What is Processing?

“Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.”

Getting Processing

Go to:  
http://processing.org  
and click on “Download Processing”

Getting Processing

Here you get to choose your operating system: Linux, Windows, or Mac

Running Processing

Click on this desktop icon

Running Processing

Menu headers  
Run program  
Stop program  
Start a new program  
Open an existing program  
Save this program  
Export this program  
Program-writing/editing area  
Processing message area
Processing comes with many great example programs. Check 'em out!

Just like "Run", but without the window borders and title bar

Copy your code to the clipboard, formatted as HTML, suitable for including on a web page

Interactively select and edit a color. Create a movie from a series of images

With Processing, I have good news, and I have bad 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!
### Introduction to Writing Processing Programs

#### Setup
- `void setup()`: This function is called once when the program starts. It is used to set up the size of the drawing window. For example, `size(800, 600);` sets the size to 800x600 pixels.
- `int Size = 30;`: This variable is declared and initialized with a value.
- `void color Background = color(200, 200, 255);`: This sets the background color.

#### Drawing
- `void draw()`: This function is called repeatedly (usually 30 times per second) to draw the image on the screen. It is where you define what gets drawn on the screen.
- `stroke(0, 0, 0);`: This sets the outline color (black in this case).
- `fill(255, 50, 50);`: This sets the fill color (orange in this case).
- `background(BackGround);`: This sets the background color.

#### Mouse Pressed
- `if (mousePressed)` and `if (mouseReleased)`: These are conditional statements to control what happens when the mouse is pressed or released.

#### Functions
- Some functions are defined in the `setup` function, and others are defined inside the `draw` function. Functions are like magical boxes in which some numbers come in and something happens because of them.

#### Comments
- Single-line comments: `//...`-comments that go from the beginning of the line to the end of the line.
- Multi-line comments: `/*...*/` - comments that span multiple lines.

### Colors
- You can use colors in Processing, including predefined colors like `red`, `blue`, `green`, etc., or define your own colors using the `color` function.

### Hexadecimal Numbers
- Hexadecimal numbers are used to represent colors. For example, `fill(182, 216, 46);` sets the fill color to #b6d82e. Hexadecimal numbers are used because they are more compact and easier to handle than RGB values.

### The Color Selector
- You can use the color selector from the Tools menu to pick your own colors.

---

**The Color Selector**

- The color selector provides a visual interface for choosing colors. You can select colors by clicking on the color selector.

- The color selector allows you to choose from predefined colors or create custom colors.

---

**Hexadecimal Numbers**

- Hexadecimal numbers are used to represent colors in Processing.

- Here are some common hexadecimal color values:
  - Red: 0000FF
  - Green: 00FF00
  - Blue: 000000
  - Yellow: FFF000
  - Cyan: 00FFFF
  - Black: 000000

- To convert from RGB to hexadecimal, you can use the following conversion:
  - R (Red) → 0–255
  - G (Green) → 0–255
  - B (Blue) → 0–255

- Hexadecimal values are used because they are more compact and easier to handle than RGB values.
Introduction to Writing Processing Programs

These are "global variables".
All parts of your program know about them.
Change their values at the top,
your entire program will use
the new values.

color   BackGround = color( 200, 200, 255 );
int     Size = 30;

void setup( )
{
    size( 800, 600 );
    background( BackGround );
    }

void draw( )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50  );
    if( mousePressed )
    {
        Magic( mouseX,  mouseY ,  Size );
        rect( mouseX,  mouseY ,  2*Size,  2*Size );
    }
}

These are "global variables".
All parts of your program know about them.
Change their values at the top,
your entire program will use the new values.

color    BackGround =  color( 200, 200, 255 );
int    Size = 30;

void setup(  )
{
    size( 800, 600 );
    background( BackGround );
}

void draw(  )
{
    stroke( 0, 0, 0 );
    fill( 255, 50, 50  );
    if( mousePressed )
    {
        Magic( mouseX,  mouseY ,  Size );
        rec( mouseX,  mouseY ,  2*Size,  2*Size );
    }
}

This is an "if statement".
You use it to have your program make decisions.

Processing’s Coordinate System

If you call "size( 800, 600 ); ", then the screen’s coordinates are:

Yes, I know in school you learn that the origin is in the LL corner. That is correct.
However, sometimes programs get it wrong.
This bugs me too. Deal with it.

A Processing Example

Squares or circles or something else magical
Same size, or random size

Automatic coloring:
UR = red
LR = yellow
LL = green
UL = black

OK, it's a start …
### void Magic( int xc, int yc, int r )

- `numSegs = 20;` determines the number of segments for the magic object.
- `dang = 6.2832 / float( numSegs );` calculates the angle per segment.
- `ang = 0.;` initializes the angle for the first vertex.
- `beginShape();` starts the shape definition.
- `for( int i = 0; i <= numSegs; i = i + 1 )` iterates through each segment.
  - `x = xc + r * cos(ang);` calculates the x-coordinate.
  - `y = yc + r * sin(ang);` calculates the y-coordinate.
  - `vertex( x, y );` defines a vertex.
  - `ang = ang + dang;` increments the angle for the next vertex.
- `endShape();` closes the shape.

This is the magic number. 6.2832 gives a circle. Try something else! 40., 80., and 200. are fun.

Don't worry if you don't understand how this works yet.

### void draw() { }

- `int r = int( map( mouseX, 0, width, 0, 255 ) );` maps the x-coordinate to an integer 0 to 255.
- `int g = int( map( mouseY, 0, height, 0, 255 ) );` maps the y-coordinate to an integer 0 to 255.
- `int b = 0;` sets the blue color to 0.
- `fill( r, g, b );` sets the color for fill.
- `stroke( 0, 0, 0 );` sets the stroke color.
- `Shape = Shape + 1;` increments the shape number.
- `Random = Random + 1;` increments the random number.

This sets the color based on where the mouse is:
- X controls red.
- Y controls green.
- Blue is turned off.

Or, you can set the color yourself. For example, you can ask for orange by saying:

```java
float r = 255, g = 128, b = 0;
```

### void draw() { }

- `if( mousePressed )` checks if the mouse is pressed.
- `Magic( mouseX, mouseY, Size );` calls the magic function.

This is fun enough, but suppose we want to get fancy by adding features:

1. Draw a magic object or a box without stopping the program
2. Make the size larger or smaller
3. Make the size random
4. Clear the screen

### void keyPressed() { }

- `if( Debug )` checks if debug is enabled.
- `println( "key = " + key );` prints the pressed key.
- `println( "keyCode = " + keyCode );` prints the key code.
- `switch( key )` handles different keys.
  - `case 'c':` clears the screen.
    - `rectMode( CORNERS );` sets the rectangle mode.
    - `fill( BackGround );` sets the background color.
    - `rect( 0, 0, width, height );` draws a rectangle.
  - `case 'd':` toggles debug mode.
  - `case 'b':` increases size by 2.
  - `case 'l':` increases size by 2.
  - `case 'r':` toggle random.
  - `case 's':` decreases size by 2.
    - Ensures size is not less than 2.
  - `case '0':` sets shape to 0.
  - `case '1':` sets shape to 1.

We Don't Need These Yet, But I Like To Include Them for Later

```java
void mousePressed() {
  if (Debug)
    println(" mouse button = "+ mouseButton);
}
void mouseMoved() {
  if (Debug)
    println(" mouse has been moved: "+ mouseX+", "+ mouseY);
}
void mouseDragged() {
  if (Debug)
    println(" mouse has been dragged: "+ mouseX+", "+ mouseY);
}
```

More Information About the Processing Program “Skeleton”

Any variables declared up here can be looked at anywhere in the program.

Processing calls your `setup` function first, before it does anything else.

Processing calls your `draw` function over and over to put graphics up on the screen.

Processing calls your `keyPressed` function whenever a keyboard key has been pressed. The exact key that was pressed is stored in a variable called `key`.

Processing calls your `mousePressed` function whenever a mouse button has been pressed. The exact button that was pressed is stored in a variable called `mouseButton`.

Processing calls your `mouseMoved` function whenever the mouse has been moved with all buttons up. The exact mouse location is stored in two variables called `mouseX` and `mouseY`.

Processing calls your `mouseDragged` function whenever the mouse has been moved with a button down. The exact mouse location is stored in two variables called `mouseX` and `mouseY`.

First, You Need to Know About Polar Coordinates

```
r = \sqrt{x^2 + y^2}
\theta = \tan^{-1}\left(\frac{y}{x}\right)
```

x = r \cos \theta

y = r \sin \theta

Can We Imitate a Spirograph?

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

Limaçons (French for “snail”)

-2 ≤ c ≤ 2.5

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


\(c = 1\) is a “cardiod”
```cpp
float Radius(float theta) {
    //return 200.;
    //return 200. * sin(8. * theta / 5.);
    //float s = sin(5. * theta / 2.);
    return -200. * (sin(theta) + s * s * s);
    float C = 1.0;
    return 100. * (1. + C * sin(theta));
}

void setup() {
    size(700, 700);
    smooth();
    background(255, 255, 255);
}

void draw() {
    noFill();
    strokeWeight(1.);
    translate(350, 350);
    stroke(30, 30, 192);
    float radius = Radius(0.);
    float px = radius;
    float py = 0.;
    for(int deg = 0; deg <= 5*360; deg = deg + 1) {
        float rad = radians((float)deg);
        radius = Radius(rad);
        float x = radius * cos(rad);
        float y = radius * sin(rad);
        line(px, py, x, y);
        px = x;
        py = y;
    }
```

Different possibilities for radius as a function of the angle.

This is a programming for-loop. It executes the commands in the curly braces a bunch of times. Its use looks like this:

```
for( ; ; ) {
}
```

Keep looping as long as this equation is true

Do this equation once at the start

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

Draw a line from (px,py) to (x,y)

Says not to do any filling-in