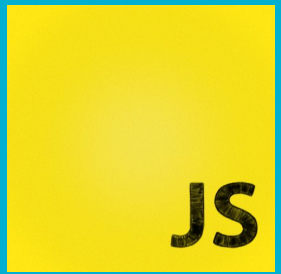
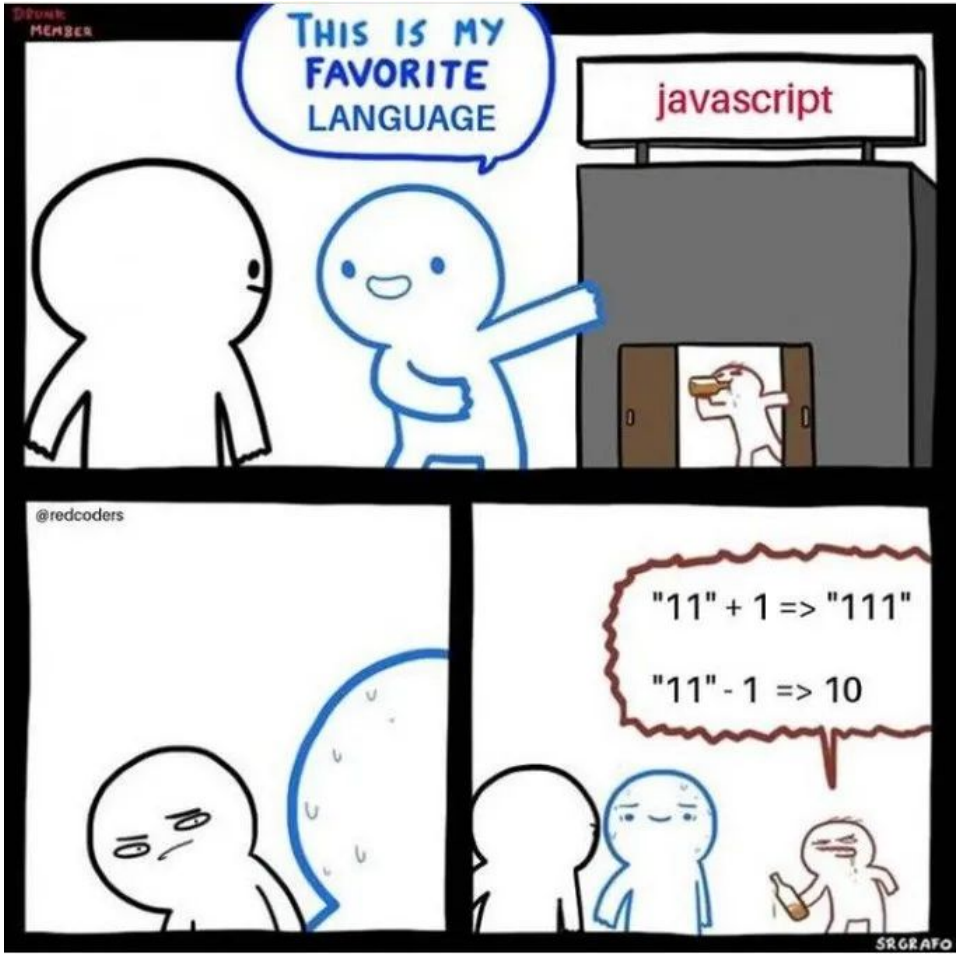
# Javascript

**Javascript and You!**




DRUNK MEMBER
THIS IS MY FAVORITE LANGUAGE
javascript
@redcoders
"11" + 1 => "111"
"11" - 1 => 10
SRGRAFO

# What is Javascript?

- JavaScript is a scripting language primarily used on the client-side.
- It is designed to run directly in the user's browser.
- JavaScript follows the ECMAScript standard, which is published by the ECMA foundation.
- It's important to note that JavaScript is not related to Java; they are completely different languages.

# What is Javascript used for?

- In web browsers, JavaScript allows you to interact with the DOM (Document Object Model), enabling you to perform various actions such as:
  - Showing and hiding elements on the web page.
  - Animating elements to create visual effects.
  - Replacing elements with different elements dynamically.
  - Making requests to the server without needing to reload the entire page.
- The DOM represents all the HTML elements on a web page in a programmatic way.

# JS Datatypes

# JS Primitives

- String: Represents a sequence of characters (e.g., `"This is a string"`).
- Number: Represents numeric values, including integers, decimals, and special values like Infinity (e.g., 12, Infinity, 3.14). All numbers in JavaScript are represented as floating-point numbers.
- Boolean: Represents either `true` or `false`.
- `null`: Represents the absence of a value.
- `undefined`: Represents a variable that has been declared but not assigned a value.
- Symbol: Represents a unique and immutable value used for creating object properties.

# JS Objects

- Objects in JavaScript are more complex data types and include:
  - Object: Represents a collection of key-value pairs (e.g., {key: value}).
  - Array: Represents an ordered list of values (e.g., [1, 2, 3]).
  - Function: Represents a reusable block of code (e.g., function() {}).

# Null vs. Undefined

- In JavaScript, undefined typically indicates that a variable has been declared but not assigned a value yet.
  - Functions that do not explicitly return a value will implicitly return undefined.

- In JavaScript, null is an assignment value used to **represent the deliberate absence of any object value**.
- It is commonly used to indicate that a variable **intentionally** does not have a value assigned.

# typeof operator

- The typeof operator is used to determine the data type of a given value. It returns a string representing the data type of the value passed to it.

```
console.log(typeof 2);        // Output: "number"
console.log(typeof 'Jon');    // Output: "string"
```

# Type Coercion

# Type Coercion

JS Quirk

Type coercion is the process of (implicitly or explicitly) converting a value from one type to another. Since JS is a weakly-typed language, type coercion can be intentional (explicit) or situational (implicit) What does this look like?


# Type Coercion

Pop Quiz!

- 2 + 2 = ?
- 2 +‘2’ = ?
- ‘2’+ 2 = ?
- ‘2’- 2 = ?

# Type Coercion

Pop Quiz Answers

- 2 + 2 = 4
  - No type conversion. Just simple math with numbers
- 2 +‘2’ = 4
  - Number first, so string is converted to number
- ‘2’+ 2 = ‘22’
  - String first, so number is converted to string and ‘+’ turns into a concatenation instead of addition
- ‘2’- 2 = ?
  - ‘-‘ coerces the string ‘2’ to a number to properly use the subtraction operator.

# Type Coercion

JS Quirk

For an interesting look at how '==' can lead to some weird and unexpected coercion results, check out this link:

https://dorey.github.io/JavaScript-Equality-Table/

# Falsey Values

# Six Falsey Values

1. 0 - zero
2. “”- empty string with no whitespace
3. null
4. undefined
5. NaN - Not a Number
6. false - of course!

# Variables

# Variable Declaration

JS Variables

As of ECMAscript 6 (ES6), there are three ways to declare a variable in JavaScript, each with different mechanics especially as it pertains to scope

# var

JS Variables

- The original method of declaring a variable in JavaScript

- Variable names declared in global scope can be reassigned by other var declarations elsewhere in your script / project if using the same variable name.

- Only contained by local (functional) scope.

# let

JS Variables

- The let keyword was introduced in ES6 (2015) as a way to declare variables.

- In the global scope, variables declared with let cannot be reassigned by subsequent let declarations with the same name in the same scope.

- The let keyword maintains block or lexical scope. If a variable is declared with let within a block (such as an if/else statement or a for loop), it will only be accessible within that specific block and not outside of it.

# const

JS Variables

- The const keyword was introduced in ES6 (2015) as a way to declare variables.

- Variables declared with const cannot be reassigned at all after they have been assigned a value. They are considered constant variables.

- const variables also maintain block or lexical scope. If a variable is declared with const within a block, such as an if/else statement or a for loop, it will only be accessible within that specific block.

# JS Variables

In JavaScript, there is a distinction between variable declaration and variable definition:

- **Declaration:** This refers to the act of using one of the variable declaration keywords (var, let, const) to declare a variable name.
  - Example: let x;
- **Definition:** This refers to the process of assigning a value to the variable that has been declared.
  - Example: x = 1;

It's worth noting that variable declaration and variable definition can often occur in a single line, such as let x = 1;. However, there are cases where you may declare a variable without immediately assigning it a value, and then assign a value to it later in your code.

Understanding the distinction between declaration and definition is important when working with variables in JavaScript.

# Functions

# functions

JS functions

In ES6, there are three ways to declare functions in JS:

1. function keyword declaration
2. function expression saved in a variable
3. ES6 Arrow functions

# function

JS functions

- The function keyword is the most straightforward way to declare functions in JavaScript.

- When using the syntax function myFunc(arg) {...}, the function declaration will be hoisted to the top of your script, meaning it can be called before its actual declaration in the code.

- The function keyword can also be used to declare anonymous functions, such as function(arg) {...}. These functions do not have a specific name and can be used as expressions or as callback functions.

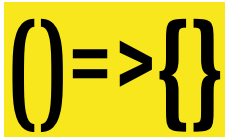
# Function Expression

JS functions

- JavaScript functions are considered first-class functions, which means they can be assigned to variables and passed around as any other data type in JavaScript.

- When declaring functions using function expressions, the definition will not be hoisted to the top of your script. It will be treated like any other variable assignment.

- Function expressions have the additional benefit of allowing a function to be self-invoking, also known as an Immediately Invoked Function Expression (IIFE). This allows the function to be executed immediately after it is defined, without the need for a separate function call.




# JS functions

- Introduced in ES6, Arrow function syntax provides a new way of creating functions with some special rules.

- Arrow functions composed on a single line will implicitly return the result of the operation, while multi-line statements will require the use of curly brackets {} and the explicit return keyword to share any information.

- One of the key features of arrow functions is that they do not create their own this context. Instead, they inherit this from the lexical scope in which they are created. They will keep going up in the scope chain until a suitable context is found.

- Due to these rules, arrow functions are better suited for non-method functions and cannot be used inside constructors.

# Arrays and Objects

# Arrays


- Arrays are used to store collections of data and can contain multiple values of any data type.

```
["string", 11, [2, 3], { key: value }];
```

- Arrays can also be assigned to variables, allowing you to store and manipulate them using the variable name.

```
let array = [1, 2, 3, 4, 5];
```

# Arrays-Access

- Once you have declared an array, you can retrieve the items inside it using their indices.
- Arrays in JavaScript are zero-indexed, which means the index of an array element corresponds to its position from the beginning of the array.
- For example, if you have an array `const cars = ["Porsche", "Camry"]`, to access the value "Porsche" in the cars array, you would use its index

```
cars[0] // “Porsche”
```

# Objects

- Objects in JavaScript are a way of organizing data using key/value pairs.
- For example, you can create an object `const car = { make: "Toyota", model: "Matrix" }`
- Similar to arrays, you can access information using bracket notation, where the content inside the brackets is the key you want to target.

```
car["make"]; //"Toyota"
```

# Objects - Dot Notation

You can also use “dot notation” to get data out of an object.

```
const user = {firstName: "Lucille", lastName: "Bluth”}

user.firstName // "Lucille"
```

# Objects - Destructuring

New syntax introduced as of ES6, destructuring allows you to break an array into its elements without mutating the original array

```
const arr = [1, 2, 3]

const [a, b, c] = arr

console.log(a, b, c) // 1, 2, 3
```

# Objects - Destructuring Continued

Similar functionality exists for objects, using the key as a variable name to access the value at that key, for example:

```
const obj = {firstName: 'Jon', favColor: 'blue'}

const {firstName, favColor} = obj

console.log(firstName,favColor) // 'Jon', 'blue'
```