

## Processing Workshop - Worksheet

*"Processing is an open source programming language and environment for people who want to program images, animation, and interactions. It is used by students, artists, designers, researchers, and hobbyists for learning, prototyping, and production. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook and professional production tool. Processing is an alternative to proprietary software tools in the same domain." -<http://processing.org/>*

### Download

Available for pretty much every operating system, you can download it from the official Processing website here: <http://processing.org/download/index.html>

However, if you don't want to install it you can use an online version at <http://opencode.media.mit.edu/> which will run your processing code with out processing on your own machine.

### Similarities to Java

If you've ever programmed in Java you'll notice that the Processing syntax is pretty much identical to that of Java. This is because Processing is mainly a library that can be used by Java. This means that you can mix and match Java code without much hassle, all you need to do is ensure that you are referring to the intended function (some Processing functions are named the same thing as common Java functions) and everything will work well.

The Processing libraries are mostly much easier to use than the standard Java libraries, which makes what we're doing a relative breeze.

### The Basics

As Processing is designed to be used to create graphics, I'll run through its really basic graphical features. Below is a really simple Processing app that I'll go through to give you a good idea how it works.

```
void setup() {
  size(400, 400);
  stroke(255, 0, 0);
  background(192, 64, 0);
}

void draw() {
  line(200, 200, mouseX, mouseY);
}
```

- The `setup()` and `draw()` functions are key functions in every Processing app. `setup()`, as the name would suggest, creates the canvas that you work with, and `draw()` draws upon it.
- `setup()` :
  - `size(width, height)` sets the size of the window.
  - `stroke(r, g, b)` sets the colour of strokes (lines) drawn from now on. If you want to use hex colour codes, or include alpha values you can do that too ([http://processing.org/reference/stroke\\_.html](http://processing.org/reference/stroke_.html)).
  - `background(r, g, b)` set the colour of the background. You can set the colour in the same ways as with `stroke()`, or use images ([http://processing.org/reference/background\\_.html](http://processing.org/reference/background_.html)).
- `draw()` :
  - `line(x1, y1, x2, y2)` draws a 2d line. In the case above it draws a line from the centre (200, 200) to the mouse pointer.

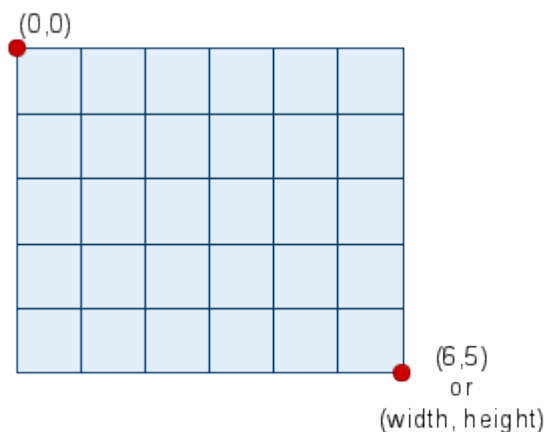
If you run the code, you'll see that the line is being redrawn, but not being removed afterwards. You can get round that simply by calling for the background to be redrawn before the line, as shown in the modified below.

```
void draw() {
  background(192, 64, 0);
  line(200, 200, mouseX, mouseY);
}
```

## Coordinates & Drawing Functions

Processing uses a simple coordinate system, with the top-left being (0,0) and every other point being referenced to that origin.

To help with positioning, Processing provides `width` and `height` variables that return the value that you entered in the initial `size(width, height)` function, this makes it really easy to ensure your shape stays where it should even if the size of the window is changed.



For example, the following code,

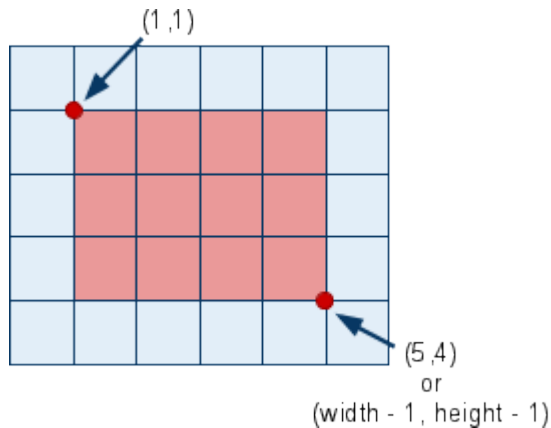
```
void setup() {
  size(60, 40);
  stroke(0, 0, 255);
  background(255, 255, 255);
}
```

```

void draw() {
  fill(255, 0, 0);
  rect(10, 10, width - 20, height - 20); // I will explain why we use - 20
  later
}

```

would create a box similar to the one below, with a gap of 10 pixels between itself and every side.



Though it may seem similar to the `line()` function, `rect()` has different meanings for its parameters. Rather than having the starting point and the finishing point determined, it requires that we give it the starting point and the width and height. In the code example above we gave the width the value of `width - 20` because we know this will create the right width to give the rectangle a 10 pixel gap on each side.

## Your Exercise

We have created a program for you called `pigen`, a processing application that works out pi using random numbers and the ratio between the size of a square and the size of the biggest circle that can fit in said square.

## The Theory

The maths is quite simple, if a square is 1 unit wide, the biggest circle that can fit inside has a radius of 0.5 units.

As the area of a circle is  $\pi r^2$  we can insert the radius and find that the area of our circle is  $\pi 0.5^2 = 0.25\pi = \frac{\pi}{4}$ . Now that we know this relationship we can calculate pi using random numbers. To do this we place points randomly inside the box, and calculate whether they are within the circle or not, and we count the number that land inside the circle and the total number of points. By using  $\pi = \frac{4 \times \text{pointsInsideCircle}}{\text{totalPoints}}$  we can calculate pi.

(More info here: <http://math.fullerton.edu/mathews/n2003/MonteCarloPiMod.html>)

## The Code

You can find the code on our wiki at <http://tinkersoc.org/wiki/presentations> along with a 3d version of the pi generator that you can play with.

## For you to do

We have programmed the main functionality of the program, and it's up to you to visualise the calculations.

At the bottom of the `pigenIncomplete` file there are three functions that you can implement:

- `drawBackground();`
- `drawText();`
  - More info on text handling here: [http://processing.org/reference/text\\_.html](http://processing.org/reference/text_.html)
- `drawPoint(float x, float y,color c);`

There is more explanation within the comments in the file, but if you want any help please dont hesitate to ask Chris ([chris@tinkersoc.org](mailto:chris@tinkersoc.org)) or Matt ([matt@tinkersoc.org](mailto:matt@tinkersoc.org)).