JavaScript Basics

IS 181

Introduction

- Is the world’s most popular programming Language
- Is the programming language for the Web
- Works on the Web Browser

Why Learn Javascript?

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

What Can JS do?

- JavaScript Can Change HTML Content
- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles (CSS)
- JavaScript Can Hide HTML Elements
- JavaScript Can Show HTML Elements

Code: js_example1

JS Where To

The <script> Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Javascript can be placed in the <head> or <body> or External JS file

JS Output Possibilities

JavaScript can "**display**" data in different ways:

- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

Code: js_example2

InnerHTML

To access an HTML element, JavaScript can use the `document.getElementById(id)` method.

The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content:

LEARN JS ON ITS OWN. NO HTML/CSS.

USE JS TO MANIPULATE HTML/CSS

Primitive Types

THE BASIC BUILDING BLOCKS

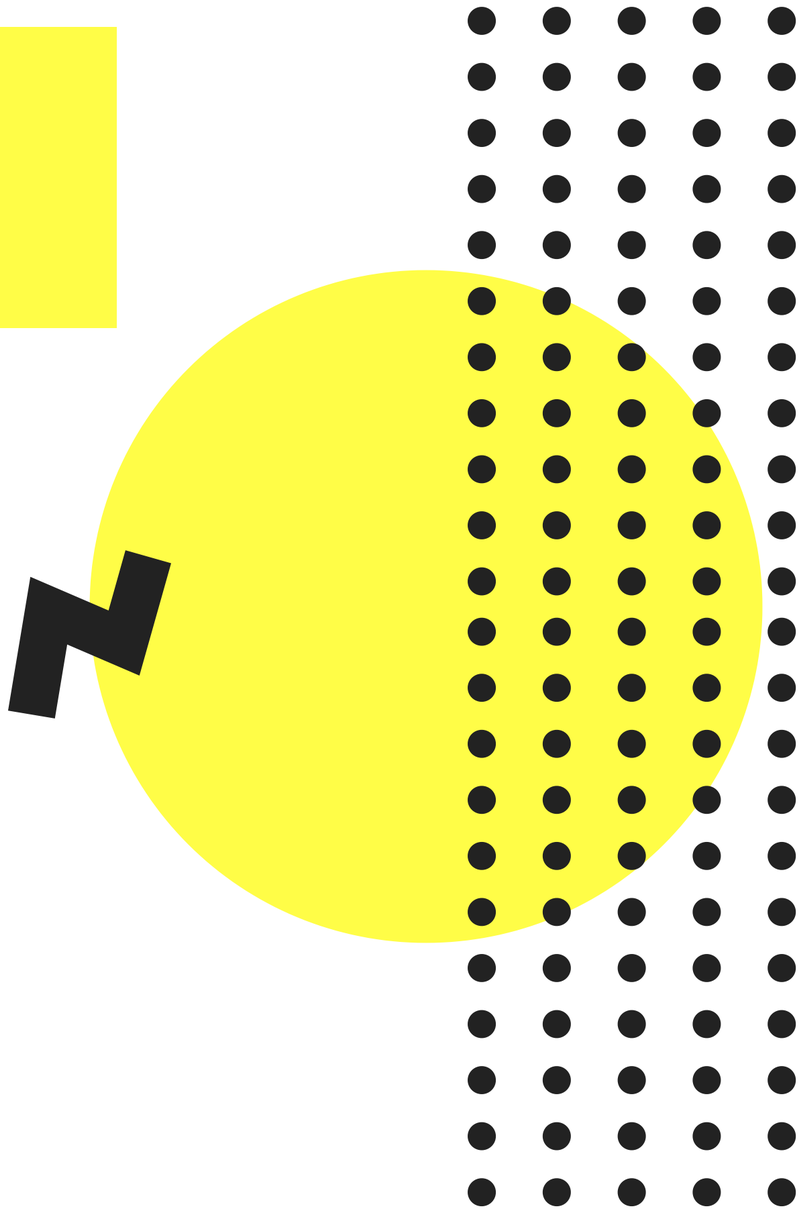
Number

String

Boolean

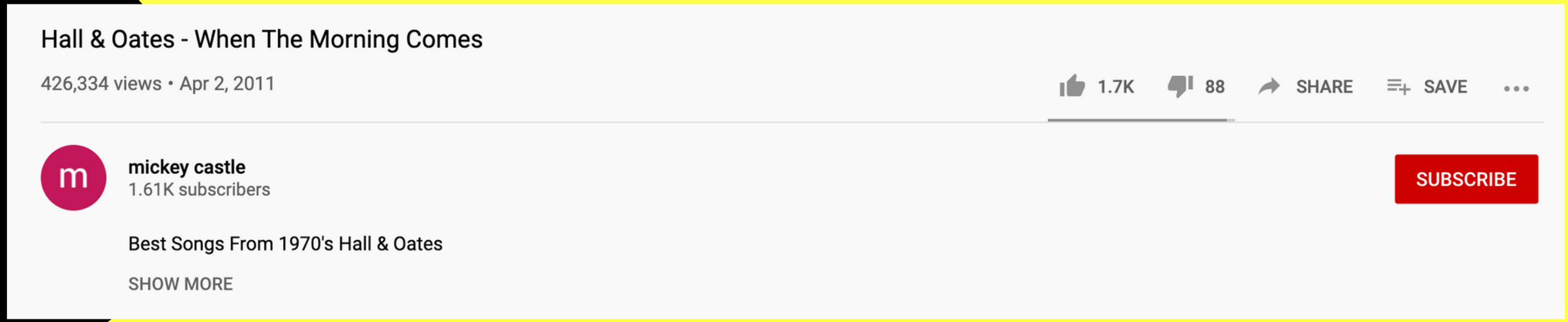
Null

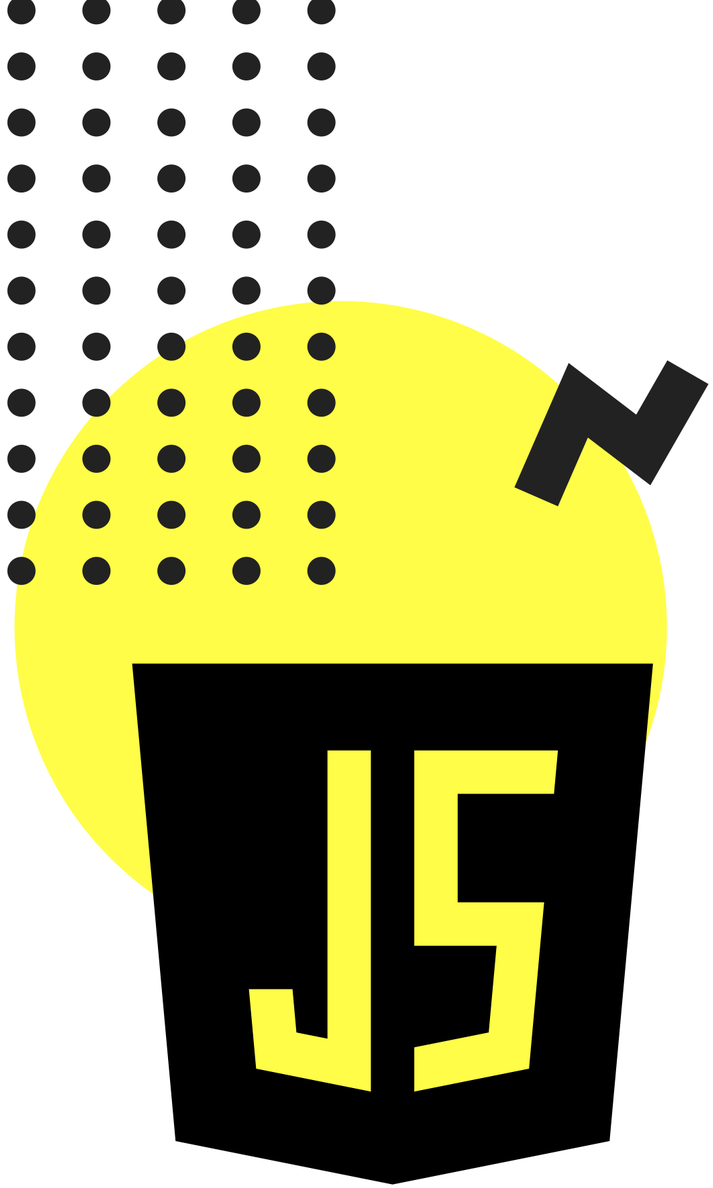
Undefined



* Technically there are two others: Symbol and BigInt

DIFFERENT DATA TYPES

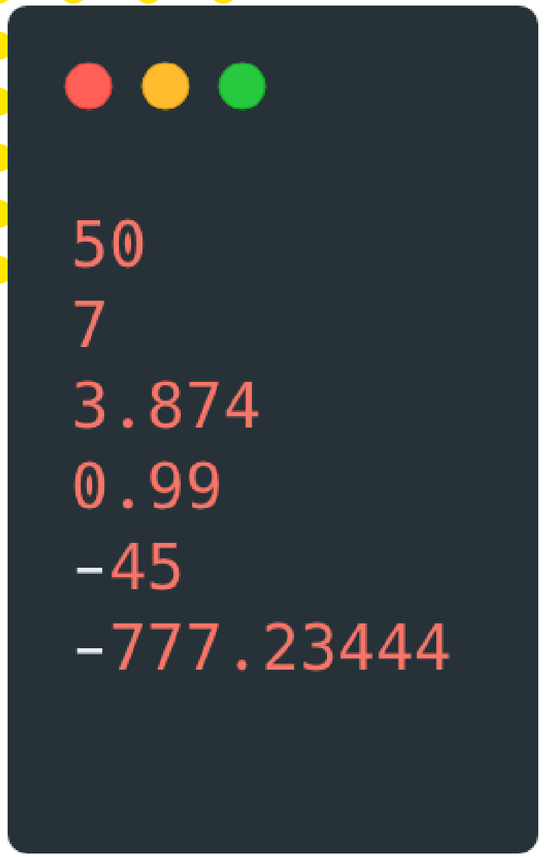




Running Code in The Console

THE EASIEST PLACE TO START

Early on, we'll run our code using the Chrome developer tools console. Then, we'll learn how to write external scripts.



Numbers

IN JAVASCRIPT

- JS has **one** number type
- Positive numbers
- Negatives numbers
- Whole numbers (integers)
- Decimal numbers

Math Operations

```
//Addition
50 + 5 //55

//Subtraction
90 - 1 //89

//Multiplication
11111 * 7 //77777

//Division
400 / 25 //16

//Modulo!!
27 % 2 //1
```

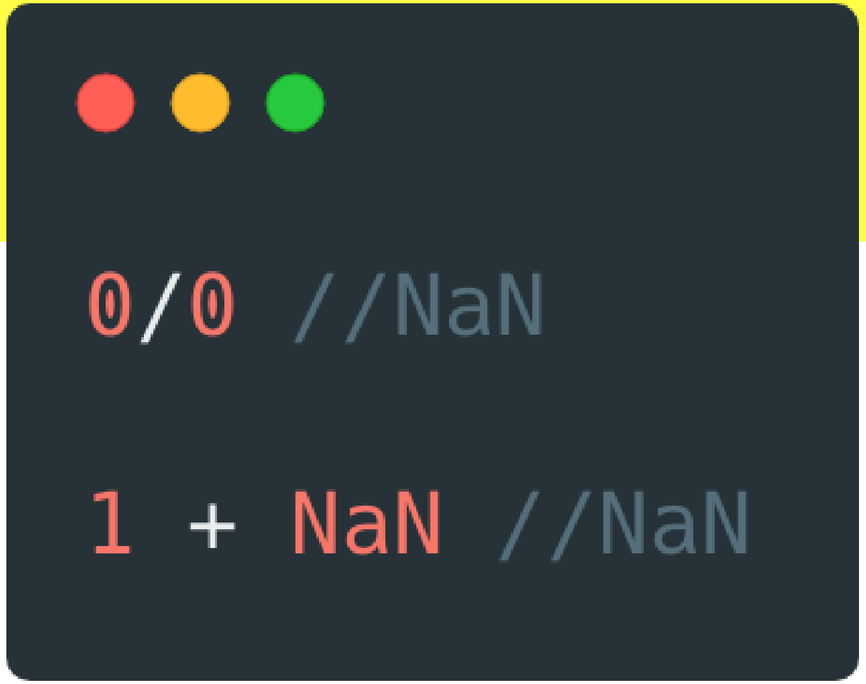
// creates a comment
(the line is ignored)

NOT A NUMBER

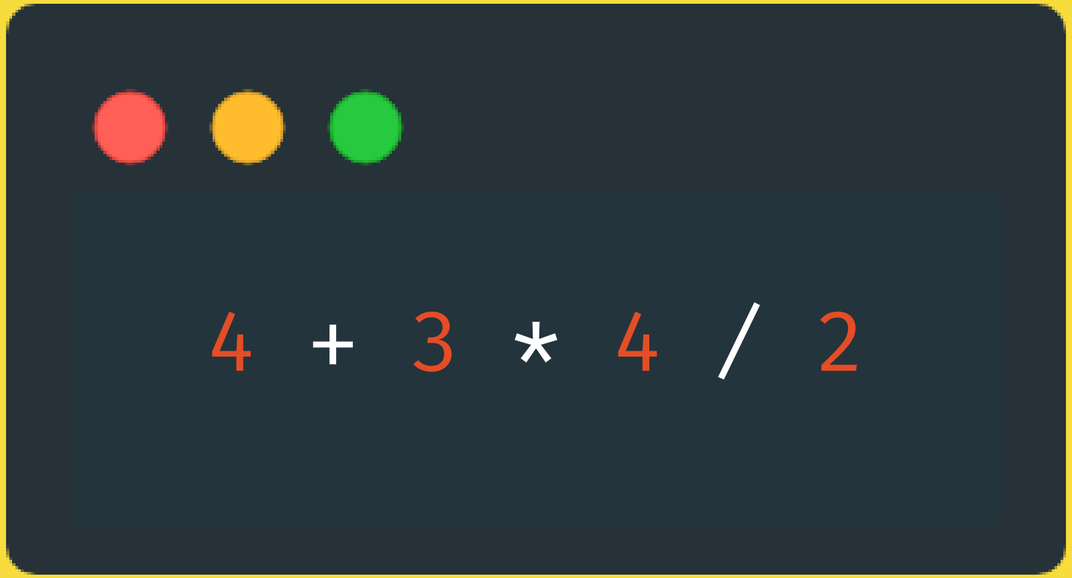
NaN

NaN is a numeric value that represents something that is...not a number

Not A Number

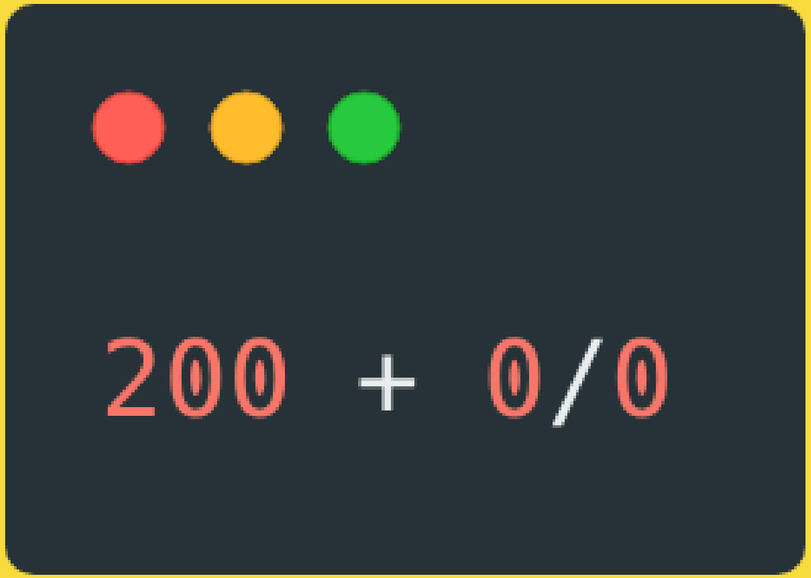


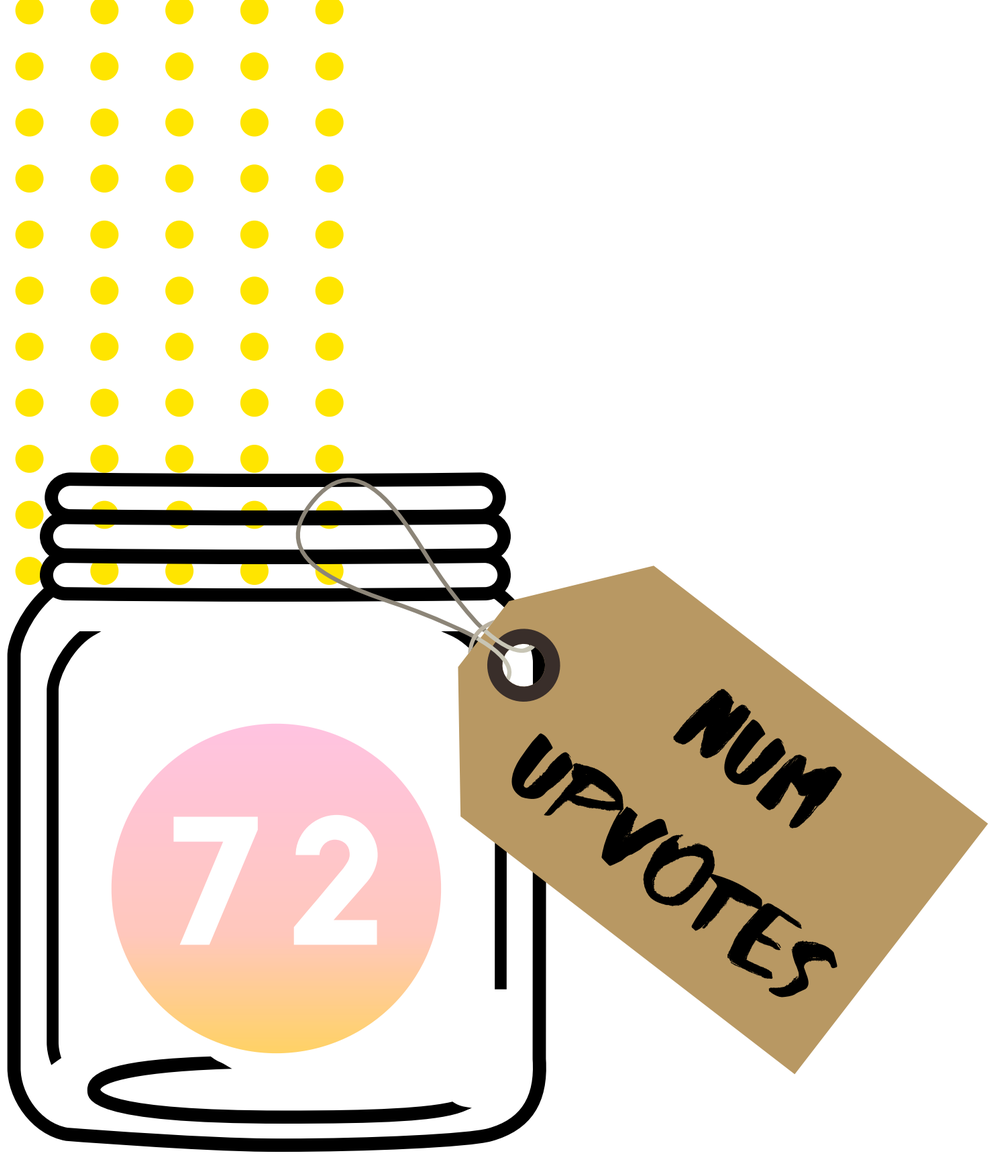
WHAT DOES THIS EVALUATE TO??



WHAT DOES THIS EVALUATE TO??

WHAT DOES THIS EVALUATE TO??





Variables

VARIABLES ARE LIKE LABELS FOR VALUES

- We can store a value and give it a name so that we can:
- Refer back to it later
- Use that value to do...stuff
- Or change it later one

BASIC SYNTAX

```
let someName = value;
```

BASIC SYNTAX

```
let year = 1985;
```

Make me a variable called "year" and give it the value of 1985

RECALL VALUES

```
let hens = 4;

let roosters = 2;

hens + roosters //6
```

RECALL VALUES

```
let hens = 4;

//A raccoon killed a hen :(
hens - 1; //3

hens; //Still 4!

//To actually change hens:
hens = hens - 1;
hens //3
```

This does not change the value stored in hens

This does!

CONST

```
const hens = 4;
hens = 20;  //ERROR!


const age = 17;
age = age + 1;  //ERROR!
```

const works just like
let, except you CANNOT
change the value

NOT ALLOWED!
YOU'RE IN TROUBLE!!
I'M TELLING MOM!!!

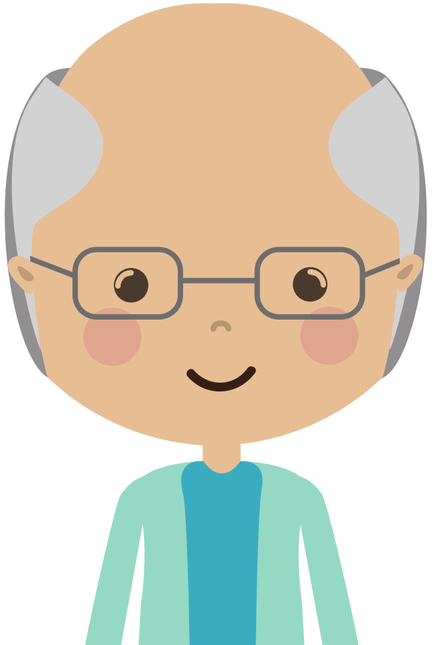
WHY USE CONST?

```
const pi = 3.14159;

const daysInWeek = 7;

const minHeightForRide  = 60;
```

Once we cover Arrays & Objects, we'll see other situations where *const* makes sense over *let*.

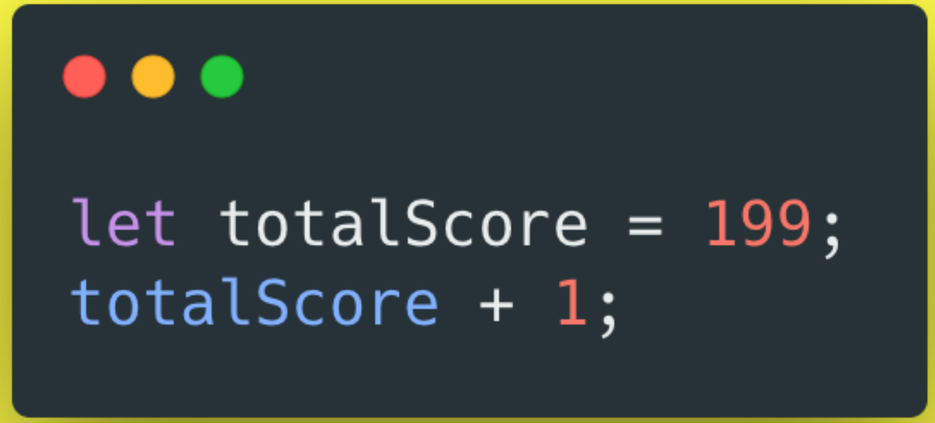
VAR

THE OLD VARIABLE KEYWORD

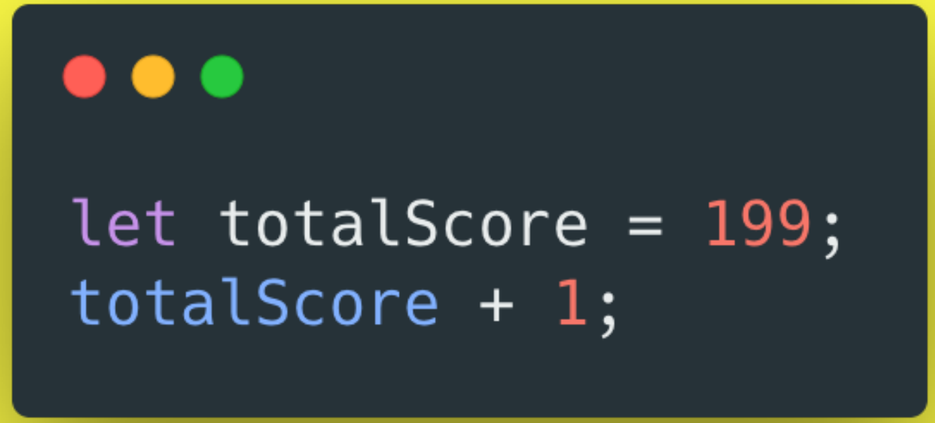


BEFORE LET & CONST, VAR WAS THE ONLY WAY OF DECLARING VARIABLES. THESE DAYS, THERE ISN'T REALLY A REASON TO USE IT.

What is the value of totalScore?

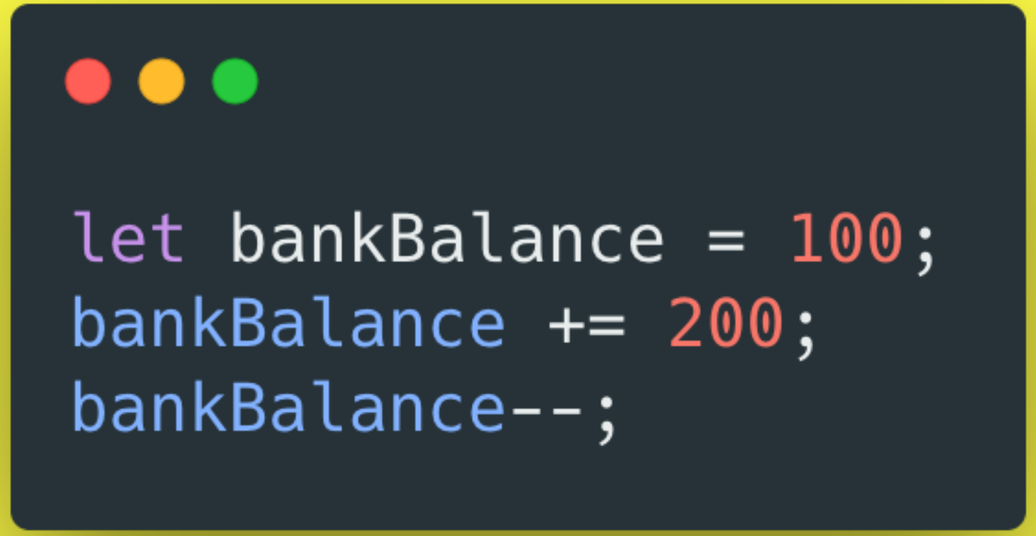


What is the value of totalScore?

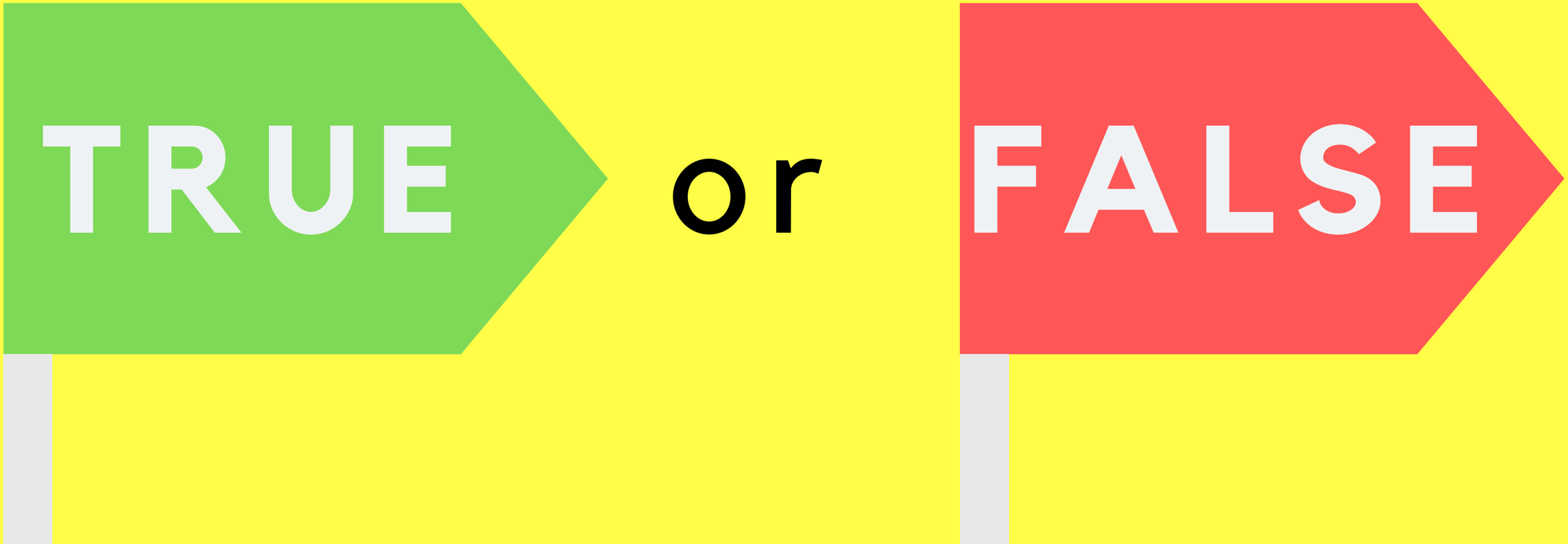


What is the value of temperature?

What is the value of bankBalance?



BOOLEANS



Booleans

TRUE OR FALSE

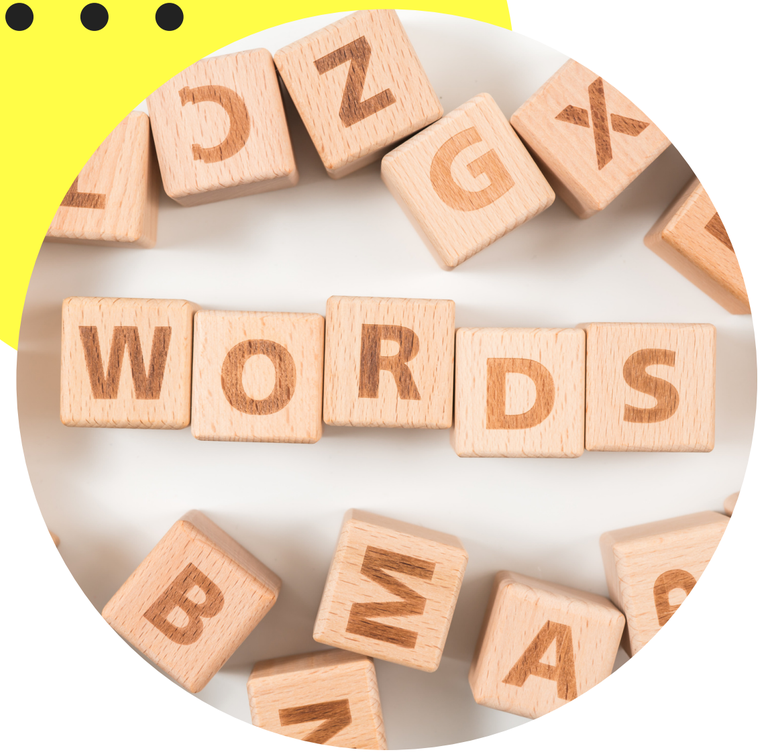
```
let isLoggedIn = true;

let gameOver = false;

const isWaterWet = true;
```

Booleans are very simple.
You have two possible options: true or false. That's it!

Variables Can Change Types

```
let numPuppies = 23; //Number
numPuppies = false;  //Now a Boolean
numPuppies = 100;    //Back to Number!
```

It doesn't really make sense to change from a number to a boolean here, but we can!



Strings

"STRINGS OF CHARACTERS"

Strings are another primitive type in JavaScript. They represent **text**, and must be wrapped in quotes.

STRINGS

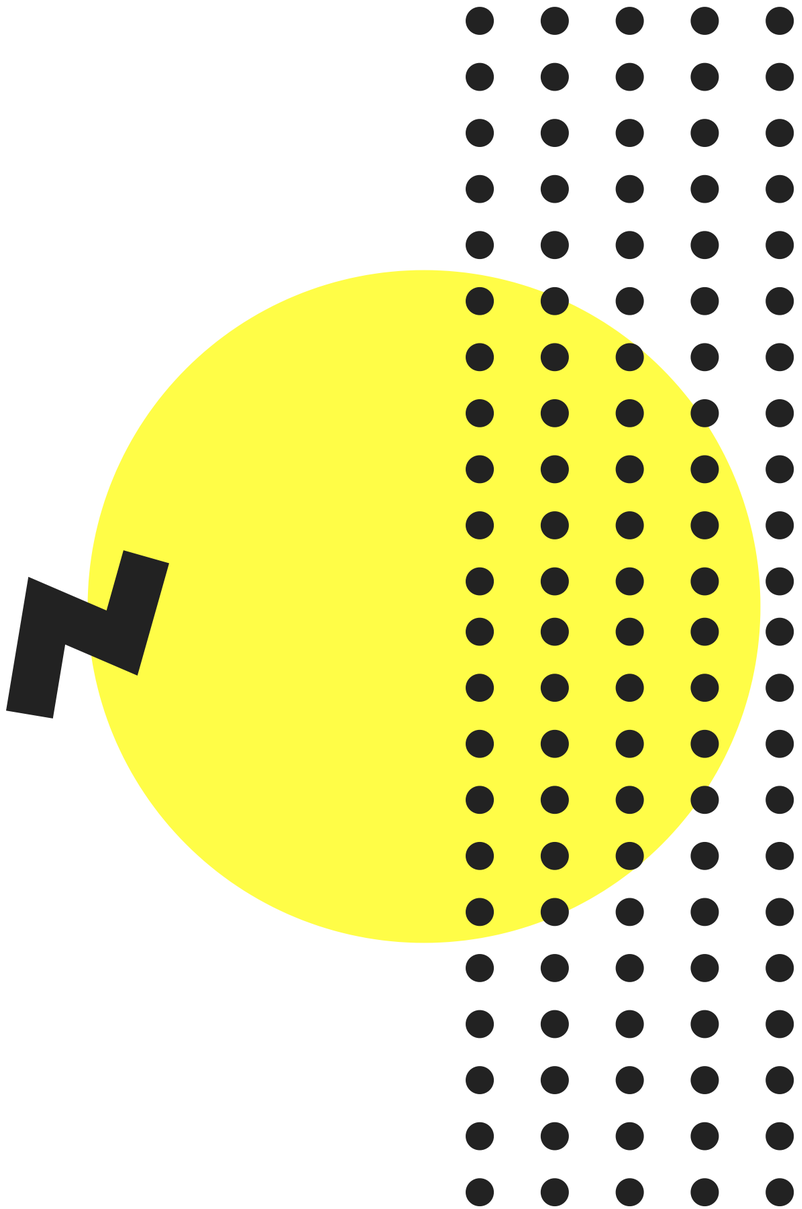
```
let firstName = "Ziggy";          Double quotes work

let msg = "Please do not feed the chimps!";

let animal = 'Dumbo Octopus';     So do single quotes

let bad = "this is wrong';     This DOES NOT work
```

It's fine to use either single or double quotes, just be consistent in your codebase.

STRINGS ARE INDEXED

Each character has a corresponding index
(a positional number)

String Methods



METHODS ARE BUILT-IN ACTIONS WE CAN PERFORM WITH INDIVIDUAL STRINGS

They help us do things like:

- Searching within a string
- Replacing part of a string
- Changing the casing of a string

```
thing.method()
```

Casing

```
let msg = 'I am king';
let yellMsg = msg.toUpperCase(); // 'I AM KING'

let angry = 'LeAvE mE aLoNe!';
angry.toLowerCase(); // 'leave me alone!'

//the value in angry is unchanged
angry; // 'LeAvE mE aLoNe!'
```

Trim

```
let greeting = '    leave me alone plz     ';

greeting.trim() // 'leave me alone plz'
```

thing.method(arg)

Some methods accept **arguments** that modify their behavior.
Think of them as inputs that we can pass in.
We pass these arguments inside of the parentheses.

indexOf

```
let tvShow = 'catdog';

tvShow.indexOf('cat'); // 0
tvShow.indexOf('dog'); // 3
tvShow.indexOf('z'); // -1 (not found)
```

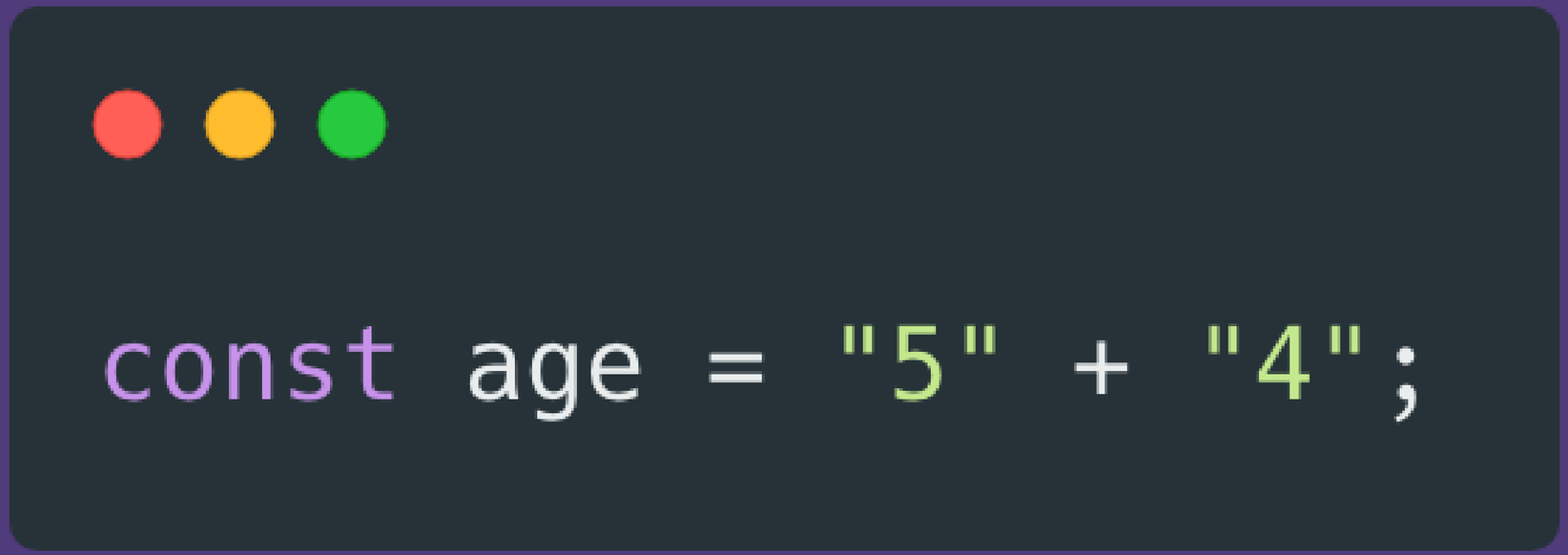
slice

```
let str = 'supercalifragilisticexpialidocious'

str.slice(0,5); //'super'

str.slice(5); // 'califragilisticexpialidocious'
```

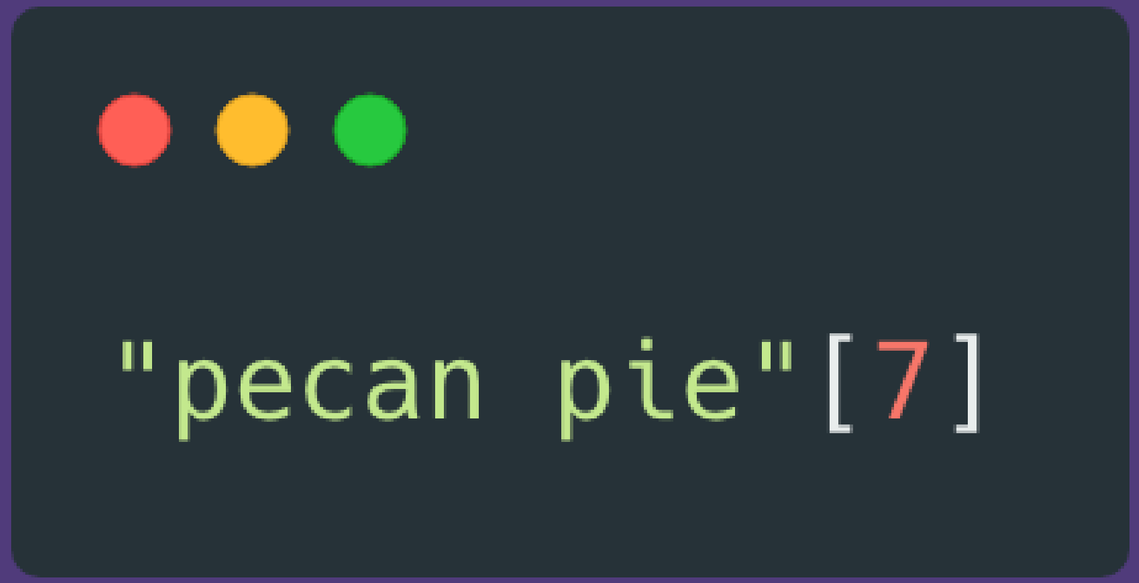
replace

```
let annoyingLaugh = 'teehee so funny! teehee!';

annoyingLaugh.replace('teehee', 'haha') // 'haha so funny! teehee!'
//Notice that it only replaces the first instance
```

WHAT IS THE VALUE OF *AGE*?

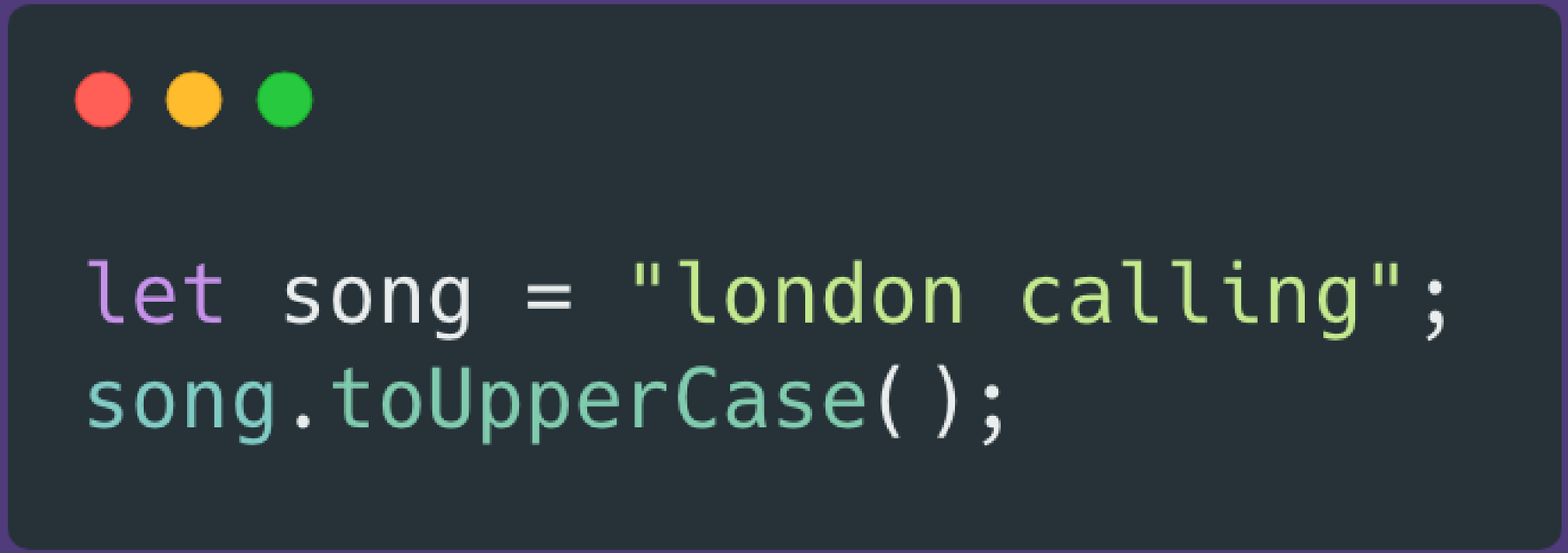


WHAT DOES THIS EVALUATE TO?



WHAT DOES THIS EVALUATE TO?

What is the value of *song*?



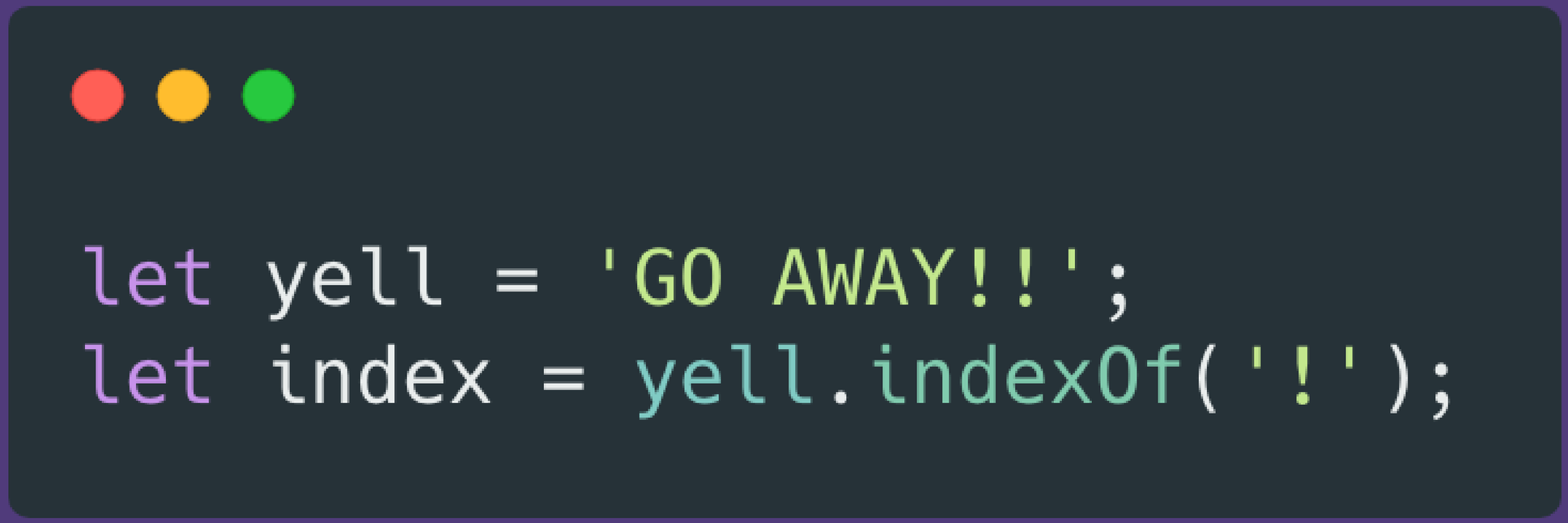
What is the value of *cleanedInput*?

```
let userInput = "  TODD@gmail.com";
let cleanedInput = userInput.trim().toLowerCase();
```

What is the value of *index*?

```
let park = 'Yellowstone';
const index = park.indexOf('Stone');
```

What is the value of *index*?



```
let yell = 'GO AWAY!!';
let index = yell.indexOf('!');
```

WHAT DOES THIS EVALUATE TO?

```
'GARBAGE!'.slice(2).replace("B",'');
```

STRING ESCAPES

- \n – newline
- \' – single quote
- \" – double quote
- \\ – backslash

Template Literals

SUPER USEFUL!

```
`I counted ${3 + 4} sheep`;  // "I counted 7 sheep"
```

TEMPLATE LITERALS ARE STRINGS THAT ALLOW EMBEDDED EXPRESSIONS, WHICH WILL BE EVALUATED AND THEN TURNED INTO A RESULTING STRING

WE USE BACK-TICKS NOT SINGLE QUOTES

`I am a template literal`

* The back-tick key is usually above the tab key

TEMPLATE LITERALS

```
let item = 'cucumbers';
let price = 1.99;
let quantity = 4;

`You bought ${quantity} ${item}, total price: $${price*quantity}`;
//"You bought 4 cucumbers, total price: $7.96"
```

NULL & UNDEFINED

- Null
  - "Intentional absence of any value"
  - Must be assigned
- Undefined
  - Variables that do not have an assigned value are undefined

NULL

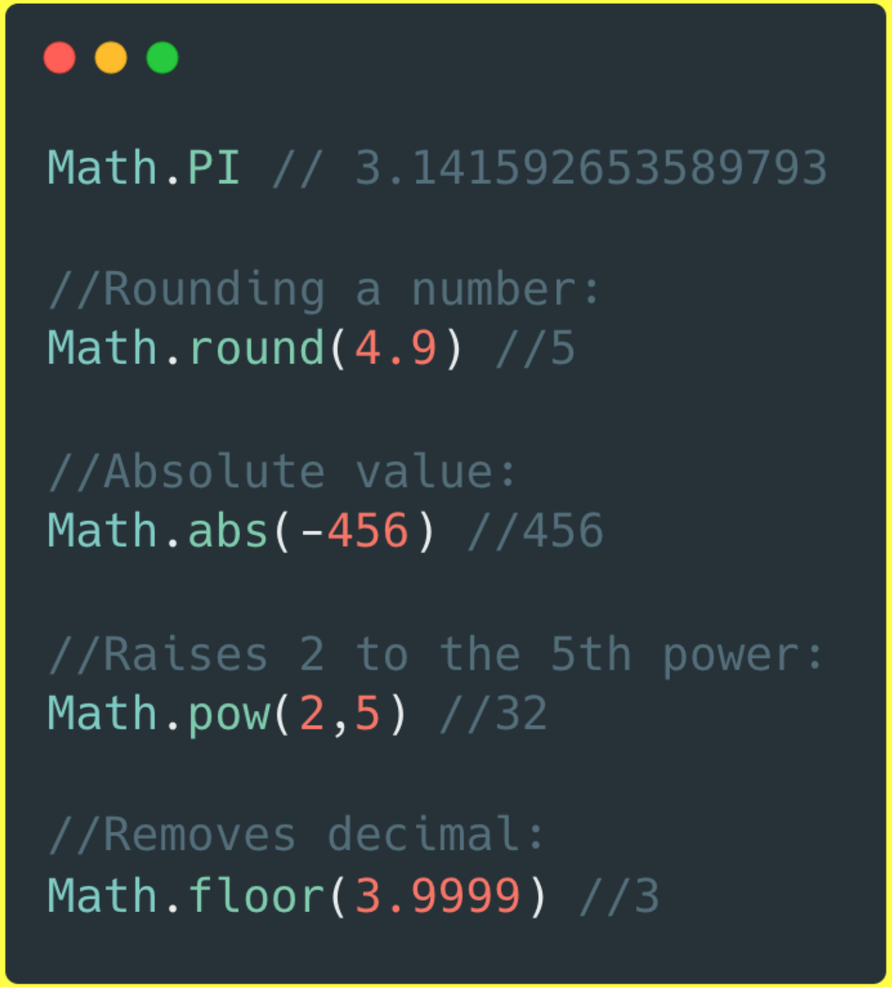
```
1 // No one is logged in yet...
2 let loggedInUser = null;  //value is explicitly nothing
3
4 // A user logs in...
5 loggedInUser = 'Alan Rickman';
```

Undefined

```
let pickles;  //We didn't assign a value
pickles; //undefined,
pickles = 'are very gross'

//Undefined also comes up in other situations:
let food = 'tacos';
food[7]; //undefined
```

MATH OBJECT

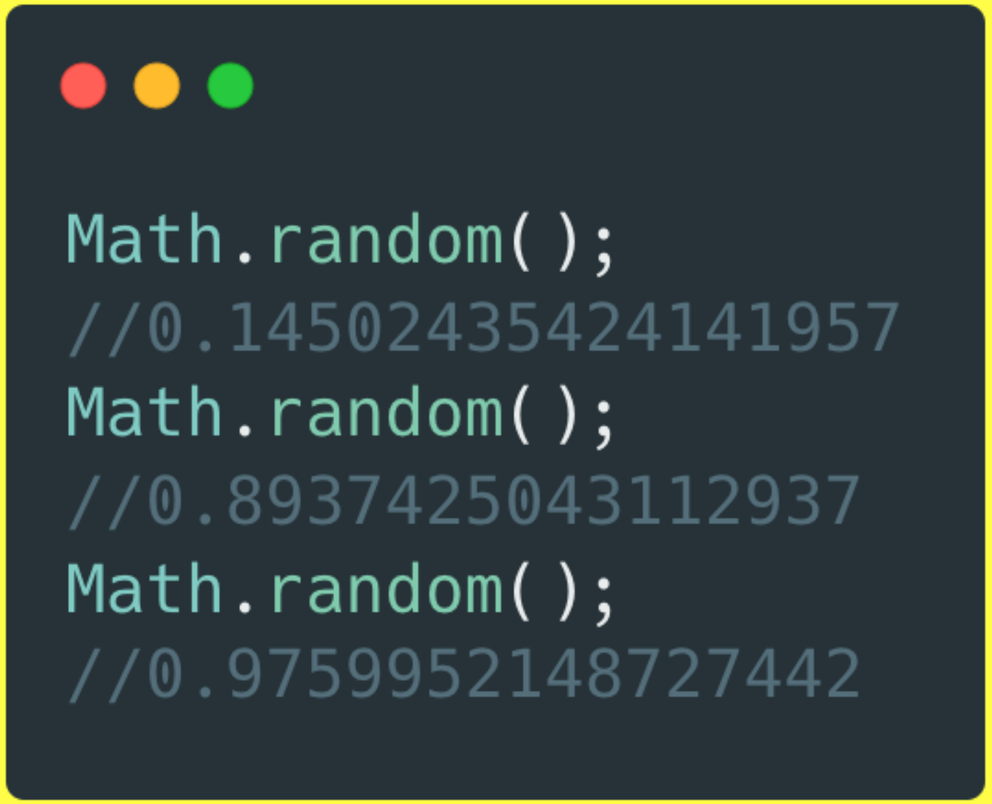
Contains properties and methods for mathematical constants and functions



```
Math.PI // 3.141592653589793

//Rounding a number:
Math.round(4.9) //5

//Absolute value:
Math.abs(-456) //456

//Raises 2 to the 5th power:
Math.pow(2,5) //32

//Removes decimal:
Math.floor(3.9999) //3
```

RANDOM NUMBERS

Math.random() gives us a random decimal between 0 and 1 (non-inclusive)



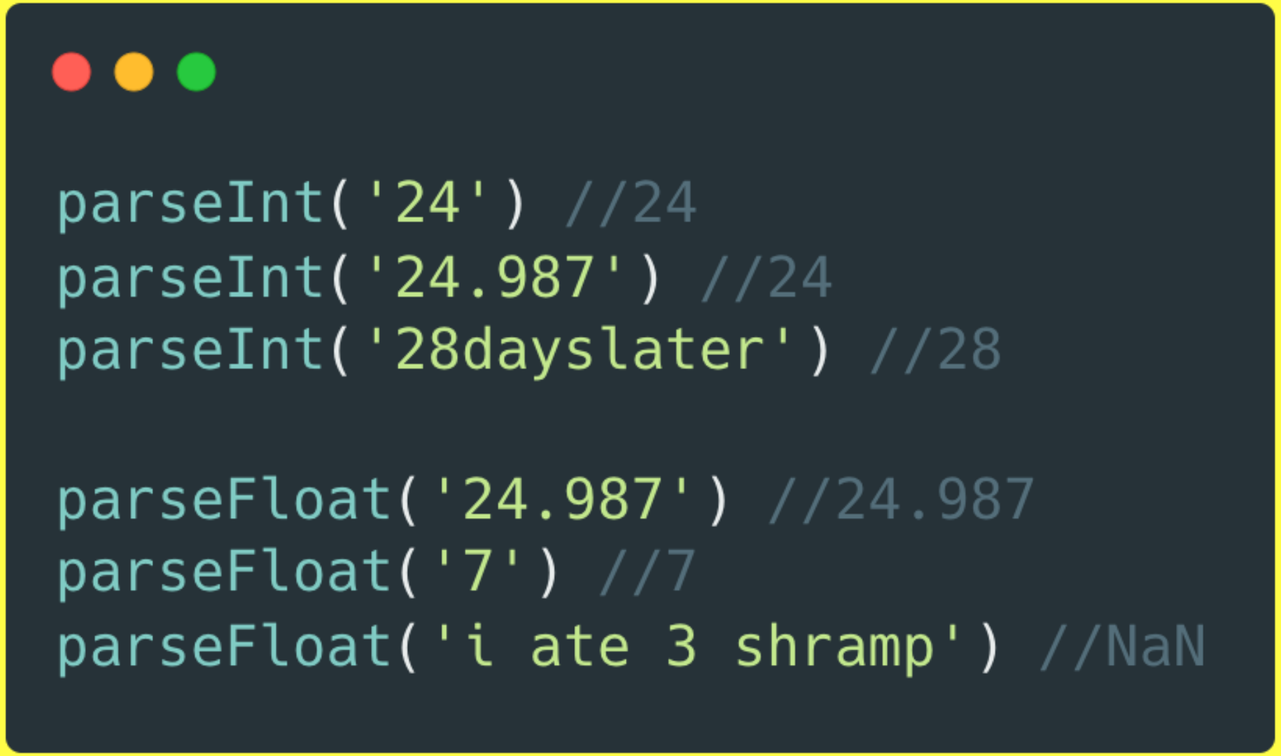
RANDOM INTEGERS

Let's generate random numbers between 1 and 10

```
const step1 = Math.random();
//0.5961104892810127
const step2 = step1 * 10
//5.961104892810127
const step3 = Math.floor(step2);
//5
const step4 = step3 + 1;
//6

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

parseInt & parseFloat

Use to parse strings into numbers, but watch out for NaN!



Web Developer Bootcamp

Boolean Logic

MAKING DECISIONS WITH JAVASCRIPT

COMPARISONS

```
> //  greater than
< //  less than
>= // greater than or equal to
<= // less than or equal to
== // equality
!= // not equal
=== // strict equality
!== // strict non-equality
```

SOME EXAMPLES

```
10 > 1;      //true
0.2 > 0.3;   //false
-10 < 0;     //true
50.5 < 5;    //false
0.5 <= 0.5; //true
99 >= 4;     //true
99 >= 99;    //true
'a' < 'b';   //true
'A' > 'a';   //false
```

Notice these all return a Boolean!

Though it's uncommon, you can compare strings. Just be careful, things get dicey when dealing with case, special characters, and accents!

== VS ===

== (double equals)

- Checks for equality of value, but not equality of type.
- It coerces both values to the same type and then compares them.
- This can lead to some unexpected results!

== WEIRDNESSS

```
5 == 5;          //true
'b' == 'c';      //false
7 == '7';        //true
0 == '';         //true
true == false; //false
0 == false;      //true
null == undefined; //true
```

TRIPLE EQUALS

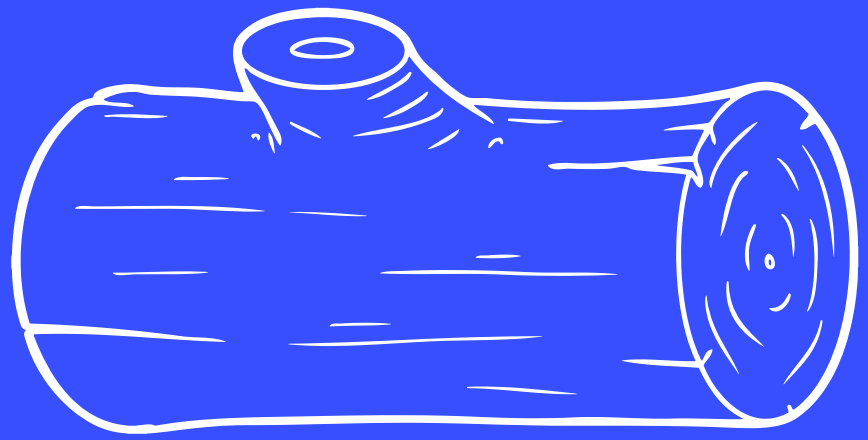
```
5 === 5; //true
1 === 2; //false
2 === '2'; //false
false === 0; //false

//Same applies for != and !==
10 != '10';  //false
10 !== '10'; //true
```

CHECKS FOR EQUALITY OF VALUE AND TYPE

console.log()

prints arguments to the console

(we need this if we're going to start working with files!)



Running Code From a File

app.js

```
//Put your code in the JS File
alert('Hello from JS!');

//Won't show up!!
"hi".toUpperCase();

//Will show up!
console.log("hi".toUpperCase());
```

Write your code in a *.js* file

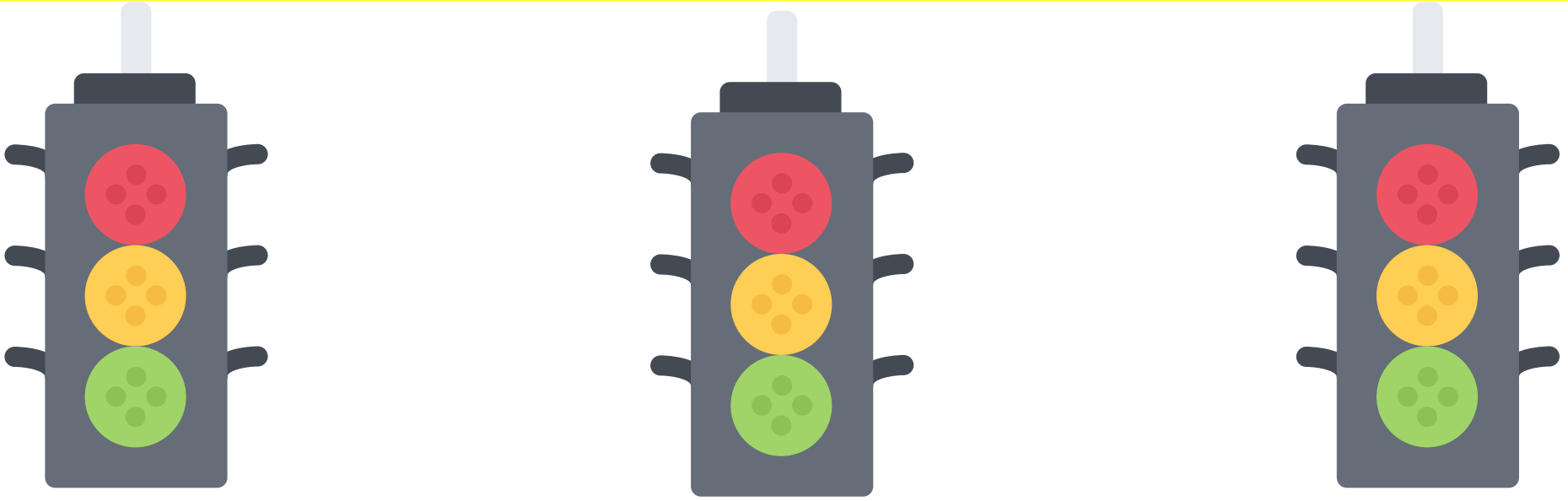
demo.html

```
<!DOCTYPE html>
<html>
<head>
  <title>JS Demo</title>
  <script src="app.js"></script>
</head>
<body>

</body>
</html>
```

Include your script in a *.html* file

Conditionals

MAKING DECISIONS WITH CODE



IF STATEMENT

Only runs code if given condition is true

```
let rating = 3;

if (rating === 3) {
  console.log("YOU ARE A SUPERSTAR!");
}
```

ELSE IF

If not the first thing, maybe this other thing??

```
let rating = 2;

if (rating === 3) {
  console.log("YOU ARE A SUPERSTAR!");
}
else if (rating === 2) {
  console.log("MEETS EXPECTATIONS");
}
```

ELSE IF

We can add multiple else ifs!

```
let rating = 1;

if (rating === 3) {
  console.log("YOU ARE A SUPERSTAR!");
}
else if (rating === 2) {
  console.log("MEETS EXPECTATIONS");
}
else if (rating === 1) {
  console.log("NEEDS IMPROVEMENT");
}
```

ELSE

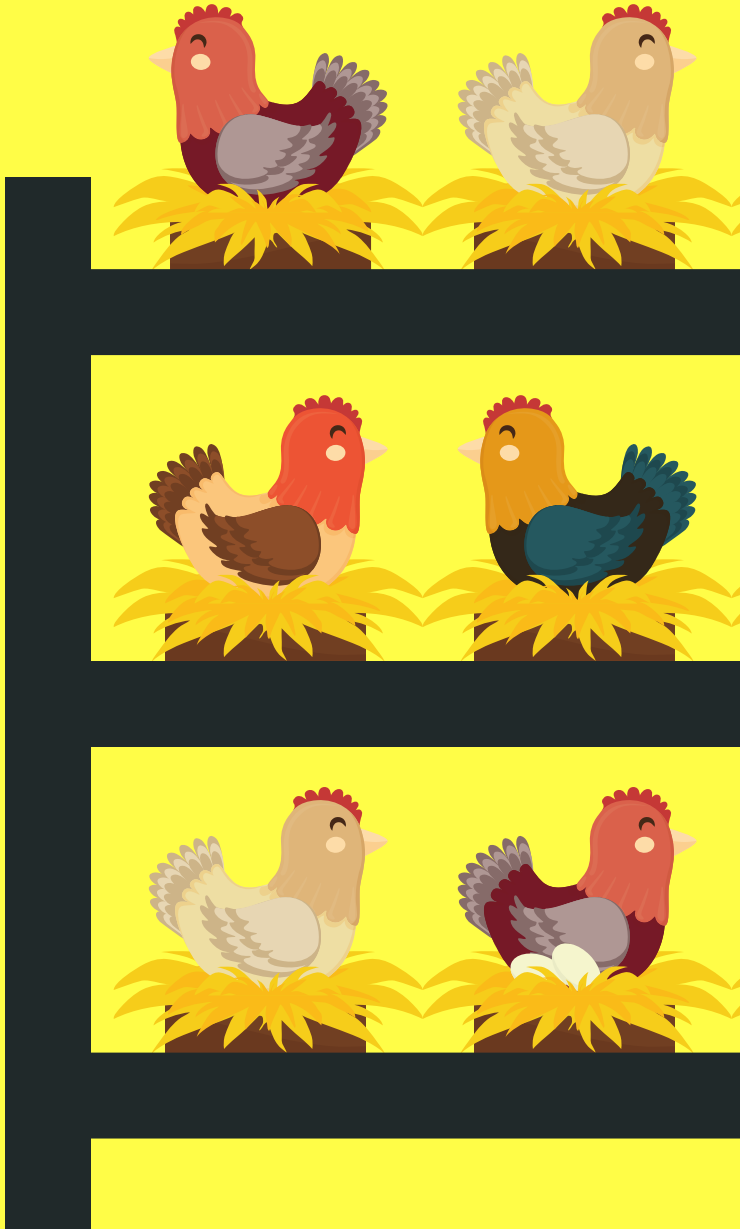
If nothing else was true, do this...

```
let rating = -99;

if (rating === 3) {
  console.log("YOU ARE A SUPERSTAR!");
}
else if (rating === 2) {
  console.log("MEETS EXPECTATIONS");
}
else if (rating === 1) {
  console.log("NEEDS IMPROVEMENT");
}
else {
  console.log("INVALID RATING!");
}
```

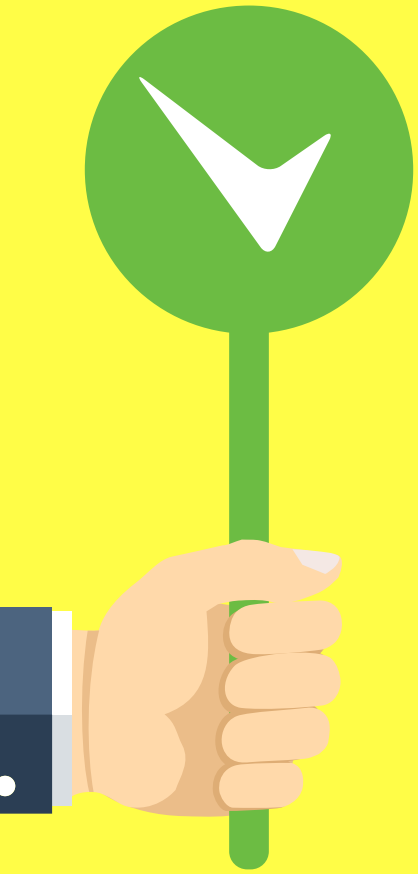
NESTING

We can nest conditionals inside conditionals

```
let password = "cat dog";
if (password.length >= 6) {
  if (password.indexOf(' ') !== -1) {
    console.log("Password cannot include spaces");
  }
  else {
    console.log("Valid password!!")
  }
}
else {
  console.log("Password too short!");
}
```



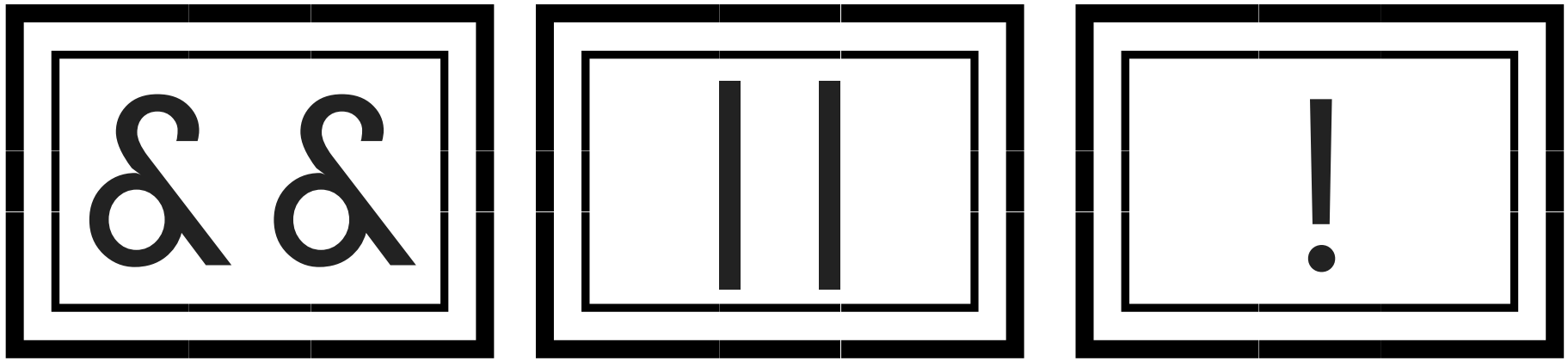
TRUTHY AND FALSY VALUES



- All JS values have an inherent truthyness or falsyness about them
- Falsy values:
  - false
  - 0
  - "" (empty string)
  - null
  - undefined
  - NaN
- Everything else is truthy!

Logical Operators

COMBINING EXPRESSIONS



AND

Both sides must be true, for the entire thing to be true

```
1 <= 4 && 'a' === 'a'; //true

9 > 10 && 9 >= 9;  //false

'abc'.length === 3 && 1+1 === 4; //false
```

AND

Both sides must be true, for the entire thing to be true

```
let password = 'taco tuesday';

if(password.length >= 6 && password.indexOf(' ') === -1){
  console.log("Valid Password!");
}
else {
  console.log("INVALID PASSWORD!");
}
```

OR

If one side is true, the entire thing is true

```
//only one side needs to be true!

1 !== 1 || 10 === 10 //true

10/2 === 5 || null //true

0 || undefined //false
```

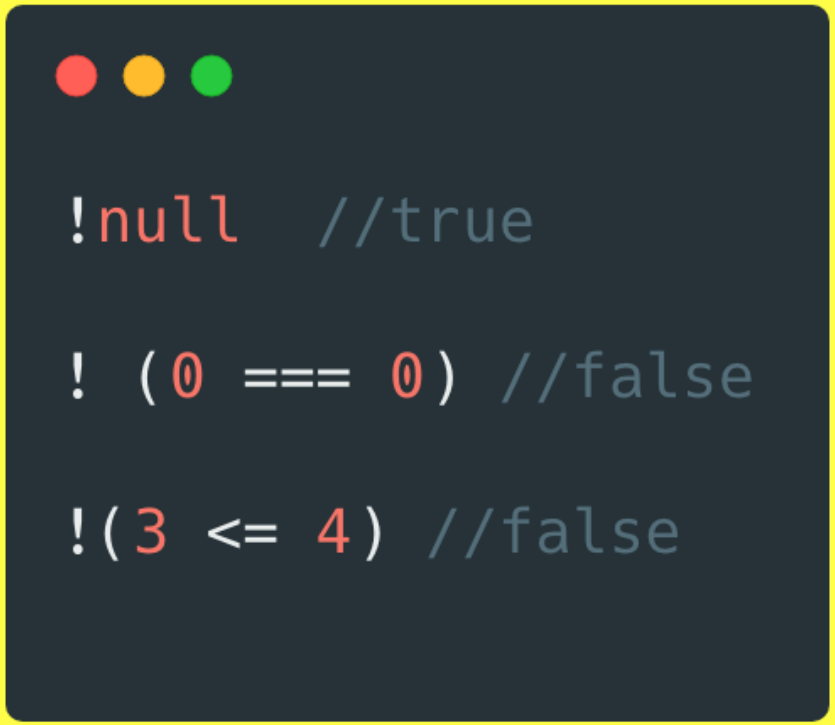
OR

If one side is true, the entire thing is true

```
let age = 76;

if(age < 6 || age >= 65){
  console.log('You get in for free!');
}
else {
  console.log('That will be $10 please');
}
```

NOT

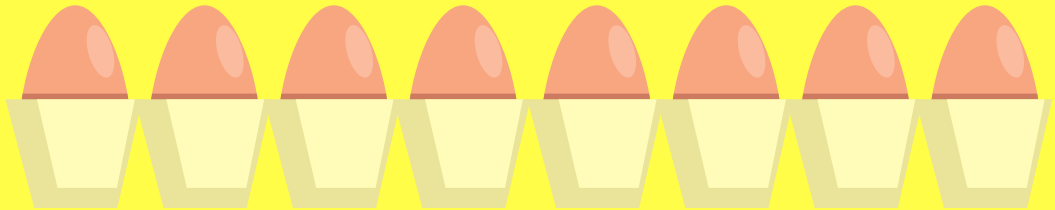
!expression returns true if expression is false



Web Developer Bootcamp

JS Arrays

OUR FIRST DATA STRUCTURE



ARRAYS

Ordered collections of values.

- List of comments on IG post
- Collection of levels in a game
- Songs in a playlist

Creating Arrays

```
// To make an empty array
let students = [];

//An array of strings
let colors = ['red', 'orange', 'yellow'];

//An array of numbers
let lottoNums = [19,22,56,12,51];

//A mixed array
let stuff = [true, 68, 'cat', null];
```

ARRAYS ARE INDEXED

Each element has a corresponding index (counting starts at 0)

Arrays Are Indexed

```
let colors = ['red', 'orange', 'yellow', 'green'];

colors.length //4

colors[0] //'red'
colors[1] //'orange'
colors[2] //'yellow'
colors[3] //'green'
colors[4] //'undefined'
```

Modifying Arrays

```
let colors = ['rad','orange','green','yellow'];

colors[0] = 'red';

colors[2] = 'yellow';
colors[3] = 'green';

colors[4]; //undefined
colors[4] = 'blue';
//["red", "orange", "yellow", "green", "blue"]
```

ARRAY METHODS

Push - add to end

Pop - remove from end

Shift - remove from start

Unshift - add to start

YOU'LL GET USED TO THE NAMES EVENTUALLY!

MORE METHODS

concat - merge arrays
includes - look for a value
indexOf - just like string.indexOf
join - creates a string from an array
reverse - reverses an array
slice - copies a portion on an array
splice - removes/replaces elements
sort - sorts an array

CONST AND ARRAYS

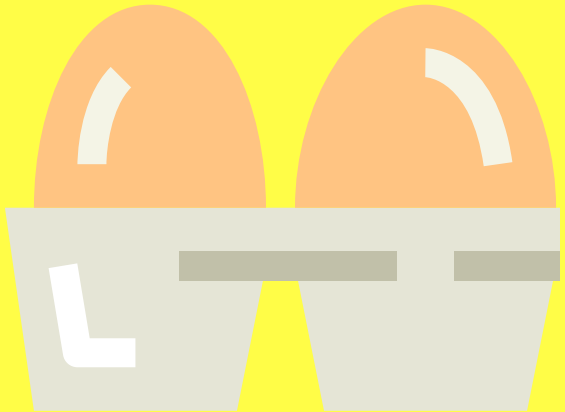
WHY DO PEOPLE USE CONST WITH ARRAYS??

THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME

```
const myEggs = ['brown', 'brown'];
```

myEggs ⟶



THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME

```
const myEggs = ['brown', 'brown'];
myEggs.push('purple');
```

myEggs ⟶

THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME

```
const myEggs = ['brown', 'brown'];
myEggs.push('purple');
myEggs[0] = 'green';
```

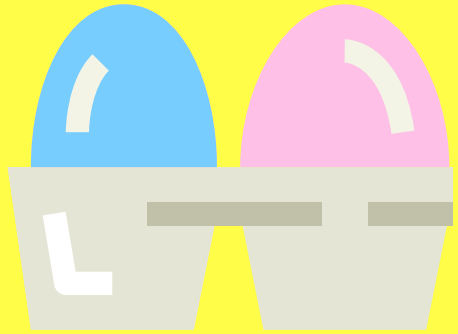
myEggs ⟶

THE VALUES CAN CHANGE

AS LONG AS THE REFERENCE REMAINS THE SAME

```
const myEggs = ['brown', 'brown'];
myEggs.push('purple');
myEggs[0] = 'green';

myEggs = ['blue', 'pink']; //NO!
```

myEggs



```
▶Uncaught TypeError: Assignment
to constant variable.
```

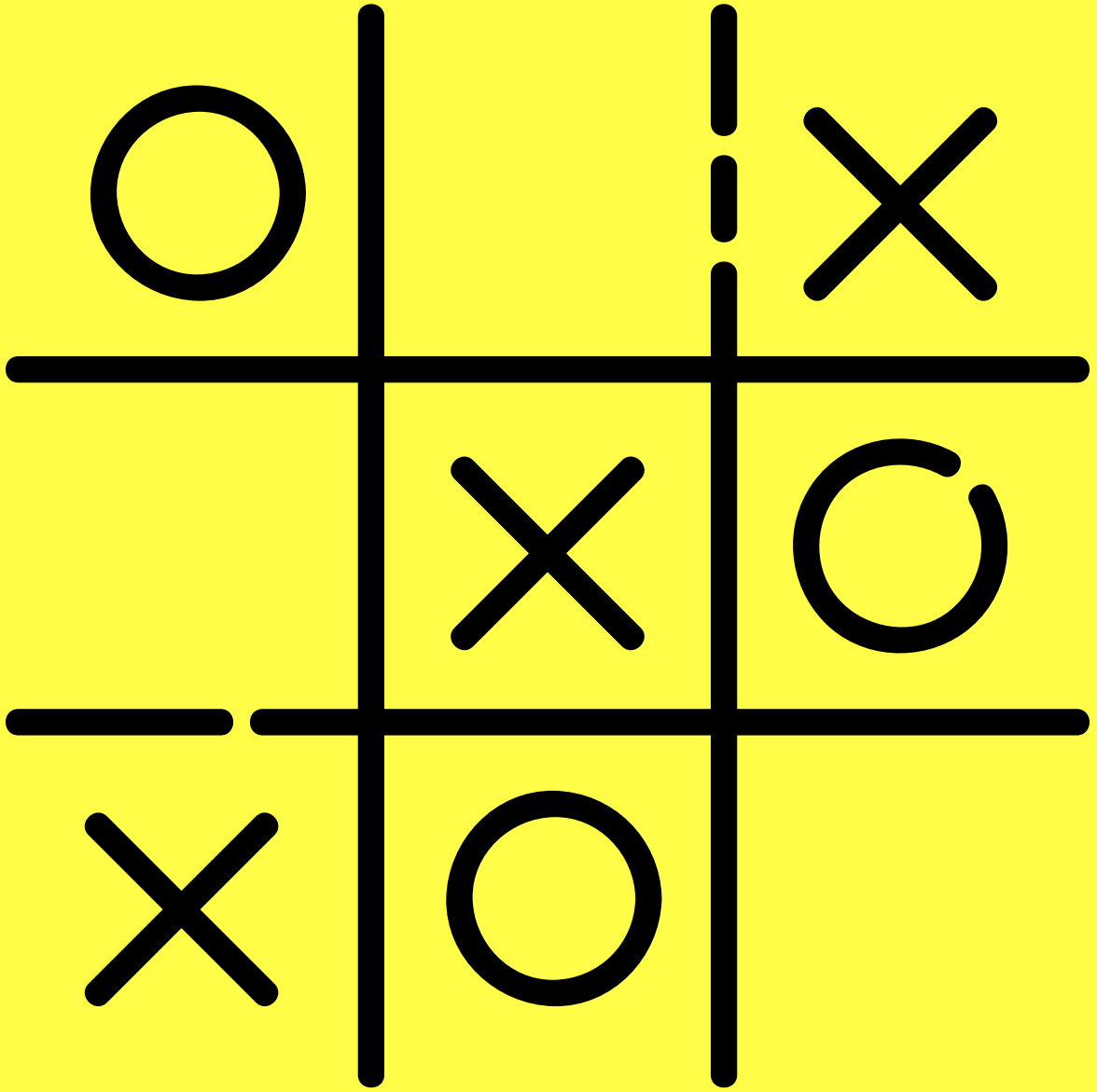
NESTED ARRAYS

We can store arrays inside other arrays!

```
const colors = [
  ['red', 'crimson'],
  ['orange', 'dark orange'],
  ['yellow', 'golden rod'],
  ['green', 'olive'],
  ['blue', 'navy blue'],
  ['purple', 'orchid']
]
```

NESTED ARRAYS

```
const board = [
  ['O', null, 'X'],
  [null,'X', 'O'],
  ['X', 'O', null]
]
```



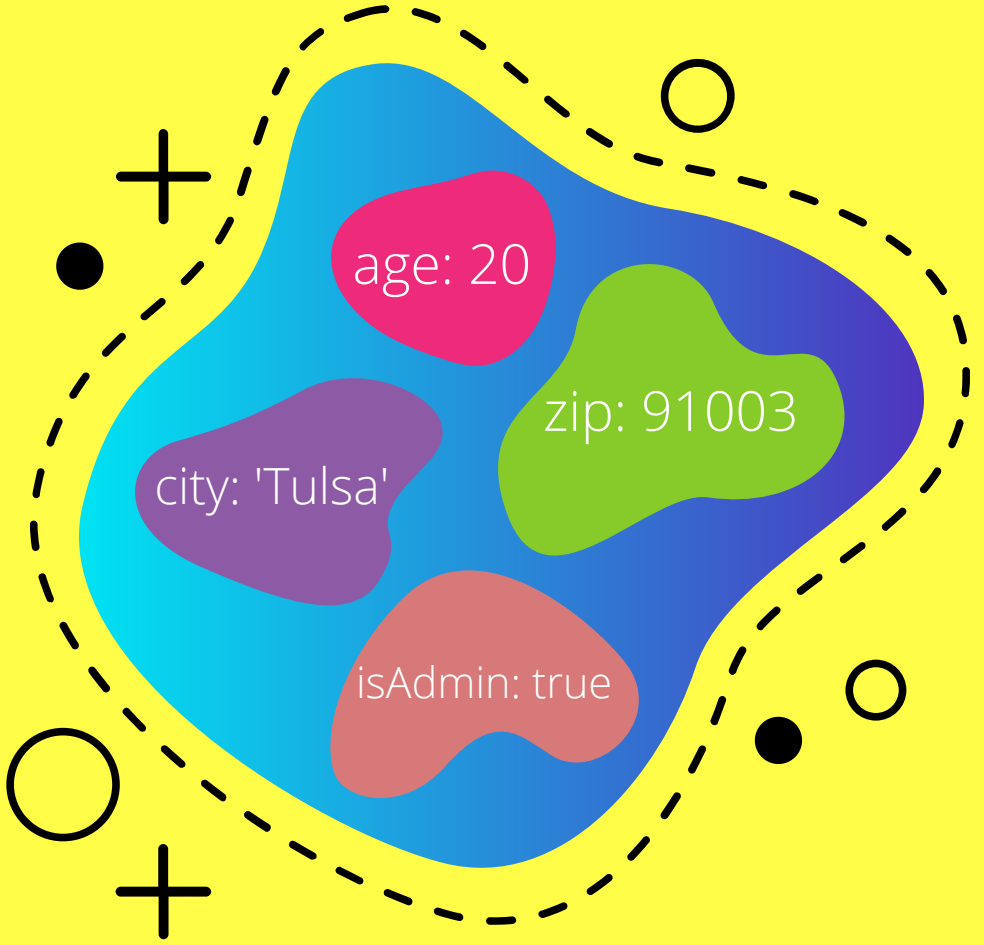
Web Developer Bootcamp

JS Objects

OUR SECOND DATA STRUCTURE!

OBJECTS

- Objects are collections of **properties**.
- Properties are a key-value pair
- Rather than accessing data using an index, we use custom keys.



HOW WOULD YOU STORE THIS?

Using an Object!

```
const fitBitData = {
    totalSteps          : 308727,
    totalMiles          : 211.7,
    avgCalorieBurn      : 5755,
    workoutsThisWeek    : '5 of 7',
    avgGoodSleep        : '2:13'
};
```

PROPERTY =

KEY

+

VALUE

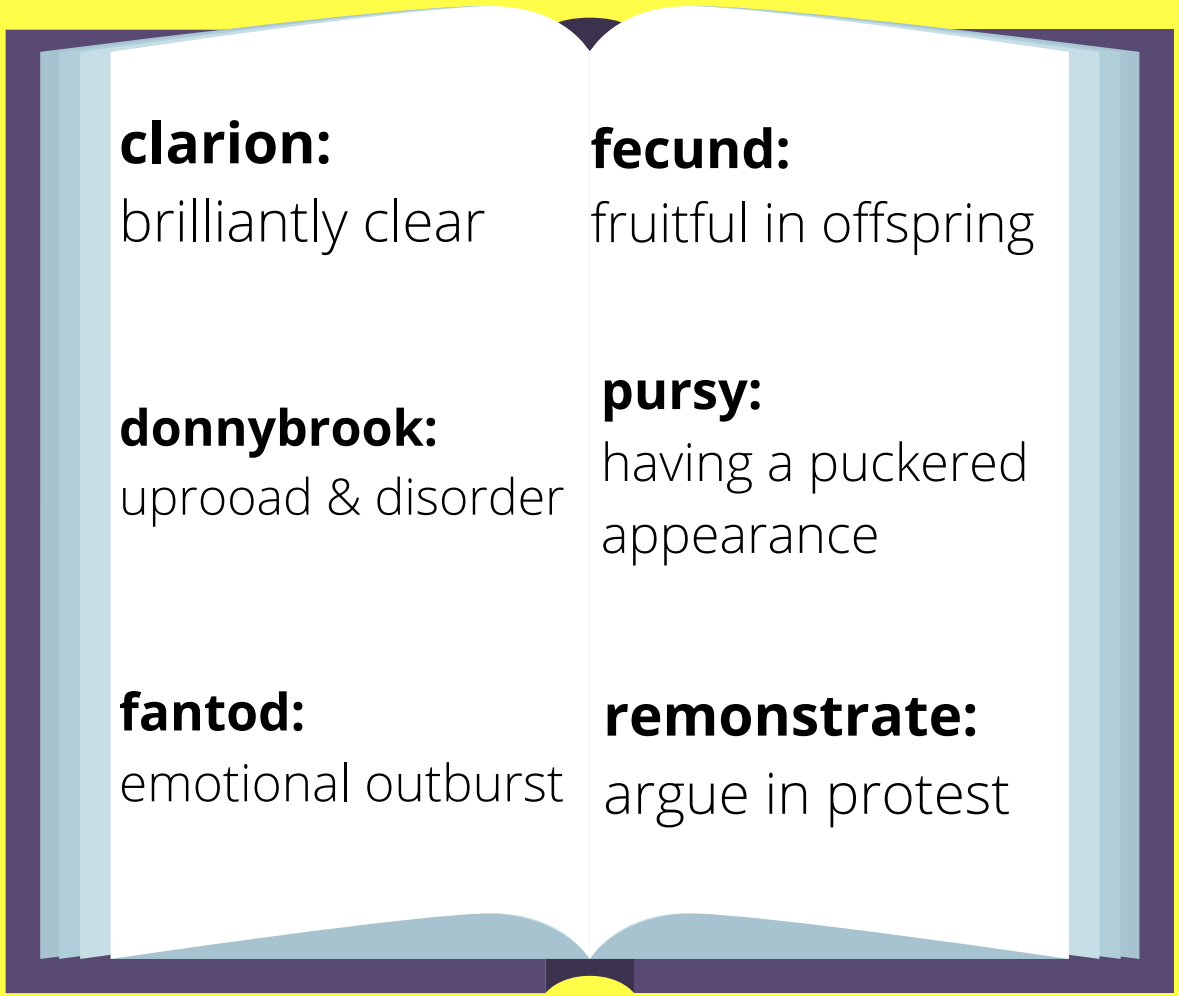
KEY-VALUE PAIRS

username: ⟶ 'crazyCatLady'

upvotes: ⟶ 7

text ⟶ 'great post!'

DICTIONARY



clarion:
brilliantly clear

donnybrook:
uprooad & disorder

fantod:
emotional outburst

fecund:
fruitful in offspring

pursy:
having a puckered appearance

remonstrate:
argue in protest

ALL TYPES WELCOME!

```
let comment = {
    username    : 'sillyGoose420',
    downVotes   : 19,
    upVotes     : 214,
    netScore    : 195,
    commentText : 'Tastes like chicken lol',
    tags: ['#hilarious', '#funny', '#silly'],
    isGilded: false
};
```

VALID KEYS

All keys are converted to strings *

* Except for Symbols, which we haven't covered yet

ACCESSING DATA

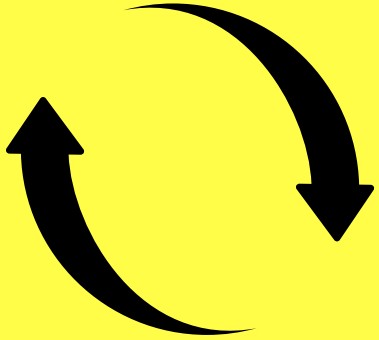
```
const palette = {
  red: '#eb4d4b',
  yellow: '#f9ca24',
  blue: '#30336b'
}
```

```
palette.red //"#eb4d4b"
```

```
palette['blue'] //"#30336b"
```

```
let color = 'yellow';
palette[color] //"#f9ca24"
```

UPDATING & ADDING PROPERTIES



```
const fitBitData = {
    totalSteps       : 308727,
    totalMiles       : 211.7,
    avgCalorieBurn   : 5755,
    workoutsThisWeek : '5 of 7',
    avgGoodSleep     : '2:13'
};
//Updating properties:
fitBitData.workoutsThisWeek = '6 of 7';
fitBitData.totalMiles += 7.5;

//Adding a new property
fitBitData.heartStillBeating = true;
```

ARRAYS + OBJECTS

```
const shoppingCart = [
  {
    product: 'Jenga Classic',
    price: 6.88,
    quantity: 1,
  },
  {
    product: 'Echo Dot',
    price: 29.99,
    quantity: 3
  },
  {
    product: 'Fire Stick',
    price: 39.99,
    quantity: 2
  }
]
```

```
const student = {
  firstName: 'David',
  lastName: 'Jones',
  strengths: ['Music', 'Art'],
  exams: {
    midterm: 92,
    final: 88
  }
}
```

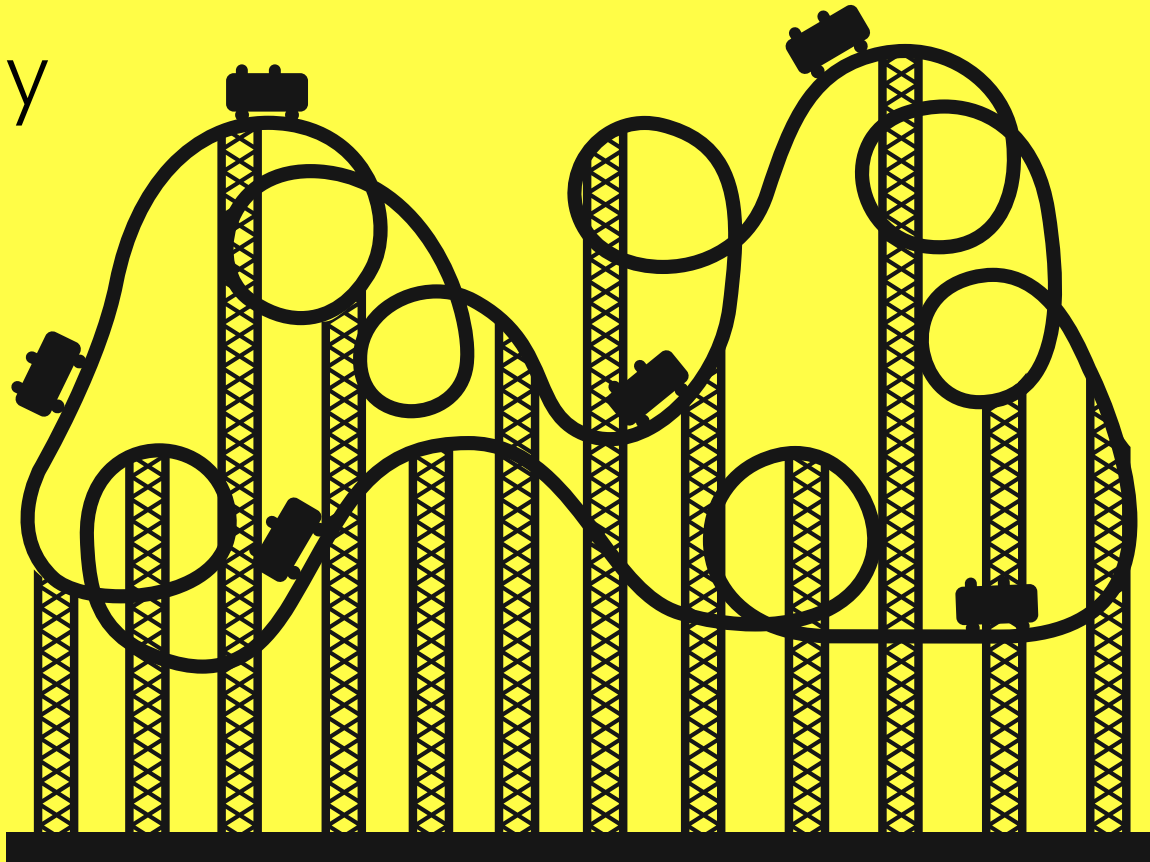
The Web Developer Bootcamp

Js Loops

REPEAT STUFF. REPEAT STUFF. REPEAT STUFF.

LOOPS

- Loops allow us to repeat code
  - "Print 'hello' 10 times
  - Sum all numbers in an array
- There are multiple types:
  - for loop
  - while loop
  - for...of loop
  - for...in loop



For Loops

Buckle Up!

For Loop Syntax

```
for (
    [initialExpression];
    [condition];
    [incrementExpression]
)
```

Our First For Loop

```
          start at 1          stop at 10     add 1 each time
for (let i = 1; i <= 10; i++) {
    console.log(i);
}
```

Another Example

```
for (let i = 50; i >= 0; i -= 10) {
    console.log(i);
}
//50
//40
//30
//20
//10
//0
```

- Start *i* at 50
- Subtract 10 each iteration
- Keep going as long as i >= 0

Infinite Loops

```
//DO NOT RUN THIS CODE!
for (let i = 20; i >= 0; i++) {
    console.log(i);
} //BADDDD!!!
```

Looping Over Arrays

```
const animals = [ 'lions', 'tigers', 'bears' ];

for (let i = 0; i < animals.length; i++) {
    console.log(i, animals[i]);
}
//0 'lions'
//1 'tigers'
//2 'bears'
```

To loop over an array, start at index 0 and continue looping to until last index (length-1)

NESTED LOOPS

```
let str = 'LOL';
for (let i = 0; i <= 4; i++) {
    console.log("Outer:", i);
    for (let j = 0; j < str.length; j++) {
        console.log('  Inner:', str[j]);
    }
}
```

```
Outer: 0
  Inner: L
  Inner: O
  Inner: L
Outer: 1
  Inner: L
  Inner: O
  Inner: L
Outer: 2
  Inner: L
  Inner: O
  Inner: L
Outer: 3
  Inner: L
  Inner: O
  Inner: L
Outer: 4
  Inner: L
  Inner: O
  Inner: L
```

While Loops

```
let num = 0;
while (num < 10) {
    console.log(num);
    num++;
}
```

```
0
1
2
3
4
```

While loops continue running as long as the test condition is true.

A Common Pattern

```
let targetNum = Math.floor(Math.random() * 10);
let guess = Math.floor(Math.random() * 10);

while (guess !== targetNum) {
    console.log(`Guessed ${guess}...Incorrect!`);
    guess = Math.floor(Math.random() * 10);
}
console.log(`CORRECT! Guessed: ${guess} & target was: ${targetNum}`);
```

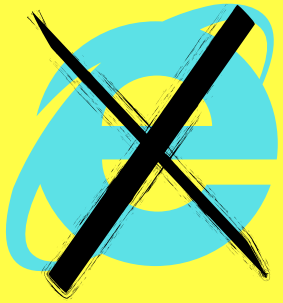
The Break Keyword

```
let targetNum = Math.floor(Math.random() * 10);
let guess = Math.floor(Math.random() * 10);

while (true) {
    guess = Math.floor(Math.random() * 10);
    if (guess === targetNum) {
        console.log(`CORRECT! Guessed: ${guess} & target was: ${targetNum}`);
        break;
    }
    else {
        console.log(`Guessed ${guess}...Incorrect!`);
    }
}
```

FOR...OF

A nice and easy way of iterating over arrays

(or other iterable objects)



No Internet Explorer Support

For...Of

```
for (variable of iterable) {
  statement
}
```

An Example

```
let subreddits = [ 'soccer', 'popheads', 'cringe', 'books' ];
for (let sub of subreddits) {
    //Do this for every item in subreddits array:
    console.log(`${sub} - www.reddit.com/r/${sub}`);
}
```

Nested For...Of

```
const magicSquare = [
  [ 2, 7, 6 ],
  [ 9, 5, 1 ],
  [ 4, 3, 8 ]
];

for (let row of magicSquare) {
    let sum = 0;
    for (let num of row) {
        sum += num;
    }
    console.log(`Row of ${row} sums to ${sum}`);
}
```

Web Developer Bootcamp

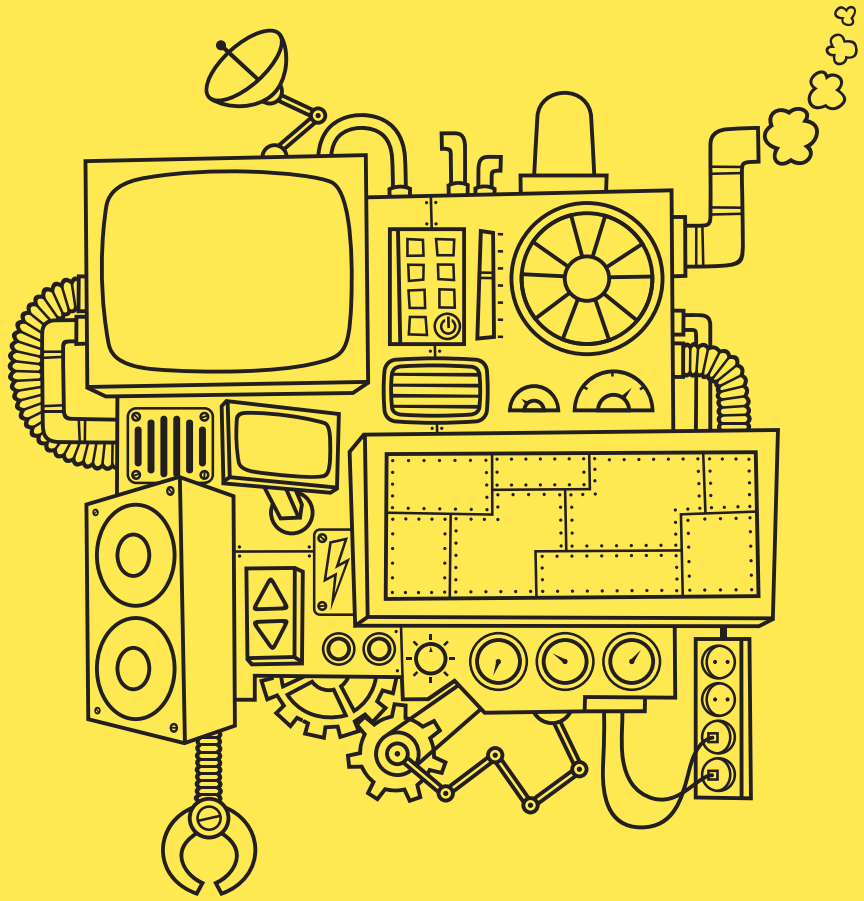
JavaScript Functions

THE LAST "BIG" TOPIC!

FUNCTIONS

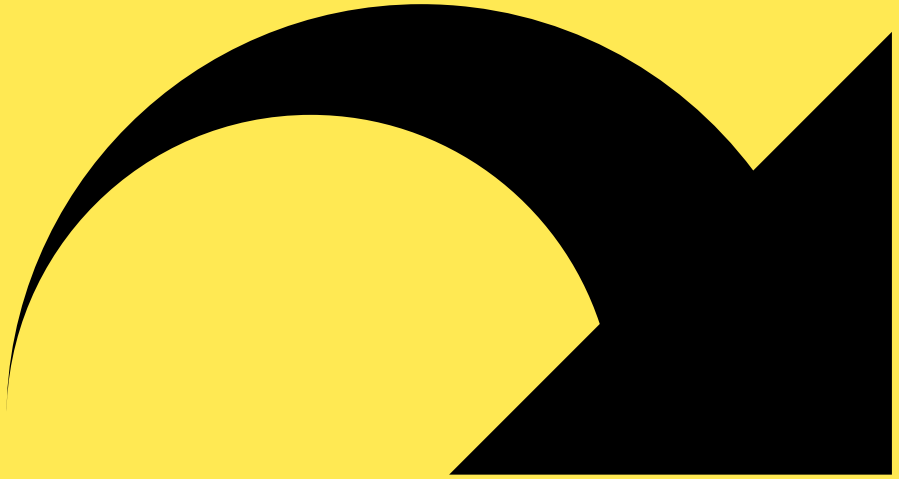
Reusable procedures

- Functions allow us to write reusable, modular code
- We define a "chunk" of code that we can then execute at a later point.
- We use them ALL THE TIME



2 STEP PROCESS

DEFINE

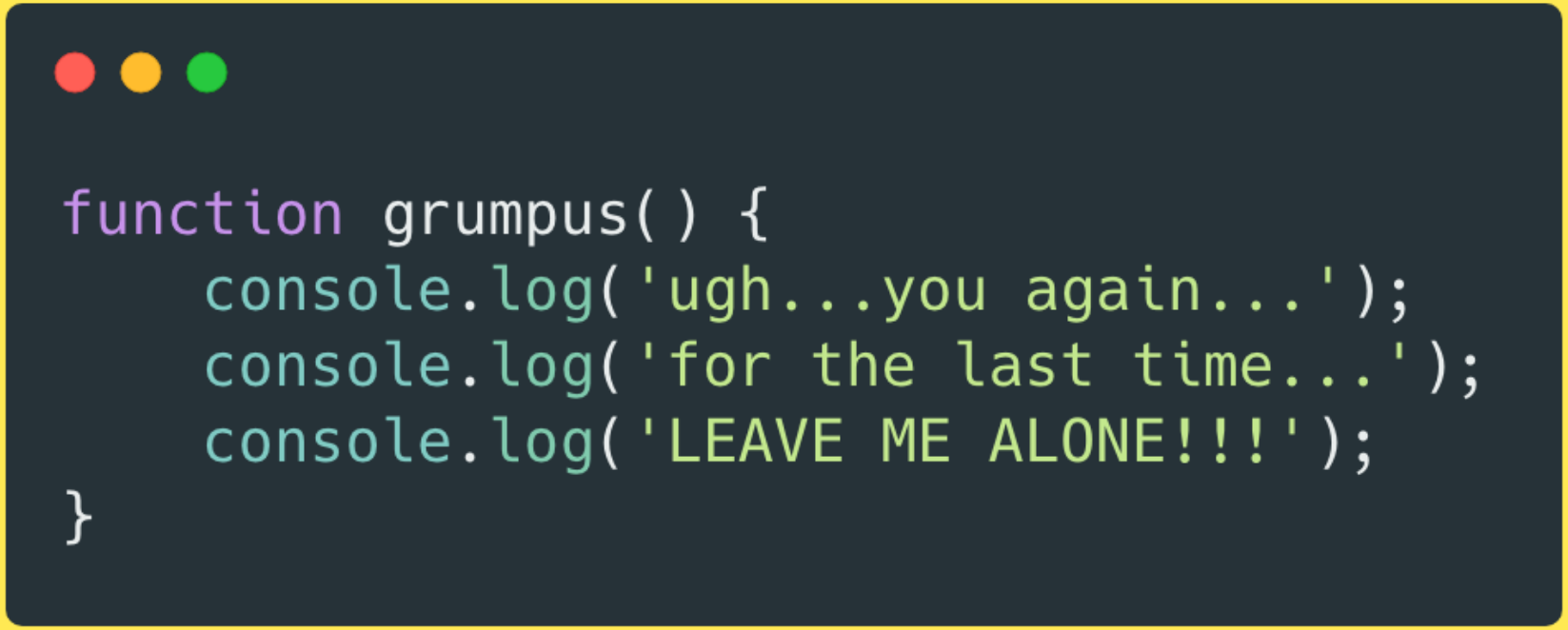


RUN

DEFINE

```
function funcName(){
  //do something
}
```

DEFINE



```
function grumpus() {
    console.log('ugh...you again...');
    console.log('for the last time...');
    console.log('LEAVE ME ALONE!!!');
}
```

RUN

```
funcName();  //run once

funcName(); //run again!
```

RUN

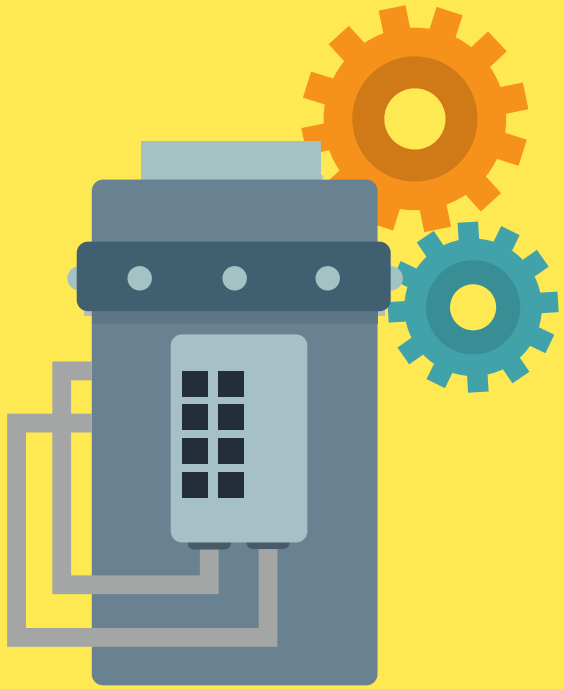
```
grumpus();
//ugh...you again...
//for the last time...
//LEAVE ME ALONE!!!

grumpus();
//ugh...you again...
//for the last time...
//LEAVE ME ALONE!!!
```

ARGUMENTS

INPUTS

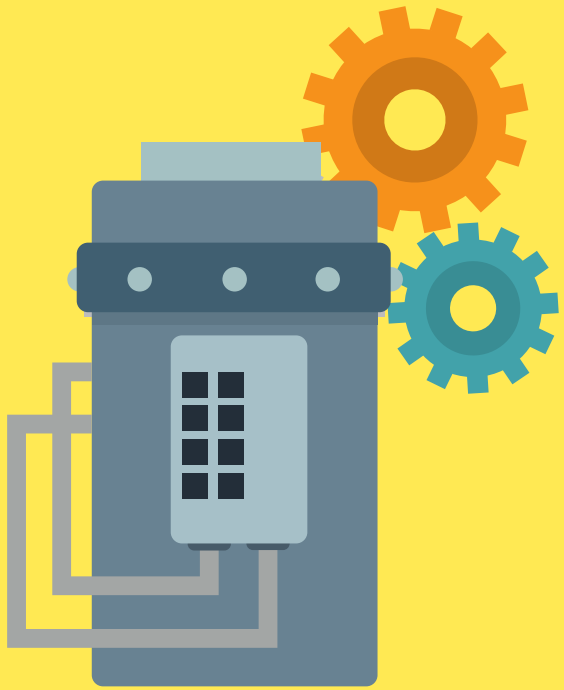
Right now, our simple functions accept zero inputs. They behave the same way every time.

NO INPUTS

`grumpus()` →  → 

`grumpus()` → → 

NO INPUTS

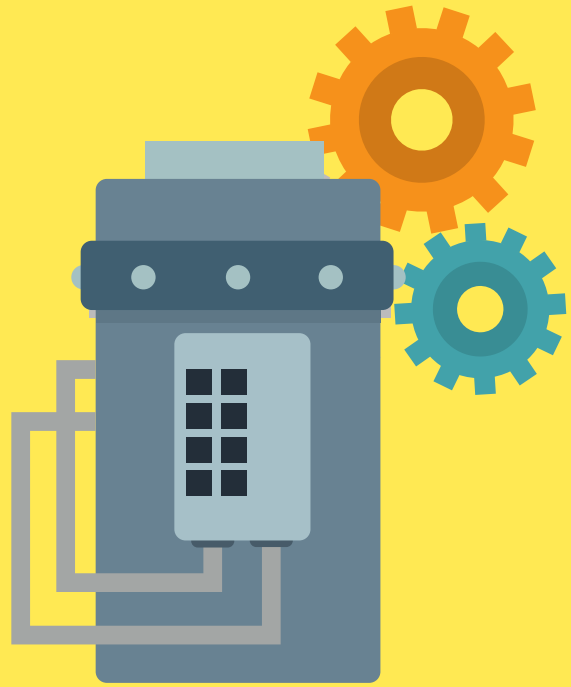
`greet()` →  → `"Hi!"`

`greet()` → → `"Hi!"`

ARGUMENTS

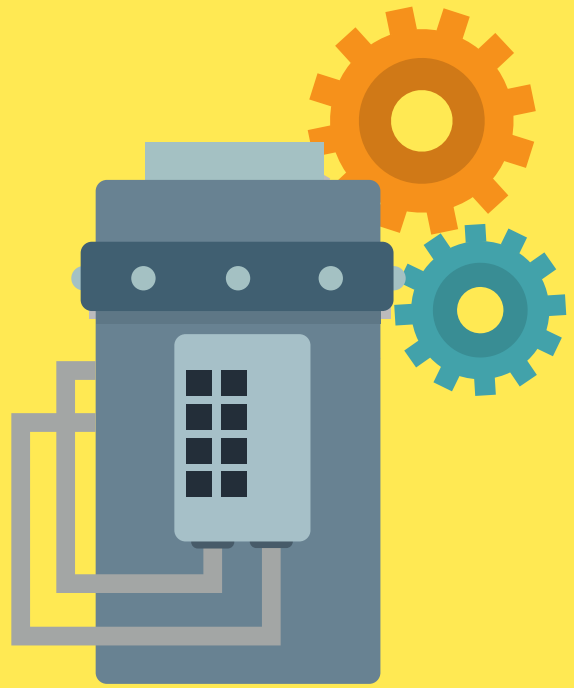
We can also write functions that accept inputs, called arguments

ARGUMENTS

`greet('Tim')` →  → `"Hi Tim!"`

`greet('Anya')` → → `"Hi Anya!"`

ONE MORE

`avg(20,25)` →  → 22.5

`avg(3,2,5,6)` → → 4

We've seen this before

No inputs

```
//No input
"hello".toUpperCase();
```

Arguments!

```
//Different inputs...
"hello".indexOf('h'); //0
//Different outputs...
"hello".indexOf('o'); //4
```

GREET TAKE 2

```
function greet(person) {
  console.log(`Hi, ${person}!`);
}
```

```
greet('Arya');
```

→ "Hi, Arya!"

```
greet('Ned');
```

→ "Hi, Ned!"

2 ARGUMENTS!

```
function findLargest(x, y) {
    if (x > y) {
        console.log(`${x} is larger!`);
    }
    else if (x < y) {
        console.log(`${y} is larger!`);
    }
    else {
        console.log(`${x} and ${y} are equal!`);
    }
}
```

```
findLargest(-2,77)
```

"77 is larger!"

```
findLargest(33,33);
```

"33 and 33 are equal"

RETURN

Built-in methods **return** values
when we call them.
We can store those values:

```
const yell = "I will end you".toUpperCase();

yell; //"I WILL END YOU"

const idx = ['a','b','c'].indexOf('c');

idx; //2
```

NO RETURN!

Our functions print values out, but do NOT return anything

```
function add(x, y) {
    console.log(x + y);
}

const sum = add(10, 16);
sum; //undefined
```

FIRST RETURN!

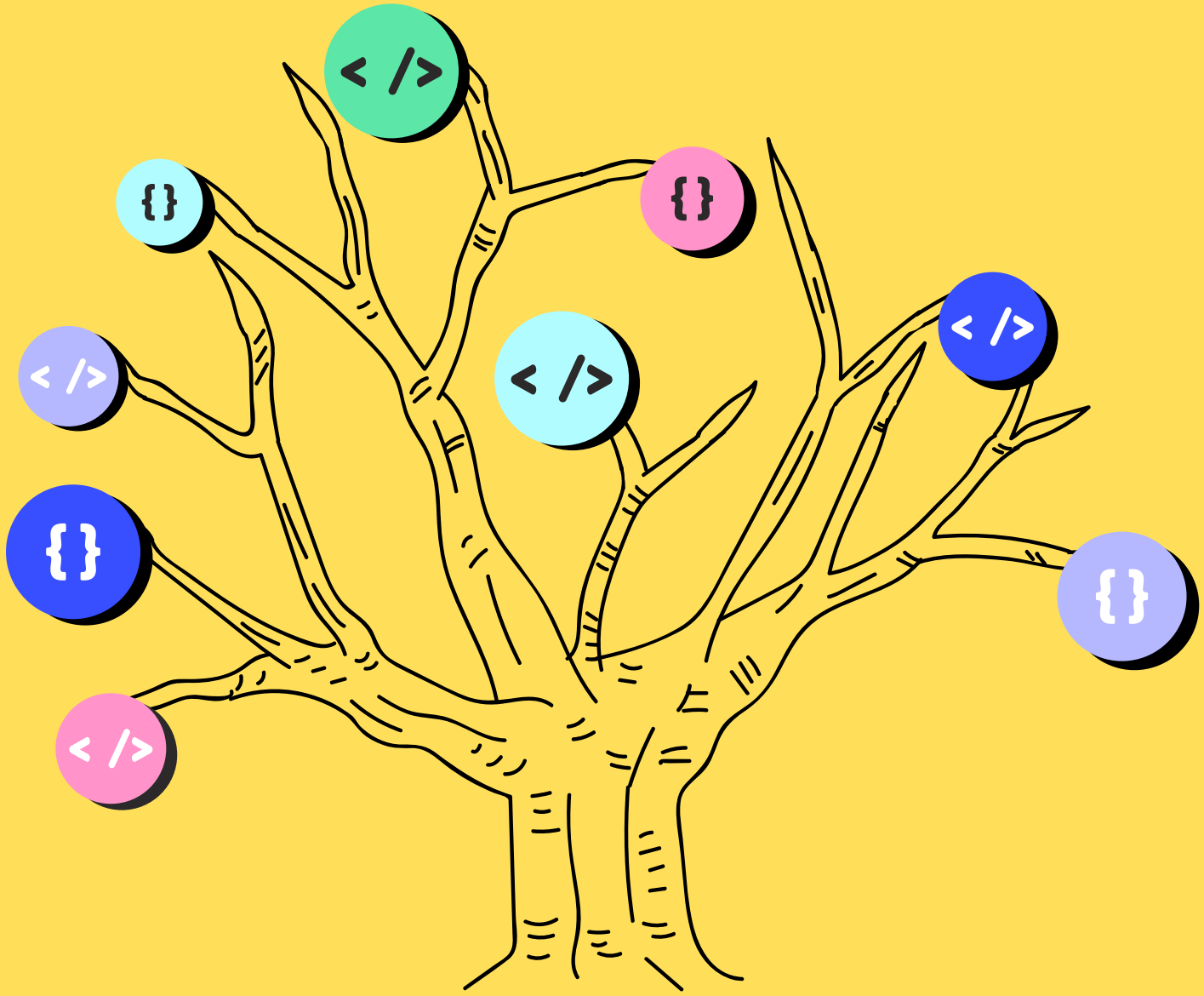
Now we can capture a return value in a variable!

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

const sum = add(10, 16);
sum; //26

const answer = add(100, 200);
answer; //300
```

RETURN

The return statement ends function execution AND specifies the value to be returned by that function

THE DOM

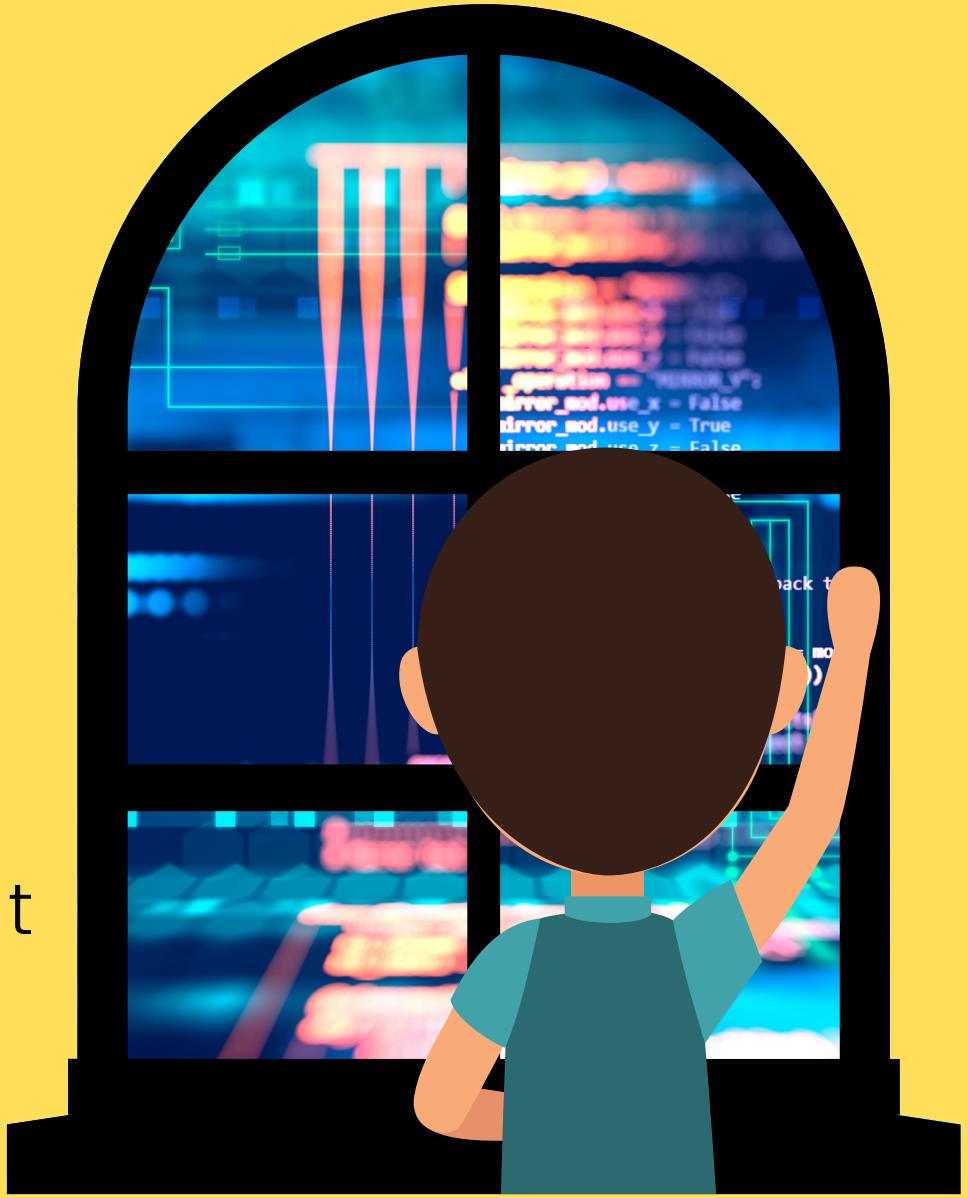


DOM

Document Object Model

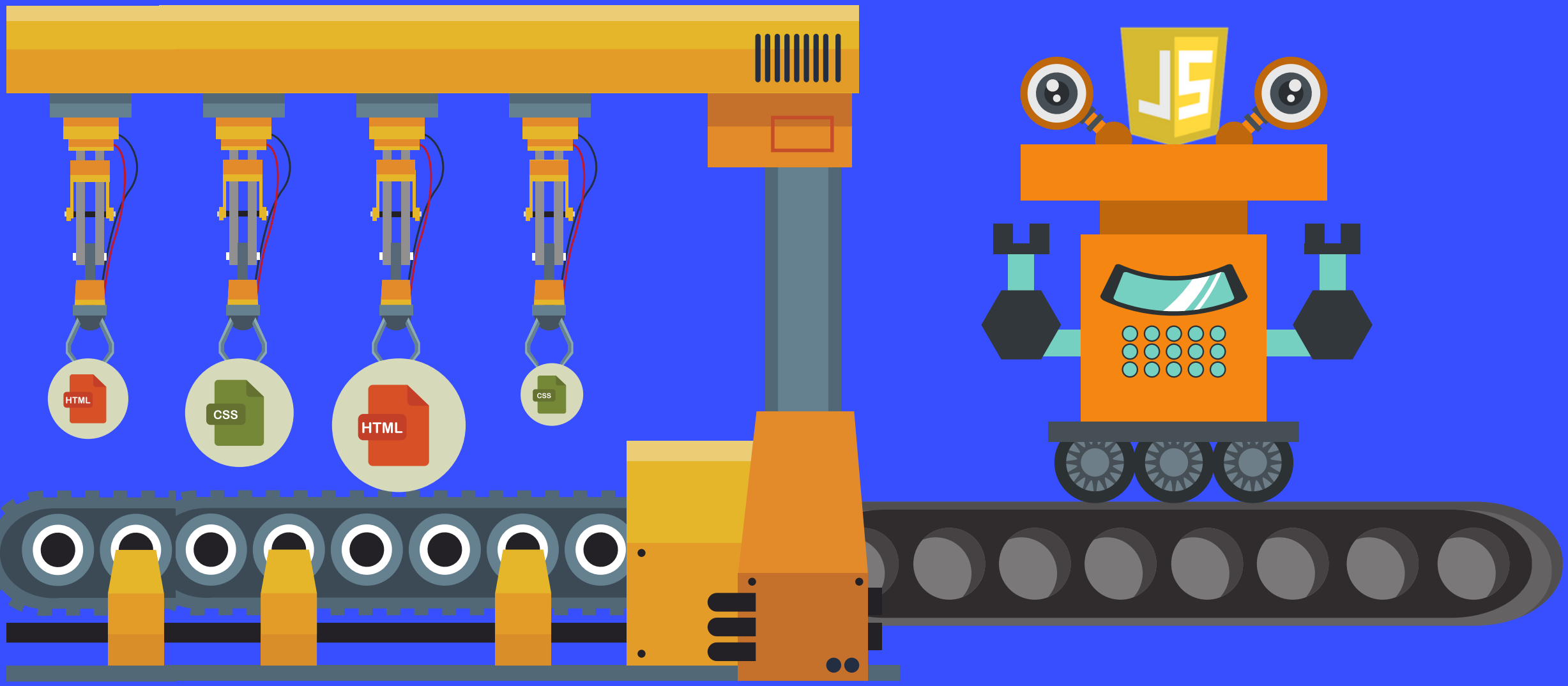
WHAT IS IT?

- The DOM is a JavaScript representation of a webpage.
- It's your JS "window" into the contents of a webpage
- It's just a bunch of objects that you can interact with via JS.

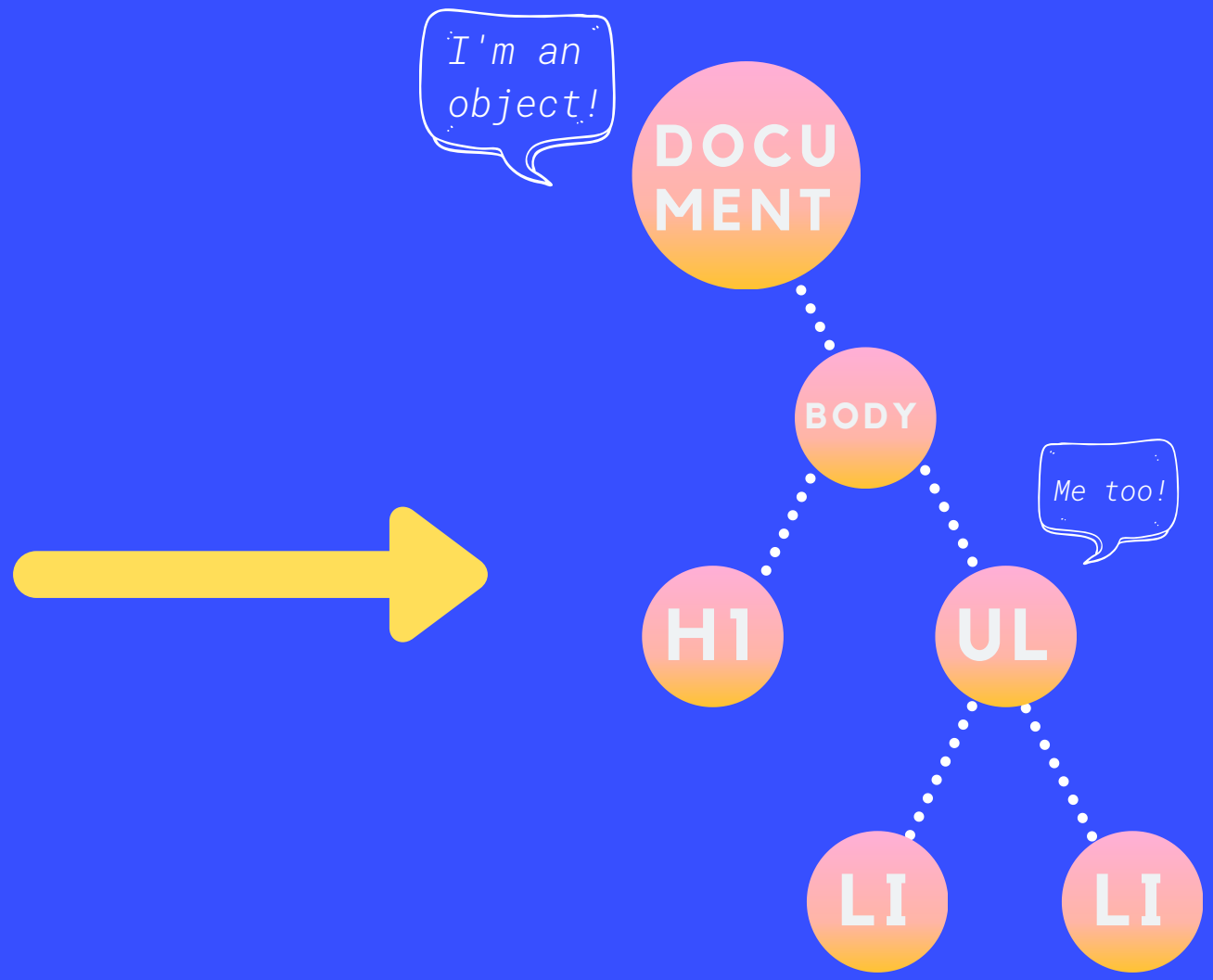


HTML+CSS Go In...

JS Objects Come Out



```
<body>
  <h1>Hello!</h1>
  <ul>
    <li>Water Plants</li>
    <li>Get Some Sleep</li>
  </ul>
</body>
```



HTML+CSS Go In...

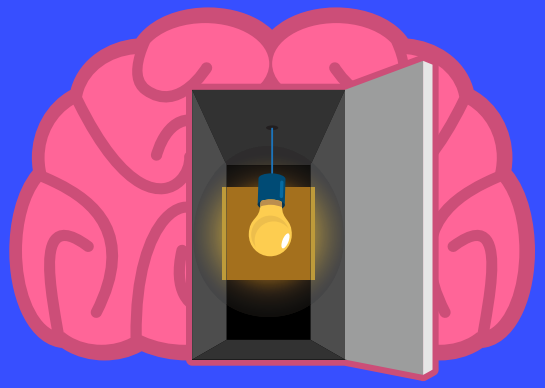
JS Objects Come Out

Let's start with the...

DOCUMENT

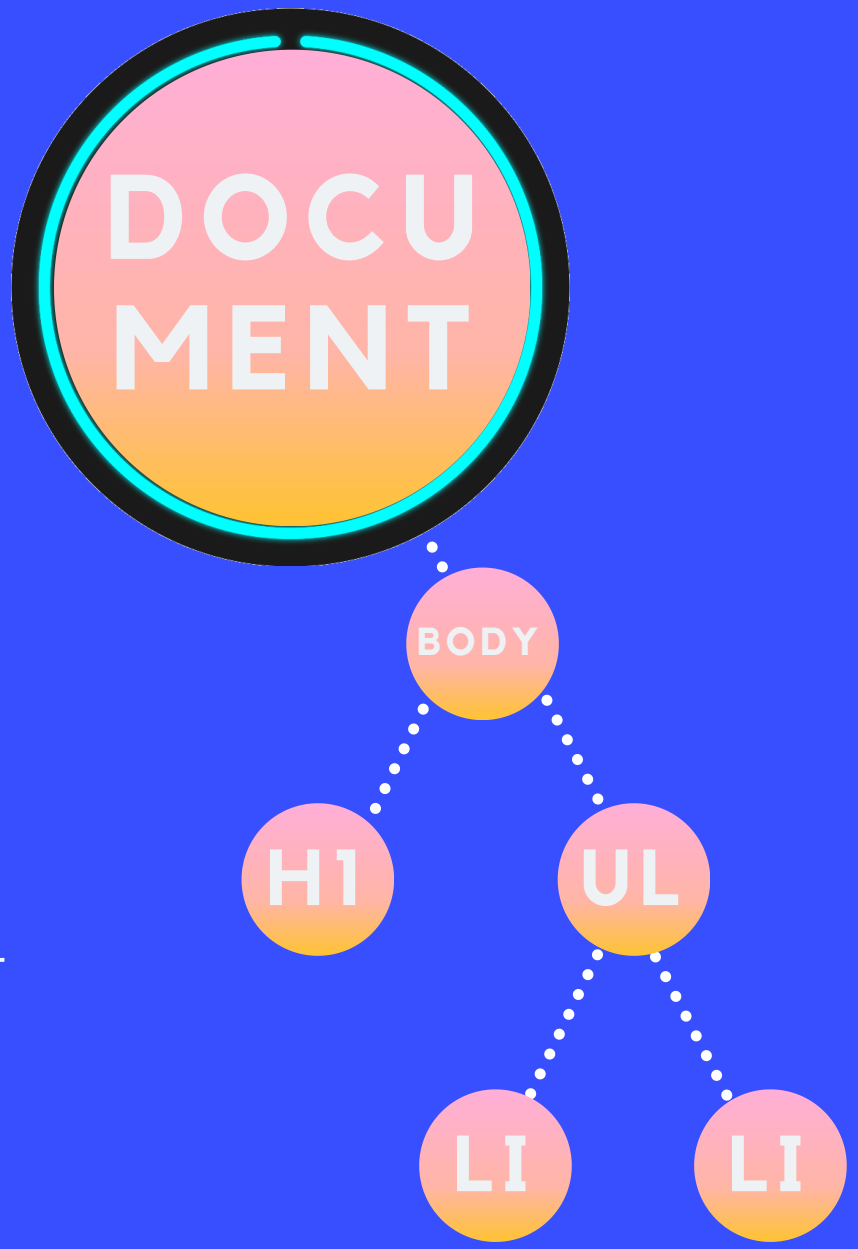
OBJECT

MODEL



Document

The document object is our entry point into the world of the DOM. It contains representations of all the content on a page, plus tons of useful methods and properties



SELECTING

1 SELECT 2 MANIPULATE

SELECTING

- getElementById
- getElementsByTagName
- getElementsByClassName

querySelector

- A newer, all-in-one method to select a single element.

```
//Finds first h1 element:
document.querySelector('h1');

//Finds first element with ID of red:
document.querySelector('#red');

//Finds first element with class of
document.querySelector('.big');
```

querySelectorAll

Same idea , but returns a collection of matching elements

1 SELECT 2 MANIPULATE

PROPERTIES & METHODS

(the important ones)

- classList
- getAttribute()
- setAttribute()
- appendChild()
- append()
- prepend()
- removeChild()
- remove()
- createElement

- innerText
- textContent
- innerHTML
- value
- parentElement
- children
- nextSibling
- previousSibling
- style

EVENTS

Responding to user inputs and actions!

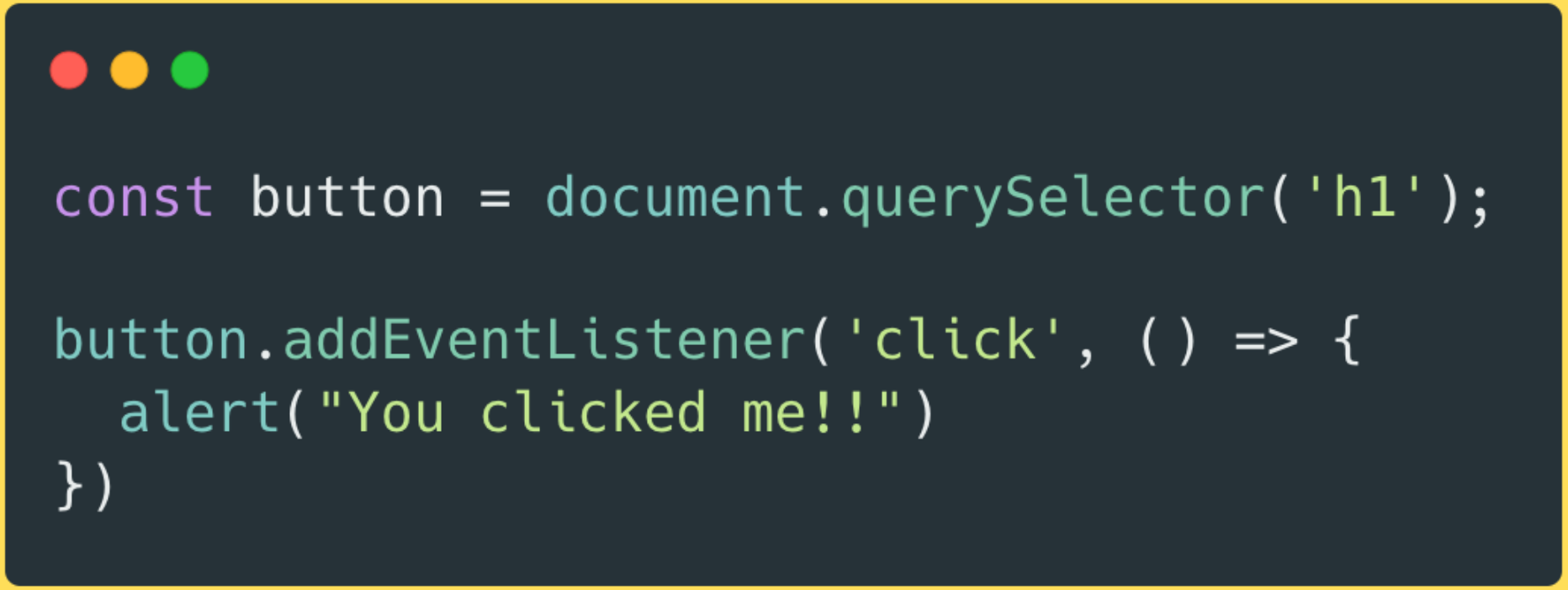
A SMALL TASTE

- clicks
- drags
- drops
- hovers
- scrolls
- form submission
- key presses
- focus/blur

- mouse wheel
- double click
- copying
- pasting
- audio start
- screen resize
- printing

addEventListener

Specify the event type and a callback to run



```
const button = document.querySelector('h1');

button.addEventListener('click', () => {
  alert("You clicked me!!")
})
```