# Living Data Workshop 1

Link to pdf of slideshow: https://bit.ly/2XJTqRl

## In this workshop...

**Processing**

- Recap *The Coding Train* material;
- Recap worksheet material (basic shapes, colour, variables etc.);
- Build example described in the prework from scratch.

## Getting Processing

Download app [http://processing.org/download](http://processing.org/download) (you don’t have to donate)

Follow installation instructions from the Worksheet (take extra care on Windows)

# *The Coding Train* recap

Drawing with Pixels

- Computer graphics coordinate system (0, 0 at top left)
- 2D Primitive shapes (`line`, `ellipse`, `rect`)

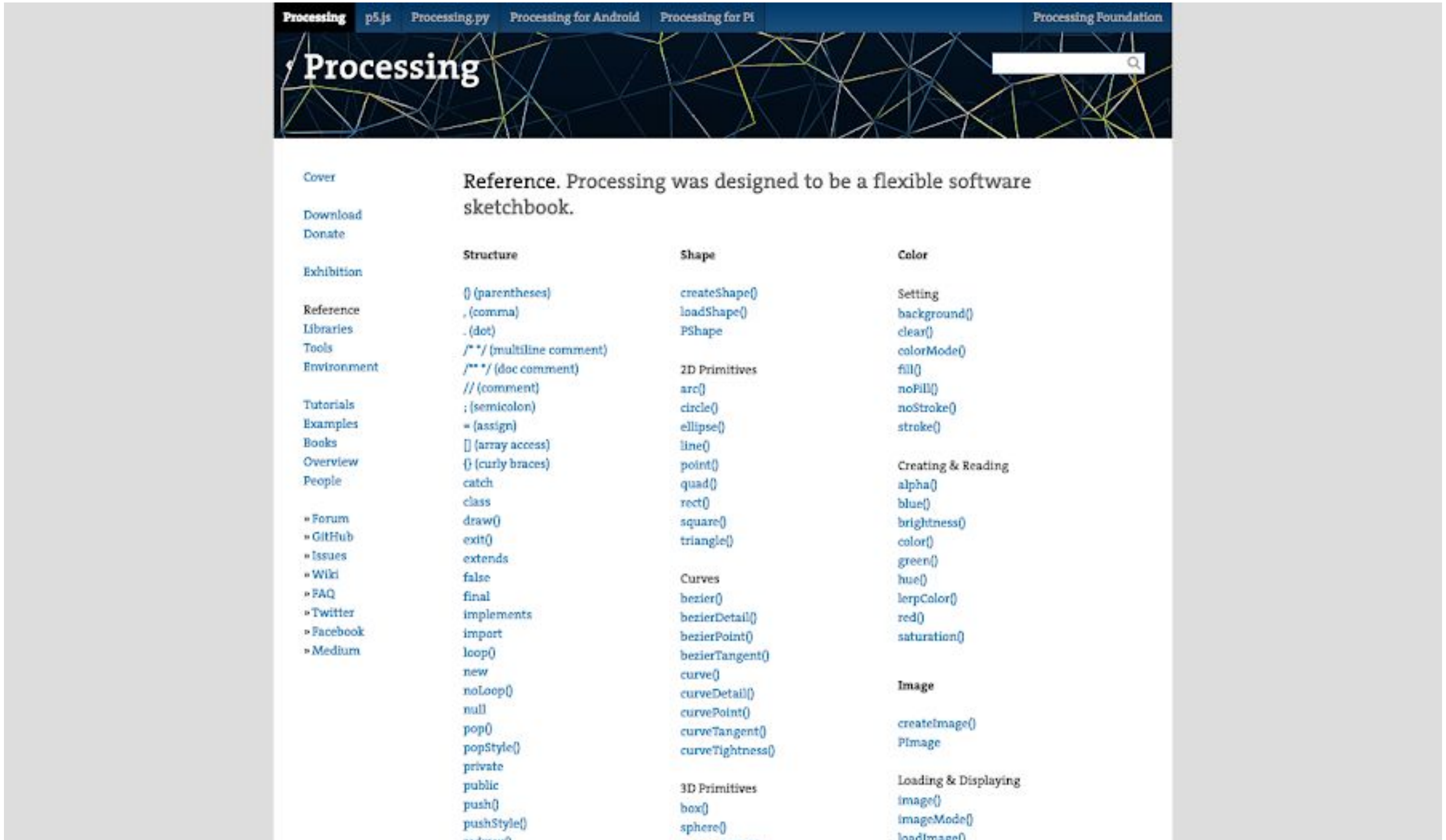
How to Use Processing

- Syntax
- Recovering from errors

RGB Colo(u)r

- Representing colour as an RGB code
- `stroke`
- `fill`

# Processing Reference (https://processing.org/reference/)



Cover

Download
Donate

Exhibition

Reference
Libraries
Tools
Environment

Tutorials
Examples
Books
Overview
People

» Forum
» GitHub
» Issues
» Wiki
» FAQ
» Twitter
» Facebook
» Medium

**Reference.** Processing was designed to be a flexible software sketchbook.

**Structure**

() (parentheses)
, (comma)
. (dot)
/* */ (multiline comment)
/** */ (doc comment)
// (comment)
; (semicolon)
= (assign)
[] (array access)
{} (curly braces)
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
pop()
popStyle()
private
public
push()
pushStyle()

**Shape**

createShape()
loadShape()
PShape

2D Primitives
arc()
circle()
ellipse()
line()
point()
quad()
rect()
square()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()
sphere()

**Color**

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

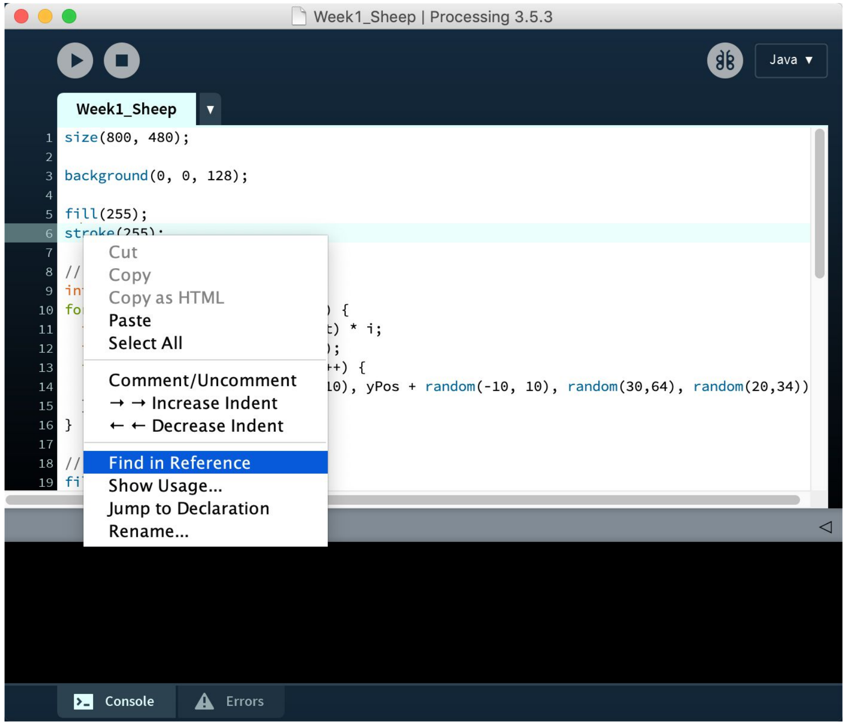
**Image**

createImage()
PImage

Loading & Displaying
image()
imageMode()

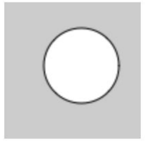
## Reference is also built-in

To look up how a function works, right-click (Ctrl+Click on Mac) on it, then select **“Find in Reference”**



```
size(800, 480);

background(0, 0, 128);

fill(255);
stroke(255);

//
in
fo                                   ) {
                                     t) * i;
                                     );
                                     ++) {
                                     10), yPos + random(-10, 10), random(30,64), random(20,34))

}

//
fi
```

# Ellipse

| | | |
|---|---|---|
| **Name** | ellipse() | |
| **Examples** |  | `ellipse(56, 46, 55, 55);` |
| **Description** | Draws an ellipse (oval) to the screen. An ellipse with equal width and height is a circle. By default, the first two parameters set the location, and the third and fourth parameters set the shape's width and height. The origin may be changed with the `ellipseMode()` function. | |
| **Syntax** | `ellipse(a, b, c, d)` | |
| **Parameters** | a | float: x-coordinate of the ellipse |
| | b | float: y-coordinate of the ellipse |
| | c | float: width of the ellipse by default |
| | d | float: height of the ellipse by default |

```
// Wider than it is tall

ellipse(50, 50, 100, 50)


// Taller than it is wide

ellipse(50, 50, 50, 100)


// Moved to the bottom right

ellipse(250, 250, 50, 50)
```

## Checkpoint 1

Everyone should:

- ✓ Have Processing installed and working
- ✓ Know where to find help documentation
- ✓ Be able to draw a circle!

# Maths

One thing computers are really good at is doing maths - really, really fast.

The representation of basic arithmetic might be a little different than what you’re used to:

| | |
|---|---|
| Addition: + | Subtraction: - |
| Multiplication: * | Division: / |

Check the Reference for more operations.

## Variables

Variables let us store a value into the computer’s memory, so that we can manipulate it or retrieve it later.

| | |
|---|---|
| `int x = 1;` | Create a new variable called `x`, and assign it the value of `1` |
| `x = x + 1;` | Retrieve the value of `x` from memory, add one to it, then store it back into `x` again. (The RHS is run before trying to fill the variable). |
| `println(x);` | Retrieve the value of `x` from memory, and show it on screen. |

## Variables

You can use variables together with other variables:

```
int xPos = (width / 2) * i;
```

Create a new variable called `xPos`, and assign it a value by doing a calculation using two other (previously created variables).

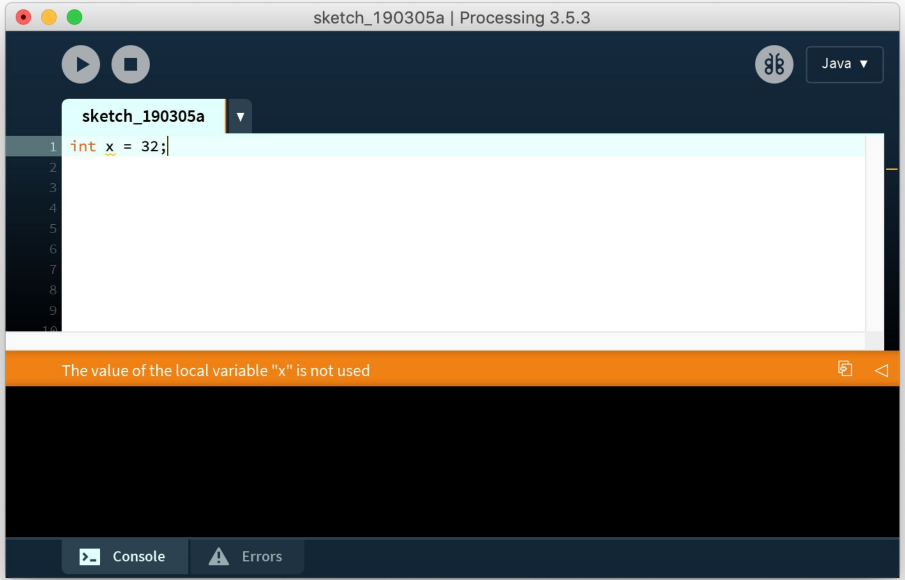
## Variable Gotchas

Sometimes you will see Processing put a squiggly underline beneath a variable in orange. A message at the bottom of the window will say:

**The value of the local variable "x" is not used.**

This isn't an error! Processing is pointing out that you have created a variable, but then don't use it anywhere. You might:

- Still be writing some code - **just keep going**
- Have rewritten some code and no-longer use it - **decide whether to keep the variable or delete it.**



```
int x = 32;
```

The value of the local variable "x" is not used

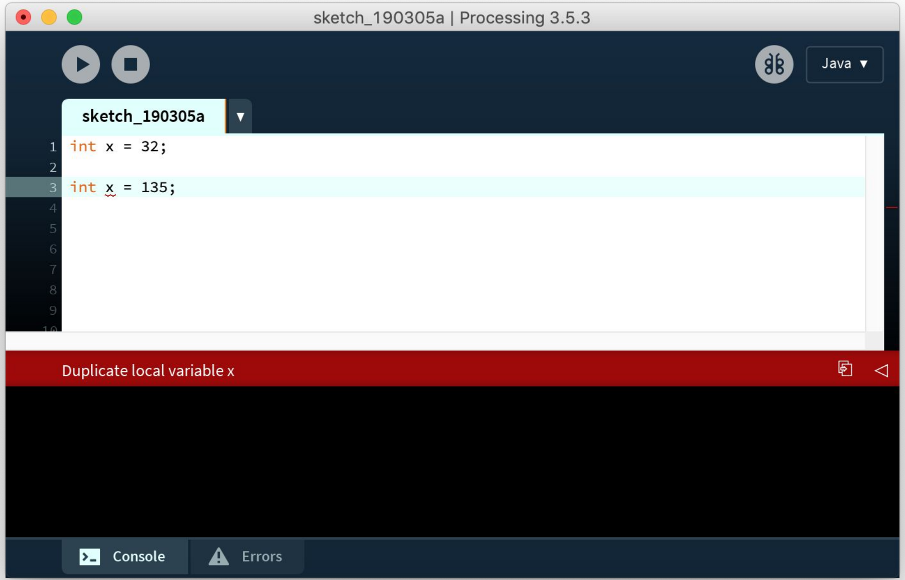
## Variable Gotchas

Sometimes you will see Processing put a squiggly underline beneath a variable in red. A message at the bottom of the window will say:

**Duplicate local variable x**

This ***is*** an error! Processing is pointing out that you have created a variable, and then try to create a second variable with the same name.

- You may have pasted some code twice - **check to see if you have accidentally done this**
- **Decide on a new name for the variable**



```
int x = 32;

int x = 135;
```

```
Duplicate local variable x
```

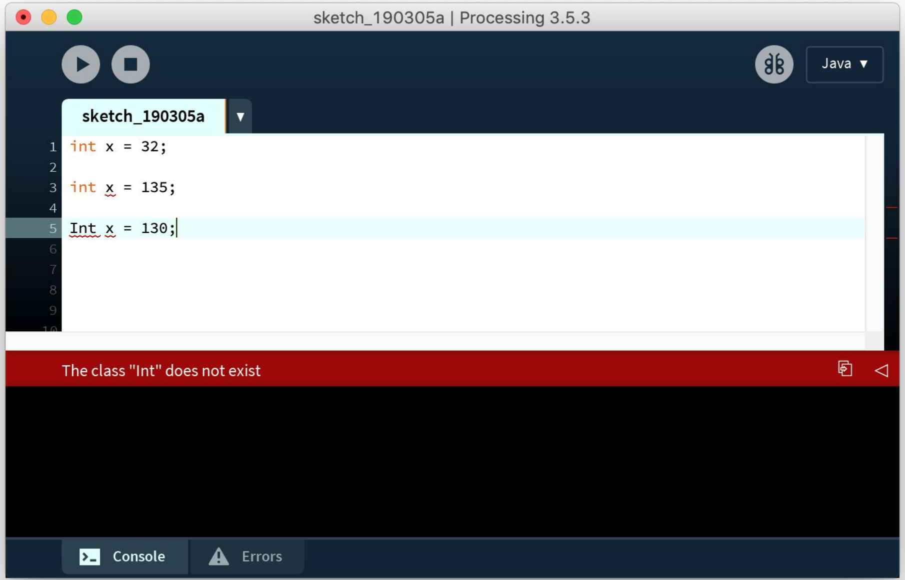
# Variable Gotchas

Sometimes you will see Processing put a squiggly underline beneath the data type and variable in red. A message at the bottom of the window will say:

**The class “Int” does not exist**

This ***is*** an error! Processing is case-sensitive.

- Check the capitalisation on “`int`” - it should always be in lower case.



```
int x = 32;

int x = 135;

Int x = 130;
```

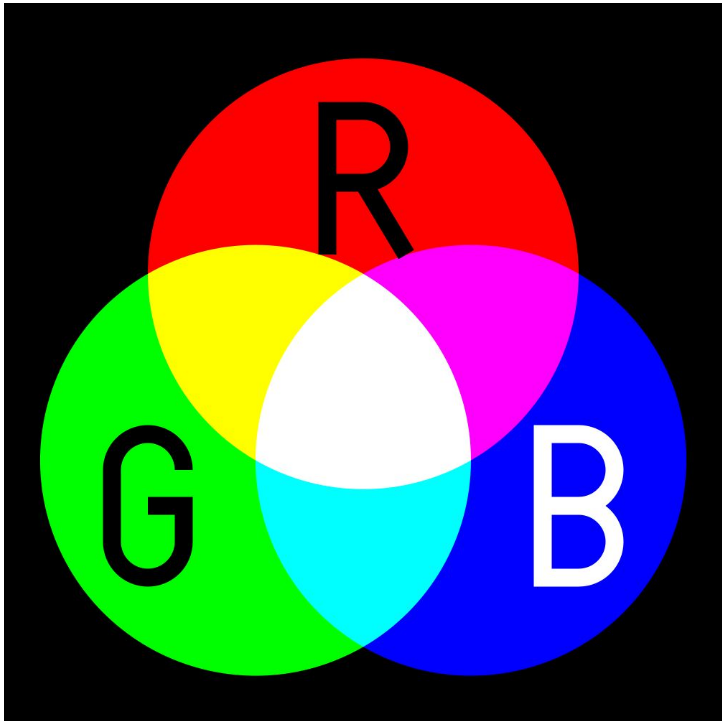
The class "Int" does not exist

# RGB Colour

Computers represent colour as a proportion of **red**, **green** and **blue**.

All other colours can be created by blending **RGB** values.

Each colour is represented as a number from **0** (none of that colour) to **255** (full intensity).

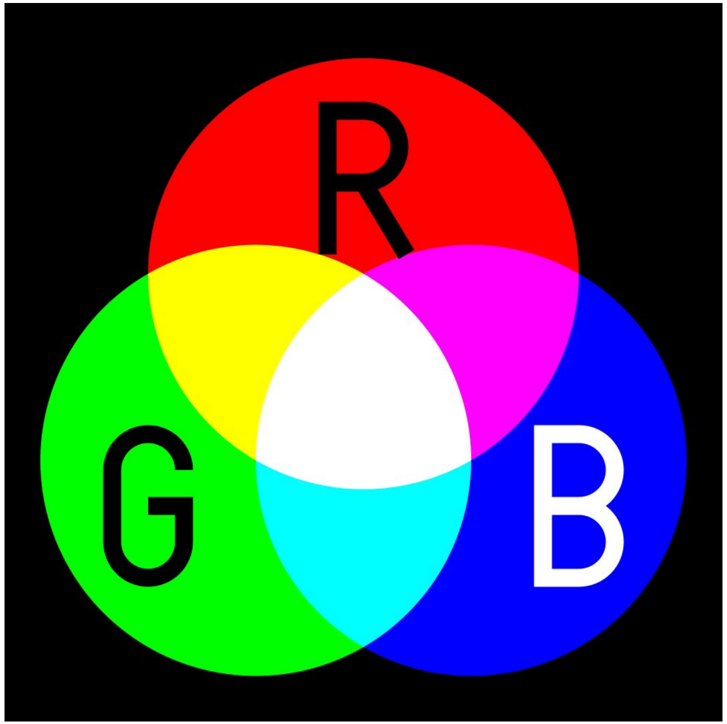


## RGB Colour

Black is no color (0, 0, 0)

Greys are balanced values of RGB colours i.e.
(128, 128, 128)

White is all of the colours (255, 255, 255)

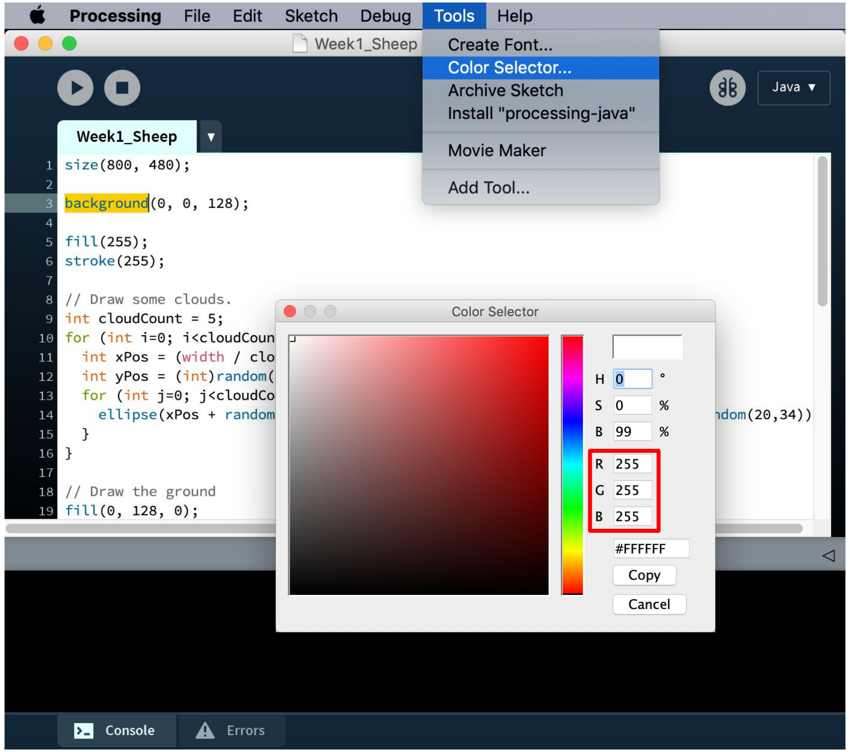


# Colour Picker

Trying to guess a colour is hard. That’s why there are tools to help with this.

In Processing go to the **‘Tools’** menu then choose **‘Color Selector…’**

Once you find the colour you want, read the **R G** and **B** values ready to put into your code.



```
size(800, 480);

background(0, 0, 128);

fill(255);
stroke(255);

// Draw some clouds.
int cloudCount = 5;
for (int i=0; i<cloudCoun
  int xPos = (width / clo
  int yPos = (int)random(
  for (int j=0; j<cloudCo
    ellipse(xPos + random                                   ndom(20,34))
  }
}

// Draw the ground
fill(0, 128, 0);
```

H 0 °
S 0 %
B 99 %
R 255
G 255
B 255
#FFFFFF
Copy
Cancel