Modern Javascript 2020 Notes

Table of Contents

Module 1: Getting started with Javascript 2
Module 2: Variables, data types, type conversion and more 3
Module 3: Operators 14
Module 4: Strings 24
Module 5: Numbers 33
Module 6: Conditional statements 45
Module 7: Arrays 55
Module 8: Date, Date methods & Math object 59
Module 9: Functions 61
Module 10: Javascript objects (basics) 77
Module 11: Javascript Document Object Model (DOM) 79
Module 12: Javascript Regular Expressions Basics 87

Module 1: Getting started with Javascript

Setting up the environment to start writing Javascript code

Code to connect external scripts:

<script type="text/javascript" src="script.js"></script>

Type the below code in your script.js file.

Console code:

Log outputs into your console.

console.log("hello");

Comments:

Comments on the code – won't be read by the browser

// - single line comment

/* */ - multi-line comment

Getting output via alert statements:

alert("Statement");

Hello world program:

alert("Hello World");

Module 2: Variables, data types, type conversion and more

Use strict

This was an ES5 specification that came out in 2009. Enables the latest modifications, basically, activates modern Javascript.

Use at the top of the code.

'use strict';

Or

"use strict";

Variables

Variables are containers of information. They store certain values, be it numbers, characters, strings, etc.

To create or declare a variable in Javascript, you should use the keyword let, like this.

let variableName;

There are some rules to what the variable name could be.

1. The first character of the variable name should be an alphabet or an underscore or dollar sign ($)
2. The rest of the characters can be alphabets, underscore or numbers or dollar signs ($)
3. Variables are case sensitive. So, apple is not equal to Apple.
4. Reserved words, or keywords, cannot be used. For example, var is a reserved word, so it can’t be a variable name.

So, taking all these rules into account,

let _apple;

let app_le12;

let $apple;

let apple;

Are all correct

But,

let 12apple;

Is not correct, because numbers should not be the first character of a variable.

Reserved words

Words that have special meaning in Javascript – can't use in variable declaration.

There is a list:

Let, var, const, break, case, catch, class, continue, debugger, default, delete, do, else, export, extends, finally, for, function, if, import, in, instanceof, new, return, super, switch, this, throw, try, typeof, void, while, with, yield

Var

Before ES2015, that is, in ES5 and before that, variables were declared using the keyword var.

It's still in use, and it works, but it is not considered good practice to use var to declare variables anymore because it has a function scope and not a block scope like "let" does.

var a = 5;

Constants

Unchanging variables declared using const.

const pi = 3.14159;

If we reassign the value of pi now:

pi = 4;

We'll get an error in the console.

Assigning a value & Data types

So, a variable can hold different types values

String – which is a string of characters

let fruit = "Apple";

Here, we've created a variable named fruit and assigned it a string value Apple.

Then, for numbers,

let num = 5;

Or

let num1 = 5.5;

So, it can be whole numbers or decimal point numbers.

Finally, Boolean. That is, true or false. You can assign true or false values to a variable.

Like,

let isHere = true;

Or

let isHere = false;

We've used camel casing above.

This type of naming variables is called Camel casing, where the first word is all small letters and after that, every word has the first letter capitalized. This is the naming convention for pretty much every big programming language, so it's better to follow it for variables with multiple words.

Dynamic variables

Can reassign different types to the same variable.

For example,

let a = 12;

Now, a is a number.

a = "apple";

I've made a a string now.

a = false;

Now, it's a Boolean value. This is how you can change the type of your variable any way you want.

Infinity & NaN

There's infinity and NaN (not a number) as well.

let a = 1/0;

console.log(a);

Result is infinity.

It's sticky.

let a = 1/0 + 1;

Still infinity.

let a = "apple" / 1;

NaN is the result.

Sticky too.

let a = "apple" / 1 + 1;

Still NaN.

More on strings

Double quotes, single quotes, backtick.

let a = "apple";

let a = 'apple';

console.log("This is an " + a);

With backtick, you don't need the + operator. You can do everything within backticks. I'll show you.

console.log(`This is an ${a}`);

Can design sentences too.

For example, if we need a newline with double or single quotes, this is how you'll do it.

console.log("Hello!\nGood morning");

With backtick, its very simple:

console.log(`Hello!

Good morning!`);

Can do operations with backticks too:

console.log(`1+2=${1+2}`);

Can do like this by adding variables too:

let a = 1, b=2;

console.log(`1+2=${a+b}`);

Boolean

let bool = "true";

console.log(bool);

let bool = "false";

let bool = 1>2; //false

Complex data types

There are more complex data types as well. Arrays and objects. We'll be looking at them in detail in future lessons, but I'll just give a brief description now.

let a = [1, 2, 'hello', 4];

Arrays are written within square brackets, and they are used to hold more than one value at a time, unlike a normal variable. They can hold values of different data types.

Objects are similar to arrays, but they are used to hold properties and values of a single "object". They're written with curly braces.

let a = {name:"Mary", age:18, hair:"blonde"};

The above is an object and the name is Mary, age is 18 and hair color is blonde. You can access the values in each of the properties.

Typeof operator

The typeof operator is used to find the type of the variable or value.

This is how it's written:

let a = 5;

typeof a; //number

typeof 5.5; //number

typeof 0; //number

typeof (5 + 6); //number

typeof ""; //string

typeof "hello"; //string

Undefined

When a variable is just declared and it hasn't been assigned a value, it has no value. When you check its type, it'll say undefined, because the browser does not know what the type of the yet to be declared value is.

let a;

typeof a; //undefined

a = 5;

typeof a; //number

You can also set the value of a variable to undefined to empty it of its value. You might want to fill it with something else later on.

So,

a = undefined;

typeof a; //undefined

But, an empty string is still recognized as a string. It won't be called empty or undefined.

typeof "";

There is also the null data type. It basically means nothing. That is, it doesn't exist.

But the data type of null is actually object, and not null.

But, it can be used to make an object null.

let a = {name:"Mary", age:18, hair:"blonde"};

a = null;

typeof a; //object

But, you can set the value to undefined, so the type is also undefined.

a = undefined;

alert(typeof a;) //undefined

So, the difference between undefined and null is the type, even though both make the variable devoid of its values:

typeof undefined; //undefined

typeof null; //object

null == undefined; //true because they do have the same value, that is, nothing

null === undefined; //false because one is of type object and the other is of type undefined

Type conversion:

Can convert values from one type to another by using pre-defined type conversion functions of javascript.

String conversion:

let a = 1;

console.log(typeof a);

Now, let's perform conversion from number to string.

a = String(a);

console.log(typeof a);

This works for Boolean too.

Number conversion:

Now, let's convert to number.

```
let a = "1";
console.log(typeof a);
a = Number(a);
console.log(typeof a);
```

Spaces will be removed and the rest will be converted to a number.

```
let a = "   1  ";
a = Number(a);
console.log(a);
let a = "   12  "; //works
let a = "   1  2  "; //will give NaN because there's space in between.
```

Long strings that aren't just one number won't work either.

```
let a = "Hello"; //This is NaN too
```

Can do numeric conversion on other things than strings too.

```
let a = true;
console.log(Number(a)); //1
let a = false; //0
let a = undefined; //NaN
let a = null; //0
```

let a = ""; //Empty string gives 0 too

Because we need something that can be converted to a number.

Boolean conversion:

let a = 1;

console.log(Boolean(a));

let a = 0; //false

let a = "Hello!"; //true

let a = ""; //false

let a = "0"; //This is true too. Basically any string with anything inside it is true.

let a = -1; //except 0, any other number is true, even negative numbers

Getting outputs

Alert boxes

Alert boxes are the easiest ways of getting outputs.

alert("Hello there!");

Can output values in variables too.

let a = "Hello there!";

alert(a);

Combine multiple strings:

let a = "Hello there!";

alert(a + " Have a nice day!");

Can do calculations within alerts too.

```
alert("1 + 2 = " + 3);
```

Prompt boxes

Can prompt for values with the pre-defined function prompt in Javascript.

```
let age = prompt("What is your age?", 15);
```

The second parameter is optional and displays the default value if the user does not input anything.

```
alert("Your age is " + age);
```

Can write like this too:

```
alert(`Your age is ${age}`);
```

Confirm boxes

Confirm can be used to get yes or no answers, and if assigned to something, it will apply true or false Boolean values to it.

```
let age = confirm("Are you less than 5 years old?");

if(age == true) {
        alert("You're a baby");
}
else {
        alert("You've grown up!")
}
```

Can do calculations and other things based on that.

Module 3: Operators

Assignment operator:

Assign values to variables with the assignment operator (Equal to '=').

let a;

a = 2;

let a = 2;

Different from equal to operator (==, ===)

let a = 10, b = 5;

a = a + b;

Multiple variable declarations & assignment:

let a, b, c;

Or

let a = 1, b = 2, c = 3;

Undefined:

let car;

Right now, the value of car is undefined because it has no value assigned to it (yet).

let a = 2;

a = 3;

Now a has a value of 3. Because previous value of a was replaced by the latest value. Only latest value counts.

Basic calculations and outputs

So, there are some basic operators in JavaScript, the ones you've learned in your Math class.

+ - addition

--Subtraction

*-Multiplication

/ - Division

% - Modulus

** - Exponentiation

Modulus might be unfamiliar for you. It's very simple. % gives the reminder of the division of the 2 operands.

For example, 3%2 = 1, because the quotient is one and the remainder is 1, so it'll output the remainder.

Exponentiation is used to find the exponent of the 2 numbers.

2 ** 2 = 4

2 ** 3 = 2 * 2 * 2 = 8

Can create square root with fractions:

4 ** (1/2) = 2 – should be in brackets

Complicated operations

Similarly, there are some more complicated operations as well,

+=, -=, *=, /=, %=

a+=5

This basically means, a = a+5

Similarly, there is

a-=5

a*=5

a/=5

a%=5

Increment & decrement operators:

Used to increment or decrement values by 1.

Increment operators (2 types) : post and pre increment operators

let a = 5;

let b = a++;

If you check the values of a and b, a will be 6 and b will be 5.

Post increment – assigns original value of a to b first and then increments a.

let a = 5;

let b = ++a;

Both b and a will be 6.

Pre increment – increments a first, and then assigns the new value to b.

Similarly:

Decrement operator (post and pre)

let b = a--;

let b = --a;

String concatenation:

+ operator, wen used with strings, concatenates them, not calculates.

let a = "hello " + "world";

alert(a);

+= can also be used for string concatenation:

let a = "hello ";

a += "world";

Basically means, a = a + "world";

let a = 1 + “hello” + 2;

Output: 1hello2

let a = "hello";

let b = "world";

let c = a + " " + b; //without having the mention the space in one of the words

Output: hello world

Evaluates expression from left to right. Different sequences produce different results:

let a = 1 + “hello”;

Output: 1hello

let a = “hello” + 1;

Output: hello1

let a = 1 + 2 + “hello”;

Output: 3hello

let a = “hello” + 1 + 2;

Output: hello12

Comparison operators

Comparison operators are used in logical statements to compare two variables or values. This could be comparison to find the equality between the two variables, or differences.

== -> equal to in value alone

5 == 5 – true

6 == 5 – false

5 == “5” – true, even though they are different types, the value is still 5. It’ll recognize them as equals.

“hello” == “hello” -> true -> Not just numbers, for strings too. Any type.

=== -> equal value and equal type

5 === 5 -> true

5 === “5” -> false, same value, but one is a number and the other is a string

!= -> Not equal in value alone, not type

6 != 5 -> true

5 != “5” -> False, because both are same in value, so this statement is true

!== -> not equal in both value and type

5 !== “5” -> True, because both aren’t of the same type

> -> greater than

5 > 5 -> false

5 > 4 -> true

< -> lesser than

5 < 5 -> False

4 < 5 -> True

>= -> Greater than or equal to

5 >= 5 -> true

5 >= “5” -> true

<= > Lesser than or equal to

5 <= 5 -> True

5 <= 6 -> True

Using comparison operators:

```
let age = 5;
If (age <= 5) {
        Alert(“Hello little one!”);
}
```

This is an if statement. Since the comparison statement is true, the alert inside runs. I’ll cover if statements in detail in a future lesson.

String comparisons

"A" > "Z" -> False

"Boo" > "Bo" -> True

Rules:

Compares first character, and if it finds a comparison, done. If equal, then next character, then next, and so on. If both strings end with same length and same characters, then equal.

If both strings end with same length, but the last character is different, then looks at the ASCII of both and decides which is bigger.

If one string has a bigger length than the other, then the bigger one is greater by default if everything else is equal. Compares with ASCII values (Unicode),

"a" > "A" -> True, because ASCII of a is 97 and ASCII of A is 65.

"ABCD" > "abc" -> False

ASCII is an international number standard for characters. Every letter and special character has an ASCII number.

Comparing different types

When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison.

An empty string converts to 0.

A non-numeric string converts to NaN which is always false.

5 < 5 -> False

5 < “5” -> False -> Converts string to number

5 < “6” -> True

“5” == “6” -> False -> converts both strings to number

“5” < “hello” -> Converts both to number, but “hello” is converted to NaN and comparison against NaN is always false.

“5” < “15” -> False – when converting strings to number, 5 is greater tan 15 because 1 is lesser than 5 and that’ll be the first comparison made.

That is, 1 vs 5 will be the first comparison made and the output will be based on that, not like with full numbers.

Logical operators

Used to determine the logic between two variables or values.

There are three logical operators.

&& - And

Statement 1 && statement 2

If both statements return a value of true, then the entire statement with the && operator is true. Otherwise, the statement is false.

This is how it works:

True, True = True

True, False = False

False, True = False

False, False = False

For example:

(5 <= 5) && (6 > 5) - > True && True -> True

(5 < 5) && (6 > 5) -> False && True -> False

|| - Or

Statement 1 || statement 2

If any one of the statements on either side of the or operator is true, then the entire statement is true. The statement returns false only when both statements are false.

True, True = True

True, False = True

False, True = True

False, False = False

! – Not

If the statement is false, then Not of that statement will return false. If the statement is true, Not of that statement will return true.

!(statement)

!(5 == 5) -> false

!(5 < 4) -> True

Ternary operator or conditional operator

The conditional operator is used to assign a value to a variable based on a comparison made.

Syntax is as follows.

variable = (condition) ? value 1 : value 2;

For example, let's create a value, and the condition is going to be, if the value of a different variable is less than 5, we send one value, if not another value.

```
let age = 5;

let output = (age < = 5) ? "Toddler" : "Kid";

alert(output);
```

Since, age is equal to 5, which makes the condition given true, the value of output will be “Toddler”, and it’ll be printed out in the alert statement.

Change value to 6, and “kid” will be printed out.

Operator precedence

So, the precedence is as follows

()

Postfix increment and decrement

Prefix increment and decrement

Multiplication, division, modulus

Addition, subtraction

Then come the bitwise operators, comparison operators and logical operators.

Module 4: Strings

Creating strings

Now, let's look more into strings. As you already know, strings are a string of characters, hence the name.

let a = "hello";

That's a string.

We can either write strings within double quotes or single quotes.

a = 'hello';

The above works the same as the first statement.

You can even write entire sentences within strings.

a = "Hello, I am here to teach you Javascript";

You can also write quotes within strings. But, the quotes in the string should not match the quote outside it.

For example:

a = "He said, 'I am going home'.";

or

a = 'He said, "I am going home".';

Both are allowed, but using double quotes inside when there are double quotes outside, or single quotes inside when there are single quotes outside will produce errors.

a = "He said, "I am going home".";

Now, a will only have "he said," because the browser will think the string ends where the 2nd double quote appears. That's why there should be a difference in quotes between the strings and the quotes surrounding it, so the browser knows which is which.

But there is a way to go around this. By using a special character.

\ -> By using the backslash in front of the quote, you're letting the browser know that the letter that comes after that is just a character, and not to be confused with a command the browser has to decipher.

a = "He said, \"I am going home\".";

Now, there are no errors. Similarly, for single quotes:

a = "He said, \'I am going home\'.";

Similarly, you can make the backspace appear on a string as well.

a = "\\ is called a backspace";

Only one of those backspaces appears, as you can see.

Creating strings as objects

You can also create strings as objects, like this.

let a = new String("hello");

It's the same as this:

let b = "hello";

If you compare both with == operator, you'll get true.

a == b -> true

But, they are of different types:

typeof a -> object

typeof b -> string

a === b > false, because different types.

But, comparing 2 objects is always false, because they are different objects.

So, if

let b = new String(“hello”);

Even though both a and b are objects with same values,

a == b -> False

a === b -> False

Both false.

String methods

String length

a = "hello";

Syntax to find length of the string:

stringName.length

Example:

alert(a.length);

Can find length of string, no matter how big it is.

Find position of a string or substring in a string:

Index position of the given sub string – use indexOf() string method

Javascript counts position from 0, so in the following string:

let x = “Welcome to Javascript!”;

let y = x.indexOf(“Javascript”);

alert(y);

It is case sensitive.

let y = x.indexOf(“javascript”);

It returns -1, because J is not equal to j. So, Javascript is not equal to javascript.

When a given sub string is not found in the string we are searching in, browser will return -1.

indexOf returns the first position of the first occurrence of the string. If the same sub string is present in the string more than once, it’ll still only consider the first string.

let x = "That that that that that";

let y = x.indexOf("that");

Answer is 5, though there are more than 1 this in x.

But, on the other hand, the lastIndexOf() method will return the last occurrence of the given string.

So, in the same example.

y = x.lastIndexOf("that");

Answer is 20. Because that’s the position of the last occurrence of “that” (count and show).

Again for lastIndexOf method, if the search string is not present in the string, it’ll return a -1.

y = x.lastIndexOf(“hello”); //returns -1

For both indexOf and lastIndexOf methods, you can mention a second parameter that specifies from where you want the search to start.

So, for the last example, for indexOf, let’s start the search from the 12^{th} index position instead of the default 0.

let y = x.indexOf("that", 12);

The answer is 15 because the search starts from 12, so the first occurrence of the given substring is the third “that”.

Search method

You also have another method called search, which allows you to search for a string within your given string, and return the position of occurrence.

let y = x.search("that");

No second parameter.

Slicing strings

You can extract a part of the string and place it another variable by using the slice method.

It literally slices the string and takes it out.

But, it won’t affect the original string. The original string says the same, it’s just that the slice will be duplicated and placed in the new variable.

The slice() method takes 2 parameters, the start position of slicing and the end position.

let x = "This is an example string";

let y = x.slice(8,15);

console.log(y);

Answer is “an exam”

But x is still the same.

If we give negative values for the parameters, it’ll calculate the position from the end of the string.

let y = x.slice(-7,-2);

If you don’t give the second parameter, it’ll slice out from the first index position to the end of the string.

let y = x.slice(8);

Can do the same with negative value.

let y = x.slice(-7);

We have a similar method called substring(). But this method does not accept negative parameters.

let y = x.substring(8,15);

let y = x.substring(8); //rest of the string

substr() – similar to slice

Second parameter specifies the length of extracted string.

None of these methods affect the original string.

let y = x.substr(11,7);

let y = x.substr(11); //rest of string

let y = x.substr(-6); //count from end of string

Replace content

Replace the content of string using replace() method. Again, does not change the actual content of original string. Just creates a new string with replaced value.

let x = "This is an example string";

let y = x.replace("an example", "a sample");

If you want to make it case insensitive, you can use a simple regular expression.

```
let y = x.replace(/this/i, "That");

let y = x.replace(/THIS/i, "That");
```

Also, by default, it'll only replace the first occurrence of the word.

```
let x = "this this this this";

let y = x.replace("this", "That");
```

But, by using the regular expression g, we can make it replace every occurrence of the word in the string.

```
let y = x.replace(/this/g, "That");
```

Convert a string to upper case

```
let x = "this this this this";

let y = x.toUpperCase();
```

Again, does not affect the original string.

```
alert(x);
```

Convert upper case to lower case

```
let x = "This is An Example";

let y = x.toLowerCase();
```

Concat strings

To concat two strings, we usually use the "+" operator.

```
let x = "Hello";

let z = "World";
```

let y = x + " " + z;

We can also use the concat method to achieve the same result.

let y = x.concat(" ", z);

Can add as many strings as you want by adding commas.

let y = x.concat(" ", z, "!", " How are you?");

Trim string

Use trim() method to remove white space from both sides of the string.

let x = " This That This ";

alert(x);

let y = x.trim();

alert(y);

Character codes

Can find characters by mentioning their position.

let x = "This is an example string";

let y = x.charAt(5);

console.log(y);

let y = x.charAt(12);

Can also get the character code of the character at a particular position.

As you know, in computers, every character has an ASCII code.

Small a is 97. Big A is 65, etc (explain more by showing all examples).

You can find the character code of a character at a particular position:

let y = x.charCodeAt(5);

Convert string to array

Can convert a string to an array by using the split method.

```
let x = "This is an example string";

let y = x.split(" ");

console.log(y);

console.log(y[2]);
```

Convert string back to an array

Here, we are splitting the string based on the space. So each word is a new array value.

What if the string has semicolons separating it?

```
let x = "This;is;an;example;string";

let y = x.split(";");

console.log(y);

console.log(y[1]);
```

Similarly, separate based on whatever the separator is in that particular string.

Module 5: Numbers

Creating numbers in Javascript

You can store both whole numbers and decimal numbers.

Whole numbers are called integers and decimal numbers are called floating point numbers.

let a = 5; //integer – whole number

let b = 5.5 //floating point – decimal

Can also store very large or very small numbers by using exponentiation.

So, to store 5000000, you can store it as

let a = 5e6; //means 5 into 1000000

This is how you store big numbers

To store 5/1000000 or 0.0000005, you make it

let a = 5e-6; //how you store very small numbers

Numbers in javascript, that is integers, are only accurate up to 15 digits. After that, you get junk values.

let x = 999999999999999;

alert(x);

let y = 99999999999999999;

alert(y);

Precision decreases after 15 digits.

For floating point numbers, precision is up to 17 digits.

But, floating point arithmetic is not always accurate.

let a = 0.1 + 0.2;

The result is 0.30000000000000004

To make floating point arithmetic accurate, multiple each floating point number by 10, and then divide the entire thing by 10. This only works for addition though.

let a = (0.1 * 10 + 0.2 * 10) / 10;

String and number operations

Do you remember how we looked at adding numbers and strings? The result is always a string right?

This problem only occurs with the addition operator. Let's look at the other operators.

let a = 10 * "20";

let a = 30 - "20";

let a = 30 / "10";

In these instances, browser looks at the operators, and automatically converts the strings to numbers to make the calculations possible.

But + does not work because when a string is present, browser automatically considers + as the concatenation operator and not the addition operator.

But in any of the above examples, it worked because the strings converted to numbers were actually numbers, but what if they were letters?

let a = 30 * "hello";

You'll get a NaN as a result. NaN means, Not a Number.

NaN

NaN is of type number though, but still, it has no value, because you can't perform calculations with numbers and a string of letters right?

typeof(NaN); //says number

You can also find if a number is NaN by using the isNaN() function, which is a predefined method in javascript.

```
let a = 30 * "hello";

console.log(isNaN(a));

let b = 30 * 10;

console.log(isNaN(b));
```

If you know a variable that has NaN in a mathematical operation, the result will also be NaN, if the other value is a number.

```
let c = a + b;

console.log(c);
```

If the other value is a string, the result will be concatenation.

```
let a = 30 * "hello";

let b = "Hello";

let c = a + b;

console.log(c);
```

Infinity

Just like NaN, infinity is also a number.

```
console.log(typeof Infinity);

console.log(typeof -Infinity);
```

Any operation against infinity is obviously infinity.

```
let a = 10 * Infinity;
```

```
console.log(a);

console.log("<br/>");

let b = 10 * -Infinity;

console.log(b);
```

Creating numbers as objects

Just like with strings, numbers can be declared as objects too, but it's not good practice to do that because it reduces the speed of execution.

```
let a = new Number(100);
```

Typeof returns object as well.

```
let a = new Number(100);

let b = 100;

console.log(typeof(a) + ", " + typeof(b));
```

Convert number to string in various formats

We can use the toString() method of Javascript to convert a number to a string, but we can also use it to display the converted number in different number formats.

```
let a = 50;

let b = a.toString(10); //decimal

let c = a.toString(16); //hexadecimal

let d = a.toString(8); //octal

let e = a.toString(2); //binary

console.log(`Decimal: ${b}
```

Hexadecimal: ${c}

Octal: ${d}

Binary: ${e}`;

Output:

Decimal: 50
Hexadecimal: 32
Octal: 62
Binary: 110010

Exponentiation method

toExponential() method returns a decimal point number after converting it to exponentials, with the number of decimal points mentioned.

So, there is a single number, rest will be converted to decimals, and the digits taken are compensated with exponents.

let a = 50.4578;

let b = a.toExponential(2);

let c = a.toExponential(3);

let d = a.toExponential(5);

console.log(b + "\n" + c + "\n" + d);

Output:

5.05e+1
5.046e+1
5.04578e+1

Can give the precision as bigger numbers too.

let d = a.toExponential(12);

Output:

5.045780000000e+1

Now with a bigger number.

let a = 50000.4578;

Output:

5.00e+4
5.000e+4
5.000045780000e+4

Fixing decimal points

toFixed method also returns a string, but it returns a string where the given decimal number has been reduced to the specified number of decimal points.

let a = 50.57698726;

let b = a.toFixed(2);

let c = a.toFixed(5);

let d = a.toFixed(7);

console.log(b + "\n" + c + "\n" + d);

console.log(typeof b);

Output:

50.58
50.57699
50.5769873
string

Fixing length of a number

toPrecision() method returns a string with the number written in a specified length. This is for the whole number, not just the decimal points like with toFixed.

```
let a = 50.57698226;

let b = a.toPrecision(2);

let c = a.toPrecision(5);

let d = a.toPrecision(7);

console.log(b + "\n" + c + "\n" + d);
```

Output:

```
51
50.577
50.57698
```

Converting other types to numbers

Number method

Number() method returns a number converted from the string. Just that. It can be used to convert Boolean values to numbers as well.

```
console.log(
        Number("10") + "\n" +
        Number(true) + "\n" +
        Number(false) + "\n" +
        Number("   10   ") + "\n" +
        Number("Hello"));
```

Output:

```
10
1
0
```

10
NaN

Strings will always return NaN, even if there are numbers inside the string quotes, the browser will not be able to read them as numbers.

You can also use to convert dates to numbers using the Number method.

In this case, you will get back the value of the number of milliseconds passed since 1.1.1970 till the date you gave.

The format of giving the date should be like this:

“Year-Month-Day”

Let’s try with one.

let a = Number(new Date("2018-07-12"));

console.log(a);

Output: 1531353600000

What happens if you give a year before 1970? You’ll get a negative value.

let a = Number(new Date("1969-07-12"));

console.log(a);

Output: -14947200000

In the Number() method, if there are spaces between two numbers, it’ll consider it as an unreadable string, and return NaN.

let a = Number("20 30");

ParseInt method

The next is parseInt() method, which parses the given string into a whole number. In here, spaces is allowed, unlike the Number() method.

Even if there are decimal numbers, it'll still convert it to a whole number.

Let's see some examples.

```
console.log(
        parseInt("2.4552") + "\n" +
        parseInt("568") + "\n" +
        parseInt("55 97") + "\n" +
        parseInt(" 10.004  ") + "\n" +
        parseInt("Hello 20") + "\n" +
        parseInt(" 20 Hello"));
```

Output:

```
2
568
55
10
NaN
20
```

Converts decimals to whole number. Even if there are space between two numbers, it'll still convert the first string to whole number.

It doesn't matter if a number has space around it.

If the first instance of the string is a word or a letter, it'll return NaN.

If the first occurrence of the string is a number, it doesn't matter if the subsequent occurrences are letters, it'll convert that first number and leave it at that.

ParseFloat method

parseFloat converts a number to a decimal point or floating point number.

If the number is a whole number, it'll leave it be. It won't add decimal points to it.

Let's look at examples.

```
console.log(
        parseFloat("2.4552") + "\n" +
        parseFloat("568") + "\n" +
        parseFloat("55 97") + "\n" +
        parseFloat(" 10.004  ") + "\n" +
        parseFloat("Hello 20") + "\n" +
        parseFloat(" 20 Hello"));
```

Output:

```
2.4552
568
55
10.004
NaN
20
```

More properties and methods

We also have functions to find the maximum and minimum possible values in Javascript.

Anything after that number is called Infinite.

```
console.log(
        "Max value: " + Number.MAX_VALUE + "\n" +
        "Min value: " + Number.MIN_VALUE + "\n");
```

Output:

Max value: 1.7976931348623157e+308
Min value: 5e-324

On overflow of the max or min value, we'll get Infinity and -Infinity respectfully.

We can get these values using the POSITIVE_INFINITY and NEGATIVE_INFINITY properties respectfully.

```
console.log(
        "Max value: " + Number.MAX_VALUE + "\n" +
        "Min value: " + Number.MIN_VALUE + "\n" +
        "Negative infinity: " + Number.NEGATIVE_INFINITY + "\n" +
        "Positive infinity: " + Number.POSITIVE_INFINITY + "\n" +
        "Not a Number: " + Number.NaN);
```

Output:

Max value: 1.7976931348623157e+308
Min value: 5e-324
Negative infinity: -Infinity
Positive infinity: Infinity
Not a Number: NaN

You can't use these properties on variables though. You'll get an undefined. These can only be used on Number, because it returns values that are predefined.

```
let a = 100;
console.log(a.MAX_VALUE);
```

Output: undefined

Number.isFinite(100); //returns true/false depending on if a number is finite or not. Returns true if number is a number and is finite.

Number.isInteger(100); //returns true if number does not have decimal points and is a number

Module 6: Conditional statements

Conditional statements: Intro

Conditional statements are used to return something if a condition returns true.

If the condition returns false, then certain lines of codes won't get executed.

There are 2 types of conditional statements:

If statements

Switch statements

If, else

So, if else is just like it sounds. In English, this is what it means:

If something is true,

Do this

Else,

Do this

That's it.

In javascript language, this is how it'll be written

```
if (condition) {

}

else {

}
```

So, to find out if the condition is true or not, we'll use some comparison operators. Some of the commonly used ones are:

== - Equal to

>-Greater than

<-Less than

>= - Greater than or equal to

<= - Less than or equal to

```
let num1 = 5;
let num2 = 6;
if (num1 > num2) {
   alert("Good!");
}
else {
  alert("Not good");
}
```

Else if statements:

Else if statement is used to specify a new condition if the first condition is false.

You can specify as many conditions as you want like this, and at the end, if none of those conditions were true, the else statement will be executed.

```
if(condition1) {
        Statements set 1
}
else if (condition2) {
        Statements set 2
}
else if (condition 3) {
        Statements set 3
```

```
}
.
.
.
else {
        Statements set n
}
```

Let’s look at an example.

```
let num = 5;
let secretNum = 10;
if(num < secretNum) {
        console.log("Go a bit higher");
}
else if(num > secretNum) {
        console.log("Go a bit lower");
}
else {
        console.log("You guessed it right!");
}
```

Switch:

With switch, you can take up multiple conditions. This is the syntax of a switch statement:

```
switch (variable or condition) {
```

```
  case "first" : statements;
                    break;
  case "second" : statements;
                      break;
  default           : statements;
}
```

So, lets create a program that displays a certain message for each number sent to the switch:

```
let num = 5;
switch (num) {
  case 1 : alert("The number is 1");
             break;
  case 3: alert("The number is 3");
            break;
  case 5: alert("The number is 5");
            break;
  case 7: alert("The number is 7");
            break;
  default: alert("The number was not specified");
}
```

Now, if you run the program, the output is “The number is 5”, because num’s value was 5, so the case 5 was executed.

Loops

Loops allow a programmer to execute the same statement or a set of statements again and again, for a number of times specified.

There are 3 kinds of loops: For loop, while loop and do while loop.

So, any loop basically does this:

It checks for a condition, and as long as the condition is true, it'll execute the statements within the loop. Once the condition becomes false, it'll stop executing.

For loop

The syntax of the for loop is as follows:

```
for (initialization; condition; increment) {
  Statements;
}
```

Example:

```
for (i = 1; i <= 5; i++) {
  console.log(i);
}
```

Output: 12345

Initialization is not compulsory. It can be done beforehand.

```
let i = 0;
for(; i<5; i++) {
        console.log(i + "<br/>");
}
```

Can initialize more than one variable in the loop.

```
for(let i=0,sum=0; i<5; i++) {
        sum += i;
}
```

Conditional statement (statement 2) is optional too. But, if you omit it, without providing some kind of break inside the for loop, the loop will run endlessly.

```
for (i = 1; i <= 5; i++) {
  console.log (i);
}
```

Statement 3 doesn't always have to be increment, can be decrement too.

```
for (let i = 5; i >=0 ; i--) {
  console.log (i);
}
```

While loop

Syntax:

```
Initialization;
while(condition) {
        Block of code;
        Increment;
}
```

Ex:

```
let i=1;
```

```
while(i<=10) {
        console.log(i + "<br/>");
        i++;
}
```

Don’t forget to include increment or decrement condition within the loop, or the loop will run endlessly and crash the browser.

For loops and while loops have similar syntaxes. One can be converted to another. Convert the above into a for loop.

```
for(let j=1; j<=10; j++) {
        console.log(j);
}
```

If you omit the first and last statements in the for loop, it looks similar to the while loop.

```
let j=1;
for(;j<=10; ) {
        console.log(j);
        j++;
}
```

Do while loop

While loop will not execute if the condition is false.

```
let i=11;
while(i<=10) {
        console.log(i);
        i++;
```

```
}
```

No output. But, do while is a variant of while loop where it executes at least once, regardless of whether the condition is false or true.

Syntax is as follows:

```
Initialization;
do {
        Block of code;
        Increment;
}while (condition);
```

Same while example as do while.

```
lt i=11;
do {
        console.log(i);
        i++;
}while(i<=10);
```

The above will execute once before it sees the condition is false.

Break, continue, labels

Break statement is used to jump out of a loop. That is, stop execution of the loop, so the statements outside of it can be executed.

```
let i=1;
while(i<=10) {
        if(i===6) {
                break;
```

```
        }
        console.log(i);
        i++;
}
```

Similarly, for for loop:

```
for(let i=1; i<=10; i++) {
        if(i===6) {
                break;
        }
        console.log(i);
}
```

Same goes for do while loop as well.

Break can be used to break out of a loop prematurely, that is, before the condition becomes false.

Continue statement skips one iteration of a loop. It doesn't completely stop the execution of the loop like break does.

With the same examples as above:

```
let i=1;
while(i<=10) {
        if(i===6) {
                continue;
        }
        console.log(i);
```

```
    i++;
}
```

Only 6 will not be printed. Rest won't printed.

Labels are blocks of code.

Let's create a label called numbers, give it a name, then a colon, and enclose within flower brackets.

```
numbers: {
    console.log(1);
    console.log(2);
    console.log(3);
    console.log(4);
    console.log(5);
    break numbers;
    console.log(6);
    console.log(7);
    console.log(8);
    console.log(9);
    console.log(10);
}
```

Give break and mention the label name.

Module 7: Arrays

Arrays allow you to store an array of values inside a variable, and access them separately, or together.

Creating/declaring an array:

let number = new Array();

Assigning values to an array:

number = [1, 2, 3, 4, 5];

Accessing values in an array (starts from index 0):

alert(number[0]);

The output is 1.

We can also change values, or add values this way.

number[5] = 6;

number[4] = 7;

Now,

alert(number[6]);

We get 7, instead of the 5 we originally gave.

We can also store strings and Boolean values, not just numbers.

let number = ["one", "two", "three"];

Can also declare arrays like this:

let array1 = []

Arrays can have multiple data types inside of them:

```
let array1 = ["orange", 2, 3, "apple", true, false, 2.0, 5.5];
```

Declare arrays as objects and assign values:

```
let array1 = new Array("orange", 2, 3, "apple", true, false, 2.0, 5.5);
```

Can print out an entire array:

```
console.log(array1);
```

Check if something is an array:

```
console.log(Array.isArray(array1));
```

You can also use the instanceof operator to get the same result:

The **instanceof** operator returns true if an object is created by a given constructor:

```
console.log(array1 instanceof Array);
```

Loop through an array:

You can loop through the entire array items, and do anything you want with it.

```
let array1 = [2, 6, 4, 9, 3, 7];
let sum = 0;
for(let i=0; i<6; i++) {
        sum += array1[i];
}
console.log("Sum of the array values is: " + sum);
```

Assign iteration variable to 0, because array indices start from 0. Should only go right before the length of the array, so 0 to 5, which is 6 iterations.

Array length

You can find the number of items in an array using the length property.

```
console.log(array1.length);
```

Push method

You can also use the push method, which is a built in javascript method, to add new items at the end of the array.

```
array1.push(5);

console.log(array1);
```

If you want to push a string:

```
array1.push("string");

console.log(array1);
```

You can also push a variable inside the array.

```
let x = 20;

array1.push(x)

console.log(array1);
```

Also, if you assign the push statement to another variable, you'll get the new length of the array.

```
let y = array1.push("New");

console.log("New length of array: " + y);
```

Pop method

you can pop items out of the array. Use the pop method to pop the last element in the array.

```
let array1 = [2, 6, 4, 9, 3, 7];
```

array1.pop();

console.log(array1);

Can also assign the popped item to a variable.

let x = array1.pop();

console.log(x);

Join method

let array1 = [2, 6, 4, 9, 3, 7];

let newString = array1.join();

console.log(newString);

console.log(typeof newString);

Similar to above.

Now, different separators.

let string2 = array1.join(".");

console.log(string2);

let string3 = array1.join(" | "); //can give space, or multiple characters as separators

console.log(string3);

let string4 = array1.join(" *^ ");

console.log(string4);

Module 8: Date, Date methods & Math object

It displays the date as a string in the time zone of the user's browser.

let d = new Date();

Get PI value

let p = Math.PI;

You can round a given number to its nearest value by using the round() method of the Math object, like this:

console.log(Math.round(5.5));

console.log(Math.round(5.6));

console.log(Math.round(5.4));

You can round a number to its ceiling value by using the ceil() method.

console.log(Math.ceil(5.4));

You can round a number to its floor value by using the floor() method.

console.log(Math.floor(5.6));

You can use the Math.pow(x,y) syntax to calculate x to the power of y.

console.log(Math.pow(3,4));

You can find the square root of a number by using the sqrt() method.

```
console.log(Math.sqrt(25));
```

You can use the trunc() method to find the integer value of a given decimal number.

```
console.log(Math.trunc(5.36));
```

You can use the abs() method to find the absolute value of a number, that is, positive value of a number.

```
console.log(Math.abs(-15));
```

You can find the minimum and maximum value of a set of values as well.

```
Math.min(10,25,-50,50,100,0);
```

```
Math.max(10,25,-50,50,100,0);
```

Math.random() – generates a number between 0 to 1.

To get random numbers from 0 to 9,

```
Math.floor(Math.random() * 10))
```

1 to 9

```
Math.floor(Math.random() * 10)) + 1
```

1 to 100

```
Math.floor(Math.random() * 100)) + 1
```

Module 9: Functions

A javascript function is a block of code that performs something, like calculations, comparisons, etc. Functions are used for a lot of purposes in programs.

They save lines of code and time, since a function can be called again and again to execute the same block of code within them.

Syntax:

Function declaration/definition:

```
function functionName() {

}
```

Function call:

```
functionName();
```

Example:

Prints “Hello there” when the function is called.

```
function printHello() {
  alert("Hello there!");
}
printHello();
```

Send values to a function:

Create a function that adds 2 numbers

```
let num1 = 5;
let num2 = 6;
```

```
function sumNum(n1, n2) {
  let sum = n1 + n2;
  return sum;
}
sumNum(num1,num2);
```

Return statement returns the given value back to the calling statement.

Local variables:

Variables created within functions only have scope within them. They are called local variables.

```
function name() {
        let num;
}
alert(num);
```

Reference error: num not defined. That's because num is a local variable that is created within the function "name" and only works within that. It has local scope or function scope.

Default parameter values:

Can give default parameter values too, in case we don't send values from function calls. Just mention the parameter with an equal to and the value.

```
function addNum(num1 = 5, num2 = 5) {
        let sum = num1 + num2;
        alert(sum);
}
addNum(2,7);
```

```
addNum();
```

Can be string or Boolean too, not just numbers.

```
function whoIsThis(child = true, message = "Child!") {
	if(child === true) {
		alert(message);
	}
	else {
		alert("Nope!");
	}
}
whoIsThis();
whoIsThis(false);
whoIsThis(true);
whoIsThis(true,"Baby!");
```

Give function as default parameter

```
function addNum(num1, num2 = mulNum()) {
	let sum = num1 + num2;
	alert(sum);
}
addNum(5);

function mulNum(num1 = 2, num2 = 3) {
	let mul = num1 * num2;
```

```
    console.log(mul);
    return mul;
}
```

Empty return statement

```
function addNum(num1, num2) {
    sum = num1 + num2;
    return;
}
addNum(5,2);
alert(sum);
```

But an empty return like above is the same as saying return undefined;. It returns undefined.

Functions as values

Functions can be used as values in calculations as well, like this:

```
"use strict";
function addNum(num1, num2) {
    let sum = num1 + num2;
    return sum;
}
let mul = addNum(5,2) * 3;
alert(mul);
```

Return multiple lines of return:

If you want to return multiple lines of return statement, then wrap them in parentheses. It should start on the same line though.

```
function addNum(num1, num2) {
    let sum = num1 + num2;
    return(`Hello there!
The sum of numbers ${num1} & ${num2} is ${sum}`);
}
alert(addNum(5,2));
```

Function expressions/Anonymous functions

```
let varName = function(param1,param2...) {
    statement1;
    statement2;
    .
    .
    .
}
```

When you write it like this, the function's just a value stored to the variable varName.

These functions are called anonymous functions because they don't have a function name.

```
let age = function(a=5) {
    alert(a);
};
```

```
console.log(age);
```

When we alert age, we get the function definition. It's the string representation of the function. That's what happens.

But, alternatively, you can call the function and let it run like this:

```
age();
```

It works similar to a normal function call.

This function can be copied to another variable like this:

```
let age1 = age;
age1();
```

Function declaration can be called before it is declared, meaning, it can be executed before we reach the line of code where the declaration is present.

Also, function declarations created within code blocks have a block scope and cannot be called from outside the block.

```
var bool = true;
var age;
if(bool == true) {
        age = function(a = 5) {
                alert(a);
        }
}
age();
```

Self-Invoking functions

You can make function expressions self-invoking. Basically, you can make them call themselves.

You can't make function declarations self-invoking though, only function expressions.

Add a () after the function to indicate that it's self-invoking.

```
"use strict";

(function() {

        alert("I am a self-invoking function!");

})();
```

Or, you can use a traditional function expression and make it self-invoking as well, like this:

```
var age = function() {

        alert("Hello child!");

}();
```

Arrow functions

It's an ES6 update. These are also called as Fat arrows.

They are used to shorten the functions.

Originally:

```
let helloFunc = function() {

        return "Hello world";

}

console.log(helloFunc());
```

Shorten it:

```
let helloFunc = () => {
        return "Hello world";
}
console.log(helloFunc());
```

Shorten it further:

If there is only one line of code in the function.

```
let helloFunc = () => "Hello world";
console.log(helloFunc());
```

Or:

```
let helloFunc = () => alert("Hello world");
helloFunc();
```

If you have parameters, usually, you'll write like this:

```
let sumNum = function(a,b) {
        return a + b;
}
console.log(sumNum(5,6));
```

Shorten it:

```
let sumNum = (a,b) => a + b;
console.log(sumNum(5,6));
```

Without parentheses:

If only one parameter, then no need for brackets:

let dispString = a => alert(a);

console.log(dispString("Hello World!"));

Doesn’t work with multiple parameters

Arguments object

There’s a built-in object called the arguments object that’ll store all the arguments in the form of an array.

You can access the arguments later as you would with array items, and also calculate the length with the length method, if needed.

If there are too many arguments in a function call, the arguments object is a great way to access them.

```
"use strict";

function addNum() {
  let sum = 0;
  for(let i=0; i<arguments.length; i++) {
          sum += arguments[i];
  }
        alert(sum);
}

let valSum = addNum(15,66,73,24,99,16,49,89,35,26,14);
```

Recursive functions

Recursive functions are functions that call themselves.

Recursions at their base can be used in place of loops and iterations.

Let’s look at a simple example of how we’d do it:

Let’s write a code to find the sum of the first n numbers.

Generally, we’d write it like this:

```
function addNum(n) {
	let sum = 0;
	for(let i=1; i<=n; i++) {
		sum += i;
		console.log("Sum at step",i,"is",sum);
	}
	return sum;
}
console.log(addNum(5));
```

But the code is bigger. We can make this smaller by calling the function within itself. That is called recursion.

It’ll be written like this:

```
function addNum(n) {
	if(n == 1) {
		return n;
	}
	else {
		return n + addNum(n-1);
	}
}
console.log(addNum(5));
```

Let’s have n as 5 to start with.

5 + addNum(5-1)

addNum(4) -> 4 + addNum(4-1)

addNum(3) -> 3 + addNum(3-1)

addNum(2) -> 2 + addNum(2-1)

addNum(1) -> return n, which is 5 + 4 + 3 + 2 + 1 = 15

Lesser number of lines of code and more efficient.

Spreading an array into function arguments

What if I send an array of numbers and ask to find the sum of it? How will the browser separate them? We can use the spread syntax to do that:

Try to below lines of code in your editor:

```
//Find the sum of the given 2 numbers
function addNum(num1, num2) {
        sum = num1 + num2;
        return sum;
}
let arr = [5,4];
console.log(addNum(arr));
```

If you run the above, you’ll get an error because the array is not separated into its separate items.

To do that, use the separator syntax while calling the function, like this:

```
console.log(addNum(...arr));
```

Now, run it and you’ll get the result.

You can use the spread syntax with pre-defined functions as well:

To find the max of given numbers using the pre-defined max function, use the below line of code:

```
console.log(Math.max(4,6,-4,2));
```

If we use arrays, we'll get an array, but we can use the spread syntax for that as well.

```
let arr = [4,6,-4,2]

console.log(Math.max(...arr));
```

Can use for multiple arrays too:

```
let arr = [4,6,-4,2];

let arr2 = [15,6,-32,54];

console.log(Math.max(...arr, ...arr2));
```

You can use it to combine arrays too:

```
let arr = [4,6,-4,2];

let arr2 = [15,6,-32,54];

let arr3 = [43,23,...arr,5,...arr2,12];

console.log(arr3);
```

Rest parameters

If a function is sent extra arguments, there won't be an error. They just won't be taken into account.

For example, look at the below lines of code:

```
function addNum(num1,num2) {
        let sum = num1 + num2;
```

```
    return sum;
}
console.log(addNum(5,3,6,7,3));
```

There won't be an error even though you're sending too many extra arguments. They just won't be taken into account at all.

But there is a way to take the extra arguments into account, accept just the extras, or the entire arguments list into an array and use them in the function. You can use the rest syntax to do that.

We'll be using the same "…", but this time, in the function definition.

```
function addNum(num1,num2,...arr) {
    let sum = num1 + num2;
    console.log(arr);
    return sum;
}
console.log(addNum(5,3,6,7,3));
```

We can use the array inside the function as well:

```
function addNum(num1,num2,...arr) {
    let sum = num1 + num2;
    for(let a of arr) {
        sum += a;
    }
    return sum;
}
console.log(addNum(5,3,6,7,3));
```

We can accept the entire list of arguments into a user defined array name as well:

```
function addNum(...arr) {
    let sum = 0;
    for(let a of arr) { //explain this syntax
        sum += a;
    }
```

setTimeout and setInterval

There are pre-defined scheduling functions in Javascript that schedule functions. They are the setTimeout and setInterval functions.

The setTimeout function runs a function after a set time delay.

Syntax is as follows:

setTimeout(functionName, delayInMilliSeconds, arg1, arg2, ….);

arg1, arg2 etc are the arguments of the function.

The function's name is case sensitive.

functionName is the function that must run after the time delay. You can give an anonymous function in here, but it's not recommended.

Give the delay in milliseconds. So, if you need the function to run after a delay of 1 second, your 2nd argument should be 1000.

Let's look at an example where we call a function after a 5 second delay.

```
function helloThere() {
    alert("Hello there!");
}
setTimeout(helloThere, 5000); //after 5 second delay
```

If you want to send arguments to the function:

```
function helloThere(name) {
        alert("Hello there, " + name + "!");
}
setTimeout(helloThere, 2000, "John"); //after 2 second delay
```

You can also run pre-defined functions like alerts from within the setTimeout call. Give them as strings and your browser will piece everything together and run it as a function.

```
setTimeout("alert('Hello there!')", 2000);
```

You can give the same as an arrow function. In fact, it's recommended that you do so.

```
setTimeout(()=>alert('Hello there!'), 2000);
```

We can cancel this time out if needed. Assign this to another variable. It returns a timer identifier, and you can use the clearTimeout function, with the timer identifier as the argument to cancel the Time out you set.

```
let tr = setTimeout(()=>alert('Hello there!'), 2000);
console.log(tr);
clearTimeout(tr);
console.log(tr);
```

The timer identifier is returned as a number.

Since we cancelled it, nothing happens.

We can give conditions based on this as well (for cancellation), if needed.

With the setInterval function, you can run the function repeatedly based on the given time interval. 'I' is caps here and its case sensitive.

The syntax is similar to the setTimeout function.

setInterval(functionName, delayInMilliSeconds, arg1, arg2, ….);

Example:

Outputs in console every 2 seconds. Keeps running.

```
let count = 0;
setInterval(()=>console.log(++count + " alert"),2000);
```

To stop the timer after a point:

```
let count = 0;
let tr = setInterval(()=>console.log(++count + " alert"),2000);
setTimeout(()=>{clearInterval(tr);console.log("Stopped")},10000);
```

Stops after 5 counts of the setInterval or after 10 seconds and logs stopped in the console.

Module 10: Javascript objects (basics)

You can model real world things and occurrences in Javascript by using a concept called objects.

This is more popularly known as object oriented programming, where you try to solve real world problems with a programming language, using objects.

Let's say these are the properties of the first person:

Person1:

Name: Susan

Age: 35

Height: 160

Hair: blond

And let's say these are the properties of the 2nd person.

Person2:

Name: Gary

Age: 40

Height: 180

Hair: blond

Let's create them as objects.

Use flower brackets, and create them as property: value pairs.

So, for person1:

```
let person1 = {name: "Susan", age: 35, height: 160, hair: "blonde"};
```

For person2:

```
let person2 = {name: "Gary", age: 40, height: 180, hair: "blond", eyeColor: "blue"};
```

Now, how do we access these values?

There are two ways to do this:

1. objectName.propertyName or
2. objectName["propertyName"]

alert(person1.height);

Or

alert(person1["height"]);

Create array of objects:

let person = [{name: "Susan", age: 35, height: 160, hair: "blonde"}, {name: "Gary", age: 40, height: 180, hair: "blond", eyeColor: "blue"}];

Accessing values:

alert(Person[1].eyeColor);

Module 11: Javascript Document Object Model (DOM)

When a web page is loaded, the browser creates a document object mode, abbreviated as DOM, of the page.

It's basically a tree that has a record of the hierarchy of a html page.

Getting elements from HTML using Javascript

You can retrieve HTML elements by their respective unique ids.

document.getElementById("id");

Example:

In your index.html file:

In the script.js file:

document.getElementById("para1");

Changing the text of a html element

You can do this by using the innerText property.

document.getElementById("para1").innerText = "Hello, how are you?";

Or:

let para = document.getElementById("para1");

para.innerText = "Hello, how are you?";

Access multiple elements by their tag name:

Create more paras:

```
<p id="para">Hello there!</p>
        <p>This is the first para</p>
        <p>This is the second para</p>
let x = document.getElementsByTagName("p");
console.log(x); //array with all 3 elements (show)
console.log(x[0]); //first element (show all)
x[1].innerText = "Changed this para";
```

Access multiple elements by their class name:

Add some classes:

```
<p id="para">Hello there!</p>
        <p class="grp">This is the first para</p>
        <p class="grp">This is the second para</p>
let x = document.getElementsByClassName("grp");
console.log(x);
```

Access each array for each element.

```
x[1].innerText = "Changed this para";
```

Now the last para is changed because that's the 1st indexed element.

Accessing based on particular CSS query selectors:

You can access based on query selectors as well, like this:

p#idName

p.className

types

attributes

etc.

For example, add a div to your example:

This is a new div

Now, if you used byClassName, you'll get the div added as well.

What if you wanted only the paragraphs with the class "grp" to be considered?

let x = document.querySelectorAll("p.grp");

Now it'll work as you want it to.

Changing the HTML outputs:

document.write() – writes directly to the browser

Never use it after a full page is loaded – it'll overwrite whatever is in the page

Use it for testing purposes for a new page, if need be. Or use it do learn javascript, just like we use the console.log()

Change HTML content:

You can do this by using the inneHTML method.

let x = document.getElementById("para");

x.innerHTML = "Demonstrating inner html!";

That works similar to innerText, but you can create actual HTML code here. That's the difference.

It'll overwrite the HTML code that was already inside.

This is div1.

```
        <p id="para">Hello there!<p>
    </div>
```

```
let x = document.getElementById("div1");
x.innerHTML = "Hello There!";
```

The entire div was erased, including the text inside and the p tag inside and only the new text we gave remains.

```
x.innerHTML = "<p>New paragraph</p><p>Second new paragraph</p>";
```

Now, it overwrote the original <p> and text, and created 2 new paras and that's displayed in the browser, nothing else.

Change value of attribute:

You can change value of HTML attributes – id, class names, src of images, href of links etc

```
<a id="link" href="https://google.com">Link</a>
```

Now, you're going to change the href of the link.

```
let x = document.getElementById("link");
x.href = "https://bing.com";
console.log(x);
```

You can change the text inside as well.

```
x.innerText = "Bing";
```

Change CSS properties:

You can access the CSS properties with the style attribute, and give the particular attribute you want to change as well.

```
x.style.textDecoration = "none";
```

This is similar to writing:

```
#link {
        text-decoration: none;
}
```

In Javascript, whatever had “-“ in CSS would be changed to Camel case, like above.

You can change multiple values like that.

You can override CSS properties with Javascript as well.

<link rel="stylesheet" type="text/css" href="style.css"> (in html)

```
#link {
        color: red;
}
```

Now, override that with Javascript.

```
x.style.color = "green";
```

Events

Events can be used to create an action-reaction effect.

We can create a click event that does something when a HTML element is clicked, like this:

```
<button id="button1" onclick="buttonClick()">Button</button>

buttonClick() {
  alert("Thanks for clicking!");
}
```

So, whenever the button is clicked, an alert that says, “Thanks for clicking” would pop up.

You can achieve the same result without having the function call in your HTML code.

It’s always best practice to separate your HTML, CSS and Javascript codes into separate files.

You can use event listeners to achieve the same result, like this:

Retrieve the element first and then create a click event listener on it:

```
document.getElementById("button1").addEventListener("click", buttonClick);
```

Or, do it like this:

```
let button1 = document.getElementById("button1");

button1.addEventListener("click", buttonClick);
```

Other mouse events:

All of them are case sensitive:

onchange – activate once a html element has changed

onmouseover – moved mouse over an element

onmouseout – moved mouse away from element

onkeydown – pushes key on element – not just click – pressing down

onload - page has finished loading on browser

You can create event listeners for each of these events:

Mouse down:

Does something when the mouse button (left) is pressed down on the element.

```
<div id="mouseTest"><div>
```

Let’s create a quick border for it, so you know where it is.

```
#mouseTest {
        border: 1px solid black;
}
```

So, first retrieve the div element.

```
let mouseTest = document.getElementById("mouseTest");
```

Now, create a mousedown event for this

```
mouseTest.addEventListener("mousedown", downEvent);

function downEvent() {
        mouseTest.style.backgroundColor = "red";
}
```

Now, create an event for what happens when the mouse pointer moves on the given element:

```
mouseTest.addEventListener("mousemove", moveEvent);
function moveEvent() {
        mouseTest.style.backgroundColor = "yellow";
}
```

Other events:

keypress – when an element is pressed on by a keyboard key

keydown – when the key is pressed

keyup – when the key is let up

mousedown – when mouse button is clicked over element or one of its children

mouseup – mouse pointer stops pressing over element

mouseenter – mouse pointer enters element

mouseleave – mouse pointer leaves element

mousemove – mouse pointer moving over an element

Module 12: Javascript Regular Expressions Basics

Using regular expressions in Javascript & flags:

Basic regex pattern

```
let string = "This is a simple Javascript string";
let rex = /Javascript/;
```

Using regex to match and test strings in Javascript

Test method:

```
let check = rex.test(string);
document.write(/Javascript/.test("This is a simple Javascript string"));
```

(or)

```
console.log(/Javascript/.test("This is a simple Javascript string"));
```

Match method:

```
let check =  string.match(rex);
console.log(check);
```

Regular expression modifiers (flags)

Global flag – gets all instances of the regular expression from the given string

```
let rex = /Javascript/g;
```

Case insensitive flag – Makes the search case insensitive (a and A are the same)

```
let rex = /Javascript/gi;
```

Multiline flag – can be used to search across multiple lines

let rex = /Javascript/m;

Regular expressions in search and replace

let string = "This is my first javascript Regex. Javascript javascript Javascript";

let rex = /Javascript/gi;

document.write(string.search(rex));

Now, the replace function.

document.write(string.replace(rex, "Typescript"));