Client-Side Scripting: JavaScript and JQuery

1. Programming Fundamentals

Programming lies at the core of all modern technology. Before the invention of electronics the core logic of programming was already in use in various applications. The abacus is regarded as the world's first calculator. The same core logic applied when manually operating an abacus, is the same core logic for all programming. Today's programming languages are far more advanced, allowing developers to leap forward in the system's logic, making it simpler to create complex and intricate websites and applications with far less work. However the same core logic as used in the abacus is still being used behind the scenes by the compilers of the language. We will be looking into the history of computers and programmatic languages, our goal is to provide a strong backbone of knowledge about the logic and history of programming. With these fundamental principles in place, you will be able to understand and read all types of code.

2. A Brief History of Computer and Programming Languages

The first computer to be programmed was invented by Charles Babbage in 1822. It was called the Difference Engine (Ferguson, 2000). It was a crank driven mechanical calculator, it's purpose was to calculate tables of polynomials for use in industries like engineering, aeronautics and the military. By providing inputs and manipulating gears in the machine they could produce these mathematical tables (i-programmer.com, 2013). Unfortunately for Charles Babbage he did not live to see the completion of his computer.

This process of inputting commands into a mechanical computer to provide instructions evolved over time into many variations, punch-cards, pins & holes and finally programmatic languages. In 1890 the Hollerith Card Tabulating Machine was created to assist the U.S. Government with collating census data (Cockford, 2011).

In 1942 the E.N.I.A.C. was built by the US Government and it allowed for the first use of programming in an electrical computer. By rewiring and setting switches, the computer could be programmed to complete specific calculations (wikipedia, 2015). 1957 saw the arrival for the first programmatic language FORTRAN (Formula Translating System). Designed and created by IBM, it was used mostly for scientific purposes. It was the first language to introduce states and conditional statements like IF, DO, and GOTO (Ferguson, 2000). Over the last 40 or so years, many programming languages have come into existence, COBOL, LISP, Algol, Pascal, BASIC, C, C++, and JAVA to name a few. These languages have built on the logic providing more methods, operands

and data-types for developers to work with, again increasing productivity and efficiency of writing code.

J.C.R Licklinder of MIT (Massachusetts Institute for Technology), wrote the original plans for the internet in August, 1962, he called it the "Galactic Network". ARPA (Advanced Research Project Agency) later known as DARPA, created the first version of this network called ARPAnet (Gosselin, 2011). Access to ARPAnet was restricted by the U.S. government primarily to academic institutions, scientific research and the military. It took 20 years for the ARPAnet to become publicly available to personal computers and businesses (Gosselin, 2011). Finally taking on the mantle as we know it, the Internet.

The Internet was invented by Robert Elliott "Bob" Khan and Vinton "Vint" Gray Cerf in the late 1980s (Kahn & Cerf, 2015). While at DARPA they worked closely on ARPAnet and were ultimately responsible for creating the protocols which allow data and computers to interact with each other across the internet. In 1990 and 1991 while working at CERN, Tim Berners-Lee created what would become known as the World Wide Web, as way to simply access documents across their network (Gosselin, 2011).

The growth and expansion of the internet into business and homes, lead to the need for GUIs to be developed. This need was met in with the creation of web scripting languages like JavaScript, which was first released in 1995 (Gosselin, 2011).

New programming and web scripting languages are consistently being created for various applications and industries. Existing web languages too are evolving, HTML for instance is in it's fifth release version (HTML5), and has a number of variations like JADE or HAML.

3. What is Programming?

Programming is the logical process an application or website will follow expressed in code. With code we speak directly to the app or website. This code becomes the logic by which an application or website adheres to - in other words, code will instruct the application or website how to behave based on what is being done or engaged with.
In the development industry, the logic of a program or system is referred to as "Business Logic" - this is a representation in code, of how the business system will conduct the logical process of the company's business model or service. In the video game industry it may be regarded as "Game Logic" - the logic of how the game will be played. All applications, electronics, and websites will have logic driving the functions of the program.

As mentioned there are many languages that can be used to code the logic of a game, application or website. Each language has it's own differences in syntax, expressions, conditions, operands and methods that it will be used by a developer/programmer to build up the logic of the app or system. It is common place in the industry to find "Use Cases", these are documents which outline how the application, game or website will be engaged with by the users. Often users fall into two categories, public and administrative. Systems can also be categorised as either public facing or internal.

A business system that allows employees at a business to perform many functions of a company will be regarded as an internal system. Websites which provide a service to the general public is obviously a public facing system. However there are many systems that handle both.

Example:
A blogging/online media company is using a CMS for a blog site, it is both public facing and internal. The system is composed of two major sections each fulfilling different roles. The internal system allows admins to write, post, update and delete articles. While the public facing section allows site visitors to read the articles, sign up and bookmark articles. The system logic will thus dictate which type of user has access to each section as well as providing the functions each user type has to their disposal. We will primarily concerning ourselves with programming for websites. However it is just as important to grasp the logic of programming for a small site with simple functions, as it is to understand the complexity of large system. The logic and programming will remain the same at it's core. If you grasp the fundamentals you will be able to learn other programming languages quickly as well as easily read and understand them.

4. Introduction to Client-Side Scripting

Websites and web applications provide multiple types of functionality. Most often this functionality is required to be coded/programmed by a developer, and will allow the site or app to perform more advanced functionality. This type of code exists in two basic forms, Client-Side and Server-Side.

Client-Side scripting as the name relates, is the process of writing code or scripts for websites and web apps which will only run when rendered on the client i.e. the user's browser. The user's web browser, like Safari, Chrome, Firefox and Internet Explorer will interpret the code and provide the functionality the developer has coded. The processing takes place on the user's computer. This functionality can be executed when the load of the web page is complete, or during an event like a click or any other interaction made by the user. Client-Side scripts can be embedded into common

web files, like HTML, PHP or ASP, but they can also exists as separate files which are linked to the website or application.

Conversely code can be run on the Server-Side of the system, meaning a request is made to the web server and the web server executes the code prior to being loaded, this Server-Side code will then be converted into a format that the user's web browser can understand.

Client-Side scripting allows web designers and developers to add advanced functionality to a website or web app that HTML and other base web languages don't permit.

JavaScript has become the preferred language for Client-Side development as it is present in all browsers. With JavaScript we can manipulate the website and introduce extended functionality, change and replace data, animate and add effects, all of this and more to provide engaging functionality for the user.

5. The Developer Environment

Designers and developers use a variety of software packages, commonly referred to as IDEs (Integrated Development Environment), these will usually consist of code editor, compiler, debugger and GUI (Graphic User Interface). Web designers often opt to use more simplified development tools like, Adobe Brackets, Sublime Text or Dreamweaver. Web developers will often use larger more complex IDEs like Visual Studio and Eclipse. These larger more complex IDEs allow for both Client-Side and Server-Side scripting, while the streamlined and simpler IDEs focus on design and Client-Side, however all programming languages can be written in an application as simple as Notepad (PC) or Text Editor (Mac).

Along with the long list of IDEs and software packages, is the internet browsers, previously some have been mentioned, but the most commonly used browsers are Safari, Chrome, Firefox, Opera and Internet Explorer (Edge as of late 2015). Browsers allow designers and developers to test their websites locally and inspect them for any issues or bugs that are occurring. All modern browsers provide tools to inspect HTML, CSS, and JavaScript files, thus allowing the designer/developer to see the relationship between the different languages and how they are affecting the site. Amongst the many tools browsers provide, is the capacity to set breakpoints in the Client-Side scripts, breakpoints are stopping points, this allows the developer to see what is happening in the code at custom set points.

For the purposes of learning JavaScript and JQuery we recommend the use of Adobe Brackets, Sublime Text or Dreamweaver. Each provides slightly different features, like "Linting" or "Hinting",

code highlighting and formatting however once you are comfortable with code, you should be able to use any IDE to write your code successfully.

6. The DOM

The DOM stands for "Document Object Model" it is a language independent convention used as a representation of interacting objects for HTML and XML documents. The DOM is build up of "Nodes" each representing elements in the structure, these combine to form a tree view, regarded as the DOM tree. JavaScript and JQuery allow us to manipulate the nodes in the DOM and in turn the data associated with nodes can be manipulated.

The DOM's tree structure also allows developers to identify objects relating to one another, these relationships are formed in the hierarchy of the DOM tree. From a particular node in the tree, developers can access the ancestors and descendants of the node, often referred to as parents and grandparents or children and grandchildren.

Lowest point or base point in the DOM tree is the <!DOCTYPE html> from there everything will branch out like family tree. This would be the base ancestor for all other elements in the DOM tree, making all other elements descendants for the <!DOCTYPE html>.

7. Common Elements of Programming Languages

There are a number of universal elements that exist between all programming languages. These elements are similar to the elements of HTML in that, they make up the pieces needed to successfully write code, like id's, classes, and tags in HTML provide the backbone of a website structure and layout.

- Variable - Container which allow developers to store data.
- Identifier - Name assigned to a variable.*
  *Some words are reserved and will cause conflict with the language syntax.
- Expression - A value, variable, combination of variables, operators (operands) and values that produce a result.
- Operands - The variables and values contained in the Expression.
- Operators - Symbols that denote the manipulation of operands. There are number of different types of operators.
- Arithmetic Operators - These perform mathematical operations, addition, subtraction, etc.
- Assignment Operators - These perform assignment of values to variables. i.e. `var myVariable = 5;`

- Comparison Operators - These are used to compare and evaluate whether a value is greater than another. i.e. `5 > 3`
- Logical Operators - Used to compare two or more boolean operands and test for equality. i.e. `!== 5` (not strictly equal to 5).
- Special Operators - Special operators are part of the code syntax and are used to make the code understandable to the browser or PC.
- Literal - A fixed value, like a number or string. Sometimes called a "static" value.
- Event - An action performed by a user (click), or by the script or a set of criteria is met by the code or a variable's value, i.e. If my "`day`" variable is equal to 19 send a message saying "`Happy Birthday`" to the user.
- Event Handler - Code that executes in response to a particular Event.
- Method - A set of code written as an executable procedure defined on an object.
- Function - A set of code written as an executable procedure not defined on an object.
- Function Definition - The code within the function that make up the procedure of the function.
- Parameter - A variable which is declared in a function definition but passed into the function's parentheses, thus defining the number or data type of objects to be passed into the function when executed.
- Call - Ask the browser/computer to execute a function or method.
- Arguments - Variables and values passed into a function's parentheses.
- Variable Scope - Variables can be defined in two scopes, Global or Local. A global variable can be used in all parts of the code. Local variables can only be used within the function it was declared in.
- Data Types - A specific category of data that a variable can hold. There are five primitive types of data in JavaScript. Number, Boolean, String, Undefined, and Null.
- Primitive Types - Data that can only be assigned a single value.
- Number - Positive or negative numbers with or without decimal places or a number written in using exponential notation.
- Boolean - Logic value for true or false.
- String - Alphanumeric numbers and letters i.e. "This is a string of text" or "42".
- Undefined - A non-existent variable, or has no value or cannot resolve the expression.
- Null - An empty value.
- Array - A list of variables, objects or values, always beginning at 0.
- Statements - Commands that are used when the app or site code needs to make a decision. There are many statements that can be utilised when asking the system to make decisions based on values or events.
- If Statement - This is conditional statement to check if the arguments provided evaluate as true.

- If Else Statement - If the conditional statement does not evaluate to true, do something else. This can also be nested in an IF statement.
- Switch Statement - Allows a developer to execute a particular set of statements based on value of a specific expression.
- While Statement - This is a loop statement, it will repeat a series of statements as long as the conditions of the expression is met.
- Do While Statement - This is another loop statement, it will execute a statement or series of statements once, then repeat as long as the condition is met.
- For Statement - Also known as “for loop”, this is used to repeat a series of statements as long as the conditional expression evaluates to true.

8. JavaScript & Basic Syntax

JavaScript is an Object-Orientated, loosely typed programming language, this means it is a programming language that allows us to interact with HTML and data as though they are individual objects. JavaScript is also a dynamic programming language, this means it is high-level programming language, which can execute code in real time as opposed to being compiled first like many Server-Side programming languages.

The syntax in a programming language are the rules about how to write code, it is the equivalent of the grammar for a spoken language.

Simple Variable

Below is the most simple syntax to declare a variable in code. We will touch on more advanced methods to declare variables in the next course. Variables are core to the syntax without them it would be very hard to write successful code, this syntax will become habit in time.

```
var myNewVariable = 42;
```

The variable is started with the shorthand “var” followed by the chosen identifier and set with an assignment operator of “=” to the value of “42”

Variable with an Expression

```
var myExpression = 42 + 6;
```

This variable has a simple operand expression, it simply adding two values with the use of an arithmetic operator.

Variable with Operand Expression of Two Variables

```
var priceOne = 42;
var priceTwo = 6;
var totalPrice = priceOne + priceTwo;
```

The first and second variables are provided with literal (static) values, the third variable "totalPrice" uses an expression to add the two variables together to create the total.

Functions

You will be using functions in almost all the coding you do. You will become very familiar with this syntax in time.

```
function mySpecialFunction(parameterOne, parameterTwo) {
      // here we will place our code statements
}
```

Important!

Remember all parameters defined in a function like "parameterOne" are variables in the scope of the function. See variable scope.

Calling Functions

For a function to be run it must either be called in code or run as method on an event or action. To call the function I've declared previously I would simply call it when I need it to run. This may be in a condition or just at the end of the code.

```
mySpecialFunction();
```

To call a function simply give the name of the function - remember to check spelling - with parentheses at the end and semi-colon to close the statement. Functions can also call other previously defined functions, this is referred to as "Nested Functions".

```
function myGlobalFunction (scope) {
      scope = "this variable is in the scope of the first function";
      console.log(scope);

      function changeVariableData () {
```

```
            scope = "this function can change the value of the variable";
            console.log(scope);
        }
        changeVariableData();
}
myGlobalFunction();
```

In the function above we can see that we have nest the the function "`changeVariableData`" inside the function "`myGlobalFunction`". You will also notice the that our parent variable ("`myGlobalFunction`") calls the nested function before it is closed and outside the "`myGlobalFunction`" function, the parent function is called.

When this set of nested functions is called, it will change the value of our argument (variable) "`scope`" to a string value and then log it to the console window. Then the function will call the nested function "`changeVariableData`", which will in turn change the value of our original variable "`scope`" to a new string value and then proceed to log it to the console window.

Notice that the variable "`scope`" has only been declared once as an argument of the function. This variable has a local scope to the parent function, however because the second function is nested and is a child of the first function ("`myGlobalFunction`") the variable has a global scope with regards to the child function. Any functions nested in the parent will have global access to the variable "`scope`".

Conditional Statements

As mentioned previously, there are a number of conditional statements used in programming. We will be looking at the most commonly used "if" statement to provide us with an idea of the basic syntax that is similar to all conditional statements.

```
if(mySpecialVariable == 4) {
        console.log(mySpecialStatement);
}
```

Conditional statements check to see if specific arguments are met. In the example above our condition is simply to check to see if the variable - "mySpecialVariable" - is strictly equal to a value of 4, if the argument passed into the if statement is met, we the log the variable value to the console window.

```
if(myOtherVariable != 4) {
      console.log("It's not equal to 4!");
}
```

In the second example above we are created an argument that is using a logic operator.
The logic operator "!=" is used to check if our variable "`myOtherVariable`" is "not equal to" a value of 4. Should the "`myOtherVariable`" not resolve to a value of 4, it will log to the console window a string that reads "It's not equal to 4!".

Important!
Remember arguments are the variables and values in an expression passed into the parentheses of a function or conditional statement.

9. Advanced JavaScript Syntax

Variables

Declaring Variables
In JavaScript we can declare an "empty" variable which will act as a placeholder until we assign data or value to it. Below is basic requirement for the JavaScript to understand that a variable had been declared.

```
var myVariable;
```

We can also declare several variables, it can be done in two different manners, first using several lines of codes each beginning with var, or writing the keyword "var" once and then use commas as separators between variable names:

```
var myFirstVariable;
var mySecondVariable;

//or (this is a single line comment in code, it will be ignored at runtime)

var myFirstVariable, mySecondVariable;
```

It is good practice to create clear readability of in code, as well as grouping your code into logical block that relate to each other. However this becomes more complex when "scope" of the code begins to play a role in your code blocks.

Naming variables

As mentioned is the first chapter, any programming language comes with its best practices, its own rules defining the way you are supposed to write your code. Thus, even if JavaScript allows you to name your variables in almost any way you would like it, you should respect the following rules:

1. The name should describe shortly the meaning of the value of the variable. One could use "`var userName;`" for a variable containing the name of the user.

2. The camelCase practice should be used. Variables names containing a compound of words or a phrase are named such that the first word begins with a lowercase letter and each next word begins with a capital letter.

```
var thisIsCamelCase;
```

3. Avoid abbreviations in which meaning is not obvious to the data or value.

4. If a variable contains a value or data that will not changed i.e. "static data", at any point of the execution, it's common to name it with capital letters separated with underscores: "`var BODY_TEMPARATURE = 37.6;`" Such a variable is described as a constant, static or literal.

Be aware that there are exceptions to these conventions as there is a non-exhaustive set of rules, however these conventions are put in place to aid in maintaining and updating code for yourself or any team member. Keep the logic of the system in mind as well. The logic of the system will act as guide to the code structure, naming conventions and readability.

Assigning Value to Variables

The next step is to assign a value to a variable.

```
var myVariable;
myVariable = 6;
```

The first line of code declares a variable named "myVariable"; the second line assigns the value six to that variable. So after the execution of that line of code, the variable will contain the value six.

It's quite common to declare a variable and assign it a value at the same time. This is a condensed way to write the previous code:

```
var myVariable = 6;
```

Also, you can assign the value of a variable to another variable.

```
var myVariable = 6;
var myOtherVariable = myVariable;
```

"`myOtherVariable`" now contains the value six. The assignment operation will always evaluate the expression on the right side of the equal sign and then assign the value to the variable on the left side of the equal sign. Thus, a code line like "`6 = myVariable`" is wrong and will give an error.

Types of Variables

So far, we only saw variables holding numeric values, such has 6. But a variable can contain many different types of values. Let's have an overview.

Numbers

They can be positive or negative, and have decimal values:

```
var count = 10;
var pi = 3.14;
```

Common arithmetic operations can be applied to numeric types, using operators like +, *, -, /.

```
var ten = 10;

var twenty = ten + ten;
//twenty === 20;

var hundred = ten * ten;
//hundred === 100;

var zero = ten - ten;
//zero === 0;


var one = ten / ten;
//one === 1;
```

The operator modulo, "%", gives the remainder of a division.

```
var remainder = 36 % 10;
//remainder === 6;
//10 * 3 + 6 = 36
```

Modulo is often used to test if a number is odd or even, as an odd number modulo two will be one and an even number modulo two will be zero.

```
var remainder = 12 % 2 ;
//remainder  === 0 ;

var remainder = 11 % 2 ;
//remainder  === 1 ;
```

Strings

A string of characters, usually just described as a string, represents text. When declaring a string value, quotes or double quotes must be wrapped around the text. The quotes or double quotes inform JavaScript that it is not a number value but an alphanumeric value of text and characters.

```
var userName = "John Doe";
var userName = 'John Doe';
```

A simple operation that you can apply on strings is concatenating them two chains using the addition operator.

```
var firstName = 'John';
var lastName = 'Doe';
var userName = firstName + ' ' + lastName;
//userName === 'John Doe';
```

To convert a string to lowercase letters, or capital letters, the medhods "toLowerCase()" and "toUpperCase()" may be used. We will talk more about functions later in this course, but the example below shows how to use these two.

```
var city = 'Cape Town';
city = city.toLowerCase();
//city === 'cape town';
```

```
city = city.toUpperCase();
//city === 'CAPE TOWN';
```

You can easily use strings to interact with HTML elements using jQuery.

```
var pColor = 'blue'; $('p').css('color', pColor);
```

That code will change the color of every "`<p>`" element to blue.

Booleans

A boolean variable can only have two values, true or false.

```
var isItAGreatCourse = true;
var amIBored = false;
```

Type Conversion

Sometimes, you need to convert one type to another. The most common conversions are to convert a number to a string and a string to a number. To convert a number to a string, you use the method "`toString()`".

```
var one = 1;
var oneAsString = one.toString();
//oneAsString === '1';

var eleven = oneAsString + oneAsString;
//eleven === '11';
```

To convert a string to a number, you use the method "`parseInt()`".

```
var twenty = '20';
var two = '2';
var result = twenty + two;
 //result === '202';

twenty = parseInt('20');
two = parseInt('2');
result = twenty + two;
```

```
//result === 22;
```

If the number has decimal places, you must use "`parseFloat()`" to get the exact value.

```
var decimal = parseInt('3.14');
 //decimal === 3;

decimal = parseFloat('3.14');
//decimal === 3.14;
```

If you try to convert a bad string to a number, you will get the result "`NaN`", not a number.

```
var notANumber = parseInt('abc');
 //notANumber === NaN;
```

Typeof

You can know the type of a variable using the operator typeof. When you declare a variable, it has the type undefined.

```
var myVar;
var myType = typeof(myVar);
//myType === 'undefined';
```

The types are dynamic, your variable can be a string at some point of the code execution and an integer later.

```
myVar = 1;
myType = typeof(myVar);
 //myType === 'integer';

myVar = '1';
myType = typeof(myVar);
//myType === 'string';
```

Variable Scope

JavaScript uses what is called functional based or function scope. This means that scope changes in relation to functions in code. JavaScript and all programming languages have code which exist in

blocks and structures. These blocks of code, like functions and objects, have access to particular parts of the rest of the code, this deemed the scope.

For instance a variable declared inside a function has a scope that is local to that function, this is also referred to as a local variable. A variable declared outside the function is regarded as having scope that is global to the function and rest of the code, referred to as a global variable.

```
var myGlobalVariable = "this variable has a global scope";

function myLocalFunction() {
      var myLocalVariable =  "this variable has a local scope";
}

// the location where the variable is declared determines its scope
```

Important!
Local variables are deleted when the function they were declared in is done running. Global variables are deleted when the page is closed.

Conditional Statements Syntax and Structures

IF Statements

Sometimes in your code, you may want to perform some action only if a certain condition is met. The if statement exist to allow you to realize that. This is how it works:

```
if(condition) {
      //do something
}
```

The condition is a boolean value, which means that it can be only true or false. The code written within the brackets will be executed only if the condition is met, thus only if the condition is true.

Else Statements

The else statement is used to perform a different action if the condition used in the if statement is not met. This is how it works.

```
if(condition) {
      //do something
```

```
} else {
      //do something else
}
```

If the condition is true, the code written between the do something brackets will be executed. Otherwise, the code written between the do something else brackets will be executed. You can combine several if else blocks if some complex logic is required.

```
if (condition) {
      //do something
} else if (second condition) {
      //do something else
} else if (third condition) {
      //do something different
} else {
      //do default action
}
```

Operators

Comparison Operators

Comparing variables is required to build conditions. The comparison operators compare two variables and return true or false depending on if the comparison condition is met.

Example with the "equal to" operator

The "equal to" operator is written using a double equal sign, == , or using the triple equals sign returns true only if the expressions on each side of the operator are equal, false otherwise. The code snippet below shows a possible usage of the is equal operator.

```
userGender = userGender.toLowerString();

if (userGender == 'f') {
      title = 'Ms.';
} else if (userGender == 'm') {
      title = 'Mr.';
}

var userWelcomeMessage = 'Hello ' + title + '  ' + userName;
```

Logic Operators

Logical operators are used to manipulate boolean values. You can use them to build more complex conditions. The “and” operator is written using a double ampersand, &&, and returns true only if the expression on its right side and the expression on its left side is true. The code snippet below show a possible usage of the and operator.

```
if (age >= 10 && age < 20) {
      category = 'teenager';
}
```

Functions

In programming, a function is a section of code which performs a specified task. For example, the code below declares a function named sayHelloWorld, which displays a popup with the message “Hello World”. The function code is all the code included in the brackets following its declaration.

```
function sayHelloWorld() {
      window.alert('hello world!');
}
```

The code inside of a function is executed when the function is called. This is done the following way.

```
sayHelloWord();
```

The purpose of functions are multiple.

- Organize your code. When there is a complex block of code, you can just read the name of the function and know what is its purpose, for example “sendEmail”.
- Follow DRY (Don’t Repeat Yourself) methodology
- Create libraries - A library is a set of functions that you can see as a toolkit for its user.

Parameters

The function declaration can be completed if needed so that it accepts parameters. Parameters are used to customize the function behaviour according to the parameter value, which is called the argument. For example, the code below declares a function named “sayHelloToUser”, which displays a popup with a message saying hello and the user full name.

```
function sayHelloToUser(userFirstName, userLastName) {
      alert('hello ' + userName + ' ' + userLastName);
}
```

Calling that function with the arguments 'John' and 'Doe' will result of a popup displaying 'hello John Doe'.

```
sayHelloToUser('John', 'Doe');
```

Return Value

Functions can also return a value. This is done by using the return statement. The code below declares a function named add which takes two parameters, x and y, and returns the value of the two parameters added. So calling sum with 2 and 3 as arguments will return the value 5.

```
function add(x, y) {
      return x + y;
}

var sum = add(2, 3);
//sum == 5;
```

When you reach the return statement, the function execution is stopped and continues where the function has been called before. So if we execute the code below, only one popup displaying 'out add' will be displayed.

```
function add(x, y) {
      return x + y;
      alert('in add');
}

var sum = add(2, 3);
//sum === 5;
window.alert(add); //show sum value in alert box *see outputting and
displaying data section
```

The final step of the code above outputs the return value in the "window.alert()" method. This causes an alert box to be generated by the browser and the value will be displayed to the user.

Functions as Variables

Like a number or a string, a function is a variable designed by its name. As a variable, you can assign it, to another variable.

```
function convertToUSD(rand) {
	return rand * 0.091;
}

function convertToEuro(rand) {
	return rand * 0.070;
}

var converter = convertToUSD;
var money = converter(1);
//money === 0.091;

converter = convertToEuro;
money = converter(1);
//same line, but now money === 0.070;
```

Outputting or Displaying Data

There are different ways to display data with JS, there are four basic methods to do so but each have their own specific uses. These are considered to be DOM methods as well.

- window.alert() - creates an alert box to display value in.
- document.write() - writes to the html output, best used for testing.
- .innerHTML - the most common and recommended method to display data to DOM elements.
- console.log() - display data in the browser console log, also good for testing.

Just like functions return value, and variables can have values, these can be written or displayed to a webpage. Below are examples of each type of display method.

```
var warning = "This is a warning Alert Box!";
window.alert(warning); // displays the string value in an alert box to user

var myTestInfo = 5 + 9;
```

```
document.write(myTestInfo); // displays the value of the expression in the
variable myTestInfo
```

Below we are using the document.write() method to show an example of displaying the value as part of a function. This can also be done with other display/DOM methods.

```
function addSales() {
      var salesPerWeek = 5;
      var salesPeriod = 12;
      var totalSales = salesPerWeek * salesPeriod;
      document.write(totalSales); //display value using write method.
}

addSales(); // runs the function
```

When using the "innerHTML" method, we can push the values into elements like divs or paragraphs.

```
<h3 id="user">
</h3>

var firstname = "John";
var surname = "Doe";

document.getElementById("user").innerHTML = firstname + " " + surname;
// here we use the innerHTML and "+" sign to bind 2 variables values to give
the final result - we leave quotes with space to create space between the
values of the variables.
```

Lastly we have "console.log()" method which writes to the browser console. To access the browser console - right + click(win) or ctrl + click(mac) on your web page and select inspect element. Find the console button and you should see the value in the console displayed.

```
console.log(8 * 4);
```

These are the base commands for writing data to a webpage, combining this with all the other syntax will allow you to update information and data where you need to.

Data Structures

The object concept is the core of most modern programming language. Developing code in these languages is called OOP, object oriented programming.

What is an object?

Everything that you can see in this room can be described numerically as an object. For instance, a book. A book has some properties : a title, an author, a number of pages, a color, a state (open or closed, the page at which it is open).

You can also perform some actions with a book : open it, close it, read it, burn it. An object, like in real life is a data structure containing properties describing the object and its state and methods corresponding to the actions you can perform with the object. An object property correspond to a variable, and an object method correspond to a function.

Declaring an object

There are several ways to declare an object, but let's just focus on the one using a prototype. A prototype is a special function used to create an object. Using the book analogy, we can say you are designing a website that sells book. You need also properties like the price of the book.

The following prototype can then be written :

```
function Book(title, author, price) {
      this.title = title;
      this.author = auhtor;
      this.price = price;
};
```

That prototype instantiates a book object that contains three properties and no method. Note that a slightly different name convention applies there: the prototype's name begins with a capital letter.

```
myBook = new Book('JavaScript for Dummies', 'John Doe', 299);
```

Properties

Our object properties are now accessible using the following way :

```
alert(myBook.title);
myBook.price = 199;
```

Another way to access a property is by its name :

```
alert(myBook['title']);
myBook['price']=199;
```

The advantage of accessing a property by its name is that it is more dynamic. For instance, you could ask an user to prompt the name of a property and then display it.

```
var propertyName = prompt();
//prompt is a function that returns an user input string
alert(myBook[propertyName]);
```

"This" and New Keywords

First, in the prototype, the keyword this is used. It refers to the owner of the function, so to the object we are currently creating. Second, when we declare "`myBook`", the keyword new is used. It indicates to instantiate a new object with a defined constructor.

Object Methods

The book has several properties but no method. Our site is required to sell books, the obvious action we can perform with a book is to buy it. A function managing the buying action must then be added:

```
Book.prototype.buy = function(){
      alert('the book' + this.title + ' has been bought');
};
```

Once again there are several ways to declare an object method and you may find something different in some other developer's code. Without losing ourselves into too much technicity, this one, using Object.prototype, is generally the one offering better performances and much more flexibility.

Now whenever the function buy is called on a book, a popup confirming the buy will be displayed.

```
myBook.buy();
```

Date Object

Let's have a quick overview of a common object type, the Date object. JavaScript offers a very handy object type to manipulate dates. First, let's see how to declare and display a date.

```
var d = new Date(); windows.alert(d.toDateString());
```

The first line of code instantiates a new date. As we provide no argument, d is initialized with the current time of the computer clock. The second line displays a message box with the date converted to an human readable string, so it could be for instance 25/12/2031, if the computer clock is set on Christmas 2031.

There are plenty different ways of converting the date to a string. toLocaleString will convert it accordingly to your computer locale settings (could be in arabic for instance), "`toTimeString`" and "`toLocaleTimeString`" will give the time of the days, etc. A date can also be instantiated using a specific day and time :

```
var d = new Date(2031, 25, 12, 6, 30, 0, 500) ;
```

This creates a date initialized at Christmas 2031, 6:30:00AM + 500 milliseconds.
You can also set or get a specific element of a date, like the day, the hour or the month.
For instance, this code snippet calculates an expiration date two weeks after the current time.

```
var d = new Date(); d.setDay(d.getDay()+14);
```

For further reading see Mozilla's documentation regarding the date object.
https://developer.mozilla.org/

Arrays & Loops

An array is a special variable used to group a collections of variables. Working with arrays generally involves using loops to iterate through the array, as we will see in this chapter.

Declaring an array

An array is declared using brackets and each element of the array is separated by a comma. For instance, the following array groups three strings.

```
var personList = ['Anna', 'Bob', 'Chris'];
```

Technically, an array is an object type. Thus it has some handy properties like its length.

```
var length = personList.length;
//length === 3
```

Note that the following code will create an empty array :

```
var empty = [];
```

Accessing Elements in an Array

A single element of the array can be accessed using its index number. The index is the position of the element in the array, starting with zero. Anna's index is zero, Bob's index is one, and Chris's index is two. The brackets operator allows to access a particular element.

```
var person = personList[2];
//person === 'Chris';

personList[1] = 'Byron';
//personList === ['Anna', 'Byron', 'Chris'];
```

Adding and Removing Elements from an Array

JavaScript arrays are dynamic. It means that as we already saw you can modify a particular element of it, but also that you can add or remove elements from it. This is done using the following functions.

- Unshift / add element to the beginning of an array
- Shift / removes element from the beginning of an array
- Push / add element to the end of an array
- Pop / removes element from the end of an array
- Splice / add or remove element from the middle of an array

The following examples describe how to use these functions.

```
var personList = ['Anna', 'Bob', 'Chris'];
//Array of people

var element = personList.shift();
```

```
//Anna is removed from the array

var length = personList.unshift('Andile');
//Add Andile to beginning of the array

element = personList.pop();
//Removes Chris from the array

length = personList.push('Craig');
//Add Craig to the end of the Array

element = personList.splice(1,0, 'Zoe');
//Add item 'Zoe' at the position one

var removed = personList.splice(1, 0, 'Zoe');
//personList === ['Andile', 'Zoe', 'Bob', 'Craig']
//removed === [] thus empty array
//Removes two items at the position one removed = personList.splice(1, 2);
//personList === ['Andile', 'Craig']
//removed === ['Zoe', 'Bob']
```

Loops

In programming, a loop is a code section that is executed repeatedly until a certain condition is met.

While loops

In JavaScript, the simplest loop is the while loop. The while loop executes some code while a certain condition is true. For instance, the following code will be executed while the length of the personList array is greater or equal than zero, thus while the array contains elements.

```
while(personList.length>0) {
      windows.alert(personList.pop()) ;
}
```

At each iteration, the last element of "`personList`" is popped and displayed in a popup. When eventually the array is empty, the condition "`personList.length > 0`" becomes false and the loop stops.

I'm sure you already came across the following annoying situation. You are quietly browsing a web page, when suddenly it freezes and become unusable until eventually the browser displays a message like "`A script on this page may be busy, or it may have stopped responding. You can stop the script now`". Why does this happen? In most of the cases, it's because JavaScript is stucked in what is called an infinite loop. The condition to stop the loop is never met, thus the loop is executed infinitely.

```
while(True) { }
```

The condition "`True`" will be true forever. That's a pretty obvious case but keep in mind that loops can in many ways be at the origin of bugs.

For loops

A for loop is a slightly more complex while loop, but the general principle is the same.
Let's examine the following for loop to see how it works.

```
for(var i=0; i<3; i++) {
      alert(i.toString()) ;
}
```

Executing this snippet of code will display three popups with the messages '0', '1' and '2'.

How does it work?

When we enter the "for loop", the code before the first semicolon, "`var i = 0`", is executed. It's called the loop initialization. Then, the condition between the two semicolons, "`i < 3`", is evaluated. If that condition is true, which is the case so far because now i equals zero, the code between the two brackets, thus the popup message, is executed.

That's when the first message box is displayed. After that, the third statement, "`i++`", is executed. "`i++`" means "`i=i+1`", it increments the value of "`i`", which was zero and is now one. Then the loop starts over again : condition evaluation, if it's true loop code execution, and incrementation of the value of "`i`". When "`i`" reaches three, the condition "`i<3`" is false and the loop stops.

For In Loops

The "for in loop" is a special loop which allows you to iterate to every property of an object. At every iteration of the loop the value before the keyword will be equal to the property value.

```
myBook = new Book('JavaScript for Dummies', 'John Doe', 199)
```

```
for(prop in myBook) {
      alert(prop + ' : ' myBook[prop].toString())
}
```

The previous code will display a popup for every property of the book object, with a message including the name of the property and its value. So if the Book prototype is the same as the one from the previous chapter, it will be 'title: JavaScript for Dummies', then 'author: John Doe', and eventually, 'price: 199'.

Iterate through Arrays with Loops

Practically, one of the most common usages of the loops is to use them in combinations with arrays to iterate through it. If you need to access to each element of an array of length equals to five, you will have to access the element zero, one, two, three and four.

Practically, it's again an example which is the better explanation. The following code will display a popup for each person in personList, containing the person's name :

```
for(var i = 0; i < personList.length; i++) {
      alert(personList[i]) ;
}
```

At the first iteration, "`i`" equals zero, so the first element of the array, "`personList[0]`", is displayed. At the second iteration, "`i`" equals one and "`personList[1]`" is displayed. This goes on until the condition "`i < personList.length === False`", thus when all the elements have been displayed.

Selecting DOM Elements with JavaScript

JavaScript allows us to manipulate the HTML and values or data displayed on a website, it allows us to add or remove and update the information on the fly. However it's useful to be able to identify where in the HTML this will occur. We would obviously need to inform our code where the data is and how to change it. For this reason we need to know how to select HTML elements.

JavaScript allows the selection of HTML elements via numerous identifiers, i.e. HTML tag names, classes applied to HTML tags and IDs applied to HTML tags.

Basic JavaScript Selector Methods

1. document.querySelector(selector) - selects HTML elements by first matching class name, ID or tag name
2. document.querySelectorAll(selector) - selects all HTML elements by matching class name, ID or tag name
3. document.getElementById(idname) - selects a single HTML element by ID
4. document.getElementsByTagName(tagname) - selects HTML elements by tag name
5. document.getElementsByClassName(class) - selects HTML elements by class name

Each HTML element is regarded by JavaScript as a Node in the DOM. Using the query selectors above we can identify these nodes and use JavaScript to manipulate them or affect the data they contain.

```
var userName = document.querySelector("#username");
// select a div with an ID of "username"
```

If know exactly what type of identifier we need to use to select the node we can use another selector method.

```
<div class="user-name">John Smith</div>
// the HTML div element we want to select with a specific class name

var userName = document.getElementByClassName("user-name");
```

We can also select HTML elements using their relationships to each other. Child to parent, parent to child or even siblings relationships.

Child and Parent DOM Relationships

A child to parent relationship is characterised by being able to identify the parent and thus asking JavaScript to select the child element in the HTML or DOM hierarchy. A parent to child relationship is characterised by being able to identify the child and then asking JavaScript to select the parent element of the child in the HTML or DOM hierarchy. Siblings are child elements which exist inside the same parent element. Siblings ask JavaScript to identify the neighbouring HTML elements.

```
<div class="heading">
      <h2>The JavaScript Document</h2>
</div>

var myText = document.getElementByClassName("heading").childNodes[0];
```

```
// here we are selecting the first child <h2> in the div with the class
"heading"
```

There are a number of different ways to select or identify nodes in JavaScript. If we wanted to create new elements or nodes in the DOM we can use a number of JS methods to do so.

Dynamically Adding Element

JavaScript allows us to create new DOM nodes that can be inserted into our webpages. The simplest method is as follows.

```
var element = document.createElement(tagName);
```

Above we are asking the document or DOM to use the "createElement" method to create a new HTML tag of our choosing.

```
<script>
      var heading = document.createElement("h1");
      var headingText = document.createTextNode("Hey look at me! I'm
alive!!");
      heading.appendChild(headingText);
      document.body.appendChild(heading);
</script>
```

Here we are simply creating a new "h1" tag that is empty and then filling it with some text. For this to happen we need to create a DOM text node then add it to the heading "h1". We can also do it another way like below.

```
<script>
      var newHeading = document.createElement("h1");
      document.body.appendChild(newHeading);

      newHeading.innerHTML = "I'm also alive! Hooray!";

</script>
```

In the example above instead of creating a text node in the DOM we simply appended the "newHeading" element we created to the end of our page and only after it was created on the page added the information to the HTML using the "innerHTML" method. This is a nice simple way to do

it quick however it may not always work. In some cases you will need to be more specific about where and how you display the values on the page. The following examples will illustrate.

Dynamically Changing Elements (Nodes)

In JS we are able to manipulate the DOM nodes as you have seen above, we can create, remove and replace nodes or elements as we need to using a number of basic methods in JS.

- .appendChild() - add element to another parent at the end
- .removeChild() - remove a child from a parent, we must remember to tell the DOM which parent we want to remove the child from
- .replaceChild() - replace one element with another in a parent element
- .insertBefore() - helps us insert the child in the parent before other parents.

These 4 methods are the commonly used to manipulate child elements. In the previous example we showed how to append a new paragraph to a div with an id of “myArticle”. This is one of the most simple methods to add a child node to an existing element on the page. The other methods are a bit more strict and require that we inform the DOM of specific information first.

In the case of the removeChild() method we need to inform the DOM which parent we want to remove the child from. Otherwise we will get an error as it won’t know which child node to remove. In the case of replaceChild() we need to list the node or element which needs to be remove and then indicate in the script which one will take it’s place. Similarly when using the insertBefore() method we need to tell the DOM before which child node of the parent we need it to insert the new node or element.

Appending Child Node

Example below shows how to append a child node to an existing div. Basically we are adding a new paragraph tag to an existing div.

```
<div id="myArticle">
    <h2>My Article</h2>
      <p id="firstParagraph">This is my original paragraph and below this I
will dynamically insert a new paragraph tag with JS.</p>
</div>

<script>
```

```
	//create new paragraph tag
	var paragraph = document.createElement('p');
	//create text node to put in p tag
	var newNode = document.createTextNode('Add this new text node to the paragraph tag.');
	//append the child to the p tag - add the text to the paragraph tags
	paragraph.appendChild(newNode);
	//identify the element on page we want to add it to - the div with id myArticle
	var element = document.getElementById("myArticle");
	// add the paragraph to the div myArticle
	element.appendChild(paragraph);
</script>
```

Replace Child Node

In the code below the new "paragraphC" element should appear before the first paragraph in the "myArticle" div.

```
<div id="myArticle">
	<p id="paragraphA">This is a paragraph.</p>
	<p id="paragraphB">This is another paragraph.</p>
</div>

<script>
	var paragraphC = document.createElement("p");
	var myNewText = document.createTextNode("This is new text for the new paragraph tag.");
	paragraphC.appendChild(myNewText);

	var parent = document.getElementById("myArticle");
	var firstChild = document.getElementById("paragraphA");
	parent.replaceChild(paragraphC, firstChild);
</script>
```

Removing Child Nodes

When removing we need to be very specific as the DOM will need to know the parent of the element we are looking to remove.

```
<div id="simpleDiv">
      <p id="p1">This is a paragraph.</p>
      <p id="p2">This is another paragraph.</p>
</div>

<script>
      var parent = document.getElementById("simpleDiv");
      var child = document.getElementById("p1");
      parent.removeChild(child); // remove "p1" from "simpleDiv"
</script>
```

Once we have new elements and data on the webpage we need to learn to style them and interact. Next we will look at CSS properties and Events.

CSS Manipulation with JavaScript

With JavaScript we can also manipulate the attributes and properties of DOM elements, CSS rules are regarded as properties of the element. Meaning with the correct code we can adjust CSS values of different elements, based on functions, conditions or events. The base structure is as follows:

```
document.getElementById(id).style.property name = new style
```

Above is the base syntax for adjusting an element's style (CSS). First we identify the element by it's ID and then we use the ".style" method to tell the DOM we will be making adjustments to those properties. After the style method we tell the code which property name to adjust. These are the same or very similar to those you use in writing your CSS. For the full list of all the elements in CSS and their JS counterparts check out this page.
http://www.sitestepper.be/en/css-properties-to-javascript-properties-reference-list.htm

Keep this link handy it's hard to remember all these properties off by heart. As you can see they are similar but not all are exactly the same.

```
<div id="error"> There has been an error!</div>
<div id="success"> Awesome Sauce! Things are working!</div>

 <script>
        var myNumber = 4;
```

```
        // set some CSS styles
        document.getElementById("error").style.color = "#ff0000"; // red
color for text
        document.getElementById("success").style.color = "#009900"; // green
color for text

        //check to see what the value of myNumber is and if it is more than 5
we show success
        if(myNumber > 5) {
            document.getElementById("success").style.display = "static";
            document.getElementById("error").style.display = "none";
        } else {
            document.getElementById("success").style.display = "none";
            document.getElementById("error").style.display = "static";
        }
 </script>
```

Try changing the value of “myNumber” to something below 5 and refresh your webpage.

Event & Listeners

An event in JavaScript is defined as interaction triggered by a user or function or change in data. In JS we can detect these changes and then run new functions or code that will respond to the events. Listeners as they are described listen for the event to take place. When the listener recognises an event it will then fire off the appropriate code or functions.

Events can be:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user pressed a key

There are two common manners in which we add interactive events to webpages in JS, we can apply an event function directly to an element. When it is interacted with appropriately it will run the code or create an event listener which waits for the appropriate action to occur and then will run the code.

```
<button onclick="this.innerHTML = 'I've been Clicked!'">Click on me!</button>
<button id="clickLink">Click on me!</button>
<button id="clickLinkAlternative">Click on me!</button>

<script>
    var link = document.getElementById('clickLink');

    link.addEventListener("click", function changeText(){
        link.innerHTML = 'I've been Clicked as well!';
    });

    // or like so

    var linkAlternative = document.getElementById('clickLinkAlternative');

    function changeText(){
        linkAlternative.innerHTML = 'Me too! Yay!!';
    }

    linkAlternative.addEventListener("click", changeText);
</script>
```

Here three separate sets of code are achieving the same results. It is recommended to use the third example as the standard as it provides good control over the code, thus allowing for more robust and maintainable code in future.

We are also able to remove listeners to ensure actions only get completed once. This is commonly used when transaction as done or submitted from a button. We want to control the amount of interactions the user can allocate to the specific interaction to ensure the don't do a double payment for example. Removing an event listener is the opposite of the adding, we need to tell it which element and what type of interaction to remove.

Using the previous click event listener example see how we would remove the event listener.

```
document.getElementById('clickLinkAlternative').removeEventListener('click',
changeText);
```

Once again the developers at Mozilla have put together the most comprehensive list available about the different events we can listen for. Check the link for reference - https://developer.mozilla.org/en-US/docs/Web/Events

10. Building A System With Logic in JavaScript

Let's determine our logic first. We will need to ask a number of questions before we begin to get an idea of what the this little system must do as well as the process and order it will do it in.

1. How would we describe this process in basic spoken language? (e.g. english)

I'm becoming very forgetful, and I would like a tool that will let me know if today is my birthday at the click of a button.

2. What does our simple system need to do?

It needs to do a check to see if today is my birthday and let me know, when I click on a button.

3. What are the different parts of the process?

We need to know what today's date is. We need to know when my birthday is. We need a button that will execute the function to compare today's date with my birthday to check if today is my birthday. Thereafter we need to show what the results are.

4. When the user engages with the system should something happen?

When the user clicks the "check my birthday" button, we check to see if today is the user's birthday.

5. What will be final outcome of the logic process?

That we have either confirmed or denied that today is the user's birthday.

You can see that point 1 and 2 are basically the same, however in larger more complex systems, asking similar questions from different perspectives can reveal small details that may be overlooked. It is good practise to ask many questions and to soundboard your thoughts with your team or your friend/girlfriend/yourself if working alone. Now that we know what we must build and how it must work, we can begin writing the logic (code) of the application. Let's look at the logic, we need to know today's date & my birthdate, let's define the basic variables.

```
var todaysDate;
var birthdayDate;
```

Now let's assign some static values to the date variables. We will deal with dynamic variables and dynamic data in the next course.

```
var todaysDate = "August 25";
var birthdayDate = "January 15";
```

Again let's look to the logic to see what will be next. Or logic tells us we need a button to call the function. We will now create some HTML for the basic UI elements the user will engage with. We are going to make a simple button object and a paragraph, each with an id and class as needed.

```
<button id=button>Check my Birthday</button>
<p id="results" class="results"></p>
```

As you can see the paragraph is empty. It will be the wrapper/container object that receives the message text to inform the user of what has occurred. If we again look at the logic we can see we will need a place to display the message, but also need to send a specific message based on whether it's my birthday or not. For this we will define two variables one for each message type.

```
var messageBirthday = "Hey look at that, it's my birthday!";
var messageNotBirthday = "Aww, it's not your birthday, sorry...";
```

We now have the assigned the values for the dates, messages and the basic HTML in place. Now we need to make the code react to the user and do a check to see if the dates correspond. In this case according to our dates we can anticipate that the dates will not match, thus we will be testing with "`false`" in mind. That is to say, we expect a result that is not my birthday.

```
function checkDates() {
	// conditional statement

	if (todaysDate == birthdayDate) {
		// our code will go here
	}
}
```

The basic "if" statement in place to begin the check, but once again we must look to the logic to see what must happen based on which condition is met - true or false.

According to our logic the user must receive a message let's them know whether it's my birthday or not. So we will need to show our messages we created previously to the user.

To do this we need to interact with the HTML objects we created. We need to use JavaScript to identify them and then based on what happens interact with them, by passing data to them or calling an function or both. But how do we do this?

```
document.getElementById('elementId');
```

The line code above will ask our document (webpage) to identify the object with an id of "results". Once we have identified it we can work with it. So let's identify our basic HTML objects and store them in variables for later.

```
var showResults = document.getElementById('results');
var checkMyBirthday = document.getElementById("button");
```

Now we need need to put the correct message in our results container based on the condition. We will need to adjust our "if" statement to become an "if else" statement to switch between the messages.

```
function checkDates() {
	// conditional statement

	if (todaysDate == birthdayDate) {
		// first action here
	} else {
		// second action here
	}
}
```

We have adjusted our "`if`" statement, declared all the variables we need, now we need to complete our function.

```
function checkDates() {
	// conditional statement

	if (todaysDate == birthdayDate) {
```

```
            // if the dates are the same put messageBirthday into our
      results container.
      showResults.innerHTML = messageBirthday;
      } else {
            // if the dates are not the same put messageNotBirthday into our
      results container.
            showResults.innerHTML = messageBirthday;
      }
}
```

Important!

"`.innerHTML = x;`" is a javscript method which allows us to change html content inside an html tag.

We have completed the "`checkDates`" function it will run and compare the values of the variables we defined as our dates to see if they correspond and then send the message which is most appropriate to the results container. However our logic says that this must happen on a button click. We have already defined our button in a variable. We need only link the functions to the button.

```
checkMyBirthday.addEventListener('click', function () {
      checkDates();
}, false);
```

At this point we have created a click function using an "`addEventListener`", the event listener will wait till a specific event occurs before it is run. In this case it waits for the click. Inside the click function we execute our "`checkDates`" function we already created. The whole code should look like the following.

```
<script type="text/javascript">
      var todaysDate = "August 25";
      var myBirthday = "August 25";
      var showResults = document.getElementById('results');
      var checkMyBirthday = document.getElementById("button");
      var messageBirthday = "Hey look at that, it's my birthday!";
      var messageNotBirthday = "Aww, it's not your birthday, sorry...";

      function checkDates() {
```

```
            if (todaysDate == myBirthday) {
                  showResults.innerHTML = messageBirthday;
            } else {
                  showResults.innerHTML = messageNotBirthday;
            }
      }

      checkMyBirthday.addEventListener('click', function () {
            checkDates();
      }, false);
</script>
```

And the HTML like this:

```
<button id="button">Check My Birthday</button>
<p id="results" class="results"></p>
```

Using only basic parts of JavaScript, we have built a basic system using some logic to guide us.

11. Setting Up JQuery

JQuery is a cross platform Javascript library designed to simplify the client side scripting of HTML.

What does that mean?

The jQuery library assists developers in writing JavaScript code, it contains many common functions and methods used in JavaScript development. This provides JavaScript developers and designers a toolkit to quickly and easily get common functionality running on a system or website.

The jQuery library makes the following common development tasks simpler:

- HTML/DOM Manipulation
- CSS Manipulation
- HTML Event Methods
- Effects and Animation
- AJAX
- Utilities

We must include JQuery on our website or system in order to access the predefined functions and methods that will make our development work simpler. Including a JavaScript library on a simple website is very easy. First we need to download the JQuery library and store it in a folder in our website. We must then create a link to the library in the appropriate HTML code, so that the site or system can identify the library. Alternatively we can create a link that points to a CDN (Content Distribution Network). A CDN is merely a server that stores common libraries and files that can be openly accessed by websites. Once again if we apply the appropriate HTML code to link the file, we can include the library in our site. To link a script file we will need to add the following HTML code to our website. This can be either added to `<head>` section of our website or before the closing `</body>` tag of our website.

```
<!DOCTYPE html>
<head>
      <!-- local file -->
      <script src="javascriptfolder/jquery.js"></script>
      <!-- CDN hosted file -->
      <script
      src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"
      ></script>

</head>
<body>
      <!-- local file -->
      <script src="javascriptfolder/jquery.js"></script>
      <!-- CDN hosted file -->
      <script
      src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"
      ></script>
</body>
</html>
```

12. JQuery Syntax

JQuery syntax is fairly similar to JavaScript syntax seeing as it is a library of JavaScript, however there are a couple noticeable differences. JQuery requires the use of "$" symbol, to indicate to the website that we want to use the JQuery library functions and methods. This ensures that the website will know which file to look in for the appropriate code we are calling.

We can write jQuery on an HTML page, much like we write embedded CSS. We need to use a <script> tag for this process and write any jQuery with in that <script> tag.

```
<script>
	$(selector).method();
</script>
```

Anything in "$(brackets)" is known as the selector. The selector queries or finds HTML elements, like classes, id's and tags, the.".action();" lastly we have what is known as the jQuery method or action - in other words, what should be performed on the element selected.

It should be noted that if you are targeting a <p> or a css .class they must be written with "", the angle brackets must be removed from the <p> and the css class must be written as if you had written it in a stylesheet.

Selecting a HTML elements by Tag name

```
$("p").method();
```

Selecting a HTML elements by Class name

```
$(".container").method();
```

Selecting a HTML elements by ID name

```
$("#myId").method();
```

The only time you don't need quotes around a selector is when the selector is part of the jQuery function, like a variable. Its also worth noting that you can add your jQuery in a separate file. Rather than embedded in your HTML file - create a functions.js file in a folder named js in the root of your website and link it to your document. *(Please note, the name of the can be anything you feel is appropriate to call it i.e. menu-actions.js or menu-functions.js, etc.)*

```
<script src="js/functions.js"></script>
```

Document Ready Function

The document ready function is a special function in jQuery that holds on to executing the jQuery code until the entire DOM is loaded.

```
$(document).ready(function(){
	//my code goes here
```

```
});
```

The Document ready function prevents any jQuery from running before the entire document is finished loading or is “ready”. It is considered good practice to do this and you can run as many functions as you like with in this code block - the code block is anything or rather the code that gets executed when jQuery calls the functions and is always contained within the parentheses.

A shorthand for the document.ready function can be written as follows.

```
$(function(){
      //my code goes here
 });
```

jQuery Selectors

Examine the following block of code.

```
$(document).ready(function(){
      $(“button”).click(function(){
            $(“p”).hide();
      });
});
```

Breaking it down line for line we can decipher exactly what the code block is attempting to do.

```
$(document).ready(function(){
```

The first line as we know, tells jQuery to load only when the rest of DOM (HTML and CSS) has loaded.

```
$(“button”).click(function(){
```

The second line looks quite similar to the first except it exists within the Document Ready Function. When the document is ready, begin jQuery, then find all HTML elements called <button>, when that button is clicked we would like to execute a function that will find all <p> tags and hide them. Close instruction, close instruction” We can use simple syntax that will be pretty familiar to you with CSS.

```
$(“button”).click(function(){
```

Is looking for all the HTML <button> elements.

```
$(".sidebar").click(function(){
```

Is looking for an element with the class of "sidebar".

```
$("#tweet").click(function(){
```

Is looking for an element with the ID of "tweet". There are a variety of Query selectors that can be used.

- $("*") - finds all elements like the universal selector in CSS.
- $(this) - will selects the current HTML element interacted with, either by the code or by the user.

```
$(document).ready(function(){
	$("button").click(function(){
		$("this").hide();
	});
});
```

- $("p.intro") - Will select any paragraph with the class of "intro".
- $("[href]") - this will select all elements with an href attribute.

jQuery Events

An event is used when a visitor can respond to action, like a click or double click, or maybe a key down action performed by the user, even scrolling can be identified and thus call a function when the event of scrolling takes place. One of the most basic events is the "ready" event.

```
$(document).ready(function(){
	// my code goes here
});
```

Once the document is "ready" or rather done loading, we can call a function inside the "ready" event. Apart from the "ready" event we can ask JQuery to listen for events that occur from the user's interactions. Clicking on a link for example.

```
$("a").click();
```

This means that we can do something when the user clicks on any HTML <a> tag element. Once we have identified an event has taken place, we can add functions to event to be executed when the event occurs. The code below shows a "document ready" event, which will listen for a user's click event. If the user clicks a HTML <a> tag element, we execute code which will make the something occur on the website or in the system.

```
$(document).ready(function(){
	$("a").click(function(){
		//something needs to happen!
	});
});
```

Below is added code to be executed when the user clicks on a HTML <a> tag element. The code below will cause the link that is clicked to be hidden using the JQuery ".hide()" method.

```
$(document).ready(function(){
	$("a").click(function(){
		$(this).hide();
	});
});
```

JQuery Effects

They are simple effects that can manipulate DOM elements. Commonly used effects are:

- hide
- show
- toggle
- slide
- fade
- animate

Let us take a look at this example - you would right it with an event within a document ready function, I will write it by itself and then with all the code necessary to get it to work:

$("p").hide();

Will hide all p's - but if we had that inside of our document ready function as soon as the page loaded, all the <p> HTML tags would disappear. We could instead have this function occur in a user event like a click function.

```
$("button.goaway").click(function(){
        $("p").hide();
});
```

When a user clicks on the button with the class of "`goaway`", all <p> HTML elements must disappear. With effects or methods we can also have what are known as "`parameters`" These are simple additions to effects that provide more detailed information.

```
$(button.goaway).click(function(){
      $("p").hide("slow")
      // or we could have $("p").hide(1000);
});
```

The speed parameter in this case can use a keyword - either slow or fast, or you can give it a number in milliseconds - in this case 1 second. There is another very useful method - known as "`.toggle()`" - switch or toggle between states, like on or off. Toggle will hide/show and element depending on the event.

The code below will toggle the visibility of the all the HTML "<p>" tag elements on the page.'

```
$(document).ready(function(){
      $("button.toggle").click(function(){
            $("p").toggle();
      });
}):
```

Fade

The fade method is just another form of animation that will fade an element. There are 4 fade methods:

```
fadeIn();
fadeOut();
fadeToggle();
```

```
fadeTo();
$("button").click(function(){
	$("div").fadeIn();
});
```

The fade method also supports the speed parameter. You can try these examples out to get more familiar with them.

Slide

The slide method is just another form of animation that will slide an element either up or down or you can toggle it. There are 3 fade methods:

```
fadeDown();
fadeUp();
fadeToggle();

$("button").click(function(){
	$("div").slideDown();
});
```

The slide method also supports the speed parameter. You can try these examples out to get more familiar with them.

Animate

You can use the animate() method for creating custom animations

```
$("button").click(function(){
	$("div").animate({left: '250px'}, 1000);
});
```

The code states that a click event on any of the HTML "<button>" tag elements any of the HTML " " tag element on the page will animate to the left by 250px - the parameters must go in the parentheses and you can add speed parameter i.e. slow, fast or milliseconds (500ms).

```
$("button").click(function(){
	$("div").animate({
		left: '250px',
		opacity: '0.5',
		height: '150px',
```

```
            width: '150px'
      });
});
```

Note, that the values are in single quotes and are separated by commas. Please also note that if for instance you wanted to animate "`margin-left`", you need to camelCase so "`margin-left`" would become "`marginLeft`".

.Stop() Method

The stop method is used to stop an animation before it is finished. You can read up about it on your own especially if you plan on writing jQuery animation plugins.

Callbacks

Callbacks are functions that can be executed within other functions to instantiate the next appropriate functionality in code at the appropriate time. Callback methods can be used as a parameter in functions.

```
$(document).ready(function(){
      $(".button").click(function(){
            $("p").hide("slow",function(){
                  alert("The paragraph was hidden");
            });
      });
});
```

This means that everything will be executed line by line and once it's done, the browser will alert the user that it was successful by displaying the message "The paragraph was hidden".

Chaining

Up until now we've been writing jQuery commands, statement by statement. with chaining we can use the dot-syntax to chain multiple methods together. The "." simply keeps the string together.

```
$("#p1").css("color","red").slideUp(2000).slideDown(2000);
```

JQuery is not a strict-typed language - because it is essentially JavaScript - thus you can employ linebreaks to make it more readable, the above line of code can be written like as below.

```
$("#p1").css("color","red")
      .slideUp(2000)
      .slideDown(2000);
```

13. Manipulating the DOM with JQuery

There are few methods just like .click or .mouseover or .hide that jQuery uses to manipulate specific elements in the DOM. The first component is Get Content. There are 3 jQuery methods available for this:

- text() - can set or return textual content of a selected element
- html() - can set or return the content of a selected element (including HTML markup)
- val() - sets or returns the value of form fields.

```
$("button").click(function(){
      alert("Text: " + $("#p1").text());
      // alert is a javascript function for making a browser popup window
});
```

The code may look complicated but it's quite simple. When a user clicks on any button the browser will alert the user, it will prepend the text "Text:" and append whatever text is located in the element which has an ID of "`p1`". If we use the same code but replace "`text()`" with "`html()`" - it will return the actual html markup within the element "`p1`".

```
$("button").click(function(){
      alert("HTML: " + $("#p1").html());
});
```

Finally, we can use the "`val()`" on a similar piece of code to find out what the value of an input field might be. In this example we assume that there is an input field with the ID of "`test`".

```
$("button").click(function(){
      alert("Value: " + $("#p1").val());
});
```

We can grab attribute values from elements in DOM quite easily using the following method.

```
$("button").click(function(){
```

```
	alert($("a#contact").attr("href");
});
```

This code will pull the value of the href attribute from a HTML <a> tag with the ID of "`contact`". These "`Get Content`" methods can be useful, but aren't very functional, ideally we need to be able to change the content within the selected DOM element. We use "`SET Content`" methods to do this.

Setting Contents

To set the contents of DOM objects or HTML elements. The same methods will apply however we work with them differently.

- text()
- html()
- val()

```
$("button").click(function(){
	alert("HTML: " + $("#p1").html());
});
```

This is the jQuery code to "`Get Content`". Find all the HTML <button> elements, when a user clicks, inform the browser to alert the user with the following content - a string with value "`HTML:`" and then to identify the HTML markup in an element with the ID of "`p1`".

To "`Set Content`" of the text in an HTML element, we can do the following:

```
$("button").click(function(){
	$(".intro").text("Welcome to my Campaign");
});
```

Or set the HTML markup inside another HTML element, we can use the "`.html()`" method.

```
$("button").click(function(){
	$(".intro").html("<p>Welcome to my Campaign</p>");
});
```

Below we are setting the value attribute of an HTML form field element.

```
$("button").click(function(){
```

```
	$(".intro").val("Batman");
});
```

However we can also set other attribute values using the "`.attr()`" method:

```
$("button").click(function(){
	$(".intro").attr("href", "http://www.google.com");
});
```

As you can see we've added two parameters, the attribute name and the value. Finally, you can set multiple attributes, but you must include them in the parentheses.

```
$("button").click(function(){
	$(".intro").attr({
		"href" : "http://www.google.com",
		"title" : "Google"
	});
});
```

Add Content

There are four basic methods available for adding HTML content to a DOM element.

- append() - Will place content at the end of selected elements
- prepend() - Will place content at the beginning of a selected element
- after() - Inserts content after the selected element
- before() - inserts content before the selected element

```
$("div").append("<p>Hello, World</p>");
```

We can also remove or empty HTML elements with the DOM, using the following methods.

- remove() - removes the selected element and its child/nested elements
- empty() - removes the child elements from the selected element.

$("div").remove();

Would remove that "`div`" element out of the DOM.

```
$("div").empty();
```

Would remove everything within that "`div`" element. You can go a step further and target specific elements within a selector. This works as a parameter with in the remove method.

```
$("div").remove("p");
```

CSS Classes

As much as jQuery can manipulate HTML elements, it can also manipulate CSS. It contains 4 methods:

- addClass() - Can add one or multiple classes to the selected elements
- removeClass() - Removes one or more classes from the selected elements
- toggleClass() - Toggles between adding/removing classes from selected elements
- css() - sets or returns the style attribute

Just like any other jQuery method we can use this with a function, or just by itself. Consider the following:

```
$("button").click(function() {
      $("h1,h2,h3").addClass("red");
});
```

```
$("button").click(function() {
      $("h1,h2,h3").addClass("red", "heading");
});
```

This will append two classes to all the "`h1,h2,h3`". You can even have multiple elements being assigned classes.

```
$("button").click(function(){
      $("h1,h2,h3").addClass("red", "heading");
      $("p").addClass("pink");
});
```

As would imagine, the removeClass() and toggleClass() work much the same:

```
$("button").click(function(){
```

```
	$("h1,h2,h3").removeClass("red", "heading");
});

$("button").click(function(){
	$("h1,h2,h3").toggleClass("heading");
});
```

The toggle class will remove or add the class "heading" on click. The CSS method can be used to get and set css properties, we will focus on set as its the most useful and appropriate for now.

```
$("p").css("background-color", "#000000");
```

This will find all <p> tags and apply a background colour of black. Remember that this won't be executed or written in your stylesheet. It will be added to the HTML element as an inline style attribute as shown below.

```
<p style="background-color: #000000;"></p>
```

14. Traversing the DOM with JQuery

With JQuery we can move through different HTML elements using the traversing methods, as we know HTML is used for structure, this structure has a hierarchy and is systematically in relation to other elements. Just like in JavaScript we can identify the child or parent (descendents or ancestors) of HTML elements from elements we are already know exist.

- Ancestor: is a parent, grandparent or great-grandparent
- Descendant: is a child, grandchild or great-grandchild
- Siblings: share the same parent.

An example of a basic HTML with an unordered list and two list items in the unordered list.

```
<div class="parent">
	<ul class="child">
		<li class="granchild"><span>Hello</span></li>
		<li class="granchild"><strong>Goodbye</strong></li>
	</ul>
</div>
```

The element is the parent of the <ul> element, and an ancestor of all of it's children. The <ul> is the parent of both the <li>, but a child of the . The <li> is the parent of the element, but child of the <ul> and descendant of the . Both the <li> are siblings.

Ancestors

We can traverse parents using:

- parent()
- parents()
- parentsUntil()

The parent method will rerun the direct parent of the selected element. It goes up by one level.

```
<body>
	<section class="container">
		<div>
			<p>Hello</p>
		</div>
	</section>
	<script>
		$(document).ready(function(){
			$("p").parent();
		});
	</script>
</body>
```

The "`parents()`" method will find its way all the way up the DOM tree to the last ancestor or root element which is the <html> element.

```
$(document).ready(function(){
	$("p").parents();
});
```

Finally, the "`parentsUntil()`" method will rerun all ancestor elements between two arguments or parameters.

```
$(document).ready(function(){
	$("p").parents("body");
```

```
});
```

This will return the div and section elements as parents.

Descendants

Just as we can traverse up, we can also traverse down. We can use the "`children()`" and "`find()`" methods . The "`children()`" method returns all direct children of the selected element. The "`parent()`" method it is only capable of traversing down by one level.

```
$(document).ready(function(){
	$("section").children();
});
```

You can use the "`find()`" method to find all descendants to the ver last.

```
$(document).ready(function(){
	$("section").find("*");
});
```

Use the Asterisk (*) operator to find all. Or we could find specific children

```
$(document).ready(function(){
	$("section").find("div");
});
```

Will return every tag that is a child of the <section>.

Siblings

As you would have guessed, we can go up and down and even sideways through the DOM, in this case we are talking about siblings. I will go through the most common methods here as there are quite a few:

The "`siblings()`" method will return all sibling elements of the <p>.

```
$(document).ready(function(){
	$("p").siblings();
```

```
});
```

The most useful is the next() method:

```
$(document).ready(function(){
	$("h2").next();
});
```

Will find the next sibling element of the <h2>. As you expect so will "`prev()`". There are also "`nextAll()`" method, "`prevAll()`", "`nextUntil()`" and "`prevUntil()`", all work off the same concepts we have already discussed.

14. JQuery Extras, AJAX and Goodies

No Conflict

Depending on your project, you might find that there is a problem with other Javascript libraries using the $ symbol. If you have two libraries both using $ symbols for functions or pro grammatical dynamics then certain scripts could stop working. jQuery has seen this coming. Instead of using the $ one can actually just use the word jQuery instead.

```
jQuery(document).ready(function(){
	jQuery("h2").next();
});
```

You can create your own alternative for the $ sign and store it inside a variable.

```
var dm = $.noConflict();

dm(document).ready(function(){
	dm("button").click(function(){
		dm("p").hide(1000);
	});
});
```

AJAX

AJAX stands for Asynchronous Javascript and XML. AJAX allows the loading data in the background and displaying it on the webpage without reloading the entire page or refreshing it. It's quite useful and you see it every single day. Gmail uses it, so does Facebook with your News Feed, Youtube,

Portfolio sites, blogs. We can do AJAX calls in both JavaScript and JQuery, each is a different method but the end results are the same.

JavaScript AJAX - https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started
JQuery AJAX - http://api.jquery.com/jquery.ajax/

These two links above will get you on the right track using AJAX. Happy Coding!

Goodies

There are hundreds, if not thousands of JavaScript and jQuery plugins/libraries available to aid your development. Below is a list of some popular libraries:

1. Typeahead.js - http://twitter.github.io/typeahead.js/
2. Windows.js - http://nick-jonas.github.io/windows/
3. ScrollPath.js - http://joelb.me/scrollpath/
4. ArcText.js - http://tympanus.net/codrops/2012/01/24/arctext-js-curving-text-with-css3-andjquery/
5. Lettering.js - http://letteringjs.com/
6. Gridster.js - http://gridster.net/
7. CountDownTimer.js - https://www.npmjs.com/package/countdowntimer
8. SocialShare.js - https://github.com/ritz078/socialShare.js
9. AnimatedModal.js - http://joaopereirawd.github.io/animatedModal.js/
10. Sortable.js - https://github.com/RubaXa/Sortable
11. jQueryTween.js - http://thednp.github.io/jQueryTween/
12. Lightcase.js - http://cornel.bopp-art.com/lightcase/
13. Vivus.js - http://maxwellito.github.io/vivus/
14. Wysiwyg.js - http://wysiwygjs.github.io/
15. 3D Responsive Scroll - https://github.com/nickavignone/responsive_3dfoldscroll

14. References

W3schools.com, (2016). JavaScript Tutorial. [online] Available at: http://www.w3schools.com/js/default.asp [Accessed 15 Feb. 2016].

jquery.org, j. (2016). jQuery. [online] Jquery.com. Available at: http://jquery.com/ [Accessed 15 Feb. 2016].

Mozilla Developer Network, (2016). JavaScript Tutorial. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference [Accessed 20 Feb. 2016].

McFarland, David Sawyer, and David Sawyer McFarland. Javascript & Jquery. Sebastopol, Calif.: O'Reilly, 2011. Print.

Pehlivanian, Ara, and Don Nguyen. Jump Start Javascript. Collingwood, Vic. SitePoint Pty. Ltd., 2013. Print.