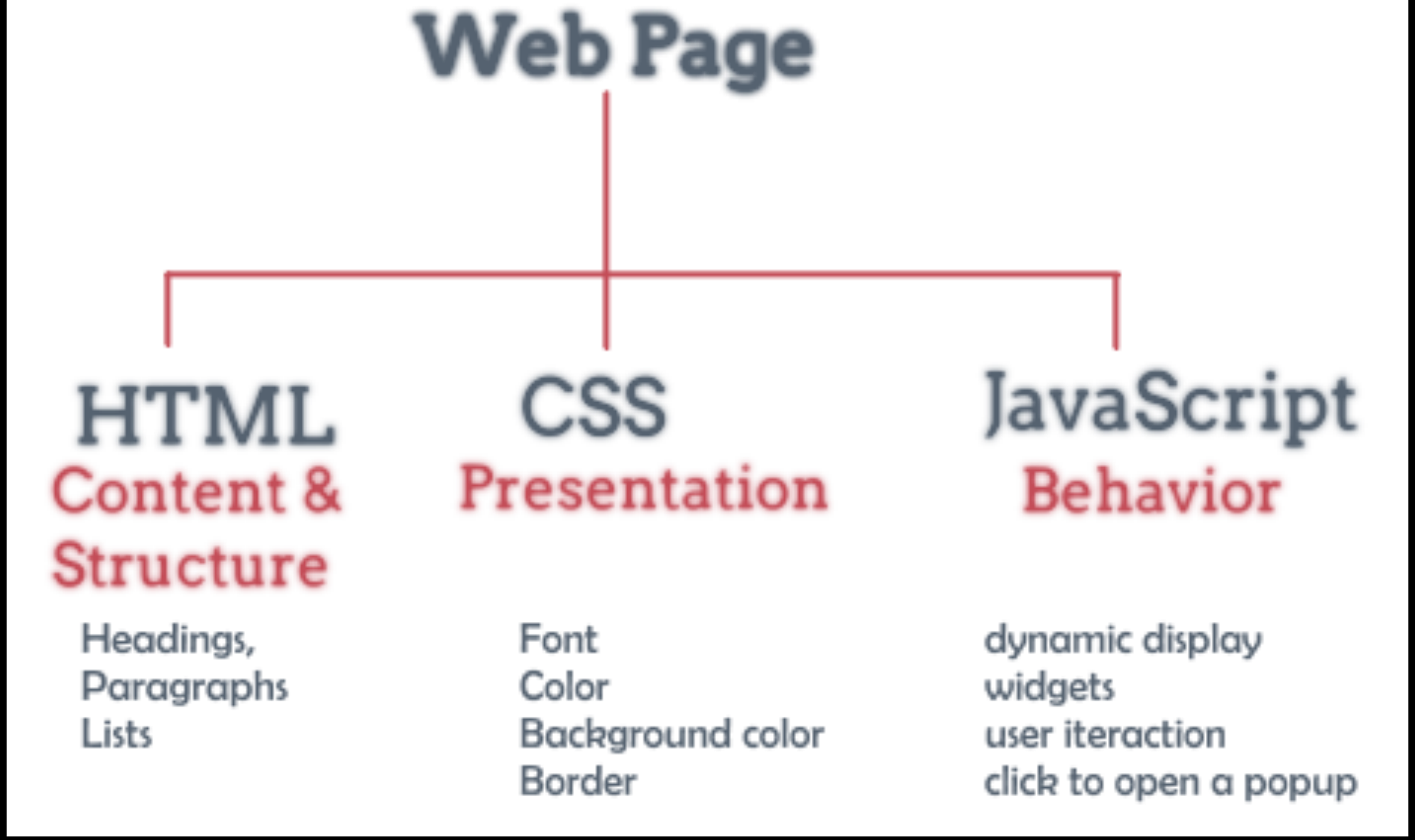
week 08: intro 2 programming

HTML CSS JS


Web Page
HTML
Content &
Structure
Headings,
Paragraphs
Lists
CSS
Presentation
Font
Color
Background color
Border
JavaScript
Behavior
dynamic display
widgets
user iteraction
click to open a popup

A Web Page **WAS:**

HTML - Hyper Text Mark Up

is a grammar for structuring web pages. It defines paragraphs, headings, data tables + media elements. HTML describes the content of the page - not how it looks.

CSS - Cascading Style Sheet

rules for styling a web page. Setting colors, typeface, and the layout. It can be used to consider the design of your page across different platforms and screen sizes.

(Web. 1.0)

HTML
CSS
JAVASCRIPT

JavaScript is a programming language specifically written to work w/ HTML and CSS. It facilitates an opportunity for more dynamic web pages with interactive media content. This led us to Web 2.0 in 2005 ish.

(Web. 2.0)

intro to JavaScript

Scripts

A script is a set of instructions that a computer can follow one-by-one. Each individual instruction or step is known as a STATEMENT.

You need to start with the big picture of what you want to achieve, and break that down into smaller steps:

1. Define the goal
2. Design the script
3. Code each step

JavaScript programs are written using the Unicode character set. **UTF-8 (8-bit Unicode Transformation Format)** is backward-compatible with **ASCII (7-bit)**. The 16-bit Unicode encoding can represent virtually every written language in common use on the planet. (This is an important feature for internationalization and is particularly important for programmers who do not speak English.)

slide shared from Ricardo (thx Ricardo!)

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.

Computers create models of the world using data.

Object-oriented programming

Objects (Things)

In computer programming, each physical thing in the world can be presented as an object.

Each object can have its own:

Properties
Events
Methods

*** (HTML elements that the browser interprets)

Properties

Each property has a **name** and a **value** + each of these name/value pairs tells you something about each individual instance of that object

*** (the values of the HTML elements)

Events

Programs are designed to do different thing when users interact with the computer in different ways. For example, clicking on a link could bring up another webpage.

Programmers choose which events they respond to. When a specific event happens, that event can be use to trigger a speck section of the code.

*** (things that the objects do: buttons click( ), windows open( ), text may be selected( ) Parenthesis signal reference methods.)

Methods

Methods represent things people need to do with object. They can retrieve or update the values of an object's properties.

They are questions + instructions that:
- Tell you something about the object (using info stored in its properties)
- Change the value of one or more of the object's properties

The code for a method can contain lots of instructions that together represent one task (ex: changeSpeed()).

When you use a method, you do not always need to know **how** it achieves its task; you just need to know **how to ask the question** and **how to interpret any answers it gives you**.

The events, methods + properties of an object all relate to each other. Events can trigger methods, and methods can retrieve or update an objects' properties.

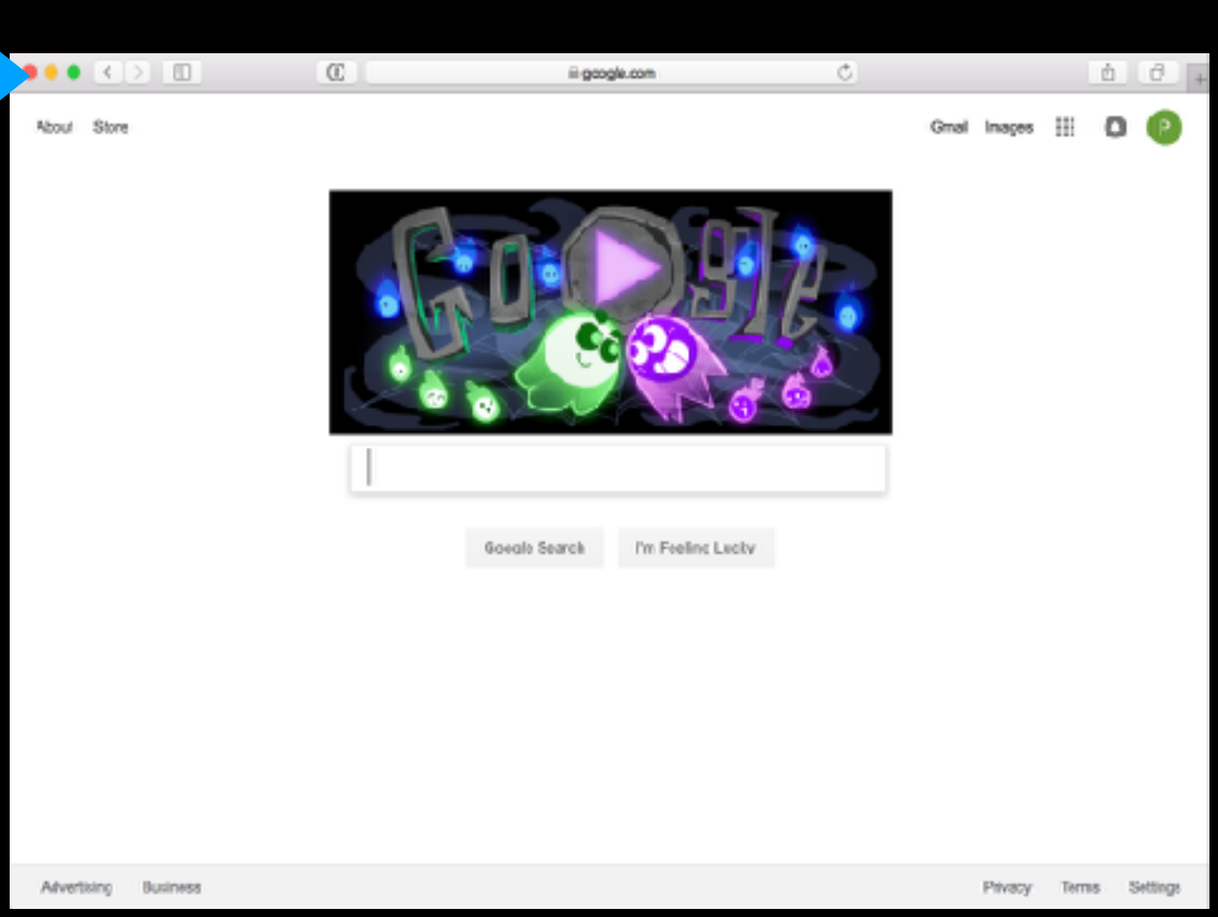
1. Event: button clicked;
2. Calls method changeBackgroundColor( );
3. Updates property backgroundColor to blue;

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)

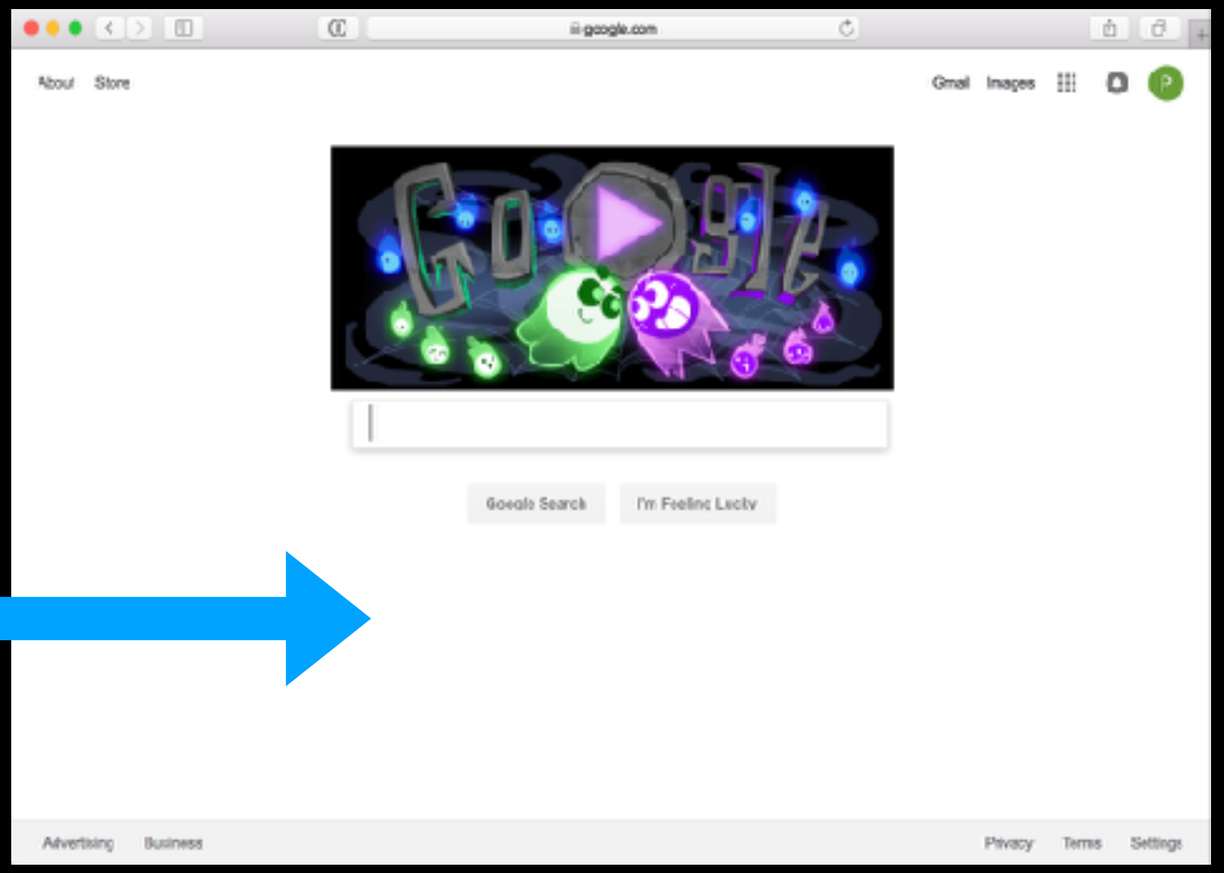
Document Object

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)

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 Variables do not have types, but the values do.

There are six primitive types (mdn):

- **Boolean** : true and false
- **Number** : everything is a double (no integers)
- **String**: in 'single' or "double-quotes"
- **Symbol:** (skipping this today)
- **Null:** null: a value meaning "this has no value"
- **Undefined:** the value of a variable with no value assigned

There are also Object types, including Array, Date, String (the object wrapper for the primitiv
type), etc.

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 theVariable1;

let theVariable2 = 'Bob';          ——> String Data
let theVariable3 = 10;             ——> Numeric Data

let theVariable1 =  false;         ——> Boolean
let theVariable2 = true;

let theVariable = [536, 3, 3354684, 325]  ——> Array

//comments look like this + are beyond helpful
```

JS Object types

```
let theArray1 = [536, 3, 3354684, 325] ;
let theMovies = [“Blade Runner”, “Sorry to Bother You”, “Groundhog Day”];
```

An array is an object that stores multiple values in a single variable.
An array may contain a string or numeric data.

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

```
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!");
 }
```

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 theSpeed = 5;
let theQuantity = 14;
let theTotal = theSpeed * theQuantity;

//maybe something changed here and the value has to change now
theSpeed = 10;

let textToShow = document.getElementById(“#thisGreatID”);
textToShow.innerHTML = "Total cost is: " + "$" + theTotal;
```

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)

let’s try these commands:

```
console.log(“Hello World”);
console.log(window.location);
console.log(document);
```

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	-	Substracts 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

Variable incrementing + decrementing

```
x++
y - -
x += 2
y =-3
```

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**.

Equality

== (is equal to)

Compares two values to see if they are the same

!= (is not equal to)

Compares two values to see if they are not the same

=== (strict equal to)

Compares two values to check that both the data and value are the same

!== (strict not equal to)

Compares two values to check that both the data and value are not the same

Equality

JavaScript's `==` and `!=` are basically broken: they do an implicit type conversion before the comparison.

```
// false
'' == '0';
// true
'' == 0;
// true
0 == '0';
// false
NaN == NaN;
// true
[''] == '';
// false
false == undefined;
// false
false == null;
// true
null == undefined;
```

Equality

Instead d of fixing `==` and `!=` ,
the ECMAScript standard kept
the existing behavior but added
`===` and `!==`

```
// false
'' === '0';
// false
'' === 0;
// false
0 === '0';
// false -??
NaN === NaN;
 // false
[''] === '';
// false
false === undefined;
 // false
false === null;
// false
null === undefined;
```

Always use `===` and `!==` and don't use `==` or `!=`

Null + Undefined

What's the difference?

null is a value representing the absence of a value, similar to null in Java and nullptr in C++.

undefined is the value given to a variable that has not been a value.

```
let x = null;
let y;
console.log(x);
console.log(y);
```

```
null                    theSketch.js:7
undefined               theSketch.js:8
```

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)**

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

code to execute during loop

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 ++;
}
```