#_ the JavaScript Ultimate CheatSheet

💠 Data Types

├ 📋 `Number` : Represents numeric values (integers and floats).
├ 📋 `String` : Represents textual data.
├ 📋 `Boolean` : Represents true or false values.
├ 📋 `Null` : Represents the intentional absence of any object value.
├ 📋 `Undefined` : Represents a variable that has been declared but has not been assigned a value.
└ 📋 `Object` : Represents a collection of key-value pairs.

💠 Variables & Constants

├ 📦 `var` : Function-scoped variable declaration (ES5).
├ 📦 `let` : Block-scoped variable declaration (ES6).
├ 📦 `const` : Block-scoped constant declaration (ES6).

💠 Operators

├ ➕ `Arithmetic` : +, -, *, /, % (remainder).
├ 📈 `Assignment` : =, +=, -=, *=, /=, %=
├ 🔄 `Increment/Decrement` : ++, --
├ ⚖️ `Comparison` : ==, ===, !=, !==, <, >, <=, >=
├ ⚛️ `Logical` : && (AND), || (OR), ! (NOT)
├ 🌟 `Ternary` : condition ? expr1 : expr2

💠 Control Flow

├ 🔄 `if` : Executes a statement if a condition is true.
├ 🔄 `else` : Executes a statement if the 'if' condition is false.
├ 🔄 `else if` : Executes a statement if another 'if' condition is true.
├ 🔁 `for` : Loops through a block of code a specified number of times.

├ 🔁 `while` : Loops through a block of code while a condition is true.
├ 🔁 `do...while` : Loops through a block of code at least once before checking the condition.
└ 🔲 `switch` : Selects one of many blocks of code to be executed.

💠 **Functions**
├ 📑 `Function Declaration` : function functionName(parameters) { ... }
├ 📑 `Function Expression` : const functionName = function(parameters) { ... }
├ 📑 `Arrow Function` : (parameters) => { ... }
├ 📑 `Default Parameters` : function functionName(param = defaultValue) { ... }
├ 📑 `Rest Parameters` : function functionName(...args) { ... }
├ 📑 `Immediately Invoked Function Expression (IIFE)` : (function() { ... })()
└ 📑 `Higher-Order Functions` : Functions that take other functions as arguments or return functions.

💠 **Arrays**
├ 📚 `Creation` : const arr = [elem1, elem2, ...];
├ 📚 `Accessing Elements` : arr[index]
├ 📚 `Adding Elements` : arr.push(elem), arr.unshift(elem)
├ 📚 `Removing Elements` : arr.pop(), arr.shift()
├ 📚 `Slicing` : arr.slice(startIndex, endIndex)
├ 📚 `Spreading` : const newArray = [...arr]
├ 📚 `Iterating` : arr.forEach(callback), arr.map(callback), arr.filter(callback)
└ 📚 `Reducing` : arr.reduce(callback, initialValue)

💠 Objects

├ 🔑 `Creating Objects` : const obj = { key: value, ... }
├ 🔑 `Accessing Properties` : obj.key or obj['key']
├ 🔑 `Adding Properties` : obj.newKey = value
├ 🔑 `Deleting Properties` : delete obj.key
├ 🔑 `Object Methods` : Methods defined within an object.
├ 🔑 `Object Destructuring` : const { key1, key2 } = obj;
└ 🔑 `Object Spread` : const newObj = { ...obj, newKey: newValue }

💠 Strings

├ 📝 `Length` : str.length
├ 📝 `Indexing` : str[index]
├ 📝 `Substring` : str.substring(startIndex, endIndex)
├ 📝 `Split` : str.split(separator)
├ 📝 `Trim` : str.trim()
├ 📝 `Concatenate` : str.concat(str1, str2, ...)
└ 📝 `Template Literal` : `Hello, ${name}!`

💠 Promises & Async/Await

├ ⏳ `Callbacks` : A function passed as an argument to another function to be executed later.
├ ⏳ `Callback Hell` : Nested and unreadable code due to excessive use of callbacks.
├ 🚀 `Promise` : An object representing the eventual completion or failure of an asynchronous operation.
├ 🚀 `Promise States` : Pending, Fulfilled, Rejected
├ 🚀 `Promise Methods` : .then(), .catch(), .finally()
├ ⏳ `Chaining Promises` : Using .then() to chain multiple asynchronous operations.
├ ⏳ `Promise.all` : Resolves an array of promises and returns a new promise that resolves to an array of resolved values.
├ ⏳ `Promise.race` : Resolves or rejects as soon as one of the promises in an iterable resolves or rejects.

├ 🚀 `Async/Await` : A syntax to write asynchronous code that looks like synchronous code.
├ ⏳ `try...catch with Async/Await` : Handling errors in asynchronous code.
└ 🚀 `Async Function` : An asynchronous function that always returns a Promise.

💠 **Modules & Imports**
├ 📦 `Module Exports` : export const funcName = () => { ... }
├ 📦 `Named Exports` : export { func1, func2 }
├ 📦 `Default Exports` : export default funcName
├ 📦 `Importing Modules` : import { funcName } from './module.js'
└ 📦 `Importing Default` : import funcName from './module.js'

💠 **Error Handling**
├ 🔍 `try...catch` : Catches errors in a block of code.
├ 🔍 `throw` : Throws a custom error.
└ 🔍 `Error Object` : new Error('Error message')

💠 **Event Handling**
├ 🎉 `addEventListener` : Attaches an event handler to an element.
├ 🎉 `Event Object` : Contains information about the event.
├ 🎉 `Event Propagation` : Bubbling & Capturing.
└ 🎉 `Preventing Default` : event.preventDefault()

💠 **DOM Manipulation**
├ 🖌️ `getElementById` : Retrieves an element by its id.
├ 🖌️ `getElementsByClassName` : Retrieves elements by their class name.
├ 🖌️ `getElementsByTagName` : Retrieves elements by their tag name.

├ 🖌️ `querySelector` : Retrieves the first element that matches a specified CSS selector.
├ 🖌️ `querySelectorAll` : Retrieves all elements that match a specified CSS selector.
└ 🖌️ `Creating Elements` : document.createElement(tagName)

💠 **AJAX & Fetch API**
├ 🌐 `XMLHttpRequest` : Making asynchronous HTTP requests.
├ 🌐 `Fetch API` : A modern alternative to XMLHttpRequest for making network requests.
└ 🌐 `Async/Await with Fetch` : Making asynchronous network requests with fetch.

💠 **Local Storage**
├ 💾 `setItem` : Stores data in local storage.
├ 💾 `getItem` : Retrieves data from local storage.
└ 💾 `removeItem` : Removes data from local storage.

💠 **Web APIs**
├ 🌍 `Geolocation API` : Retrieves the user's geographic location.
├ 🌍 `Notification API` : Displays desktop notifications.
├ 🌍 `Canvas API` : Draws graphics on a web page.
├ 🌍 `Audio & Video API` : Controls audio and video playback.
├ 🌍 `WebSockets API` : Enables real-time communication between clients and servers.
└ 🌍 `Service Workers` : Enables progressive web app features like offline support.

💠 **Error & Debugging Tools**
  ├ 🐞 **`console.log`** : Outputs a message to the console.
  ├ 🐞 **`console.warn`** : Outputs a warning message to the console.
  ├ 🐞 **`console.error`** : Outputs an error message to the console.
  ├ 🐞 **`debugger`** : Pauses the execution of code and opens the browser's debugger.
  └ 🐞 **`DevTools`** : Browser developer tools for inspecting and debugging.

💠 **Regular Expressions (Regex)**
  ├ 🔍 **`Creation`** : const regex = /pattern/modifiers;
  ├ 🔍 **`Test`** : regex.test(str)
  ├ 🔍 **`Match`** : str.match(regex)
  ├ 🔍 **`Modifiers`** : g (global), i (case-insensitive), m (multiline)
  ├ 🔍 **`Character Classes`** : \d (digit), \w (word), \s (whitespace), ...
  ├ 🔍 **`Quantifiers`** : + (one or more), * (zero or more), ? (zero or one), {n} (exactly n times), {n,} (n or more), {n,m} (between n and m times)
  └ 🔍 **`Groups and Capturing`** : (group), (?:non-capturing group), /(pattern)/ (capturing group)

💠 **Unit Testing**
  ├ 🧪 **`Jest`** : A popular JavaScript testing framework.
  ├ 🧪 **`describe`** : Groups test cases.
  ├ 🧪 **`it`** : Defines a test case.
  ├ 🧪 **`expect`** : Defines assertions for test validation.
  └ 🧪 `mock` : Creates mock functions and modules for testing.

By: Waleed Mousa

💠 **ES6+ Features**

├ 🌟 `Destructuring` : const { key } = obj;
├ 🌟 `Spread Operator` : const newArray = [...arr];
├ 🌟 `Rest Parameters` : function functionName(...args) { ... }
├ 🌟 `Arrow Functions` : (parameters) => { ... }
├ 🌟 `Classes` : class ClassName { ... }
└ 🌟 `Modules` : export, import

💠 **Web Development Libraries & Frameworks**

├ 🧱 `React.js` : A JavaScript library for building user interfaces.
├ 🧱 `Angular` : A TypeScript-based web application framework.
├ 🧱 `Vue.js` : A progressive JavaScript framework for building user interfaces.
└ 🧱 `jQuery` : A fast, small, and feature-rich JavaScript library.

💠 **JavaScript Design Patterns**

├ 🧩 `Singleton` : Ensures only one instance of a class is created and provides a global point of access to it.
├ 🧩 `Observer` : Allows an object to publish changes to its state to other objects.
├ 🧩 `Factory` : Creates objects without specifying the exact class of the object that will be created.
├ 🧩 `Decorator` : Dynamically adds behavior to objects at runtime.
├ 🧩 `Adapter` : Converts the interface of a class into another interface that clients expect.
├ 🧩 `Facade` : Provides a unified interface to a set of interfaces in a subsystem.
└ 🧩 `Command` : Encapsulates a request as an object, allowing for parameterization of clients with different requests, queuing of requests, and logging of the requests.

💠 **Resources**

├ 📖 **`MDN Web Docs`** : Official Mozilla Developer Network JavaScript documentation.

├ 📖 **`w3schools`** : Online tutorials and reference materials for web development.

├ 📖 **`JavaScript.info`** : Modern JavaScript tutorials and reference.

├ 📖 **`Eloquent JavaScript`** : A comprehensive JavaScript book by Marijn Haverbeke.

├ 📺 **Traversy Media :** Comprehensive web development tutorials by Brad Traversy. (Link: https://www.youtube.com/user/TechGuyWeb)

├ 📺 **The Net Ninja :** Web development tutorials with a focus on JavaScript and frameworks. (Link: https://www.youtube.com/c/TheNetNinja)

├ 📺 **freeCodeCamp.org :** Covers a wide range of topics, including JavaScript and frontend development. (Link: https://www.youtube.com/c/Freecodecamp)

├ 📺 **Fireship :** Short and to-the-point JavaScript tips and tricks. (Link: https://www.youtube.com/c/Fireship)

├ 📺 **Programming with Mosh :** Practical JavaScript and web development tutorials. (Link: https://www.youtube.com/user/programmingwithmosh)

├ 📺 **Academind :** Web development tutorials, including JavaScript and frameworks. (Link: https://www.youtube.com/c/Academind)

├ 📺 **The Coding Train :** Creative coding tutorials, including JavaScript and p5.js. (Link: https://www.youtube.com/c/TheCodingTrain)

├ 📺 **LevelUpTuts :** Covers various frontend technologies, including JavaScript. (Link: https://www.youtube.com/c/LevelUpTuts)

└ 📺 **Codevolution :** JavaScript and frontend development tutorials. (Link: https://www.youtube.com/c/Codevolution)

By: Waleed Mousa