[the academy_of_code]

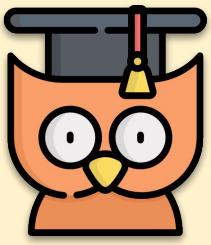
Grade 4
Unit 2

Variables, Flags, and Mapping

Make sure you know all these topics before you start:

- ✔ Using **size()** and coordinates
- ✔ Drawing circles using **ellipse()**
- ✔ Adding colour using **fill()** and **background()**
- ✔ Using void **setup()** and void **draw()**

Lesson 1 - Variables I

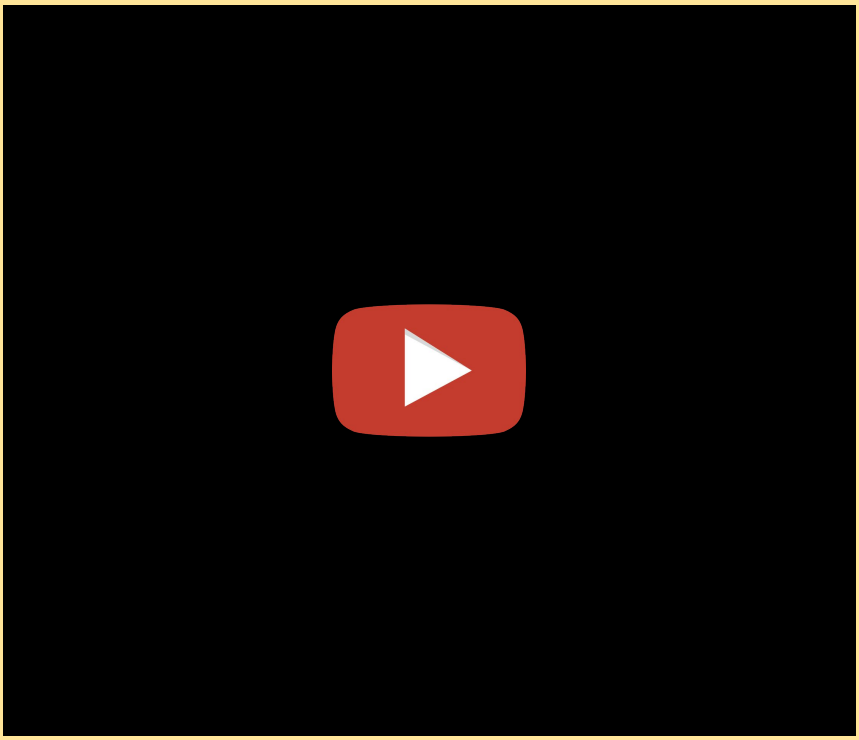


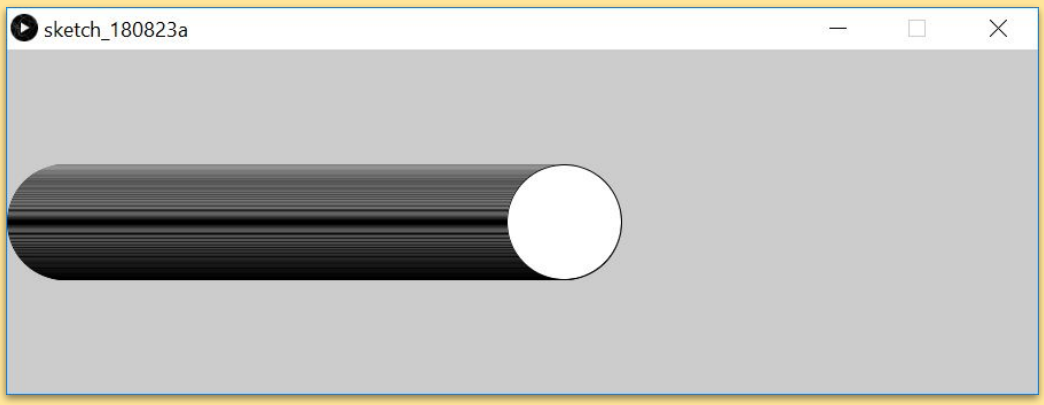
Learning Outcomes:

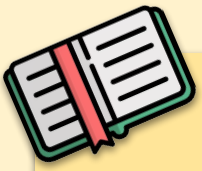
- We will learn about **variables**; what they are and why we use them
- We will use **float** and **int** variables for the first time
- We will make our first animation using Processing

What is a variable?

Before we begin, watch this video explainer on what variables are.

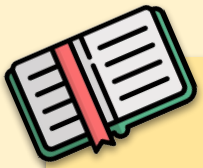






What are variables?

A variable is **data that can be altered while the program is running**. Variables let us store this data in our program, allowing us to create more complicated programs, like games, animations and other applications. There are **different types** of variables for different types of data like numbers, words, colours and many more!



What are ints?

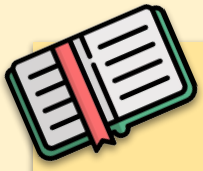
int is short for **“integer”**. **int** variables are used to store **numbers**. They can store any **whole** number from -2,147,483,648 to 2,147,483,647.

What are floats?

float is short for **“floating point number”**. **floats** also store **numbers**, but **unlike ints**, they can store **decimal numbers**. Now we’re gonna write our first program using **floats**, which will also be our first animation.

Did you know?

If your variable goes **over** 2,147,483,647 your program will crash. This is called an **overflow**. And if your variable goes **under** -2,147,483,648 your program will also crash. This is called an **underflow**. If you need to use huge numbers in your program, you might want to use a **long** variable, which can store numbers up to **9,223,372,036,854,775,807**!



Three steps for using variables

1. We need to **declare** our variable and give it a **meaningful name**
2. We must **initialize** our variable by giving it an **initial value**
3. We will **update** our variable by **changing its value**

Let's get coding!

1. Create a **new** processing sketch by clicking **File - New**.

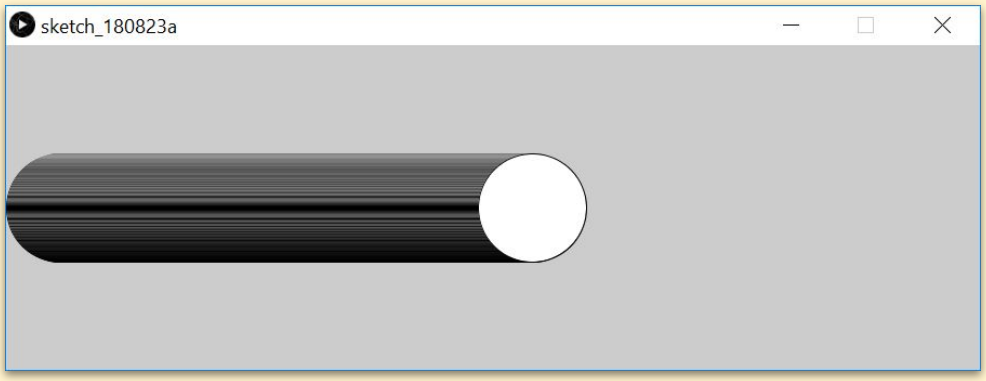
2. At the **top** of your text editor, write this line:
   `float ballX;` **Declares** variable as **"ballX"**.

3. Inside `void setup()`, write this line:
   `ballX = 50;` **Initializes** variable to **50**.

4. Inside `void draw()`, create an ellipse using **ballX** as its x-position and write the following line below it `ballX = ballX + 1;`
   This **updates** the variable, **adding 1** to it **every frame**.

```
float ballX; //declare (give it a name) 2

void setup() {
  size(600, 600);
  ballX = 50; //initialise (give it a starting value) 3
}

void draw() {
  ellipse(ballX, 300, 100, 100);
  ballX = ballX + 1; //update (60 times per second) 4
}
```

If you've done the steps correctly, you should see a circle moving from left to right across the screen.



Now that you're an animation expert, are you ready to step it up?
Try the next few tasks and take your animation to the next level!

Task 1

Change the program so that we can only see **one ball** at a time.

Hint: use **background()**.

Task 2

Change the **speed** of the ball.

Hint: Think about how we are **updating** the variable.

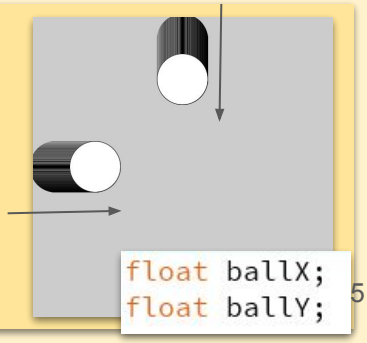
Task 3

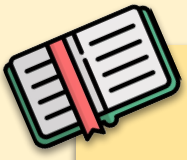
Now make the circle start on the **right** and move to the **left**.

Hint: Think of where the circle starts and how you update it.

Task 4

Create a second circle with its own new variable (maybe *ballY*) and have it moving down the screen at the same time as the original circle.

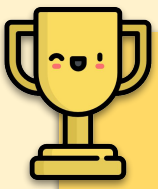




How does our animation work?

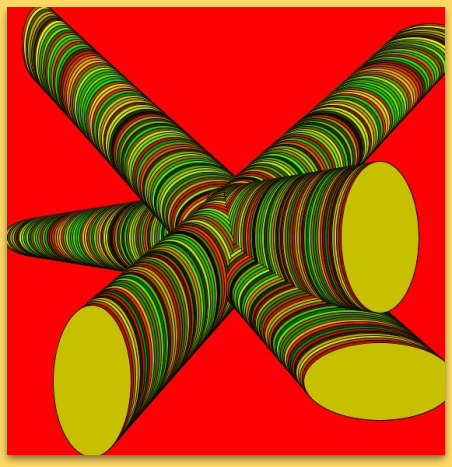
What is happening in this program, is that we are drawing **multiple** circles every second**.** We use **void draw()** in this program because it runs **60 times a second**. So any code that we have inside this block is run 60 times a second. Once the end of the block has been reached, it **loops back** up to the top of the block.

In our program, we add 1 to our variable each time we loop through the code. In the next loop, a **new circle** will be drawn with this **updated position**. This happens again and again, making it look like the circle is **moving across** the screen.



Challenge Task - Grade 4

Create multiple circles/rectangles (at least four) that have multiple variables that change their positions **and** their sizes. Be creative on this using **random()** for example. Use the **Snipping Tool** to take a screenshot of your art.

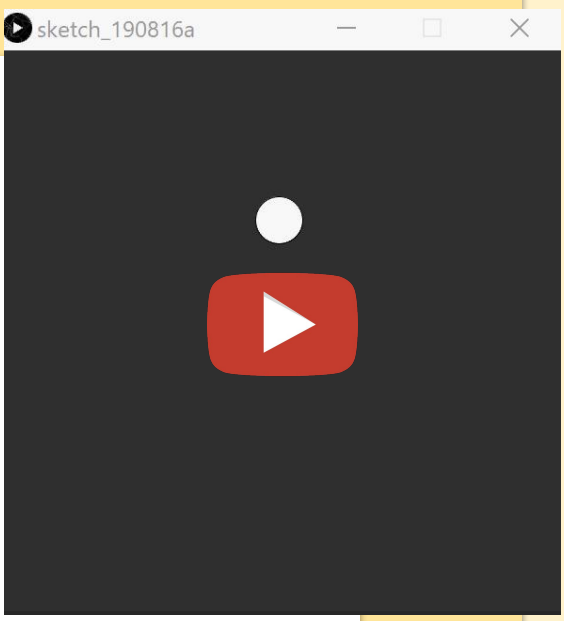


Grade 5 Power Up

Taking the above example:

- use the **random()** function in some way other than in the fill/background or in the size of the ellipse
- Create an ellipse that moves in the opposite direction to the mouse. So when the mouse moves right the ellipse should move left
- *Add an **on-screen button** (Grade 5 Unit 1 Lessons 3) that makes your variable do something.*

Crack the Pseudocode - Using Random() for ‘jitters’




```
fl Declare ‘circleX’ X;

void setup() {
  Pick a size canvas
  Initialise ‘circleX’ as the width divided by 2
}

void draw() {
  b Black background
  f White fill
  ellipse( (circleX, ____ , ___ , ___ );

  circleX = circleX + random(-5, 5);
}
```

Try the above example which makes a shaking circle. it’s position moves in a jittery fashion (like Rob after a pre-workout) at random numbers **between -5 and +5.**

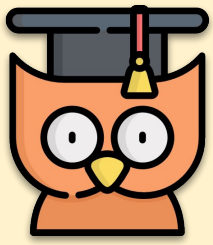
Grade 5 Power Up

Taking the above example:

- create a sketch that has the ellipse move randomly up and down, as well as side to side
- Create a second ellipse that moves randomly, but in a different way to the first ellipse

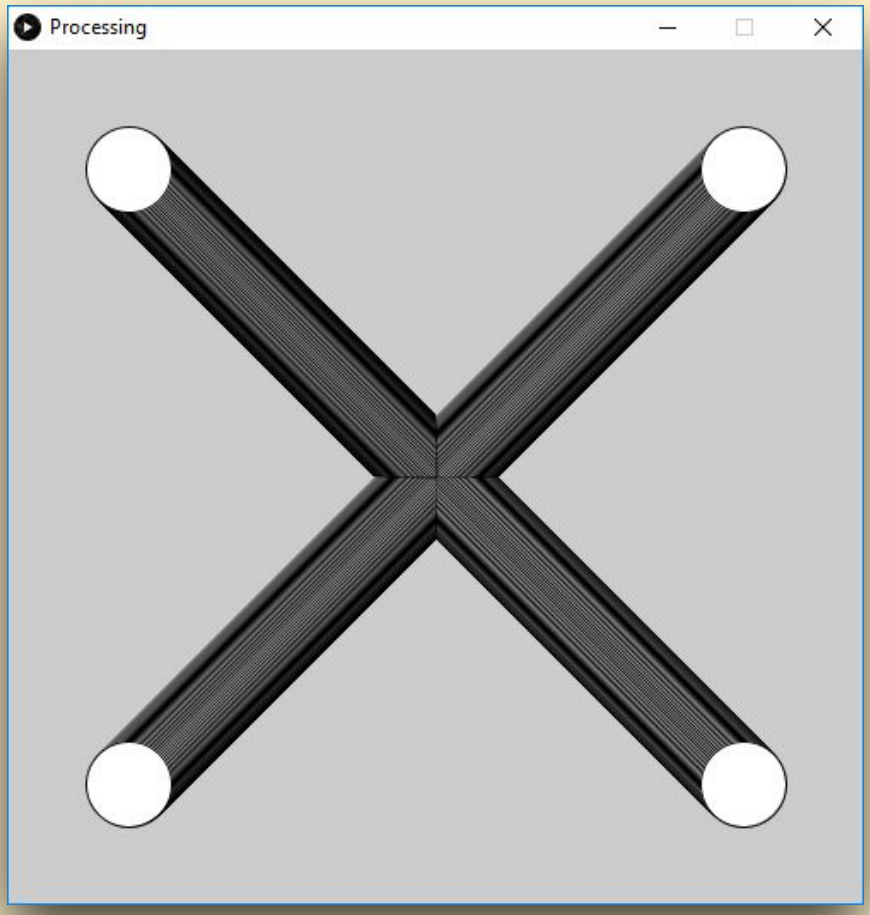
EXTRA POWER UP - Look at the Grade 5 - Unit 1 handouts to learn about another type of variables.

Lesson 2 - Variables II



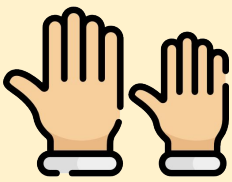
Learning Outcomes:

- We will learn **more** about variables
- We will make more **advanced** animations
- We will learn about naming **conventions** for variables



Make sure you know all these topics before you start:

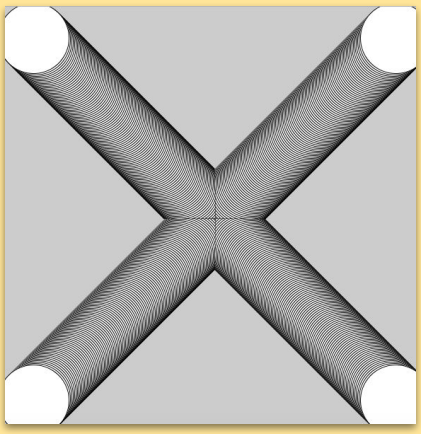
- ✓ Using void **setup()** and void **draw()**
- ✓ **Variables**; what they are and what they do
- ✓ **Difference** between **ints** and **floats**
- ✓ How to **declare**, **initialise** and **update** a variable



REMEMBER: If you have any questions, stay in your seat and put up your hand. We love to help!

Let's get coding!

You're going to use variables to make this design. All circles start at the same point, in the middle of your window.



1 **Declare** the variables for each circles' **x-position** and **y-position**.

```
float xPos1;
float yPos1;
float xPos2;
float yPos2;
float xPos3;
float yPos3;
float xPos4;
float yPos4;
```

2 Initialise each variable to the centre of the screen inside your void **setup()**.

```
xPos1 = 300;
yPos1 = 300;
xPos2 = 300;
yPos2 = 300;
xPos3 = 300;
yPos3 = 300;
xPos4 = 300;
yPos4 = 300;
```

3 Use your new variables to get your four ellipses to move **diagonally** from the centre of the screen.

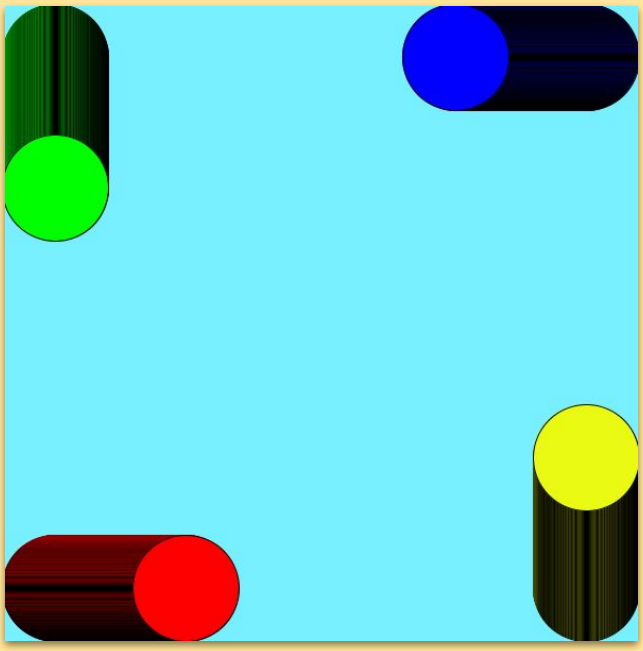
This code:

```
ellipse(xPos1, yPos1, 100, 100);
xPos1 += 1;
yPos1 += 1;
```

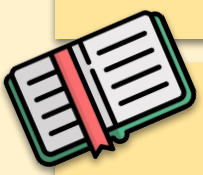
makes this ball move.

Challenge Task

Create a new sketch which recreates this drawing below where four ellipses start in each corner and move in different ways.



This challenge was designed by *Senan*, a student in *Oatlands College.* Look at Grade 5 Unit 1 to learn about another type of variable



Proper Naming Conventions

Let's go over the **naming conventions** for variables in Processing. Variable names should always be **short** and **meaningful.**

The following are **BAD** examples of variable names for the x-position of a circle:

theHorizontalCoordinateOfMyCircle

fortnite

myAuntRobin

Names like these are **a lot better**:

ballX

ballXPos

circleX

Expert Tip

Since you can't have spaces in your variables, if your variable name has more than one word, write them all together, start the **first** word all **lowercase**, but **capitalize** the **first letter** of each word **after** it. This is called **camelCase** in reference to a camel's back.

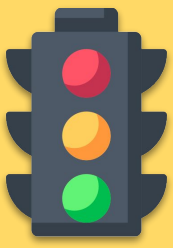
Expert Tip

Programmers, very time efficient people, always try to find shorter ways of writing the same code. So instead of writing `ballX = ballX + 1;` try `ballX += 1;` or even `ballX++;` if you're feeling extra lazy!

Make sure you understand all these concepts and have competed all of the tasks before moving on to the next lesson

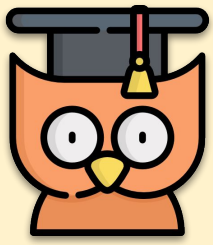
Concepts:

- **Variables; what they are and what they do**
- **Difference between ints and floats**
- **How to declare, initialise and update a variable**
- **How variables should be named**



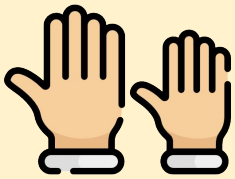
If you don't understand all these concepts, read back through the lesson or ask a tutor for help.

Lesson 3 - Variables III



Learning Outcomes:

- We will learn **more** about variables
- We will make more **advanced** animations
- We will learn about naming **conventions** for variables



REMEMBER: If you have any questions, stay in your seat and put up your hand. We love to help!

Making Flags with Variable

In this lesson, we're going to create a code with **variables** which will animate a flag. The code below was put together by Luca, a student in *Oatlands College* as an advanced example.

Let's get coding!

Task One - Ireland

Using the code on the right as a starting point, use variables to make an Irish flag similar to the one below.

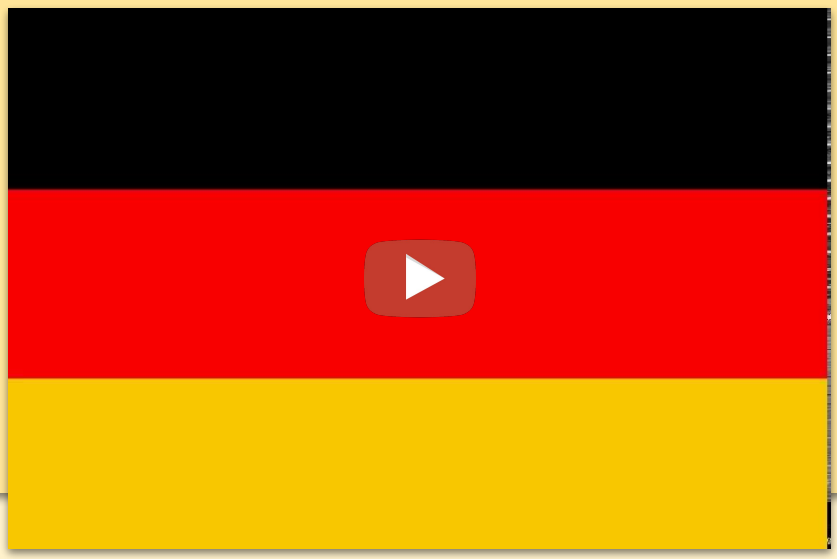
```
void setup() {
  size(900, 600);
}

void draw() {
  rectMode(CENTER);
  noStroke();
  rect(150, 100, 300, 200);
  rect(450, 500, 300, 200);
  rect(750, 100, 300, 200);
}
```

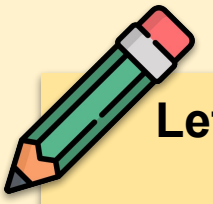
Expert Tip

Remember, to make a vertically moving animation, you'll need to use variables in the Y column.

Task Two - Germany

Changing the above code, make a German flag that appears in a 'stairs-like' fashion.





Let's get coding!

Task Three - Japan

Over the next few lessons, we'll be looking at how ***if statements*** work. In the example below; **if** the value of our variable (called *flagExpander*) is **less than or equal to 300**, the variable *flagExpander* will add 2 to itself. It will stop this action when it then gets to 300 meaning the circle will stop expanding.

```
ellipse(width/2, height/2, flagExpander, flagExpander);

if (flagExpander <= 300) {
  flagExpander = flagExpander + 2;
}
```

Task Four - Scotland

In a new project using two variables (in both x and y positions of two ellipses or rects) make a flag of Scotland.

You will need to put the blue background colour into *void setup* or the ellipses/rects won't leave a trail.

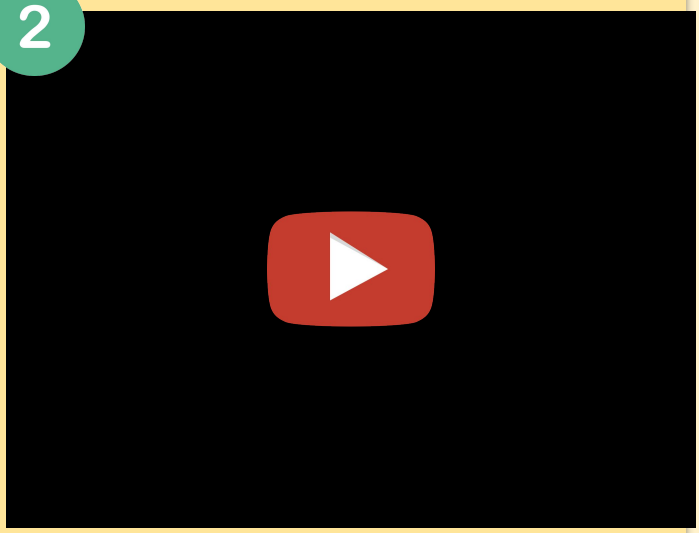
Challenge

Task Five - United Kingdom

The UK flag animation (which coded by Luca in Oatlands College) has **three variables**. Adapt your Scottish flag code so that all of the white lines travel first. Then adapt these 4 ellipses so that they are delayed (eg “variable - 200”).

1

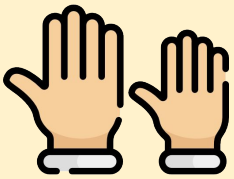
2



Lessons 4 and 5 - map() function

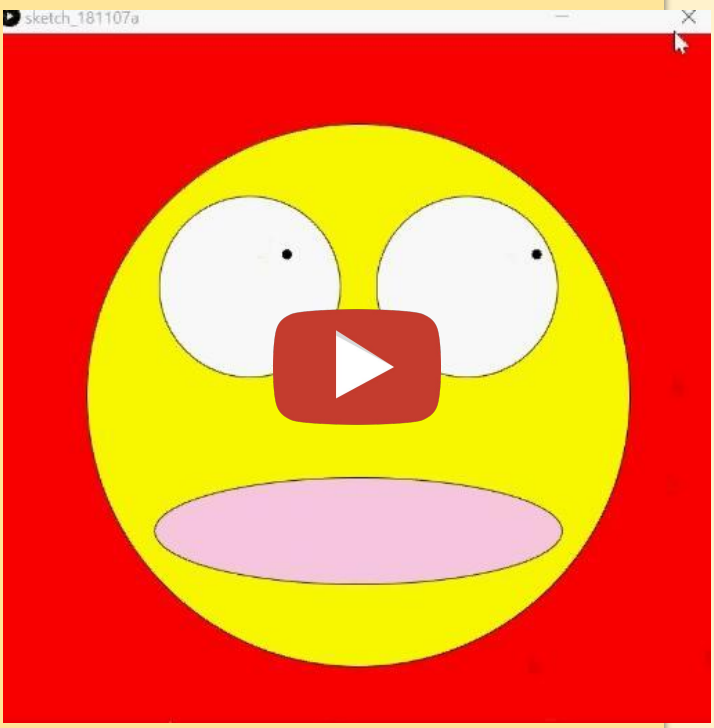
Learning Outcomes:

- Learning about the map() function
- Putting what we've learned into practice



REMEMBER: If you have any questions, stay in your seat and put up your hand. We love to help!

The map() function



In short, mapping allows us to take a value from a **old or given range** and to 'map' it to a **new range**.

In the example on the right, the pupils and mouth have been mapped to new ranges based and are being controlled by the mouse position.

We could use this function to create a health bar in a game or even to help surgeons perform microsurgery with precision robots in hospitals. Let's have a look at how it works.

How it works

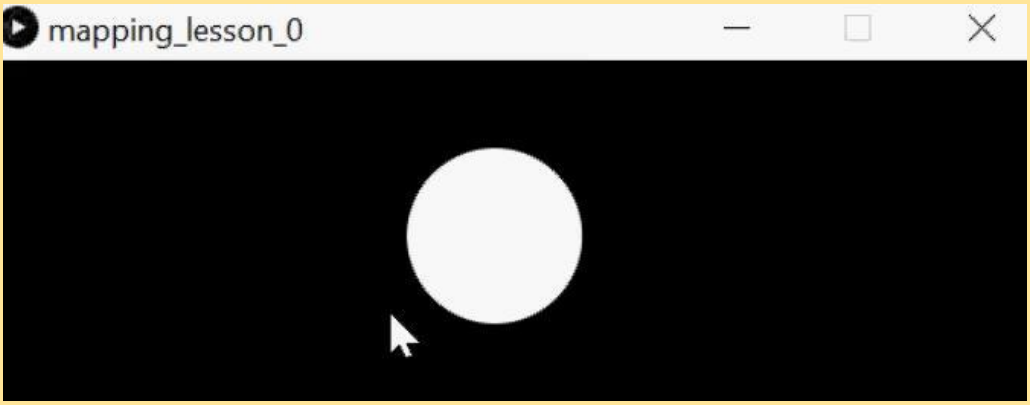
Mapping *re-maps* a number from one range to another like so.

The value's current range

```
map(value, start1, stop1, start2, stop2)
```

The incoming value to be used.

The new value range



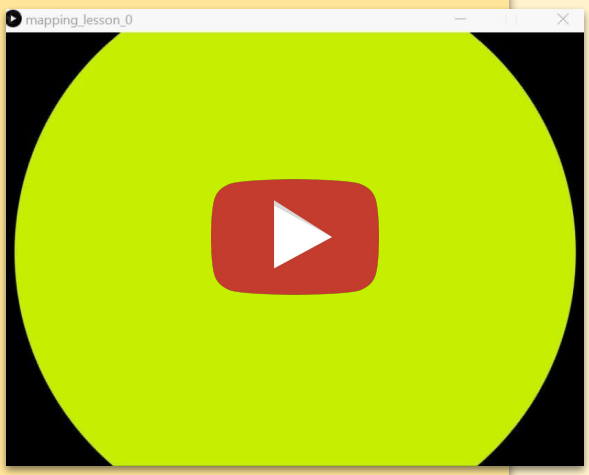

```
x1 = map(mouseX, 0, 600, 250, 350);
```

In the example above

- we determine our values from **mouseX**
- the old range goes from 0 to 600
- we 'map' that onto a new area that ranges from 250 to 350.

i.e. when the mouse is at 0, the value of **x1** is 250.
when the mouse is at 600, the value of **x1** will be 350.

Lets get coding



Task One

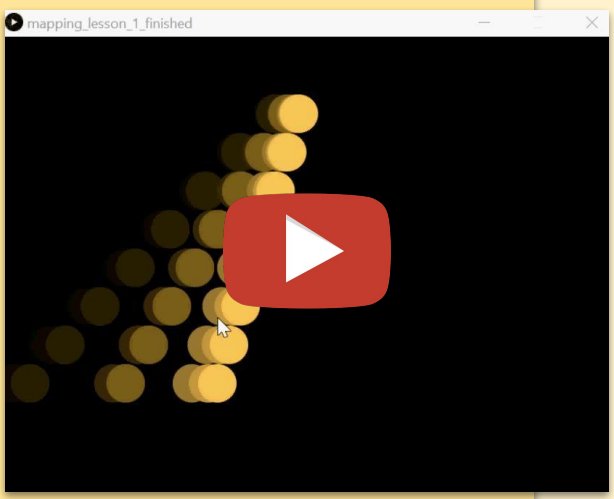
For the first example we're going to create an interactive programme like the one on the Right. The circle's X and Y positions will remain the same but it's size will need a mapped variable. Put **size**(800,600) in **void** setup.

HINTS

```
float size;
float colour;
```

```
ellipse(400, 300, size, size);
```

```
size = map(mouseX, 0, width, 40, 800);
```



Task Two

Using the same method as the previous exercise, try and see if you can make the animation on the right.

We will need to start with two variables for this, one for **xPos** and one for **colour**.

Each circle will need it's own variable.

Make **void setup()** have a canvas **size of (800,600)**. In **void draw()**, insert the following code to create an opaque **rectangle** that covers the whole canvas. This gives us the 'trail' effect.

HINTS

```
float x1;
float x2;
float colour;
```

```
fill(0, 50);
rect(0, 0, width, height);
```

0 for black, 50 is the alpha value

Our code should now look like this. The black rectangle acts as a background. Add the following code to get your first circles mapped to a new ranges. Don't forget to give your ellipses a **fill.**

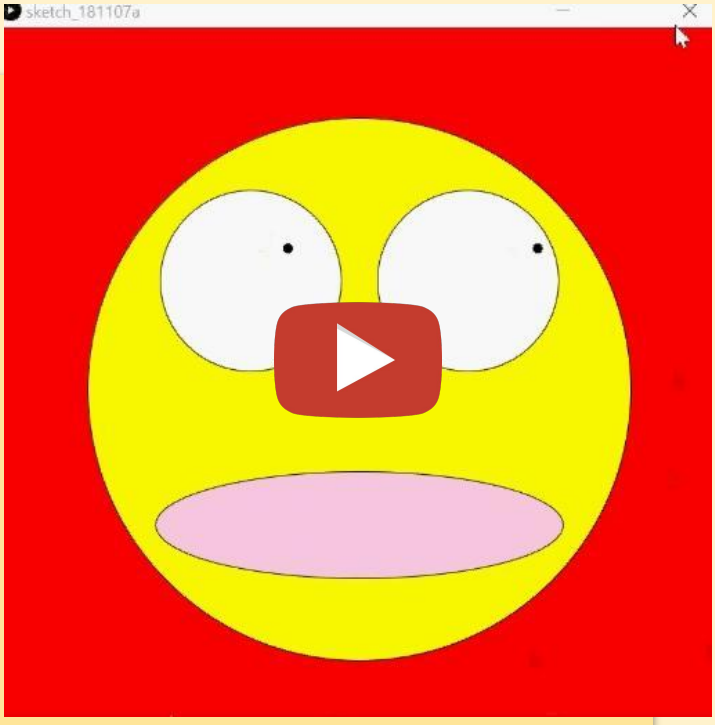
```
float x1;
float x2;
float colour;

void setup() {
  size(800, 600);
}
void draw() {
  fill(0, 50);
  rect(0, 0, width, height);
}
```

In this example, when mouseX is at 0, our circle's X position (x1) is 350. When mouseX is at 800 (the canvas `width`) x1 will be 450. Add more circles to finish this. Try and see if you can spot a pattern.

When finished see if you can make your colours change too similar to the example.

```
x1 = map(mouseX, 0, width, 350, 450);
ellipse(x1, 100, 50, 50);
```

```
x2 = map(mouseX, 0, width, 300, 500);
ellipse(x2, 150, 50, 50);
```



We're now going to look at using mouseX and mouseY to control this emoji's eyes and mouth.

We're going to use 5 variables here:

```
float mouthHeight;
float leftPupilXPos;
float leftPupilYPos;
float rightPupilXPos;
float rightPupilYPos;
```

Begin by making a face with a mouth, eyes and pupils similar to the one above. Choose your own colours, but have the design/layout similar to ours.

```
background(255, 0, 0); // red background
fill(255, 255, 0); // yellow
ellipse(400, 400, 600, 600); // head
fill(255); // white
ellipse(280, 280, 200, 200); // left eye
ellipse(520, 280, 200, 200); // right eye
```

Take some time to see if you can get the mouth moving.

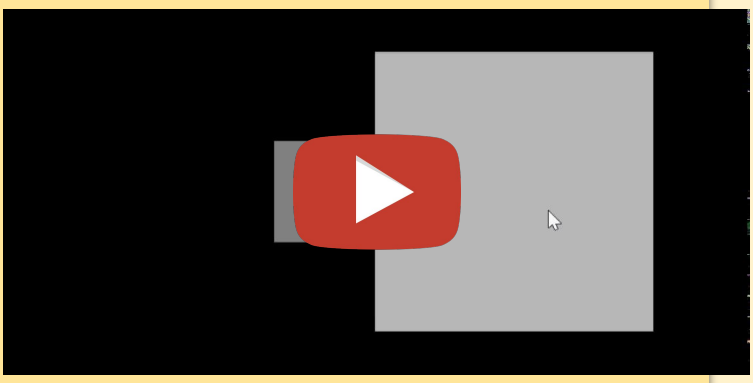
Mapping the Mouth and Eyes

Add the following code and fill in the blanks to see if you can get the mouth and eyes working.

```
fill(___, ___, ___); // mouth colour
mouthHeight = map(mouseY, ___, ___, ___, ___);
ellipse(___, ___, ___, mouthHeight); // mouth
```

```
fill(0); // black
leftPupilXPos = map(mouseX, 0, width, ___, ___);
leftPupilYPos = map(mouseY, 0, height, ___, ___);
ellipse(leftPupilXPos, leftPupilYPos, 10, 10);
```

We're now going to look at using mouseX to control to rect's at the same time.



We're going to use four variables. One for each rectangle size and 0ne to control the alpha levels/ opacity of each rectangle.

```
float rect1;
float rect2;
float opacityR1;
float opacityR2;
```

What do these lines do?

Declare these variables, and in **void setup**, set the canvas **size** to **(1000,500)** and put in `rectMode(CENTER);` and `noStroke();`

In **void draw**, pick a **background** colour. We picked black above.

```
rect1 = map(      X, [ ], width, 0, height);
opacityR1 = map(      X, 0, width, [ ], 255);

opacityR2 = height-rect1;
rect2 = height-rect1;

fill(opacityR1);
rect(width/2 + rect1/2, height/2, [      ], [      ]);

fill(opacityR2);
rect(width/2 - rect2/2, height/2, [      ], [      ]);
```

Use variables here

Try and see if you can fill in the blanks to get the above animation working.

If you're finished Have a look at Grade 5 Unit 1 to see buttons in action.