**Processing.js implements Processing, but not all of Java**

Processing.js is compatible with Processing, but is not, and will never be, fully compatible with Java. If your sketch uses functions or classes not defined as part of Processing, they are unlikely to work with Processing.js. Similarly, libraries that are written for Processing, which are written in Java instead of Processing, will most likely not work.

**Processing.js only has two rendering modes**

Processing has many rendering modes to choose from, depending on the desired quality and speed for graphics (e.g., OPENGL, P3D, JAVA2D, etc.). Processing.js uses <canvas> which provides either a 2D drawing context or a 3D context based on WebGL (a version of OpenGL for the web). Therefore, whatever you choose, you will end-up with either the 2D or 3D context.

**Division which is expected to produce an integer might need explicit casting**

There are a class of bugs that arise when converting Processing code to Processing.js that involve integer vs. floating point division. What was straight-up integer division in Processing code, when converted to Processing.js, can sometimes become problematic, as numbers become doubles, and introduce a fractional part. The fix is to explicitly cast any division to an integer that exhibits this behaviour:

1 *// before*

2 **int** g **=** mouseX **/** i**;**

3

4 *// after*

5 **int** g **=** **(int)(**mouseX **/** i**);**

#### Processing.js has to cheat to simulate Processing's synchronous I/O

Processing uses a synchronous I/O model, which means that functions like **loadImage()** take time to execute, and while they are running, nothing else happens: the program waits until **loadImage()** is done before moving on to the next statement. This means that you can count on the value returned by a function like**loadImage()** being usable in the next line of code.

Web browsers don't work like this. The web uses an asynchronous I/O model, which means that functions which load external resources can't make the program wait until they finish. In order to replicate Processing's load\* functions, you have to use a special Processing.js Directive.

The Processing.js Directives are hints to the browser that are written in comments rather than in the Processing code itself. Here's a typical Processing sketch that loads an image synchronously and then draws it:

1 PImage img**;**

2

3 **void** **setup()** **{**

4 img **=** loadImage**(**"picture.jpg"**);**

5 image**(**img**,** 0**,** 0**);**

6 **}**

This code will not work in the browser with Processing.js, because the call to **image()** will happen before the file **picture.jpg** has been downloaded. The fix is to ask Processing.js to download the image before the sketch starts, and cache it--a technique known as preloading. Here is the modified code:

1 */\* @pjs preload="picture.jpg"; \*/*

2 PImage img**;**

3

4 **void** **setup()** **{**

5 img **=** loadImage**(**"picture.jpg"**);**

6 image**(**img**,** 0**,** 0**);**

7 **}**