skillcrush

The ULTIMATE JavaScript Fundamentals Study Guide

Class Lessons

- Lesson 1
- Lesson 2
- Lesson 3
- Lesson 4
- Lesson 5
- Lesson 6
- Lesson 7
- Lesson 8
- Lesson 9
- Lesson 10
- Lesson 11
- Lesson 12
- Lesson 13
- Lesson 14

Terms & Definitions

- \n
- + operator
- Addition operator
- alert()
- API
- API parameters
- API URL
- append()
- Arguments
- Arithmetic operators
- Arrays
- Assignment operator
- async keyword
- await keyword
- Block scope
- Boolean
- Boolean data type
- Bracket notation
- Callback function
- Calling
- CamelCase
- Change event
- classList property
- classList.add()
- classList.remove
- Comparison operators
- Compound assignment operators
- Concatenating
- Conditional statement
- Console
- console.log()
- const keyword
- const vs. let vs. var
- Context
- createElement()
- Data type
- Date()
- Debugging
- Decomposition
- defer attribute
- Delimiter
- disabled property
- DOM
- DOM events
- DOM tree
- Dot notation
- Elements
- else if keyword
- else keyword
- Endpoints
- Event handler
- Event listener
- Expressions
- Factory functions
- fetch()
- Floating point number
- for loop
- for...of loop
- for...in loop
- forEach()
- Function
- Function body
- Function expression
- Function scope
- getHours()
- Global scope
- if keyword
- Index
- innerHTML property
- innerText property
- Input event
- Integer
- Iterating
- JSON files
- json()
- Keydown event
- Keys
- Key-value pair
- let keyword
- Loop
- Loop body
- match()
- matches()
- Math.floor()
- Math.max()
- Math.min()
- Math.random()
- Method (Lesson 4)
- Method (Lesson 10)
- Modal window
- Modulus (%) operator
- Mouse events
- Nested if statement
- Null
- Number()
- Object literal
- Objects
- Parameters
- Primitive data types
- prompt()
- Properties
- querySelector()
- querySelectorAll
- Refactored code
- Regular expression
- Reserved keywords
- REST APIs
- return keyword
- Scope
- Statement
- String
- style property
- Template literals
- this keyword
- toFixed()
- toLowerCase()
- toUpperCase()
- trim()
- Type conversion
- Undefined
- value property
- Values
- Variable

Lesson 1 - Getting Started with JavaScript

Variable - A tool for pointing towards information. The associated value can vary or change. Variables can be declared using the keyword `var` (Lesson 1) and the keywords `let` and `const` (Lesson 9).

Values - Information stored in a variable. Specifically, a value is a sequence of bits that is interpreted according to some data type (e.g., number, string, boolean).

Assignment operator - An operator that assigns a value to a variable. The assignment operator uses the equal sign (=). See the Javascript Operators Cheatsheet (Lesson 3) for a list of assignment operators.

Statement - A single instruction to the program. Often, semicolons appear at the end of a statement to show it's complete.

```
var cerealTypes = 16;
```

```
console.log("We love JS!");
```

String - A series of characters, like numbers, letters, and symbols. Strings will have quotes around them to group the characters and keep them in a sequence.

```
var vacationSpot = "beach";
```

```
var phoneNumber = "555-555-1234";
```

CamelCase - The standard naming convention for variables in JavaScript. The first words are all lowercase letters, while each proceeding word begins with an uppercase letter. Examples: `bankDeposit`, `userInputDate`, and `ageLimit18`

Console - An environment in your browser where you can execute, or run, JavaScript. The console lets you see the output of your code and troubleshoot errors. In CodeSandbox, your console is located under the "Console" tab. In Google Chrome, go to More Tools > Developer Tools > Console tab.

console.log() - A method to log out a message to the console.

```
console.log("Party time! Excellent!);
// Party time! Excellent!
```

```
var cats = 4;
console.log("I have" + cats + "cats.");
// I have 4 cats.
```

+ operator - An operator that uses the plus sign (+) to combine strings and variables.

```
var name = "Giorno Giovanna";
console.log("His name is" + name + ".");
// His name is Giorno Giovanna.
```

Concatenating - The process of joining strings together using the + operator.

```
var ringMetal = "gold";
console.log("She gave her a" + ringMetal + "ring.");
// She gave her a gold ring.
```

Back to Top ↑

Lesson 2 - Data Types & Arithmetic Operators

Template literals - Output strings using placeholders and backticks (`). Compared to outputting strings with single or double quotes and the plus operator, template literals

make it easier to output multi-line strings and combine variables with strings. In addition, you can calculate expressions inside the string.

```
var jewelry = "watch";
var event = "dinner";

console.log(`They wore a ${jewelry} to ${event}.`);
// They wore a watch to dinner.
```

```
var pizzaType = "veggie";
var slicesEaten = 4;
console.log(`The ${pizzaType} pizza has ${8 - slicesEaten} slices
left.`);
// The veggie pizza has 4 slices left.
```

Expressions - Code that results in a value. For example, expressions can result in numeric, string, and logical values (Lesson 3).

```
console.log(8 - 5);
// 3
```

```
console.log("I love" + " coding.");
// I love coding.
```

```
console.log(5<8);
// true
```

Integer - A whole number, like 5100 or -258. Integers can be positive or negative.

Floating point number - A number with a decimal, like 2134.3625 or -562.12. Floating point numbers can be positive or negative.

Addition operator - An operator to add two numbers together. The addition operator

uses the plus sign (+).

```
var applesBananas = 5 + 8;
console.log(applesBananas);
// 13
```

```
var floor1 = 10;
var floor2 = 15;
console.log(`There are ${floor1 + floor2} tables in the restaurant.`);
// There are 25 tables in the restaurant.
```

Arithmetic operators - Symbols for math operations, like the addition (+), subtraction (-), multiplication (*), and division (/) operators. See the JavaScript Operators Cheatsheet (Lesson 3) for a complete list of arithmetic operators.

Data type - The type of value a variable points to. Examples include numbers, strings, booleans (Lesson 3), undefined, null, arrays (Lesson 8), and objects (Lesson 10).

Primitive data types - Values with only a single value, like numbers, strings, booleans (Lesson 3), undefined, and null.

Undefined - A variable with no value assigned to it.

```
var happiness;

console.log(happiness);
// undefined
```

Null - A data type that represents an intentionally empty, or non-existent, value.

```
var ideas = null;
```

```
console.log(ideas);
// null
```

Type conversion - Changing one value to a different value to complete an operator. Type conversion is beneficial for changing strings into numbers so you can calculate them.

Number() - Convert a string into a number. Number() is useful when gathering input from a user and then changing it to a number so that you can calculate a value.

```
var tvShows = Number("23");
var movies = 12;
console.log(tvShows + movies);
// 35
```

prompt() - Displays a field to gather information from the user. Users will see a pop-up dialog box on their screen asking for input.

```
var favoriteGenre = prompt("What's your favorite music genre?");
console.log(favoriteGenre);
```

```
var oldFunds = 1500;
var newFunds = Number(prompt("How much funds were raised?"));
console.log(
`The fundraiser total is now $${oldFunds + newFunds}!.`
);
```

toFixed() - Convert a number data type into a string and then round to a specified number of decimal places. Add a number inside toFixed() to specify the number of decimal places to round to.

```
var taxAmount = 7.23335651;
console.log(taxAmount.toFixed(2));
// 7.23
```

```
var tempFahrenheit = 98.6785;
console.log(`Her temperature is ${tempFahrenheit.toFixed(1)}.`);
// Her temperature is 98.7.
```

```
var people = 27;
var payout = 800.29;
console.log(`You won $${(payout / people).toFixed(2)}.`);
// You won $29.64.
```

Back to Top ↑

Lesson 3 - Comparisons & Conditionals

Conditional statement - Code that will only run if a condition is true.

Boolean - Represent just two values: true or false.

Boolean data type - A primitive data type with true or false values.

```
var lightsOn = true;
var fanOn = false;
console.log(lightsOn);
//true
```

Comparison operators - Operators that use symbols to compare two or more values, like >, <, and ===. See the JavaScript Operators Cheatsheet (Lesson 3) for a complete list of comparison operators.

if keyword - Keyword to use in a statement to test a condition. If the condition evaluates to true, then the program runs the code inside the if block. You won't include a semicolon

after the condition.

```
var hotWeather = true;

if (hotWeather === true) {
  console.log("Wear shorts and a tank top today!");
}
// Wear shorts and a tank top today!
```

else keyword - Keyword to use in a statement to perform another action if the previous condition evaluates to false.

```
var hotWeather = false;

if (hotWeather === true) {
  console.log("Wear shorts and a tank top today!");
} else {
console.log("Grab a sweater, it might be chilly.");
}
// Grab a sweater, it might be chilly.
```

else if keyword - Keyword to use in a statement to test a new condition, and then perform an action if the previous condition evaluates to false. As soon as a condition evaluates to true, the code block that the condition is associated with runs and the conditional block is exited, regardless if there are subsequent conditions that would also evaluate to true.

```
var hotWeather = false;
var snowyWeather = true;
var windyWeather = true;

if (hotWeather === true) {
  console.log("Wear shorts and a tank top today!");
} else if (snowyWeather === true)  {
console.log("Put on a heavy jacket and boots!");
} else if (windyWeather === true)  {
```

```
console.log("Time to slip on your windbreaker.");
} else {
console.log("Grab a sweater, it might be chilly.");
}
// Put on a heavy jacket and boots!
```

alert() - Displays a pop-up message for users to see. The prompt includes an OK button for users to click and close the pop-up.

```
alert("Hello, welcome to my site!");
```

Date() - A method to retrieve the current date.

```
var weekday = new Date().toLocaleString("en-US", { weekday: "long" });
```

getHours() - A method to retrieve the current time. The time will reflect the 24-hour clock, AKA military time.

```
var time = new Date().getHours();
```

Back to Top ↑

Lesson 4 - JS, HTML, & CSS

---

defer attribute - Instructs the browser to load the script after the page has loaded. The attribute creates a faster loading experience for the user because all the HTML renders first, even if the JavaScript hasn't run yet. It also makes sure the HTML elements are loaded so the JavaScript can modify them. You'll add the <script> tag and defer attributes in the head section of the HTML page.

```
<!DOCTYPE html>
```

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

DOM - Short for Document Object Model, the DOM represents the structure and content of a web page. The document is the web page. The objects include HTML elements, text, and attributes.

DOM tree - A graphical representation of the DOM which shows relationships between objects. The DOM tree is useful for determining how to access different objects on the document.

Methods - JavaScript actions performed on objects. Examples of methods include console.log(), prompt(), alert(), , and querySelector(). Methods are also a type of object property (Lesson 10).

querySelector() - A method to access the first element that matches a specified selector. To select multiple items, you'll need to use an array (Lesson 8) with querySelectorAll() (Lesson 9).

```
var available = document.querySelector("h3");
var mainTitle = document.querySelector(".main-title");
var firstItem = document.querySelector("ul li");
var intro = document.querySelector(".intro p");

console.log(available, mainTitle, firstItem, intro);
// <h3>We're here for you every day of the week.</h3>
// <h1 class="main-title">Ryan's Roses</h1>
// <li>Today's Specials</li>
// <p>Available today</p>
```

```
var firstImg = document.querySelector("img");
firstImg.src = "img/lulu.jpeg";
```

```
firstImg.alt = "Lulu bouquet";

console.log(firstImg);
// <img src="img/lulu.jpeg" alt="Lulu bouquet"></img>
```

style property - A property that allows you to change the style of an element. If the property name is two words, like background-color, change it to one word using camelCase (backgroundColor).

```
var intro = document.querySelector(".intro p");

intro.style.color = "purple";
intro.style.fontSize = "3em";
intro.style.fontStyle = "italic";

console.log(intro);
// <p style="color: purple; font-size: 3em; font-style:
italic;">Available today</p>
```

innerText property - A property that accesses the text within an element. This property is useful when you want to change or retrieve the text inside an element.

```
var firstCaption = document.querySelector("figcaption");
firstCaption.innerText = "The Lulu.";

console.log(firstCaption);
// <figcaption>The Lulu.</figcaption>
```

innerHTML property - A property that changes the HTML of an element on the page. This property is useful for updating or adding elements to a page.

```
firstCaption.innerHTML = "The <strong>Lulu</strong>";

console.log(firstCaption);
//<figcaption>The<strong>Lulu</strong></figcaption>
```

```
var intro = document.querySelector(".intro p");

intro.innerHTML = 'Available <span
class="increase__size">today</strong>';

console.log(intro);
// <p>Available<span class="increase__size">today</span></p>
```

Debugging - Identifying and removing errors in your code.

Back to Top ↑

Lesson 5 - Events & Event Listeners

DOM events - Actions that happen in the document (web page). Events can be triggered by the browser or by the user. In this class, you'll use mouse, change, keydown, and input events. See Mozilla's Event Reference page for a complete list of events.

Mouse events - An event that happens when a pointing device, like a mouse, joysticks, keyboard, or adaptive switch interacts with the document. Common mouse events are `"click"`, `"mouseover"`, and `"select"`.

Event listener - A method that "listens" for events to happen and then takes action. Use the method `addEventListener()` to listen for events in the DOM.

```
var title = document.querySelector("h1");

title.addEventListener("mouseover");
```

Event handler - A function that runs code when an event occurs.

```
var title = document.querySelector("h1");
```

```
title.addEventListener("mouseover", function () {
      title.innerText = "Let's PARTY!";
      title.style.color = "maroon";
});
```

Function - A block of code that can be called or invoked to run as many times as needed without repeating code. Functions are vital to writing streamlined JavaScript. Lesson 6 contains a full dive into functions.

Function body - The part of the function that contains the statements that specify what the function does. Curly braces surround the function body.

classList property - A property to add, remove, or toggle CSS classes on an element. This property let's you apply (or remove) multiple styles at once. You can use the classList property with the add() and remove() methods: `classList.add()` and `classList.remove()`.

classlist.add() - A method to add a new class.

```
var darkModeButton = document.querySelector(".dark-mode");
var body = document.querySelector("body");

darkModeButton.addEventListener("click", function () {
body.classList.add("dark-palette");
});
```

classlist.remove() - A method to remove a new class.

```
var lightModeButton = document.querySelector(".light-mode");

lightModeButton.addEventListener("click", function () {
```

```
body.classList.remove("dark-palette");
});
```

Modal window - A web page element that overlays a box in front of a web page. A modal is also called a lightbox.

Back to Top ↑

Lesson 6 - Functions

Function expression - A syntax for writing functions that begins with a variable name and then uses the `function` keyword to define the function.

```
var welcome = function () {
console.log();
};
```

Reserved keywords - A word that can't be used as a variable name in JavaScript. See a complete list of reserved keywords.

Parameters - Placeholders for values you want to pass to the function. If there's more than one parameter, separate the parameters with a comma.

```
var welcome = function (name) {
  console.log(`Welcome, ${name}. Have a great day!`);
}
```

Calling - An action which will cause a function to run. If the function expects arguments, you must provide them in the function call.

```
var welcome = function (name) {
```

```
  console.log(`Welcome, ${name}. Have a great day!`);
}

welcome("Sadie");
// Sadie
```

Arguments - Values passed to the function when it's called. If there's more than one argument, separate the arguments with a comma.

```
var addTogether = function (num1, num2) {
  console.log(num1 + num2);
};

addTogether(13, 72); // 85
addTogether(36, -2.88); // 33.12
```

return keyword - A keyword to return the value of a function and end its execution. Use the `return` keyword to make the result of a function available to other parts of your code. Unlike `console.log()` which only outputs a message to the console, the `return` keyword allows a value to be used by other parts of the code, including `console.log()`.

```
var addTogether = function (num1, num2) {
  return num1 + num2;
};

alert(addTogether(36, -2.88));
console.log(addTogether(13, 72)); // 85

var lunchForTwo = addTogether(24.56, 18.99);
console.log(lunchForTwo); // 43.55
```

Lesson 7 - Keydown & Change Events

Callback function - A function that's passed to another function as an argument. For example, an event handler is a callback function.

```
button.addEventListener("click", function () {
  cat.classList.add("show");
});
```

Keydown event - An event that occurs when a key is pressed on a keyboard, like a letter, number, or Enter key. Inside the callback function for a keydown event, you'll pass a parameter that will hold the event object. Most coders use `e` as the parameter to represent "event."

```
var body = document.querySelector("body");

document.addEventListener("keydown", function (e) {
  // console.log(e);
  if (e.key === "l") {
    body.classList.add("light");
  }
});
```

Nested if statement - An if statement testing the condition of another if, else if, or else statement.

In this example, the second if statement (`if (body.classList.contains("light"))`) is the nested if statement.

```
var body = document.querySelector("body");

document.addEventListener("keydown", function (e) {
 // console.log(e);
 if (e.key === "l") {
   body.classList.add("light");
 } else if (e.key === "d") {
   if (body.classList.contains("light")) {
     body.classList.remove("light");
   }
 }
```

```
});
```

Change event - An event that occurs when the user changes a drop-down list (i.e., the <select> element) or input areas like the <input> or <textarea> elements. Inside the callback function for a change event, you'll pass a parameter that will represent the change event. Most coders use `e` as the parameter to represent "event."

```
var fave = document.querySelector("#favorite");
var heading = document.querySelector("h1");
var selection = "regular";

fave.addEventListener("change", function (e) {
 selection = e.target.value;
 if (selection === "stealth") {
   heading.innerText = "Stealth Quincy 😎";
 } else if (selection === "party") {
   heading.innerText = "Party Quincy 🥳";
 } else {
   heading.innerText = "Quincy";
 }
});
```

toUpperCase() - A method for converting a string value into all uppercase letters.

```
var louder = "Speak up, please!"

console.log(louder.toUpperCase());
// SPEAK UP, PLEASE!
```

Math.floor() - A method for rounding a number down to the next whole number.

```
var seatingCapacity = 1256.3;

console.log(Math.floor(seatingCapacity));
//1256
```

Math.random() - A method for producing a random number between 0 and 1. Multiply it by another number to output a larger random number. Pair it with `Math.floor()` to round

the number to the nearest whole number.

```
console.log(Math.random());
// 0.15884857919099582
```

```
console.log(Math.random() * 36);
// 18.873096475917126
```

```
console.log(Math.floor(Math.random() * 12));
// 10
```

Back to Top ↑

Lesson 8 - Arrays & Loops

Arrays - A data type that contains one or more values. You'll add square brackets around the array values (elements). See the Javascript Arrays Cheatsheet (Lesson 8) for a list of array methods.

```
var timeOfDay = [6, "noon", 8, "morning", "evening", 12];
```

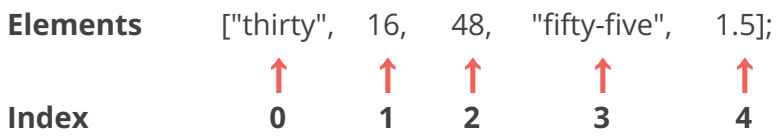
You can also create an empty array to add items.

```
var medicine = [];
```

Elements - The values stored in an array. Elements can be strings, numbers, and floating point numbers data types.

```
var ages = ["thirty", 16, 48, "fifty-five", 1.5];
```

Index - The position of an element in an array. In JavaScript, the first element starts at index 0. The second element would start at index 1, and so on.



Loop - A statement that allows you to repeat code multiple times.

Iterating - Each time a loop runs through a block of code. Each pass of the loop is called an iteration.

for loop - A type of loop that iterates through a block of code a designated number of times. Examples of for loops include the for...of loop (Lesson 8) and the for...in loop (Lesson 11).

for...of loop - A type of for loop that iterates over the values of an array. A `for...of` loop only has access to the values of an array, not the index. You can use the `for...of` loop with the for...in loop (Lesson 11) to loop through multiple object properties.

```
var timeOfDay = [6, "noon", 8, "morning", "evening", 12];

for (var time of timeOfDay) {
  console.log(`It is ${time}.`);
}
// It is 6.
// It is noon.
// It is 8.
// It is morning.
// It is evening.
// It is 12.
```

Loop body - The loop section where you'll write the statements you want to execute on each loop iteration.

forEach() - A method to iterate through elements in an array and then execute a function for each array item. Unlike `for...of` loops, forEach() lets you access the array elements' value and index.

```
var timeOfDay = [6, "noon", 8, "morning", "evening", 12];

timeOfDay.forEach(function (time, index) {
  console.log(`The ${time} element is at position ${index}.`);
});
// The 6 element is at position 0.
// The noon element is at position 1.
// The 8 element is at position 2.
// The morning element is at position 3.
// The evening element is at position 4.
// The 12 element is at position 5.
```

Modulus operator - An operator to return the remainder of two numbers divided. The modulus operator uses the percent sign (%). The modulus operator is also called the "modulo" operator.

```
var candy = 14;
var kids = 4;

console.log(`There are ${candy % kids} pieces of candy remaining.`);
// There are 2 pieces of candy remaining.
```

```
var num = 45;

if (num % 2 === 0) {
  console.log("This is an even number.");
} else {
  console.log("This is an odd number.");
};
// This is an odd number.
```

Back to Top ↑

Lesson 9 - Scope

Scope - The context where variables are visible to certain parts of your program. Scope can be divided into global scope, function scope, and block scope.

Context - The place the code is evaluated and executed, like inside a function or loop.

Global scope - The context for the whole program. Globally scoped variables are available to any part of the program.

Function scope - The context inside a function. Variables defined within a function are scoped only to that function or nested functions.

Block scope - The context inside a block of code. Unlike declaring variables with var, declaring your variables with let and const keeps variables in block scope.

let keyword - A keyword to declare variables and prevent them from being accessed outside the block they were declared in. Use `let` inside code blocks (e.g., loops, if/else if statements) and when you want to reassign the value of a variable.

```
if (numOfDrinks === 5) {
  let soda = "lemon-lime";
  console.log(soda);
}
// lemon-lime

console.log(soda);
// ReferenceError: soda is not defined
```

const keyword - A keyword to declare variables to constrain a variable to block scope and prevent the value from being reassigned. Using `const` will prevent data types like

strings, booleans, and numbers from being reassigned to a different value. For data types like arrays and objects, `const` will prevent reassigning the variable but still allow you to modify the elements inside the array/object.

```
const numOfDrinks = 5;

const drinks = function () {
  const tea = 6 + numOfDrinks;
  console.log(tea);
};

drinks();
// 11
```

If you try to reassign a variable declared with const, you'll receive an error in the console like “TypeError: Assignment to constant variable” or "<variable name> is read-only" when attempting to reassign a variable.

```
const numOfDrinks = 5;
numOfDrinks = 7;

console.log(numOfDrinks);
// SyntaxError: /script.js: "numOfDrinks" is read-only
```

const vs. let vs. var - For most uses, you'll want to use const to declare your variables, except when you need to reassign variables (`let`) or you're working with legacy code (`var`).

	const	let	var
Function scoped	Yes	Yes	Yes
Block scoped	Yes	Yes	No
Reassignable	No	Yes	Yes
Redeclareable	No	No	Yes
Summary	Use `const` as the default way to declare variables.	Declare variables with `let` when you need to reassign the	Use `var` when working with legacy code that already

	You'll use `const` 95% of the time.	value of your variables, like in a loop or if/else if statement.	uses var. Declaring with `var` is also helpful when learning to write code and scope issues aren't a factor.

value property - A property to capture the content entered into a text box.

createElement() - A method to create a new HTML element.

append() - A method to add elements at the end of another DOM element, like a list.

querySelectorAll() - A method to select all the elements that match a specific selector. The querySelectorAll() returns a list of elements in an array-like structure..

```
addShowButton.addEventListener("click", function () {
  const show = showInput.value;
  if (show !== "") {
    let listItem = document.createElement("li");
    listItem.innerText = show;
    showList.append(listItem);
    let shows = document.querySelectorAll(".show-list li");
    showCount.innerText = shows.length;
  }
});
```

length property - A property to identify the number of elements in an array.

```
var daysOfWeek = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
console.log(daysOfWeek.length);
// 5
```

Refactored code - Code that was restructured without changing or adding to its functionality, usually with the goal to make it more readable, better performing, or both.

disabled property - A property to indicate if an element can be interacted with or not. The `disabled` property uses the boolean values of true and false. For example, if the `disabled` property is set to `true` for a button, the user can no longer click the button.

```
assignButton.addEventListener("click", function () {
  assignItems();
  assignButton.disabled = true;
});
```

Math.min() - A method that finds the smallest value passed to it. Use the spread (...) operator to send individual array elements to Math.min() instead of the whole array.

```
console.log(Math.min(2, -12, 71));
// -12
```

```
const employees = [12, 68, 333, 56, 1250];
const smallestNum = Math.min(...employees);

console.log(smallestNum);
// 12
```

Math.max() - A method that finds the largest value in an array. Use the spread (...) operator to send individual array elements to Math.max() instead of the whole array.

```
console.log(Math.max(2, -12, 71));
// 71
```

```
const employees = [12, 68, 333, 56, 1250];
const largestNum = Math.max(...employees);

console.log(largestNum);
// 1250
```

Back to Top ↑

Lesson 10 - Objects & Methods

Objects - A data type used to group multiple properties and their corresponding values into a single, unordered entity. Like an object in real life, a JavaScript object represents a thing with characteristics (properties), like a person, animal, instrument, or house. An object is a collection of key-value pairs.

Properties - Different values of an object. A property represents the different characteristics of your object. Properties can be any data type, like a number, string, array, boolean, or function.

Dot notation - A syntax to assign or access the property of an object using a period between the object name and property.

```
const cat = {};

cat.name = "Fred";
cat.nickname = "Flooficus";
cat.age = 5;
cat.isSleeping = true;
cat.favoriteToys = ["spring", "ping pong balls", "bird stuffy"]
cat.pet = function () {
  return "purrrrrrrr";
}
```

Method (object property) - A function that's a property in an object. Use the `return` keyword to return the value of the method and make the result available to other parts of your code.

Methods can be added to an object following its creation:

```
const cat = {};

cat.pet = function () {
  return "purrrrrrrr";
}
```

Methods can also be created inside an object literal:

```
const cat = {
  pet: function () {
    return "purrrrrrrr";
  }
};
```

Keys - An object's unique elements which are used to access its values. Keys are also known as "identifiers" or "names." An object's keys must be unique and cannot be duplicated in the same object.

Key-value pair - An object's property consisting of a key and its associated value.

Dot Notation

key-value pair

```
house.color = "blue"
```

object **key** **value**

Bracket Notation

key-value pair

```
house["color"] = "blue"
```

object **key** **value**

Object Literal

```
const house = {      ← object
  color: "blue"      ← key-value pair
};
```

Factory Function (w/parameter)

```
const createHouse = function (color) {
  const house = {    ← object
    color: color     ← key-value pair
  };
```

```
    return house;
};
```

Bracket notation - A syntax to access or assign the property of an object using square brackets around the between property. Add quotation marks around the property name inside the square brackets.

```
const cat = {
  name: "Fred",
  nickname: "Flooficus",
  age: 5,
  isSleeping: true,
  favoriteToys: ["spring", "ping pong balls", "bird stuffy"],
  pet: function () {
    return "purrrrrrrr";
  }
};

cat["color"] = "orange";

console.log(cat["isSleeping"]);
// true
```

Object literal - A collection of key-value pairs inside the object's curly braces, separated by a comma. The key and value are separated by a colon (:). You can add or change existing properties of an object literal by using either dot or bracket notation and the `=` assignment operator.

```
const cat = {
  name: "Fred",
  nickname: "Flooficus",
  age: 5,
  isSleeping: true,
  favoriteToys: ["spring", "ping pong balls", "bird stuffy"],
  pet: function () {
    return "purrrrrrrr";
  }
};
```

```
cat.isSleeping = false;
cat["color"] = "orange";

console.log(cat);
// {name: "Fred", nickname: "Flooficus", age: 5, isSleeping: false,
favoriteToys: Array(3)...}
```

this keyword - In a method, the `this` keyword allows you to reference another property from the same object.

An example of the `this` keyword used with a method that's outside the object declaration:

```
const house = {
  windows: 20
};

house.windowWash = function () {
  if (this.windows > 15) {
    return `That's a lot of windows to wash!`;
  }
};

console.log(house.windowWash());
//That's a lot of windows to wash!
```

Here's an example of `this` keyword used with a method that's declared in an object literal:

```
const house = {
  windows: 20,
  windowWash: function () {
    if (this.windows > 15);
    return `That's a lot of windows to wash!`;
  }
};

console.log(house.windowWash());
//That's a lot of windows to wash!
```

Compound assignment operators - An assignment operator that combines the

assignment operator (=) with an arithmetic operator (+, -, *, /, and %). Compound assignment operators provide a shorter, cleaner syntax for performing calculations. See the JavaScript Operators Cheatsheet (Lesson 3) for a full list of assignment operators.

```
let paperclips = 10;
paperclips += 2;
console.log(paperclips);
// 12
```

```
let candy = 15;
candy %= 6;
console.log(`There's ${candy} candies leftover.`);
// There's 3 candies leftover.
```

Back to Top ↑

Lesson 11 - Factory Functions

Factory functions - Patterns to create multiple objects. Factory functions let you quickly build several objects that share the same characteristics, AKA properties. You must return your object at the bottom of your factory function. You'll use factory functions when you want to create and manage multiple objects that have the same characteristics (e.g., color) that are expressed differently (e.g., blue, green, yellow).

```
const createContact = function () {
  const contact = {
    name: "Noelle Silva",
    phoneNum: "555-555-1234",
    isNew: true,
    message: function () {
      this.isNew = true;
      console.log("You've added a new contact!");
    }
  };
  return contact;
};
```

```
console.log(createContact());
// {name: "Noelle Silva", phoneNum: "555-555-1234", isNew: true, message: f
message()}
```

You can provide parameters to your factory function in order to make your object more flexible and easy to reuse:

```
const createContact = function (name, phone) {
  const contact = {
    name: name,
    phoneNum: phone,
    isNew: true,
    message: function () {
      this.isNew = true;
      console.log("You've added a new contact!");
    }
  };
  return contact;
};

const contact1 = createContact("Noelle Silva", "555-555-1234");
const contact2 = createContact("Yami Sukehiro", "555-321-5555");

console.log(contact1, contact2);
// {name: "Noelle Silva", phoneNum: "555-555-1234", isNew: true, message: f
message()}
// {name: "Yami Sukehiro", phoneNum: "555-321-5555", isNew: true, message:
f message()}
```

for...in loop - A type of `for` loop that will allow you to loop over an object's key-value pairs. When looping over objects, you may want to access just the keys, just the values, or both the keys and the values.

```
const createContact = function (name, phone) {
  const contact = {
    name: name,
    phoneNum: phone,
    isNew: true,
    message: function () {
```

```
      this.isNew = true;
      console.log("You've added a new contact!");
    }
  };
  return contact;
};

const contact1 = createContact("Noelle Silva", "555-555-1234");

for (let key in contact1) {
  console.log(key, contact1[key]);
}
// name  Noelle Silva
// phoneNum  555-555-1234
// isNew true
// message ƒ message() {}
```

To loop through multiple objects, add the objects to an array and then loop through the array using the for...of loop (Lesson 8). After the `for...of` loop, nest the `for...in` loop to access the object's key, value, or keys and values.

```
const contact1 = createContact("Noelle Silva", "555-555-1234");
const contact2 = createContact("Yami Sukehiro", "555-321-5555");

const contactsArray = [contact1, contact2];

for (let contact of contactsArray) {
  for (let key in contact) {
    console.log(key, contact[key]);
  }
}
// name  Noelle Silva
// phoneNum  555-555-1234
// isNew true
// message ƒ message() {}
// name  Yami Sukehiro
// phoneNum  555-321-5555
// isNew true
// message ƒ message() {}
```

Back to Top ↑

Lesson 12 - Intro to APIs

API - An Application Programming Interface (API) is a way to allow information from an internal or external source to interact with your program.

API URL - The address to get access to the API. The API developers determine the API URL and associated endpoints and parameters, which can be found in the API's documentation. To access specific data from the API, you'll need the API URL combined with endpoints and possibly parameters.

Example API URLs:

- https://quote-garden.herokuapp.com/api/v3/
- https://api.tvmaze.com/

JSON files - A type of text file used for exchanging data. Most programming languages can interpret JSON files. JSON stands for JavaScript Object Notation. JSON files end with a .json file extension. Install an extension on your browser, JSON Formatter, to reformat JSON data and make it easier to read.

REST APIs - A type of API for making use of HTTP requests. You'll use the REST API's documentation to discover the API URL, endpoints, and parameters.

Endpoints - The "end" of the API URL that determines the type of information available.

- Example API endpoint for all quotes: https://quote-garden.herokuapp.com/api/v3/**quotes**

- Example API endpoint for a subset of quotes: https://quote-garden.herokuapp.com/api/v3/**quotes/random**

API parameters - Placeholders for data in the API URL. You'll add a question mark (?) between the endpoint and the parameters. If the parameter has more than one word,

replace the space between the words with the "%20" character. To chain more than one parameter together, add the ampersand (&) sign between the parameters.

Example API URL with a single parameter:
https://quote-garden.herokuapp.com/api/v3/quotes?**author=maya%20angelou**

Example API URL with multiple parameters (separated by a "&" sign):
https://quote-garden.herokuapp.com/api/v3/quotes?**author=maya%20angelou&limit=1**

fetch() - A method to allow you to get resources over a network, like data from an API.

async keyword - A keyword to enable asynchronous communication between your program and the API.

await keyword - A keyword that tells the program to wait on that line in the function until the API data are received.

json() - A method to parse (interpret) the JSON data from the API call and transform it into a JavaScript object.

```
const getData = async function () {
  const res = await fetch(
    "https://quote-garden.herokuapp.com/api/v3/quotes?author=beyonce"
  );
  const data = await res.json();
  console.log(data);
};
getData();
```

```
const getShows = async function () {
  const showRequest = await fetch("https://api.tvmaze.com/schedule/web");
  const data = await showRequest.json();
  console.log(data);
};
getShows()
```

```
// {61) [Object, Object, Object, Object, Object, Object, Object, Object,
Object, Object, ...]
```

Back to Top ↑

Lesson 13 - Project: Guess the Word Game

Decomposition - A computer science term that means breaking down a larger problem into smaller problems. Decomposition makes tackling a large project easier by breaking it into smaller problems that need to be solved.

Regular expression - A sequence of characters that lets you find text that matches a specific pattern. You'll use a regular expression when searching or replacing text. Regular expressions are also called "regex" or "regexp" for short. To learn more about regular expressions, check out this JavaScript Regex article.

match() - A method used with a regular expression to search the strings and match them to the regular expression.

```
const str = 'CanyoufindthesecretchocolatesnackIhaveinthislongstring';
const snackMatch = str.match(/chocolate/);

if (snackMatch) {
  console.log("Found the chocolate!")
};
// Found the chocolate!
```

Delimiter - A character to separate words in a string.

\n - A delimiter to create a line break (AKA newline).

```
console.log("First,\nsecond,\nand third!");
// First
// second,
// and third!
```

trim() - A method to remove extra whitespace before and after a string.

```
var happiness = "    Happiness is bug-free code.     ";

console.log(happiness);
//     Happiness is bug-free code.

console.log(happiness.trim());
// Happiness is bug-free code.
```

Back to Top ↑

Lesson 14 - Projects: GitHub Repo Gallery

---

matches() - A method to check if the target element (i.e., where the user clicks on the page) matches a specific selector.

```
const h2 = document.querySelectorAll('h2');
for (let heading of h2){
    if (heading.matches(".highlight")){
        heading.style.backgroundColor = "yellow";
    }
}
```

Input event - An event triggered when the value of the <input> element changes, like when a user inputs text in the search box. Inside the callback function for an input event, you'll pass a parameter that will hold the data for the text input. Most coders use `e` as the parameter to represent "event."

```
const namefield = document.querySelector("input.name");

namefield.addEventListener("input", function(e) {
  console.log(e.target.value)
}
```

toLowerCase() - A method for converting a string value into all lowercase letters.

```
var quiet = "PLEASE Lower Your Voice"

console.log(quiet.toLowerCase());
// please lower your voice
```

Back to Top ↑