

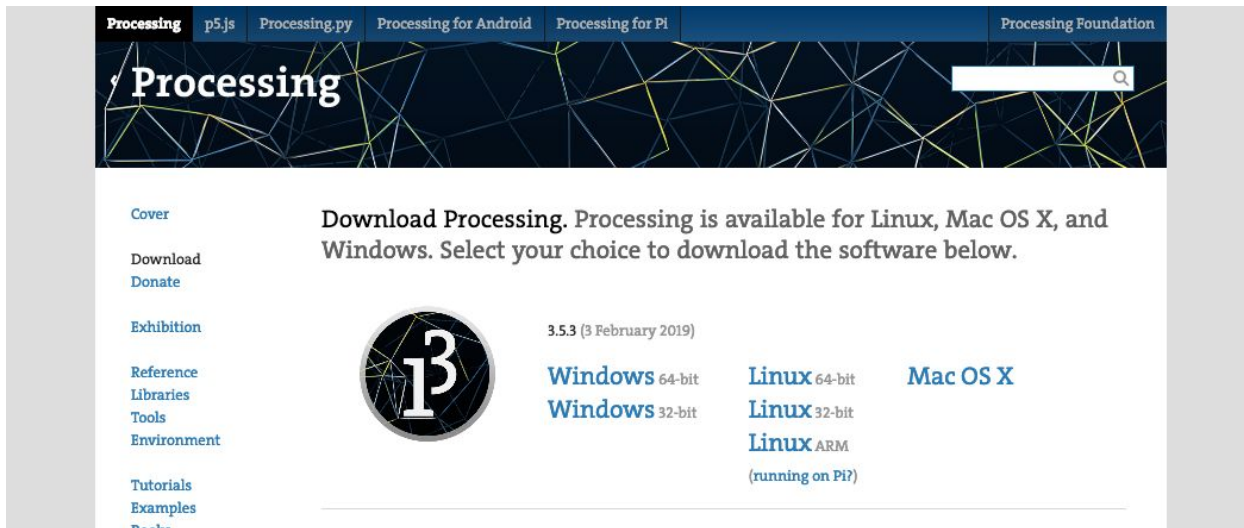
Living Data Workshop #1 Task (weeks 3/4)

What to do before Workshop 1

1. Follow the instructions below to [download and install Processing on your computer](#). If you can't run Processing on your computer, you can try the following:
 - a. Install Processing onto a USB drive, and run it from there
 - b. You can run Processing in a web browser at
 - <https://www.openprocessing.org/> (registration or sign-in is required) or
 - <http://sketchpad.cc/> (no sign-in required, but can be a little more complicated to use).
- Note:** all of the browser-based options use may have some minor differences to the installed version. We recommend installing Processing if you have any questions. Please post a question on Piazza if you're having difficulty.
2. [Watch the Processing tutorial videos](#)
3. [Read and complete the Processing exercise](#)
4. [Create a screenshot of something that you will draw in Processing](#) and upload to Canvas. It is worth 3% of your total unit mark.

Download Processing

Go to the Processing website at <https://processing.org/download/>



Download the Windows 64-bit or Mac OS X version depending on your computer. In either case, this will download a .zip file containing the Processing software and all of it's supporting files.

At the time of writing, the version is 3.5.3, but if it has been updated by the time you read this, just download the current version.

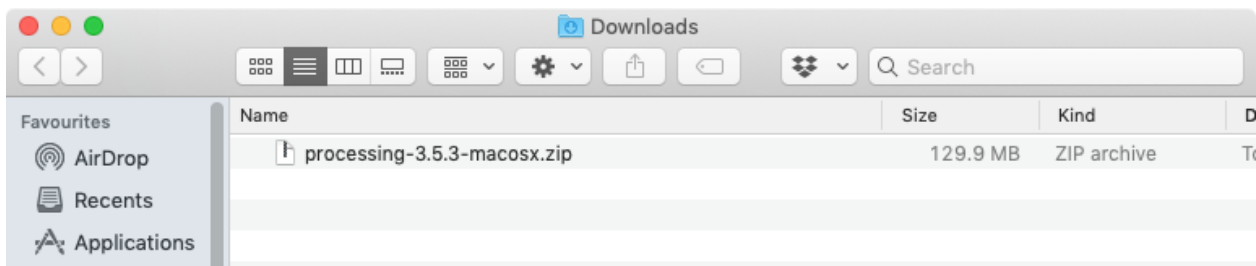
While the file is downloading, it may ask you to make a donation. You don't need to do this in order to use the software.

Install Processing

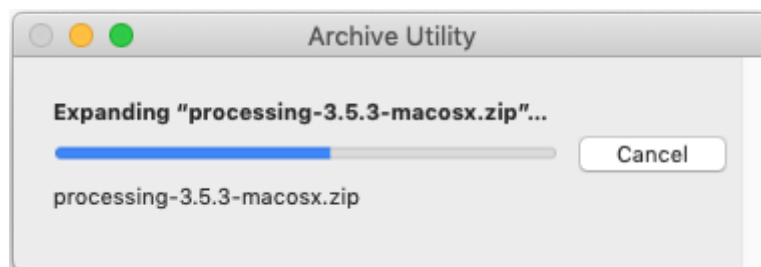
Like all computer software, it needs to be installed before it will work properly. To make sure you don't get stuck, make sure to closely follow these instructions.

Instructions for Mac ([Windows users click here](#))

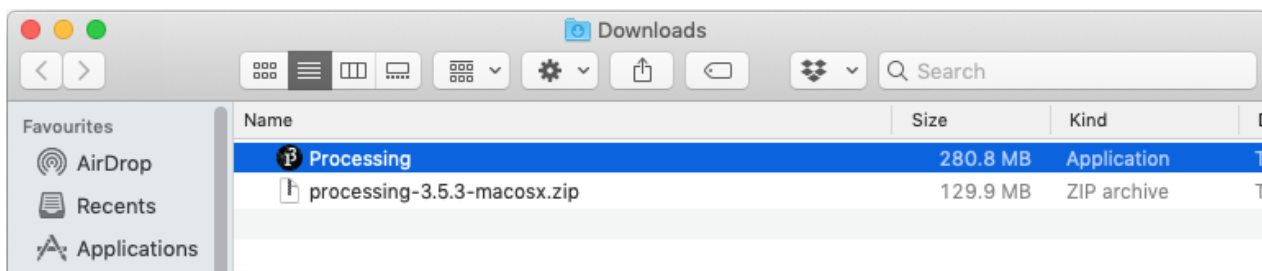
Step 1: Find the **processing-3.5.3-macosx.zip** file - it should be in your **Downloads** folder unless you chose to save it somewhere else.



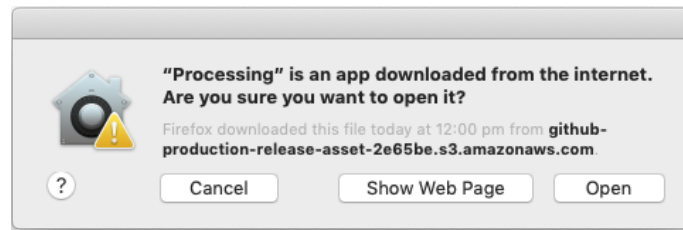
Step 2: Double-click the zip file to expand it:



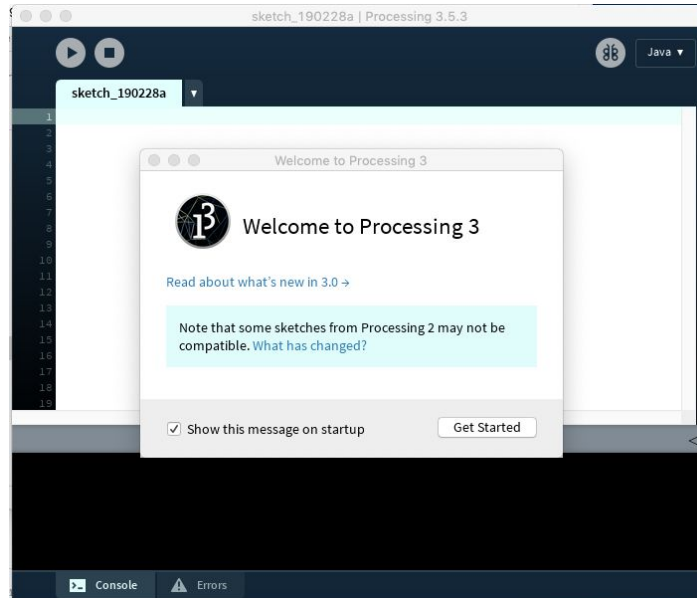
The Processing app file will now be in the same folder as the .zip.



Step 3: Double click the Processing app to launch it. A window may pop up with the following message, if it does, click **“Open”** to continue.



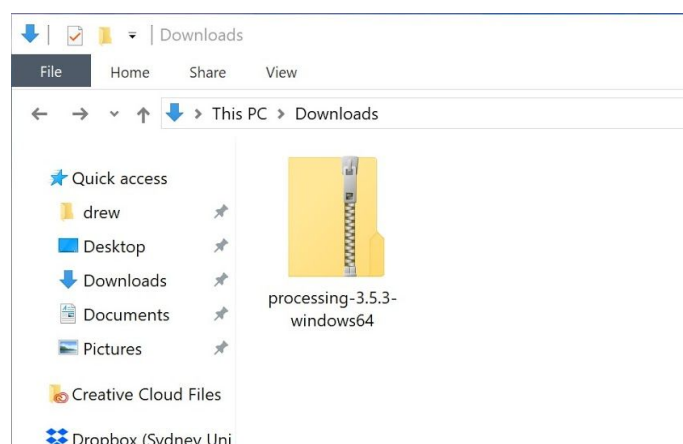
Success: If you see the following screen then you have set up Processing on your computer correctly. (You may want to untick the “Show this message on startup” to avoid getting annoyed every time you open it back up again!)



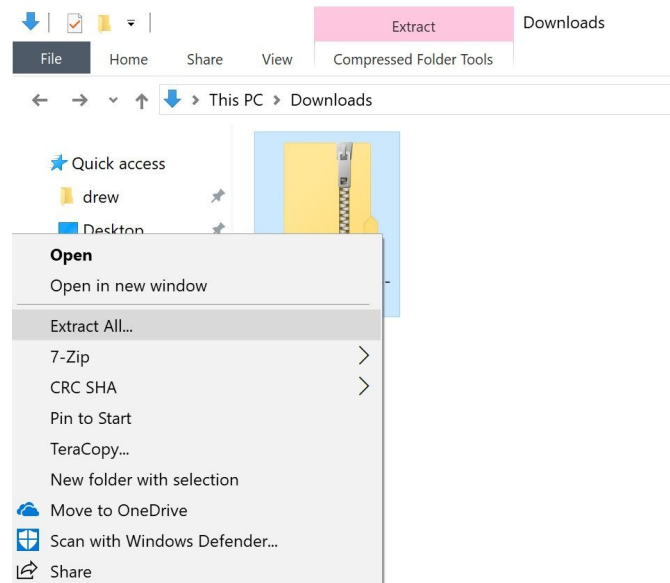
Instructions for Windows

This is one of those times that Windows can make a mess of things. Make sure to follow these steps closely.

Step 1: Find the **processing-3.5.3-windows64.zip** - it should be in your **Downloads** folder unless you chose to save it somewhere else.



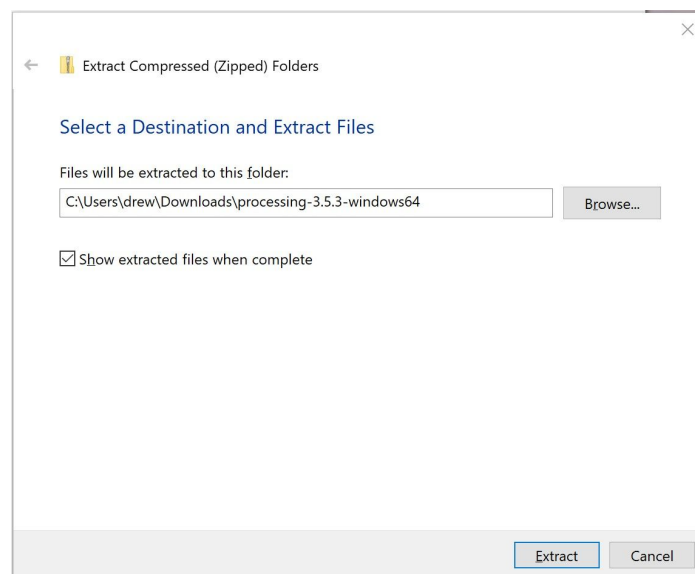
Step 2: Right-click on the zip file and click **“Extract All”**:



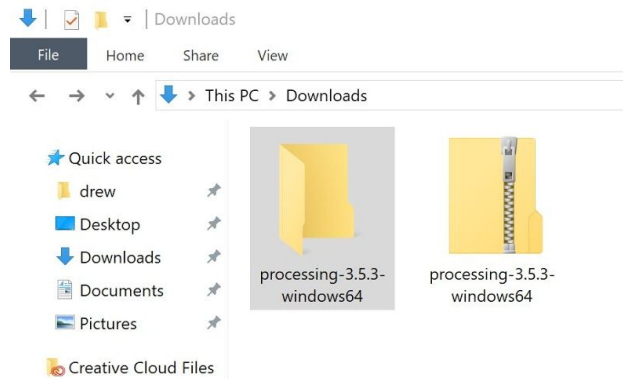
Note: If you double-click the zip file to open it, Windows will show you a preview of what’s in the file. Processing will crash straight away if you try and run it like this. Make sure to use the “Extract All” option first.

Geek Note: If you have another app installed like *WinRAR* or *WinZIP*, use those instead they do this step faster.

Step 3: In the “Extract Compressed (Zipped) Folders” window that will appear, just hit the **“Extract”** button:

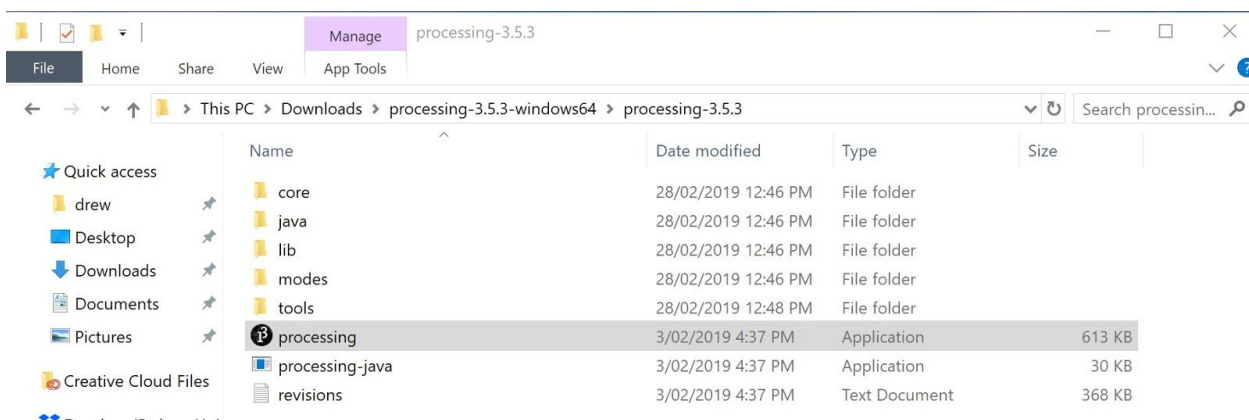


Step 4: Double-click the new “processing-3.5.3-windows64” folder that has just appeared:

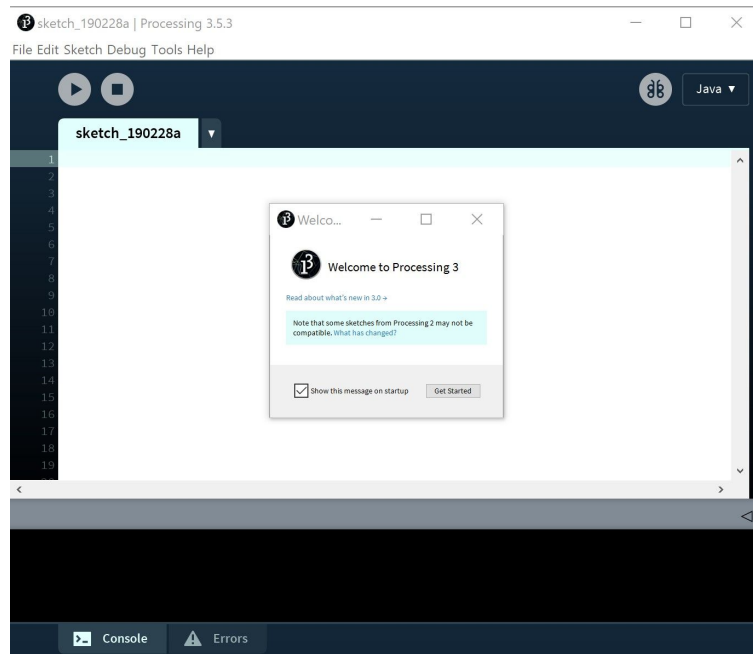


And double-click the “processing-3.5.3” folder that’s inside of it...

Step 5: Double-click the “processing” app file to run it (the one with the P3 icon):



Success: If you see the following screen then you have set up Processing on your computer correctly. (You may want to untick the “Show this message on startup” to avoid getting annoyed every time you open it back up again!)



Watch Processing Tutorial Videos

All aboard the Coding Train! This series of videos by Daniel Shiffman introduces the Processing language and the fundamental concepts of programming.

First, we would like you to watch the following video which will introduce you to the basics of creating pictures with code:

1. [Drawing with Pixels](#)


And then, the following two related videos which introduce more detail about Processing, and explain how computers represent colour:

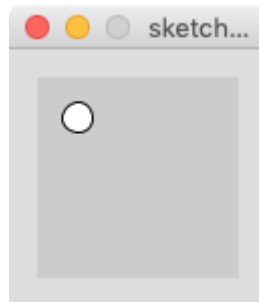
2. [How to use Processing](#)
3. [RGB Color](#)

Processing Recap

Let's recap some of the things we saw in the video about processing. Here we have a function that draws an ellipse. It's a program in a single line!

```
ellipse(20, 20, 15, 15);
```

If we type this into Processing and hit the run button , Processing will open a window with a canvas and draw a small circle near the upper-right corner.



If we change the **parameters** of the function in between the (), and press run again, it will draw in a different place or with a different size. Try it on your computer.

The ellipse function

```
ellipse(x, y, w, h);
```

Draws an ellipse, where:

- x is the position of the center of the circle from left to right (x-axis)
- y is the position from the top of the canvas down (y-axis)
- w is the width of the ellipse
- h is the height of the ellipse

Even experienced programmers sometimes can't remember how every function in a programming language works, so remember that you can look up a function's use in the reference at:

<https://processing.org/reference/>.

Maths and Variables

What if we want to have Processing do some maths for us and use the result of that calculation to determine the position or size of our ellipse? In algebra we learned how to make equations like $y = 3x + 2$. Programming lets us do something similar by assigning values to **variables**.

An analogy for variables could be a table with a bunch of boxes, each with a piece of paper inside:

- **To create a variable called x:** you find an unlabelled box, and write x on the label
- **To give x a value:** you find the box labelled x, and write the number onto the sheet of paper
- **To find out the value of x:** you find the box labelled x and read the number written on the sheet of paper
- **To update the value of x:** you find the box labelled x, read the number, do the calculation you need to do using that number, cross out the old number and put the sheet back into the box

Unlike in algebra, Processing lets us write some instructions that initially might not make sense - replace your `ellipse()` function with this:

```
int x = 1;  
x = x + 1;
```

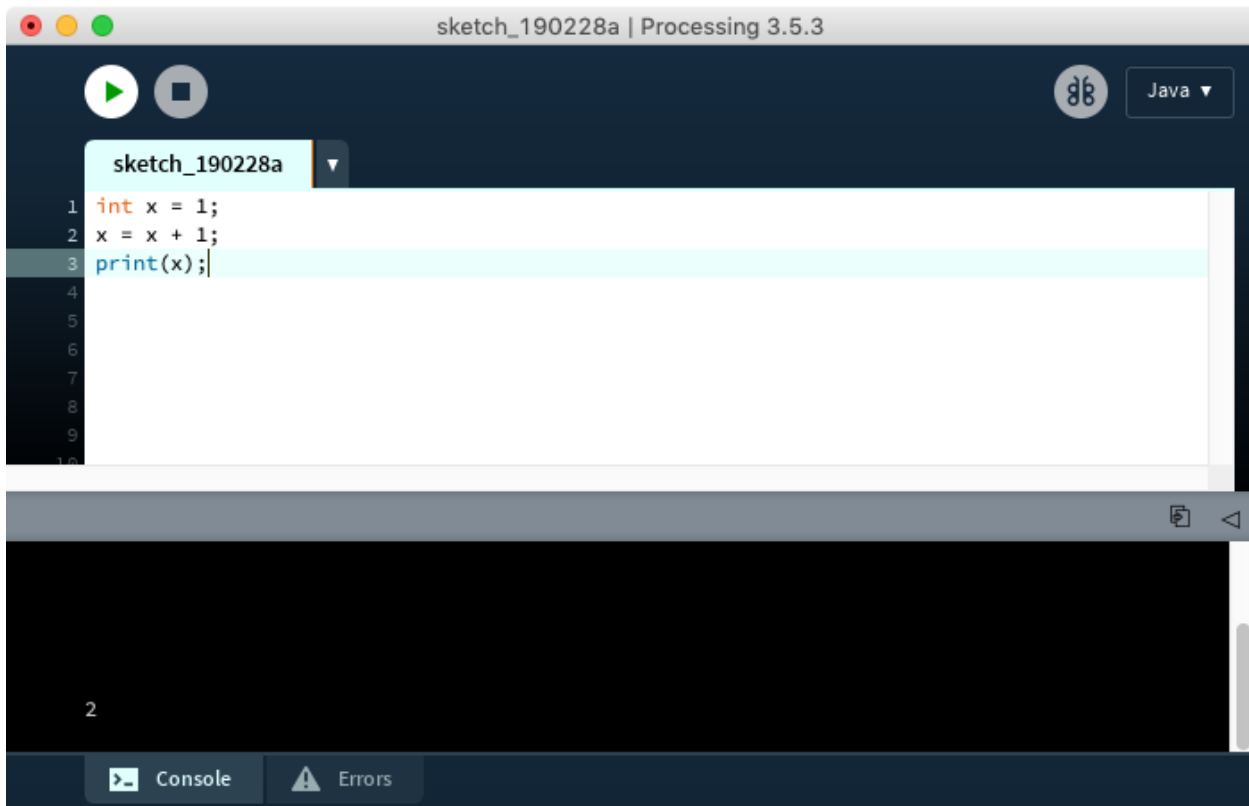
This creates a new variable called x, and assigns it the value of 1. Then, it changes the value of x by adding 1 to whatever it happens to be in x at the moment, then stores the new value back in the x variable again. If you run it, it looks like nothing happens. That's because we aren't doing anything to make the value of x visible - but we can peek at the value stored in a variable with the `println()` function. `println()` is short for "print line" so every time you use this function it will output the value to a new line.

In Processing we have to tell it what type of information our variable will hold. This is represented by `int`. This tells Processing that the variable x will be an **integer** (a whole number with no decimal points). In the next workshop we will look at how to use decimals.

By adding this line after the `x = x + 1;` line we can see what x is equal to:

```
println(x);
```

If we run the code again, in the black area at the bottom of the Processing window, we can see the value of x has become 2.



In Processing we have to represent basic arithmetic a little differently than in mathematics, but if you have ever used the calculator on your computer you might be familiar with this syntax:

Addition: +

Subtraction: -

Multiplication: *

Division: /

Loops

According to Larry Wall (in *Programming Perl*, 2nd Edition, O'Reilly & Associates, 1996), the original author of the Perl programming language, one of the three great virtues of a programmer is *Laziness*.

Laziness: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.

Drawing one circle is easy. Maybe even drawing 5 or 10. But what if we want to draw 50 circles? 1000? Sure we can write LOTS of code to draw lots of circles, but what's the easier way?

Introducing: **Loops**.

Remove the code from the math example function and type in these lines of code:

```
for (int i = 0; i < 5; i++) {  
    int xOffset = i * 5;  
    int yOffset = i * 12;  
    ellipse(xOffset, yOffset, 10, 10);  
}
```

Now we see 5 circles being drawn. Why? Because the for loop tells the program to repeat a block of code.

But first, we see three statements inside the (), separated by semicolons, ;

- `int i=0`

The first statement creates a variable called `i`, with a value of 0, that only exists inside the { } of the loop. We don't *need* to call it 'i', but that's the convention we will stick with for now.

- `i < 5`

The loop will keep repeating as long as the second statement is true (that is, as long as `i < 5`). If we change the 5 to 10 or 50, we will see more circles. We could also use a variable here, instead of a number.

- `i++`

The value of `i` increases by 1 after each loop. If we increase `i` then it will eventually become greater than 5, `i < 5` will become false, then the loop ends.

- If we want to increase by 2 (or any other number) we can change the last statement inside `for()`. Try replacing the `i++` for this and see what happens.
- **Tip:** The following statements will all have the same effect on the value of `i`:

- `i++`
- `i += 1`

■ $i = i + 1$

Adding Colour

So far we can only draw white circles, with a black outline. But since we have these fancy colour screens, we can draw in colour too!

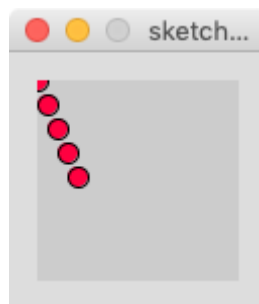
In Processing shapes can have two colours - a fill colour and a stroke colour. These are controlled with the `fill()` and `stroke()` functions. Both functions work in the same way, taking three parameters:

```
fill(r, g, b);
```

- `r` controls the intensity of **red**
- `g` controls the intensity of **green**; and
- `b` controls the intensity of **blue**

If we modify our code to add a fill function, we can add colour.

```
for (int i = 0; i < 5; i++) {  
    int xOffset = i * 5;  
    int yOffset = i * 12;  
    fill(255, 0, 64);  
    ellipse(xOffset, yOffset, 10, 10);  
}
```



Now our ellipses are red. But what about combining everything together? We can use maths to change the colour depending on the value of `i`.

Add a line before the `fill()` line that calculates the intensity of red:

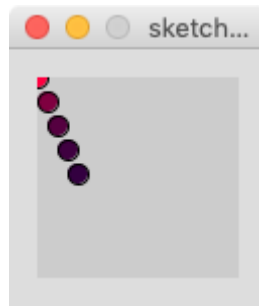
```
int red = 255 / (i+1);
```

This will start with as much red as we can have (i.e. 255), and get less and less red as `i` gets bigger.

We also need to make sure that the fill function uses this new variable, instead of always using the value of 255 for red. Just replace the 255 with our new variable named: `red` :

```
fill(red, 0, 64);
```

Now, the colour changes depending on how many iterations of the loop we have made.



Your code should now look like this:

```
for (int i = 0; i < 5; i++) {  
  int xOffset = i * 5;  
  int yOffset = i * 12;  
  int red = 255 / (i+1);  
  fill(red, 0, 64);  
  ellipse(xOffset, yOffset, 10, 10);  
}
```

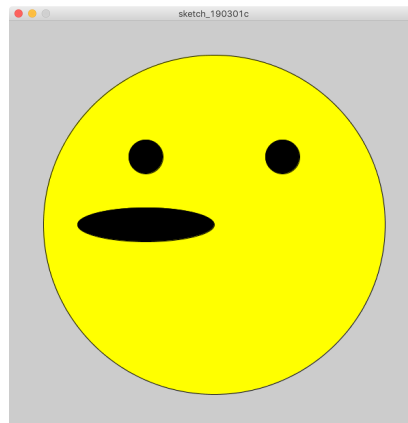
How to get your 3 marks on Canvas

Draw us a picture! Unleash your inner Picasso and use Processing to draw something.

You only need to use lines, shapes and colours (and for-loops, if you feel up to it). We have included some sample code, but you should start from your own idea.

Zero Marks	1 Mark	2 Marks	3 Marks
No submission or Submission does not modify the example code	You have created an image that uses only one basic shape, without use of colour.	You have created an image with two basic shapes, without changing colour	You have created an image with two or more basic shapes and have made colour changes.

Maybe you can draw a self-portrait:



```
// Set the sketch size to 600 pixels wide by 600 pixels high
size(600, 600);

// Set the fill color to yellow, with a black outline.
fill(255, 255, 0);
stroke(0, 0, 0);

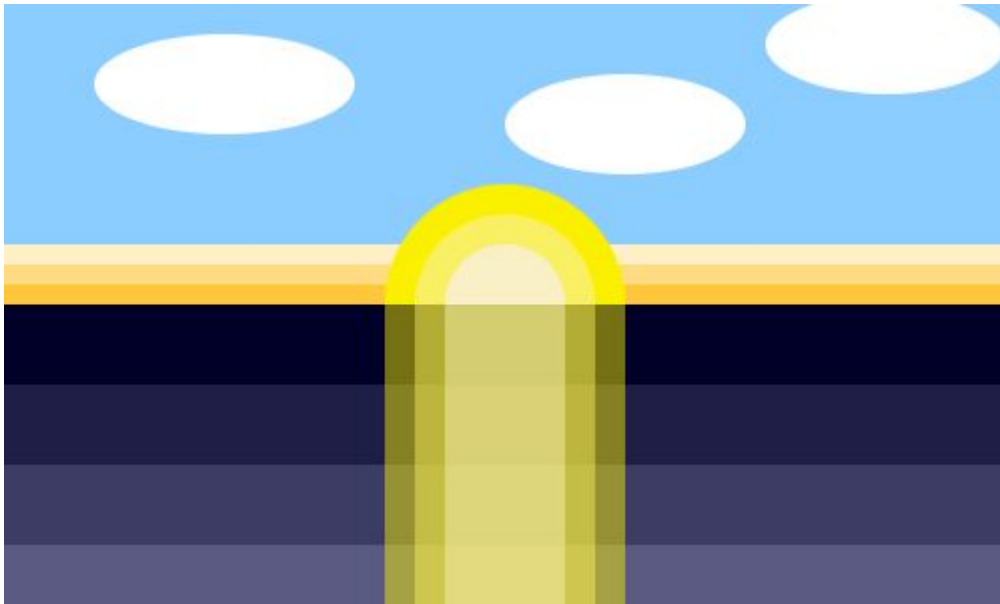
// Draw the head
ellipse( 300, 300, 500, 500 );

// Change the fill colour to black.
fill(0, 0, 0);

// Draw the eyes
ellipse( 200, 200, 50, 50 );
ellipse( 400, 200, 50, 50 );

// Draw the mouth (try and find a better set of coordinates!)
ellipse( 200, 300, 200, 50 );
```


Or a sunset over the ocean:



```
// make the drawing canvas 500px wide by 300px tall
size(500, 300);
// turn off the outlines of shapes we draw
noStroke();

// blue sky for background
background(140, 205, 255);

// draw the yellow on the horizon
for(int i = 0; i < 3; i++) {
  fill(255, 200 + i * 20, 60 + i * 70);
  rect(0, 140 - i * 10, 500, 10);
}

// draw the sun, circles getting more white toward the middle
for (int i = 0; i < 3; i++) {
  fill(250, 240, i * 100);
  ellipse(250, 150, 120 - i * 30, 120 - i * 30);
}

//draw the waves of the ocean
for (int i = 0; i < 4; i++) {
  fill(i * 30, i * 30, 40 + i * 30);
  rect(0, 150 + i * 40, 500, 150 - i * 40);
}

// draw the reflection on the sun in the ocean
for (int i = 0; i < 3; i++) {
  // the 4th number here is the opacity
  fill(250, 240, i * 90, 120);
  rect(250 - (120 - i * 30) / 2, 150, (120 - i * 30), 150);
}

// add some clouds
fill(255, 255, 255);
```

```
ellipse(110, 40, 130, 50);  
ellipse(310, 60, 120, 50);  
ellipse(440, 20, 120, 50);
```