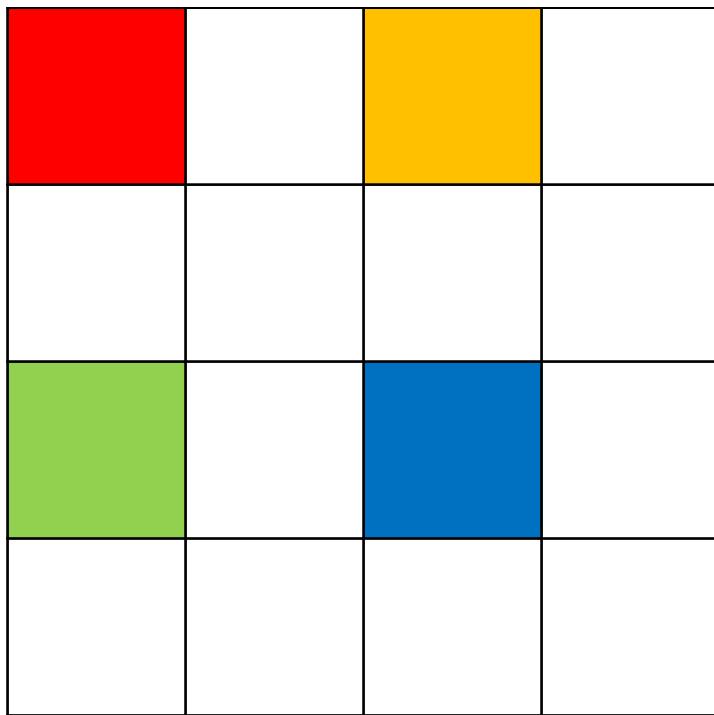
Shrink

Given a pixel (x, y) in our smaller image, how do we know which pixel in our larger image should go there?



Shrink

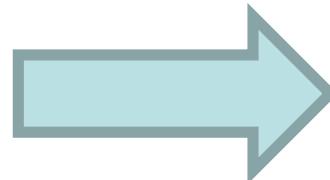
Given a pixel (x, y) in our smaller image, how do we know which pixel in our larger image should go there?



Shrink

Given a pixel (x, y) in our smaller image, how do we know which pixel in our larger image should go there?

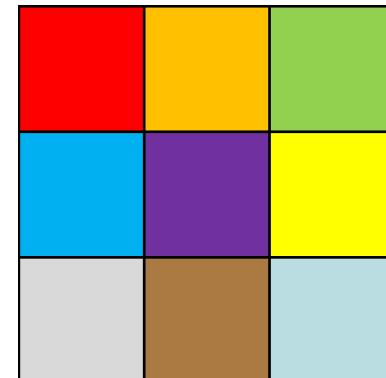
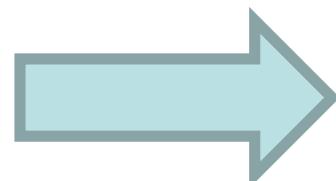
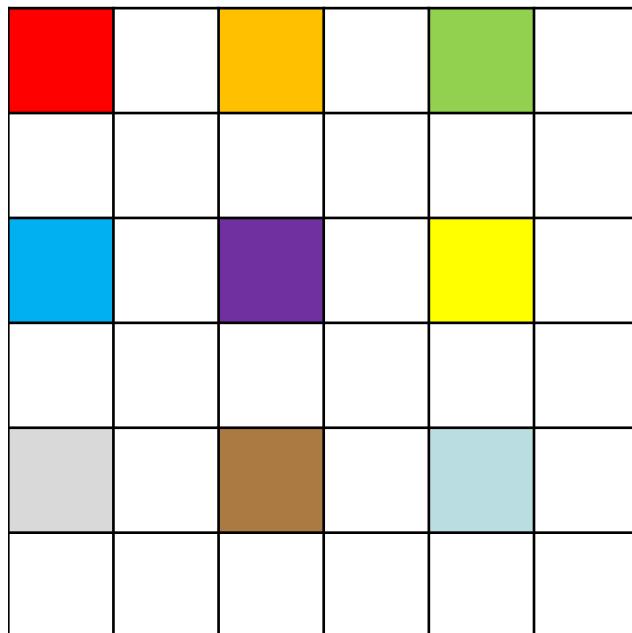
(0 , 0)		(0, 2)	
(2, 0)		(2, 2)	



(0, 0)	(0, 1)
(1, 0)	(1, 1)

Shrink

Given a pixel (x, y) in our smaller image, how do we know which pixel in our larger image should go there?



Shrink

```
int[][] pixels = image.getPixelArray();
int[][] result =
    new int[pixels.length/2][pixels[0].length/2];

for (int r = 0; r < result.length; r++) {
    for (int c = 0; c < result[0].length; c++) {
        result[r][c] = pixels[r*2][c*2];
    }
}

image.setPixelArray(result);
```

Shrink

```
int[][] pixels = image.getPixelArray();
int[][] result =
    new int[pixels.length/2][pixels[0].length/2];

for (int r = 0; r < result.length; r++) {
    for (int c = 0; c < result[0].length; c++) {
        result[r][c] = pixels[r*2][c*2];
    }
}

image.setPixelArray(result);
```

Shrink

```
int[][] pixels = image.getPixelArray();
int[][] result =
    new int[pixels.length/2][pixels[0].length/2];

for (int r = 0; r < result.length; r++) {
    for (int c = 0; c < result[0].length; c++) {
        result[r][c] = pixels[r*2][c*2];
    }
}

image.setPixelArray(result);
```

Shrink

```
int[][] pixels = image.getPixelArray();
int[][] result =
    new int[pixels.length/2][pixels[0].length/2];

for (int r = 0; r < result.length; r++) {
    for (int c = 0; c < result[0].length; c++) {
        result[r][c] = pixels[r*2][c*2];
    }
}

image.setPixelArray(result);
```

Shrink

```
int[][] pixels = image.getPixelArray();
int[][] result =
    new int[pixels.length/2][pixels[0].length/2];

for (int r = 0; r < result.length; r++) {
    for (int c = 0; c < result[0].length; c++) {
        result[r][c] = pixels[r*2][c*2];
    }
}

image.setPixelArray(result);
```

Shrink

```
int[][] pixels = image.getPixelArray();
int[][] result =
    new int[pixels.length/2][pixels[0].length/2];

for (int r = 0; r < result.length; r++) {
    for (int c = 0; c < result[0].length; c++) {
        result[r][c] = pixels[r*2][c*2];
    }
}

image.setPixelArray(result);
```

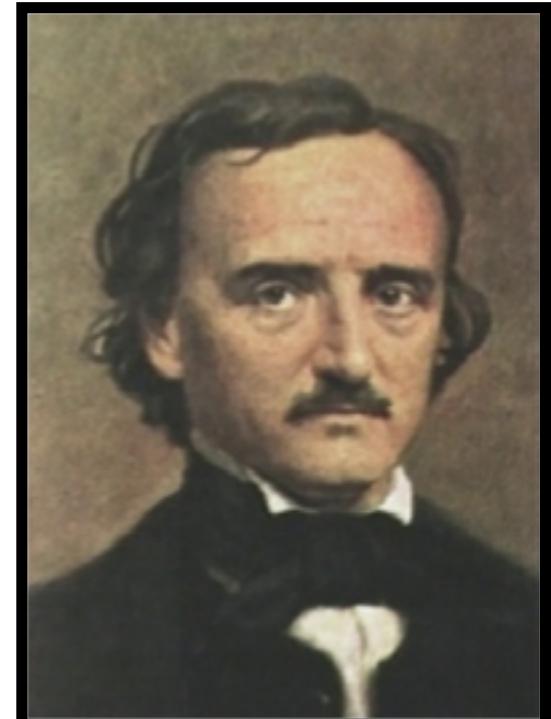
Plan for Today

- Recap: 2D Arrays and Images
- Practice: Shrink
- Practice: Cryptogram
- Practice: Tic-Tac-Toe

Cryptogram

A *cryptogram* is a puzzle in which a message is encoded by replacing each letter in the original text with some other letter. Your job in solving a cryptogram is figuring out this substitution pattern.

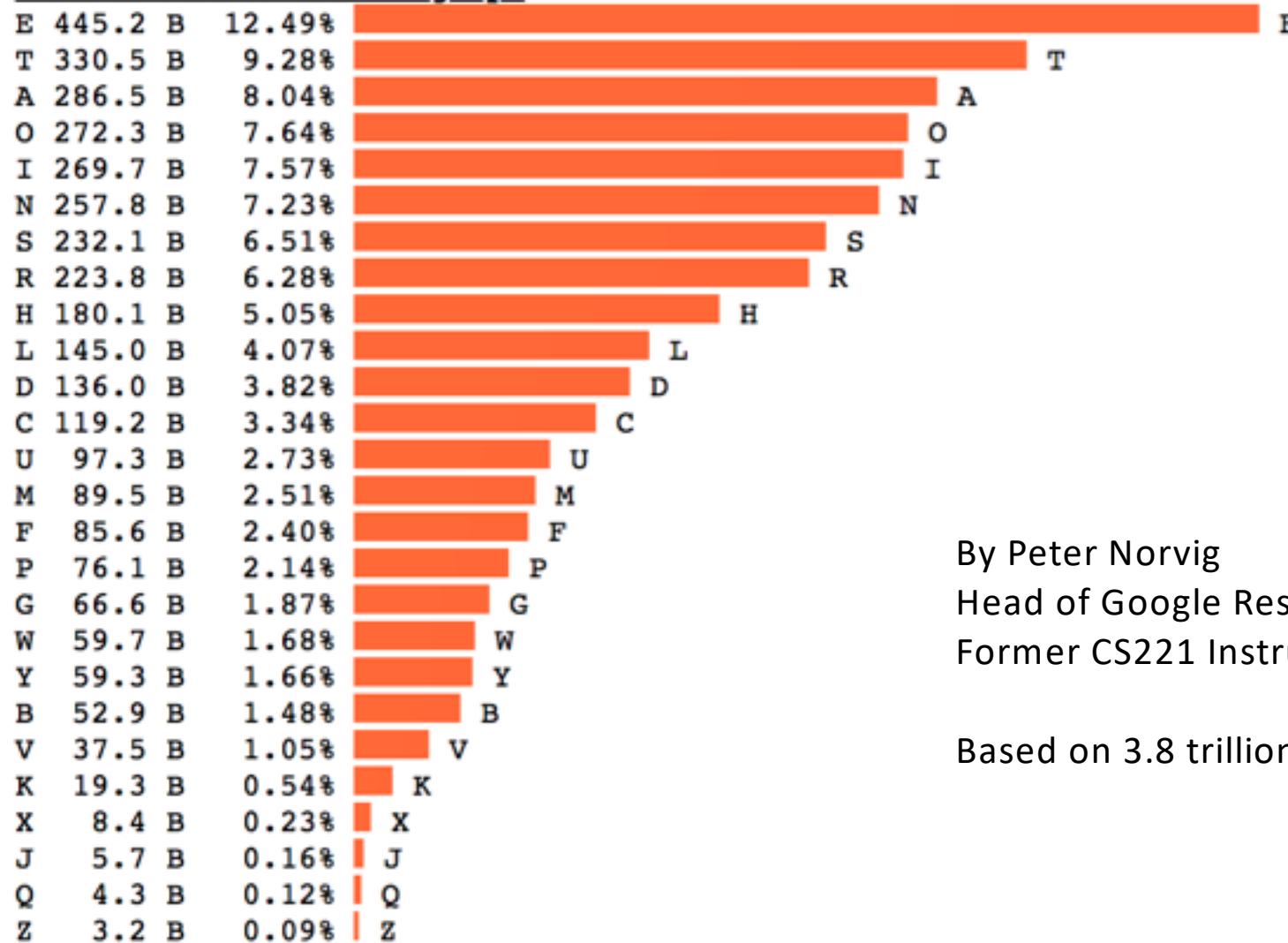
A common technique is assuming the most common letters in the coded message correspond to the most common letters in English.



Edgar Allan Poe (1809-1849)

Letter Frequency

LET COUNT PERCENT bar graph



By Peter Norvig
Head of Google Research
Former CS221 Instructor

Based on 3.8 trillion letters

Poe's Cryptographic Puzzle

5 3 ‡ ‡ † 3 0 5)) 6 * ; 4 8 2 6) 4 ‡ •) 4 ‡) ; 8 0 6 * ; 4 8 + 8 ¶
6 0)) 8 5 ; 1 ‡ (; : ‡ * 8 + 8 3 (8 8) 5 * † ; 4 6 (; 8 8 * 9 6 *
? ; 8) * † (; 4 8 5) ; 5 * † 2 : * † (; 4 9 5 6 * 2 (5 * - 4) 8 ¶
8 * ; 4 0 6 9 2 8 5) ;) 6 + 8) 4 ‡ ‡ ; 1 (‡ 9 ; 4 8 0 8 1 ; 8 : 8 ‡
1 ; 4 8 + 8 5 ; 4) 4 8 5 + 5 2 8 8 0 6 * 8 1 (‡ 9 ; 4 8 ; (8 8 ; 4 (
‡ ? 3 4 ; 4 8) 4 ‡ ; 1 6 1 ; : 1 8 8 ; ‡ ? ;

А С О О Д Г Н А \$ \$ Б Н Т И Е В Б \$ И О Р \$ И О \$ Т В О Б Н Т И Е В У
Б О \$ \$ Б А Т Ф О R T Y O N E D E G R E E S A N D T H I G H T E E N M I N
U T E S N O R T H E A S T A N D B Y N O R T H I M A B N V R A N C I S E V
B N T A D Б И В В А С T S Б Р Е С И Ф О Т Г О М Т И В О В Г Т В У Е
Г Т И Е Ф Е А Т И \$ И В А Д А В В Е В О Б Н Е F R O M T I E T R E E T H R
О У Г И Т И Е \$ И О Т Г Б Г Т Y F E E T F O T

8	33
;	26
4	19
‡	16
)	16
*	13
5	12
6	11
(10
†	8
1	8
0	6
9	5
2	5
:	4
3	4
?	3
¶	2
-	1
•	1

Poe's Cryptographic Puzzle

5 3 ‡ ‡ † 3 0 5)) 6 * ; 4 8 2 6) 4 ‡ •) 4 ‡) ; 8 0 6 * ; 4 8 † 8 ¶
6 0)) 8 5 ; 1 ‡ (; : ‡ * 8 † 8 3 (8 8) 5 * † ; 4 6 (; 8 8 * 9 6 *
? ; 8) * † (; 4 8 5) ; 5 * † 2 : * † (; 4 9 5 6 * 2 (5 * - 4) 8 ¶
8 * ; 4 0 6 9 2 8 5) ;) 6 † 8) 4 ‡ ‡ ; 1 (‡ 9 ; 4 8 0 8 1 ; 8 : 8 ‡
1 ; 4 8 † 8 5 ; 4) 4 8 5 † 5 2 8 8 0 6 * 8 1 (‡ 9 ; 4 8 ; (8 8 ; 4 (
‡ ? 3 4 ; 4 8) 4 ‡ ; 1 6 1 ; : 1 8 8 ; ‡ ? ;

AGOOGGLASSINTHEBISHOPSHOSTELINTHEDEV
ILSSEATFORTYONEDEGREESANDTHIRTEENMIN
UTESNORTHEASTANDBYNORTHMAINBRANCHSEV
ENTHLIMBEASTSIDESHOOTFROMTHELEFTEYE
FTHEDEATHSHEADABEELINEFROMTHETREETHR
OUGHTHESHOTFIFTYFEETOUT

8	33
;	26
4	19
‡	16
)	16
*	13
5	12
6	11
(10
†	8
1	8
0	6
9	5
2	5
:	4
3	4
?	3
¶	2
-	1
•	1

Idea: Array of Counters

- For problems like this, where we want to keep count of many things, a *frequency table* (or *tally array*) can be a clever solution.
 - *Idea:* The element at index i will store a counter for the character value ‘A’ + i .
 - example: count of letter frequency for “FIDDLE”

<i>letter</i>	...	D	E	F	G	H	I	J	J	L	...
<i>index</i>	...	3	4	5	6	7	8	9	10	11	...
<i>value</i>		2	1	1	0	0	1	0	0	1	

Plan for Today

- Recap: 2D Arrays and Images
- Practice: Shrink
- Practice: Cryptogram
- Practice: Tic-Tac-Toe

Tic-Tac-Toe

Let's use 2D arrays to create a ConsoleProgram version of Tic-Tac-Toe.



Recap

- Recap: 2D Arrays and Images
- Practice: Shrink
- Practice: Cryptogram
- Practice: Tic-Tac-Toe

Next Time: More data structures