Key Concept: Learn how to use variables!
Table of Contents

Colouring

Page 6
 Colour Basics

Pages 7-8
 Colour Wheels
  RGB Colour Wheel
  Skin Tones

Pages 9-13
 Background, Fill, Stroke
  Order of Colours
  Background
  Fill
  Stroke
  noFill and noStroke

Pages 14-15
 Colour Modes
  HSB Colour Mode
  RGB Colour Mode

Key Concepts: Learn about Colours!
Key Concept: Learn how to add and format shapes, text, lines and images to your projects.
<table>
<thead>
<tr>
<th>Pages</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-25</td>
<td>Lines</td>
</tr>
<tr>
<td></td>
<td>Lines, Bezier</td>
</tr>
<tr>
<td>26</td>
<td>Text</td>
</tr>
<tr>
<td>27-28</td>
<td>Images</td>
</tr>
<tr>
<td></td>
<td>Hatch Image</td>
</tr>
<tr>
<td></td>
<td>Internet Image</td>
</tr>
<tr>
<td>29-33</td>
<td>Modes</td>
</tr>
<tr>
<td></td>
<td>Rect Mode</td>
</tr>
<tr>
<td></td>
<td>Ellipse Mode</td>
</tr>
<tr>
<td></td>
<td>Text align</td>
</tr>
</tbody>
</table>
Table of Contents

Doing

Commenting  Page 34

Common Functions  Pages 35-38
  Creating a Function
  Types of Functions
  Calling a Function
  Draw Function

Mouse Positions  Page 39

Mathematics and Operators  Pages 40-42
  Assignment Operators
  Arithmetic Operators
  Comparison Operators

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

## Doing

### Arrays
- Intro to Arrays
- Array Types
- Changing Arrays
- Accessing Arrays
- Array Length
- Array Patterns

### Mouse Functions
- Types of Mouse Functions
- Mouse Pressed
- Mouse Clicked
- Mouse Released
- Mouse Button
- Mouse Moved
- Mouse Dragged
- Mouse Scrolled

**Pages 54-60**

**Page 60-67**
Table of Contents

Doing

Pages 68-71  Keyboard Functions

Key Pressed
Key Released
Key Typed
Key Code

Page 72-79  Objects and Prototypes

Object Literals
Object Prototypes
Use and make Objects
Object Methods

Note:
Talk to your coach about Object-oriented Programming.
## Table of Contents

### Doing - Specifics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor</td>
<td>Pages 80-81</td>
</tr>
<tr>
<td>Switch</td>
<td>Page 82</td>
</tr>
<tr>
<td>Trigonometry</td>
<td>Pages 83-85</td>
</tr>
<tr>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>Sine Functions</td>
<td></td>
</tr>
<tr>
<td>Cos Functions</td>
<td></td>
</tr>
<tr>
<td>Prompt</td>
<td>Page 86</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>Page 87</td>
</tr>
<tr>
<td>Pull Color</td>
<td>Pages 88-90</td>
</tr>
</tbody>
</table>

**Key Concepts**: Additional components to add to your projects and programs.
Key Concepts: Rotate, move and change the size of objects in your code through matrix changes.
Table of Contents

Converting

Translating Variables  Page 99

Pseudocode to Code  Pages 100-104

- Code Blocks
- Steps to Translate
- Examples of Translation

JS to Python  Page 105

Convert from Processing  Pages 106-107

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

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 `=` Equal Sign to assign a value to a variable.

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

```
var name = “Hatch”;
```

A-Z Text must be within quotes

“Hatch”
Variables can hold text and numbers including integers and decimals.

1. Use quotes to store text
   ```javascript
   var name = "Hatch";
   ```

2. Storing numbers
   ```javascript
   var num = 5.3;
   ```

How to Name your Variables

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

1. Starts with a Letter
   ```javascript
   var name;
   ```

2. Must contain ONLY
   ```javascript
   A - Z, a - z, 0 - 9, $, _
   ```

3. Cannot be a JavaScriptKeyword

4. Data Types
   Arrays
   Objects
   Functions
You can change a value stored in a variable while the code runs.

1. Original Variable
   ```javascript
   var num = 1;
   ```

2. New Variables
   - **Adds 5 to num**
     ```javascript
     num += 5;
     num = 6
     ```
   - **Multiplies num by 5**
     ```javascript
     num *= 5;
     num = 5
     ```
   - **Subtract num by 5**
     ```javascript
     num -= 5;
     num = 4
     ```
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

   ```javascript
   var var = "Hatch";  // Error
   var name = "Hatch";  // Correct
   ```

EX. Here are some examples:

   - **Reserved**
     - `var`
     - `loop`
     - `draw`

   - **User**
     - `name`
     - `cat`
     - `x`
Booleans allow you to turn specific components on and off. They are stored as true or false.

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

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

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

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

This will draw a circle if var circle is true.

This will draw an X if it is false.
In a computer, colors 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!

- Really Red! `color (255, 0, 0);`
- Really Green! `color (0, 255, 0);`
- Really Blue! `color (0, 0, 255);`

---

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

RGB Colour Wheel

Yellow: 255, 199, 20
Light Orange: 255, 189, 13
Orange: 255, 0, 0
Dark Orange: 235, 68, 0
Red: 255, 0, 0
Magenta: 255, 189, 13
Purple: 207, 0, 255
Blue: 100, 0, 255
Light Blue: 0, 0, 255
Light Green: 255, 240, 10
Light Orange: 255, 199, 20
Light Green: 255, 240, 10
Green: 0, 255, 0
Turquoise: 0, 235, 208
Light Blue: 0, 176, 235
Light Blue: 0, 255, 255
Green: 0, 255, 0
Light Green: 255, 240, 10
Skin Tones

- Colouring

- Skin Tones with RGB values:
  - 205, 154, 111
  - 191, 130, 107
  - 138, 86, 61
  - 66, 24, 21
  - 221, 156, 86
  - 238, 188, 153
  - 253, 239, 213
  - 248, 219, 187
Set the colour of multiple shapes. All shapes will be filled with the colour after the fill function.

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

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

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

Set the background colour of the canvas.

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

The amount of Blue and the Transparency

The amount of Red and Green

Setting the Background Colour

The background function sets the colour of the canvas.

```javascript
background ( 255 , 0 , 128 );
rect ( 100 , 150 , 150 , 150 );
ellipse ( 250 , 150 , 150 , 150 );
```
Fills acts on the inside of shapes and text and fills them with colour.

The amount of Red and Green

The amount of Blue and the Transparency

Filling Shapes with Colour

Place the fill function before the shapes.

\[
\text{fill}(\text{R}, \text{G}, \text{B}, \text{A})\;
\]

\[
\text{fill}(255, 0, 128)\;
\]

\[
\text{rect}(100, 150, 150, 150)\;
\]

\[
\text{ellipse}(250, 150, 150, 150)\;
\]
This is the outline of a shape, or the colour of a line.

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

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

noStroke(); removes the outline.

---

StrokeWeight

This changes the thickness of the outline.

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

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

**noStroke**

Using noStroke removes the outline of a shape.

```javascript
noFill();
fill(139, 121, 184);
rect(150, 150, 150, 100);
```
You can use HSB to adjust colours differently.

**HSB**

HSB means hue / saturation / brightness.

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

**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 grey.

- 0 = Grey
- 400 = Colour

**Lightness**

Is how much Black vs. White.

- 0 = Black
- 400 = White

Max Value

You can set the top value as the second argument.

HSB allows you to draw rainbows easily.
You can use RGB or you can use HSB as parameters in `colorMode()`;

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

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

Max Value

I used 400 to have a more gradual change and fill all the canvas.
What is the Coordinate Plane?

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

How to use the Coordinate Plane

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

point \((250, 125)\);
Draws an Oval or Circle from the center point $A$.

The $W$ and $H$ of the circle:

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

The $X$ and $Y$ position of the oval’s center point $A$:

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

How to Draw an Oval

Set the ellipse’s center point first, then set the width and height.
Draws a Rectangle or Square from the upper left corner point \( \text{A} \)

- The \( W \) and \( H \) of the rectangle

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

- The \( X \) and \( Y \) position of the rectangle's center point \( \text{A} \)

**How to Draw a Rectangle**

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

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

- The \( X \) and \( Y \) position of the rectangle's center point \( \text{A} \)
Draws the three points of a triangle, each point will need an X and Y position.

\[
\text{triangle} (125, 125, 300, 100, 100, 350);
\]

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.

```
quad (100, 50, 275, 75, 100, 350, 75, 350)
```

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. Then width and height as a second pair. Finally, the starting and end angles.

arc (200, 200, 300, 150, 0, 180);
To draw a custom shape, list all the vertices or points and put them between `beginShape()` and `endShape()`.

```
beginShape();
vertex (50, 50);
vertex (200, 50);
vertex (200, 200);
vertex (300, 200);
vertex (300, 250);
vertex (250, 50);
endShape(CLOSE);
```

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

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

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

```
line (125, 125, 300, 325);
```

1. Set the values of the 1st point of the X and Y values on the canvas.
2. Continue by assigning the X and Y of the 2nd point.
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 the curve points.
Text

Draws text on the canvas.

Hello World!

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

How to Set the Size of Text

This 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 ) ;
Displays an image on the canvas.

To add or change an image, go to the Hatch library and copy the text under the image. Paste this between the ( “ ” ).
Internet Image

Displays an image on the canvas taken from Internet.

1. To add or change an image from Internet, go to your browser and search for an image that you would like to use. The format of the image has to be an image or gif file.

2. Once you found the image, open the image in a new tab and copy the URL. Come back to Hatch Studio and paste the URL in between the (" ").
These change the starting location of what is being drawn.

```javascript
rectMode();
```

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

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

This is the normally drawn rectangle.

```javascript
rectMode(CORNER);
```

This is `rectMode(CORNER)`. 

The $W$ and $H$ of the circle

The $X$ and $Y$ of the center
These are the other possible options for `rectMode`:

- `
  rectMode (CORNERS);
  rect (100, 100, 150, 150);
`  
  The X and Y of the 1st point
  The X and Y of the 2nd point

- `
  rectMode (CENTER);
  rect (100, 100, 150, 150);
`  
  The X and Y position is the center point

- `
  rectMode (RADIUS);
  rect (100, 100, 150, 150);
`  
  The X and Y position is the center point
The default mode for ellipse is CENTER. This draws an oval from the center point.

```
ellipseMode();
```

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

The X and Y of the center point

The W and H of the circle

This is ellipseMode(CENTER);
These are the other possible options for `ellipseMode`.

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

- The X and Y of the 1st point
- The X and Y of the 2nd point

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

- The X and Y of the 1st point
- The W and H of the circle

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

- The X and Y of the center point
- 1/2 the W and H size of the oval

---

**Drawing**

**Ellipse Mode**

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

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

The X and Y of the right bottom corner

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

The X and Y of the center point

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

The X and Y of the left bottom corner
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  
ellipse (200, 200, 150, 150);

/* ellipse (25, 25, 25, 25);
    rect (100, 25, 25, 25);
*/

Only the ellipse at 200, 200 will show on the canvas.
Functions help organize and contain code into sections. Below shows how to define a 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 and end it with a semicolon.
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.

---

```
var drawSun = function () {
  background (158, 150, 200);
  fill (255, 255, 0);
  ellipse (200, 200, 100, 100);
}
```

---

Note: Check Out Draw Function for an example
Functions

Calling a Function

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

```javascript
drawSun();
fill(255, 255, 100);
```

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. This are reserved functions.

```javascript
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);
};
```

1st Loop
radius = 200

2nd Loop
radius = 55

3rd Loop
radius = 362

4th Loop
radius = and so on ...
Mouse Position

Horizontal Position of the Cursor

The keyword `mouseX` always contains the current horizontal position at the mouse cursor location.

Vertical Position of the Cursor

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

Using `mouseX` and `mouseY`

The keyword `mouseX` and `mouseY` act as variables and are used as parameters.

```javascript
var draw = function () {
    background (255, 0, 128);
    line (0, 0, mouseX, mouseY);
};
```
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
- `+=` adds the value
- `-=` subtracts the value
- `*=` multiplies a variable
- `/=` divides a variable
- `++` increment
- `--` decrement

```javascript
var num = 1;
num += 3;
```

Num is assigned a value of 1
Num has 3 added to it.
Num is now 4.
Arithmetic operators are used like normal math. These are often used to change a value in a parameter.

**Examples**

+  addition
-  subtraction
*  multiplication
/  division
%  modulus

```javascript
var num = 2;
point(0, num*10)
```

Num is assigned a value of 2
The point created have 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 less than
&&  and
||  or

>=  is greater than or equal to
<=  is less than or equal to
===  is equal to
!==  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 smaller than 2. If it is the fill is changed to black.
Random allows the program to randomly pick a value from a range.

```cpp
random()
```

Instead of writing a number of variables use random:

```cpp
ellipse(random(100, 250), random(100, 300), 30, 30);
```

This could become any of these or any in-between:

![Grid with random positions](image)

The random only applies to the x location and y location.
User created random variables will have the same value through the program.

Random in Variables

Instead of writing a number of variables use random

```javascript
var ran = random(0, 255);
ellipse(ran, ran, 30, 30);
```

This means a value from 0,255 will be assigned to "ran".

When you declare them inside a function all values will be random, outside will be the same.
What if you want something to look random, but always actually be the same?

```
rando

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 produce.
Try it yourself!

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

This will always produce.
Try it yourself!
"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

```javascript
var randomNum = random(0, 1000);
if (mouseX > 500) {
    random(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.
If the number is not bigger than 900 or smaller than 500 Winston appears.
Else if statements works 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 it is true. You can have multiple if else statements.

```javascript
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 red.
2. If not, it checks if it smaller than 300. If it is, it is green.

<table>
<thead>
<tr>
<th>randomNum = 400</th>
<th>randomNum = 200</th>
<th>randomNum = 600</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="White Square" /></td>
<td><img src="image2.png" alt="Red Square" /></td>
<td><img src="image3.png" alt="Green Square" /></td>
</tr>
</tbody>
</table>
It is important to make sure all of your conditions are different than the previous ones.

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

It would never turn red because the if statement would be true every time the else if statement would be true.

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

**Coding Example**

<table>
<thead>
<tr>
<th>randomNum = 400</th>
<th>randomNum = 800</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="White Square" /></td>
<td><img src="image2.png" alt="Green Square" /></td>
</tr>
</tbody>
</table>
Conditionals work like YES or NO questions. These are some additional examples of conditionals:

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

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

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

```java
if (my age is older than 16) {
    I can drive
} else if (my age is less than 16) {
    I can walk
}
```

```java
if (age > 16) {
    I can drive
} else if (age < 16) {
    I can walk
}
```

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

```java
if (weather !== raining && today === Saturday) {
    take weather bottle
    play soccer outside
}
```
For loops let you repeat parts of your code multiple times.

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

**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
   
   ```
   for ( var i = 0 ; i <= 10 ; i += 2 ) {
     // Code to repeat
   }
   ```

3. This loop will repeat 5 times
   
   ```
   for ( var i = 5 ; i < 10 ; i += 1 ) {
     // Code to repeat
   }
   ```

4. This loop will repeat 10 times
   
   ```
   for ( var i = 10 ; i < 0 ; i -= 1 ) {
     // Code to repeat
   }
   ```
This type of loop runs while the statement is true.

```javascript
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.
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.

```javascript
xPosition = [1, 2, 3, 4];
```

This is indexed in the 0 place.

Values in an array start at 0 and count up. The place a value is in the array is called the index value.
Array Types

Arrays with Information

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

```javascript
var names = [ "Steve", "Tom", "Laura" ];
```

This section is an array with numbers.

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

Empty Arrays

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

```javascript
var name = [ ];
```
Changing Arrays

Arrays allow you to store a list of items.

```javascript
var xPosition = [0, 20, 100, 300];
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays can hold any variables like numbers and words</td>
<td>Every variable gets a unique number to keep track of its position in the list</td>
</tr>
</tbody>
</table>

1. Changing an item

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

2. Adding an item

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

3. Removing an item

```javascript
xPosition.splice(2, 1);
```
Accessing Arrays

To access a specific value in an array we use [ ] next to the name of the array. This will get indexed value from the array.

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

```javascript
var array = [1, 2, 3, 65];
text (array [3], 200, 200);
```

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

This value would be blank as there is no value stored in index 4.
Arrays can be used in loops and in programs to allow the user to move onto the next item. It is often important to know the length of the array.

**Array.length**

We can find the length of the array by adding .length to any array.

```javascript
var number = [1, 2, 3, 65];
```

For this array, `number.length` would equal 4.

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

For this array, `values.length` would equal 6.
Using Array

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

```javascript
var array = 0;
var values = [1, 2, 3, "P", index];
```

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

```javascript
for (var i = 0; i < array.length; i++) {
    text(array[index], 50 + i * 20, 200);
    index++;
}
```

The for loop will work even if the array has more or less values stored in it! Adaptive code is awesome!
Array Patterns

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

```javascript
var index = 0;
var array = [1, 2, 3, "P", index];
textSize(20);
for (var i = 0; i < array.length; i++) {
    text(array[index], 50 + i * 20, 200);
    index++;
    if (index === 2) {
        index = 0;
    }
}
```

The code still only runs 6 times, which is `array.lenth`

This makes it easy to set up patterns that happen in your projects and let users select specific choices.
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.
Mouse Functions

mousePressed

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

```javascript
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.

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

Before you press  
While you press  
After you let go

The mouse needs to be pressed during the program for this function to work.
Mouse Functions

mouseReleased

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

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

Before you press

While you press

After you let go

The mouse does not need to be pressed during the program for this function to work.
Here's a code snippet demonstrating the `mouseButton()` function:

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

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

This function runs every time the mouse is moved.

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

Before the mouse is moved

While your press

These allows the ellipse to move with the mouse
Mouse Functions

mouseDragged ( )

This has the function run once every time the mouse moves while it is pressed,

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

Before you pressed

While you press the mouse and it is moved

After the mouse is moved
Mouse Functions

mouseScrolled ( )

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

// This pulls the page up and down using a middle bottom or 2 fingers.

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

This is shown when the mouse is scrolled
You can use functions to check if the keyboard is being used.

**KeyPressed**

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

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

When key Clicked

When KeyPressed for 5 seconds

When Released

When you declare them inside a function all values will be random, outside will be the same.
KeyReleased

The contents of the KeyReleased function runs when any key is release.

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

When keyPressed

When a key is let go

This will work only when the key is let go.
Key Typed

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

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

When keyPressed

When a key is let go

This only works when the key is pressed and let go.
You can use specific key is a key ____ function.

Key Codes

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

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

Dots will draw when "1" is pressed.
An **object** can be anything that may have multiple values, and these **values** will be what describes each **property**.

```javascript
var tommyTheCat = {
  age: 4,
  furColour: "orange",
  likes: ["naps", "milk"]
};
```

**Accessing Object Properties**

To access an object we use dot notation. This is how we edit objects.

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

The number 4 will be displayed in the canvas.

**Modifying an Object Literal**

Use dot-notation.

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

Changes tommy's age to 5.
This creates more than one instance of an object.

```javascript
var ball = function () {
    this.x = random(300, 20);
    this.y = random(300, 20);
    this.speedX = 5;
    this.speedY = 5;
};
```

Constructors are used to make a new instance of the object. Object Prototypes act like functions so they need brackets when constructed.

```javascript
var myBall = new ball();
myArray.push(new ball());
```
Let’s say you want to create many cars in your project. We can make them one at a time using objects.

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

Part 1

Prototypes allows you to make a pattern to create multiple objects, you only change...

```javascript
var car = {
  To
};
```

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

This car was randomly selected to be green because the colour is random.
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. You want it to have 4 wheels, 2 seats and be yellow! Create it using the code below.

```javascript
var sportscar = new car ( 4 , 2 , “yellow” );
```

You can not use the code above because the properties have not been written in the code.
Let’s say you want to create a van style of car. When you initially create it, it will be based on a normal car.

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

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

```javascript
var seats = 7;
```

Now your van style car has 7 seats!
Part 4 - Code this yourself!

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

var sportscar = new Car(4, 12, [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; numWheels < sportscar.Seats; numSeats++) {
    stroke(0);
    line(numSeats * 30 + 115, 250, numSeats * 30 + 130, 250);
    line(numSeats * 30 + 130, 225, numSeats * 30 + 130, 250);
}
```
Object methods are functions that affect or act on an object. These are declared like other properties and can be used in object literals or object prototypes.

**Object Literal Method**

This method will act on the car object literal.

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

```javascript
var cars = function () {
    this.x = 10;
    this.move = function () {
        this.x += 1
    };
};
```
You can change the cursor seen on the canvas with `cursor()`;

There are different types of cursors add the one you want as the parameter.

<table>
<thead>
<tr>
<th>Cursor</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>✈️</td>
</tr>
<tr>
<td>ARROW</td>
<td>⬅️</td>
</tr>
<tr>
<td>CROSS</td>
<td>✖️</td>
</tr>
<tr>
<td>HAND</td>
<td>⏱️</td>
</tr>
<tr>
<td>TEXT</td>
<td>⏳</td>
</tr>
<tr>
<td>WAIT</td>
<td>⏳</td>
</tr>
</tbody>
</table>

Make sure that you use capitals.
`Using Cursor();`

```javascript
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. You create cases that are the parameters in the switch();

```javascript
var circle = function () {
    switch (key + 0) {
    case 49:
        ellipse (100, 100, 100, 100);
        break;
    case 50:
        ellipse (200, 200, 100, 100);
        break;
    default:
        ellipse (300, 300, 100, 100);
    }
};
```

This is the "1" button

This is the "2" button

This is all other keys

You need to include break on non-default cases.

If 1 is pressed

If 2 is pressed

If anything pressed

Break will also stop loops (for and while) if you want the code in a loop to stop write break;
There are several trigonometry functions! This is a part of math. These let you animate in way patterns! We don't normally use them like in math.

**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!
Use this to draw a moving picture!

Using Trigonometry

```javascript
var x = 0;
fill(255);
background(255);
frameRate(255);
var draw = function () {
    background(255);
    ellipse(x, sin(x) + 200, 10, 10);
    x += 0.1;
};
```
Using Trigonometry

var x = 0;
fill (255);
background (255);
frameRate (255);
var draw = function () {
    background (255);
    ellipse (x, cos (x) + 200, 10, 10);
    x += 0.1;
};
If you want the user to type text for your program you can use `prompt`. `Prompt` has two parameters. The first writes the question the user would see, the second has a possible answer already inputted.

```
prompt ("1", "2");
```

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

The box below would appear on the screen. What ever is written by the user, will be stored in the variable name

```
app.hatchcoding.com says
Write Your First Name
Hatch Student Name
```

**NOTE:** `Prompt` works better in `mouseClicked` function compared to in `draw` functions. Draw functions keep calling the `prompt` and your program become difficult to use.
Doing - Specifics

Frame Rate

Normally we want our program to run quickly and smoothly. The base frame rate is 60 frames per second. You can change it using frameRate.

Changing the FrameRate

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

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

Decreasing the frame rate will make the program seem like it is skipping or slow.

```javascript
var draw = function () {
    frameRate (10);
    ellipse (mouseX, mouseY, 200, 200);
};
```
You can extract a section of colour using the keywords red, green, blue.

**red(); Example**

red(); will extract the red value from a colour, In fill this would be fill(255,0,0);

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

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

**green( ); Example**

green( ); will extract the red value from a colour, In fill this would be fill(255,0,0);

```
var g = color (20, 130, 20);
fill (g);
ellipse (135, 200, 150, 225);
var greenValue = green (g);
fill (greenValue);
ellipse (300, 200, 150, 225);
```
You can extract a section of colour using the keywords red, green, blue.

**blue(); Example**

blue(); will extract the blue value from a colour, In fill this would be fill(255,0,0);

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

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

Our normal Canvas is 400 by 400

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

size (200, 400);
You can change the size, location and orientation of objects by changing the canvas.

**Canvas Changing keywords**

- `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 Resetting keywords**

- `resetMatrix ();` Removes all canvas changes
- `pushMatrix ();` `popMatrix ();` Removes all canvas changes between the push and pop.

---

**Matrix Changes**

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

```javascript
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.
Matrix Changes

Rotate Example

Code to Rotate

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

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

This will rotate and print each square. The loop runs 5 times and 5 squares are printed!
You can also move or edit the canvas. You can change the Matrix using the translate, rotate and scale functions.

Translate

Translate moves the canvas in the X and Y direction.

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

2. translate (100, 50);
   rect (100, 100, 100, 100);
You can also change the size of the canvas and make objects look bigger or smaller.

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

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

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

Reset Matrix

Reset Matrix removes all changes currently impacting the matrix.

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

All lines following Matrix change impact in all following lines!

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

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

You can set the top value as the second argument.
Using push and pop matrix allows you to move specific things on your canvas.

Push and Pop Matrix

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 are rotated and translated.

Push and Pop Matrix ();

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

2. rect (300, 300, 100, 100);

Only the orange section impacted by the Matrix Changes
Translating code from other languages is an important skill. You can also try to convert syntax from Hatch.js to Hatch.py.

### Variables

<table>
<thead>
<tr>
<th>TYPE</th>
<th>PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true, false</td>
</tr>
<tr>
<td>float</td>
<td>decimal numbers</td>
</tr>
<tr>
<td>int</td>
<td>whole numbers</td>
</tr>
</tbody>
</table>

If you use processing documentation you will need to redefine your variables.

```
int color red = color (26);  
To
var red = color (26);        
```
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!
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 \(\texttt{71, 173, 12}\)

Using the Steps

1. The keyword for this line is fill.
2. The syntax for this word is \(\texttt{fill(#, #, #)}\);

\(\texttt{fill(71, 173, 12)}\)
## Pseudocode Translation Examples

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

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

You may want to use other Reference Manual Page and past project to help!
# Pseudocode Translation Examples

Having trouble with a specific line of pseudocode?

<table>
<thead>
<tr>
<th>CODE TYPE</th>
<th>PSEUDOCODE</th>
<th>TWYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loops</td>
<td>Create a for loop that begins with counter $i$ equal to 0, and for each loop where $i$ is less than 400, $i$ increases by 1</td>
<td><code>for(var i=0, i&lt;0, i++);</code></td>
</tr>
<tr>
<td>Object</td>
<td>Create Box as an object</td>
<td><code>var Box(255, 0, 0);</code></td>
</tr>
<tr>
<td>Images</td>
<td>Draw the image &quot;starwars/c3po&quot; at position 10, 10 with a size of 100, 100</td>
<td><code>image(getImage(&quot;starwars/c3po&quot;), 100, 100);</code></td>
</tr>
<tr>
<td>Mouse</td>
<td>At position mouseX</td>
<td><code>mouseX</code></td>
</tr>
</tbody>
</table>

You may want to use other Reference Manual Page and past project to help you with other specific examples!
Pseudocode Translation Examples

Having trouble with a specific line of pseudocode?

<table>
<thead>
<tr>
<th>CODE TYPE</th>
<th>PSEUDO CODE</th>
<th>TWYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Create an array myNums and assign it a list of numbers between 1 and 3</td>
<td>var myFoods = [1, 2, 3];</td>
</tr>
<tr>
<td>Translate</td>
<td>Translate the canvas by 100, 100</td>
<td>translate(100, 100);</td>
</tr>
<tr>
<td>Modes</td>
<td>Set the ImageMode to Center</td>
<td>imageMode(CENTER);</td>
</tr>
<tr>
<td>Random</td>
<td>Set to a random number between 1 and 100</td>
<td>random(1, 100);</td>
</tr>
</tbody>
</table>

You may want to use other Reference Manual Page and past project to help you with other specific examples!
Big Difference Between JS and PY

1. Bad indentation will break your project in Py.

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

Looks wrong, but it will function.

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

Looks wrong, but it will not function.

2. Less brackets and semicolons in PY

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

Looks wrong, but it will function.

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

Looks wrong, but it will not function.

3. You must call global variables

If you use global variable in your project you need to tell the program you will use them in this specific function in Py.

```py
def draw():
    global chompSpeed, pacMouth, pacMouthClose
```
Using the processing.js or processing.py documentation maybe 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 ____________ = function();`

**Processing.js**

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

**Hatch Studio**

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

Note there is NO semicolon

Both will show this!
In processing.js normally you need to call a canvas ad setup 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.

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

This would create the base canvas used in the Hatch IDE.