Programming with Processing!

A Warning about the Note Coverage

Why Do We Have These Notes?

Programming Through the Ages

- A "program" is a set of instructions that you can store and playback later. This sounds like a computer-thing, but the idea of a "program" has been around for hundreds of years.

- The earliest known "program" is (apparently) a mechanical music playback device developed in Baghdad in the 9th century. ([https://en.wikipedia.org/wiki/Music_box](https://en.wikipedia.org/wiki/Music_box)) You can find a similar device in Oregon today...
Music Box Programming

... at the Albany (Oregon) Carousel and Museum

Another Historic Example is Textile Programming

Jacquard Loom, circa 1804
And, of course, there is the ever-fun Player Piano

https://en.wikipedia.org/wiki/Piano_roll

Computers Eventually Imitated Historic Methods using Punch Cards

circa 1972

The Processing Programming Language

Where to Find Processing

In your favorite web browser, go to: https://p5js.org/

Here’s what you will see:

Go here to start using Processing

These are good links to check out!

Processing includes a collection of spectacular example programs
Click on the Editor link, or navigate to: https://editor.p5js.org/
Either way, here's what you will see:

Running Processing

Now, click this button!
Here's what you will get

Don't worry – it will get better 😊
With *Processing*, you get to do real-world programming that gives you visual output. You get to make cool pictures at the same time you are learning to program. This opens up a world of opportunities for you!

First, Remember How Graph Paper Works

The Greek letter delta, $\Delta$, is the mathematics symbol for “the change in”

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

The smallest number you can use for each is 0.
The largest number you can use for each 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 Computer Graphics Monitors: Additive Colors (RGB)

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

Yes, Our Vision System Really Does Mush Red and Green Together to Make Yellow!

Colors for Paints, Toners, and Clear Plastic: Subtractive Colors (CMYK)

Green = C + Y
R = Red
G = Green
B = Blue
W = White
C = Cyan
M = Magenta
Y = Yellow
K = Black
Writing a Processing Program – Try This!

```java
function setup() {
    createCanvas(800, 600);
    background(200, 200, 255);
    stroke(0, 0, 0);
    fill(255, 50, 50);
}

function draw() {
    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.

You don't need to type all this in — you can copy-and-paste it from these notes!

Enjoying the Output of Your First Processing Program

Don't worry – it will get better 😊

Running Your Processing Programs

Click here to run your program

Some Functions to use when Writing Processing Programs

- `color(x, y, z, w)` — Set the current color to `(x, y, z, w)`
- `Drawing(x, y)` — Filling the color `x`
- `Drawing(fill, color)` — Don’t use `fill`ing
- `Drawing(stroke)` — Don’t use `strok`ing
- `Drawing(strokeWeight(w))` — Thickness of the `strok`e
- `Math(abs(x))` — Absolute value
- `Math(lerp(x, y, z, blend))` — Linearly map the input variable from the range `[low, high]` to `[lowest, highest]`
- `Math(map(x, fi, fl, hi, hl)` — Minimum of the two numbers
- `Math(random(low, high))` — Randomness returns a random number between low and high
- `Setup(background(x, y, z, w))` — Set the background to `(x, y, z, w)`
- `Setup(createCanvas(x, y, z, w))` — Set the size of the drawing window to `(x, y, z, w)`
- `Setup(draw)` — The function that gets called when your program starts
- `Setup(setup)` — The function that gets called when your program starts
- `Shapes(startVertex)` — Start drawing to vertices
- `Shapes(endVertex)` — Finish drawing to vertices
- `Shapes(points, p1, p2, p3)` — Draw a line
- `Shapes(point(x, y))` — Draw a point at (x,y)
- `Shapes(square(x, y, z, w)` — Draw a square
- `Shapes(ellipse(x, y, z, w)` — Draw an ellipse
- `Shapes(triangle(x, y, z, w)` — Draw a triangle
- `Shapes(quad(x, y, z, w)` — Draw a quadrilateral
- `Shapes(point(x, y, z, w)` — Draw a point at (x,y,z,w)
- `Text(text, x, y)` — Draw the text `x` on the screen at (x,y) with the current fill color
- `Variables` — Screen height in pixels
- `Variables` — Screen width in pixels
- `Variables` — Width
- `Variables` — Height

Don't worry – it will get better 😊
Variables

Arithmetic operations in programming are:

+ Addition
- Subtraction
* Multiplication
/ Division
( ) Grouping

Variables – using symbols instead of just numbers

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

```
function draw()
{
    let x = 100;
    let y = 2*x;
    rect(x, y, 150, 50);
}
```

"let" just says that you are defining a variable.

We can use variables to create relationships.

```
function draw()
{
    let x = 100;
    let y = 2*x;
    rect(x, y, 150, 50);
}
```

When you assign a number to x, then y will automatically be twice as big as whatever you set x to be.

Variables – using symbols instead of just numbers

```
function draw()
{
    let x = 100;
    let y = 200;
    rect(x, y, 150, 50);
}
```

Drawing Lines and Polygons
Rectangles are Fun, but Arbitrary Lines and Polygons are Funner

Easy – just list the coordinates:

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

for-loops

```javascript
rect( 100, 200, 150, 50 );
```
But, This Gets Awfully Boring if You Want to Draw 100 Rectangles!

```javascript
function draw() {
  for( let x = 0 ; x < 400 ; x = x + 10 ) {
    let y = x;
    rect( x, y, 150, 50 );
  }
}
```

for-loops to the Rescue!

Repeating a code pattern is a common theme in programming. This line is called a "for-loop". It is very handy for repeating patterns of code.

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

1. Do this equation once at the start
2. Keep looping as long as this test is true
3. Do this at the end of one loop, but before the start of the next one

```javascript
for( let x = 0 ; x < 400 ; x = x + 10 )
```

Yes, the semi-colons (;) are necessary!

Drawing Circles and Other Regular Polygons, I
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 $\theta$.

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

The $X$ and $Y$ are different for each $\theta$.

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.

In Earlier Times, People Looked up Sines and Cosines in Books and on Slide Rules – Fortunately We Now Have Calculators and Computers
Cosines and Sines are Really Ratios

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

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

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

\[ X = R \times \cos \theta \]
\[ Y = R \times \sin \theta \]

Processing Doesn't Include Regular Polygon-Drawing Functions, So We Add Our Own to the End of the Program

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

Why 2.*PI (= 2\pi)??

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

There are 360° (degrees) in a complete circle.
There are 2\pi (~6.28) radians in a complete circle.

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

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

```
let rad = radians( deg );
let deg = degrees( rad );
```
Circles, Pentagons, and Octagons -- oh my!

```plaintext
function draw() {
    fill(255, 50, 50);
    Shape(200, 200, 100, 36);
    fill(50, 255, 50);
    Shape(300, 300, 100, 5);
    fill(50, 50, 255);
    Shape(400, 400, 100, 8);
}
```

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

```plaintext
function Shape2(xc, yc, rx, ry, numsegs) {
    let dang = (2.*PI) / float(numsegs);
    let ang = 0.;
    beginShape();
    for( let i = 0; i <= numsegs; i = i + 1 ) {
        let x = xc + rx * cos(ang);
        let y = yc + ry * sin(ang);
        vertex(x, y);
        ang = ang + dang;
    }
    endShape();
}
```

The Processing `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:

```javascript
let red = int(map(x, 0, 399, 0, 255));
```

---

**Interpolate one forward and the other one backwards**

```javascript
function draw() {
  for(let x = 0; x < 400; x = x + 10) {
    let y = x;
    let red = int(map(x/0, 399, 0, 255));
    let green = int(map(y, 0, 399, 0, 255));
    fill(red, green, 0);
    rect(x, y, 150, 50);
  }
}
```

---

**All-Green morphs into All-Red**

```javascript
function draw() {
  for(let x = 0; x < 400; x = x + 10) {
    let y = x;
    let red = int(map(x, 0, 399, 0, 255));
    let green = int(map(y, 0, 399, 255, 0));
    fill(red, green, 0);
    rect(x, y, 150, 50);
  }
}
```
Drawing Circles and Other Regular Polygons, II

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

function Spiral( xc, yc, r1, r2, numsegs, numturns )
{
    let dang = numturns * (2.*PI) / float( numsegs );
    let ang = 0.;
    beginShape( );
    for( let i = 0; i <= numsegs; i = i + 1 )
    {
        let newrad = map( i,   0, numsegs,  r1, r2 );
        let x = xc + newrad * cos(ang);
        let 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( )
{
    let numobjects = 10;
    let radius = 200.;
    let xc = 300;
    let yc = 300;
    let numsegs = 20;
    let r = 50;
    let dang = (2.*PI) / float( numobjects - 1 );
    let ang = 0.;
    for( let i = 0; i < numobjects; i = i + 1 )
    {
        let x = xc + radius * cos(ang);
        let y = yc + radius * sin(ang);
        let red   = int( map( i,   0, numobjects - 1,     0, 255 ) );
        let blue = int( map( i,   0, numobjects - 1, 255,     0 ) );
        fill( red, 0, blue );
        Shape( x, y, r, numsegs );
        ang = ang + dang;
    }
}
Polar Equations

function Polar(xc, yc, factor, numsegs, numturns)
{
    let dang = numturns * (2.*PI) / float(numsegs);
    let theta = 0.;
    beginShape();
    for(let i = 0; i <= numsegs; i = i + 1)
    {
        let r = 200. * sin(factor*theta);
        let x = xc + r * cos(theta);
        let y = yc + r * sin(theta);
        vertex(x, y);
        theta = theta + dang;
    }
    endShape();
}

Setting the radius as a function of the angle

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

The Processing function `random()` takes in two numbers and returns a random number between them. Here it is being used to randomly position and size shapes:

```javascript
function setup() {
  createCanvas(300, 300);
  background(200, 200, 255);
  stroke(0, 0, 0);
  fill(255, 50, 50);
  noLoop();
}

function draw() {
  for (let i = 0; i < 20; i = i + 1) {
    let x = random(0, 300);
    let y = random(0, 300);
    let sizex = random(10, 70);
    let sizey = random(10, 70);
    rect(x, y, sizex, sizey);
  }
}
```

Or, also use it to pick colors:

```javascript
function draw() {
  for (let i = 0; i < 20; i = i + 1) {
    let x = random(0, 300);
    let y = random(0, 300);
    let sizex = random(10, 70);
    let sizey = random(10, 70);
    let r = random(50, 255);
    let g = random(50, 255);
    let b = random(50, 255);
    fill(r, g, b);
    rect(x, y, sizex, sizey);
  }
}
```

The Flower Garden
function setup( )
{
createCanvas( 600, 600 );
background( 200, 200, 255 );
stroke( 0, 0, 0 );
noLoop( );
}

function draw( )
{
for( let i = 0 ; i < 200 ; i = i + 1 )
{
let r = random( 50, 255 );
let g = random( 50, 255 );
let b = random( 50, 255 );
let xc = random( 0, width );
let yc = random( 0, height );
let factor = random( 3, 12 );
let size = random( 5, 40 );
fill( r, g, b );
Flower( xc, yc, factor, size, 150, 1 );
}
}

function Flower( xc, yc, factor, size, numsegs, numturns )
{
let dang = numturns * (2.*PI) / float( numsegs );
let theta = 0.;
bEGINShape( );
for( let i = 0; i < numsegs; i = i + 1 )
{
let r = size * sin(factor*theta);
let x = xc + r * cos(theta);
let y = yc + r * sin(theta);
vertex( x, y );
theta = theta + dang;
}
endShape( );
}

The Flower Garden

Be sure the call to noLoop( ) is included in setup( )!

Number of flowers to draw

You Get a Different Garden Every Time You Run the Program!

The Program Randomly Chooses the Flower's Color, Position, Size, and Number of Petals

Drawing Text

ABC
DEF
Setting the size and drawing the text

```javascript
function setup( ) {
    createCanvas( 400, 400 );
    background( 200, 200, 255 );
}

function draw( ) {
    fill( 0, 0, 0 );
    textSize( 20 );
    text( "ABC", 50, 50 );
    fill( 0, 0, 255 );
    textSize( 30 );
    text( "DEF", 50, 100 );
}
```

Text height in pixels
Text to draw
Where (x,y) to draw the text
Use fill( ) to set the text color

Processing Doesn't Save to Your Local Machine

It saves to the cloud. But it only does it if you have an account.

Fortunately, AWSEM / STEM Academy already has one.
So, go to the upper-right corner of your Processing window and click on Log in.
Then enter:

Username: awsem
Password: corvallis72542

You can create your own account if you want, but only do it with your parents' help.

Saving Your Processing Program and Getting It Back Later

The next trick is to click here and change the goofy name it gave your program to something more sensible, preferably something with your name in it and maybe something about what you were working on.
Then click File → Save

To bring back programs, click File → Open, look at the list of program names there, then click on the one you want to bring back.

The two I put there are FlowerGarden and PaintProgram.

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

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

These Operators Are the Possible Conditions to Test For:

<   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**

```javascript
function draw() {
    let x = 100;
    fill(0, 255, 0);
    for(let y = 0; y <= 500; y = y + 100) {
        if(y >= 200) {
            fill(255, 0, 0);
        }
        rect(x, y, 200, 50);
    }
}
```

---

**Your Code Often Wants to Test Something and Make a Decision Based On It or the Opposite Condition**

```javascript
if(condition) {
    do this1;
    do this2;
} else {
    do that1;
    do that2;
}
```

---

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

```javascript
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**

```javascript
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);
}
```

*key* is a Processing built-in variable that tells you what key has been hit on the keyboard.
Your Code Often Wants to Test Something and Make a Decision Based On It or Lots of Alternatives -- a Better Way

```
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 setup() {
  createCanvas( 600, 600 );
  background( 200, 200, 255 );
  stroke( 0, 0, 0 );
  fill( 255, 255, 0 );
}
```

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

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

```
function setup() {
  createCanvas( 600, 600 );
  background( 200, 200, 255 );
  stroke( 0, 0, 0 );
  fill( 255, 255, 0 );
}
```

```
function draw() {
  if( mouseIsPressed ) {
    ellipse( mouseX, mouseY, 50, 50 );
  }
}
```
The `mouseIsPressed`, `mouseX`, and `mouseY` Variables

```java
function draw() {
  if (keyIsPressed) {
    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 `isKeyPressed` and `key` Variables

- `keyIsPressed` 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 on the keyboard.
- The `switch/case` statements are Processing’s way of checking many values without having a whole slew of if-statements.

mouseIsPressed is a built-in variable that is always telling you if a mouse button has been pressed.

---

What if you want to read the Special Keys?

```java
if (keyIsPressed) {
  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
Let's Use Our Rectangle Object as an Example of Transformations

```javascript
function setup() {
  createCanvas( 800, 800 );
  background( 200, 200, 200 );
  stroke( 0, 0, 0 );
  fill(   0, 255, 255 );
}
function draw() {
  rect( 0, 0, 100, 50 );
}
```

It is Often Nice to Transform Entire Objects at Once

```javascript
function setup() {
  createCanvas( 800, 800 );
  background( 200, 200, 200 );
  stroke( 0, 0, 0 );
  fill(   0, 255, 255 );
}
function draw() {
  translate( 100, 200 );
  rect( 0, 0, 100, 50 );
}
```

The word “translate” means to “move around”.

Rotations and Scaling Happen Around the Origin

```javascript
function setup() {
  createCanvas( 800, 800 );
  background( 200, 200, 200 );
  stroke( 0, 0, 0 );
  fill(   0, 255, 255 );
}
function draw() {
  rect( 0, 0, 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 = pi/180 degrees

But, don't worry about this.

Processing gives you a function, `radians()`, to automatically convert degrees into radians, like this:

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

Use it!

There is also a shearY transformation function

Transformations Accumulate!

```cpp
is the same as:
```

```cpp
```
**Transformation Order Matters!**

```javascript
function draw() {
    rotate( radians(60.) );
    translate( 200, 300 );
    rect( 0, 0, 100, 50 );
}
```

**You Can Save and Restore Transformations**

```javascript
function draw() {
    translate(200, 300);
    push();
    shearX( radians(45.) );
    rect(0, 0, 200, 100);
    pop();
    fill(255, 0, 0);
    rotate( radians(-45.) );
    rect(0, 0, 200, 100);
}
```

**Transformations and for-loops**

```javascript
function draw() {
    translate( 200, 300 );
    for (let degrees = 0; degrees <= 360; degrees = degrees + 36 ) {
        push();
        rotate( radians(degrees) );
        rect( 0, 0, 100, 30 );
        pop();
    }
}
```

**Transformations and for-loops**

```javascript
function draw() {
    translate( 200, 300 );
    for (let degrees = 0; degrees <= 360; degrees = degrees + 36 ) {
        push();
        rotate( radians(degrees) );
        rect( 0, -15, 100, 30 );
        pop();
    }
}
```
What's the Difference?

Transformations and for-loops

function draw() {
  translate(200, 300);
  for(let degrees = 0; degrees <= 360; degrees = degrees + 36) {
    push();
    rotate( radians(degrees) );
    rect(100, -15, 100, 30);
    pop();
  }
}

Rotating While Changing Color and Size

Images in Processing Programming
Let's Start with a Favorite Image

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 Your Image into Your Program's assets Area

Step #1: Click on this arrow

Step #2: Click on this arrow and select Create folder

Step #3: Enter assets as the name of the folder and click on Add Folder

If you already have an assets folder, then you can skip steps #2 and #3.

Loading and Drawing an Image

let MyImage;

function setup( )
{
    createCanvas( 800, 800 );
    MyImage = loadImage( "assets/zelda.jpg" );
    background( 200, 200, 200 );
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
}

function draw( )
{
    image( MyImage, 100, 100, 400, 400 );
}
Loading and Drawing an Image

```javascript
let MyImage;

function setup( )
{
    createCanvas( 800, 800 );
    MyImage = loadImage( "assets/zelda.jpg" );
    background( 200, 200, 200 );
    stroke( 0, 0, 0 );
    fill( 255, 50, 50 );
}

function draw( )
{
    image( MyImage, 100, 100, 400, 400 );
}
```

What Happens if You Ask For a Different Aspect Ratio?

```javascript
function draw( )
{
    image( MyImage, 100, 100, 400, 200 );
}
```

Translating an Image

```javascript
function draw( )
{
    for( let i = 0 ; i < 6 ; i = i + 1 )
    {
        push( );
        translate( i*100, i*100 );
        image( MyImage, 0, 0, 200, 200 );
        pop( );
    }
}
```

Notice how transforming images works just like transforming rectangles does!

Rotating an Image

```javascript
function draw( )
{
    for( let i = 0 ; i < 6 ; i = i + 1 )
    {
        push( );
        1. translate( 300, 300 );
        2. rotate( radians(90) );
        image( MyImage, 0, 0, 200, 200 );
        pop( );
    }
}
```

Notice how transforming images works just like transforming rectangles does!
Advanced Polar Patterns

Some Other Polar Patterns

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

Note: \( x^3 = x \times x \times x \)

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


Imitating a Spirograph™

Looks like an Oreo, but it’s not. ©
let BigR  = 200.;
let SmallR = 150.;
let D      = 120.;

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

function
draw( )
{
    translate( 400, 400 );
    beginShape( );
    for( let t = 0; t <= 10*360; t = t + 2 )
    {
        let bigTheta = radians( t );
        let smallTheta = - ( BigR / SmallR ) * bigTheta;
        let x = ( BigR - SmallR ) * cos( bigTheta )  +  D * cos( smallTheta );
        let y = ( BigR - SmallR ) * sin( bigTheta )   +  D * sin(  smallTheta );
        vertex( x, y );
    }
    endShape( );
}