



Tweet

**Jessica Rosenworcel** 

@JRosenworcel



Happy 50th birthday internet.

On this day in 1969 the first message went out on ARPANET, the network of computers that would grow to become the internet.

I still believe being connected can do good. So let's use this birthday to fight for the web we want.

H/T [@timberners_lee](#)

11:01 AM · 10/29/19 · [Twitter Web App](#)

13 Retweets **47** Likes

Tweet your reply





ALEXANDRIA OCASIO-CORTEZ (D-NY)



MARK ZUCKERBERG

RE OF
BOOK

House Financial Services Committee
Rayburn House Office Building

C-S



jack



@jack



We've made the decision to stop all political advertising on Twitter globally. We believe political message reach should be earned, not bought. Why? A few reasons...



4:05 PM · Oct 30, 2019 · [Twitter for iPhone](#)

[link to thread](#)

HTML



CSS



JS




intro to JavaScript


May 1995 - **Brendan Eich** - Netscape developer creates Mocha, renamed LiveScript
Dec 1995 - Netscape renames to JavaScript
1996/1997 - Netscape standardizes ECMA Standards Organization
2014 - Quits Mozilla after pressure from shareholders for supporting far right groups
trying to undo Prop 8.

Brendan Eich
Image from wikipedia

Brendan Eich



Brendan Eich, official Mozilla Foundation photograph, August 21, 2012

Born	July 4, 1961 (age 57) Pittsburgh, Pennsylvania, U.S.
Residence	San Francisco Bay Area
Alma mater	University of Illinois at Urbana-Champaign Santa Clara University
Known for	JavaScript
Website	brendaneich.com 

JavaScript has nothing to do with Java
Named that way for marketing reasons

The first version was written in 10 days

Several fundamental language decisions were made because of company politics and not technical reasons

"I was under marketing orders to make it look like Java but not make it too big for its britches ... [it] needed to be a silly little brother language." [\(source\)](#)

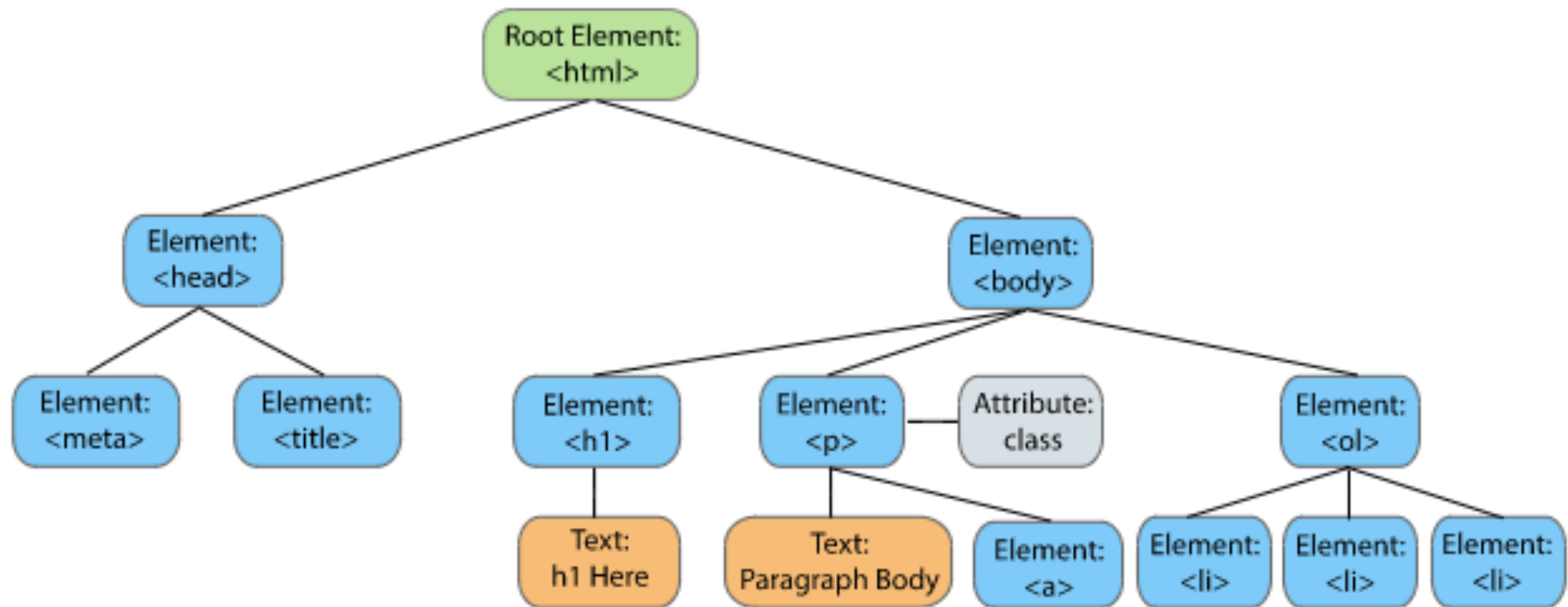
There is **no "main method"**

- The script file is executed from top to bottom.

There's **no compilation** by the developer

- JavaScript is compiled and executed on the fly by the browser

(Note that this is slightly different than being "interpreted": see [just-in-time \(JIT\) compilation](#))



types of DOM nodes

There are four main types of nodes.

- The **Document** node, which represents the entire page
- **Element** nodes, which represent individual HTML tags
 - **Attribute** nodes, which represent attributes of HTML tags, such as class
- **Text** nodes, which represents the text within an element, such as the *content* of a p tag

We talk about the relationship between element nodes as “parents,” “children,” and “siblings.”

DOM queries

JavaScript methods that find elements in the DOM tree are called “**DOM queries**”

DOM queries may return one element, or they may return a “node list”

Which DOM query you use depends on what you want to do and the scope of browser support required

For Example: JavaScript methods that return a single element node:

`getElementById()`

`querySelector()`

When an HTML document is loaded into a web browser, it becomes a **document object**. The document object provides properties and methods to access all node objects, from within JavaScript.

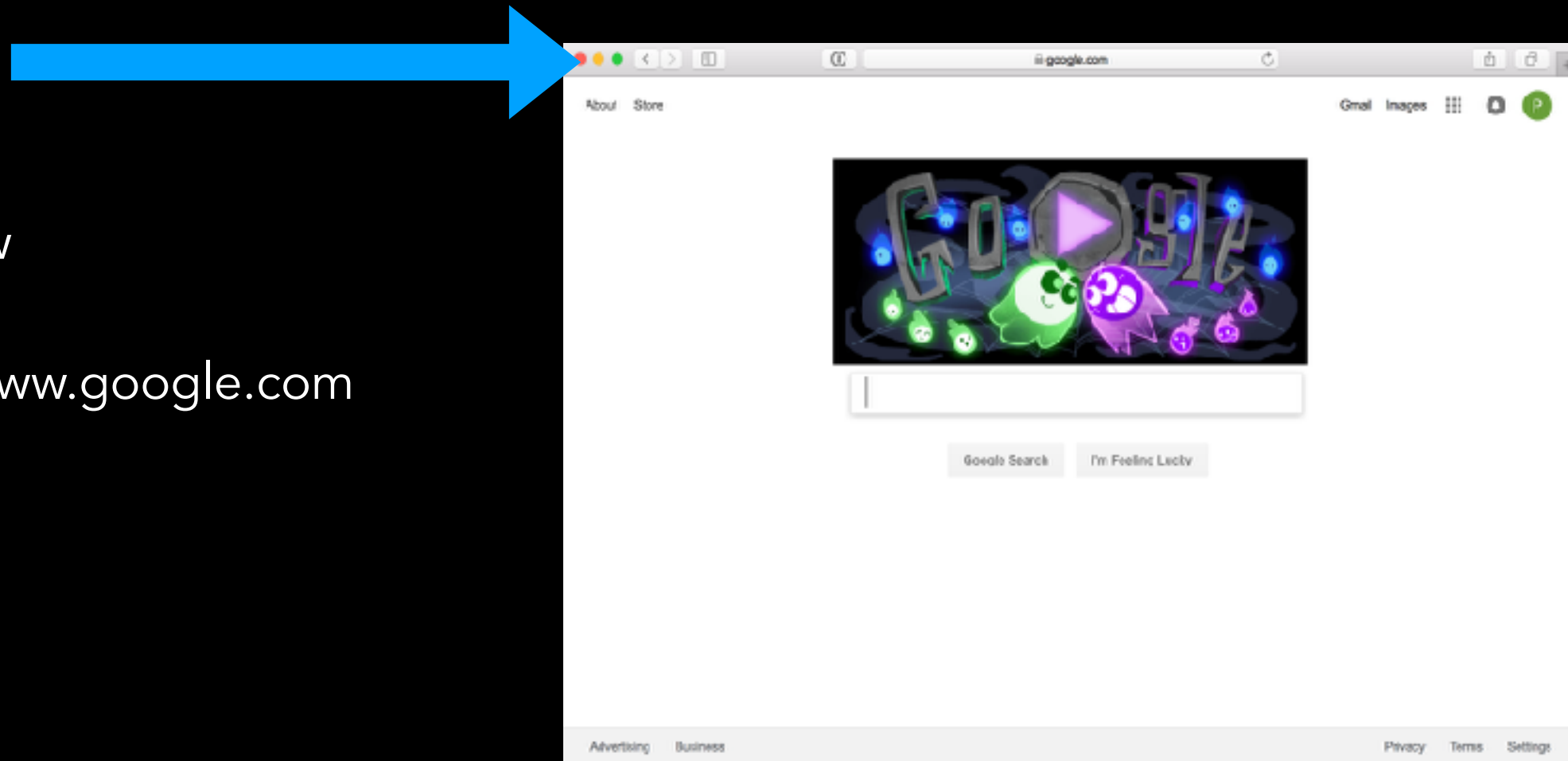
Window Object

The browser represents each window or tab using a **window object**. The **location property** of the **window object** will tell you the URL of the current page.

Object type: window

Properties:

location <http://www.google.com>



try this command in the window inspect console: **console.log(window.location)**
(it will return the current url)

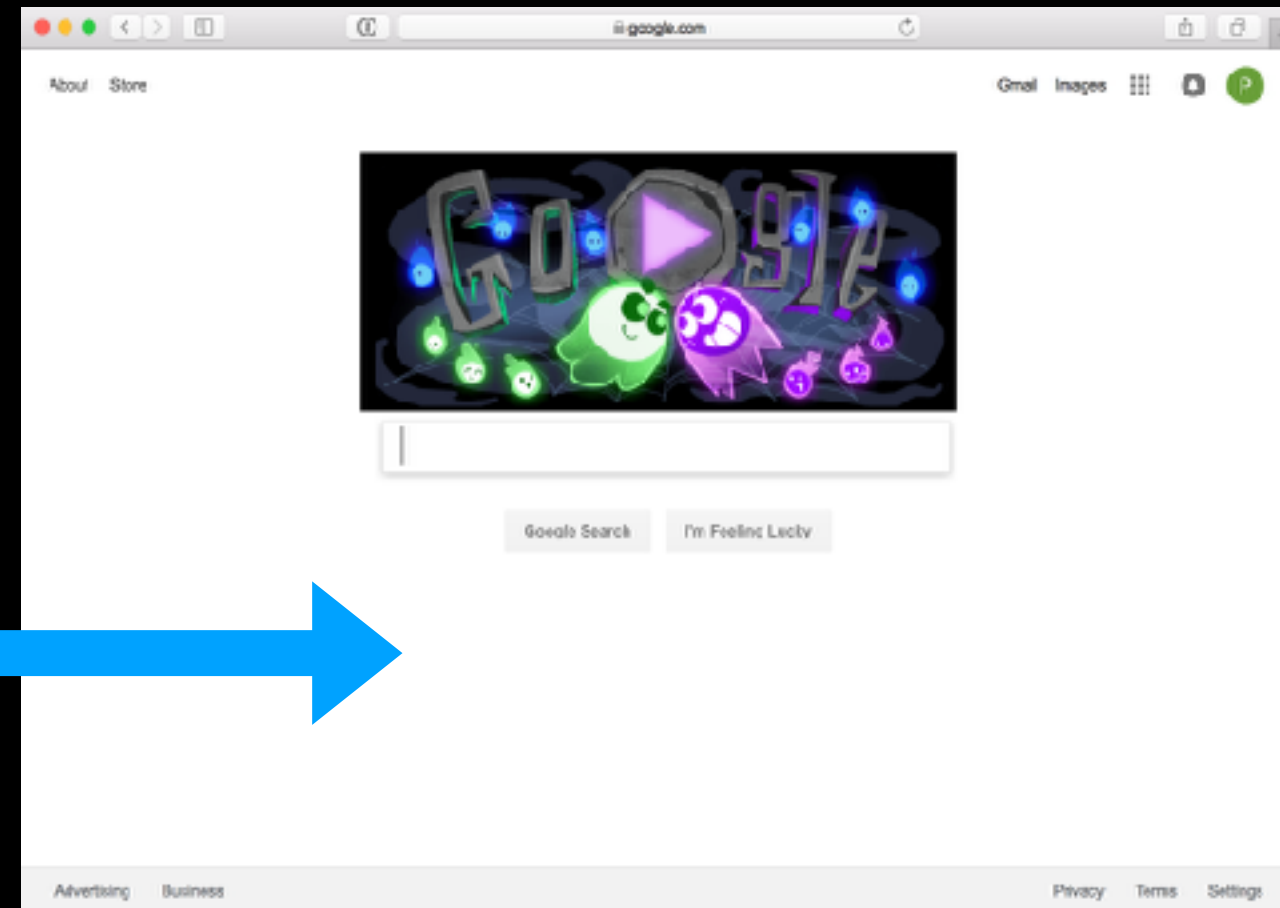
Document Object

The current webpage loaded into each window is modeled using a document object.

Object type: document

Properties:

URL	http://www.google.com
lastModified	10/31/18 6:12 pm
title	Google Search Home Page



try this command in the window inspect console: `console.log(document.title)`
(it will return the current url)

What JavaScript Can Do:

Access content: you can use JavaScript to select any element, attribute or text from an HTML page.

Modify content: you can use JavaScript to add elements, attributes, and text to the page or remove them.

Program rules: you can specify a set of steps for the browser to follow, which allows it to access or change the content of a page.

React to events: you can specify that a script should run when a specific event has occurred.

JS Syntax

1. A semicolon notes that a step is over + JavaScript interpreter should move to the next step
2. Each statement starts on a new line
3. JavaScript is case sensitive: myName is not equal to myname or MYNAME
4. Some statements can be organized into code blocks:

```
if (value > theVariable) {  
    //do this;  
}
```

notice no ; after the code block

JS Syntax

let

allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the **var** keyword, which defines a variable globally, or locally to an entire function regardless of block scope.

At the top level of programs and functions, let, unlike var, does not create a property on the global object.

**** Note:** the browser is not able to identify local variables.

const

Constants are block-scoped, much like variables defined using the let statement. The value of a constant cannot change through reassignment, and it can't be redeclared.

JS Syntax - Variables best practices

- Use **const** whenever possible.
- If you need a variable to be reassignable, use **let**.
- Don't use **var**.
- You will see a ton of example code on the internet with var since const and let are relatively new.

However, **const** and **let** are well-supported, so there's no reason not to use them. (This is also what the Google and AirBnB JavaScript Style Guides recommend.)

JS Syntax

Variables

Variables are containers that you can store values in. You start by declaring a variable with the var keyword, followed by any name you want to call it

let theVariable;

let theVariable = 'Bob';

——> String Data

let theVariable = 10;

——> Numeric Data

let theVariable = true;

——> Boolean

let theVariable = [536, 3, 3354684, 325]

——> Array

Data Type: Numeric

1. Numeric data type handles numbers
2. Use **typeof** to log what kind of data type you're using

```
let thePrice;
```

```
let theQuantity;
```

```
let theTotal;
```

```
thePrice = 5;
```

```
theQuantity= 14;
```

```
theTotal = thePrice * theQuantity;
```

```
console.log(typeof theTotal);
```

Data Type: Strings

1. The string data type consists of letters and other characters
2. They have to be enclosed with a pair of quotes (single or double)
Opening quote must match the closing quote
3. They can be used working with any kind of text

```
let myName;
```

```
let myMessage;
```

```
myName = "Rebecca";
```

```
myMessage = "Atlanta is a really great show y'all should check it out!";
```

```
console.log(typeof myName);
```

Data Type: Boolean

1. Boolean data type can have one of two values: true or false
2. They are helpful when determining which part of a script should run (when code can take more than one path);

```
let todayIsCloudy;  
todayIsCloudy = true;
```

```
if (todayIsCloudy == true) {  
    // if true, show this  
    console.log("Yes, today is cloudy!");  
    // otherwise show this  
} else {  
    console.log("No, it's actually sunny today!");  
}
```


Shorthand for creating variables

You can use these shorthands to create variables:

1:

```
let speed = 5;
```

```
let quantity = 14;
```

```
let total = speed * quantity;
```

2:

```
let speed, quantity, total;
```

```
speed = 5;
```

```
quantity = 14;
```

```
total = speed * quantity;
```

3:

```
let speed = 5, quantity = 14;
```

```
let total = speed * quantity;
```

Changing the value of a variable

Once you have assigned a value to a variable, you can then change it later in a script

```
let speed = 5;  
let quantity = 14;  
let total = speed * quantity;
```

//maybe something changed here and the value has to change now

```
speed = 10;
```

```
var textToShow = document.getElementById("hello");  
textToShow.innerHTML = "Total cost is: " + "$" + total;
```

Rules for naming variables

1. The name must begin with a letter, \$ sign or an underscore _, it cannot start with a number (e.g. name, \$name, _name) - I would avoid \$, because it might confuse it with jquery
2. You cannot use dash - or a dot . in a variable name (e.g. my-name, my.name)
3. You cannot use keywords or reserved words as variable names (e.g. function, type, this, etc.) - it usually changes the color when you use it
4. All variables are case sensitive (it is bad practice to create the same name using different cases (e.g. myName & Myname) - do not start with a capital letter in general
5. Use a name that describes the kind of information that the variable stores (e.g. firstName, lastName)
6. If a variable name is made up of more than one word use capital letter for the first letter of every word after the first one or underscore (e.g. myFirstName, myLastName, my_first_name, my_last_name)

Operators allow us to create single values from one or more values.

Types of operators

Assignment operators (assign values to variables):

```
let theMovie = "Blade Runner";
```

Arithmetic operators (perform basic math):

```
let height = 50 * 3;
```

String operators (combine two strings):

```
let sentence = "My name is " + "Rebecca";
```

Comparison operators (compare two values and return true or false):

```
height = 50 > 3 (will return false)
```

Logical operators (combine expressions and return true or false):

```
height = (50 < 3) && (3 > 2) (will return true)
```

Arithmetic Operators in JS

NAME	OPERATOR	PURPOSE & NOTES	EXAMPLE	RESULT
ADDITION	+	Adds one value to another	10+5	15
SUBTRACTION	-	Subtracts one value from another	10-5	5
DIVISION	/	Divides two values	10/5	2
MULTIPLICATION	*	Multiplies two values	10*5	50
INCREMENT	++	Adds one to the current number	i=10; i++;	11
DECREMENT	--	Subtracts one from the current number	i=10; i--;	9
MODULUS	%	Divides two values and returns the remainder	10%3;	1

String Operators in JS

There is only one string operator **+**

It's used to join the strings together

```
let firstName = "Rebecca";
```

```
let lastName = "Leopold";
```

```
let fullName = firstName + lastName;
```

Process of joining two or more strings together into a new one is: **concatenation**.

If you'll try to add other arithmetic operators on a string, it will return NaN

```
let fullName = firstName * lastName;
```

returns: "Not a Number"

Logical Operators

LOGIC	OPERATOR	EXAMPLE	NOTES
AND	&&	exprss1 && express2	Returns expr1 if it can be converted to false ; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true ; otherwise, returns false .
OR		exprss1 express2	Returns expr1 if it can be converted to true ; otherwise, returns expr2. Thus, when used with Boolean values, returns true if either operand is true .
NOT	!	! express	Returns false if its single operand can be converted to true ; otherwise, returns true .

An **expression** results in a single value (produces a value and is written whenever a value is expected).

Types of expressions

Expressions that assign a value to a variable

```
let theMovie = "Blade Runner";
```

Expressions that use two or more values to return a single value

```
let theHeight = 50 * 3;
```

```
let theSentence = "My name is " + "Rebecca";
```

Arrays

An array is a special type of variable. It doesn't just store one value; it stores a list of values.

You should use an array whenever you're working with a list of values that are related to each other. (ex: an array of images you want the user to click through, an array of colors to randomize a design)

Create an array and give it a name just like any other variable:

```
let theMovies = [ ];
```

2. Create an array using array literal technique:

```
let theMovies = ["Blade Runner", "Sorry to Bother You", "Groundhog Day"];
```

3. Create an array using array constructor technique:

```
let theMovies = new Array ("Blade Runner", "Sorry to Bother You", "Groundhog Day");
```

Note: values in an array do **not have to be the same data type** (could be string, number, boolean in one array).

Note2: array literal is the preferred way to create an array in JS.

Values in Arrays

Values in an array are accessed through their numbers they are assigned in the list. The number is called index and starts from 0.

```
let theMovies = ["Blade Runner", "Sorry to Bother You", "Groundhog Day"];
                  index [0]           index [1]           index [2]
```

You can check the number of items in an array using **length** property

```
let theMoviesLength = theMovies.length;
```

Changing Values in Arrays

Let's say we want to update the value of the second item (change "Sorry to Bother You" to something else)

To access the current third value, we have to call the name of the array followed by the index number for that value inside the square brackets:

```
let theMovies = ["Blade Runner", "Sorry to Bother You", "Groundhog Day"];  
theMovies[1]
```

After we select a value, we can assign a new value to it:

```
theMovies[1] = "Clueless";
```

When we log the updated array, we see updated values:

```
console.log(theMovies) ;  
["Blade Runner", "Clueless", "Groundhog Day"];
```


Adding and removing values from the array

You can add values to the array using `.push()` method:

```
let theMovies = ["Blade Runner", "Clueless", "Groundhog Day"];  
theMovies.push("The Shining");
```

You can remove values from the array using `.splice()` method:

```
let theMovies = ["Blade Runner", "Clueless", "Groundhog Day", "The Shining"];  
theMovies.splice(0,1); (will remove "Blade Runner" movie)
```

Here - 0 is an index at what position an item should be removed and 1 how many items should be removed (in this case only one movie)

For Loops

A For loop uses a counter as a condition.

It instructs code to run a specified number of times.

Good to use when the number of repetitions is known, or can be supplied by the user.

```
let theYear = ["January", "February", "March", "April", "May", "June", "July", "August", "September",  
"October", "November", "December"];
```

Keyword

Condition (counter)

```
for (var i = 0; i < theYear.length; i ++){  
    console.log(theYear[i]);  
}
```

code to execute during loop

For Loops

We can use for loops to programmatically work through arrays + get their values.

```
let theYear = ["January", "February", "March", "April", "May", "June", "July", "August", "September",  
"October", "November", "December"];
```

Keyword	Condition (counter)
<pre>for (var theIndex = 0; theIndex < theYear.length; theIndex ++){ console.log(theYear[theIndex]); }</pre>	<pre>code to execute during loop</pre>

While Loops

Good to use in applications with numeric situations and when we don't know how many times the code should run.

In other words: the loop repeats until a certain "condition" is met.

If the condition is false at the beginning of the loop, the loop is never executed.

```
let theIndex = 1;
```

```
while ( theIndex < 10 ){  
  console.log( theIndex );  
  theIndex ++;  
}
```

Do while loop

Same concept as the while loop.

Except that this loop will always execute the loop at least once (even if the condition is false).

Good to use when you are asking a question, whose answer will determine if the loop is repeated.

```
let i = 1;
```

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

Functions group a series of statements together to perform a specific task.

Prgrmmng Vocabulary

When you ask function to perform its task, you **call** a function

Some functions need to be provided with information in order to achieve a given task - pieces of information passed to a function are called **parameters**

When you write a function and expect it to provide you with an answer, the response is known as **return value**

Declaring a function

To create a function you give it a name and write statements inside to achieve a task you want it to achieve

function keyword

function name

```
function buttonClicked() {
```

code statement

```
    console.log("hello");
```

```
}
```

code block inside curly braces

Calling a function

You call a function by writing its name somewhere in the code.

```
function buttonClicked() {  
    console.log("hello");  
}
```

```
buttonClicked() // code after
```

Declaring functions with parameters

Sometimes a function needs specific information to perform its task (**parameters**)

Inside the function the parameters act as variables

parameters

```
function countTotal(itemNumber, price) {  
    return itemNumber * price;  
}
```

parameters are used like variables inside the function

Calling functions with parameters

When you call a function that has **parameters**, you need to specify the values it should take in. Those values are called **arguments**.

Arguments are written inside parentheses when you call a function + can be provided as values or as variables

```
function countTotal(itemNumber, price) {  
    return itemNumber * price;  
}
```

```
countTotal(7,15); //will return 105
```

```
(itemNumber = 7 * price = 15) countTotal(itemNumber, price);
```

Using "return" in a function

return is used to return a value to the code that called the function

The interpreter leaves the function when return is used and goes back to the statement that called it

```
function countTotal(itemNumber, price) {  
    return itemNumber * price;  
    // interpreter will skip any code found after return  
}
```

Variable Scope

Where you declare a variable affects where it can be used within your code

If it's declared inside a function, it can only be used inside that function

It's known as variable's **scope**

Local + Global Variables

When a variable is created inside a function

It's called **local variable** or **function-level variable**

When a variable is created outside a function

It's called **global variable** can be called anywhere
the in code + will take up more memory

```
var salesTax = .08
```

```
function countTotal(itemNumber, price) {  
    var theSum = itemNumber * price;  
    var theTax = theSum * salesTax;  
    var theTotal = theSum + theTax;  
    return theTotal;  
}
```

```
console.log(typeof salesTax);
```

Events

Adding Event Listeners

Each DOM object has the following function:

```
addEventListener(event name, function name);
```

event name is the string name of the JavaScript event you want to listen to

- Common ones: click, focus, blur, etc

function name is the name of the JavaScript function you want to execute when the event fires

Removing Event Listeners

Each DOM object has the following function:

```
removeEventListener(event name, function name);
```

event name is the string name of the JavaScript event you want to stop listening to

function name is the name of the JavaScript function you no longer want to execute when the event fires

defer

You can add the defer attribute onto the script tag so that the JavaScript doesn't execute until after the DOM is loaded (mdn):

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

Other old school ways of doing this (**not best practice - don't do!**):

- Put the <script> tag at the bottom of the page
- Listen for the "load" event on the window object

You will see tons of examples on the internet that do this. They are out of date. defer is wide supported and better