Why Do We Have These Notes?

Processing has thousands of "buttons" you can press. These notes are here to show you what certain combinations of buttons do in order to learn them the first time, and to remind you later when you've forgotten.

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. You can find a similar device in Oregon today…
Another Historic Example is Textile Programming

And, of course, there is the ever-fun Player Piano

Computers Eventually Imitated Historic Methods using Punch Cards

The Processing Programming Language

Where to Find Processing

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

Processing includes a collection of spectacular example programs
Running Processing

Click on the Editor link, or navigate to: https://editor.p5js.org/
Either way, here's what you will see:

Menu headers
- Run your program
- Stop the program

Program-writing/editing area

Processing message area

Running Processing

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

Don't worry – it will get better 😊

Writing Processing Programs

Introduction to Writing Processing Programs

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”.

$(X=0, Y=200)$

$(X=100, Y=200)$

$(X=200, Y=200)$
This is the “Graph Paper” System for Processing Programs

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

(X=100, Y=200) ΔX=150

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

Colors for Computer Graphics Monitors: Additive Colors (RGB)

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>

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

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

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

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

Green = C + Y
Yellow = R + G
Cyan = B + G
Magenta = R + C + M
Yellow = B + G
Cyan = G + C + Y
Magenta = R + M + Y

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

You must add code to the \texttt{setup()} function. Processing calls this \textit{once} when your program starts.

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

Running Your Processing Programs

Click here to run your program

Some Functions to use when Writing Processing Programs

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.
Variables – using symbols instead of just numbers

We can use variables to create relationships.

```javascript
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.

Arithmetic operations in programming are:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Grouping (

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();
```

Drawing Lines and Polygons

Rectangles are Fun, but Arbitrary Lines and Polygons are Funner

```javascript
function draw() {
  beginShape();
  vertex(100, 100);
  vertex(100, 400);
  vertex(200, 400);
  vertex(300, 300);
  vertex(400, 50);
  endShape();
}
```

Drawing One Rectangle is Pretty Straightforward

```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);
  }
}
```

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

for-loops to the Rescue!

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 θ.

One of the things we notice is that each angle θ has a unique X and Y that goes with it. The X and Y are different for each θ.
First, We Need to Understand Something about Angles

Centuries ago, people developed tables of those X and Y values as functions of \( \theta \).

They called the X values \textit{cosines} and the Y values \textit{sines}.

These are abbreviated \( \cos \) and \( \sin \).

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

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.

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

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

Function

```java
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( );
}
```

\( \text{numsegs} \) is the number of line segments making up the circumference of the circle.

\( \text{numsegs}=36 \) gives a nice circle.

5 gives a pentagon.

8 gives an octagon.

4 gives a square. Etc.

Why \( 2.*\text{PI} (= 2\pi) \)?

We commonly measure angles in \textit{degrees}, but scientists, engineers, and computers like to measure angles in something else called \textit{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:

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

```cpp
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...

```cpp
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();
}
```

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

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:

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

More Sophisticated Relationships:
The `map()` function

```cpp
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);
}
```
### More Sophisticated Relationships: The `map()` function

```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 ) );
    green = 3 * green / 4;
    fill( red, green, 0 );
    rect( x, y, 150, 50 );
  }
}
```

The `map()` function can also do blending. Interpolate one forward and the other one backwards.

### Drawing Circles and Other Regular Polygons, II

```javascript
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( );
}
```

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

```javascript
function draw() {
  strokeWeight( 5 );
  noFill( );
  Spiral( 300, 300, 20, 200, 1000, 10 );
}
```

We Can Also Use This Same Idea to Arrange Things in a Circle

```javascript
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

Setting the radius as a function of the angle

```
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();
}
```

200 is the radius of the circle the shape fits in

Setting the radius as a function of the angle

```
function 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!

```
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);
  }
}

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:
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 );
    }
}

Randomness
Or, also use it to pick colors:

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( );
}

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

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

Saving Your Processing Program and Getting It Back Later

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.

Saving Your Processing Program and Getting It Back Later

Then click File → Save

Getting Your Processing Programs Back Later

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

```cpp
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

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

```cpp
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

```cpp
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 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 );
    }
}
```

Reacting to the Mouse and Keyboard: Creating Your Own Paint Program

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

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

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;
            default:
                fill( 100, 100, 100 );
        }
    }
    if( mouseIsPressed ) {
        ellipse( mouseX, mouseY, 50, 50 );
    }
}
```

The `isKeyPressed` and `key` Variables

```cpp
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;
            default:
                fill( 100, 100, 100 );
        }
    }
    if( mouseIsPressed ) {
        ellipse( mouseX, mouseY, 50, 50 );
    }
}
```
```javascript
if( keyIsPressed )
{
  if( key == CODED )
  {
    switch(keyCode)
    {
      case UP: // up-arrow
        ...
        break;
    }
  }
}
```

What if you want to read the Special Keys?

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

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

Let's Use Our Rectangle Object as an Example of Transformations

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

It is Often Nice to Transform Entire Objects at Once

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

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

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, like this:

\[
\text{rad} = \text{radians( deg )};
\]

Use it!

There is also a `shearY` transformation function.

Transformations Accumulate!

is the same as:

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

You Can Save and Restore Transformations

```
function draw() {
  save();
  shearX( radians(45.) );
  rect(0, 0, 200, 100);
  restore();
}
```
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?

```javascript
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

```javascript
function draw() {
  translate(200, 300);
  for (let degrees = 0; degrees <= 360; degrees = degrees + 10) {
    push();
    let blue = map(degrees, 0, 360, 255, 0);
    fill(0, 255, blue);
    rotate(radians(degrees));
    let xsize = map(degrees, 0, 360, 100, 10);
    let ysize = map(degrees, 0, 360, 30, 5);
    rect(100, -15, xsize, ysize);
    pop();
  }
}
```

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 );
}

What Happens if You Ask For a Different Aspect Ratio?

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( );
        translate( 300, 300 );
        rotate( radians(i*60) );
        image( MyImage, 0, 0, 200, 200 );
        pop( );
    }
}
```

Notice how transforming images works just like transforming rectangles does!

Advanced Polar Patterns

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

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


Some Other Polar Patterns

```
r = \sin \left( \frac{5\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();
}