[The Ultimate JavaScript CheatSheet]

1. Variables and Data Types

- Declare a variable: `let x;`
- Declare and initialize a variable: `let x = 5;`
- Declare a constant: `const PI = 3.14159;`
- Declare a variable with block scope: `let x = 10;`
- Declare a variable with function scope: `var y = 20;`
- Number: `let num = 42;`
- String: `let str = "Hello, World!";`
- Boolean: `let isTrue = true;`
- Undefined: `let x;`
- Null: `let y = null;`
- Symbol: `let sym = Symbol("description");`
- BigInt: `let bigNum = 1234567890123456789012345678901234567890n;`
- Object: `let obj = {key: "value"};`
- Array: `let arr = [1, 2, 3];`
- Function: `let func = function() {};`
- Check type of variable: `typeof variable`
- Check if variable is an array: `Array.isArray(variable)`
- Convert to number: `Number(value)`
- Convert to string: `String(value)`
- Convert to boolean: `Boolean(value)`
- Parse integer: `parseInt("42")`
- Parse float: `parseFloat("3.14")`
- Check if value is NaN: `isNaN(value)`
- Check if value is finite: `isFinite(value)`
- Get positive infinity: `Infinity`
- Get negative infinity: `-Infinity`

2. Operators

- Addition: `let sum = a + b;`
- Subtraction: `let diff = a - b;`
- Multiplication: `let product = a * b;`
- Division: `let quotient = a / b;`
- Modulus: `let remainder = a % b;`
- Exponentiation: `let power = a ** b;`
- Increment: `x++;` or `++x;`

By: Waleed Mousa

- Decrement: `x--;` or `--x;`
- Unary plus: `let num = +x;`
- Unary negation: `let negNum = -x;`
- Logical AND: `let result = a && b;`
- Logical OR: `let result = a || b;`
- Logical NOT: `let result = !a;`
- Nullish coalescing: `let result = a ?? b;`
- Optional chaining: `let value = obj?.prop?.method?.();`
- Equality: `let isEqual = a == b;`
- Strict equality: `let isStrictEqual = a === b;`
- Inequality: `let isNotEqual = a != b;`
- Strict inequality: `let isStrictNotEqual = a !== b;`
- Greater than: `let isGreater = a > b;`
- Less than: `let isLess = a < b;`
- Greater than or equal: `let isGreaterOrEqual = a >= b;`
- Less than or equal: `let isLessOrEqual = a <= b;`
- Ternary operator: `let result = condition ? trueValue : falseValue;`
- Bitwise AND: `let result = a & b;`
- Bitwise OR: `let result = a | b;`
- Bitwise XOR: `let result = a ^ b;`
- Bitwise NOT: `let result = ~a;`
- Left shift: `let result = a << b;`
- Sign-propagating right shift: `let result = a >> b;`
- Zero-fill right shift: `let result = a >>> b;`
- Assignment: `x = y`
- Addition assignment: `x += y`
- Subtraction assignment: `x -= y`
- Multiplication assignment: `x *= y`
- Division assignment: `x /= y`
- Remainder assignment: `x %= y`
- Exponentiation assignment: `x **= y`
- Left shift assignment: `x <<= y`
- Right shift assignment: `x >>= y`
- Unsigned right shift assignment: `x >>>= y`
- Bitwise AND assignment: `x &= y`
- Bitwise XOR assignment: `x ^= y`
- Bitwise OR assignment: `x |= y`
- Logical AND assignment: `x &&= y`
- Logical OR assignment: `x ||= y`
- Nullish coalescing assignment: `x ??= y`

By: Waleed Mousa

3. Control Flow

- If statement: `if (condition) { }`
- If-else statement: `if (condition) { } else { }`
- If-else if-else statement: `if (condition1) { } else if (condition2) { } else { }`
- Switch statement:

```
switch(expression) {
  case x:
     // code block
     break;
  case y:
     // code block
     break;
  default:
     // code block
}
```

- For loop: `for (let i = 0; i < 10; i++) { }`
- While loop: `while (condition) { }`
- Do-while loop: `do { } while (condition);`
- For...in loop (objects): `for (let key in object) { }`
- For...of loop (iterables): `for (let value of iterable) { }`
- Break statement: `break;`
- Continue statement: `continue;`
- Labeled statement: `label: statement`
- Try-catch: `try { } catch (error) { }`
- Try-catch-finally: `try { } catch (error) { } finally { }`
- Throw an error: `throw new Error("message");`
- Conditional (ternary) operator: `condition ? expr1 : expr2`
- Short-circuit evaluation: `expr1 && expr2`
- Nullish coalescing operator: `expr1 ?? expr2`
- Optional chaining: `obj?.prop?.method?.()`

4. Functions

- Function declaration: `function name(params) { }`
- Function expression: `let func = function(params) { };`
- Arrow function: `let func = (params) => { };`
- Immediately Invoked Function Expression (IIFE): `(function() { })();`
- Function with default parameters: `function name(param = defaultValue) { }`

By: Waleed Mousa

- Rest parameters: `function name(...args) { }`
- Spread operator in function call: `func(...array);`
- Closure: `function outer() { let x = 10; return function inner() { return x; }; }`
- Currying: `let curriedFunc = a => b => a + b;`
- Generator function: `function* generator() { yield 1; yield 2; }`
- Async function: `async function name() { }`
- Function as object property: `let obj = { method() { } };`
- Getter: `let obj = { get propName() { } };`
- Setter: `let obj = { set propName(value) { } };`
- Bind method: `let boundFunc = func.bind(thisArg, arg1, arg2);`
- Call method: `func.call(thisArg, arg1, arg2);`
- Apply method: `func.apply(thisArg, [arg1, arg2]);`
- Function length property: `func.length`
- Function name property: `func.name`
- Check if value is function: `typeof value === 'function'`
- Higher-order function: `function higherOrder(callback) { callback(); }`
- Pure function: `function pure(x) { return x * 2; }`
- Recursive function: `function factorial(n) { return n <= 1 ? 1 : n * factorial(n - 1); }`
- Memoization: `javascript function memoize(fn) { const cache = {}; return function(...args) { const key = JSON.stringify(args); if (key in cache) { return cache[key]; } const result = fn.apply(this, args); cache[key] = result; return result; } }`

5. Objects

- Object literal: `let obj = {key: "value"};`
- Accessing object properties (dot notation): `obj.key`
- Accessing object properties (bracket notation): `obj["key"]`
- Adding a property: `obj.newKey = "value";`
- Deleting a property: `delete obj.key;`
- Object.keys(): `let keys = Object.keys(obj);`
- Object.values(): `let values = Object.values(obj);`
- Object.entries(): `let entries = Object.entries(obj);`
- Object destructuring: `let {key1, key2} = obj;`
- Shallow clone object: `let clone = {...obj};`
- Deep clone object: `let clone = JSON.parse(JSON.stringify(obj));`
- Merge objects: `let merged = {...obj1, ...obj2};`
- Object.freeze(): `Object.freeze(obj);`

By: Waleed Mousa

- Object.seal(): `Object.seal(obj);`
- Object.is(): `let isSame = Object.is(value1, value2);`
- Create object with prototype: `let obj = Object.create(protoObj);`
- Get object prototype: `Object.getPrototypeOf(obj);`
- Set object prototype: `Object.setPrototypeOf(obj, protoObj);`
- Define property: `Object.defineProperty(obj, 'key', { value: 42, writable: false });`
- Define multiple properties: `Object.defineProperties(obj, { prop1: {}, prop2: {} });`
- Get property descriptor: `Object.getOwnPropertyDescriptor(obj, 'key');`
- Get all property descriptors: `Object.getOwnPropertyDescriptors(obj);`
- Prevent extensions: `Object.preventExtensions(obj);`
- Check if object is extensible: `Object.isExtensible(obj);`
- Check if object is sealed: `Object.isSealed(obj);`
- Check if object is frozen: `Object.isFrozen(obj);`
- Get own property names: `Object.getOwnPropertyNames(obj);`
- Get own property symbols: `Object.getOwnPropertySymbols(obj);`
- Check if object has property: `obj.hasOwnProperty('key')`
- Object method shorthand: `let obj = { method() { } };`
- Computed property names: `let obj = { [expression]: value };`
- Object.assign(): `let assigned = Object.assign(target, source1, source2);`
- Object.fromEntries(): `let obj = Object.fromEntries([['key1', 'value1'], ['key2', 'value2']]);`

6. Arrays

- Array literal: `let arr = [1, 2, 3];`
- Array constructor: `let arr = new Array(1, 2, 3);`
- Accessing array elements: `let element = arr[0];`
- Setting array elements: `arr[0] = 10;`
- Array length: `let length = arr.length;`
- Push element to array: `arr.push(element);`
- Pop element from array: `let lastElement = arr.pop();`
- Unshift element to array: `arr.unshift(element);`
- Shift element from array: `let firstElement = arr.shift();`
- Slice array: `let subArray = arr.slice(start, end);`
- Splice array: `arr.splice(start, deleteCount, item1, item2, ...);`
- Join array elements: `let str = arr.join(separator);`
- Reverse array: `arr.reverse();`

By: Waleed Mousa

- Sort array: `arr.sort((a, b) => a - b);`
- Find element in array: `let found = arr.find(element => condition);`
- Find index of element: `let index = arr.findIndex(element => condition);`
- Filter array: `let filtered = arr.filter(element => condition);`
- Map array: `let mapped = arr.map(element => transformation);`
- Reduce array: `let result = arr.reduce((accumulator, currentValue) => operation, initialValue);`
- Reduce array right-to-left: `let result = arr.reduceRight((accumulator, currentValue) => operation, initialValue);`
- Every (all elements satisfy condition): `let allSatisfy = arr.every(element => condition);`
- Some (at least one element satisfies condition): `let someSatisfy = arr.some(element => condition);`
- ForEach: `arr.forEach(element => operation);`
- Includes: `let includes = arr.includes(element);`
- IndexOf: `let index = arr.indexOf(element);`
- LastIndexOf: `let lastIndex = arr.lastIndexOf(element);`
- Fill array: `arr.fill(value, start, end);`
- Flatten array: `let flattened = arr.flat(depth);`
- FlatMap: `let flatMapped = arr.flatMap(element => operation);`
- Array from iterable: `let arrFromIterable = Array.from(iterable);`
- Array.of: `let arr = Array.of(1, 2, 3);`
- Array.isArray: `let isArray = Array.isArray(arr);`
- Spread operator: `let newArr = [...arr];`
- Destructuring assignment: `let [a, b, ...rest] = arr;`
- Concat arrays: `let newArr = arr1.concat(arr2, arr3);`
- Copying array: `let copy = arr.slice();`
- Clear array: `arr.length = 0;`
- Remove falsy values: `arr = arr.filter(Boolean);`
- Get unique values: `let unique = [...new Set(arr)];`
- Get max value: `let max = Math.max(...arr);`
- Get min value: `let min = Math.min(...arr);`
- Sum of array: `let sum = arr.reduce((a, b) => a + b, 0);`
- Average of array: `let avg = arr.reduce((a, b) => a + b, 0) / arr.length;`
- Shuffle array: `arr.sort(() => Math.random() - 0.5);`
- Check if array is empty: `arr.length === 0`
- Create array of numbers: `let numbers = Array.from({length: 5}, (_, i) => i + 1);`

By: Waleed Mousa

7. Strings

- String literal: `let str = "Hello, World!";`
- String object: `let strObj = new String("Hello");`
- String length: `let length = str.length;`
- Accessing characters: `let char = str[0];`
- Substring: `let sub = str.substring(start, end);`
- Slice string: `let sliced = str.slice(start, end);`
- Split string: `let arr = str.split(separator);`
- Concatenate strings: `let newStr = str1.concat(str2);`
- Trim whitespace: `let trimmed = str.trim();`
- Trim start: `let trimmedStart = str.trimStart();`
- Trim end: `let trimmedEnd = str.trimEnd();`
- To uppercase: `let upper = str.toUpperCase();`
- To lowercase: `let lower = str.toLowerCase();`
- Replace: `let replaced = str.replace(searchValue, replaceValue);`
- Replace all: `let replacedAll = str.replaceAll(searchValue, replaceValue);`
- Includes: `let includes = str.includes(searchString);`
- StartsWith: `let startsWith = str.startsWith(searchString);`
- EndsWith: `let endsWith = str.endsWith(searchString);`
- IndexOf: `let index = str.indexOf(searchString);`
- LastIndexOf: `let lastIndex = str.lastIndexOf(searchString);`
- Char at index: `let char = str.charAt(index);`
- Char code at index: `let charCode = str.charCodeAt(index);`
- Repeat string: `let repeated = str.repeat(count);`
- Pad start: `let padded = str.padStart(targetLength, padString);`
- Pad end: `let padded = str.padEnd(targetLength, padString);`
- Match: `let matches = str.match(regexp);`
- Match all: `let matchesIterator = str.matchAll(regexp);`
- Search: `let index = str.search(regexp);`
- LocaleCompare: `let result = str1.localeCompare(str2);`
- FromCharCode: `let str = String.fromCharCode(65, 66, 67);`
- FromCodePoint: `let str = String.fromCodePoint(65, 66, 67);`
- Raw: `let raw = String.rawtemplateString;`
- Normalize: `let normalized = str.normalize();`
- Template literals: `let greeting = `Hello, ${name}!`;`
- Tagged template literals: `function tag(strings, ...values) { }`

8. ES6+ Features

- Let and const: `let x = 5; const y = 10;`
- Arrow functions: `let add = (a, b) => a + b;`
- Default parameters: `function greet(name = "World") { }`
- Rest parameters: `function sum(...numbers) { }`
- Spread operator (array): `let newArr = [...arr1, ...arr2];`
- Spread operator (object): `let newObj = {...obj1, ...obj2};`
- Destructuring assignment (array): `let [a, b] = [1, 2];`
- Destructuring assignment (object): `let {x, y} = {x: 1, y: 2};`
- Enhanced object literals: `let obj = {x, y, method() {}};`
- Template literals: `let greeting = `Hello, ${name}!`;`
- Multi-line strings: `let multiline = `Line 1 Line 2`;`
- Symbol: `let sym = Symbol("description");`
- Iterators: `let iterator = arr[Symbol.iterator]();`
- Generators: `function* generator() { yield 1; yield 2; }`
- Promise: `let promise = new Promise((resolve, reject) => { });`
- Async/Await: `async function fetchData() { let response = await fetch(url); }`
- Map: `let map = new Map();`
- Set: `let set = new Set();`
- WeakMap: `let weakMap = new WeakMap();`
- WeakSet: `let weakSet = new WeakSet();`
- Classes: `class ClassName { constructor() {} }`
- Class inheritance: `class Child extends Parent { }`
- Static methods: `static methodName() { }`
- Getters and setters: `get propertyName() { }` and `set propertyName(value) { }`
- Modules (export): `export { name1, name2 };`
- Modules (import): `import { name1, name2 } from "./module.js";`
- Default export: `export default expression;`
- Default import: `import defaultExport from "./module.js";`
- Dynamic import: `import("./module.js").then(module => { });`
- Object.assign(): `Object.assign(target, source1, source2);`
- Object.is(): `Object.is(value1, value2);`
- Array.from(): `Array.from(arrayLike, mapFn, thisArg);`
- Array.of(): `Array.of(1, 2, 3);`
- String.repeat(): `"abc".repeat(3);`
- String.startsWith(): `"Hello".startsWith("He");`
- String.endsWith(): `"World".endsWith("ld");`

By: Waleed Mousa

- String.includes(): `"Hello World".includes("Wor");`
- Number.isFinite(): `Number.isFinite(10);`
- Number.isNaN(): `Number.isNaN(NaN);`
- Number.isInteger(): `Number.isInteger(10);`
- Number.isSafeInteger(): `Number.isSafeInteger(Number.MAX_SAFE_INTEGER);`
- Math.trunc(): `Math.trunc(4.9);`
- Math.sign(): `Math.sign(-10);`
- Object.entries(): `Object.entries(obj);`
- Object.values(): `Object.values(obj);`
- Object.getOwnPropertyDescriptors(): `Object.getOwnPropertyDescriptors(obj);`
- Trailing commas in function parameters: `function f(a, b, ) { }`
- Async iterators: `for await (const x of asyncIterable) { }`
- RegExp named capture groups: `/(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/`
- RegExp lookbehind assertions: `/(?<=\$)\d+(\.\d*)?/`

9. DOM Manipulation

- Get element by ID: `let element = document.getElementById("id");`
- Get elements by class name: `let elements = document.getElementsByClassName("class");`
- Get elements by tag name: `let elements = document.getElementsByTagName("tag");`
- Query selector: `let element = document.querySelector("selector");`
- Query selector all: `let elements = document.querySelectorAll("selector");`
- Create element: `let element = document.createElement("tag");`
- Create text node: `let textNode = document.createTextNode("text");`
- Append child: `parent.appendChild(child);`
- Remove child: `parent.removeChild(child);`
- Replace child: `parent.replaceChild(newChild, oldChild);`
- Insert before: `parent.insertBefore(newNode, referenceNode);`
- Clone node: `let clone = node.cloneNode(deep);`
- Set attribute: `element.setAttribute("name", "value");`
- Get attribute: `let value = element.getAttribute("name");`
- Remove attribute: `element.removeAttribute("name");`
- Has attribute: `let hasAttr = element.hasAttribute("name");`
- Set inner HTML: `element.innerHTML = "content";`

By: Waleed Mousa

- Get inner HTML: `let content = element.innerHTML;`
- Set text content: `element.textContent = "text";`
- Get text content: `let text = element.textContent;`
- Add class: `element.classList.add("class");`
- Remove class: `element.classList.remove("class");`
- Toggle class: `element.classList.toggle("class");`
- Check if has class: `let hasClass = element.classList.contains("class");`
- Set style: `element.style.property = "value";`
- Get computed style: `let style = getComputedStyle(element);`
- Get bounding client rect: `let rect = element.getBoundingClientRect();`
- Get offset width: `let width = element.offsetWidth;`
- Get offset height: `let height = element.offsetHeight;`
- Get client width: `let width = element.clientWidth;`
- Get client height: `let height = element.clientHeight;`
- Scroll into view: `element.scrollIntoView(options);`
- Focus element: `element.focus();`
- Blur element: `element.blur();`
- Get parent element: `let parent = element.parentElement;`
- Get child elements: `let children = element.children;`
- Get first child element: `let firstChild = element.firstElementChild;`
- Get last child element: `let lastChild = element.lastElementChild;`
- Get next sibling element: `let nextSibling = element.nextElementSibling;`
- Get previous sibling element: `let prevSibling = element.previousElementSibling;`

10. Events

- Add event listener: `element.addEventListener("event", handler);`
- Remove event listener: `element.removeEventListener("event", handler);`
- Dispatch event: `element.dispatchEvent(new Event("event"));`
- Prevent default behavior: `event.preventDefault();`
- Stop event propagation: `event.stopPropagation();`
- Stop immediate propagation: `event.stopImmediatePropagation();`
- Get event target: `let target = event.target;`
- Get event current target: `let currentTarget = event.currentTarget;`
- Get event type: `let type = event.type;`
- Check if event bubbles: `let bubbles = event.bubbles;`
- Check if event cancelable: `let cancelable = event.cancelable;`

- Get event timestamp: `let timestamp = event.timeStamp;`
- Custom event: `let customEvent = new CustomEvent("eventName", { detail: {} });`
- Mouse event coordinates: `let x = event.clientX; let y = event.clientY;`
- Keyboard event key: `let key = event.key;`
- Keyboard event code: `let code = event.code;`
- Touch event touches: `let touches = event.touches;`
- Drag event dataTransfer: `let dataTransfer = event.dataTransfer;`
- Form event submit: `form.addEventListener("submit", (e) => { e.preventDefault(); });`
- Window load event: `window.addEventListener("load", handler);`
- Document ready event: `document.addEventListener("DOMContentLoaded", handler);`
- Window resize event: `window.addEventListener("resize", handler);`
- Window scroll event: `window.addEventListener("scroll", handler);`
- Mutation observer: `let observer = new MutationObserver(callback);`
- Intersection observer: `let observer = new IntersectionObserver(callback, options);`

11. AJAX and Fetch API

- XMLHttpRequest: `let xhr = new XMLHttpRequest();`
- XMLHttpRequest open: `xhr.open("GET", url, true);`
- XMLHttpRequest send: `xhr.send();`
- XMLHttpRequest onload: `xhr.onload = function() { };`
- XMLHttpRequest onerror: `xhr.onerror = function() { };`
- Fetch API: `fetch(url).then(response => response.json()).then(data => console.log(data));`
- Fetch with options: `fetch(url, { method: "POST", body: JSON.stringify(data) });`
- Fetch with headers: `fetch(url, { headers: { "Content-Type": "application/json" } });`
- Fetch abort: `let controller = new AbortController(); fetch(url, { signal: controller.signal });`
- Async/Await with Fetch: `let response = await fetch(url); let data = await response.json();`
- Axios get: `axios.get(url).then(response => console.log(response.data));`

- Axios post: `axios.post(url, data).then(response => console.log(response.data));`
- jQuery AJAX: `$.ajax({ url: url, method: "GET", success: function(data) { } });`

12. JSON

- Parse JSON: `let obj = JSON.parse(jsonString);`
- Stringify JSON: `let jsonString = JSON.stringify(obj);`
- Stringify with replacer: `JSON.stringify(obj, replacer);`
- Stringify with space: `JSON.stringify(obj, null, 2);`
- Parse with reviver: `JSON.parse(jsonString, reviver);`

13. Promises and Async/Await

- Create Promise: `let promise = new Promise((resolve, reject) => { });`
- Promise then: `promise.then(result => { });`
- Promise catch: `promise.catch(error => { });`
- Promise finally: `promise.finally(() => { });`
- Promise all: `Promise.all([promise1, promise2]).then(results => { });`
- Promise race: `Promise.race([promise1, promise2]).then(result => { });`
- Promise allSettled: `Promise.allSettled([promise1, promise2]).then(results => { });`
- Promise any: `Promise.any([promise1, promise2]).then(result => { });`
- Async function: `async function name() { }`
- Await: `let result = await promise;`
- Async/Await with try/catch: `try { let result = await promise; } catch (error) { }`

14. Web APIs

- Local Storage set item: `localStorage.setItem("key", "value");`
- Local Storage get item: `let value = localStorage.getItem("key");`
- Local Storage remove item: `localStorage.removeItem("key");`
- Local Storage clear: `localStorage.clear();`
- Session Storage set item: `sessionStorage.setItem("key", "value");`
- Cookies set: `document.cookie = "key=value; expires=Thu, 18 Dec 2023 12:00:00 UTC; path=/";`
- Cookies get: `let value = document.cookie.split('; ').find(row => row.startsWith('key=')).split('=')[1];`

By: Waleed Mousa

- Geolocation: `navigator.geolocation.getCurrentPosition(success, error, options);`
- Web Workers: `let worker = new Worker('worker.js');`
- Service Workers: `navigator.serviceWorker.register('/sw.js');`
- Notifications: `Notification.requestPermission().then(permission => { });`
- Push API: `registration.pushManager.subscribe(options);`
- Fetch API: `fetch(url).then(response => response.json());`
- Canvas API: `let ctx = canvas.getContext('2d');`
- WebGL: `let gl = canvas.getContext('webgl');`
- WebRTC: `let pc = new RTCPeerConnection();`

By: Waleed Mousa