Programming with Processing!

Oregon State University
Mike Bailey
mjba@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

Programming Through the Ages

Oregon State University
Mike Bailey
mjba@cs.oregonstate.edu
Textiles

Jacquard Loom, circa 1804

Music

Albany Carousel and Museum
Music

Albany Carousel and Museum

Computer Punch Cards
Running Processing

In your favorite web browser, go to: https://editor.p5js.org/
Here’s what you will see:
Introduction to Writing Processing Programs

With *Processing*, I have bad news, and I have good news.

The bad news is that you have to write a program. This will involve some learning.

The good news is that you *get* to write a program. You will end up being ever-so-more knowledgeable than you started out, and, once you get the hang of this, there is nothing you won’t be able to do with it!
Introduction to Writing Processing Programs

With Processing, I have bad news, and I have good news.

The bad news is that you have to write a program. This will involve some learning.

The good news is that you get to write a program. You will end up being ever-so-more knowledgeable than you started out, and, once you get the hang of this, there is nothing you won’t be able to do with it!
Coordinate Systems for Processing Programs

Colors for Processing Programs

Colors are formed with combinations of red, green, and blue.

The smallest number you can use is 0
The largest number you can use is 255

<table>
<thead>
<tr>
<th>Color</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>White</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Red</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Orange</td>
<td>255</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>Yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>Cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
</tbody>
</table>
Colors for Processing Programs

This is referred to as “Additive Color”

Red + Green = Cyan
Red + Blue = Magenta
Red + Green = Yellow
Red + Green + Blue = White

Writing a Processing Program – Try This!

You must add code to the `setup()` function. Processing calls this `once` when your program starts.

```java
function setup() {
  createCanvas( 800, 600 );
  colorMode( RGB );
  background( 200, 200, 255 );
}

function draw() {
  stroke( 0, 0, 0 );
  fill( 255, 50, 50 );
  rect( 100, 200, 150, 50 );
}
```

You must add code to the `draw()` function. Processing calls this every time it wants to re-draw the scene.
Running Your Processing Programs

Click here to run your program

```javascript
function setup()
{
  createCanvas( 800, 600 );
  colorMode( RGB );
  background( 200, 200, 255 );
}

function draw()
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50 );
  rect( 100, 200, 150, 50 );
}
```

Enjoying the Output of Your Processing Program
### Other Functions to use when Writing Processing Programs

<table>
<thead>
<tr>
<th>Setup</th>
<th>Set the background to r, g, b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>Set the current color to (r, g, b) in RGB space</td>
</tr>
<tr>
<td>ColorMode</td>
<td>Set the color specification mode to RGB or HSB</td>
</tr>
<tr>
<td>Setup</td>
<td>The function that gets called over and over to draw your scene</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw an ellipse in CENTER mode</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw an ellipse in RADIUS mode</td>
</tr>
<tr>
<td>Shapes</td>
<td>CORNERS, CENTER, RADIUS</td>
</tr>
<tr>
<td>Drawing</td>
<td>Fill using the color c</td>
</tr>
<tr>
<td>Variables</td>
<td>Screen height in pixels</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw a line</td>
</tr>
<tr>
<td>Setup</td>
<td>Starts automatic calling of draw()</td>
</tr>
<tr>
<td>Math</td>
<td>Linearly map the input variable from the range [lowin,highin] to [lowout,highout]</td>
</tr>
<tr>
<td>Shapes</td>
<td>Put a dot at (x,y)</td>
</tr>
<tr>
<td>Printing</td>
<td>Print the string into the console, adding a return</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw a quadrilateral</td>
</tr>
<tr>
<td>Randomness</td>
<td>Return a random number between low and high</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw a rectangle in CENTER mode</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw a rectangle in CORNERS mode</td>
</tr>
<tr>
<td>Shapes</td>
<td>CORNER, CENTER, RADIUS</td>
</tr>
<tr>
<td>setup</td>
<td>The function that gets called when your program starts</td>
</tr>
<tr>
<td>Printing</td>
<td>Print a string into the status area</td>
</tr>
<tr>
<td>Drawing</td>
<td>Outline using the color c</td>
</tr>
<tr>
<td>Drawing</td>
<td>Thickness of the outline</td>
</tr>
<tr>
<td>Shapes</td>
<td>Draw a triangle</td>
</tr>
<tr>
<td>Variables</td>
<td>Screen width in pixels</td>
</tr>
</tbody>
</table>

---

### Variables and For-loops

**Oregon State University**

Mike Bailey

mjb@cs.oregonstate.edu

---

**Oregon State University**

Computer Graphics
Variables are the process of replacing concrete values with symbols in order to generalize a computation to work in more than one situation.

```c
void draw(  )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    int x = 100;
    int y = 200;
    rect( x, y, 150, 50 );
}
```

"int" stands for "integer", a whole number with no decimal digits, e.g., 3.

"float" designates a number that can have decimal digits, e.g., 3.14.

We can use variables to capture relationships.

```c
void draw(  )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    int x = 100;
    int y = 2 * x;  // Arithmetically, y = 2 * x;
    rect( x, y, 150, 50 );
}
```

Arithmetic operations in programming are:
- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `( )` Grouping
Drawing One Rectangle is Pretty Straightforward

```plaintext
rect( 100, 200, 150, 50 );
```

But, This Gets Awfully Boring if You Want to Draw 100 Rectangles!

```plaintext
rect( 100, 200, 150, 50 );
rect( 110, 210, 150, 50 );
rect( 120, 220, 150, 50 );
```
Repeating a code pattern is a recurring theme in programming.

This line is called a “for-loop”. It is very handy for repeating patterns of code. It expresses those patterns as relationships.

The for-loop executes the commands in the curly braces a bunch of times. Its use looks like this:

```java
for( int x = 0 ; x < 400 ; x = x + 10 )
    do this equation once at the start
    keep looping as long as this equation is true
    do this at the end of one loop, but before the start of the next one
```

For-loops to the Rescue!
More Sophisticated Relationships:

The `map()` function

This function takes an input value, the range of values it lives between, and the range of output values. It returns the output value that corresponds to the input value.

So, for example, if we wanted to turn an x value into a red color, we might say:

```c
int red = int( map( x, 0, width - 1, 0, 255 ) );
```

The `map()` function

```c
void draw( )
{
  stroke( 0, 0, 0 );
  fill( 255, 50, 50 );
  for( int x = 0 ; x < 400 ; x = x + 10 )
  {
    int y = x;
    int red = int( map( x, 0, 399, 0, 255 ) );
    int green = int( map( y, 0, 399, 0, 255 ) );
    //println( "x = " + x + " red = " + red );
    fill( red, green, 50 );
    rect( x, y, 150, 50 );
  }
}
More Sophisticated Relationships:
The \textit{map()} function

```c
void
draw()
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    for( int x = 0 ; x < 400 ; x = x + 10 )
    {
        int y = x;
        int red = int( map( x, 0, 399, 0, 255 ) );
        int green = int( map( y, 0, 399, 0, 255 ) );
        green = 3 * green / 4;
        //println( "x = " + x + " red = " + red );
        fill( red, green, 50 );
        rect( x, y, 150, 50 );
    }
}
```

The \textit{map()} function can also do blending

```c
void
draw()
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    for( int x = 0 ; x < 400 ; x = x + 10 )
    {
        int y = x;
        int red = int( map( x, 0, 399, 0, 255 ) );
        int green = int( map( y, 0, 399, 255, 0 ) );
        //println( "x = " + x + " red = " + red );
        fill( red, green, 50 );
        rect( x, y, 150, 50 );
    }
}
```
If-statements

Your Code Often Wants to Test Something and Make a Decision Based On It

```cpp
if( condition )
{
    do this;
    do that;
}
```

These Operators Make up the Possible Conditions:

- `<`  Is less than
- `<=` Is less than or equal to
- `>`  Is greater than
- `>=` Is greater than or equal to
- `==` Is equal to
- `!=` Is not equal to
- `&&` And
- `||` Or
Example #1

```plaintext
int x = 100;
fill( 0, 255, 0 );
for( int y = 0; y < 800; y = y + 100 )
{
    if( y >= 200 )
    {
        fill( 255, 0, 0 );
    }
    rect( x, y, 200, 100 );
}
```

Example #2

```plaintext
fill( 0, 255, 0 );
for( int y = 0; y < 800; y = y + 100 )
{
    int x = y / 5;
    if( x < 100 && y >= 200 )
    {
        fill( 255, 0, 0 );
    }
    rect( x, y, 200, 100 );
}
```
Your Code Often Wants to Test Something and Make a Decision Based On It or the Opposite Condition

```
if( condition )
{
    do this;
}
else
{
    do that;
}
```

Your Code Often Wants to Test Something and Make a Decision Based On It or on Other Conditions

```
if( condition )
{
    do this;
}
else if( another_condition )
{
    do it;
}
else
{
    do that;
}
```
Your Code Often Wants to Test Something and Make a Decision Based On It or Lots of Alternatives

```cpp
if ( key == 'r' )
{
    fill( 255, 50, 50 );
} else if( key == 'g' )
{
    fill( 50, 255, 50 );
} else if( key == 'b' )
{
    fill( 50, 50, 255 );
} else
{
    fill( 100, 100, 100 );
}
```

Your Code Often Wants to Test Something and Make a Decision Based On It or Lots of Alternatives -- a Better Way

```cpp
switch( key )
{
    case 'r':
        fill( 255, 50, 50 );
        break;

    case 'g':
        fill( 50, 255, 50 );
        break;

    case 'b':
        fill( 50, 50, 255 );
        break;

    default:
        fill( 100, 100, 100 );
}
```
Some of Processing's Variables Already Have the Condition Built-In

```javascript
function draw( )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    if( mousesPressed )
    {
        rect( mouseX, mouseY, 50, 20 );
    }
}
```

- `mousesPressed` is a built-in variable that is always telling you if a mouse button is currently pressed.
- `mouseX` and `mouseY` are built-in variables that are always telling you where the mouse cursor is.

Reacting to the Mouse and Keyboard

[Images of Processing sketches]
The `mouselsPressed`, `mouseX`, and `mouseY` Variables

```javascript
function draw( ) {
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    if( mouselsPressed )
    {
        ellipse( mouseX, mouseY, 50, 50 );
    }
}
```

- `mouselsPressed` is a built-in variable that is always telling you if a mouse button is currently pressed.
- `mouseX` and `mouseY` are built-in variables that are always telling you where the mouse cursor is.
The `isKeyPressed` and `key` Variables

```java
function draw(  )
{
    if( isKeyPressed )
    {
        switch( key )
        {
            case 'r':
                fill( 255, 50, 50 );
                break;
            case 'g':
                fill( 50, 255, 50 );
                break;
            case 'b':
                fill( 50, 50, 255 );
                break;
        }
    }
    if( mouseIsPressed )
    {
        ellipse( mouseX, mouseY , 50, 50 );
    }
}
```

- The `stroke()` and `fill()` calls have been moved to `setup()`.
- `isKeyPressed` is a built-in variable that is always telling you if a keyboard key has been pressed.
- `key` is a built-in variable that tells you what key has been hit.
- The `switch/case` statements are Processing’s way of checking many values without having a whole slew of `if`-statements.
What if you want to read the Special Keys?

```c
if( isKeyPressed )
{
    if( key == CODED )
    {
        switch( keyCode )
        {
            case UP: // up-arrow
                ...
            break;
        }
    }
}
```

Values for `keyCode` can be:
- UP
- DOWN
- LEFT
- RIGHT
- ESC
- DELETE
- BACKSPACE
- TAB
- ENTER
- RETURN

---

Drawing Arbitrary Polygons

Mike Bailey
mjb@cs.oregonstate.edu

Oregon State
University
Computer Graphics

arbitrary_polygons.pptx

Oregon State
University
Computer Graphics

arbitrary_polygons.pptx
Rectangles are Good, but Arbitrary Polygons are nice too

Easy – just list the coordinates:

```java
beginShape();
   vertex( x0, y0 );
   vertex( x1, y1 );
   vertex( x2, y2 );
   ...
endShape();
```

```java
void draw( )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    beginShape( );
    vertex( 100, 100 );
    vertex( 100, 400 );
    vertex( 200, 400 );
    vertex( 300, 300 );
    vertex( 400, 50 );
    endShape( );
}
```
Drawing Text

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

text.pptx

Setting the size and drawing the text

```java
void setup()
{
    size( 400, 400 );
    background( 50, 255, 50 );
    textSize( 18 );
}

void draw()
{
    fill( 0, 0, 0 );
    text( 'ABC', 50, 50 );

    fill( 0, 0, 255 );
    text( 'DEF', 50, 80 );
}
```
Transformations

It is Often Nice to Transform Entire Objects at Once

```c
void setup()
{
  size(800, 800);
  stroke(0, 0, 255);
  fill(0, 255, 255);
}

void draw()
{
  rect(0, 0, 100, 50);
}
```
Translation

```cpp
void draw()
{
    translate( 100, 200 );
    rect( 0, 0, 100, 50 );
}
```

Rotations and Scaling Happen Around the Origin
In math, science, and computer programming, angles are not given in degrees, they are given in radians.

1 radian = 0.01745 degrees
1 radian = π/180. degrees

But, don’t worry about this.

Processing gives you a function, `radians()`, to automatically convert degrees into radians.

Use it!

```
void draw()
{
    rotate( radians(45.));
    rect( 0, 0, 100, 50);
}
```

Scaling

```
void draw()
{
    scale( 5., 1.);
    rect( 0, 0, 100, 50);
}
```
Shearing

There is also a \texttt{shearY} transformation function

```
void draw( )
{
  shearY( radians(45.));
  rect( 0, 0, 100, 50);
}
```

Transformations Accumulate!

```
function draw( )
{
  rotate( radians(10.) );
  rotate( radians(10.) );
  ...
}
```

is the same as:

```
function draw( )
{
  rotate( radians(20.) );
  ...
}
```
Transformation Order Matters!

```c
void draw()
{
  2. translate(100, 200);
  1. rotate( radians(20) );
  rect( 0, 0, 100, 50 );
}
```

You Can Save and Un-do Transformations

```c
void draw()
{
  2. translate(200, 300);
  pushMatrix();
  1. shearX( radians(45) );
      rect( 0, 0, 200, 100 );
  popMatrix();
      fill( 255, 0, 0 );
  1. rotate( radians(-45) );
      rect( 0, 0, 200, 100 );
}
```
Transformations and for-loops

```cpp
void draw() {
    translate(200, 200);
    for(int degrees = 0; degrees <= 360; degrees = degrees + 36) {
        pushMatrix();
        rotate(radians(degrees));
        rect(0, 0, 100, 30);
        popMatrix();
    }
}
```

Transformations and for-loops

```cpp
void draw() {
    translate(200, 200);
    for(int degrees = 0; degrees <= 360; degrees = degrees + 36) {
        pushMatrix();
        rotate(radians(degrees));
        rect(0, -15, 100, 30);
        popMatrix();
    }
}
```
What's the Difference?

```cpp
void draw() {
    translate(208, 208);
    for(int degrees = 0; degrees <= 360;
        pushMatrix();
        rotate( radians(degrees) );
        rect(0, 8, 100, 30);
        popMatrix();
    }
}
```

Transformations and for-loops

```cpp
void draw() {
    translate(288, 208);
    for(int degrees = 0; degrees <= 360;
        pushMatrix();
        rotate( radians(degrees) );
        rect(6, -15, 100, 30);
        popMatrix();
    }
}
```
Rotating While Changing Color and Size

```javascript
void draw() {
  translate(200, 200);
  for( int degrees = 0; degrees < 360; degrees = degrees + 10 ) {
    pushMatrix();
    // map color from cyan to green
    int blue = int( map( degrees, 0, 360, 0, 255, 0 ) );
    TINT(0, 255, blue);
    // transform by rotating:
    rotateY( radians( degrees ) );
    // change rectangle size:
    int xsize = int( map( degrees, 0, 360, 100, 10 ) );
    int ysize = int( map( degrees, 0, 360, 30, 5 ) );
    // draw rectangle away from the origin:
    rect(100, -15, xsize, ysize);
    popMatrix();
  }
}
```

And, there are even 3D Transformations

```javascript
translate( x, y, z );
scale( x, y, z );
rotateX( radians );
rotateY( radians );
rotateZ( radians );
```

But, we will get to those later …
Let's Start with a Favorite Image of Yours

It can be in .jpg, .bmp, or .png format

Each pixel contains a red-green-blue, each in the range 0-255

The image has an aspect ratio, which is the ratio of the number of Y pixels : the number of X pixels (this image’s aspect ratio is 1:1)
PImage MyImage;

function setup()
{
    createCanvas(800, 800);
    //MyImage = loadImage("C:/MJB/Processing/ImageSketchBook/zelda.jpg");
    MyImage = loadImage("zelda.jpg");
}

function draw()
{
    image(MyImage, 0, 0, 800, 800);
}

“PImage” is a variable type, just like int and float, but for images.

Declaring a variable up here, ahead of everything else, makes it so that it can be seen from anywhere in the program.

This loads the image from the file into the variable called MyImage.

This draws the image from the variable called MyImage.

How many pixels to use to draw the image.

What X-Y to draw its upper-left corner at.
What Happens if You Use Less Pixels than the Window Has?

```c
void draw()
{
    image(MyImage, 50, 50, 400, 400);
}
```

What Happens if You Use a Different Aspect Ratio?

```c
void draw()
{
    image(MyImage, 50, 50, 600, 300);
}
```
Translating an Image

```c
void draw()
{
    for( int i = 0; i < 6; i++ )
    {
        pushMatrix();
        translate(100, i*100);
        image(MyImage, 0, 0, 200, 200);
        popMatrix();
    }
}
```

Rotating an Image

```c
void draw()
{
    for( int i = 0; i < 6; i++ )
    {
        pushMatrix();
        translate(100, 200);
        rotate(radians(1*60));
        image(MyImage, 0, 0, 200, 200);
        popMatrix();
    }
}
```
Overwriting an Image

```java
MyImage = loadImage("lesson.jpg");
MyImage.width = 32;
void draw() {
    Image(MyImage, 0, 0, 800, 800);
    for (int x = 0; x < 1200; x = x + 1) {
        for (int y = 0; y < MyImage.height; y = y + 1) {
            color diff = select (255, 0, 0);
            MyImage.set (x, y, diff);
        }
    }
}
```

Retrieving Image Colors

```java
void draw() {
    Image(MyImage, 8, 6, 600, 800);
    for (int x = 0; x < MyImage.width; x = x + 1) {
        for (int y = 6; y < x; y = y + 1) {
            color get = MyImage.get (x, y);
            MyImage.set (MyImage.width - x, y, get);
        }
    }
}
```
First, We Need to Understand Something about Angles

If a circle has a radius of 1.0, then we can march around it by simply changing the angle that we call $\theta$. 
First, We Need to Understand Something about Angles

One of the things we notice is that each angle $\theta$ has a unique $X$ and $Y$ that goes with it.
These are different for each $\theta$.

Fortunately, centuries ago, people developed tables of those $X$ and $Y$ values as functions of $\theta$.
They called the $X$ values cosines and the $Y$ values sines. These are abbreviated $\cos$ and $\sin$.

\[
\cos \theta = X \\
\sin \theta = Y
\]
How People used to Lookup Sines and Cosines – Fortunately We Now Have Calculators and Computers

First, We Need to Understand Something about Angles

If we were to double the radius of the circle, all of the X’s and Y’s would also double.

So, really the cos and sin are ratios of X and Y to the circle Radius

\[
\cos \theta = \frac{X}{R} \\
\sin \theta = \frac{Y}{R}
\]
First, We Need to Understand Something about Angles

So, if we know the circle Radius, and we march through a bunch of $\theta$ angles, we can determine all of the X's and Y's that we need to draw a circle.

\[ \cos \theta = \frac{X}{R} \quad X = R \cdot \cos \theta \]
\[ \sin \theta = \frac{Y}{R} \quad Y = R \cdot \sin \theta \]

function Circle( int xc, int yc, int r, int numsegs )
{
    float dang = (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );
    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float x = xc + r * cos(ang);
        float y = yc + r * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }
    endShape( );
}

numsegs is the number of line segments making up the circumference of the circle.
numsegs=20 gives a nice circle.
5 gives a pentagon.
8 gives an octagon.
4 gives you a square. Etc.

Why 2.*PI ?
Why 2.*PI ?

```c
float dang = (2.*PI) / float( numsegs );
```

We commonly measure angles in degrees, but science and computers like to measure them in something else called radians.

There are 360° in a complete circle.
There are 2π radians in a complete circle.

The built-in cos( ) and sin( ) functions expect angles given in radians.

Processing has built-in functions to convert between the two:

```c
float rad = radians( deg );
float deg = degrees( rad );
```

Circle, Pentagon, Octagon!

```c
void draw( )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    Circle( 200, 200, 100, 20 );
    fill( 50, 255, 50 );
    Circle( 300, 300, 100, 5 );
    fill( 50, 50, 255 );
    Circle( 400, 400, 100, 8 );
}
```
If We Move the Mouse, We Could Get:

Or, even:
function Ellipse( int xc, int yc, int rx, int ry, int numsegs )
{
    float dang = (2.*PI) / float(numsegs);
    float ang = 0.;
    beginShape( );
    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float x = xc + rx * cos(ang);
        float y = yc + ry * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }
    endShape( );
}

And, there is no reason the X and Y radii need to be the same…

There is actually no reason the X and Y radii need to be the same ...

void draw( )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    Ellipse( 200, 200, 150, 75, 20 );

    fill( 50, 255, 50 );
    Ellipse( 300, 300, 150, 75, 5 );

    fill( 50, 50, 255 );
    Ellipse( 400, 400, 150, 75, 8 );
}
There is also no reason we can't gradually change the radius …

```c
function Spiral( int xc, int yc, int r0, int r1, int numsegs, int numturns )
{
    float dang = numturns * (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );
    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float newrad = map( i,  0, numsegs,  r0, r1 );
        float x = xc + newrad * cos(ang);
        float y = yc + newrad * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }
    endShape( );
}
```

There is also no reason we can't gradually change the radius …

```c
void
draw( )
{
    stroke( 50, 50, 255 );
    strokeWeight( 5 );
    noFill( );
    Spiral( 300, 300, 20, 200, 1000, 10 );
}
```
We Can Also Use This Same Idea to Arrange Things in a Circle

function
draw( )
{
    stroke( 0, 0, 0 );
    int numobjects = 10;
    float radius = 200.;
    int xc = 300;
    int yc = 300;
    int numsegs = 20;
    int r = 50;
    float dang = (2.*PI) / float( numobjects - 1 );
    float ang = 0.;
    for( int i = 0; i < numobjects; i = i + 1 )
    {
        float x = xc + radius * cos(ang);
        float y = yc + radius * sin(ang);
        int red   = int( map( i, 0, numobjects – 1, 0, 255 ) );
        int blue = int( map( i, 0, numobjects – 1, 255, 0 ) );
        fill( red, 0, blue );
        Circle( int(x), int(y), r, numsegs );
        ang = ang + dang;
    }
}
Three Dimensions

The 3D Processing Functions You Will Care About

rotateX( radians );
rotateY( radians );
rotateZ( radians );
scale( sx, sy, sz );
translate( tx, ty, tz );
box( sizex, sizey, sizez );
sphereDetail( slices, stacks );
sphere( radius );
beginShape( );
vertex( x, y, z );
endShape( );
Sample 3D Program – the Global Variables at the top of the program

```java
int LastMouseX;
int LastMouseY;
int Udetail = 20;
int Vdetail = 20;
float Yangle = 0.;
float Xrot = 0., Yrot = 0.;

boolean FillSphere = false;
boolean StillPressed = false;
boolean Animate = false;
```

Sample 3D Program – the `setup()` Function

```java
function setup() {
    createCanvas(800, 800, P3D);
    background(200, 200, 255);
    stroke(0, 0, 0);
    fill(255, 255, 0);
}
```

This third argument to `createCanvas()` tells Processing to allow it to do 3D
Sample 3D Program – the draw( ) Function, part I

```java
function
draw() {
    background(200, 200, 255);
    if(isKeyPressed)
    {
        if(!StillPressed) // same as saying "if(StillPressed == false)"
        {
            switch(key)
            {
                case 'a':
                    Animate = !Animate;
                    break;
                case 'f':
                    FillSphere = !FillSphere;
                    break;
                case 'l':
                    Udetail = Udetail - 1;
                    Vdetail = Vdetail - 1;
                    break;
                case 'm':
                    Udetail = Udetail + 1;
                    Vdetail = Vdetail + 1;
                    break;
            }
            StillPressed = true;
        }
        else
        {
            StillPressed = false;
        }
    }
}
```

Sample 3D Program – the draw( ) Function, part II

```java
if(mouseIsPressed) {
    int dx = mouseX - LastMouseX;
    int dy = mouseY - LastMouseY;
    Xrot = Xrot + dy;
    Yrot = Yrot + dx;
    LastMouseX = mouseX;
    LastMouseY = mouseY;
    translate(width/2, height/2);
    rotateY(radians(Yrot));
    rotateX(radians(Xrot));
    fill(255, 30, 30);
    pushMatrix();
    rotateX(radians(Yangle));
    translate(0, 300, 0);
    box(20, 20, 20);
    popMatrix();
}
if(FillSphere) {
    fill(255, 255, 0);
} else {
    noFill();
    pushMatrix();
    rotateY(radians(Yangle));
    sphereDetail(Udetail, Vdetail);
    sphere(200);
    if(Animate) {
        Yangle = Yangle + 1.0;
    }
    popMatrix();
}```
Randomness

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu
Start With Something We've Seen Before

```cpp
void setup() {
  size( 800, 600 );
  colorMode( RGB );
  background( 255, 255, 255 );
  stroke( 0, 0, 255 );
  strokeWeight( 4.0 );
  fill( 255, 50, 50 );
  noFill();
}

void draw() {
  beginShape();
  vertex( 188, 166 );
  vertex( 188, 406 );
  vertex( 288, 406 );
  vertex( 288, 300 );
  vertex( 488, 56 );
  endShape();
}
```

Pure Randomness is Pretty Jarring

```cpp
void draw() {
  background( 255, 255, 255 );
  beginShape();
  for( int x = 0; x < width; x += 5 )
  {
    int y = int( random( 0, height ) );
    vertex( x, y );
  }
  endShape();
}
A Better Approach – Add a Random Number to the Current Value

```java
void draw() {
    background(255, 255, 255);
    int y = height / 2;
    beginShape();
    for (int x = 0; x < width; x++)
    {
        int dy = int(random(-height/10, height/10));
        y = y + dy;
        vertex(x, y);
    }
    endShape();
}
```

Computer Graphics Noise

- The built-in `noise()` function is a smoothly-changing sequence of values.
- It returns values from 0 to 1.
- It is centered around 0.5, i.e., the midline.
- It can be spread out (made smoother) by making the argument smaller.
- It can be compressed (made more jagged) by making the argument larger.
- It is **Coherent** in that the noise value at one point is close to the noise value at the next point.
- Setting `noiseSeed()` makes it **Repeatable** in that the same input always gives the same output.
Noise Octaves Create More Detail

A Noise Octave is another noise wave with lower amplitude (height) and higher frequency (jagginess). We add octaves together to get a combination of smoothness and jagginess.

```
float NoiseFactor = 200.; // larger to make the noise gentler
int NoiseSeed = 22019;  // start the random number sequence
int MinOctaves = 1;
int MaxOctaves = 8;

function
setup( )
{
    createCanvas( 800, 600 );
    colorMode( RGB );
    noFill();
    noiseSeed( NoiseSeed );
}
```

The Setup
function draw( )
{
  background( 200, 200, 255 );
  stroke( 128, 0, 0 );
  strokeWeight( 1. );
  beginShape( );
  vertex( 0, height/2 );
  vertex( width, height/2 );
  endShape( );

  for( int octaves = MinOctaves; octaves <= MaxOctaves; octaves = octaves*2 )
  {
    noiseDetail( octaves );
    int green = int( map( octaves, MinOctaves, MaxOctaves, 0, 255 ) );
    beginShape( );
    for( int x = 0; x < width; x = x + 5 )
    {
      int y = ( height / 2 ) + int( ( height ) * ( noise( x / NoiseFactor ) - 0.5 ) );
      vertex( x, y );
    }
    endShape( );
  }
Using Noise to Affect Size

```javascript
float NoiseFactor = 200.;    // larger to make the noise gentler
int NoiseSeed = 22019;      // start the random number sequence

function setup( )
{
    createCanvas( 800, 800 );
    colorMode( RGB );
    background( 200, 200, 255 );
    fill( 255, 255, 0 );
    stroke( 0, 0, 0 );
    noiseSeed( NoiseSeed );
    noiseDetail( 4 );
}

In draw( ):  
{
    if ( mouseIsPressed )
    {
        float nx = noise( mouseX/NoiseFactor );
        float ny = noise( mouseY/NoiseFactor );
        ellipse( mouseX, mouseY,  200*nx, 200*ny );
    }
}
```

Using Noise to Affect Color

```javascript
float NoiseFactor = 200.;    // larger to make the noise gentler
int NoiseSeed = 22019;      // start the random number sequence

function setup( )
{
    createCanvas( 800, 800 );
    colorMode( RGB );
    background( 200, 200, 255 );
    fill( 255, 255, 0 );
    stroke( 0, 0, 0 );
    noiseSeed( NoiseSeed );
    noiseDetail( 4 );
}

In draw( ):  
{
    if ( mousesPressed )
    {
        float nx = noise( mouseX/NoiseFactor );
        float ny = noise( mouseY/NoiseFactor );
        int red  = int( nx*255. );
        int green = int( ny*255. );
        fill( red, green, 0 );
        ellipse( mouseX, mouseY,  200*nx, 200*ny );
    }
}
Using 2D Noise to Affect Color

float NoiseFactor = 200.;  // larger to make the noise gentler

function setup()
{
    createCanvas( 600, 600 );
    colorMode( RGB );
    background( 200, 200, 255 );
    fill( 255, 255, 0 );
    stroke( 0, 0, 0 );
    noiseDetail( 4 );
}

function draw()
{
    for( int x = 0; x < width; x++ )
    {
        for( int y = 0; y < height; y++ )
        {
            noiseSeed( 0 );
            int red = int( 255.*noise( x/NoiseFactor, y/NoiseFactor ) );
            noiseSeed( 1000 );
            int green = int( 255.*noise( x/NoiseFactor, y/NoiseFactor ) );
            noiseSeed( 2000 );
            int blue = int( 255.*noise( x/NoiseFactor, y/NoiseFactor ) );
            stroke( red, green, blue );
            point( x, y );
        }
    }
}

//noLoop();
//saveFrame( "ColorClouds.png" );

Here are some fun things to try (make the window size smaller first!):
• What happens if you make NoiseFactor larger? Smaller?
• What happens if you only stroke with (red, green, 0.)?
• What if you only use red and blue? Green and blue?
De-bouncing Keyboard Keys

boolean StillPressed = false;

function draw( )
{
    background( 200, 200, 255 );
    if( isKeyPressed )
    {
        if( ! StillPressed )    // same as saying "if( StillPressed == false )"
        {
            switch( key )
            {
                case 'a':
                    Animate = ! Animate;
                    break;
                case 'f':
                    FillSphere = ! FillSphere;
                    break;
            }

            StillPressed = true;
        }
        else
        {
            StillPressed = false;
        }
    }

    "Booleans" are variables that can be either “true” or “false”
    The exclamation point means “not”. It changes a true into a false, and a false into a true.
    If the key is not still pressed from before, go ahead and process the decisions in the switch statement
    If the key is still pressed from before, skip around the switch statement
    If the key is not pressed any more, set StillPressed to false
Remember This?
There is also no reason we can’t gradually change the radius …

```cpp
function Spiral( int xc, int yc, int r0, int r1, int numsegs, int numturns )
{
    float dang = numturns * (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );

    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float newrad = map( i, 0, numsegs, r0, r1 );
        float x = xc + newrad * cos(ang);
        float y = yc + newrad * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }

    endShape( );
}
```
Remember This?
There is also no reason we can’t gradually change the radius …

```cpp
void draw( )
{
    stroke( 50, 50, 255 );
    strokeWeight( 5 );
    noFill( );
    Spiral( 300, 300, 20, 200, 1000, 10 );
}
```

But, what if we change the radius as a function of the angle we are at right now?

Setting the radius as a function of the angle

```cpp
function Polar( int xc, int yc, int factor, int numsegs, int numturns )
{
    float dang = numturns * (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );

    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float newrad = 200. * sin(factor*ang);
        float x = xc + newrad * cos(ang);
        float y = yc + newrad * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }

    endShape( );
}
```
Setting the radius as a function of the angle

```c
void
draw(  )
{
    stroke( 50, 50, 255 );
    strokeWeight( 5 );
    noFill( );
    Polar( 300, 300, 4, 1000, 8 );
}
```

It's a lot of fun to experiment with different values for the factor variable!

Some Other Polar Patterns

\[
r = \sin \theta + \sin^3 \left( \frac{5\theta}{2} \right)
\]

\[
r = \sin \left( \frac{8\theta}{5} \right)
\]

Limaçons (French for “snails”):
\[ r = 1 + c \cdot \sin \theta \]
\[-2 \leq c \leq 2.5\]

\[ c = 1 \text{ is a “cardiod”} \]

Can We Imitate a Spirograph™?

Looks like an Oreo, but it's not. 😊
float BigR = 200.;
float SmallR = 150.;
float D = 120.;

function setup()
{
    createCanvas(800, 800);
    stroke(0, 0, 0);
    strokeWeight(2);
    noFill();
}
function draw() {
    background( 200, 200, 255 );
    translate( 400, 400 );

    beginShape();
    for( int t = 0; t <= 10*360; t = t + 2 )
    {
        float bigTheta = radians( t );
        float smallTheta = - ( BigR / SmallR ) * bigTheta;
        float x = ( BigR - SmallR ) * cos( bigTheta ) + D * cos( smallTheta );
        float y = ( BigR - SmallR ) * sin( bigTheta ) + D * sin( smallTheta );
        vertex( x, y );
    }
    endShape();
}
Arrays

Arrays Can Hold and Use Multiple Numbers with the Same Name, and, they let You Write a for-loop to Use Them!

```java
int X[ ] = { 100, 200, 300, 400, 500, 600, 700 };
int Y[ ] = { 100, 100, 100, 200, 200, 200, 300 };

function setup( )
{
    createCanvas( 800, 800 );
    background( 200, 200, 255 );
    stroke( 0, 0, 0 );
    fill( 255, 255, 0 );
}

function draw( )
{
    for( int i = 0; i < X.length; i = i + 1 )
    {
        ellipse( X[i], Y[i], 100, 50 );
    }
}
```

Processing will tell you the length of an array
### A Way of Declaring Arrays That Lets You Fill Them Later

```java
int NumPoints = 5000;
int W = 800;
int H = 800;
int[ ] X;
int[ ] Y;
int[ ] R;
int[ ] G;
int[ ] B;
```

Total number of points

The arrays that will hold the points and the colors. They have only been declared. They don’t yet have any memory given to them.

### Allocating and Filling Arrays

```java
function setup( )
{
    X = new int[NumPoints];
    Y = new int[NumPoints];
    R = new int[NumPoints];
    G = new int[NumPoints];
    B = new int[NumPoints];

    createCanvas( 800, 800 );
    for( int i = 0; i < NumPoints; i = i + 1 )
    {
        X[i] = int( random( 0, W ) );
        Y[i] = int( random( 0, H ) );
        R[i] = int( random( 0, 255 ) );
        G[i] = int( random( 0, 255 ) );
        B[i] = int( random( 0, 255 ) );
    }
}
```

Allocate the arrays to hold the random points and the random colors. At this point, memory has been given to them, but they don’t have any values assigned.

Each point is assigned a random location.

Each point is assigned a random color. There is no significance to each point’s specific color – it is to look cool!

Note: Array indices start at 0 and end at the number of elements minus one.
So, an array dimensioned [10] indexes from 0 to 9.
Using the Arrays in a for-loop

```cpp
function draw() {
    background(200, 200, 255);
    stroke(0, 0, 0);

    for(int i = 0; i < NumPoints; i = i + 1)
    {
        fill(R[i], G[i], B[i]);
        ellipse(X[i], Y[i], 8, 8);
    }
}
```

Draw each point with its color.

A Cool Pattern
For our next trick, during each frame we are going move each point halfway towards one of three target triangle vertices. Which target to use is chosen at random.

### Declaring Arrays

```
int NumPoints = 5000;  // Total number of random points
int TARGET_SIZE = 40;  // Size of the target vertices
int W = 800;
int H = 800;
int [ ] X;
int [ ] Y;
int [ ] R;
int [ ] G;
int [ ] B;
int [ ] XC = { 50, W/2, W-50 };  // The arrays that hold the points and the colors. They have only been declared. They don’t yet have any memory given to them.
int [ ] YC = { H-50, 50, H-50 };  // The arrays that hold the three center points. Because of the way this was coded, these arrays do have memory given to them.
```
function setup( )
{
    X = new int [NumPoints];
    Y = new int [NumPoints];
    R = new int [NumPoints];
    G = new int [NumPoints];
    B = new int [NumPoints];

    createCanvas( W, H );
    for( int i = 0; i < NumPoints; i = i + 1 )
    {
        X[i] = int( random( 0, W ) );
        Y[i] = int( random( 0, H ) );
        R[i] = int( random( 0, 255 ) );
        G[i] = int( random( 0, 255 ) );
        B[i] = int( random( 0, 255 ) );
    }

    frameRate( 2 );
}

Each point is assigned a random location.

Each point is assigned a random color. There is no significance to each point's specific color – it is to look cool!

Perform the animation at 2 frames per second so that we can actually see it. Otherwise, it will be too fast.

function draw( )
{
    background( 200, 200, 255 );
    fill( 255, 0, 0 );
    ellipse( XC[0], YC[0], TARGET_SIZE, TARGET_SIZE );
    fill( 0, 255, 0 );
    ellipse( XC[1], YC[1], TARGET_SIZE, TARGET_SIZE );
    fill( 0, 0, 255 );
    ellipse( XC[2], YC[2], TARGET_SIZE, TARGET_SIZE );

    stroke( 0, 0, 0 );
    for( int i = 0; i < NumPoints; i = i + 1 )
    {
        fill( R[i], G[i], B[i] );
        ellipse( X[i], Y[i], 8, 8 );
    }
}

Draw the three targets.

Draw each point with its color.

Re-compute each point's position by randomly picking one of the targets (0, 1, or 2) and moving halfway towards it.
A Surprising Result

Mathematicians call shapes like this “attractors”

Data: Reading, Analyzing, Plotting

Oregon State University
Computer Graphics

Mike Bailey
mjb@cs.oregonstate.edu
function
setup()
{
  createCanvas( 800, 800 );
  noFill( );

  String [ ] lines = loadStrings( "data.txt" );
  if( lines == null )
  {
    println( "Cannot open data.txt" );
    exit( );
  }

  int numPoints = int( lines[0] );
  println( "numPoints = " + numPoints );

  Everything is done in setup( ) because it only needs to happen once

float [ ] x = new float [ numPoints ];
float [ ] y = new float [ numPoints ];
for( int i = 0; i < numPoints; i = i + 1 )
{
  y[ i ] = int( lines[i+1] );
  println( "y[" + i + "] = " + y[ i ] );
}
Reading from a File, III

Everything is done in `setup( )` because it only needs to happen once

```c
float sum = 0.;
for( int i = 0; i < numPoints; i = i + 1 )
{
    sum = sum + y[i];
}
float average = sum / float(numPoints);
println( "average = " + average );

sum = 0.;
for( int i = 0; i < numPoints; i = i + 1 )
{
    float diff = y[i] - average;
    sum = sum + ( diff * diff );
}
float stdev = sqrt( sum / float(numPoints - 1) );
println( "stdev = " + stdev );
```

Reading from a File, IV

```c
float ymin = y[0];
float ymax = y[0];
for( int i = 1; i < numPoints; i = i + 1 )
{
    if( y[i] < ymin )
        ymin = y[i];
    if( y[i] > ymax )
        ymax = y[i];
}
float xscale = float(width) / float(numPoints - 1);
float yscale = float(height) / ( ymax - ymin );
background( 200, 255, 200 );
stroke( 0, 0, 255 );
strokeWeight( 3 );
beginShape( );
for( int i = 0; i < numPoints; i = i + 1 )
{
    vertex( xscale * float(i), height - yscale * ( y[i] - ymin ) );
}
endShape( );
```

Find the minimum and maximum values so we know how to scale the vertical part of the graph.
The Data File:

- 12 Number of Points
- 47.
- 51.
- 56.
- 61.
- 67.
- 73.
- 82.
- 83.
- 77.
- 65.
- 53.
- 46.

Average monthly temperatures in Corvallis

Challenge question: How could you draw little circles at each data point?

Processing !