In your favorite web browser, go to: https://editor.p5js.org/
Here's what you will see:

Menu headers
- Run your program
- Stop this program

Program-writing/editing area
Processing message area

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

- **Initial Position:** (X=0, Y=0)
- **Final Position:** (X=width - 1, Y=height - 1)
- **ΔX:** 150
- **ΔY:** 50

### 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

#### Colors

- **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!

```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 `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.

### Running Your Processing Programs

Click here to run your program:

```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);
}
```

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stroke()</td>
<td>Draws a line with a given color</td>
</tr>
<tr>
<td>fill()</td>
<td>Fills a shape with color</td>
</tr>
<tr>
<td>rect()</td>
<td>Draws a rectangle</td>
</tr>
<tr>
<td>ellipse()</td>
<td>Draws an ellipse</td>
</tr>
<tr>
<td>triangle()</td>
<td>Draws a triangle</td>
</tr>
<tr>
<td>circle()</td>
<td>Draws a circle</td>
</tr>
<tr>
<td>beginShape()</td>
<td>開始使用一個新的閉合形狀，如矩形或多邊形</td>
</tr>
<tr>
<td>endShape()</td>
<td>結束當前的閉合形狀</td>
</tr>
</tbody>
</table>

Variables and For-loops

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

Variables – using symbols instead of just numbers

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;
  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);
```
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
Do this at the end of one loop, but before the start of the next one

Keep looping as long as this equation is true

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 ) );
```

For-loops to the Rescue!

The `map()` function can also do blending.

Interpolate one forward and the other one backwards

All-green morphs into all-red.
If-statements

Mike Bailey
mjb@cs.oregonstate.edu

THEREST.pptx

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

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

These Operators Make up the Possible Conditions:
<c> 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

```java
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

```java
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

```java
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

```java
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

```
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 );
}
```

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

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

Reacting to the Mouse and Keyboard

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

The `mouseIsPressed`, `mouseX`, and `mouseY` Variables

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

The `mouseIsPressed`, `mouseX`, and `mouseY` Variables

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

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

`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?

```java
if (isKeyPressed) {
  if (key == CODED) {
    switch (keyCode) {
      case UP: // up-arrow
        // draw code
        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

```
beginShape();
  vertex(x0, y0);
  vertex(x1, y1);
  vertex(x2, y2);
  // ... other vertices
endShape();
```

Rectangles are Good, but Arbitrary Polygons are nice too

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

It is Often Nice to Transform Entire Objects at Once

Translation

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 = \pi/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.

Transformation Order Matters!

You Can Save and Un-do Transformations
Transformations and for-loops

What's the Difference?

Rotating While Changing Color and Size

And, there are even 3D Transformations

```c
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)

Loading and Drawing an Image

```
PImage MyImage;

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

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

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

```
void draw( )
{
    image( MyImage, 50, 50, 480, 480 );
}
```

What Happens if You Use a Different Aspect Ratio?
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 $\theta$. 

$\theta$
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.

How People used to Lookup Sines and Cosines – Fortunately We Now Have Calculators and Computers

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.

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.

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

```cpp
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. $6$ gives an octagon. $4$ gives you a square. Etc.
Why 2.0PI?

```cpp
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 2\(\pi\) 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:

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

Circle, Pentagon, Octagon!

```cpp
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:

```cpp
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...

```cpp
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 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 ...

```
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( );
}
```

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 );
orotateY( 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);
}
```

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

```java
function draw() {
    background(200, 200, 255);
    if (isKeyPressed) {
        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;
    }
    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.;
    }
    popMatrix();
}
```

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

Randomness
Start With Something We’ve Seen Before

Pure Randomness is Pretty Jarring

A Better Approach – Add a Random Number to the Current Value

Computer Graphics Noise

Noise Octaves Create More Detail

The Setup

- The built-in \texttt{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 \textit{Coherent} in that the noise value at one point is close to the noise value at the next point.
- Setting \texttt{noiseSeed()} makes it \textit{Repeatable} in that the same input always gives the same output.

\begin{align*}
\text{float NoiseFactor} & = 200.; \quad \text{larger to make the noise gentler} \\
\text{int NoiseSeed} & = 22019; \quad \text{start the random number sequence} \\
\text{int MinOctaves} & = 1; \\
\text{int MaxOctaves} & = 8; \\
\end{align*}

\begin{function}
\texttt{setup}() \\
\texttt{createCanvas(800, 600);} \\
\texttt{colorMode( RGB);} \\
\texttt{noFill();} \\
\texttt{noiseSeed( NoiseSeed );} \\
\end{function}

\begin{function}
\texttt{draw}() \\
\texttt{beginShape();} \\
\texttt{vertex(100, 100);} \\
\texttt{vertex(200, 400);} \\
\texttt{vertex(300, 200);} \\
\texttt{vertex(400, 100);} \\
\texttt{endShape();} \\
\end{function}
```javascript
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( );
    }
}
```

```
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);
}
```

```
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

Mike Bailey
mjb@cs.oregonstate.edu

Debouncing_keys.pptx

booleanStillPressed = false;

functiondraw()
{
    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"

If the key is not still pressed from before, go ahead and process the decisions in the switch statement

The exclamation point means "not". It changes a true into a false, and a false into a true.

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

Polar Equations

Mike Bailey
mjb@cs.oregonstate.edu

polar.pptx

Remember This?
There is also no reason we can’t gradually change the radius ...

functionSpiral(intxc, intyc, intr0, intr1, intnumsegs, intnumturns)
{
    floatdang = numturns * (2.*PI) / float(numsegs);
    floatang = 0.;
    beginShape();
    for(inti = 0; i <= numsegs; i = i + 1)
    {
        floatnewrad = map(i, 0, numsegs, r0, r1);
        floatx = xc + newrad * cos(ang);
        floaty = 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?

functionPolar(intxc, intyc, intfactor, intnumsegs, intnumturns)
{
    floatdang = numturns * (2.*PI) / float(numsegs);
    floatang = 0.;
    beginShape();
    for(inti = 0; i <= numsegs; i = i + 1)
    {
        floatnewrad = 200. * sin(factor*ang);
        floatx = xc + newrad * cos(ang);
        floaty = yc + newrad * sin(ang);
        vertex(x, y);
        ang = ang + dang;
    }
    endShape();
}
It's a lot of fun to experiment with different values for the factor variable!

$$r = \sin \theta + \sin \left( \frac{5\theta}{2} \right)$$

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

Limaçons (French for “anaila”):

$$r = 1 + c \cdot \sin \theta$$

$$-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$$

$$c = 1$$ is a “cardiod”

Can We Imitate a Spirograph™?

Looks like an Oreo, but it's not.

Spirograph™

float BigR = 200.;
float SmallR = 150.;
float D = 120.;

function setup()
{
  createCanvas(800, 800);
  stroke(0, 0, 0);
  strokeWeight(2);
  noFill();
  Polar(360, 300, 4, 1000, 8);
}

Spirograph™
Spirograph™

```cpp
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 r = BigR - SmallR;
        float x = r * cos(bigTheta) + D * cos(smallTheta);
        float y = r * 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!

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

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));
    }
}
```

Allocating and Filling Arrays

Allocating 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.

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

Processing will tell you the length of an array.

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.
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 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 hold the three center points. Because of the way this was coded, these arrays do have memory given to them.

Intializing the arrays to hold the random points
Intializing the arrays to hold the random colors
Intializing the three target points

int randTarget = int(random(0.000, 2.999));
X[i] = X[randTarget] + (X[i] - XC[randTarget]) / 2;
Y[i] = Y[randTarget] + (Y[i] - YC[randTarget]) / 2;

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

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

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!

Alloacting and Filling Arrays

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.

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!

Allocating and Filling Arrays

Using the Arrays in a for-loop

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.

Function to draw the three targets.
A Surprising Result

Mathematicians call shapes like this "attractors"

Data: Reading, Analyzing, Plotting

Mike Bailey
mjb@cs.oregonstate.edu
data.pptx

Reading from a File, I
Everything is done in setup() because it only needs to happen once

```java
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);
}
```

Reading from a File, II
Everything is done in setup() because it only needs to happen once

```java
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

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

```java
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:

<table>
<thead>
<tr>
<th>Number of Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
<tr>
<td>47</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>56</td>
</tr>
<tr>
<td>61</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>73</td>
</tr>
<tr>
<td>82</td>
</tr>
<tr>
<td>83</td>
</tr>
<tr>
<td>77</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>46</td>
</tr>
</tbody>
</table>

Average monthly temperatures in Corvallis

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