THE JAVASCRIPT MATH OBJECT

JavaScript provides a Math object for us that has many properties and methods to help us with mathematical functions and equations including: pi, square root, & trigonometry functions. You can find a comprehensive list at http://www.w3schools.com/jsref/jsref_obj_math.asp.
It is important to note that these cannot be assigned to a variable and Math. (note the capital M!) must be used.
The most useful functions for 90% of what we'll ever do are:

Math.round()

This function will round a number to the nearest integer.
`Math.round(2.5);` This will return 3.

Math.floor()

This function will always round a number down to the nearest integer.
`Math.floor(2.999);` This will return 2, even though it is almost 3.

Math.ceil()

The counterpart to floor(), ceil is short for ceiling and will always round a number to the next highest integer.
`Math.ceil(2.001);` This will return 3, even though it is barely more than 2.

Math.random()

This function will generate and return a random decimal number between 0 and 1 (not counting 1).
`Math.random();` This could return 0.791426395997405.

To create a random whole number generator, we need to do a little work:

```
newNum = Math.floor(Math.random() * 49) + 1;
```

Here's how it works…
Math.random() gives us our random number between 0 and 1:
0.791426395997405

multiplying by 49 gives us a decimal number between 0 and 48:
38.77978340387285
enclosing it in **Math.floor()** gets rid of the fraction: **38**
adding one gives us a number between 1 and 49 – otherwise the range will be 0 to 48.

THE JAVASCRIPT DATE OBJECT

To work with dates dynamically and logically, JavaScript provides a Date object for us that has many methods to help us display dates and use them in calculations. You can find a comprehensive list at:
http://www.w3schools.com/jsref/jsref_obj_date.asp.
Note that the capital D must be used when calling the Date object.

To create a new date:

```
var now = new Date();
```

This gets the time from your computer the moment it was called. If you output the value, you will see:

```
Thu Oct 17 2017 14:00:20 GMT-0400 (Eastern Daylight Time).
```

Fortunately, JavaScript has several methods to help us deal with this unwieldy info blob in simpler formats.

.toDateString()

This method returns just the date part of the Date object as a string.
`now.toDateString()` returns `Thu Oct 17 2017`

.toTimeString()

This method returns the second half of the Date object as a string.
`now.toTimeString()` returns `14:00:20 GMT-0400 (Eastern Daylight Time).`

.getDate()

This function will return the day of the month (1-31) for the specified date.
`now.getDate()` will return `17`.

.getFullYear()

This function returns the four digit year for the specified date. Note: getYear() is no longer used and should be avoided. It returns a three digit value. Yeah , I know.

`now.getFullYear()` This will return 2016.

.getMonth()

This function will return the month as an integer (0-11) for the specified date. Note that, since the range starts at 0, January = 0, and December = 11.

`now.getMonth()` will return `9`.

.getDay()

This function will return the day of the week as an integer (0-6) for the specified date. Note that, since the range starts at 0, Sunday = 0, and Saturday = 6.

`now.getDay()` will return `4`.

Similarly, we can access time details with self-explanatory methods:
getHours() getMinutes() getSeconds()

One useful method that requires some explanation is **getTime()**.
This returns the number of milliseconds since midnight on January 1st, 1970. I will spare you the academic details and simply say that this lets us use dates in mathematical functions. For example, if you want to know how many days from today until your next birthday, you need to subtract today from your next birthday. Rather than dealing with years and how many days in a particular month, getTime() lets you subtract one integer from another.

To give you an idea of the kind of numbers we're looking at, on September 6, 2016 at 9am, the number of milliseconds since January 1st, 1970 was 1473166800000. This is a larger number than we are used to seeing, but it makes handling dates a matter of simple addition, subtraction, multiplication and division. There are 1000 milliseconds in 1 second, and 86400000 milliseconds in one 24 hour day. To get the number of days until your next birthday, subtract today from your next birthday date and divide by 86400000 - don't forget to use Math.floor() to get rid of the decimals! To convert dates, I use:
http://www.onlineconversion.com/unix_time.htm , or
http://www.epochconverter.com/ which will convert in both directions.

IMPORTANT SOME CONVERTERS WILL BE THE NUMBER OF SECONDS, NOT MILLISECONDS, SO YOU'LL HAVE TO MULTIPLY OR DIVIDE BY 1000 TO WORK WITH THE JAVASCRIPT TIME METHODS**
`now.getTime()` returns `1382032820000`.

Setting Dates & Times

JavaScript also allows us to set the Date object to whatever date/time we want, and supplies many methods and ways to do so. Starting with the raw Date object itself:

```
var newYear = new Date(2014, 0, 01);
var newYear2 = new Date("January 1, 2014 00:00:00");
```

Mirroring the "get" methods, you can set individual time objects with specific self-explanatory methods:
setDate() setMonth() setHours() setSeconds() setTime() et cetera…

Additionally, we can use the objects with simple math:
`myDateObject.setDate(myDateObject.getDate() + 7);` sets the date one week (7 days) from the date variable.

TIMING EVENTS

There are two very useful methods in JavaScript that let us time events in our code. One is a timer that waits a set amount of time before executing a function; and the other will execute a function over and over waiting a set amount of time between executions.

setTimeout() calls a function after waiting a set amount of time. It takes two parameters: the function (or name of the function) to run after the time has expired; and the number of milliseconds to wait before executing the code (Remember: 1 second = 1000 milliseconds). Unlike other methods, this method needs to be set to a variable to start the clock.

```
var myGreetingTimer = setTimeout(helloAlert, 5000);
```

Declaring this variable starts a timer that will call a function named '**helloAlert'** after 5 seconds.

In case you need to cancel the timer, you can call **`clearTimeout(name of setTimeout variable)`** to do so.

setInterval() lets you set an amount of time, and continues to call a specified function each time the set time elapses. The parameters and function options are the same as setTimeout(). This will run forever (or until the window is closed) unless you stop it using **`clearInterval(name of setInterval variable)`**.

```
var showSec = setInterval(function(){console.log("2 seconds have gone by…");}, 2000);
```

Declaring this variable starts a timer that will output "2 seconds have gone by..." to the JS console every 2 seconds.

Note in the above that, instead of providing the name of the function as the first parameter, I have literally coded in a function as the first parameter (see the comma before the 2000?). This is referred to as an 'anonymous' function because I did not give it a name. Generally, with timers it is better and safer to provide a function name as the first parameter, however, I thought I would take this opportunity to give you a glimpse ahead because they are used frequently in some of the libraries and frameworks that we will be looking at in the future.