Programming with Processing!

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

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

Programming Through the Ages

Textiles

Jacquard Loom, circa 1804

Music

Albany Carousel and Museum

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

These images depict various topics including programming with Processing, programming through the ages, textiles, and music.
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!
Coordinate Systems for Processing Programs

(X=0, Y=0) (X=width - 1, Y=0)

(X=100, Y=200) ΔX=150

(X=0, Y=height - 1) (X=width - 1, Y=height - 1)

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

Black 0 0 0
White 255 255 255
Red 255 0 0
Orange 255 128 0
Yellow 255 255 0
Green 0 255 0
Cyan 0 255 255
Blue 0 0 255
Magenta 255 0 255

Writing a Processing Program – Try This!

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 setup() function. Processing calls this once when your program starts.

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

Colors for Processing Programs

This is referred to as “Additive Color”
Running Your Processing Programs

- Click here to run your program

Enjoying the Output of Your Processing Program

Other Functions to use when Writing Processing Programs

- setup
- createCanvas
- color
- stroke
- fill

Variables and For-loops

- setup
- color
- stroke
- fill
- draw
- rect

Variables:
- height
- width
- strokeWeight
- stroke

Looping:
- for
- while

Examples:
- setBackground
- setColor
- drawImage
Variables – using symbols instead of just numbers

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.

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

Arithmetic operations in programming are:
+ Addition
- Subtraction
* Multiplication
/ Division
( ) Grouping

Drawing One Rectangle is Pretty Straightforward

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

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

```c
rect( 100, 200, 150, 50 );
rect( 110, 210, 150, 50 );
rect( 120, 220, 150, 50 );
```

```
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:

```c
for( ; ; )
```

- 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!**

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 ) );
```

**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.
More Sophisticated Relationships:

The map() function

```c
void
    draw()
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
    for( int x = 0; x < 400; x = x + 18 )
    {
        int y = x;
        int red = int( map( x, 0, 399, 0, 255 ) );
        int green = int( map( y, 0, 299, 0, 255 ) );
        green = 3 * green / 4;
        //println( "x = " + x + " y = " + y + " red = " + red + " green = " + green );
        fill( red, green, 50 );
        rect( x, y, 150, 50 );
    }
}
```

The 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 + 18 )
    {
        int y = x;
        int red = int( map( x, 0, 399, 0, 255 ) );
        int green = int( map( y, 0, 399, 0, 255 ) );
        //println( "x = " + x + " y = " + y + " red = " + red + " green = " + green );
        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

```c
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

```c
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

```c
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

```c
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

```c
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

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

Reacting to the Mouse and Keyboard

- `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 `mousPressed`, `mouseX`, and `mouseY` Variables

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

`mousPressed` 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( mousPressed )
  {
    ellipse( mouseX, mouseY, 50, 50 );
  }
}
```

`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.

The `stroke()` and `fill()` calls have been moved to `setup()`.

The `stroke()` and `fill()` calls have been moved to `setup()`.

The `stroke()` and `fill()` calls have been moved to `setup()`.
What if you want to read the Special Keys?

```java
... 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

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( );
```

void `draw( )`

```
  stroke( 0, 0, 0 );
  fill( 255, 50, 50 );
  beginShape( );
    vertex( 109, 100 );
    vertex( 199, 100 );
    vertex( 200, 300 );
    vertex( 300, 300 );
    vertex( 400, 50 );
  endShape( );
```
Setting the size and drawing the text

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

void draw()
{
  fill(8, 0, 0);
  text("ABC", 50, 50);
  fill(8, 0, 255);
  text("DEF", 50, 80);
}
```

It is Often Nice to Transform Entire Objects at Once

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

void draw()
{
  rect(0, 8, 100, 50);
}
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!
There is also a shearY transformation function:

```cpp
void draw() {
    shearY(radians(45.));
    rect(8, 6, 100, 50);
}
```

Transformations Accumulate!

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

is the same as:

```cpp
function draw() {
    rotate(radians(20.));
    ...
}
```

Transformation Order Matters!

```cpp
void draw() {
    translate(180, 280);
    rotate(radians(20.));
    rect(0, 6, 100, 50);
}
```

You Can Save and Un-do Transformations

```cpp
void draw() {
    translate(200, 300);
    pushMatrix();
    shearX(radians(45.));
    rect(6, 6, 200, 108);
    popMatrix();
    fill(255, 8, 8);
    rotate(radians(-45.));
    rect(8, 0, 200, 200);
}
```
Transformations and for-loops

```cpp
void draw()
{
    translate(200, 200);  // Move the origin to (200, 200)
    for (int degrees = 0; degrees <= 360; degrees += 36)  // Use degrees rather than radians
    {
        pushMatrix();  // Save the current transformation matrix
        rotate(radians(degrees));  // Rotate by degrees
        rect(0, 0, 100, 30);  // Draw a rectangle
        popMatrix();  // Restore the original transformation matrix
    }
}
```

What's the Difference?

```cpp
void draw()
{
    translate(200, 200);  // Move the origin to (200, 200)
    for (int degrees = 0; degrees <= 360; degrees += 36)
    {
        pushMatrix();  // Save the current transformation matrix
        rotate(radians(degrees));  // Rotate by degrees
        rect(100, -15, 100, 30);  // Draw a rectangle
        popMatrix();  // Restore the original transformation matrix
    }
}
```
Rotating While Changing Color and Size

```c
void draw() {
    // translate to the origin
    translate(0, 0, 0);
    // scale to the desired size
    scale(2, 2, 2);
    // rotate around the origin
    rotateX(radians);
    rotateY(radians);
    rotateZ(radians);
}
```

And, there are even 3D Transformations

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

But, we will get to those later …

Images

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)
Loading and Drawing an Image

`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`.

What X-Y to draw its upper-left corner at.

How many pixels to use to draw the image.

What Happens if You Use Less Pixels than the Window Has?

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

What Happens if You Use a Different Aspect Ratio?

```java
void draw()
{
  image( MyImage, 50, 50, 600, 300 );
}
```
First, We Need to Understand Something about Angles

θ

X

Y

One of the things we notice is that each angle θ has a unique X and Y that goes with it.

These are different for each θ.

Fortunately, centuries ago, people developed tables of those X and Y values as functions of θ.

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

θ
X
Y

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 θ angles, we can determine all of the X's and Y's that we need to draw a circle.

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

Processing Doesn’t Include a Circle-Drawing Function, So We Add Our Own

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

Why 2.*PI?

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 build-in functions to convert between the two:
float rad = radians( deg );
float deg = degrees( rad );

Circle, Pentagon, Octagon!

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( );
  }

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

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

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 );
  }

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 ...
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;
    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;
    }
}
```

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

```cpp
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

```cpp
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

```cpp
function draw() {
    background(200, 200, 255);
    if (isKeyPressed) {
        if (!StillPressed) {
            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;
        }
    } else {
        StillPressed = false;
    }

    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

Start With Something We've Seen Before

```javascript
void setup() {
  size(800, 800);
  colorMode(HSB);
  background(230, 200, 255);
  stroke(4, 8, 155);
  strokeWeight(4.0);
  fill(255, 59, 50);
  rect();
}

void draw() {
  beginShape();
  vertex(189, 100);
  vertex(189, 400);
  vertex(289, 400);
  vertex(389, 200);
  vertex(489, 50);
  endShape();
}
```

Pure Randomness is Pretty Jarring

```javascript
void draw() {
  background(200, 200, 255);
  beginShape();
  for (int x = 0; x < width; x = x + 5) {
    int y = int(random(80, height));
    vertex(x, y);
  }
  endShape();
}
A Better Approach – Add a Random Number to the Current Value

```plaintext
void draw() {
  background(200, 200, 255);
  float y = height / 2;
  beginShape();
  for (int x = 0; x < width; x++) {
    int dy = int(random(-height/10, height/10));
    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

```
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 );
}
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 ) );
    stroke( 255, green, 0 );
    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

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

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 );
    int red  = int( nx*255. );
    int green = int( ny*255. );
    fill( red, green, 0. );
    ellipse( mouseX, mouseY, 100, 100 );
  }
}
Using 2D Noise to Affect Color

```java
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() {
    noLoop( );
    saveFrame( "ColorClouds.png" );

    // takes about 40 seconds to do 600x600 = approx 9,000 points/sec

    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 );
        }
    }
}
```

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

```java
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 …

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; 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( );
}

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( );
}

But, what if we change the radius as a function of the angle we are at right now?
It's a lot of fun to experiment with different values for the \textit{factor} variable!

\begin{verbatim}
void draw( )
{
    stroke( 50, 50, 255 );
    strokeWeight( 5 );
    noFill();
    Polar( 300, 300, 4, 1000, 8 );
}
\end{verbatim}

\begin{equation}
r = \sin \theta + \sin \left( \frac{5\theta}{2} \right)
\end{equation}

\begin{equation}
r = \sin \left( \frac{8\theta}{5} \right)
\end{equation}

\textbf{Some Other Polar Patterns}

\textbf{Limaçons (French for “snails”):}

\[ r = 1 + c \cdot \sin \theta \]
\[-2 \leq c \leq 2.5\]

\textit{c = 1} is a “cardiod”

\textbf{Can We Imitate a Spirograph™?}

Looks like an Oreo, but it’s not. 😊
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!

```plaintext
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 );
    }
}
```

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

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

```
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.

A Cool Pattern will be made even Cooler

int NumPoints = 5000;
int TARGET_SIZE = 40;
int W = 800;
int H = 800;
int [ ] X;
int [ ] Y;
int [ ] R;
int [ ] G;
int [ ] B;
int [ ] XC = {50, W/2, W-50};
int [ ] YC = {H-50, 50, H-50};

Total number of random points
Size of the target vertices
The arrays that hold the three center points. Because of the way this was coded, these arrays do have memory given to them.

Declaring Arrays

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.

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.

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.
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.
Setup 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.

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);
elipse(XC[0], YC[0], TARGET_SIZE, TARGET_SIZE);
fill(0, 255, 0);
elipse(XC[1], YC[1], TARGET_SIZE, TARGET_SIZE);
fill(0, 0, 255);
elipse(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]);
elipse(X[i], Y[i], 8, 8);
}
for(int i = 0; i < NumPoints; i = i + 1) {
int randTarget = int(random(0.000, 2.999));
X[i] = (X[i] + XC[randTarget]) / 2;
Y[i] = (Y[i] + YC[randTarget]) / 2;
}

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.

Mathematicians call shapes like this "attractors".
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);
    int numPoints = int(lines[0]);
    println("numPoints = " + numPoints);
}

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]);
}

float sum = 0.0;
for(int i = 0; i < numPoints; i = i + 1) {
    sum = sum + y[i];
}
float average = sum / float(numPoints);
println("average = " + average);
float sum = 0.0;
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);

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();
The Data File:

- Number of Points
- Average monthly temperatures in Corvallis

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