Hatch Reference Manual

Table of Contents

Storing

Key Concept : Learn how to use variables!

Table of Contents

Colouring

Key Concepts : Learn about Colours!

Table of Contents

Drawing

Key Concept : Learn how to add and format shapes, text, lines and images to your projects.

Table of Contents

Drawing

Table of Contents

Doing

Key Concepts : Add computational thinking to your projects, these are used in a variety of coding languages!

Table of Contents

Doing

Table of Contents

Doing

Table of Contents

Doing

Note:
Talk to your coach about Object-oriented Programming.

Table of Contents

Doing - Specifics

Key Concepts : Additional components to add to your projects and programs.

Table of Contents

Matrix Changes

Key Concepts : Rotate, move and change the size of objects in your code through matrix changes.

Table of Contents

Converting

Key Concepts : Translate your knowledge to different environments and learn new skills.

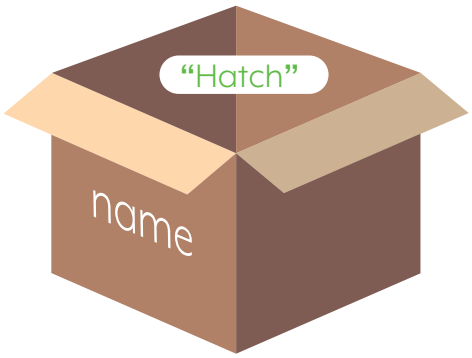
Variables

What is a Variable?

A variable is a container that *stores* data. This data can change.

The text **"Hatch"** is stored in the variable **name.**

```
var name = "Hatch";
```



Assigning Values to Variables

Use the **= Equals Sign** to assign a value to a variable.

1. The **value** *always* goes to the **right** of the equal sign.

```
var name = "Hatch";
```

2. Text must be within **quotes.**

```
"Hatch"
```

Variables

Types of Values you can Store

Variables can hold **text** and **numbers,** including **integers** and **decimals.** They can hold many other things as well!

1. Use quotes to store text

```
var name = "Hatch";
```

2. Storing numbers

```
var num = 5.3;
```

How to Name your Variables

Variable names are **case sensitive** and must be **unique** and **MUST** follow these **3 rules:**

1. Starts with a **Letter**

```
var name;
```

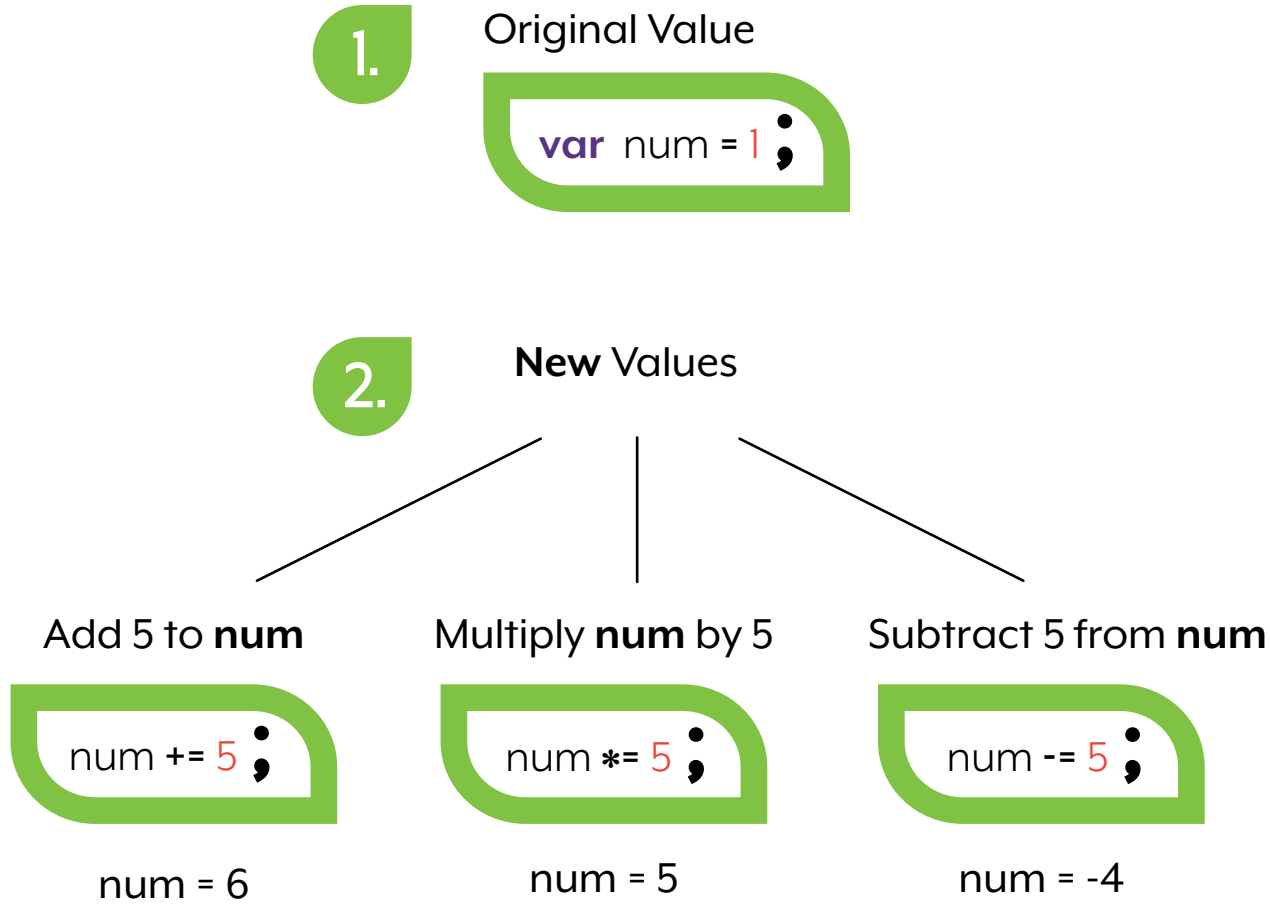
2. Must contain **ONLY**

3. **Cannot** be a JavaScript Keyword

Variables

Reassigning Variables

You can change a value stored in a variable while the code runs.

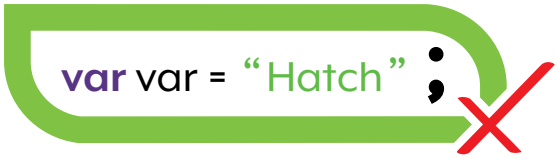


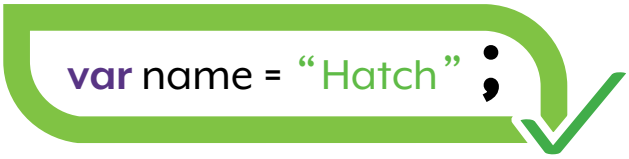
Reserved Keywords

Reserved Keywords

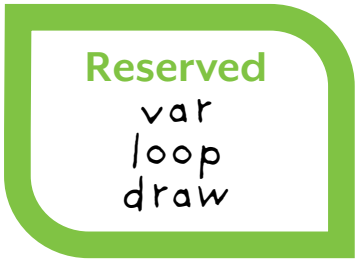
In English “CAT” is a defined word that has a specific meaning. JavaScript has those as well.

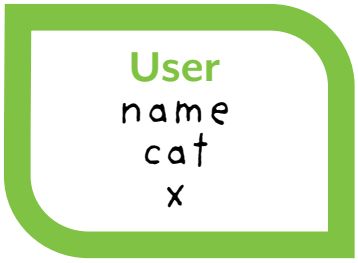
1. The word **var** can only be used as a JavaScript Keyword





Here are some examples





Using Booleans

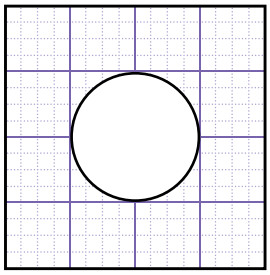
Booleans allow you to turn specific components on and off. They are stored as true or false.

Using Booleans

This is a boolean that stores true or false in a variable named circle.

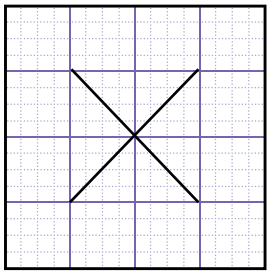
```
var circle = true;
```

```
var circle = true;
if (circle) {
  ellipse (200, 200, 200, 200);
}
```



This will draw a circle if var circle is true.

```
var circle = false;
if (!circle) {
  line (100, 100, 300, 300);
  line (100, 300, 300, 100);
}
```



This will draw an X if it is false.

Colour Basics

RGB Colours

In a computer, colours are made by **combining** three colours together: **Red**, **Green**, and **Blue**.

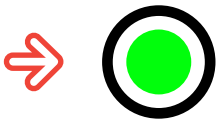
By setting different values for **Red**, **Green**, and **Blue** you can create any colour! 255 means the most of a colour, 0 means no colour.

Really Red!

```
color(255, 0, 0);
```

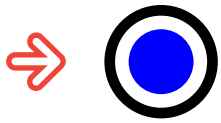
Really Green!

```
color(0, 255, 0);
```



Really Blue!

```
color(0, 0, 255);
```

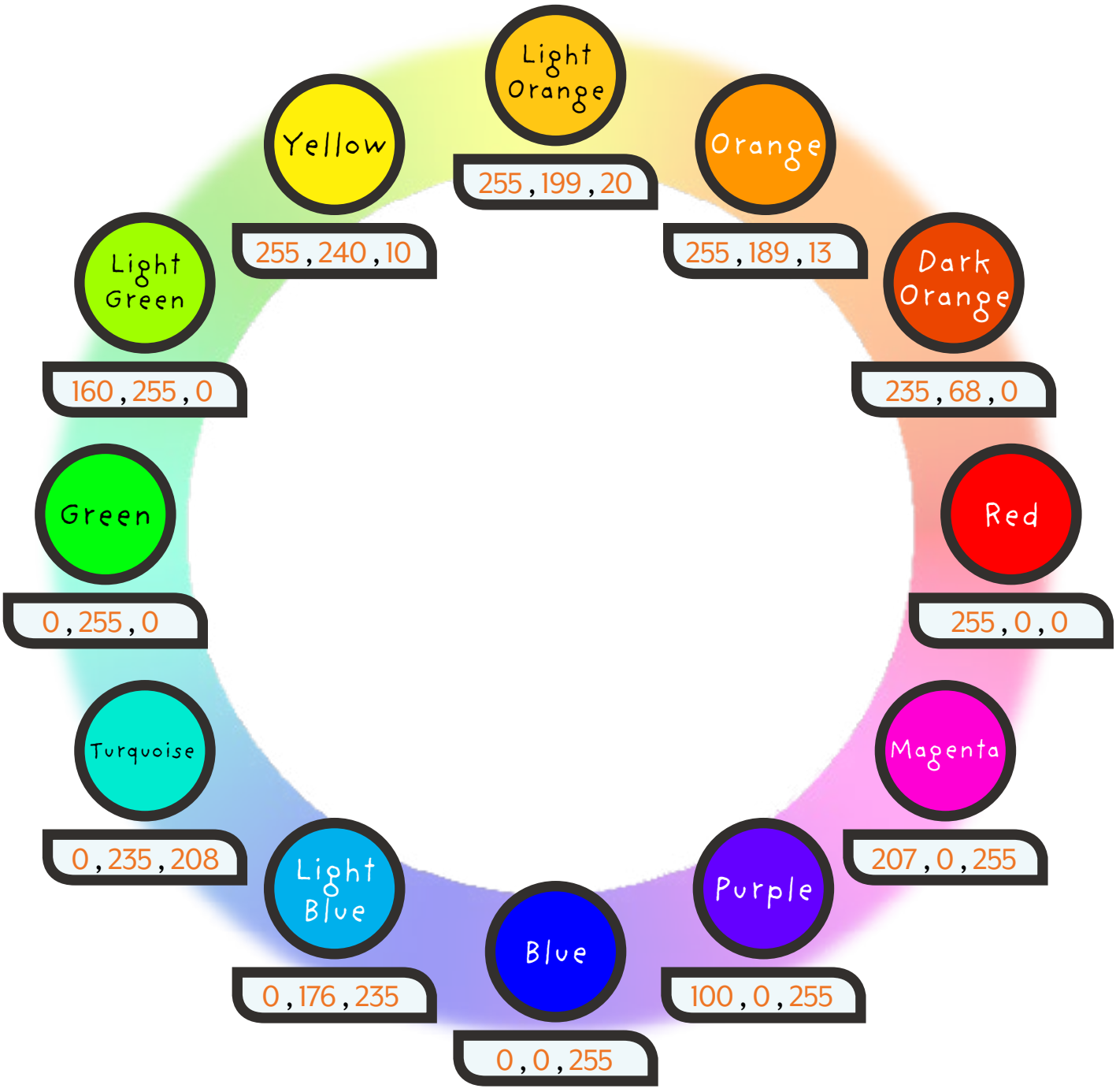


White	Light Grey	Grey	Dark Grey	Black
255, 255, 255	192, 192, 192	128, 128, 128	64, 64, 64	0, 0, 0

The reserved keyword color does not have a u. This means you can use colour as a user-defined keyword.

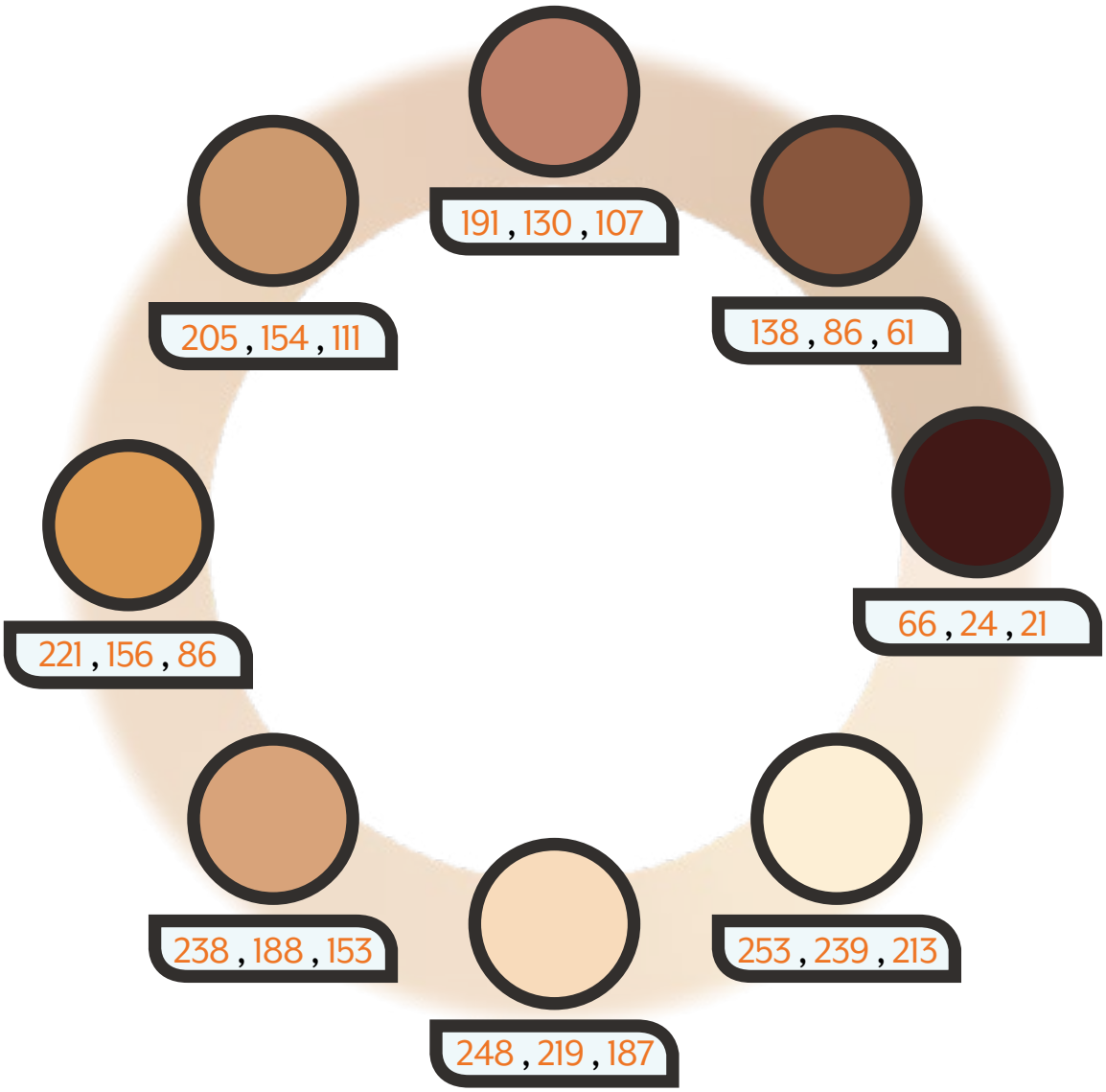
Colouring

RGB Colour Wheel



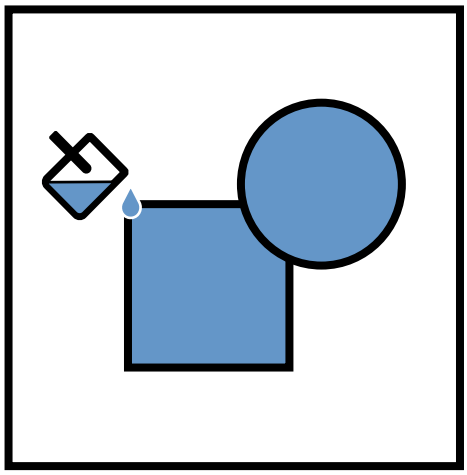
Colouring

Skin Tones



Order of Colours

Set the colour of multiple shapes. All shapes will be filled with the colour after the fill function.



Stroke is the outline,
Fill is the inside.

```
fill(100, 150, 200);
rect(100, 150, 150, 150);
ellipse(250, 150, 150, 150);
```

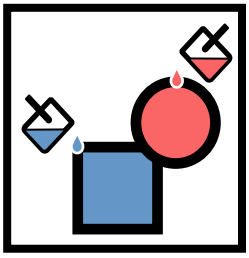
1. Fill and Stroke act on all shapes below them.

Acts on both shapes

```
stroke(10);
fill(100, 150, 200);
rect(100, 150, 150, 150);
fill(250, 100, 100);
ellipse(250, 150, 150, 150);
```

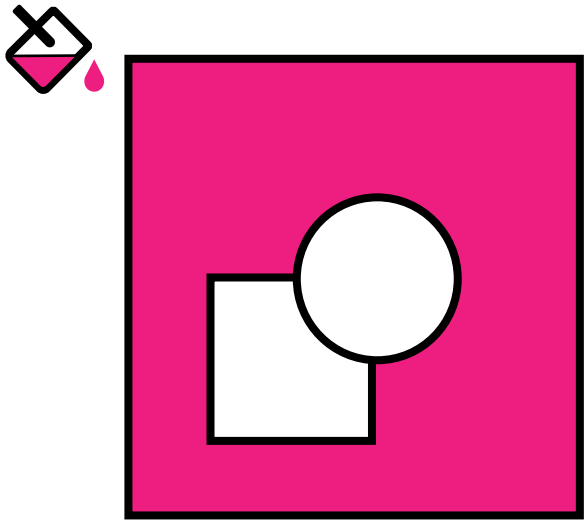
Fills the rectangle

Fills the ellipse



Background

Set the **background** colour of the **canvas.**



The amount of **Blue** and the **Transparency**

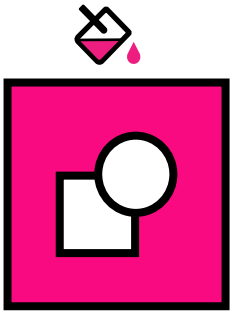
```
background ( R , G , B , A ) ;
```

The amount of **Red** and **Green**

Setting the Background Colour

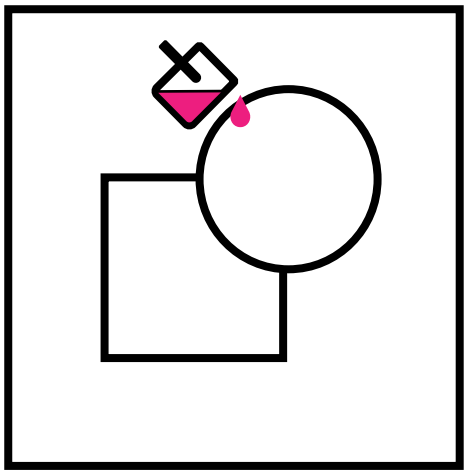
The background function sets the colour of the canvas.

```
background ( 255 , 0 , 128 ) ;
rect ( 100 , 150 , 150 , 150 ) ;
ellipse ( 250 , 150 , 150 , 150 ) ;
```



Fill

Fill acts on the inside of shapes and text and fills them with colour.



Transparency means see through

If you only write one number R, G and B will be the same number. This will make the fill colour grey.

The amount of **Blue** and the **Transparency**

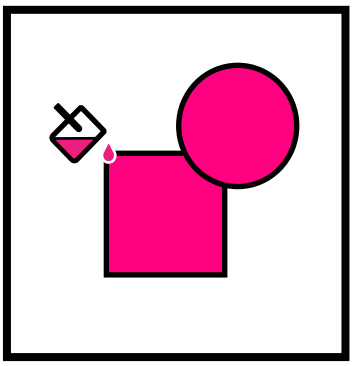
```
fill ( R , G , B , A ) ;
```

The amount of **Red** and **Green**

Filling Shapes with Colour

Place the fill function before the shapes.

```
fill ( 255 , 0 , 128 ) ;
rect ( 100 , 150 , 150 , 150 ) ;
ellipse ( 250 , 150 , 150 , 150 ) ;
```

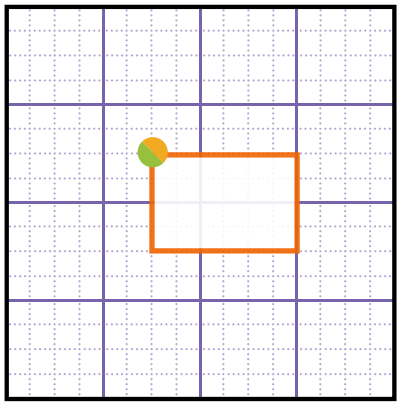


Stroke

This is the outline of a shape or the colour of a line.

1. The stroke function changes the colour of the outline.

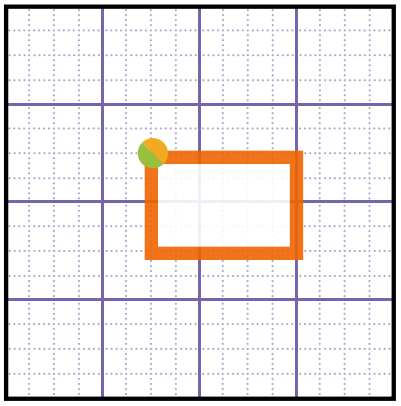
```
stroke(240, 100, 0);
rect(150, 150, 150, 100);
```



strokeWeight

This changes the thickness of the outline.

```
strokeWeight(5);
rect(150, 150, 150, 100);
```



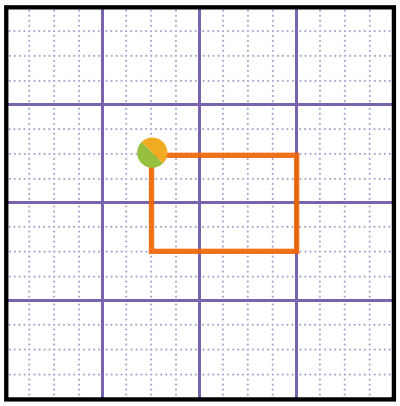
noStroke & noFill

You can remove the colour or the outline by calling the noFill or noStroke functions. The inside of these functions are empty. They do not require parameters.

noFill

Using noFill removes the inside colour of a shape or text.

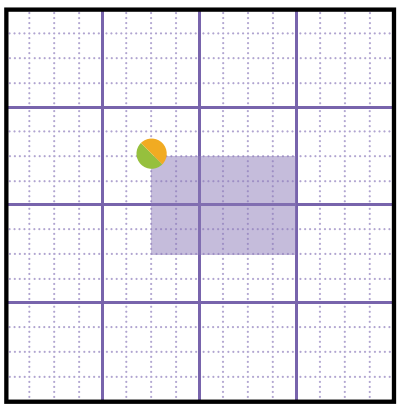
```
noFill ( ) ;
stroke ( 240 , 100 , 0 ) ;
rect ( 150 , 150 , 150 , 100 ) ;
```



noStroke

Using noStroke removes the outline of a shape

```
noStroke ( ) ;
fill ( 139 , 121 , 184 ) ;
rect ( 150 , 150 , 150 , 100 ) ;
```



ColorMode

You can use HSB to adjust colours differently.

HSB

HSB means hue / saturation / brightness.

Max Value

```
noStroke ( ) ;
colorMode (HSB , 400) ;
for (var i = 0; i < 4 0 0 ; i ++) {
    for (var j = 0; j < 4 0 0 ; j ++) {
        fill (i , j , 400) ;
        ellipse (i , j , 1 , 1) ;
    }
}
```

HSB allows you to draw rainbows easily.

Hue

Is the colour Red, Green, and Blue.

0 = Red

200 = Green

400 = Blue

Red is the lowest value, green is the middle value and blue is the highest.

Saturation

Is how much colour.

0 = Grey

400 = Full Colour

Brightness

Is how bright the colour is.

0 = Dark

400 = Bright

You can set the top value as the second argument

ColorMode

You can use RGB or you can use HSB as parameters in colorMode( );

RGB

This is Red, Green, and Blue. You can also add in the max value as the second argument.

Max Value

```
noStroke ( ) ;
colorMode (RGB , 400) ;
for (var i = 0; i < 4 0 0 ; i ++) {
    for (var j = 0; j < 4 0 0 ; j ++) {
      fill (i , j , 0) ;
      ellipse (i , j , 1 , 1) ;
    }
} ;
```

I used 400 to have a more gradual change and to fill all the canvas

Coordinate Plane

What is the Coordinate Plane?

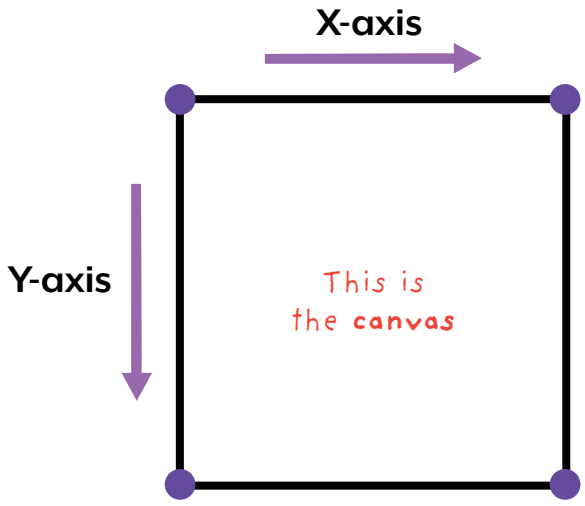
The coordinate plane / grid is a **two-dimensional surface** formed by the **x-axis** and **y-axis**.

This does **NOT** act like in math. When the Y value increases the Y location moves down.

(X, Y)

Horizontal Position

Vertical Position



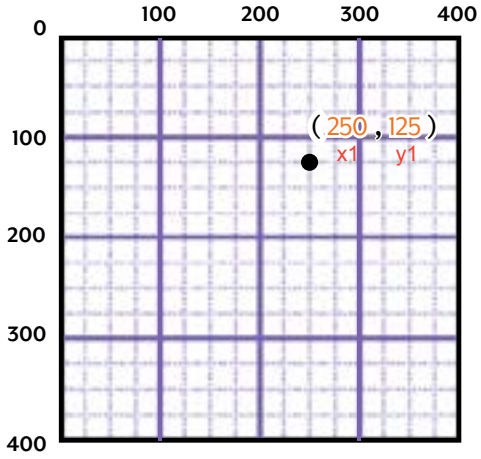
How to use the Coordinate Plane

Use the **x** and **y coordinates** to place items on the canvas.

```
point (250 , 125) ;
```

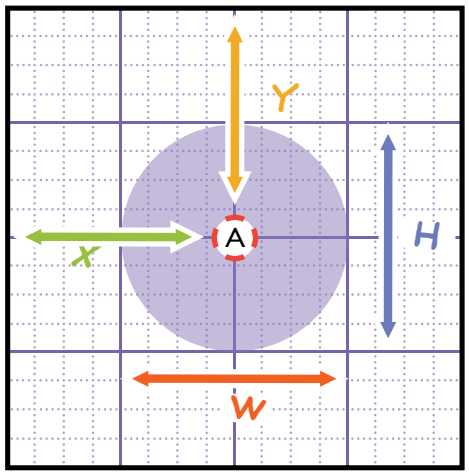
x1 y1

Point



Ellipse

Draws an **Oval** or **Circle** from the **center point** Ā



The Width and Height of the circle

```
ellipse ( 200 , 200 , 200 , 200 ) ;
```

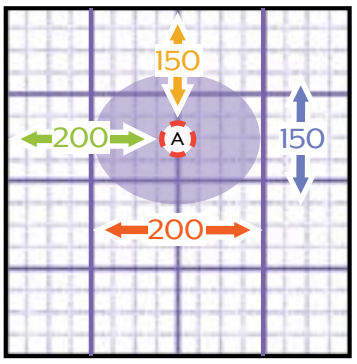
The X and Y position
of the oval's center point Ā

How to Draw an Oval

Set the ellipse's **center point** first,
then set the **width** and **height.**

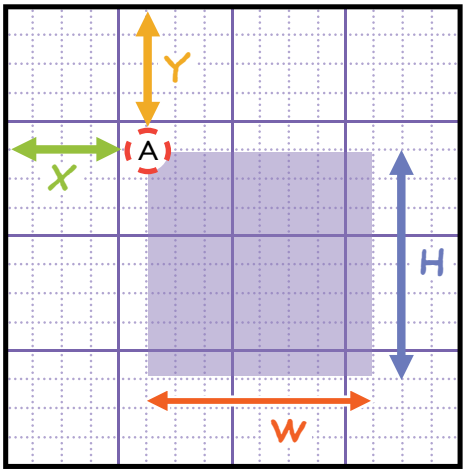
```
ellipse ( 200 , 150 , 150 , 200 ) ;
```

The X and Y position
of the oval's center point Ā



Rect

Draws a **Rectangle** or **Square** from the upper **left corner point** (A)



The **Width** and **Height** of the rectangle

```
rect ( 125 , 125 , 200 , 200 ) ;
```

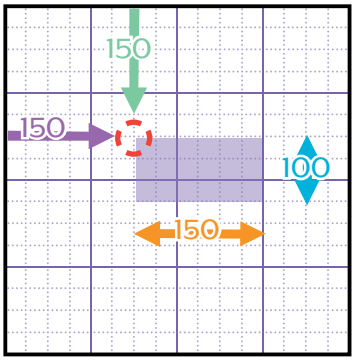
The X and Y position
of the rectangle's center point (A)

How to Draw a Rectangle

Set the rectangle's **upper left corner** first,
then set the **width** and **height**.

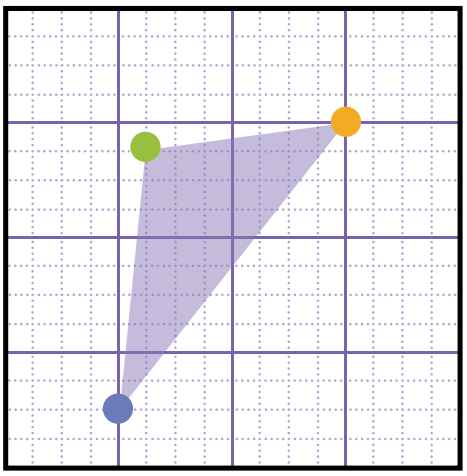
```
rect ( 150 , 150 , 150 , 100 ) ;
```

The X and Y position
of the rectangle's center point (A)

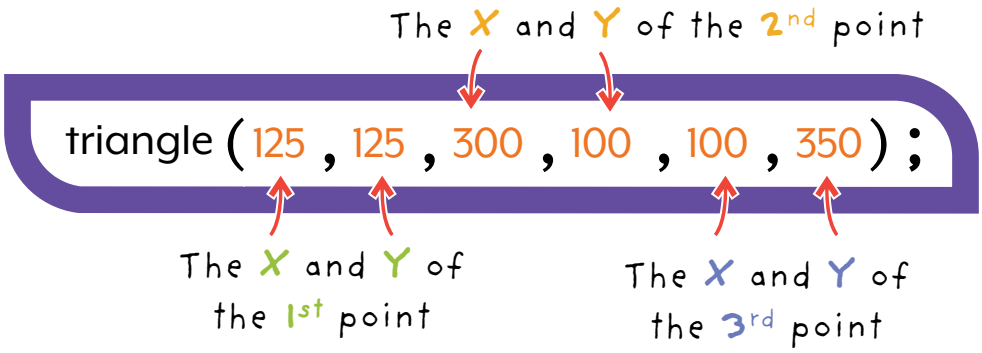


Triangle

Draws a triangle using three given points, each point will need an **X** and **Y** position.



A vertex is another name for a point



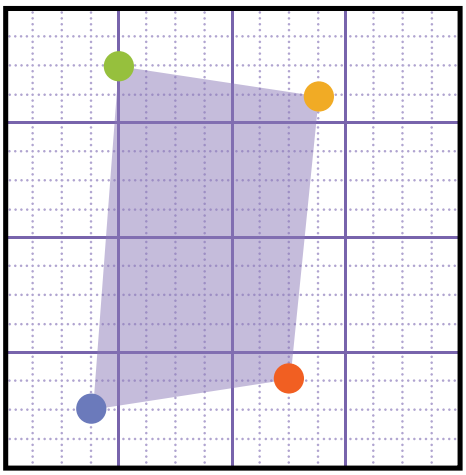
1. Set the values of the 1st vertex on X and Y in the canvas.

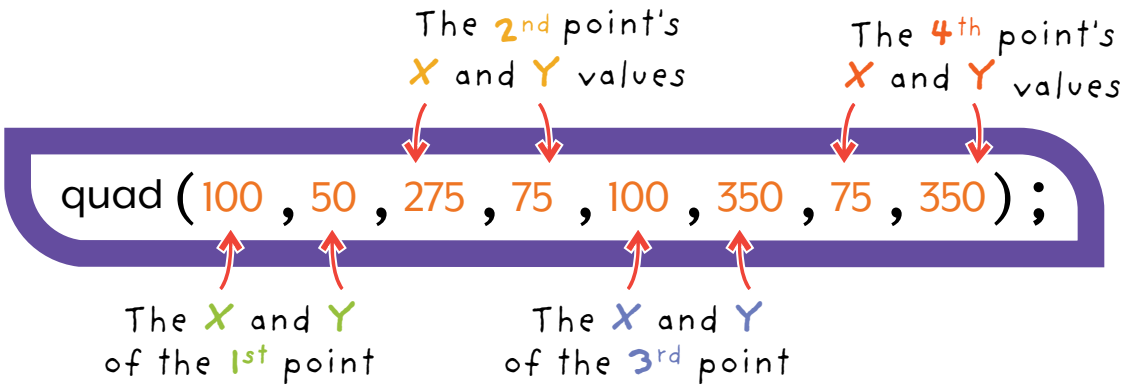
2. Continues by assigning the X and Y for the 2nd vertex.

3. And then the 3rd vertex, set the X and Y in the canvas and the triangle will show up.

Quad

Draws a four sided figure like a rectangle,
except the points can be drawn from anywhere.





1. Set the 1st vertex's X and Y values on the canvas.

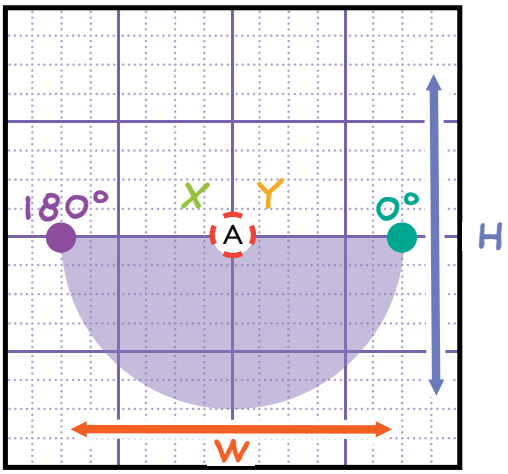
2. Continue by assigning the X and Y values for the 2nd vertex.

3. Then set the X and Y values for the 3rd vertex.

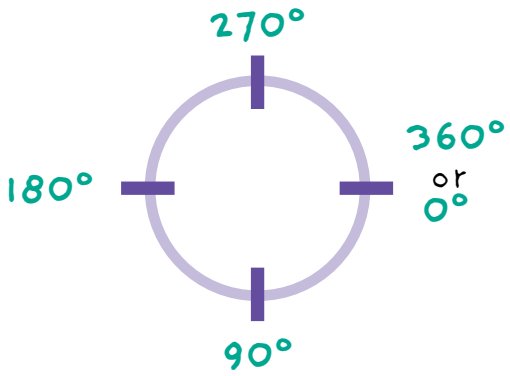
4. Finally, set the X and Y values of the 4th vertex.

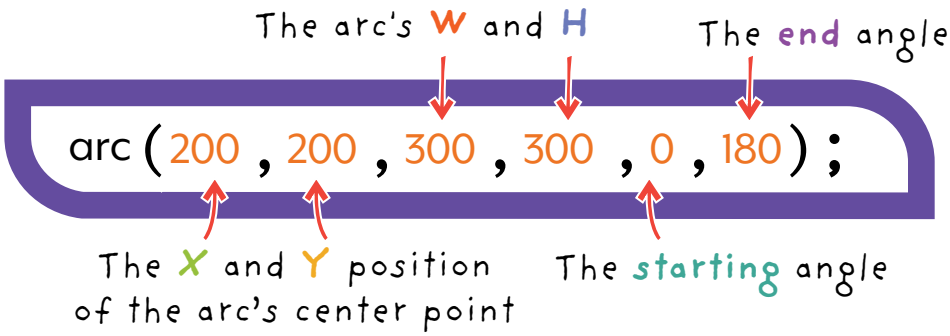
Arc

Draws a part of a circle. It will need the X and Y values to set where the center of the arc is. Then **width** and **height** as a second pair. Finally, the **starting** and **end angles** to set how much of the circle we want to draw

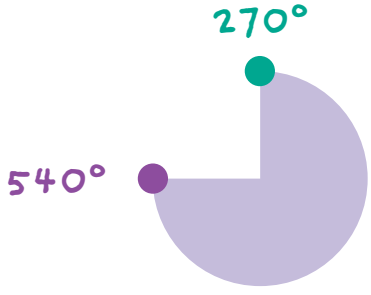


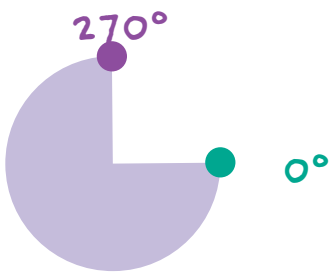
Basic Angles



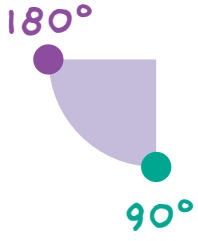


Sample Arcs



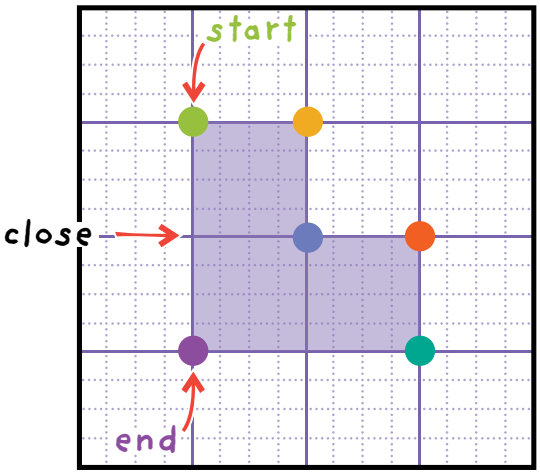


NOTE: The end angle must be bigger than the starting angle for the arc to appear.



Special Shapes

To draw a custom shape, list all the vertices or points using vertex(x, y) and put them between **beginShape( )** and **endShape( )**.




```
beginShape ( ) ;
vertex ( 100 , 100 ) ;
vertex ( 200 , 100 ) ;
vertex ( 200 , 200 ) ;
vertex ( 300 , 200 ) ;
vertex ( 300 , 250 ) ;
vertex ( 100 , 300 ) ;
endShape ( CLOSE ) ;
```

1st point of X and Y

2nd point of X and Y

3rd point of X and Y

4th point of X and Y

5th point of X and Y

6th point of X and Y

NOTE: If you do not add in CLOSE in endShape, you will have to add an additional vertex that has the same X and Y as the first.

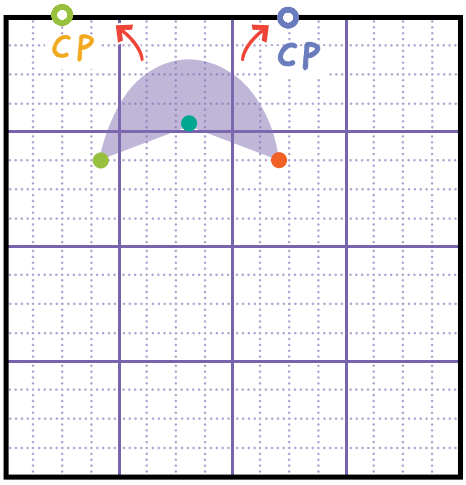
Complex Shapes

BezierVertex

You can use **bezierVertex**() to create shapes with curves. These work similar to bezier but without the first anchor point.

The vertex ● before the bezier vertex starts the curve.

The bezier vertex ● completes the curve.




```
beginShape();
vertex(80, 125);
bezierVertex(100, 0, 220, 0, 240, 120);
vertex(160, 90);
endShape(CLOSE);
```

Point of X and Y

Point of X and Y

Curve Point 1

Curve Point 2

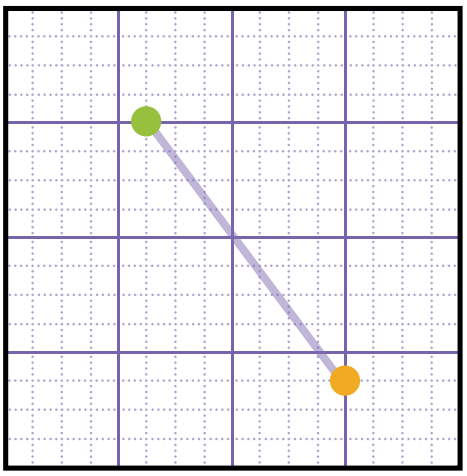
Anchor Point X Y

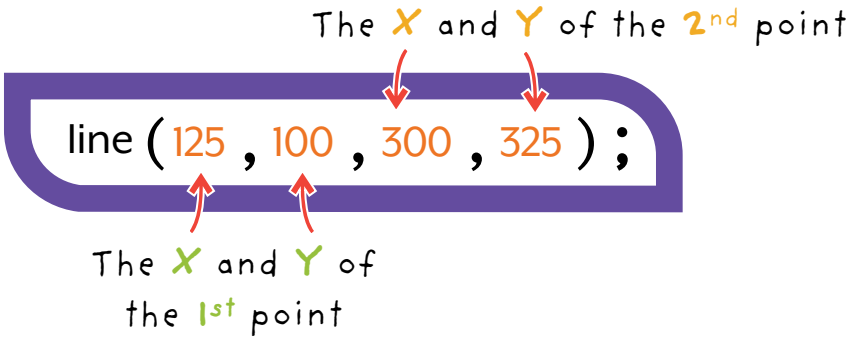
The control points pull the line drawn toward them.

Tip: Make sure you know all your vertex points and plan your pattern first on paper.

Line

Draws a line from one point to another on the screen.





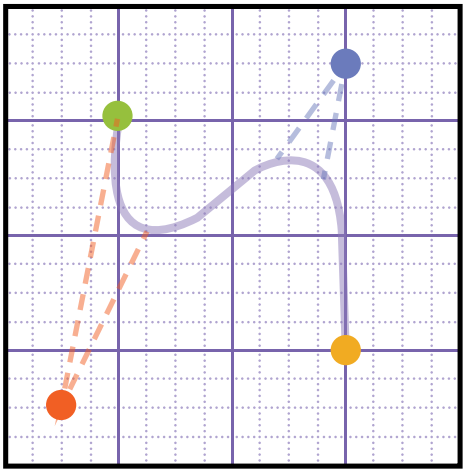
1. Set the X and Y values for where you want the 1st point to be on the canvas.

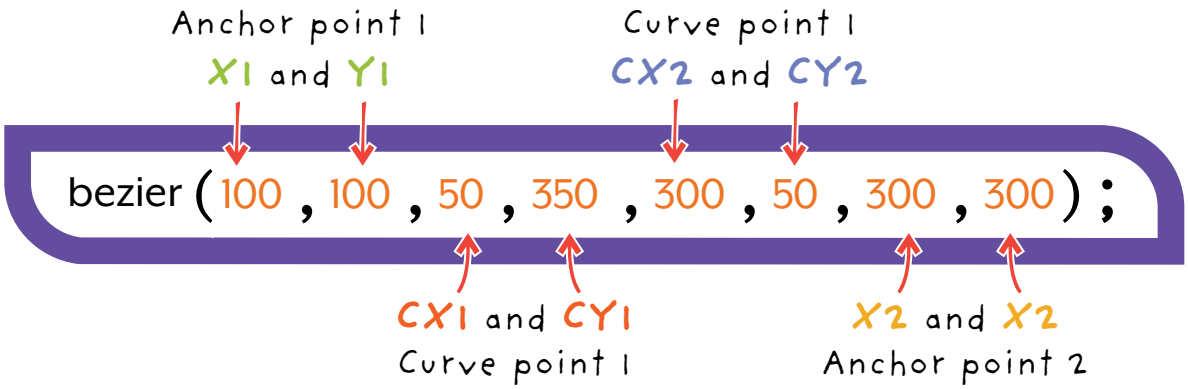
2. Continue by assigning the X and Y values of the 2nd point.

Tip: stroke() sets the lines colours and strokeWeight() sets the thickness

Bezier

This allows you to make a curved line.



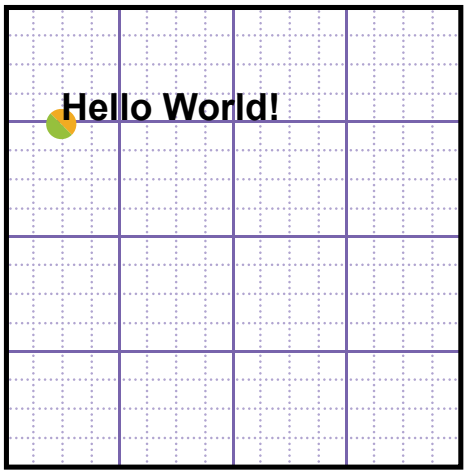


1. Anchor points are where the line starts and ends.

2. Curve points pull the line towards themselves. You can see this with the dotted lines.

Text

Draws text on the canvas.



Remember text is coloured using fill, not stroke. Text acts like shapes.

```
text ( "Hello World" , 50 , 100 ) ;
```

What the text says. Type the text between quotes (" ")

The X and Y position of the start point in the text

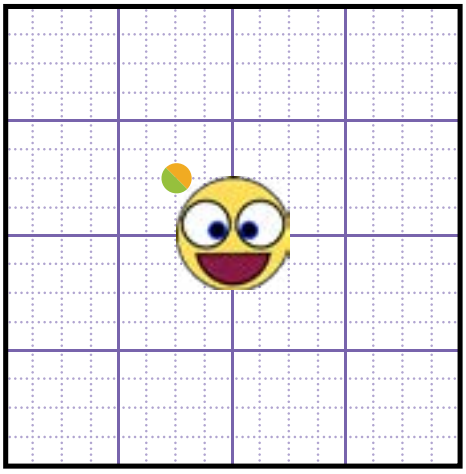
How to Set the Size of Text

The first line of code sets the size of any text below it. The size of the text is determined by a number.

```
textSize ( 14 ) ;
text ( "Hello World" , 50 , 100 ) ;
```

Image

Displays an image on the canvas.



To add or change an image, go to the Hatch Image library and copy the text under the image. Paste this between the (" "). Below are two different ways to make an image appear.

1.

```
image(getImage("creature/sub3"), 150, 150, 100, 100);
```

The image's W and H

The image's X and Y

2.

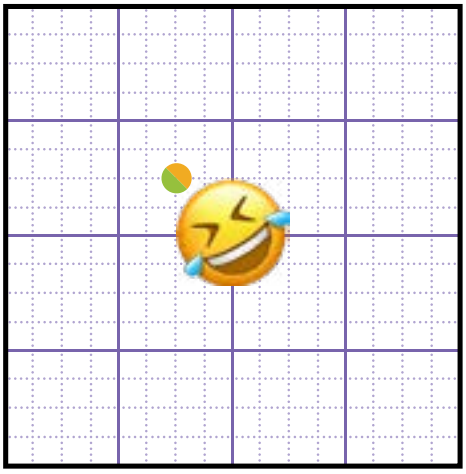
```
var img = getImage("creature/sub3");
image(img, 150, 150, 100, 100);
```

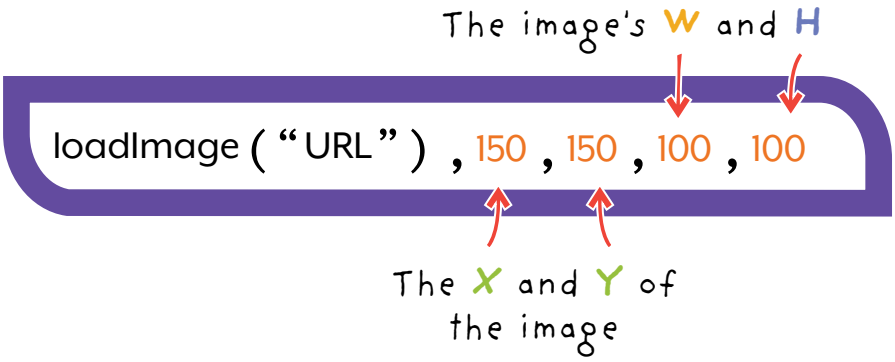
The image's X and Y

W and H of the image

Internet Image

Displays an image taken from the Internet on the canvas.



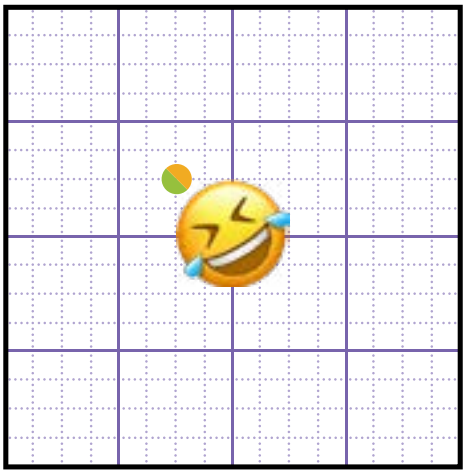


1. To add or change an image from the Internet, go to your browser and search for an image that you would like to use. The file type needs to be .png, .gif, .jpg, .jpeg, or it may not load.

2. Once you've found the image, open it in a new tab. Highlight the URL and drag it into the coding area to use it in your program. You can not copy and paste into projects.

Internet Image

To use an Internet Image, you need to call your image inside of a draw function.



```
var img = loadImage("URL");
var draw = function(){
    image(img, 150, 150, 100, 100);
};
```

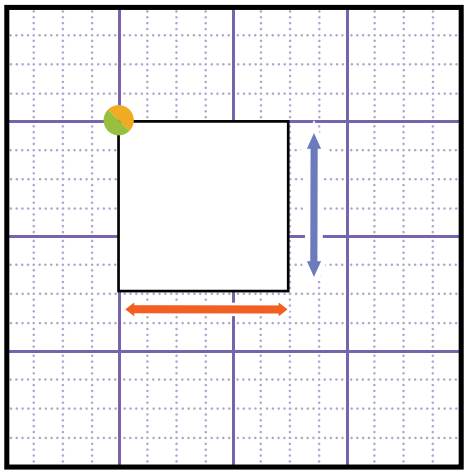
Without calling it in a draw function, the program won't correctly load your image.

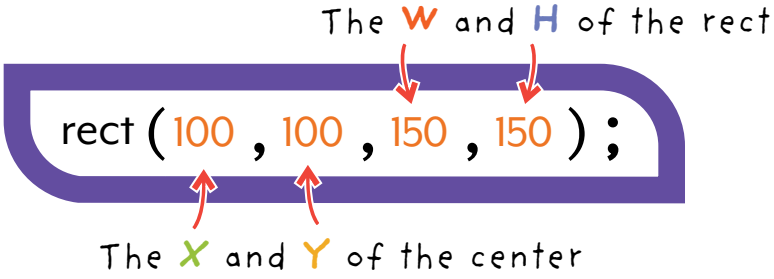
Rect Mode

This changes the starting location of what is being drawn.

rectMode ( );

The default mode for rectangles draws them from the top left corner.





This is the normally drawn rectangle.

This is **rectMode(CORNER) ;**

Rect Mode

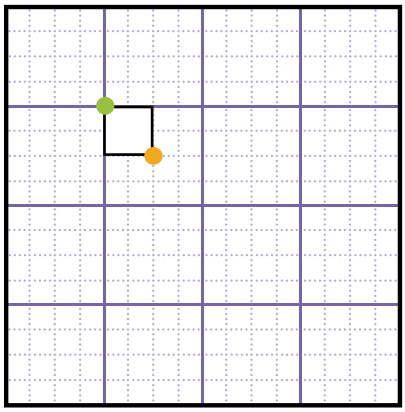
These are the other possible options for **rectMode.**

Define the corners of your rect

```
rectMode(CORNERS);
rect(100, 100, 150, 150);
```

The X and Y of the 1st point

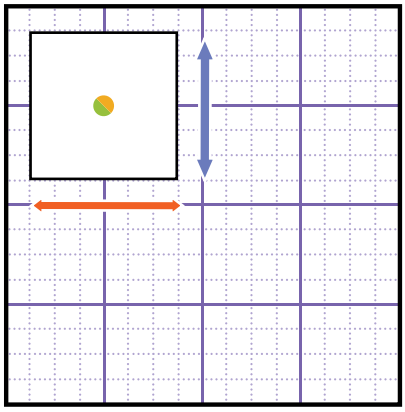
The X and Y of the 2nd point



Set the X and Y to the center and regular Width and Height

```
rectMode(CENTER);
rect(100, 100, 150, 150);
```

The X and Y position is the center point

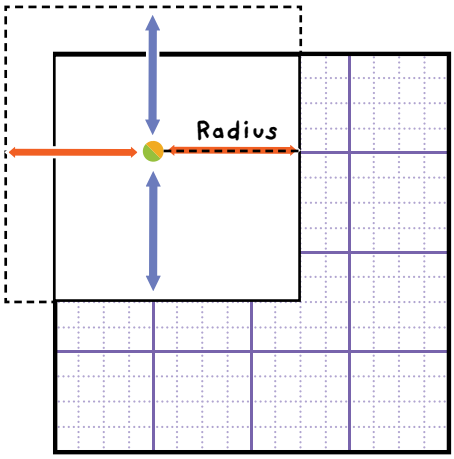


Set the X and Y to the center and double the Width and Height

```
rectMode(RADIUS);
rect(100, 100, 150, 150);
```

The X and Y of the center point

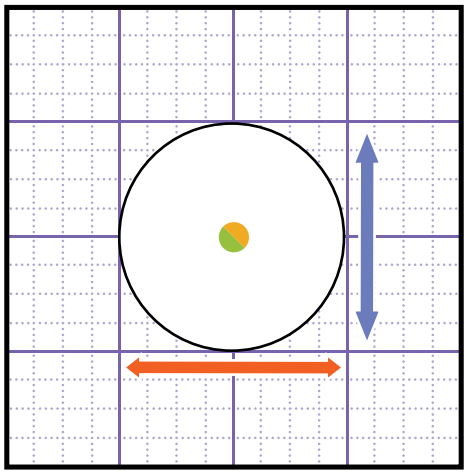
1/2 the W and H size of the rect



Ellipse Mode

ellipseMode ( );

The default mode for ellipse is CENTER.
This draws an oval from the center point.



The W and H of the circle

```
ellipse ( 200 , 200 , 200 , 200 ) ;
```

The X and Y of the center point

This is **ellipseMode(CENTER) ;**

Ellipse Mode

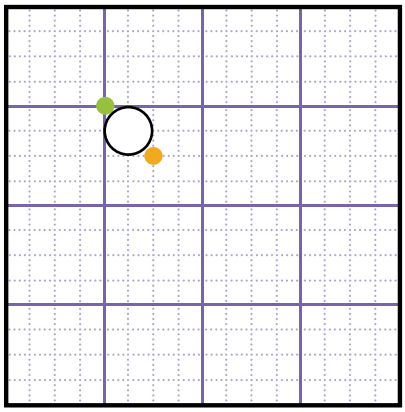
These are the other possible options for **ellipseMode.**

Define the corners of your ellipse

```
ellipseMode(CORNERS);
ellipse(100, 100, 150, 150);
```

The X and Y of the 1st point

The X and Y of the 2nd point

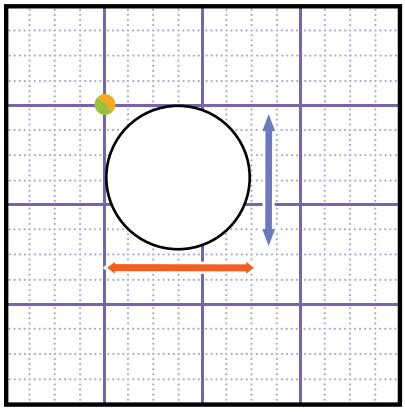


Set the X and Y to the center and regular Width and Height

```
ellipseMode(CORNER);
ellipse(100, 100, 150, 150);
```

The X and Y of the 1st point

The W and H of the circle

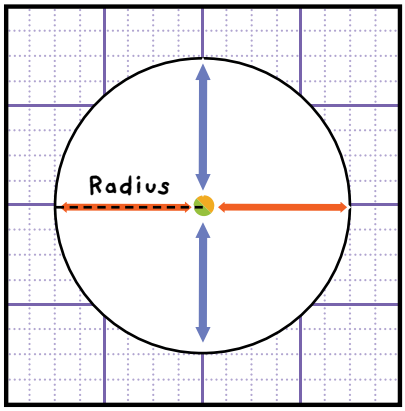


Set the X and Y to the center and double the Width and Height

```
ellipseMode(RADIUS);
ellipse(200, 200, 150, 150);
```

The X and Y of the center point

1/2 the W and H size of the oval

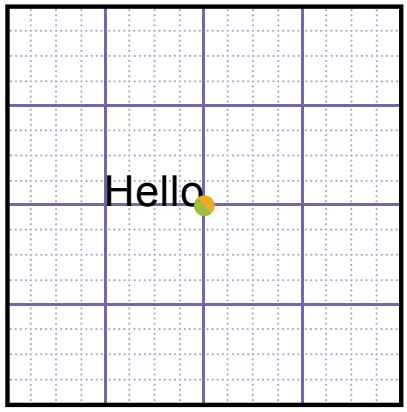


Text Align

You can choose the location where the text starts to be drawn by using textAlign. You can use RIGHT, LEFT or CENTER.

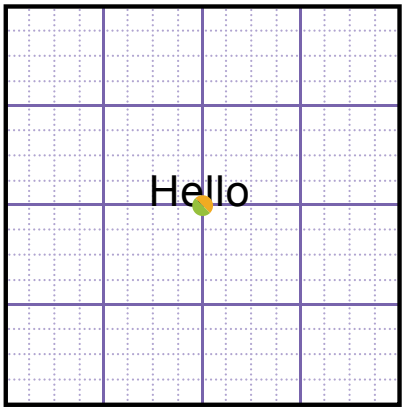
```
textAlign(RIGHT);
textSize(100);
text("Hello", 200, 200);
```

The X and Y of the bottom right corner



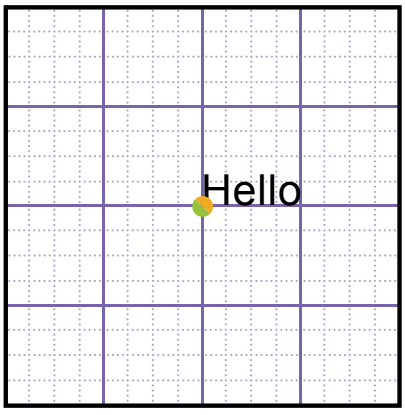

```
textAlign(CENTER);
textSize(100);
text("Hello", 200, 200);
```

The X and Y of the center point




```
textAlign(LEFT);
textSize(100);
text("Hello", 200, 200);
```

The X and Y of the bottom left corner



Commenting

Commenting code makes it easier to read and understand.
This is really important when doing group work!

// makes a comment
// any text after "//" will not run.

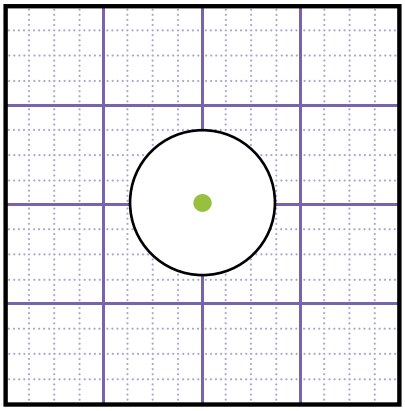
Comment Sections

/* Allows you to comment out several lines of code.
This is a quick way to remove code, without deleting it */

EX.

```
// draw a circle          ← Not running
ellipse (200 , 200 , 150 , 150);
/* ellipse (25 , 25 , 25 , 25);
   rect (100 , 25 , 25 , 25);
*/
```

Not running



Only the ellipse at **200 , 200**
will show on the canvas.

Doing

Functions

Creating a Function

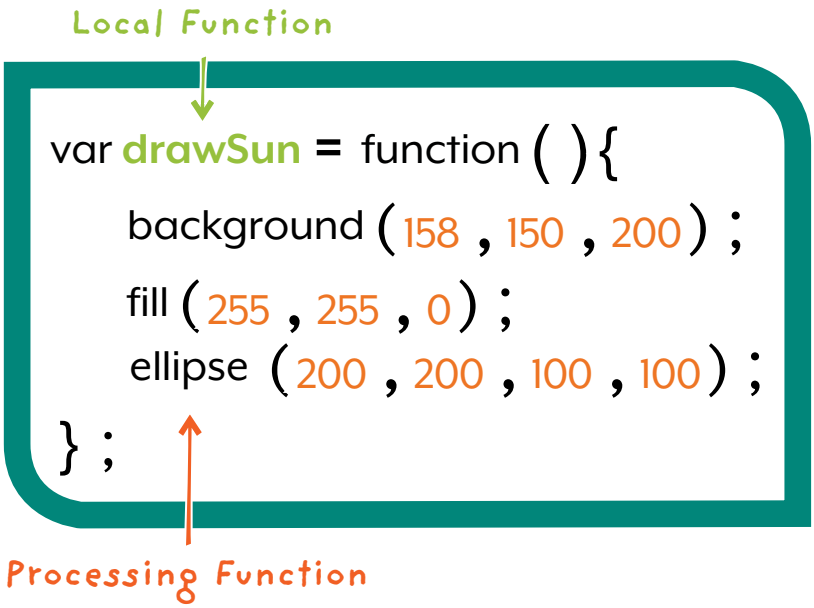
Functions help organize code by splitting it up into sections. This also lets you repeat code without having to type it out again. All you need to do is call the function!

```
var NameOfFunction = function (parameters) {
    // the function's code lives here
};
```

1. Start with the **var** keyword.
2. Define the name of your **function.**
3. Set your named **function** equal to the function keyword.
4. Write the **parameters** in between the brackets.
5. Place the **code** that will be executed by the function between two curly brackets.
6. End the function by putting a semicolon after the closing curly bracket.

Functions

Local Vs. Processing Function



Local Function

Created by the programmer (You!) as a specific collection of code needed for this particular project.

Processing Function

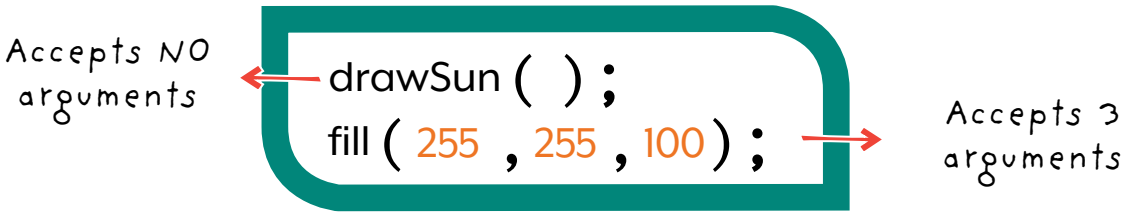
Developed as part of the programming language.
A useful function that the computer already knows and can be used in any project.

Note: Check Out Draw Function for an example

Functions

Calling a Function

Functions only perform the code inside of them when they are called.



Calling a Function Inside a Function

If you want to add animation to your projects, call functions within a draw, keyPressed or mouseClicked style of function. These are reserved functions.

```
var draw = function ( ){
    background (255 , 0 , 128) ;
} ;
```

Draw Function

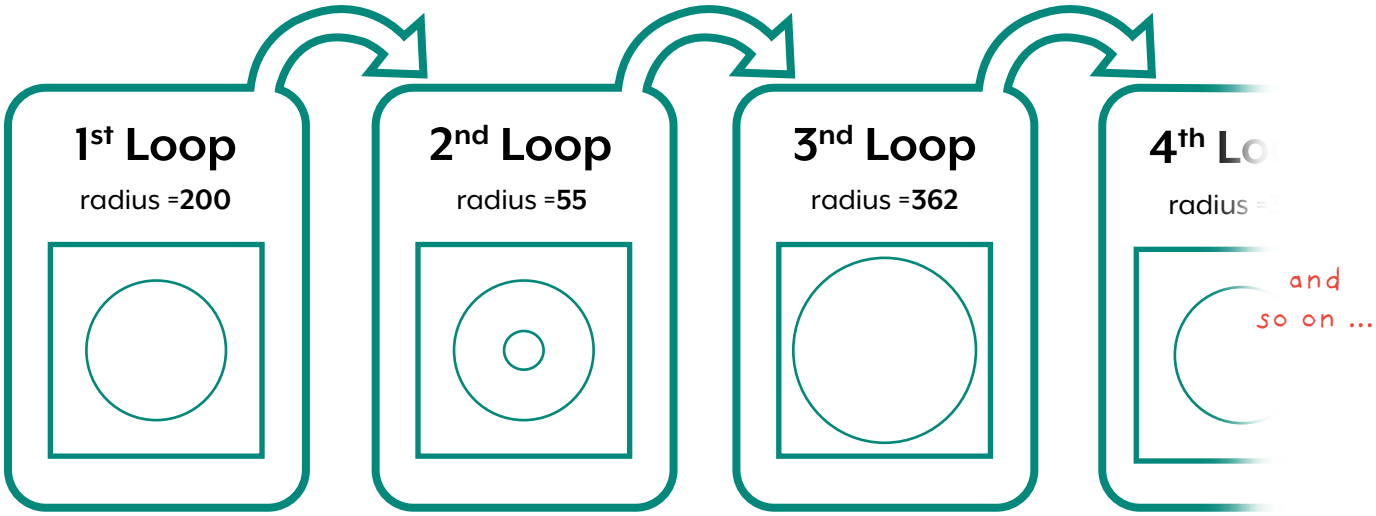
Imagine the draw function like drawing or pulling out of a box, not like drawing a picture.

How does the Draw Function Work?

The draw function allows you to **animate shapes**, **images** and **colours** by redrawing the canvas 60 times per second.

This code will draw circles continuously. They will be drawn on top of each other with a random radius.

```
var draw = function( ){
    var radius = random ( 0 , 400 ) ;
    ellipse ( 200 , 200 , radius , radius ) ;
} ;
```



This example shows that the code is drawn the first time, then it is drawn on top of the first version, then on top of the second and continues as long as the code is running

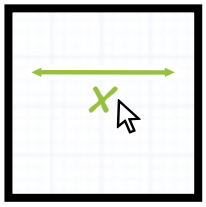
Mouse Position

Horizontal Position of the Cursor

The keyword **mouseX** always contains the current **horizontal position** of the mouse cursor's location.

mouseX

The right and left position of the

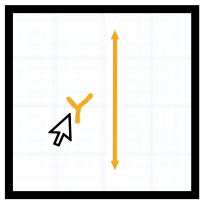


Vertical Position of the Cursor

The keyword **mouseY** always contains the current **vertical position** at the mouse cursor location.

mouseY

The up and down position of the



Using mouseX and mouseY

The keywords **mouseX** and **mouseY** act as variables and can be used as parameters.

```
var draw = function( ){
    background(255,0,128);
    line(0, 0, mouseX, mouseY);
};
```

Mathematics

Assignment Operators

Assignment Operators set or change a value to a different value. These are usually used to change the value of a variable.

Examples

=	assignment	*=	multiplies by a value
+=	adds the value	/=	divides by a value
-=	subtracts the value	++	increment (adds 1)
		--	decrement (subtracts 1)

```
var num = 1;

num += 3;
```

num is assigned a value of 1

num has 3 added to it.
num is now 4.

Mathematics

Arithmetic Operators

Arithmetic operators are used like normal math.
These are often used to change a value in a parameter.

Examples

+	addition	*	multiplication
-	subtraction	/	division
		%	modulus

```
var num = 2;

point(0, num*10)
```

Num is assigned a value of 2

The point created has an X value of 0 and a Y value of 20.

The value for num does not change.

Mathematics

Comparison Operators

These check to see if something is true or false.
These are often used in if statements and for loops.

Examples

>	is greater than	>=	is greater than or equal to
<	is less than	<=	is less than or equal to
&&	and	===	is equal to
||	or	!==	is not equal than

```
var num = 1;
if (num < 2) {
    fill(0);
}
```

num is assigned a value of 1

The code checks if num is less than 2.
If it is, the fill is changed to black.

Random

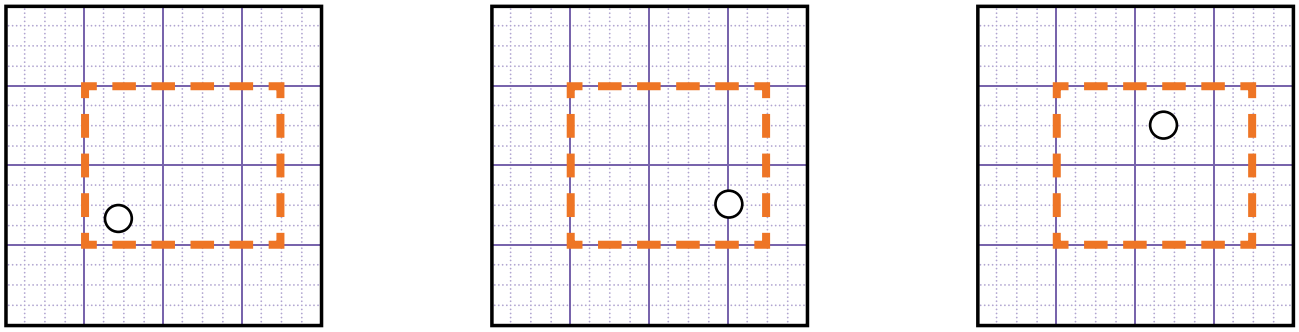
Random( ) allows the program to randomly pick a value from a range.

random( )

Instead of writing a number of variables, use random

```
ellipse ( random ( 100 , 350 ) , random ( 100 , 300 ) , 30 , 30 ) ;
```

This could become any of these or any in-between



The random() function only applies to the x location and y location in this example

Random

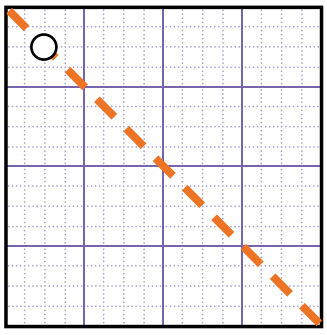
User-created random variables will have the same value throughout the program.

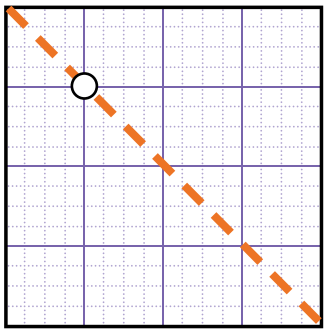
Random in Variables

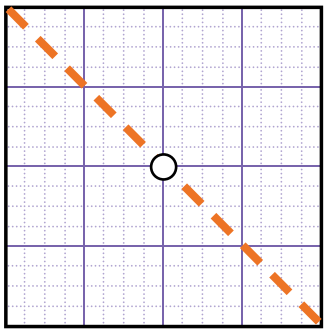
You can use the same random value more than once.

```
var ran = random(0, 400);
ellipse(ran, ran, 30, 30);
```

This means a value between 0 and 400 will be assigned to the variable ran.







When you use random() as the parameter of a function, the value will be random each time it is called.

If you assign random() to a variable and use the variable as the parameter, the value will remain the same every time you call the function, until you actively change the value.

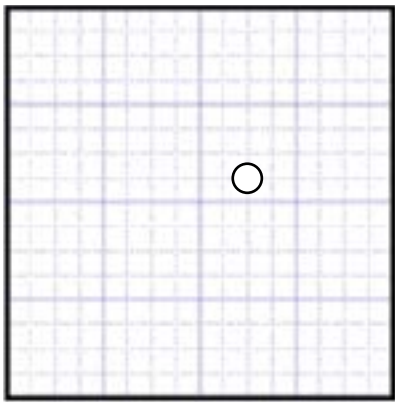
Random

What if you want something to look random, but always actually be the same?

randomSeed( )

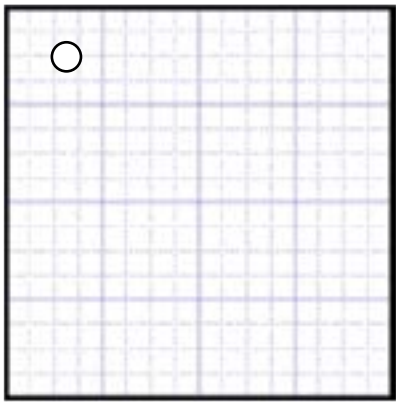
Ever used Minecraft? A seed value determines what version of a map will be made!

```
randomSeed (124) ;
ellipse (random (124 , 255),
         random (124 , 255),
         30 , 30 ) ;
```



This will always create the image to the right. Try it yourself!

```
randomSeed (4) ;
ellipse (random (0 , 255),
         random (0 , 255),
         30 , 30 ) ;
```



This will always create the image to the right. Try it yourself!

Doing

Conditionals - If

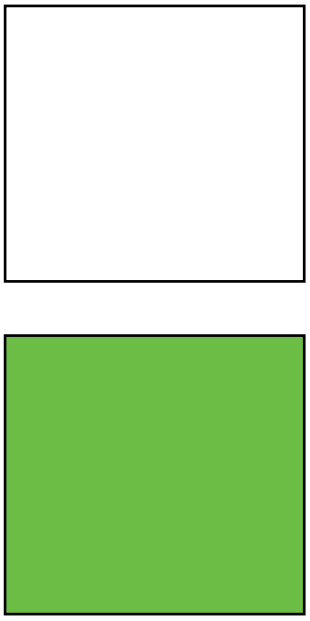
"If" statements check if something is true or false. If it is true, something happens. If false, it doesn't.

Real World Example

If you are hungry, then go get some food.
If you are not hungry, you would not need to get food.

Coding Example

```
var randomNum = random(0, 1000);
if (mouseX > 500) {
  background(100, 200, 0);
}
```



This code randomly picks a number from 0 to 1000. If the number is bigger than 500 the screen will turn green. If not, the screen will stay white.

Doing

Conditionals - Else

Else code runs when all if statements and else if statements are false.

Real World Example

I want to wear a shirt, but a sweater is ok, after that a coat is ok. However, if I have nothing else, I need to go shopping.

Coding Example

```
var randomNum = random (0 , 1000);
if (randomNum > 500) {
    background (100 , 200 , 0);
} else if (randomNum < 300) {
    background (255 , 0 , 0);
} else {
    image (getImage ("creatures/winston"));
}
```

randomNum = 400

If the number is not bigger than 500 and not smaller than 300 then Winston will appear

Conditionals - Else If

Else if statements work with an if statement.
If the first condition is **true**, this code runs.
If it is **false**, the next if else statement will check if the second condition is true. You can have multiple else if statements.

```
var randomNum = random(0, 1000);
if (randomNum > 500) {
    background(100, 200, 0);
} else if (randomNum < 300) {
    background(255, 0, 0);
}
```

Coding Example

1. This checks if the number is bigger than 500.
If it is, then it is green.
2. If not, it checks if it smaller than 300.
If it is, it is red.

randomNum = 200

randomNum = 400

randomNum = 600

Doing

Conditionals - Else If

It is important to make sure all of your conditions are different than the previous ones.

```
var randomNum = random(0, 1000);
if (randomNum > 500) {
    background(100, 200, 0);
} else if (randomNum > 600) {
    background(255, 0, 0);
}
```

All if statements connected to else if statements should be indented in the same lines.

In the example above, the background would never turn red because the if statement would be true every time the else if statement would be true

Coding Example

randomNum = 400

randomNum = 550

randomNum = 650

Conditionals Examples

Conditionals work like **YES** or **NO** questions. These are some additional examples of conditionals.

```
if ( question ) {
    ( response )
}
```

```
if ( weather is raining ) {
        take an umbrella
}
```

```
if ( weather === raining ) {
        take an umbrella
}
```

```
if ( my age is older than 18 ) {
        I can buy a dog
} else if ( my age is less than 18 ){
        I need to ask my parents
        to buy a dog
}
```

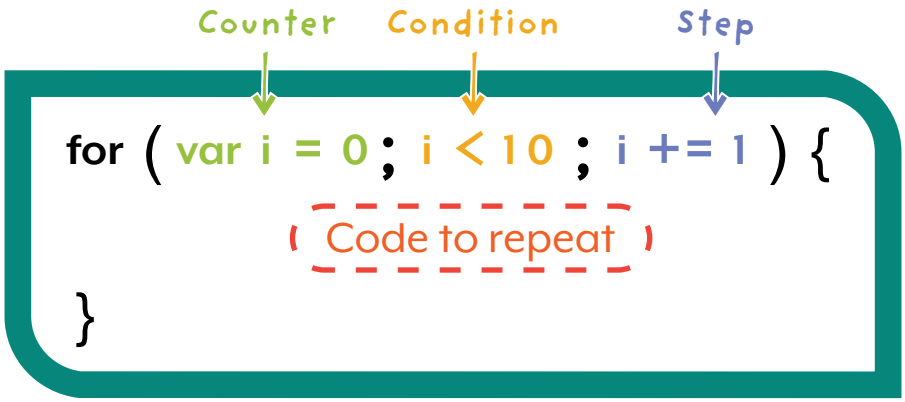
```
if ( age > 18 ) {
        I can buy a dog
} else if ( age < 18 ) {
        I need to ask my parents
        to buy a dog
}
```

```
if ( weather is not raining
    and today is Saturday ) {
        take water bottle
        play soccer outside
}
```

```
if ( weather !== raining
    && today === Saturday) {
        take water bottle
        play soccer outside
}
```

For Loop

For loops let you repeat parts of your code multiple times.



Counter

This sets the initial counter value

Condition

When the counter meets the condition, the code stops

Step

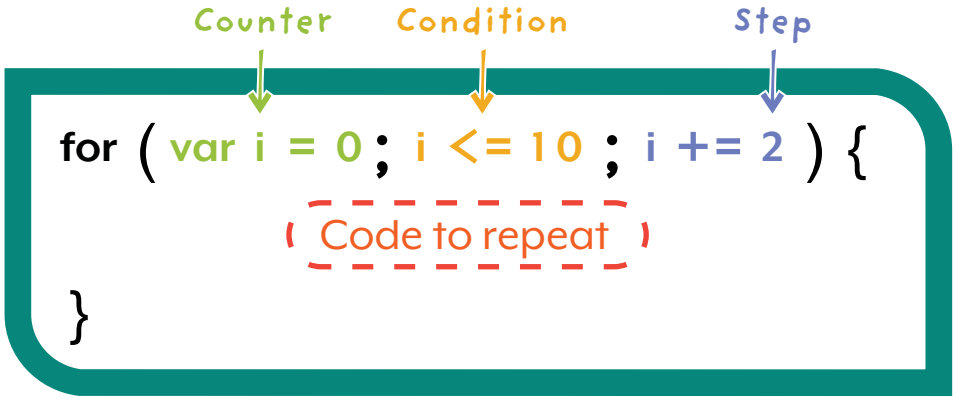
How the counter changes every time the code is repeated

1. This loop will repeat 5 times

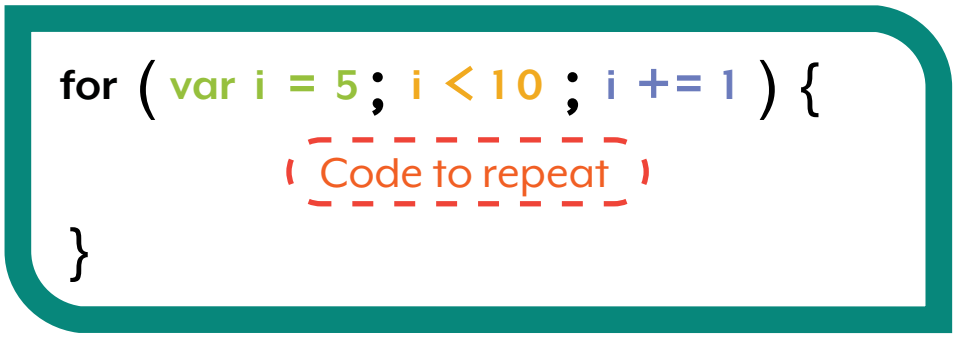
```
for ( var i = 0 ; i < 5 ; i += 1 ) {
    Code to repeat
}
```

For Loop

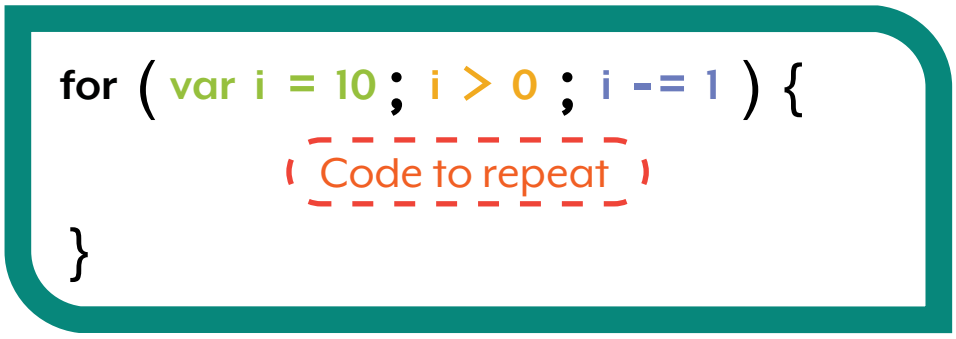
2. This loop will repeat 6 times



3. This loop will repeat 5 times

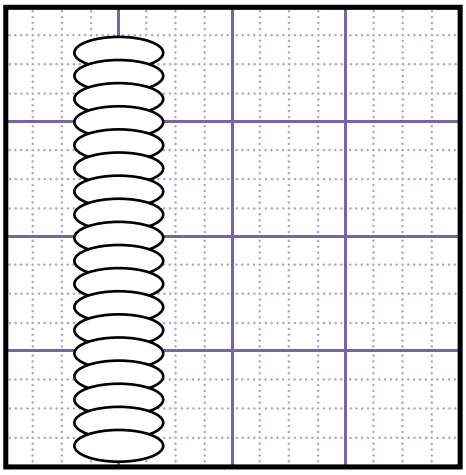


4. This loop will repeat 10 times



While Loop

This type of loop runs while the statement is true.



```
var i =10;
while (i<100){
      ellipse(130,i*4,80,30);
      i = i+5;
}
```

While loops often use booleans like true or false to check a condition.

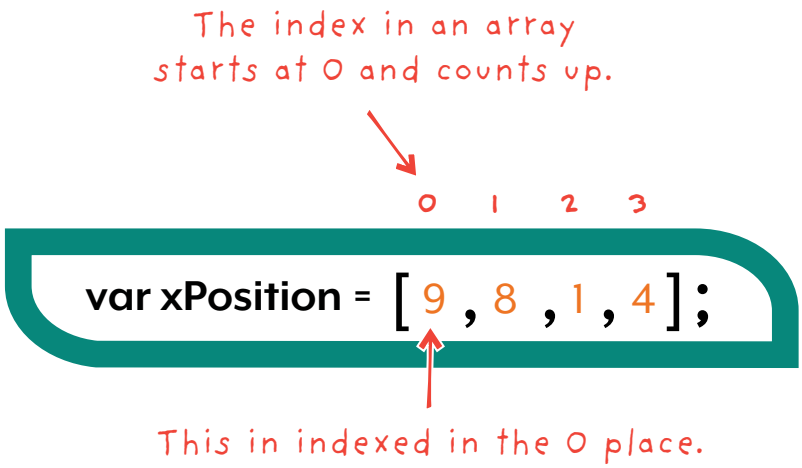
Doing

Intro to Arrays

Arrays allow you to store and access information easily. You can store anything in an array that you could store in a variable!

Create an Array

We create arrays with square brackets [ ];
Below is an array that has 4 numbers stored inside it.



The location of a value in an array is called the index.
Things stored in an array are called values or elements.

Array Types

Arrays with Information

These are arrays with information added by the coder.
These examples shows you how to create an array with words.

```
var names = [ "Kobe" , "Jill" , "Aditya" ];
```

This example is an array with numbers.

```
var nums = [ 1 , 2 , 3];
```

Don't forget your quotations around strings and images.

Empty Arrays

An empty array creates a place for information to be stored. The information is later pushed into the array during the program.

```
var things = [ ];
```

Doing

Changing Arrays

Arrays allow you to store a list of items.

```
var xPosition = [ 0 , 20 , 100 , 300 ] ;
                  0    1     2     3
```

items

index

Item

Arrays can hold any values like numbers and words

Index

Every variable gets a unique number to keep track of its position in the list

1. Changing an item

```
[ 0 , 20 , 5 , 300 ]
  0    1   2    3
```

items

index

```
xPosition [ 2 ] = 5 ;
```

2. Adding a value

```
[ 0 , 20 , 5 , 300 , 20 ]
  0    1   2    3    4
```

items

index

```
xPosition.push ( 20 ) ;
```

3. Removing a value

```
[ 0 , 20 ,  , 300 , 20 ]
  0    1      2    3
```

items

index

```
xPosition.splice ( 2 , 1 ) ;
```

Index

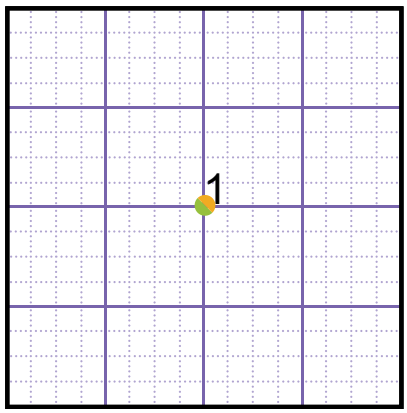
How many items to remove

Accessing Arrays

To access a specific value in an array, we put the index number in square brackets [ ] next to the name of the array. This will get indexed value from the array.

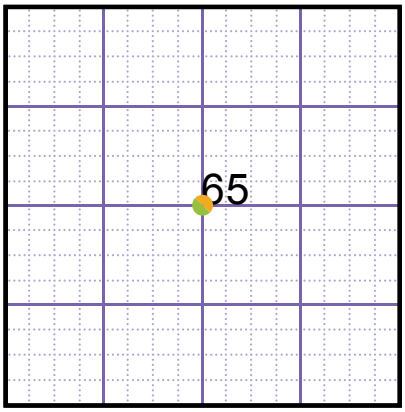
```
var nums = [1, 2, 3, 65];
           0  1  2  3
text(nums[0], 200, 200);
```

items
index



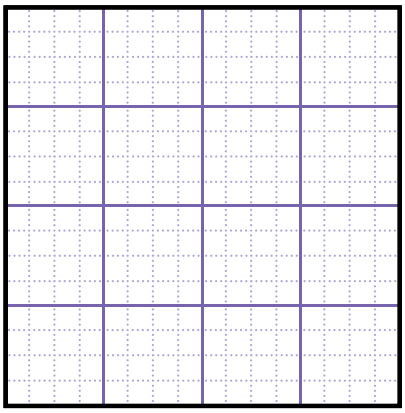

```
var nums = [1, 2, 3, 65];
           0  1  2  3
text(nums[3], 200, 200);
```

items
index




```
var nums = [1, 2, 3, 65];
           0  1  2  3
text(nums[4], 200, 200);
```

items
index



This value would be blank as there is no value stored in index 4.

Doing

Array Length

Sometimes you want your program to loop through an entire array. The .length property lets us easily do that by telling us how many elements the array has.

Array.length

We can find the length of the array by adding .length to the end of it's name.

```
var numbers = [1, 2, 3, 65];
```

For this array, number.length would equal 4.

```
var values = [1, 2, 3, "P", index];
```

For this array, values.length would equal 5.

```
var numbers = [];
```

For this array, number.length would equal 0.

Doing

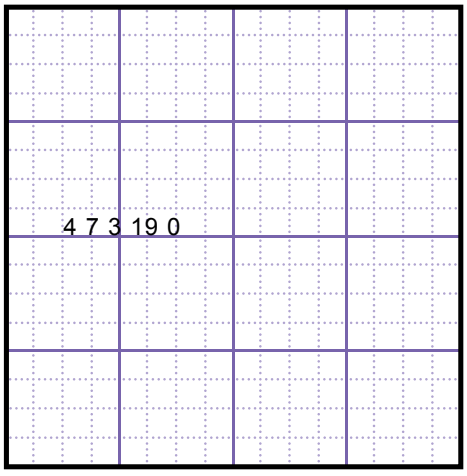
Using Array

This is an array with multiple types of data.
It has 3 numbers, a string, and a stored variable.

```
var iter = 0;
var values = [ 4 , 7 , 3 , 19 , iter];
```

Use values.length to adapt code if your array size changes
or you want to add it to code that may change.

```
for (var i = 0; i < values.length; i ++) {
    text (values [ i t e r ] , 50+i* 20 , 200);
    iter ++ ;
}
```



This for loop will work no matter how many elements are in
the array! Adaptive code is awesome!

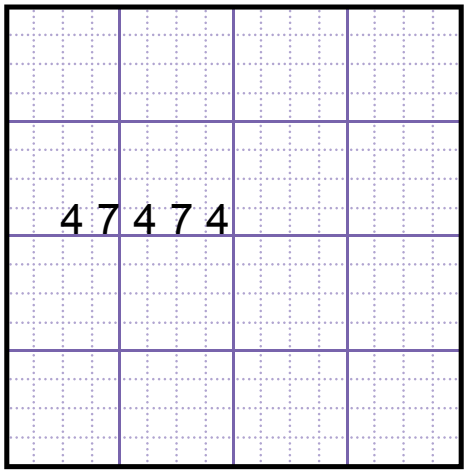
Doing

Array Patterns

We can use multiple keywords to make arrays function in different ways

```
var iter = 0;
var values = [ 4 , 7 , 3 , 19 , iter];
textSize (20) ;
for (var i = 0; i < values.length; i ++) {
    text (values [ iter ] , 50+i* 20 , 200);
    iter ++ ;
    if (iter === 2) {
    iter = 0;
    }
}
```

The code runs 5 times, which is values.length



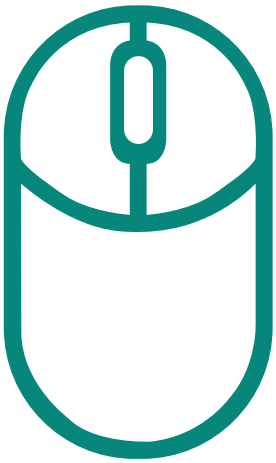
This makes it easy to set up patterns and lets users select specific choices in your code.

Mouse Functions

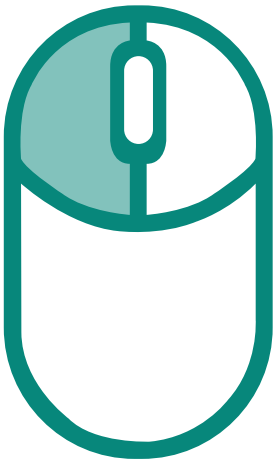
mouseClicked vs. mousePressed vs. mouseReleased

mouseClicked is almost like **mousePressed**, It only works after you let go out the mouse. **mouseReleased** works when the mouse button is let go.

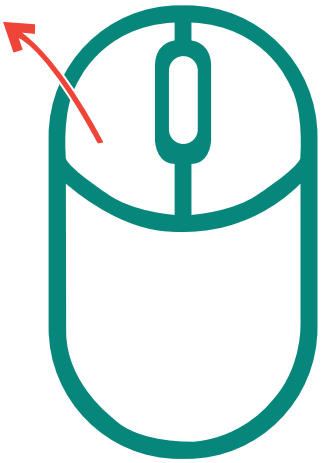
```
var mousePressed = function ( ) {
  // The code inside will run when the mouse is clicked.
};
```



Not Pressed



Pressed



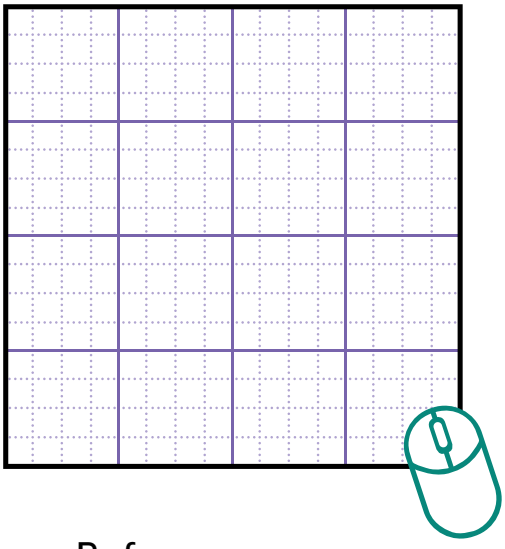
Clicked and Released

Mouse Functions

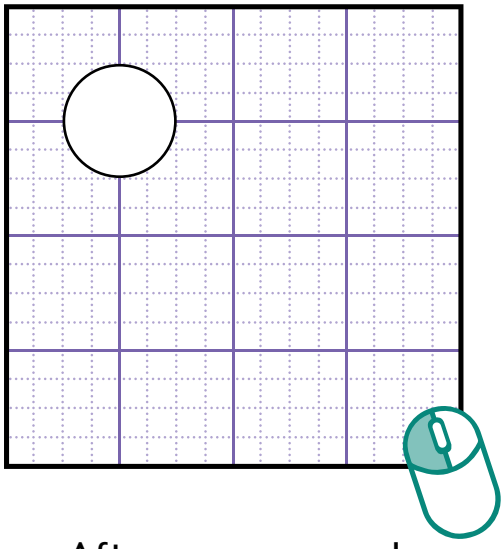
mousePressed

This makes whatever is in the function happen once after each time the mouse is pressed.

```
var mousePressed = function ( ) {
  ellipse (100 , 100 , 100 , 100);
};
```



Before you press the mouse



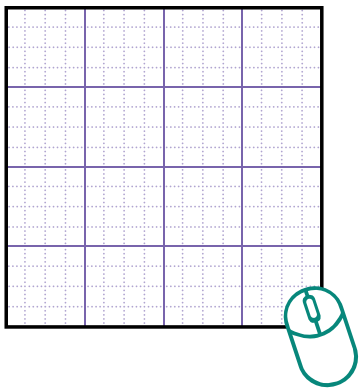
After you pressed the mouse

Mouse Functions

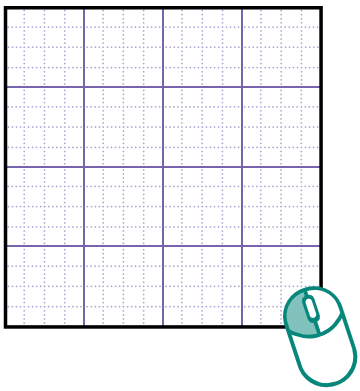
mouseClicked

The code inside the mouseClicked function will run whenever the mouse button is pressed and released.

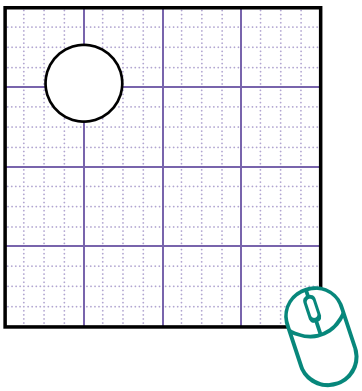
```
var mouseClicked = function ( ) {
  ellipse (100 , 100 , 100 , 100);
};
```



Before you press



While you press



After you let go

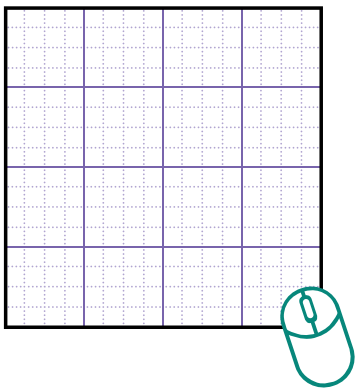
If the mouse is not pressed while the program is running, the code will not work. For example, if you press the mouse, then refresh, and let go, the program will not run.

Mouse Functions

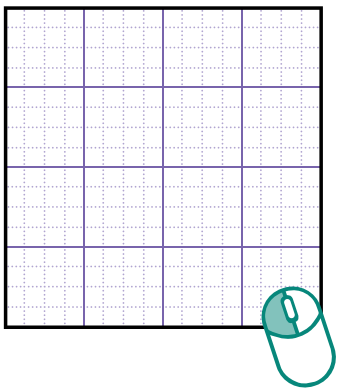
mouseReleased

This works almost the same as mouseClicked. It only responds to the release component.

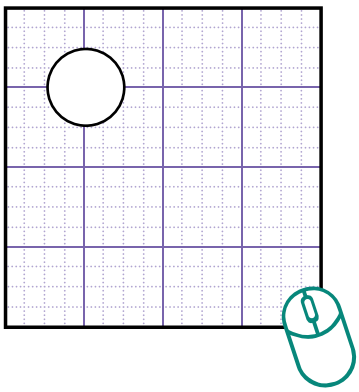
```
var mouseReleased = function ( ) {
  ellipse (100 , 100 , 100 , 100);
};
```



Before you press



While you press



After you let go

If the mouse is not pressed while the program is running, the code will not work. For example, if you press the mouse, then refresh, and let go, the program will run, unlike mouseClicked.

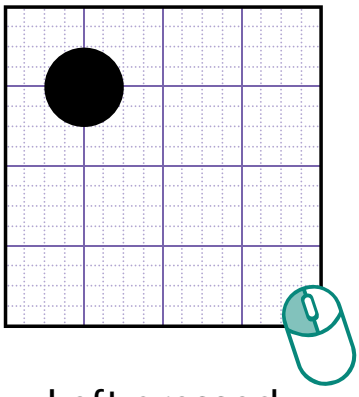
Doing

Mouse Functions

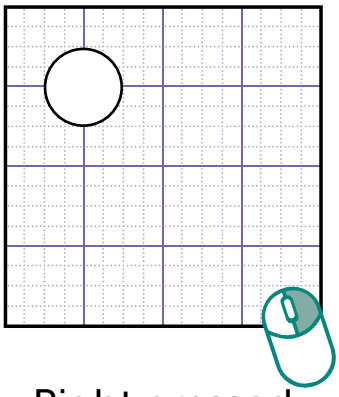
mouseButton( )

This allows something to happen when a specific mouse button is pressed.

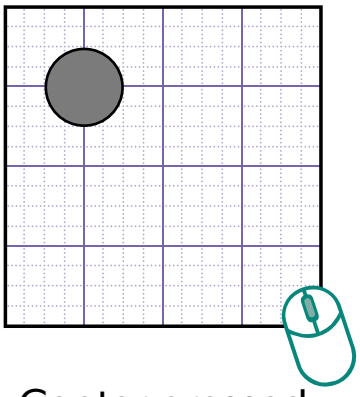
```
var draw = function ( ) {
    if (mouseButton === LEFT){
      fill (0);
    } else if (mouseButton === RIGHT){
      fill (255);
    } else if (mouseButton === CENTER){
      fill (125);
    }
    ellipse (100 , 100 , 100 , 100);
};
```



Left pressed



Right pressed



Center pressed

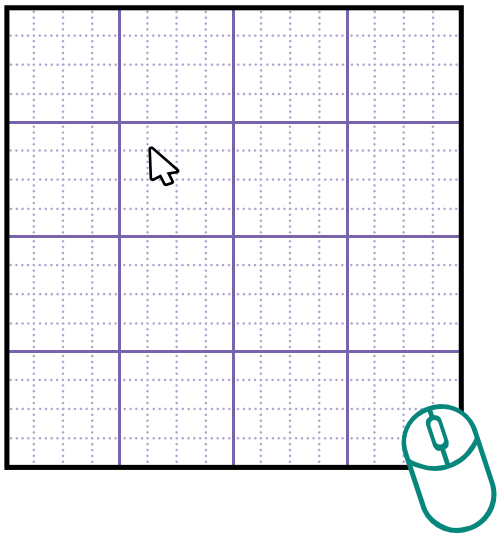
Mouse Functions

mouseMoved( )

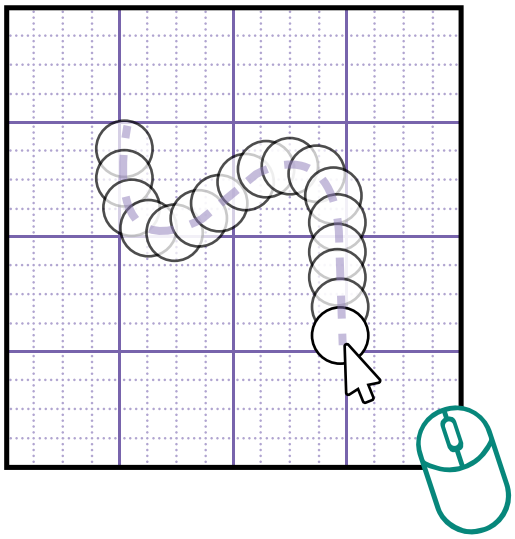
This function runs every time the mouse is moved.

```
var mouseMoved = function ( ) {
  ellipse ( mouseX , mouseY , 50 , 50 );
};
```

These allows the ellipse to move with the mouse



Before the mouse is moved



While the mouse is moved

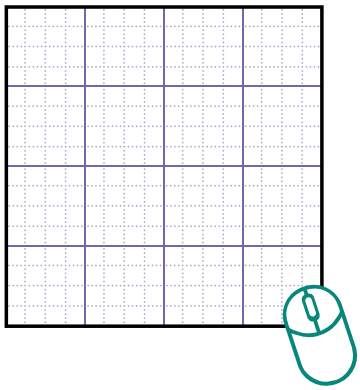
Mouse Functions

mouseDragged ( )

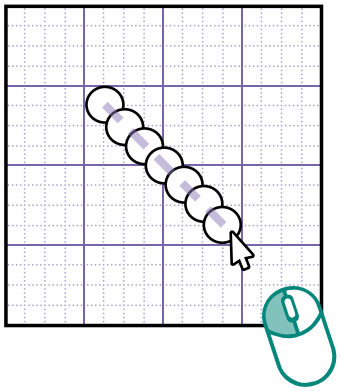
This function runs every time the mouse moves while the button is pressed.

```
var mouseDragged = function ( ) {
  ellipse ( mouseX , mouseY , 50 , 50 );
};
```

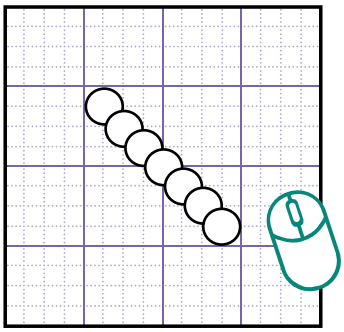
This will work whether you press the LEFT, CENTER, or RIGHT button.



Before you pressed the mouse button



While you press the mouse button and it moves



After the mouse button is released

This function only runs when the mouse button is pressed down AND the mouse is moving! If only one of those happens and not the other, the function will not run.

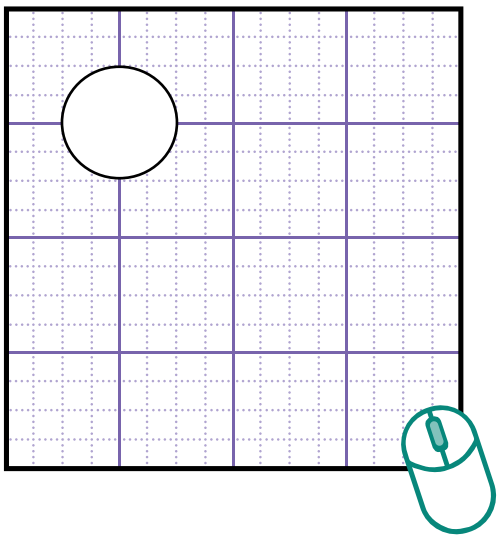
Mouse Functions

mouseScrolled ( )

This calls the function when the user scrolls with their mouse.

// Heads up! This will also scroll the page up or down if the window is not maximized.

```
var mouseScrolled = function ( ) {
  ellipse ( 100 , 100 , 100 , 100 );
};
```



This is shown when the mouse is scrolled

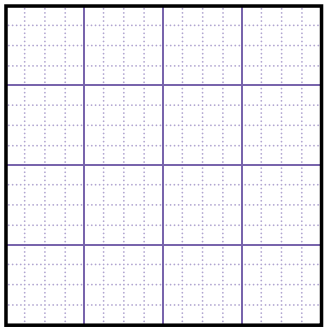
Keyboard Functions

You can use functions to check if they keyboard is being used.

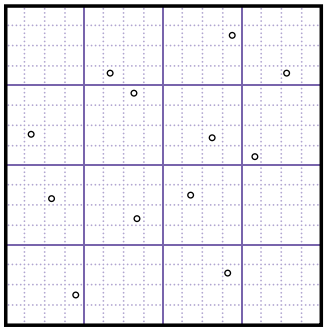
KeyPressed

The contents of the keyPressed function runs while any key is pressed down.

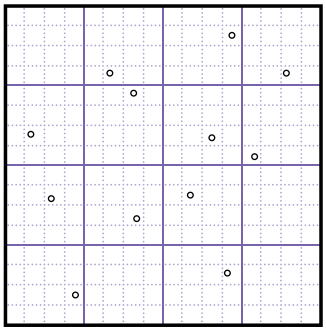
```
var keyPressed = function ( ) {
  ellipse ( random ( 0 , 400 ) , random ( 0 , 400 ) , 10 , 10 );
};
```



When a key is pressed



When a key is pressed for 5 seconds



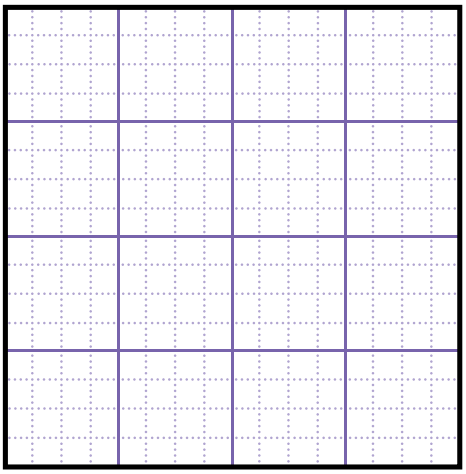
When a key is released

Keyboard Functions

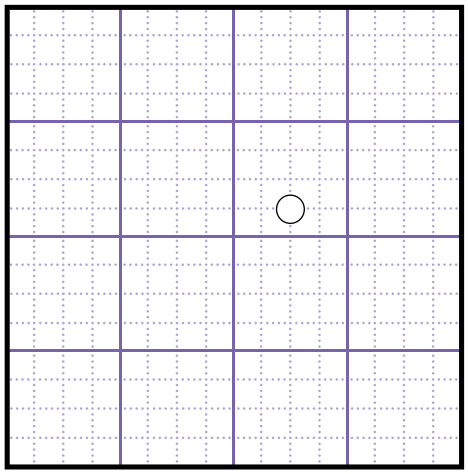
KeyReleased

The contents of the KeyReleased function runs when any key is released

```
var keyReleased = function ( ) {
  ellipse ( random ( 0 , 400 ) , random ( 0 , 400 ) , 25 , 25 );
};
```



When a key is pressed



When a key is let go

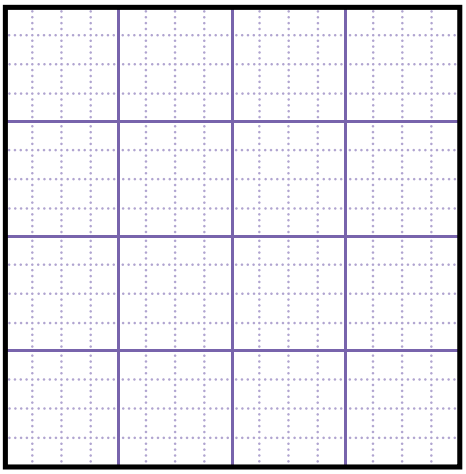
This will work only when the key is let go.

Keyboard Functions

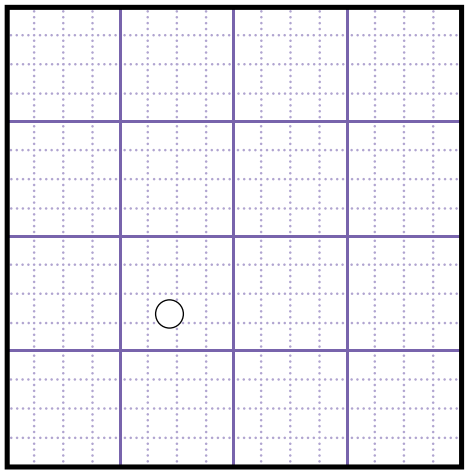
Key Typed

This works similarly to KeyReleased. It needs the user to press and released a key for it to work.

```
var keyTyped = function ( ) {
  ellipse ( random ( 0 , 400 ) , random ( 0 , 400 ) , 25 , 25 );
};
```



When a key is pressed



When a key is let go

This only works when the key is pressed and let go.

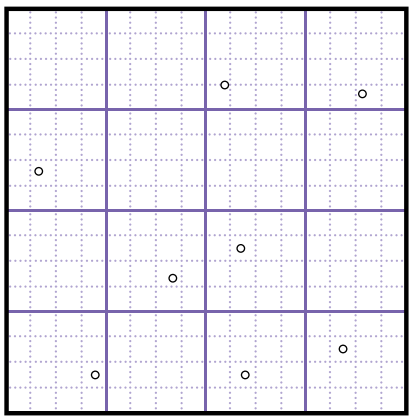
Keyboard Functions

Use a specific key to add functionality to your project.

Key Codes

You can use keyCodes (numbers that represent specific keys) or words that detect special keys.

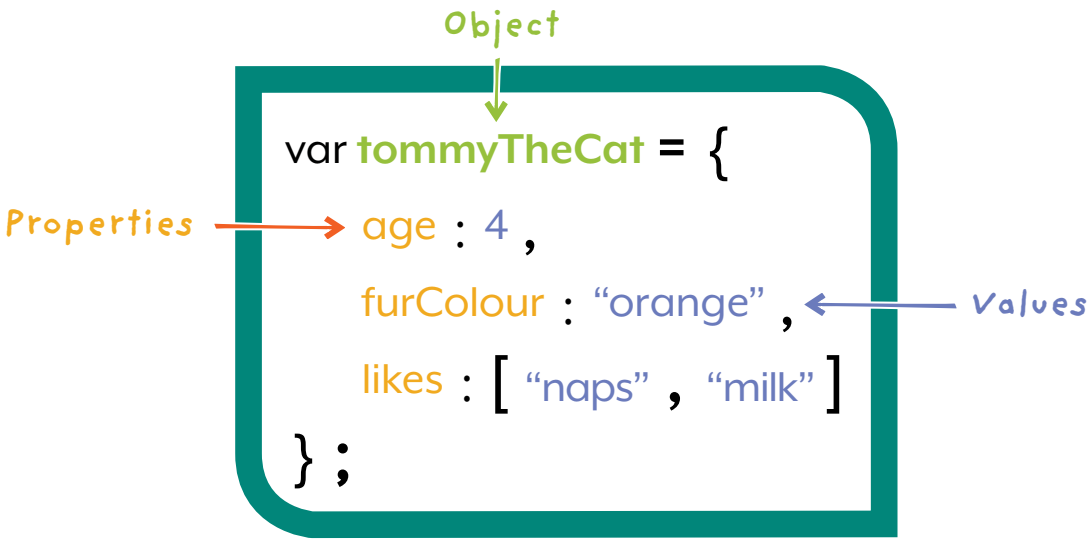

```
var keyPressed = function ( );
  if ( keyCode = = = 49 ) {
  ellipse ( random ( 0 , 400 ) ,
            random ( 0 , 400 ) ,
            10 , 10 );
}
```



Dots will draw when "1" is pressed, but not when any

Object Literals

An **object** is like a variable that holds multiple **values**, and these **values** can be thought of as the variable's **properties**.



Accessing Object Properties

To access an object's property, we use **dot notation**.

```
text ( tommyTheCat.age , 200 , 200 );
```

The number 4 will be display in the canvas

Modifying an Object Literal

We can also use dot notation to change an object's properties.

```
tommyTheCat.age = 5 ;
```

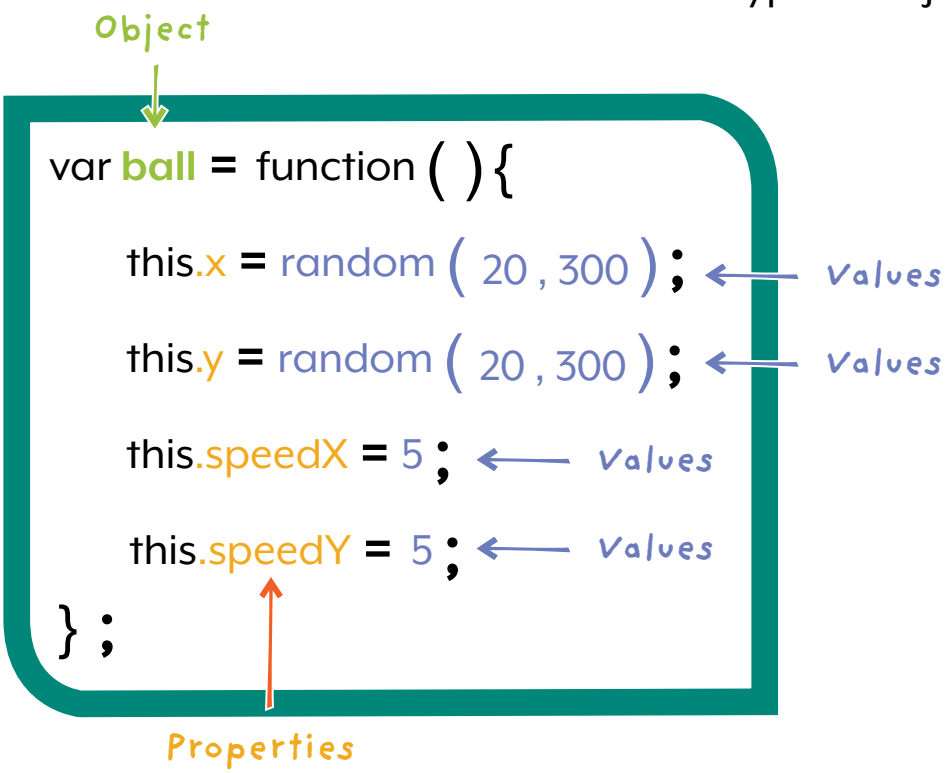
Changes tommy's age to 5

Doing

Object Prototypes

Object Prototype

Object Literals let us create one object and that's it. Object Prototypes let us create more than one of the same type of object.



Constructors

Constructors are what we use to make a **new instance** of the Object Prototype.

```
var myBall = new ball ( ) ;
myArray.push ( new ball ( )) ;
```

Doing

Using Objects

We can use Object Literals for anything we want to create!
Here we can use this Object Literal to create a car.

Object

```
var car = {
    wheels : 4 ,
    seats : 5 ,
    colour : [random(0,255) ,
              random(0,255) ,
              random(0,255)] ,
} ;
```

Values

Properties

This car was randomly selected to be green because the colour is random

Part 1

But what if we wanted to make more than one car? In that case, we can easily turn the Object Literal into an Object Prototype!
Prototypes allows you to make a pattern to create multiple objects

```
var car = {
```

To

```
var car = function ( parameters ){
```

How do you use it? With a constructor.
This will build your new types of cars!!

Doing

Using Objects

Part 2

Now imagine you want to make a sports car! You already know how it will look and you want to set the variables when you create it! It still follows a pattern of a car and has wheels, seats, and a colour.

```
var car = function ( wheels, seats, colour ){
    this.wheels = wheels;
    this.seats = seats;
    this.colour = colour;
};
```

You want it to have 4 wheels, 2 seats and be yellow!
Create it using the code below.


```
var sportscar = new car ( 4 , 2 , [255 ,  255 ,  0]);
```

Send in **parameters** to our Object Prototype!

Doing

Using Objects

Part 3

Let's say we want to make a van, but we don't know about it. We can still make a new instance of our Object Prototype without parameters."

```
var van = new car ( );
```

Vans are usually bigger. It should have 7 seats.
We will change the value of seat by doing the following.

```
van.seats = 7;
```

Now your van style car has 7 seats!

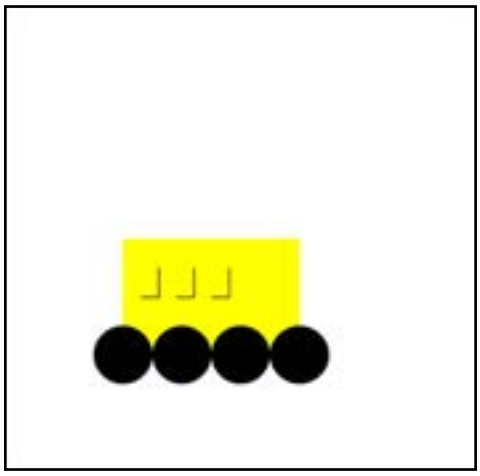
Doing

Using Objects

Part 4 - Code this yourself!

```
var car = function ( wheels , seats , rgb ){
    this.wheels = wheels;
    this.seats = seats;
    this.color = rgb;
};

var sportscar = new car ( 4 , 3 , [ 255 , 255 , 0 ] );
```



```
noStroke ( ) ;
fill ( sportscar.color [ 0 ] , sportscar.color [ 1 ] , sportscar.color [ 2 ] );
rect ( 100 , 200 , 150 , 100 ) ;
for ( var numWheels = 0 ; numWheels < sportscar.wheels ; numWheels ++ ) {
      fill ( 0 ) ;
      ellipse ( numWheels *50 +100 , 300 , 50 , 50 ) ;
}
for ( var numSeats = 0 ; numSeats < sportscar.seats ; numSeats ++ ) {
      stroke ( 0 ) ;
      line ( numSeats * 30 +115 , 250 , numSeats *30 +130 , 250 ) ;
      line ( numSeats * 30 +130 , 225 , numSeats *30 +130 , 250 ) ;
}
```

Doing

Object Methods

Objects with Functions

Objects with properties are great, but what if we want our object to **do** something?
That's where Object Methods come in!

Object Literal Method

Object methods are functions that can easily use the object's properties. They are declared just like the other properties and let each object instance act on its own, without any extra work from you!

```
var ball = {
    x : 10 ,
    move : function ( ) {
        ball.x += 1 ;
    } ,
} ;
```

Since Object Literals only create one instance, we use the Object Literal's name and dot notation to access its properties

Object Methods

Object methods are functions that affect or act on an object. Each object created with the prototype will be linked to the object method created.

Object Prototype Method

This method will act on the car object prototype.

```
var balls = function ( ) {
    this.x= 10 ;
    this.move= function ( ) {
        this.x += 1 ;
    } ;
} ;
```

Cursor

You can change the cursor seen on the canvas with the cursor function.

There are many different types of cursors. Change the parameter to use the one you want!

Using Cursor( );

```
cursor (MOVE);
```

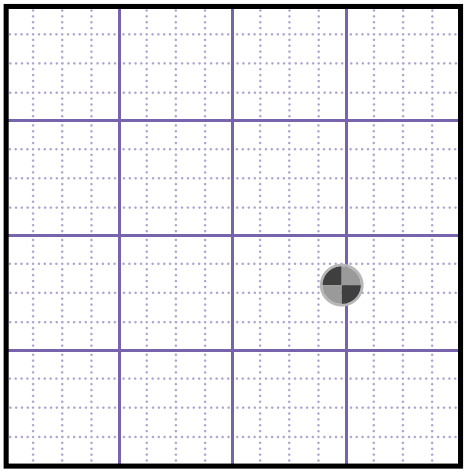
MOVE	
ARROW	
CROSS	
HAND	
TEXT	
WAIT	

Make sure the parameter is in capital letters!

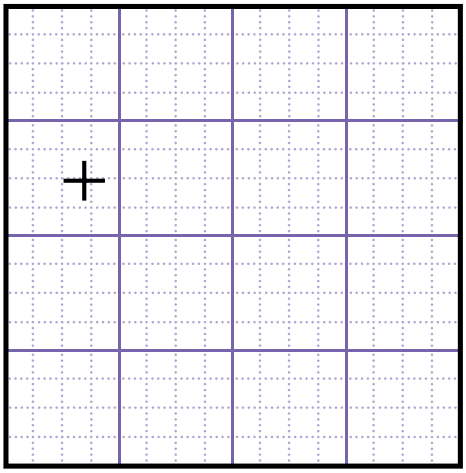
Cursor

Using Cursor( );

```
var draw = function ( ) {
  if ( mouseX > 100 ){
      cursor (WAIT) ;
  }else {
      cursor (CROSS) ;
  }
};
```



When mouseX
is > 100



When mouseX
is not > 100

Switch

This works like an if else statement. The cases are checked against the parameter in the switch() statement.

```
var keyPressed = function ( ) {
  switch ( key + 0 ){
    case 49 ;  // the 1 key
    ellipse ( 100 , 100 , 100 , 100 ) ;
    break ;
    case 50 ;  // the 2 key
    ellipse ( 200 , 200 , 100 , 100 ) ;
    break ;
    default ;  // any other key
    ellipse ( 300 , 300 , 100 , 100 ) ;
  }
} ;
```

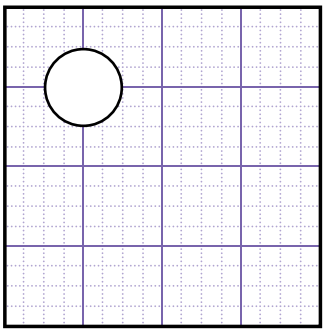
This converts our key into its keyCode so we can check it against our cases.

This is the "1" button

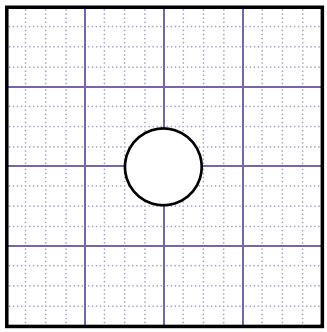
This is the "2" button

This is all other keys

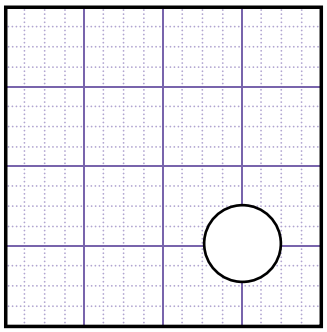
You need to include break; at the end of each case.



If 1 is pressed



If 2 is pressed



If anything else pressed

Using Trigonometry

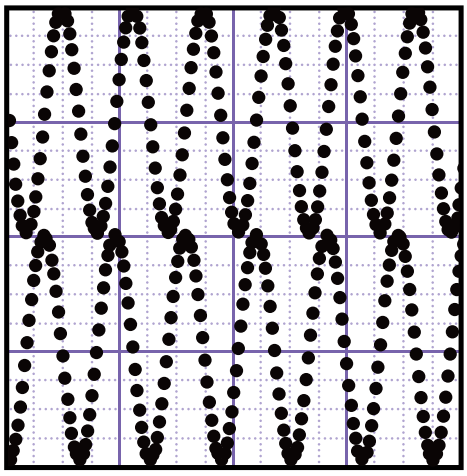
There are several trigonometry functions!
This is a part of math. These let you animate in wavy patterns!

Sin and Cos

These are both wavy functions! Sin starts at $(0, 0)$ and Cos starts at $(1, 0)$

Sin
This is the sine function

Cos
This is the cosine function



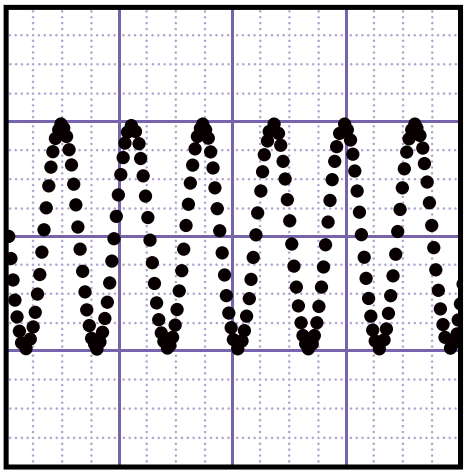
Sin and cos make a value increase to 1 then decrease to -1.
This happens forever!

If you are familiar with these functions, they are inverted.
This is because the y-values start at the top.
In math they normally start at the bottom!

Using Trigonometry

Use this to draw a moving picture!

Sin

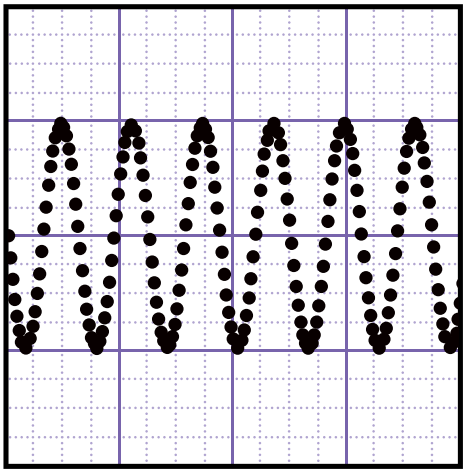


```
var shiftX = 0;
fill(255);
var draw = function() {
    ellipse(shiftX*10, sin(x)*100+200, 10, 10);
    shiftX += 0.1;
};
```

Using Trigonometry

Use this to draw a moving picture!

Cos



```
var shiftX = 0;
fill(255);
var draw = function() {
    ellipse(shiftX*10, cos(x)*100+200, 10, 10);
    shiftX += 0.1;
};
```

Prompt

If you want the user to input text for your program, you can use the prompt function. Prompt takes in two parameters: what the user is inputting, and an example input.
You can store the result to a variable to print it later!

```
var ex = prompt("Question", "Sample Answer");
```

```
var mouseClicked = function() {
    background(255, 255, 255);
    var name = prompt("Write Your First Name",
                      "Hatch Student Name");
    text(name, 200, 250);
};
```

When the user clicks their mouse, a box below appears on their screen. Whatever the user writes is stored in the variable called name.

NOTE: Prompt works better in mouseClicked function compared to in draw functions. Draw functions keep calling the prompt and your program becomes difficult to use.

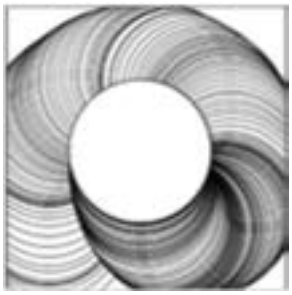
Frame Rate

Normally we want our program to run quickly and smoothly. The default frame rate is 60 frames per second, but you can change it using the frameRate function!

Changing the FrameRate

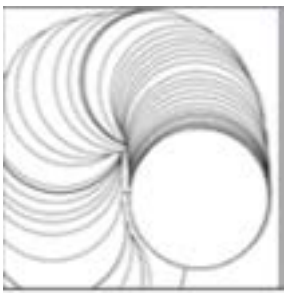
Increasing the frame rate will not significantly change the way the program runs.

```
var draw = function ( ) {
      frameRate (120);
      ellipse (mouseX , mouseY , 200 , 200 );
} ;
```



Decreasing the frame rate will make the draw function refresh more slowly, which can lead to some cool animations!

```
var draw = function ( ) {
      frameRate (10);
      ellipse (mouseX , mouseY , 200 , 200 );
} ;
```



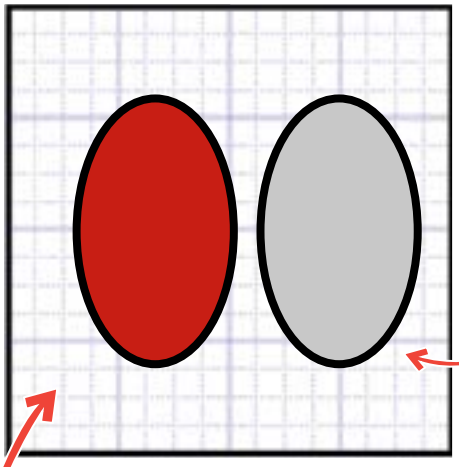
Pull Color - Red

You can extract a section of colour using the functions red, green, blue.

red( ); Example

The red function will extract the red value from a colour. This is the first of the RGB values.

When all values are the same fill. The colour will appear gray!



Fills with the value from the red in all arguments. (200,200,200) or (200)

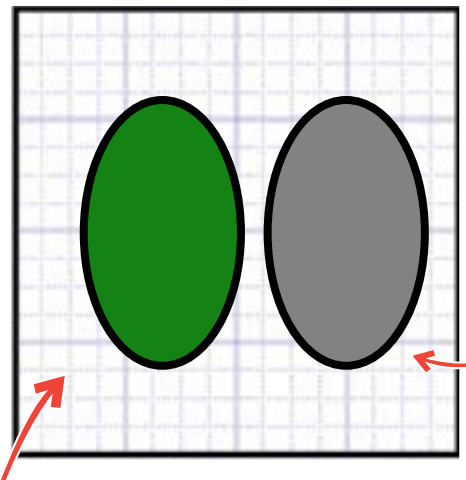
```
var r = color(200, 30, 20);
fill(r);
ellipse(135, 200, 150, 225);
var redValue = red(r);
fill(redValue);
ellipse(300, 200, 150, 225);
```

Pull Color - Green

You can extract a section of colour using the functions red, green, blue.

green( ); Example

The green function will extract the green value from a colour. This is the first of the RGB values.



When all values are the same fill. The colour will appear gray!

Fills with the value from the green in all arguments. (130,130,130) or (130)

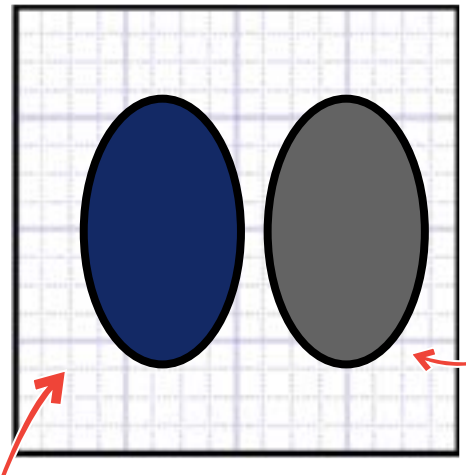
```
var g = color(20, 130, 20);
fill(g);
ellipse(135, 200, 150, 225);
var greenValue = green(g);
fill(greenValue);
ellipse(300, 200, 150, 225);
```

Pull Color - Blue

You can extract a section of colour using the functions red, green, blue.

blue( ); Example

The blue function will extract the blue value from a colour. This is the first of the RGB values.



When all values are the same fill. The colour will appear gray!

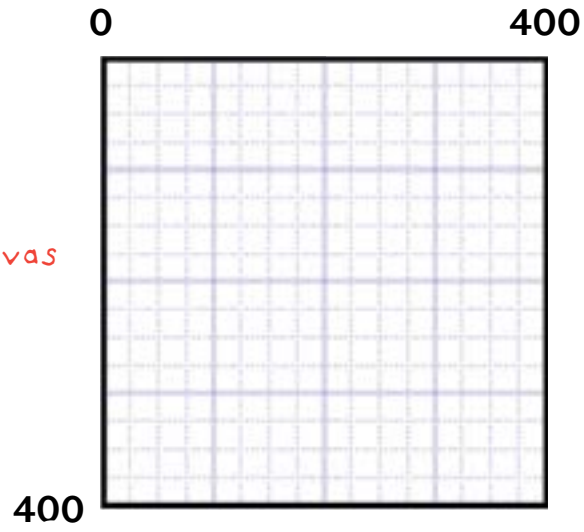
Fills with the value from the blue in all arguments. (100,100,100) or (100)

```
var b = color(20, 40, 100);
fill(b);
ellipse(135, 200, 150, 225);
var blueValue = blue(b);
fill(blueValue);
ellipse(300, 200, 150, 225);
```

Canvas

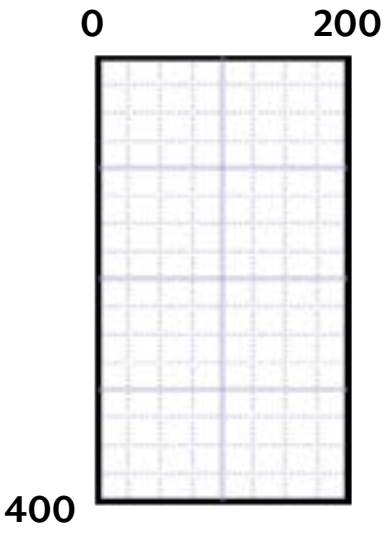
The canvas is where you see your code. It is a type of Matrix, this is a 2D Array.

Our normal Canvas is



You can change the size of the canvas using the size keyword.

```
size ( 200 , 400 ) ;
```



Change the Canvas

You can change the size, location and orientation of objects by changing the canvas.

Canvas Changing Functions

```
rotate ( degrees ) ;
```

Rotate the canvas in degrees

```
translate ( x , y ) ;
```

Translate the canvas in pixels

```
scale ( multiple ) ;
```

Change the scale with a number.

Bigger than 1 will increase the size, smaller than 1 will decrease the size.

Canvas Reseting Functions

```
resetMatrix ( ) ;
```

Removes all canvas changes

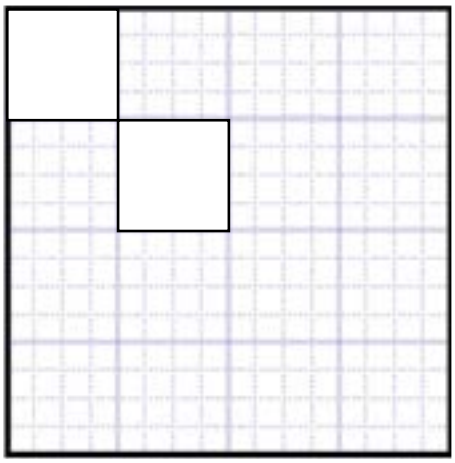
```
pushMatrix ( ) ;
popMatrix ( ) ;
```

pushMatrix saves any changes to the canvas, and popMatrix puts the canvas back to how it was when it was saved.

Rotate the Canvas

Using rotate will rotate the matrix.
The matrix is also known as the Canvas.

```
for (var i = 0; i < 5; i ++) {
    rect(0, 0, 100, 100);
    rect(100, 100, 100, 100);
};
```

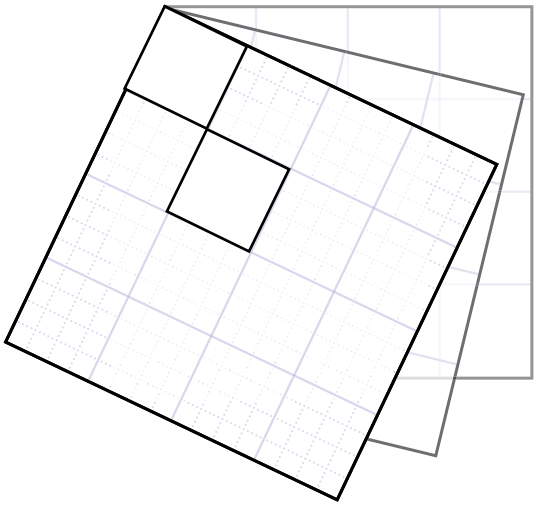


You can only see the two rectangles as the loop draws them on top of each other.

How it works

We can add rotate into the for loop to rotate the canvas.

```
rotate(6);
```

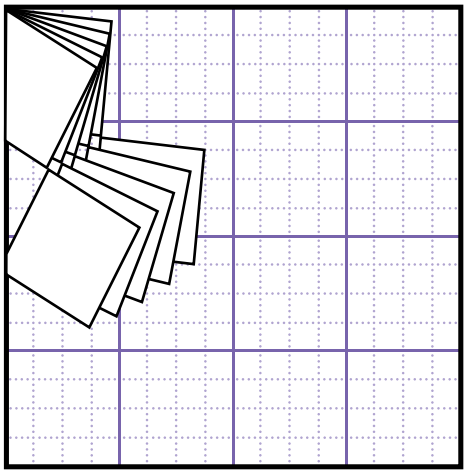


The canvas or Matrix rotates around the (0,0) point on the canvas. This is also called the origin.

Rotate Example

You can use rotate in a for loop to make a cool design or animation!

Code to Rotate



The second square looks like it is rotating in a big circle.
This is because the canvas or matrix is rotating not the square.

```
for (var i = 0; i < 5; i ++) {
    rotate(6);
    rect(0, 0, 100, 100);
    rect(100, 100, 100, 100);
};
```

This will rotate and print each square. The loop runs 5 times and 5 squares are printed!

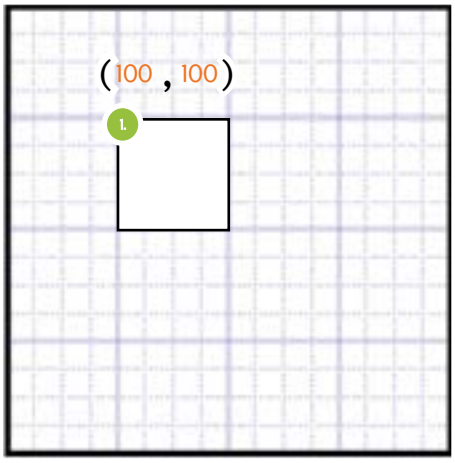
Translate

You can use the translate function to move the (0, 0) point on the canvas to a new (x, y) location.

Translate

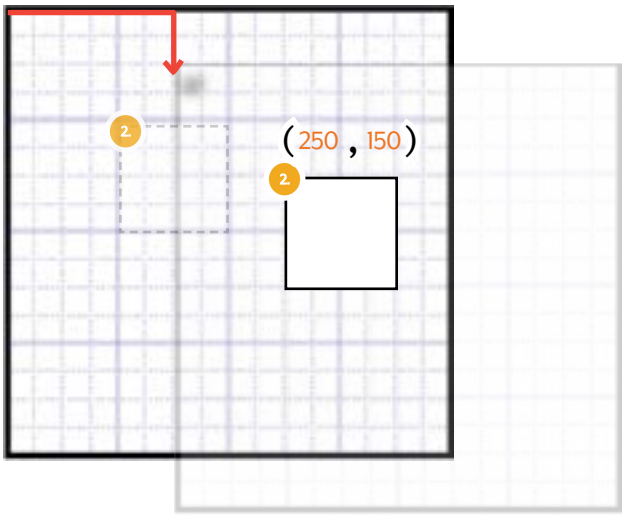
Translate moves the canvas in the X and Y direction.

```
1. rect ( 100 , 100 , 100 , 100 ) ;
```



The faded shape at 100, 100. The canvas moved 150 to the right and 50 down!

```
2. translate ( 150 , 50 ) ;
   rect ( 100 , 100 , 100 , 100 ) ;
```



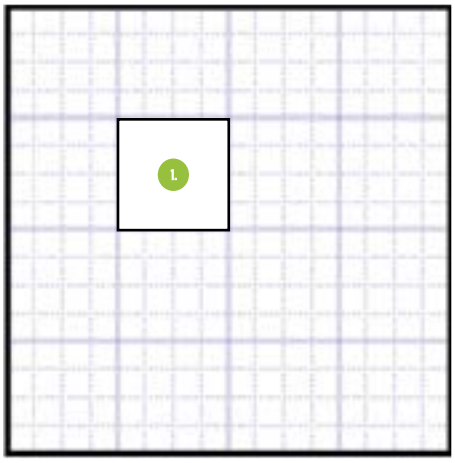
Scale

You can also change the size of the canvas and make objects look bigger or smaller.

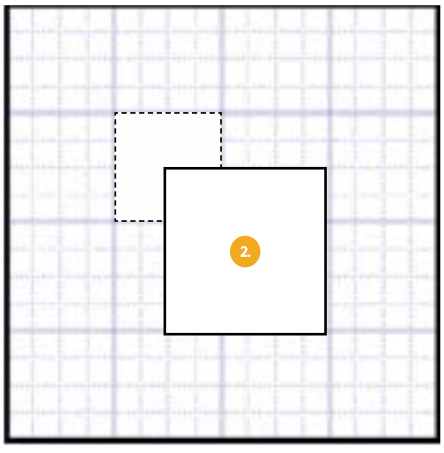
Scale

Scale makes the canvas larger starting from the lop left corner.

```
1. rect ( 100 , 100 , 100 , 100 ) ;
```



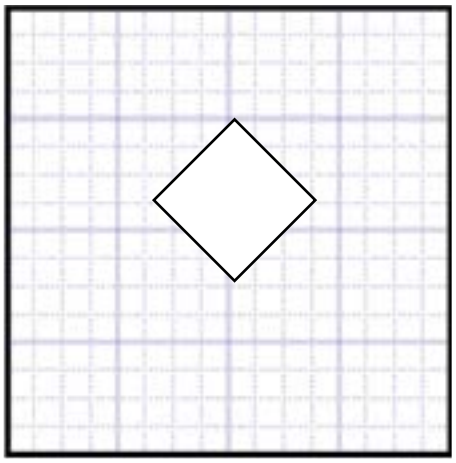

```
2. scale ( 1.5 ) ;
   rect ( 100 , 100 , 100 , 100 ) ;
```



Reset Matrix

Reset Matrix removes all changes currently impacting the matrix.

```
translate(255, 100);
rotate(45);
rect(0, 0, 100, 100);
```

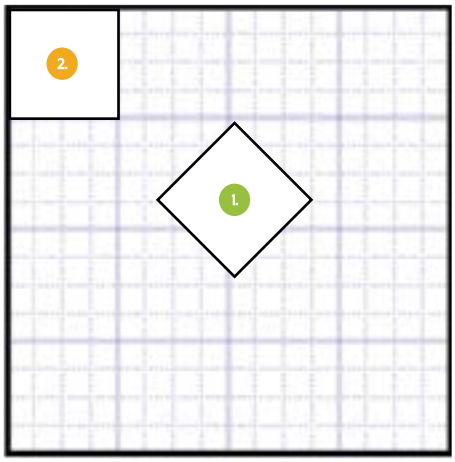


All lines following Matrix change impact in all following lines!

resetMatrix( );

Even if both rectangles have the same parameters they can act differently!

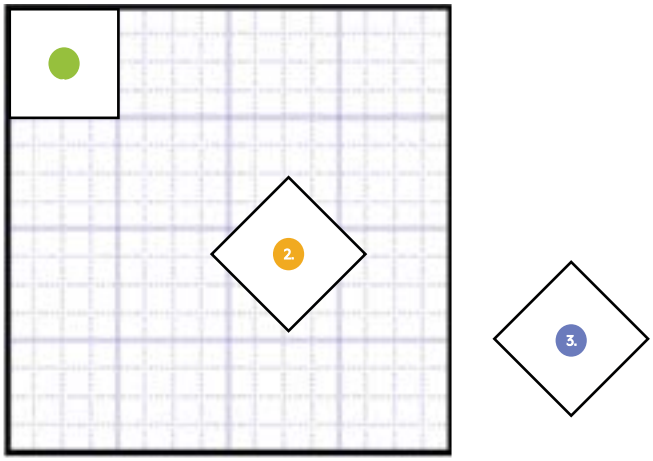
```
translate(255, 100);
1. rotate(45);
   rect(0, 0, 100, 100);
   resetMatrix( );
2. rect(0, 0, 100, 100);
```



Push/Pop Matrix

Using pushMatrix and popMatrix allows you to save and restore changes to the canvas.

```
1. rect(0, 0, 100, 100);
   translate(255, 150);
2. rotate(45);
   rect(0, 0, 100, 100);
3. rect(300, 300, 100, 100);
```



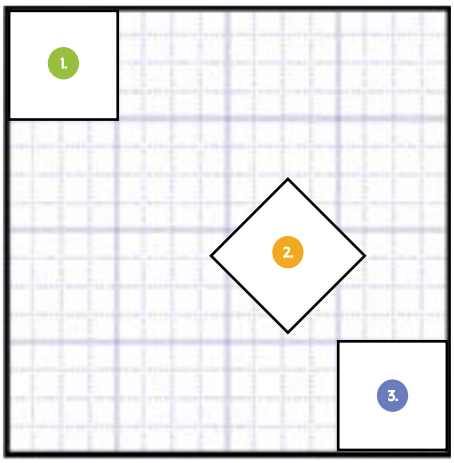
Both rectangles 2. and 3. are rotated and translated.

Push and Pop Matrix ( );

Only the orange section impacted by the Matrix Changes

```
1. rect(0, 0, 100, 100);
   pushMatrix( );
   translate(255, 150);
   rotate(45);
   rect(0, 0, 100, 100);
   popMatrix( );
3. rect(300, 300, 100, 100);
```

2.



Convert

Translating Variables

Translating code from other languages is an important skill. You can also try to convert syntax from Hatch.js to Hatch.py

Variables

TYPE	PARAMETERS
boolean	true, false
float	decimal numbers
int	whole numbers

If you use processing documentation, or you code in a different code editor, you may need to change how you define your variables.

```
int
```

To

```
var
```

```
color red = color(26);
```

To

```
var red = color(26);
```

Pseudocode to Code

Moving to Pseudocode can be hard!
You need to learn how to convert ideas into code!
The pseudocode is the written format of code.

Code Blocks in Pseudocode

This is the text that is directly used in the TWYS.
These are the specific numbers or keywords used in the TWYS.

Set the fill colour to `71, 173, 12`

These are code blocks.

Create a `draw` function.

Set the background colour to `0, 130, 196`

The indentation shows that **background** is inside the draw function.
This can help you think about if you need curly brackets!

Convert

Pseudocode to Code

Translate Pseudocode to Code

Use these steps to help you if you are having trouble!

1. What are the **keywords** or **functions** in the **pseudocode**.
2. What is the **syntax** of the **keyword** or **function.**
3. What **numbers** / **parameters** should I use.

Set the fill colour to `71, 173, 12`

Using the Steps

1. The **keyword** for this line is fill.
2. The **syntax** for this word is fill(#, #, #);
3. The numbers I should use are the ones given.

```
fill (71, 173, 12);
```

Convert

Pseudocode to Code

Pseudocode Translation Examples

Having trouble with a specific line of pseudocode?
Use these examples to help you translate pseudocode!

CODE TYPE	PSEUDOCODE	TWYS
Shapes	Draw an ellipse at the position 50 , 50 with the size 50 , 50.	ellipse(50 , 50 , 50 , 50) ;
Variable	Declare a variable box and assign it to the value 0	var box = 0 ;
Colours	Set the fill value to red	fill(255 , 0 , 0) ;
Draw Function	Create a draw function	var draw = function() { } ;
If Statement	Create an if statement that triggers when mouseX is less than 150	if(mouseX < 150) { } ;

You may want to use other Reference Manual Pages and past projects to help you with other specific examples!

Convert

Pseudocode to Code

Pseudocode Translation Examples

Having trouble with a specific line of pseudocode?

CODE TYPE	PSEUDOCODE	TWYS
Loops	Create a for loop that begins with counter **i** equal to 0, and for each loop where **i** is less than 4, **i** increases by 1	for(var **i**=0 ; **i**< 4 ; **i**++){ }
Object Literal	Create Dot as an object	var Dot = { x: 50, y: 50, } ;
Images	Draw the image "starwars/c3po" at position 10, 10 with a size of 100, 100	image(getImage ("starwars/c3po"), 10, 10, 100, 100) ;
Mouse	At position mouseX	mouseX

You may want to use other Reference Manual Pages and past projects to help you with other specific examples!

Convert

TWYS to Pseudocode

Pseudocode Translation Examples

Having trouble with a specific line of pseudocode?

CODE TYPE	PSEUDOCODE	TWYS
Array	Create an array myNums and assign it a list of numbers between 1 and 3	var myNums = [1, 2, 3];
Translate	Translate the canvas by 100, 100	translate(100, 100);
Modes	Set the ImageMode to Center	imageMode(CENTER);
Random	Set to a random number between 1 and 100	random(1, 100);

You may want to use other Reference Manual Pages and past projects to help you with other specific examples!

Convert

JS to Python

Big Difference Between JS and PY

JS

1. Bad indentation will break your project in PY.

PY

```
var draw = function ( ) {
ellipse ( 100 , 100 , 100 , 100 ) ; } ;
```

In JavaScript, indentation and spacing won't break your code.

```
def draw ():
ellipse ( 100 , 100 , 100 , 100 )
```

Python code won't run because the indentation is off.

2. Less brackets and semicolons in PY

```
var draw = function ( ) {
ellipse ( 100 , 100 , 100 , 100 ) ;
} ;
```

There are more brackets and punctuation in JavaScript

```
def draw ( ) :
  ellipse ( 100 , 100 , 100 , 100 )
```

Looks wrong but in Python we use less brackets and semicolons

3. You must call global variables

If you use global variables in your project you need to tell the program you will use or alter them inside your specific functions.

```
def draw():
    global chompSpeed, pacMouth, pacMouthClose
```

Convert Processing Docs

Using the processing.js or processing.py documentation may be hard. Here are some tips to help you understand how to use this in the Hatch Studio.

Void

Void is just how to define a function. In the Hatch Studio we use var _functionName_ = function( );

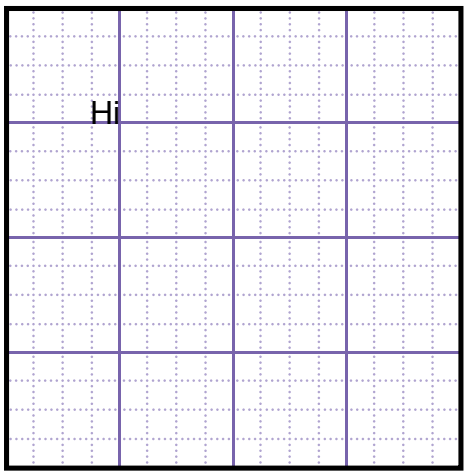
Processing.js

```
void draw ( ) {
    text ( "Hi" , 100 , 100 ) ;
}
```

Note there is NO semicolon

Hatch Studio

```
var draw = function ( ) {
    text ( "Hi" , 100 , 100 ) ;
} ;
```



Both will show this!

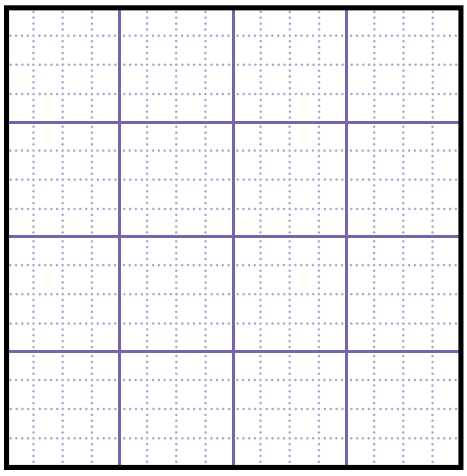
Convert Processing Docs

In processing.js normally you need to call a canvas and set up the background. Hatch does this for you.

Setup

To convert to processing.js to use in another compiler you will need to set up a canvas.

```
void setup ( ) {
      size ( 400 , 400 ) ;
}
```



This would create the base canvas used in the Hatch IDE