The following program was designed to count and print the number of duplicates in the myArray String array. Unfortunately, it doesn't work properly. When I test it with the given data, it tells me that I have 11 duplicates, but I know that there are only two. Fix the program so that it works correctly.

```
// Count and print the number of duplicate strings in myArray

String [] myArray = {"A", "B", "C", "D", "A", "F", "C"};

void setup() {
  int count = 0;

  for (int i=0; i<myArray.length; i++) {
    for (int j=0; j<myArray.length; j++) {
      if (myArray[i].equals( myArray[j] )) {
        count++;
      }
    }
  }

  println("There are " + count + " duplicates.");
}
```

Exam 2 Review
Recursion, Transformations, Image Processing

Factorial

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$4! = 4 \times 3 \times 2 \times 1$

$5! = 5 \times 4!$

$N! = N \times (N-1)!$

Factorial can be defined in terms of itself

$4! = 4 \times 3 \times 2 \times 1 = 24$

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$5! = 5 \times 4!$

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1$

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		

Last In First Out (LIFO) Stack of Plates

Call Stack

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3		
factorial(2)	2		
factorial(1)	1		
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3		
factorial(2)	2		
factorial(1)	1	1 ⟷	1
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3		
factorial(2)	2	1 ⟷	2
factorial(1)	1	1	1
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3	2 ⟷	6
factorial(2)	2	1	2
factorial(1)	1	1	1
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10	362880	3628800
factorial(9)	9	40320	362880
factorial(8)	8	5040	40320
factorial(7)	7	720	5040
factorial(6)	6	120	720
factorial(5)	5	24	120
factorial(4)	4	6	24
factorial(3)	3	2	6
factorial(2)	2	1	2
factorial(1)	1	1	1
factorial(0)	0	--	1

```
String[] parts = new String[] {"a", "b", "c", "d"};

void setup() {}
void draw() {}

void mousePressed() {
  String joined = reverseJoin(3);
  println( joined );
}

String reverseJoin( int i ) {
  if (i == 0) {
    return parts[0];
  }
  else {
    String rjim1 = reverseJoin(i-1)
    return parts[i] + rjim1;
  }
}
```

Call	i	parts[i]	rjim1	returns
reverseJoin(3)	3	"d"	"cba"	"dcba"
reverseJoin(2)	2	"c"	"ba"	"cba"
reverseJoin(1)	1	"b"	"a"	"ba"
reverseJoin(0)	0	"a"	--	"a"

Three ways to transform the coordinate system:

1. **Translate**
   - Move axes left, right, up, down …
2. **Scale**
   - Magnify, zoom in, zoom out …
3. **Rotate**
   - Tilt clockwise, tilt counter-clockwise …

Scale

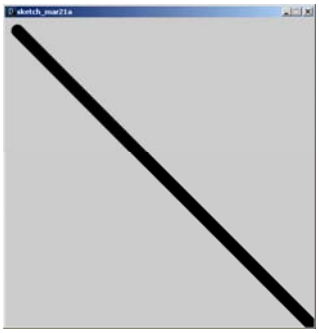
- All coordinates are multiplied by an x-scale-factor and a y-scale-factor.
- The size of everything is magnified about the origin (0,0)
- Stroke thickness is also scaled.

```
scale( factor );
scale( x-factor, y-factor );
```

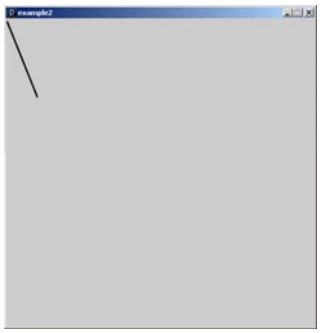
```
void setup() {
  size(500, 500);
  smooth();
  noLoop();

  line(1, 1, 25, 25);
}
```

```
void setup() {
  size(500, 500);
  smooth();
  noLoop();

  scale(2,2);
  line(1, 1, 25, 25);
}
```

```
void setup() {
  size(500, 500);
  smooth();
  noLoop();

  scale(20,20);
  line(1, 1, 25, 25);
}
```



```
void setup() {
  size(500, 500);
  smooth();
  noLoop();

  scale(2,5);
  line(1, 1, 25, 25);
}
```

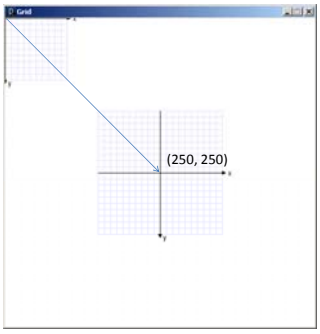


Translate

– The origin of the coordinate system (0,0) is shifted by the given amount in the x and y directions.

```
translate( x-shift, y-shift);
```

```
void draw() {
  grid();
  translate(250,250);
  grid();
}
```



Transformations can be combined

– Combine Scale and Translate to create a coordinate system with the y-axis that increases in the upward direction

– Axes can be flipped using negative scale factors

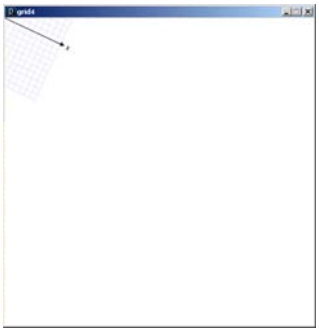
– Order in which transforms are applied matters!

Rotate

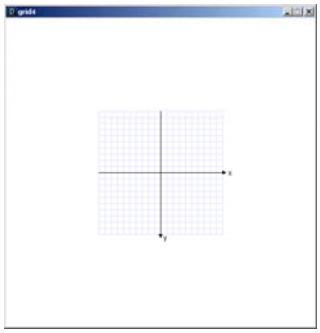
– The coordinate system is rotated around the origin by the given angle (in radians).

```
rotate( radians );
```

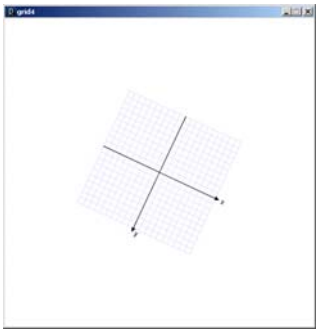
```
void draw() {
  rotate( 25.0 * (PI/180.0) );
  grid();
}
```



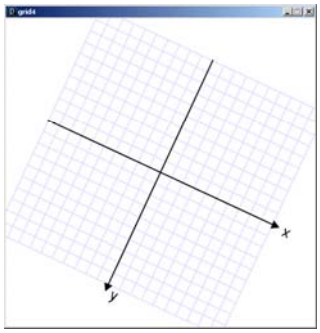
```
void draw() {
  translate(250.0, 250.0);
  //rotate( 25.0 * (PI/180.0) );
  //scale( 2 );
  grid();
}
```



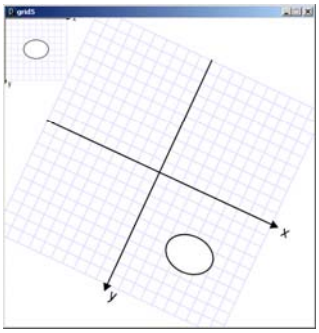
```
void draw() {
  translate(250.0, 250.0);
  rotate( 25.0 * (PI/180.0) );
  //scale( 2 );
  grid();
}
```



```
void draw() {
  translate(250.0, 250.0);
  rotate( 25.0 * (PI/180.0) );
  scale( 2 );
  grid();
}
```




```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  translate(250.0, 250.0);
  rotate( 25.0 * (PI/180.0) );
  scale(2);
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```



grid5.pde

Last In First Out (LIFO) Stack of Plates

Transformation Matrix Stack

Transformation Matrix Stack

- All transformations are matrix multiplications
- Transformation matrices can be managed using the **Matrix Stack.** (Recall, a stack is LIFO)
- All drawing commands are always multiplied by the current transformation matrix, which is the identity matrix by default
- A transformation matrix can be temporarily pushed on to the Matrix Stack, and popped off when drawing is done.

pushMatrix()

- Pushes a copy of the current transformation matrix onto the Matrix Stack

popMatrix()

- Pops the last pushed transformation matrix off the Matrix Stack and replaces the current matrix

resetMatrix()

- Replaces the current transformation matrix with the identity matrix

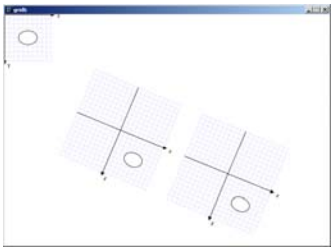
applyMatrix()

- Multiplies the current transformation matrix with a given custom matrix.

printMatrix()

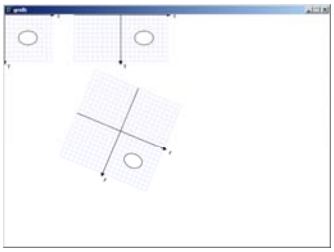
- Prints the current transformation matrix in effect.

```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  translate(250, 250);
  rotate(PI/8.0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  translate(250, 0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```



relXform.pde

```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  pushMatrix();
  translate(250, 250);
  rotate(PI/8.0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
  popMatrix();

  translate(250, 0);

  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```

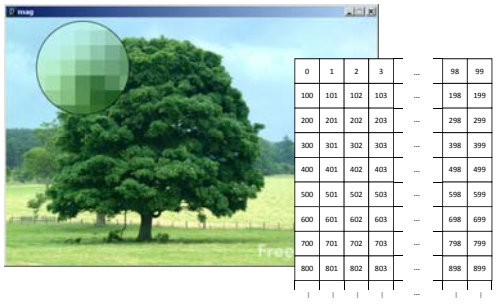


relXform.pde

Some things to remember:

1. Transformations are cumulative.
2. All transformations are cancelled each time draw() exits.
   - They must be reset each time at the beginning of draw() before any drawing.
3. Rotation angles are measured in radians
   - radians = (PI/180.0) * degrees
   - radians()
4. Order matters

An image is an array of colors

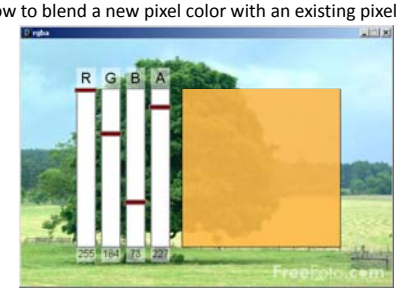


Pixel : Picture Element

mag.pde

Color

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



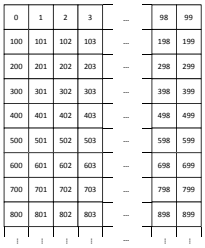
rgba.pde

Accessing the pixels of a sketch

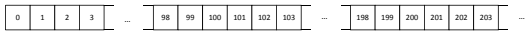
- loadPixels()
  - Loads the color data out of the sketch window into a 1D array of colors named pixels[]
  - The pixels[] array can be modified
- updatePixels()
  - Copies the color data from the pixels[] array back to the sketch window

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




```
// whiteNoise

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

void draw() {
  float b;

  // Load colors into 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();
}
```

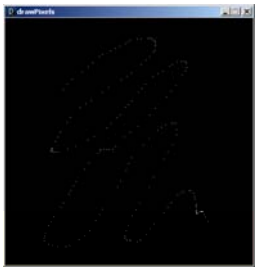
See also colorNoise.pde

Accessing Pixels as a 2D Array

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

```
Index = y*width + x
```

```
// drawPixels
void setup() {
  size(400, 400);
  background(0);
}

void draw() {
  if (mousePressed == true) {
    // Load colors into the
    // pixels array
    loadPixels();

    // Compute pixel location
    // from mouse coordinates
    int i = mouseY*width + mouseX;

    // Set pixel color
    pixels[i] = color(255);

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



drawPixels.pde

ArrayList

- Constructors

```
ArrayList lst1 = new ArrayList();
ArrayList lst2 = new ArrayList(int initialSize);
```

- Methods

```
size()                          // Returns the num of items held.
add(Object o)                   // Appends o to end.
add(int idx, Object o)          // Inserts o at pos idx.
remove(int idx)                 // Removes item at pos idx.
get(int idx)                    // Gets items at idx. No removal.
set(int idx, Object o)          // Replaces item at idx with o.
clear()                         // Removes all items.
isEmpty()                       // true if empty.
toArray()                       // returns an array that contains
                                // the contents of the list
```

- When removing from an ArrayList, loop back to front

What does the following program print? Justify your answer if you're not certain.

```
void setup() {
  int n = op(5,3);
  println( n );
}

int op (int val, int divisor) {

  if (val < divisor) {
    return val;
  } else {
    int v = op(val - divisor, divisor);
    return v;
  }
}
```

Which built-in mathematical operator does the op() function simulate?

Add the necessary transformations to `draw()` to render the rectangle at the center of the sketch, twice its size, and rotated by 45 degrees (PI/4 radians).

```
void setup() {
  size(400, 400);
  rectMode(CENTER);
}

void draw() {
 // Add transformations here



  rect( 0, 0, 50, 50);
}
```

Add the necessary code within the nested for-loops to color all pixels on the diagonal white, and the others black.

```
void setup() {
  size(100,100);

  loadPixels();

  for (int y=0; y<height; y++) {
    for (int x=0; x<width; x++) {

      // Add code here


    }  // Closing brace for the x-loop
  } // Closing brace for the y-loop

  updatePixels();
}
```