WWW.
NEXGEN
CODECAMP
.COM.AU

Coding in the Classroom - Notes

Lesson 1 - Introduction to Javascript

What is Javascript?

Javascript is an 'interpreted' general purpose computer language that can be run pretty much anywhere: on your desktop, laptop, browser, phone, devices, client and server. It is easy to learn and does not require anything to be installed on your computer to be able to use it. It is also the 'language of the web' and is run on most of the websites in the World today. This helps us to answer the next question.

Why use Javascript

- Javascript is easy
- No installation required
- You can be productive very quickly
- Javascript is everywhere
- Javascript is an excellent first language to learn

How do we use Javascript

Do you have a computer with a browser? Then you can use Javascript!
We use an editor such as Atom to create our javascript files which we then run in the browser.

How Javascript is relevant to the Classroom

It is easy to come up with a simple teaching example
It can be easily hooked up to all kinds of data
It can be used on most microcontrollers/devices
It is the foundation of the web so relevant for websites

Lesson 2 - Data, Data Types and Variables

Data

What is data?
It is (hopefully relevant) information about a topic you are interested in/studying/researching.
It is measurements, observations, facts, ideas & discoveries. It is what drives an information society.

Data Types

In programming languages, we need to be able to model different kinds of data. These can be simple types of data or complex. Typically most programming languages have similar datatypes.

In Javascript we have the following *'simple'* datatypes:

Number:

0, 1, 10, 3.142, -0.55, 55000000

String:

"Hallo World", 'xyz', "this is a string", '1234'

Boolean:

This can be either true or false

...and then we have the following *'complex'* datatypes

Array

An array is a list of data. It could be a list of strings, numbers, booleans or objects or it could be a mix of these. An array is delimited by square brackets - [ ]. An example of an array or list of school subjects is below:

["Maths", "English", "French", "Geography", "Art", "History"]

The brackets have been colored red to make them stand out! Arrays are very useful as we tend to run our lives by lists - ie. they are *very real world!*

Two things we ABSOLUTELY MUST KNOW about arrays
How to find the length of an array
Use the 'length' property. Eg. [ ].length
Try it with the above array in Chrome.

How to access an element in an Array
```
var subjects = ["Maths", "English", "French", "Geography", "Art", "History"]
console.log(subjects[0]);
console.log(subjects[2]);
console.log(subjects[4]);
```

What do you think these three lines will print out? Try it in Chrome!

Object

An object is able to model complex 'real-world' things. This is what an empty object looks like:

```
{}
```

Not very inspiring is it? Of course, whilst valid, an empty object isn't very useful.

Take the following (slightly more useful) example of a school class that may be modelled in a Learning Management System:

```
{
    className: "Magee",
    teacherName: "Miss K",
    numStudents: 20,
    location: "A Block
}
```

An object is 'bounded' with braces {} and contains one or more 'properties'. Each property has a *name* and a *value*. In the above example, the first property is shown in red. The *name* is className and the *value* is "Magee".

Note also that the property name is suffixed with a colon ':' - this is crucially important and if not there, the object will not be recognised as an object.

The value of a property may be a simple piece of data such as a string or a number as in the above example, or it could be a complex piece of data such as another object. In this case we would have an object inside another object.

Take the following example of the Movie 'Good Will Hunting' - modelled by our Javascript object below:

```
{
    "Title": "Good Will Hunting",
    "Year": "1997",
    "Rated": "R",
```

```
        "Released": "09 Jan 1998",
        "Runtime": "126 min",
        "Genre": "Drama",
        "Director": "Gus Van Sant",
        "Writer": "Matt Damon, Ben Affleck",
        "Actors": "Matt Damon, Robin Williams, Ben Affleck
        "Country": "USA",
        "Awards": "Won 2 Oscars. Another 22 wins & 55 nominations.",
        "Poster": "https://images-na.ssl-images-amazon.com/images/V1_SX300.jpg",
        "Ratings": [
            {
                "Source": "Internet Movie Database",
                "Value": "8.3/10"
            },
            {
                "Source": "Rotten Tomatoes",
                "Value": "97%"
            },
            {
                "Source": "Metacritic",
                "Value": "70/100"
            }
        ],
        "Metascore": "70",
        "imdbRating": "8.3",
        "imdbVotes": "643,024",
        "imdbID": "tt0119217",
        "Type": "movie",
        "DVD": "08 Dec 1998",
        "BoxOffice": "N/A",
        "Production": "Miramax Films",
        "Website": "http://www.miramax.com/movie/good-will-hunting/",
        "Response": "True"
}
```
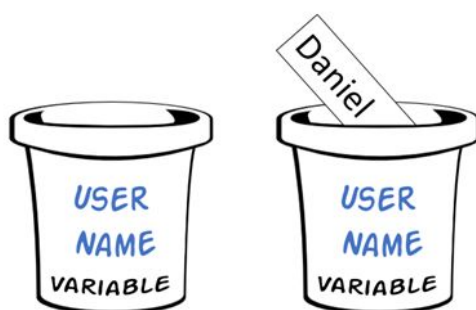
Variables

In any software system that models some data structure(s), we MUST have the capability of storing data in memory otherwise we would all be like the proverbial goldfish with the mythical 3 second memory.

THE TRAGEDY OF A THREE SECOND MEMORY
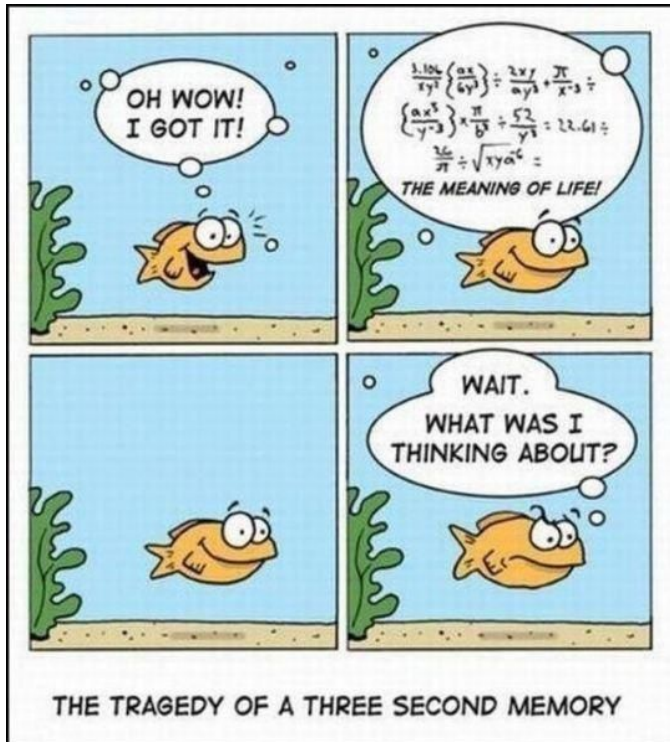
We store data in **variables**. These 'variables' can be thought of as buckets that we put data into and retrieve when we need it again.

See the example below which models a variable called **username** that holds a single value (a **String**). In the first bucket the variable is empty. In the second, we have populated it with a value of "Daniel".

The variable has no value until the user enters a name.

Variables in Javascript

In Javascript, variables are typically the first thing we will code. Without variables we really can't do much to be honest.

Let's take the example of a person. We are going to create 3 variables below. One called **personName**, one called **personAge** and one called **personVocation**.

```
1
2    var personName = "Pete";
3    var personAge = 47;
4    var personVocation = "Developer";
5
6
```

Things to callout:
1. 'var' is the keyword in Javascript we must use to create a variable of any description.
2. personName, personAge and personVocation are all variable names.
3. "Pete", 47 and "Developer" are the variable values.
4. = is used to assign a value to a variable name.
5. We use a naming convention called camelcase when naming variables. The 2nd and subsequent word in a 'camelcased' word are always capitalized.

Here are a couple of slightly more complex examples:

```
 4
 5      var myFavouriteAlbums = ["Goodbye Yellow Brick Road", "Achtung Baby", "An Innocent Man", "Definitely Maybe", "Rumours"];
 6
 7      var cars = {
 8          "Models": [
 9              {
10                  "model_name": "Maranello",
11                  "model_make_id": "ferrari"
12              }, {
13                  "model_name": "Mondial",
14                  "model_make_id": "ferrari"
15              }, {
16                  "model_name": "Mythos",
17                  "model_make_id": "ferrari"
18              }, {
19                  "model_name": "P2",
20                  "model_make_id": "ferrari"
21              }, {
22                  "model_name": "P5",
23                  "model_make_id": "ferrari"
24              }, {
25                  "model_name": "Pinin",
26                  "model_make_id": "ferrari"
27              }, {
28                  "model_name": "Rossa",
29                  "model_make_id": "ferrari"
30              }, {
31                  "model_name": "Superamerica",
32                  "model_make_id": "ferrari"
33              }, {
34                  "model_name": "Testarossa",
35                  "model_make_id": "ferrari"
36              }
37          ]
38      }
```

Another way to think about variables is that sometimes our program may have a particular lifetime, for example a computer game has a lifetime of the length of the game. If I play Pacman for 15 minutes achieving a record breaking score of 200,000 after having eaten my way through 15 levels, my program will have had to keep track of certain data/information:

My score
My current level
Which fruits I had eaten

..and probably a lot more 'stuff'.

All of these data items would be stored in variables and might look a bit like this when they are defined at the beginning of the game.

```
// At the beginning of the game
var score = 0;
var currentLevel = 0;
var fruitsEatenByPlayer = [];
```

Naming Conventions

At this point we need to stop and think about naming conventions. You will recall how to camelcase a variable or any text for that matter - if I have a variable called mybignumber, then the camelcase would look like this: myBigNumber - capitalization of the second and subsequent word in the 'word'.

Now, what do you think this function does?

```
dsfadsyfpowuhwljenrqfhdsjn(n) {
      return n + 2;
}
```

You might have said it accepts a single parameter, adds 2 to it and returns the result. I seriously doubt you got that information from the name of the function though!!
From this example you can see that it is very important to name variables, functions and anything else well. Normally we will not be working in isolation and it is very important for good teamwork and collaboration that others can understand our code. A better function would be:

```
function add2(n) {
      return n + 2;
}
```

Lesson 3 - Functions

A function is a discrete unit of work that performs some (useful) activity, expressed in a particular way which is dependant on the programming language being used. Functions must be *'called'* in order for them to run. A function may take one or more *'parameters'*

Let's think about the real world for a minute (it can't hurt!).

A Calculator performs a number of functions. One of these is 'add'. If we were to write this add function in javascript it might look like this:

```
1
2    // This is the function...
3    function add(a, b){
4        return a + b;
5    }
6
7
8    //... and this is how we call the function
9    add(3, 6)
10
11
```

Important points about functions

They must start with the keyword *'function'*
They (generally) have a name - as in the above case the name is *'add'*
They take one or more parameters - *a* and *b* are parameters in our *add* function.
A function's parameters are always surrounded by parentheses *(a, b)* above.
The parentheses must ALWAYS be present even if there are no parameters.
The function body (the actual code of the function) is ALWAYS bounded with braces {...}

When/Why to use functions

When you have a logical unit of work to perform.
Functions are typically less than 30 lines of code, ie. don't do more than one thing

Functions help to describe how your system // works

Calling Functions

One of the things we find that is not very well understood for people new to programming is that there is a huge difference between the function and calling the function. A function can exist but may never be called. Take the calculator example. I may never press the '+' button on my calculator for as long as I have it. Therefore it will never do any addition. Therefore it's 'add' function will never be called, however it still exists !!

Likewise in our example piece of code above, our add function in lines 3-5 exist but may never be called. In actual fact, we do call the function on line 9.

So, to call a function in code we simply write the name of the function followed by a single pair of parentheses. Inside the parentheses, we can pass in values to the function. These values are called parameters. Our add function itself takes 2 parameters - a and b. We do not specify whether these are numbers, strings, booleans or anything else. Our function is simple and we assume that it will be called correctly ie. with numbers. Imagine calling our function like this:

add("eggs", "potatoes")

What do you think the function would return to you? (HINT: if you add 2 strings together, it simply concatenates them together).

Returning Values from Functions

A function may or may not return a value back to the *callee*. To return a value we simply use the keyword *'return'*.

Comments

A comment in Javascript or any programming language is simply help text to enable anyone who reads your program to understand what is going on. All programs should be well commented without going overboard. A Javascript comment looks like this:

```
// This is a comment
/* this is a multi-line
   Comment.
*/
```

Amendment to our Function example

You may have noticed that in our 'add' example we return a value in the function but we do not do anything with the returned value like store it in a variable for example.

Contrast the first example with the example below:

```
1
2    // This is the function...
3    function add(a, b){
4        return a + b;
5    }
6
7
8    //... and this is how we call the function
9    var answer = add(3, 6)
10
11   console.log(answer);
12
13
```

In this variation of the first example, we are 'returning' the value from the function 'into' a variable called 'answer'. We are then printing out the value of the variable to the console. If I copy the code and paste it into the console in Chrome, I get this:
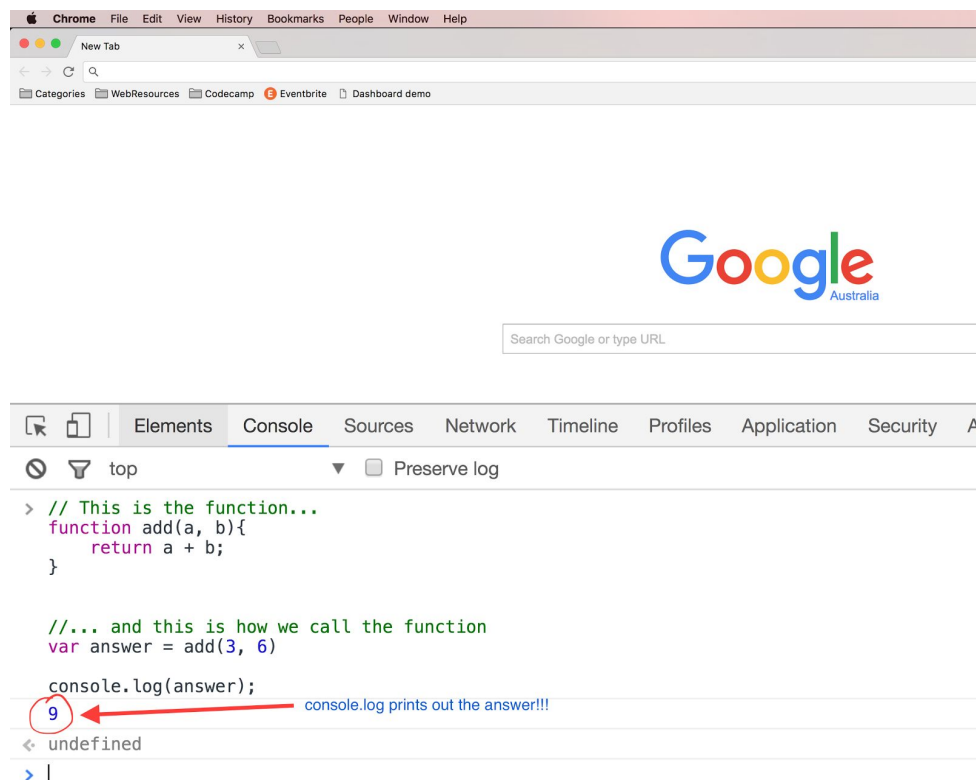
Console.log

You may have already noticed that console.log is a very handy statement. It allows us to print out anything to the Chrome console. So as a quick example, in Chrome - bring up the Dev Console (F12 on Windows, ⌘+OPTION + I on Mac). Select the console tab:

Paste in the following code to the console:

```
console.log("This is just some text");
console.log(1+1);
console.log("This" + "is" + "some" + "more" + "text");
console.log({name: "Pete"});
console.log(["Maths", "French", "English", "Geography", "Technology",
"History"]);
```

Hit enter and you should see the answers:
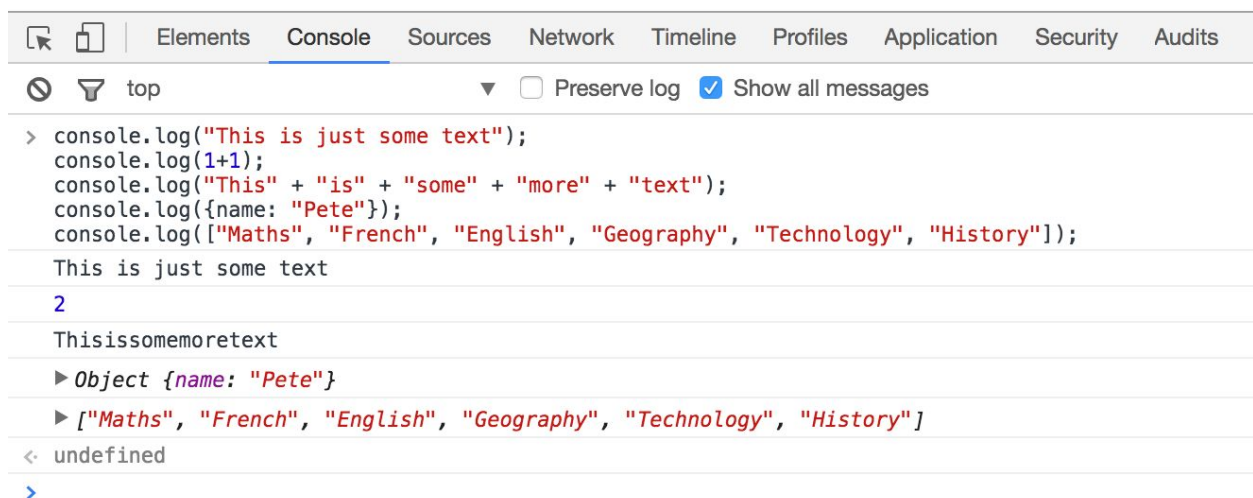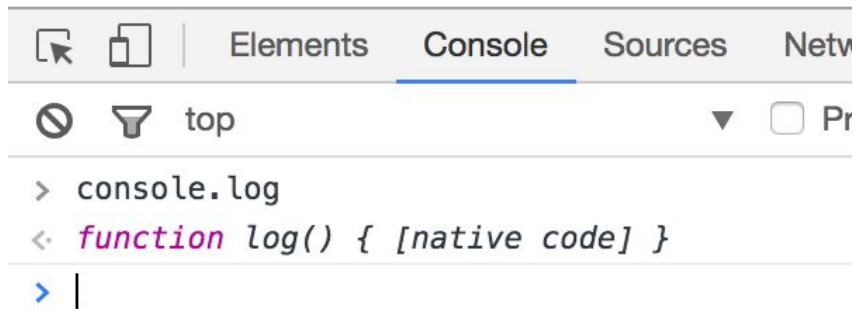
Technically speaking, console.log("some text") means the following:
1. The console object is calling the log function and passing a single parameter to it - in this case a string "some text".
2. The log function runs and prints to the Chrome Developer console.

If you type console.log and then ENTER you will see the following:

Yes - by omitting the parentheses on the end of a function name we are just asking the language interpreter to tell us about the function. And it obliges happily by telling us the console.log is indeed a function but contains native code, which is code built into Chrome that we don't need to know about.

Of course we ALL know how to *call* the 'log' function now don't we ?

Semicolons

Another thing that you may have noticed is that some lines are terminated with a semicolon ';' thus.

It is generally good practice to terminate lines with a semicolon. Be careful, however that you do NOT put semicolons after an opening brace { or an opening bracket (because these symbols denote the starting boundary of a block of code. It would not make any sense at all to put a semicolon after these symbols.

Here is an example function with semicolons added to illustrate the point:

```
function addOneToMyNumber(num) {

    // This function takes a single parameter - num, adds 1 to it
and
    // returns the result

    var result = num + 1;
    return result;
}
```

I have highlighted the semicolons in red above just to make them stand out. If you left them off the code would still be valid, however I would encourage you to use them.

# Lesson 4 - Modelling Complex Objects

We have already touched on the Object datatype. Objects allow us to model information in the real world with a bit more realism. Taking our example of a Calculator before, we are going to build a Calculator program, much like the Calculator in Microsoft Windows but without the Graphical interface.

Create a new file in Atom called Calculator.js
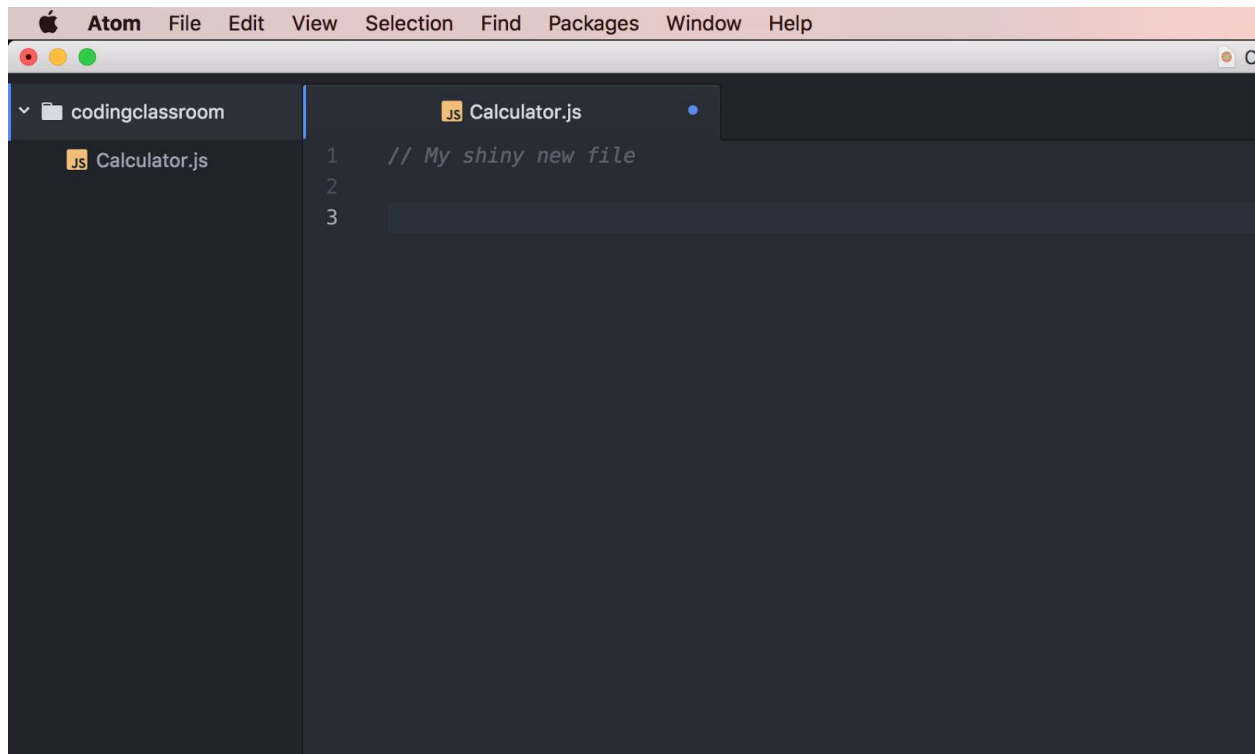
You will notice that I have created this file under the 'codingclassroom' folder and then added this to Atom (File -> Add Project Folder…).

Create a new variable of type Object with a value of an empty object and a name of Calculator:
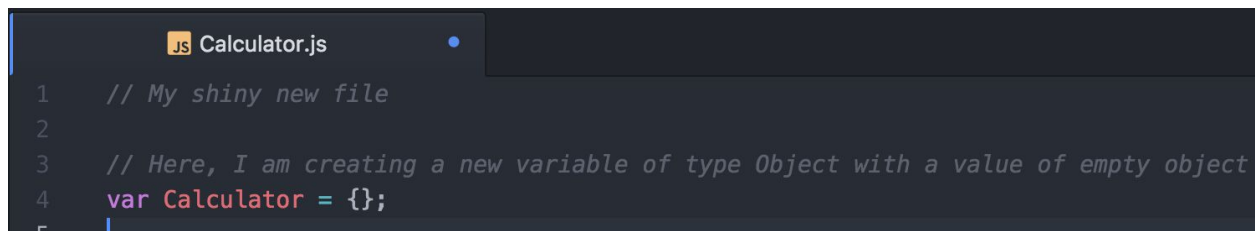
Note the capital 'C'. When we create Object variables which will actually contain some functionality, we normally give them a capital letter.

Now add a single property to it. The property name is 'add' and the property value is an empty function. The function is slightly different to before in that it has no name. The name of the function is the property name 'add' which we have already defined. Remember a function called 'add' would look like this:

```
6
7      function add() {
8
9      }
```
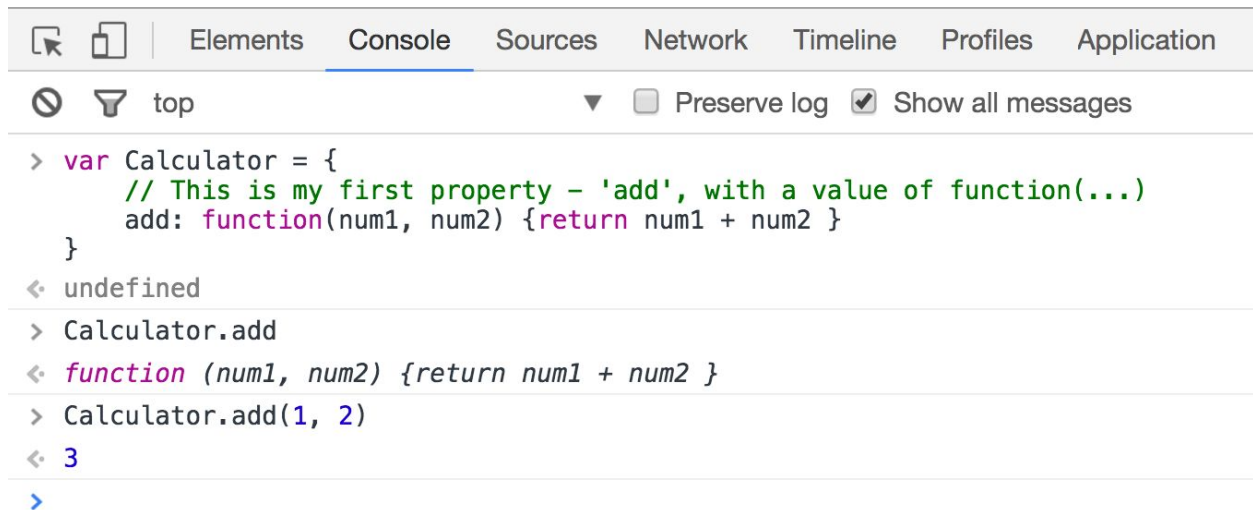
A function with no name would look like this:

```
6
7      function () {
8
9      }
```

A function as a value of the 'add' property in our simple object would look like this:

```
1    // My shiny new file
2
3    // Here I am creating a new variable of type Object with a value of empty object
4    var Calculator = {
5        // This is my first property – 'add', with a value of function(...)
6        add: function(num1, num2) {return num1 + num2 }
7    };
```

We can now test the code in Chrome Dev Tools. Copy and paste it into the console.

```
> var Calculator = {
      // This is my first property - 'add', with a value of function(...)
      add: function(num1, num2) {return num1 + num2 }
  }
< undefined
> Calculator.add
< function (num1, num2) {return num1 + num2 }
> Calculator.add(1, 2)
< 3
>
```

Above, you can see we have done a few things:
Firstly we have pasted the code in.

Secondly, we have asked Chrome what Calculator.add is. Notice that we are not running the function here, just kind of inspecting it. To run it, we would have to pass in 2 numbers so that the function could return the result of adding them up.

Thirdly, we are running the function and passing the parameters 1 and 2 into the function. When we hit ENTER the function runs and returns the result back to us.

QUESTION:
How would we turn this Calculator object into a more useful calculator ie. by adding functions for subtract, multiply and divide ?

ANSWER:
Add 3 more properties to the Calculator object called 'subtract', 'multiply' and 'divide'. Each property should have a function as it's value. The function should take 2 parameters and should **_return_** the appropriate result.

The code is on the next page but have a go yourself before looking. The big clue is - you already have the answer staring you in the face ie. the 'add' property.

```js
// My shiny new file

// Here I am creating a new variable of type Object with a value of empty object
var Calculator = {
    // This is my first property — 'add', with a value of function(...)
    add: function(num1, num2) {return num1 + num2 },

    subtract: function(num1, num2) {return num1 - num2 },

    multiply: function(num1, num2) {return num1 * num2 },

    divide: function(num1, num2) {return num1 / num2 }
};
```

More Chrome tips

Once you copy and paste the above code (or any code) into Chrome and hit ENTER, Chrome will read that code into it's memory and store it. That means that it now knows about your Calculator object (bearing in mind that the code is error free, otherwise Chrome will 'spit it out').

This means that you can start writing anything in the Chrome console and if Chrome recognizes it as an object or function or variable in its memory, it will try and fill in the word for you, thus saving you valuable seconds of typing! Seriously though this is an amazing feature of Chrome because it lets you know that your code 'worked' or at least was valid and is now in memory.

Check this out…
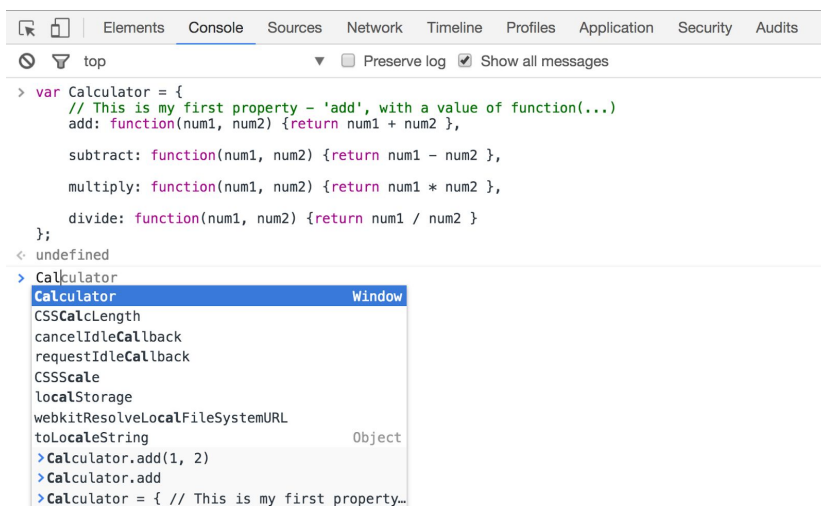
I have pasted the Calculator code into Chrome. The 'undefined' that you see simply means that nothing was returned. So at this point, having had no errors I can see that my code has been accepted by Chrome which means it is in memory.

I now start writing 'Calculator' and a popup window shows me the possible 'completions' to my word. Lo and behold - Calculator is in the list. All I have to do now is select it and then choose a method on the object.

Dot Notation

One thing that we may not have yet covered is how to call a function or method (the two words are synonymous but usually we talk of Object methods and not functions).

To do this we use the dot notation. If I have a Calculator object with certain methods, I must separate the object name and the object method with a dot '.', therefore the add method would look like this:

Calculator.add

However, if you hit ENTER at this junction, Chrome will merely tell you what the 'add' method is rather than call it:

```
> Calculator.add
< function (num1, num2) {return num1 + num2 }
>
```

That isn't very helpful in terms of running the function/method but at least it tells you what to expect when you do eventually run it.

To run any function/method you **ABSOLUTELY MUST** call it with brackets and parameters (if any):

```
> Calculator.add(100, 999)
< 1099
> |
```

Calculator.add is the equivalent of showing me your Calculator has an add button and then wondering why it isn't doing anything. Yes - I can see it is an 'add' button but you need to enter

some numbers and press the button to actually get it to do something or in programming parlance 'be called' or 'run'.

Lesson 5 - Conditional Statements

Making decisions is a very important aspect of computer programs. A payroll system needs to know when to trigger the pay() function for each of the employees in the database. A game may need to decide if the player should be awarded a bonus.

We make these decisions with if...then...else statements. They look like this:

```
if (some boolean expressions) {
      // do this…!
}
else {
      // do that...!
}
```

So in the above code… if the boolean expression is true - the orange code will be run. If the expression is false, the blue code will run.

It doesn't get a lot more complicated than that!

Here is an example we are going to turn into a game:

```
16    var players = ['p1','p2','p3','p4','p5','p6'];
17
18    // Let's say that players[i] has been shot at!!
19    // Check if player has been shot at
20    if(players[i].hasBeenShotAt == true) {
21        // Call the duck() method on the player
22        players[i].duck();
23
24            // Player to the left should shoot right
25        players[i-1].shoot(RIGHT);
26
27            // Player to the right should shoot left
28        players[i+1].shoot(LEFT);
29
30        var player = whichPlayerHasBeenHit();
31        if(player != null) {
32            player.isDead = true;
33
34            // Remove player from array 'players'
35        }
36    }
```

Else..If

You may have multiple conditions going on in a conditional statement:

```
if (some boolean expressions) {
     // do this…!
}
else if (some other boolean expression) {
     // do that...!
}
else {
     // do something else...!
}
```

Best Practice

Make you boolean expression as simple as possible
Make your if...statement as simple as possible
Format your code to be easy to read

Boolean Expressions

You may be wondering what we mean by 'boolean expression'. This is simply something that evaluates to either **true** or **false**. Eg:

```
myTestVariable == true; // is this equal to that?
```

Notice the '=='. If we had used a single =, this would have meant 'assignment' and would have assigned the value of true to the variable 'myTestVariable'.

```
myTestVariable != true; // This means 'not equal to'
```

Code Formatting

Some simple rules of thumb. Why? Because you normally develop software in teams so other people need to be able to understand your code. Even if you don't, you will still want to format your code well because you will come back to the code in 6 months and wonder what it does unless it is well commented and well structured.

So:
- Comment your code well
- Make your code as simple as you can
- Use well named variable names

- Use camelcase all the time
- Use well named function/method names
- Use capital letters for Object variables
- Always indent your code
- Use blank lines judiciously
- Don't repeat yourself
- Don't repeat yourself - put code in a function/method if the same code is appearing everywhere.
- Don't use magic numbers. Use constants with well named identifiers.
- Think about what data structures you need: Arrays for lists, Objects for complex items.
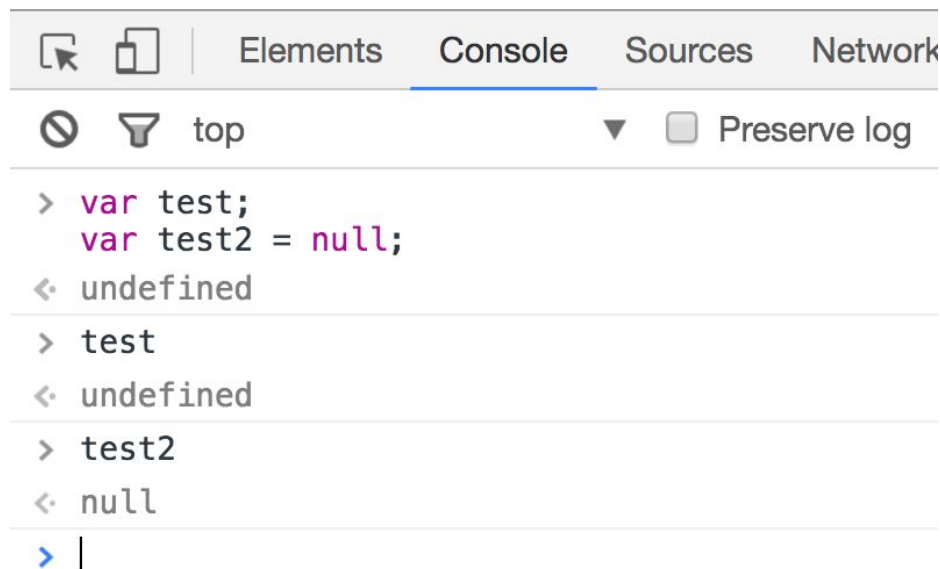
Two Special Values

null and undefined

'null' and 'undefined' are special values in Javascript that both effectively mean 'nothing' or 'empty'.

When you define a variable without a value it is `undefined` eg:

```
// Copy and paste into Chrome Dev Tools
var test;
var test2 = null;
```

Lesson 6 - Loops

What is the problem with this bit of code?

```
37
38    function payEmployee(employeeName){
39        // Some code to pay employees!
40    }
41
42    function init(){
43        payEmployee('Adam');
44        payEmployee('Bill');
45        payEmployee('Charlie');
46        payEmployee('Dave');
47        payEmployee('Ed');
48        payEmployee('Frank');
49        payEmployee('George');
50        payEmployee('Harry');
51        payEmployee('Illyia');
52        payEmployee('Jeff');
53    }
54
```

I hope this is obvious to you! What happens if we suddenly get another employee or if one of our employees leaves? Even despite these issues, this code looks and smells awful!

What about this?

```
37
38    // Fetch our employees from our database
39    var employees = database.getEmployees();
40    // Now employees looks like this:
41    // ["Adam", "Bill", "Charlie", "Dave", "Ed", "Frank", "George", "Harry", "Illyia", "Jeff"]
42    //
43    function payEmployee(employeeName){
44        // Some code to pay employees!
45    }
46
47    function init(){
48        for( i=0; i < employees.length; i++){ // Remember how to find the length of an array?
49            payEmployee(employees[i]);        // Remember how to get an item from an array?
50        }
51    }
```

A loop enables us to perform repetitive tasks many hundreds, thousands or millions of times with the same amount of code - purely by repeating the same code over and over again.

The loop has a counter (in this case 'i') which starts at 0 and is incremented each time a single pass of the task is complete. The for...loop performs a test for each iteration. Every time the test passes, the code inside the for...loop is executed and we are sent back to the beginning of the for loop again….. Until the test fails. In our case above, it will fail when we get to employee number 11 because that person does not exist. In other words, when `i = 11`, the test `i < employees.length` fails and we exit the for...loop and continue with the rest of the code.
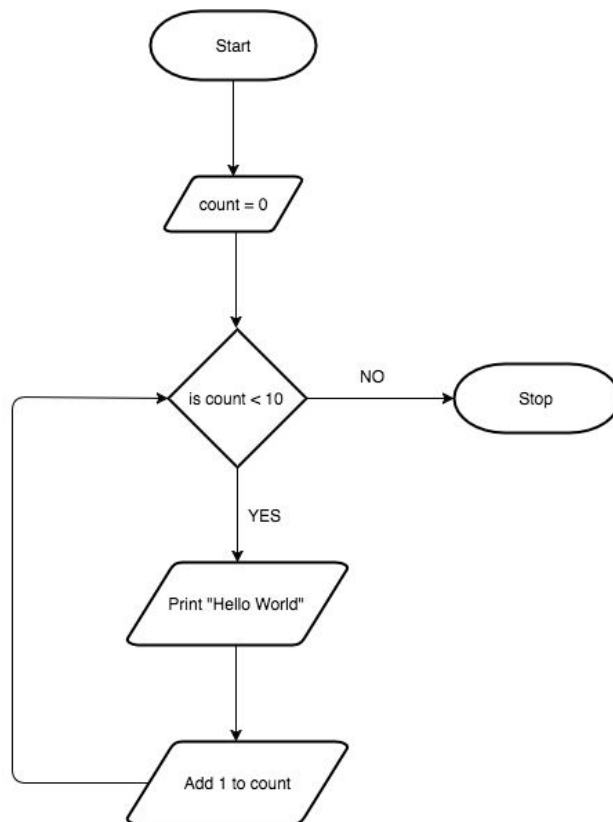
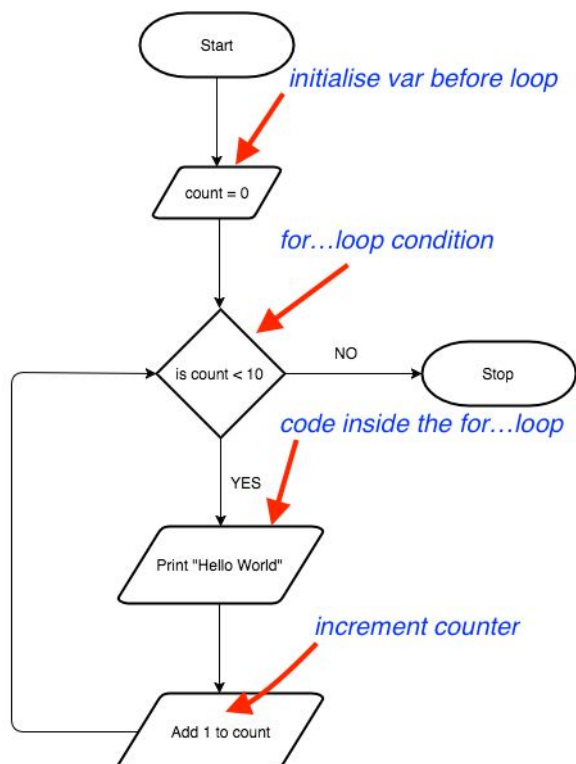Take a look at this annotated example of the same thing:
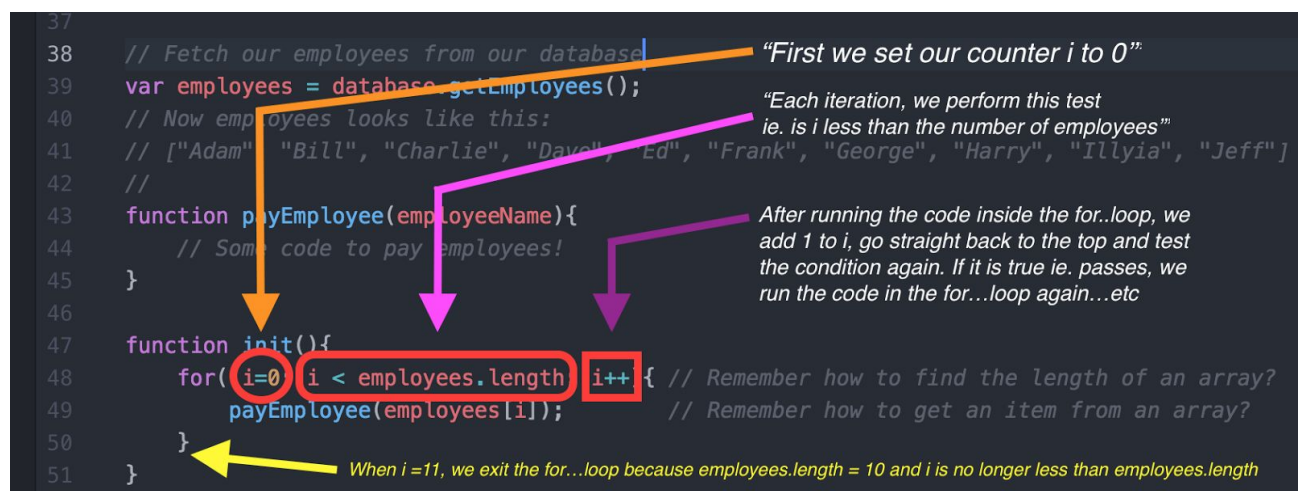
Flow charts are also a fairly familiar way of thinking and could be a very good way of introducing loops to a class of students. The next example shows the most ubiquitous program in history being written first with a flow chart chart. Yes you guessed it - the 'hello world' program!!

In the example flowchart below, we do the following:

- Create a variable called `count` and set it equal to 0
- Check whether count is less than 10
- If it is NOT less than 10 we stop (in other words if it is > 10)
- If it is less than 10 we print "Hello World" to the screen
- We then add 1 to count which means that count is now count+1
- We go back to the test and run the rest of the code again

Start

count = 0

is count < 10 —— NO —→ Stop

YES

Print "Hello World"

Add 1 to count

Here is another way of looking at it:

Start

initialise var before loop

count = 0

for…loop condition

is count < 10 —— NO —→ Stop

code inside the for…loop

YES

Print "Hello World"

increment counter

Add 1 to count

Here is yet another way of looking at it:

```
54      function printHelloWorld() {
55          var count = 0;
56
57          for(var i=0; i < 10; i++) {
58              console.log("Hello World");
59          }
60      }
```

There are so many ways of teaching/explaining loops. I like this one which the originator uses to teach to younger students although I believe you could easily adapt it or use it for older students as well.

"With my younger students I do a very simple activity to show how loops are "easier." I have one student stand in the middle of the room all the way in the back. I tell them to follow my directions. I say "student name take one step forward." I literally repeat that same line about 15/20 times. As I get to the 10th, 11th, 12th time saying it I act so tired. I am taking bigger breaths and my body language is hunched over because I am sooooo tired. Eventually the student reaches me and I am just so tired and out of breath I can't teach anymore. I ask the students to explain why I am so tired. I then ask them to figure out a way to make my directions easier. They soon start re-enacting the steps taken and begin counting how many were taken. Eventually, they determine a much easier way to give the directions and they are so excited to tell me that I should have said "student name take 15 steps forward."

Lesson 7 - Animation and Sprites

We recognise that not all students will want to write lines of code, some may be more inclined to work on the design or the look of something.

There are lots of programs that allow students to create sprites for games. If you are in a public school your students already have access to Adobe's Creative Cloud and Photoshop, Illustrator and Premiere Pro along with the rest of the suite. The are alternatives but the challenge we have found is the steep learning curve.

Consider using Piskel: An open source, cross platform program that has both a cloud and downloadable versions.

With Piskel you can export a sprite or game asset as a .png and reference it in game. Alternatively you can find free sprites via google.

When exporting your sprite make sure you are referencing the correct pixel size and frames. You'll need to know these for your code.
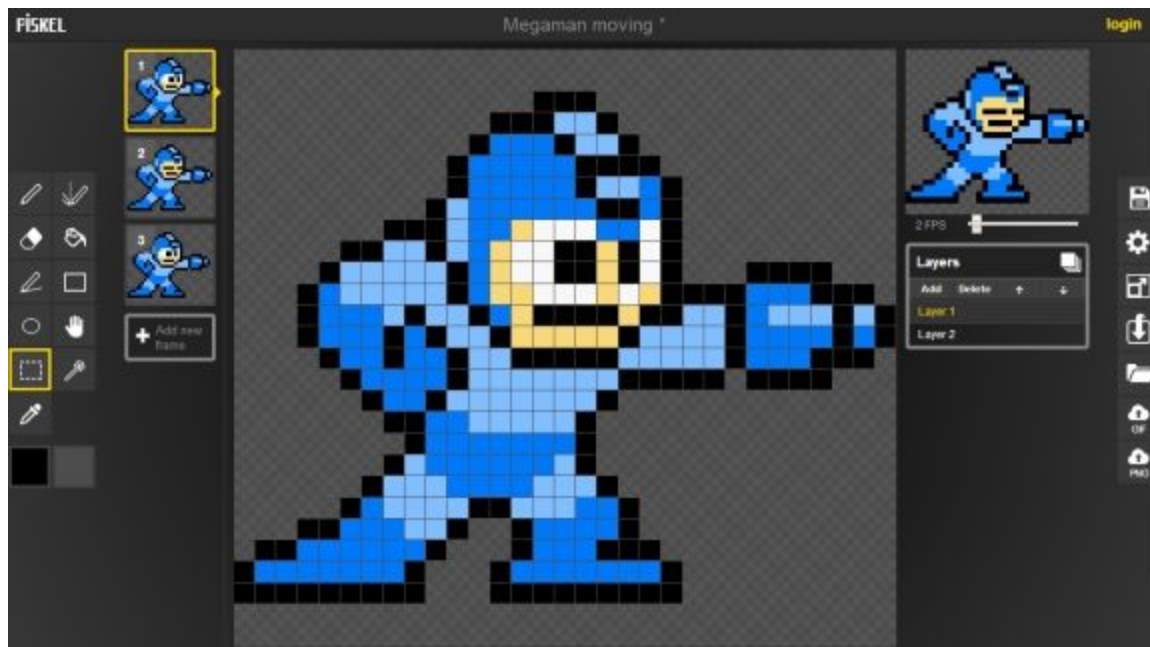
Lesson 8a - Examples of How to use Code in the Classroom

In applying code to your class we suggest you first look at what you want to accomplish, then choose the most appropriate tool for the job. Today we are mostly looking at Javascript because it is one of the most used languages and has thousands of libraries that will assist you in your task.

A code library is like a regular library - (the one students don't know how to use anymore because of google). If we look for the section on engineering we are given all the engineering books needed for the task but instead of writing a book on solar power we reference the page number, paragraph and line of text. A code library is similar in that we use a pre-written library of code to make our coding experience faster and easier.

We will look at a game library later called crafty.js but for now, along with a few examples of how you can use Javascript and C+ in your classroom, are a number of libraries that will make your coding experience faster and easier. All the libraries listed are open source and you can use www.javascripting.com to find many more. At last count on javascripting alone there there are more than 1200 libraries to choose from.

Similarly if you want to program development boards like an Arduino or Raspberry Pi look at the respective websites. There is a heap of prewritten code and examples that will help you program your boards.

Suggestions

    a. Science + Engineering
- i. [Chem] Program an Arduino or Raspberry Pi and take sensor readings for chemical reactions.
- ii. [Physics] Physics game, code physics examples where students change the values to modify and object.
- iii. [Bio] cell growth simulation.
- iv. Data graphing and statistical analysis
  Plot.ly
  eg https://plot.ly/javascript/ & https://plot.ly/feed
  eg https://github.com/d3/d3/wiki/Gallery
- v. Stream data from the space station
- vi. Robotics + IOT (Internet of Things)

    b. Maths
- i. Numeracy
- ii. Data graphing and statistical analysis (eg https://plot.ly/)
- iii. Graphing
- iv. Maths (http://mathjs.org/) - build a Calculator in Javascript !
- v. 2D and 3D animation, isometric shapes

c. Geography
   i. Topological data 2D & 3D
   ii. Data streaming from weather stations
   iii. https://romsson.github.io/dragit/example/nations.html
   iv.
d. English
   i. Numeracy and Literacy game
e. History
   i. Mapping human movement through history
   ii. Building a timeline
f. Design
   i. Webpages + applications (eg Facebook)
   ii. Robotics + IOT
g. Art
   i. Generative art (Dwitter) - a way to use maths and science data in art
   ii. Challenge the idea of the artist
   iii. Robotics + IOT
   iv. Jonathan McCabe.
   v. https://www.flickr.com/photos/jonathanmccabe/sets

Lesson 8b - Coding with Maths and English

Summary

Integrating technology with other subjects is a fascinating idea. In our view, technology is an enabler - it is not the end goal. Technology is useless unless it ultimately makes our lives better - right? Coding is just one aspect of technology but it certainly is a special one because it permeates every aspect of technology. All tech devices need coding otherwise they won't do anything!

Our belief is that code can be brought into any subject on the curriculum with a bit of creativity and inspiration. The goal is to bring the subject in question to life in a different way to how it is normally learned - say from a book or from listening to a teacher/lecturer. Code has the ability to make things dynamic, solve problems faster and more effectively, add visual animated aspects and so much more.

We are going to look at an example based on an article from Education Matters magazine by Byron Scaf, CEO of Stile Education.

The Problem

Each letter in the alphabet is given a value - actually we are going to use values similar to the game Scrabble, but multiplied by a factor of 5

- 5 point: A, E, I, O, L, N, R, S, T, U
- 10 points: D, G.
- 15 points: B, C, M, P
- 20 points: F, H, V, W, Y
- 25 points: K
- 40 points: J, X
- 50 points: Q, Z

You must come up with as many 100 point words as you can!

The Solution

Obviously there are a couple of ways to do this.
1. Stare very hard at these letters and come up with as many words as you can. With this solution, you may get about 10 words if you are lucky.
2. Write a simple program to trawl through a list of words, check the score of the word and add it to a list of 'words that = 100 points'.
We are going to take the second route surprisingly.

Create an Input List of Words

This is pretty simple. All we need to do is create a huge array of words which we will then use to score each word and find the ones that add up to 100 points. The array will look something like this:

```
["a","aa","aah","aahed","aahing"..... and so on and so forth …..
275000 words later….. "zzz", "zzzs"]
```

We will use a skeleton project that you can download from:

https://github.com/nexgencodecamp/codingintheclassroom/archive/master.zip

We will be coding a single function called `generateScores()`.

You will notice that the function is heavily commented. The comments are an exact description of what we will need to code to get the application working.

To test the application, we will be running the index.html file in the words directory.

The rest of the code is already complete. In our function, we will be using:
- Variables
- Arrays - adding to and extracting items from
- We will be calling functions
- Conditional (if/else) statements
- Loops
- We will be printing to the Chrome Dev console

Lesson 9 - Coding a simple Game

Summary

Coding a game is a great way to get young people into coding. Engineering or Computing related courses at University will typically include an exercise to create a game in Java, Javascript or whatever language is being taught. This piques the interest of the student and opens up the possibilities of the language. More often than not it revisits Maths concepts such as Trigonometry (which may or may not be a good thing ;) - I recently had the opportunity of helping a first year student with a game assessment he had been set and was a little confused about - I had to remember the Math! I think it was good for both of us!!

Game Libraries

The options at hand are simple. Either code a game from scratch or use game specific library code to help you. The benefits of using a library are huge. A library will give you most of the functionality you need such as gravity, collision detection, sprites, animation...and much more. Just imagine if you had to code all this.

Having said this, if you find that you are advancing and finding the library very easy to use, you should try to write a game from scratch. To give you an idea of the difference in difficulty, a 200 line game written with a game library backing it up is equivalent to a 2000 line game written from scratch.

Introduction to CraftyJS

CraftyJS (http://craftyjs.com/) is one of the best game libraries around. It contains about 20K lines of code and is designed incredibly well - because of its 'component oriented' nature, it is a fantastic teaching tool unlike some other libraries which although more commercial, have not been designed as well (in this author's opinion).

Using CraftyJS is simple. All you have to do is add a 'script' tag in your HTML file, a bit like this:

<script src="crafty-0.8.js"></script>

We will be building a simple Pong-like game with Crafty… which incidentally was the *first ever* video game to achieve commercial success - in 1972 for Atari.

Creating an Object in CraftyJS

One of the fundamental ideas about CraftyJS is that everything on the screen is an object (pretty much). To create an object (called entity in Crafty) - we do this:

```
Crafty.e();
```

We would describe the above code like this - Using the 'Crafty' object we call the 'e' method/function which returns an object.

So if we wrote it like this:

```
var myObj = Crafty.e();
```

Then 'obj' would end up as a Crafty object.

We can give a Crafty object behaviours or capabilities. In essence, the object we have created above cannot do anything or another way of putting it - the object isn't very useful at all.

To give it behaviours or capabilities we must pass a single parameter into the `e()` function. The parameter is a String and contains all the capabilities that we want. In other words:

```
var objWithPositionAndColor = Crafty.e("2D, Color")

var objWithPositionHTMLElementPositionColor = Crafty.e("2D, DOM, Color")

var objWithPositionAndColor = Crafty.e("2D, DOM, Color, Multiway")
```

To the first object, we give 2D positioning and color.
To the second object, we give the same plus the ability to be created as an HTML element
To the third object, we give the same plus the ability to be moved from the keyboard.

We will pick up more CraftyJS when we code the example...