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.

http://xkcd.com

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 easily 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:

Go here to start using Processing

These are good links to check out:

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

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

Don’t worry – it will get better😊

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

\[(\Delta X = 150, \Delta Y = 50)\]

(\(X = 100, Y = 200\))

(\(X = 0, Y = 0\))
This is the "Graph Paper" for Processing Programs:

- **(X=0, Y=0)**
- **(X=width - 1, Y=0)**
- **(X=100, Y=200)**
- **ΔX=150**
- **ΔY=50**
- **(X=width - 1, Y=height - 1)**
- **(X=0, Y=height - 1)**

Colors are formed with combinations of red, green, and blue. The smallest number you can use is 0 for each. The largest number you can use is 255 for each:

- **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

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

- **Y (=R+G)**
- **C (=B+G)**
- **M (=R+B)**
- **K (=C+M)**

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

Colors for Computer Graphics Monitors:
Additive Colors (RGB):

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

Subtractive Colors (CMYK):
Writing a Processing Program – Try This!

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.

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

```
function draw() {
  rect(100, 200, 150, 50);
}
```

Running Your Processing Programs

Click here to run your program

Some Functions to use when Writing Processing Programs

```
1 function
2  setup() {  
3    createCanvas(800, 600);
4    background(200, 200, 255);
5    stroke(0, 0, 0);
6    fill(255, 50, 50);
7  }
8  
9 function
10 draw() {  
11    rect(100, 200, 150, 50);
12  }
```

Enjoying the Output of Your First Processing Program

Don't worry – it will get better 😊

Variables

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

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 = 200;
    rect(x, y, 150, 50);
}
```
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 Good, but Arbitrary Lines and Polygons are Fun Too

Easy – just list the coordinates:

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

Rectangles are Good, but Arbitrary Lines and Polygons are Nice too

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

Drawing Lines and Polygons

Drawing One Rectangle is Pretty Straightforward

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

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

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

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

for-loops to the Rescue!

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.

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

The Oregon State

University
First, We Need to Understand Something about Angles

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.

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

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

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.

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

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

Cosines and Sines are Really Ratios

So, if we know the circle Radius, and we march through a series of θ 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 Function, So We Add Our Own to the End of the Program

```pseudocode
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( );
endShape();
```

Why 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 2π (~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:

```pseudocode
let rad = radians( deg );
let deg = degrees( rad );
```
Circle, Pentagon, Octagon!

```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();
}
```

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

```cpp
function draw() {
    fill(255, 50, 50);
    Shape2(200, 200, 150, 75, 36);
    fill(50, 255, 50);
    Shape2(300, 300, 150, 75, 5);
    fill(50, 50, 255);
    Shape2(400, 400, 150, 75, 8);
}
```

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
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);
    }
}
```
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

All-Green morphs into All-Red

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 = 0.3;
  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 ...

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 = 0.3;
  for( let i = 0; i < numobjects; i = i + 1 )
  {
    let x = xc + r * cos( ang );
    let y = yc + r * 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

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

Randomness

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);
    }
}

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:

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);
    }
}
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);
  }
}

Or, also use it to pick colors:

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);
}

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.
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.

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, 200, 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( )!

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

You Get a Different Garden Every Time You Run the Program!
if-statements

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

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

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

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

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 Lots of Alternatives -- a Better Way

```java
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

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

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

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

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

```java
function draw( )
{
    if( mouseIsPressed )
    {
        ellipse( mouseX, mouseY, 50, 50 );
    }
}
```

The `isKeyPressed` and `key` Variables

```java
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 );
}
```
What if you want to read the Special Keys?

If keys are pressed:

```
if ( keyIsPressed ) {
  if ( key === CODED ) {
    switch( keyCode ) {
    case UP: // up-arrow
      // handle UP
      break;
    case DOWN: // down-arrow
      // handle DOWN
      break;
    case LEFT: // left-arrow
      // handle LEFT
      break;
    case RIGHT: // right-arrow
      // handle RIGHT
      break;
    case ESC:
      // handle ESC
      break;
    case DELETE:
      // handle DELETE
      break;
    case BACKSPACE:
      // handle BACKSPACE
      break;
    case TAB:
      // handle TAB
      break;
    case ENTER:
      // handle ENTER
      break;
    case RETURN:
      // handle RETURN
      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’

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

Rotations and Scaling Happen Around the Origin

“The word ‘translate’ means to ‘move around’

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

### Transformations

Transformations are operations that change the state of geometric objects. Common transformations include translation, rotation, and scaling. These transformations are often used to manipulate objects on a 2D or 3D canvas.

#### Translation

Translation moves an object from one position to another. This is often done using the `translate()` function in JavaScript. For example:

```javascript
translate(x, y);
```

#### Rotation

Rotation involves turning an object around a point, typically the origin (0, 0). The `rotate()` function is used for this purpose:

```javascript
rotate(angle);
```

#### Scaling

Scaling changes the size of an object. The `scale()` function is used to scale objects:

```javascript
scale(x, y);
```

These transformations are fundamental in computer graphics and are used extensively in software that deals with visual representation and animation.
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

Transformations Accumulate!

Transformation Order Matters!

You Can Save and Restore Transformations
function draw() {
    translate(200, 300);
    for(let degrees = 0; degrees <= 360; degrees = degrees + 36) {
        push();
        rotate(radians(degrees));
        rect(0, 0, 100, 30);
        pop();
    }
}

What's the Difference?

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

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 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 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
Step #4: Hover over the word assets and then click on this arrow
Step #5: Click on Upload file
Step #6: Drag your image file into this window or click on this window to browse to the image file

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?

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, 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

\[
\begin{align*}
3 \sin \sin 2 \theta & = \frac{5\theta}{2} \\
8 \sin \frac{8\theta}{5}
\end{align*}
\]


Some Other Polar Patterns

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();
}