Complete JavaScript Cheatsheet

Table of Contents

Variables & Data Types

Variable Declaration

```javascript
var oldWay = "Function/global scoped";
let blockScoped = "Block scoped, can be reassigned";
const constant = "Block scoped, cannot be reassigned";
```

Data Types

javascript

```javascript
// Primitives
let str = "Hello";          // String
let num = 42;               // Number
let bigInt = 123n;          // BigInt
let bool = true;            // Boolean
let undef = undefined;      // Undefined
let nul = null;             // Null
let sym = Symbol("id");     // Symbol

// Objects
let obj = { name: "John" }; // Object
let arr = [1, 2, 3];        // Array
let func = function() {};   // Function
let date = new Date();      // Date
let regex = /pattern/g;     // RegExp
```

Type Checking

javascript

```javascript
typeof "hello"      // "string"
typeof 42           // "number"
typeof true         // "boolean"
typeof undefined    // "undefined"
typeof null         // "object" (legacy bug)
typeof {}           // "object"
typeof []           // "object"
typeof function() {} // "function"

// Better type checking
Array.isArray([])   // true
obj instanceof Date  // true
```

Operators

Arithmetic

javascript

```
+   // Addition: 5 + 3 = 8
-   // Subtraction: 5 - 3 = 2
*   // Multiplication: 5 * 3 = 15
/   // Division: 15 / 3 = 5
%   // Modulo: 5 % 3 = 2
**  // Exponentiation: 2 ** 3 = 8
++  // Increment: x++ or ++x
--  // Decrement: x-- or --x
```

Assignment

javascript

```
=   // x = 5
+=  // x += 3 (same as x = x + 3)
-=  // x -= 3
*=  // x *= 3
/=  // x /= 3
%=  // x %= 3
**= // x **= 3
```

Comparison

javascript

```
==  // Equal (with type coercion): 5 == "5" is true
=== // Strict equal: 5 === "5" is false
!=  // Not equal (with type coercion)
!== // Strict not equal
>   // Greater than
<   // Less than
>=  // Greater than or equal
<=  // Less than or equal
```

Logical

javascript

```
&&  // AND: true && true = true
||  // OR: true || false = true
!   // NOT: !true = false
??  // Nullish coalescing: null ?? "default" = "default"
```

Ternary

javascript

```
condition ? trueValue : falseValue
let status = age >= 18 ? "adult" : "minor";
```

Control Flow

If/Else

javascript

```
if (condition) {
    // code
} else if (anotherCondition) {
    // code
} else {
    // code
}
```

Switch

javascript

```
switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
}
```

Loops

javascript

```javascript
// For loop
for (let i = 0; i < 5; i++) {
    console.log(i);
}

// While loop
while (condition) {
    // code
}

// Do-while loop
do {
    // code
} while (condition);

// For...of (iterates values)
for (const item of array) {
    console.log(item);
}

// For...in (iterates keys/indices)
for (const key in object) {
    console.log(key, object[key]);
}

// Break and continue
for (let i = 0; i < 10; i++) {
    if (i === 3) continue;  // Skip iteration
    if (i === 7) break;     // Exit loop
}
```

Functions

Function Declaration

javascript

```javascript
function add(a, b) {
    return a + b;
}
```

Function Expression

javascript

```javascript
const add = function(a, b) {
    return a + b;
};
```

Arrow Functions

javascript

```javascript
const add = (a, b) => a + b;
const square = x => x * x;
const greet = () => "Hello!";
const complex = (x, y) => {
    const sum = x + y;
    return sum * 2;
};
```

Parameters & Arguments

javascript

```javascript
// Default parameters
function greet(name = "Guest") {
    return `Hello, ${name}!`;
}

// Rest parameters
function sum(...numbers) {
    return numbers.reduce((a, b) => a + b, 0);
}

// Spread operator
const nums = [1, 2, 3];
console.log(Math.max(...nums));
```

Higher-Order Functions

javascript

```javascript
// Function as argument
function apply(func, value) {
    return func(value);
}

// Function returning function
function multiplier(factor) {
    return x => x * factor;
}
const double = multiplier(2);
```

Arrays

Array Methods

javascript

```javascript
const arr = [1, 2, 3, 4, 5];

// Mutating methods
arr.push(6);          // Add to end
arr.pop();            // Remove from end
arr.unshift(0);       // Add to beginning
arr.shift();          // Remove from beginning
arr.splice(2, 1, 99); // Remove/add elements
arr.reverse();        // Reverse in place
arr.sort();           // Sort in place

// Non-mutating methods
arr.concat([6, 7]);   // Combine arrays
arr.slice(1, 3);      // Extract portion
arr.join(', ');       // Join to string
arr.indexOf(3);       // Find index
arr.includes(3);      // Check existence

// Iteration methods
arr.forEach(x => console.log(x));
arr.map(x => x * 2);            // Transform
arr.filter(x => x > 2);         // Filter
arr.reduce((sum, x) => sum + x); // Reduce
arr.find(x => x > 3);           // Find first
arr.findIndex(x => x > 3);      // Find index
arr.some(x => x > 4);           // Any match
arr.every(x => x > 0);          // All match
```

Array Destructuring

javascript

```
const [a, b, ...rest] = [1, 2, 3, 4, 5];
const [x, , z] = [1, 2, 3];  // Skip elements
```

Objects

Object Creation

javascript

```
// Object literal
const person = {
    name: "John",
    age: 30,
    greet() {
        return `Hello, I'm ${this.name}`;
    }
};

// Constructor function
function Person(name, age) {
    this.name = name;
    this.age = age;
}

// Object.create()
const proto = { greet() { return "Hello"; } };
const obj = Object.create(proto);
```

Object Methods

javascript

```
const obj = { a: 1, b: 2, c: 3 };

Object.keys(obj);         // ["a", "b", "c"]
Object.values(obj);       // [1, 2, 3]
Object.entries(obj);      // [["a", 1], ["b", 2], ["c", 3]]
Object.assign({}, obj);   // Shallow copy
Object.freeze(obj);       // Make immutable
Object.seal(obj);         // Prevent add/delete
```

Object Destructuring

javascript

```
const { name, age } = person;
const { x: newName } = { x: "value" };  // Rename
const { a, ...rest } = { a: 1, b: 2, c: 3 };
```

Property Access

javascript

```
obj.property      // Dot notation
obj["property"]   // Bracket notation
obj?.property     // Optional chaining
```

Classes & OOP

Class Declaration

javascript

```
class Animal {
    constructor(name) {
        this.name = name;
    }

    speak() {
        return `${this.name} makes a sound`;
    }

    static compare(a, b) {
        return a.name === b.name;
    }
}

class Dog extends Animal {
    constructor(name, breed) {
        super(name);
        this.breed = breed;
    }

    speak() {
        return `${this.name} barks`;
    }
}
```

Getters & Setters

javascript

```
class Circle {
    constructor(radius) {
        this._radius = radius;
    }

    get area() {
        return Math.PI * this._radius ** 2;
    }

    set radius(value) {
        if (value < 0) throw new Error("Invalid radius");
        this._radius = value;
    }
}
```

ES6+ Features

Template Literals

javascript

```
const name = "John";
const message = `Hello, ${name}!`;
const multiline = `
    Line 1
    Line 2
`;
```

Destructuring

javascript

```
// Array destructuring
const [a, b] = [1, 2];

// Object destructuring
const { x, y } = { x: 1, y: 2 };

// Nested destructuring
const { a: { b } } = { a: { b: 1 } };
```

Spread & Rest

javascript

```javascript
// Spread
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];
const obj1 = { a: 1 };
const obj2 = { ...obj1, b: 2 };

// Rest
const [first, ...rest] = [1, 2, 3, 4];
const { a, ...others } = { a: 1, b: 2, c: 3 };
```

Optional Chaining & Nullish Coalescing

javascript

```javascript
// Optional chaining
const value = obj?.prop?.nested;
const result = func?.();

// Nullish coalescing
const val = null ?? "default";  // "default"
const val2 = 0 ?? "default";    // 0
```

Asynchronous JavaScript

Callbacks

javascript

```javascript
function fetchData(callback) {
    setTimeout(() => {
        callback("Data received");
    }, 1000);
}

fetchData(data => console.log(data));
```

Promises

javascript

```javascript
// Creating a promise
const promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("Success!");
        // or reject("Error!");
    }, 1000);
});

// Using promises
promise
    .then(result => console.log(result))
    .catch(error => console.error(error))
    .finally(() => console.log("Complete"));

// Promise methods
Promise.all([p1, p2, p3]);      // All must resolve
Promise.race([p1, p2, p3]);     // First to settle
Promise.allSettled([p1, p2]);   // All results
Promise.any([p1, p2, p3]);      // First to resolve
```

Async/Await

javascript

```javascript
async function fetchData() {
    try {
        const response = await fetch('/api/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}

// Parallel execution
async function parallel() {
    const [result1, result2] = await Promise.all([
        fetchData1(),
        fetchData2()
    ]);
}
```

DOM Manipulation

Selecting Elements

javascript

```
document.getElementById('id');
document.getElementsByClassName('class');
document.getElementsByTagName('tag');
document.querySelector('.class');
document.querySelectorAll('.class');
```

Creating & Modifying Elements

javascript

```
// Create
const div = document.createElement('div');
const text = document.createTextNode('Hello');

// Modify
element.textContent = 'New text';
element.innerHTML = '<span>HTML</span>';
element.setAttribute('class', 'active');
element.classList.add('new-class');
element.classList.remove('old-class');
element.classList.toggle('active');
element.style.color = 'red';
```

DOM Traversal

javascript

```
element.parentElement
element.children
element.firstElementChild
element.lastElementChild
element.nextElementSibling
element.previousElementSibling
```

Events

javascript

```
// Adding event listeners
element.addEventListener('click', (e) => {
    console.log('Clicked!', e.target);
});

// Event delegation
parent.addEventListener('click', (e) => {
    if (e.target.matches('.child')) {
        // Handle child click
    }
});

// Common events
'click', 'dblclick', 'mouseenter', 'mouseleave'
'keydown', 'keyup', 'keypress'
'submit', 'change', 'input'
'load', 'DOMContentLoaded'
```

Error Handling

Try/Catch/Finally

javascript

```
try {
    // Code that may throw
    riskyOperation();
} catch (error) {
    console.error('Error:', error.message);
} finally {
    // Always executes
    cleanup();
}
```

Throwing Errors

javascript

```javascript
throw new Error('Something went wrong');
throw new TypeError('Invalid type');
throw new RangeError('Out of range');

// Custom errors
class CustomError extends Error {
    constructor(message) {
        super(message);
        this.name = 'CustomError';
    }
}
```

Regular Expressions

Creating RegExp

javascript

```javascript
const regex1 = /pattern/flags;
const regex2 = new RegExp('pattern', 'flags');
```

Common Patterns

javascript

```javascript
/^start/        // Start of string
/end$/          // End of string
/[abc]/         // Character set
/[^abc]/        // Negated set
/[a-z]/         // Range
/\d/            // Digit [0-9]
/\w/            // Word character [a-zA-Z0-9_]
/\s/            // Whitespace
/./             // Any character
/a*/            // 0 or more
/a+/            // 1 or more
/a?/            // 0 or 1
/a{3}/          // Exactly 3
/a{2,4}/        // 2 to 4
/(group)/       // Capturing group
/(?:group)/     // Non-capturing group
```

RegExp Methods

javascript

```javascript
regex.test(string);        // Returns boolean
regex.exec(string);        // Returns match array
string.match(regex);       // Find matches
string.replace(regex, ''); // Replace matches
string.search(regex);      // Find index
string.split(regex);       // Split by pattern
```

Modules

ES6 Modules

javascript

```javascript
// Named exports (module.js)
export const name = 'John';
export function greet() { }
export { varA, varB };

// Default export
export default function() { }

// Imports
import defaultExport from './module.js';
import { name, greet } from './module.js';
import * as module from './module.js';
import defaultExport, { named } from './module.js';
```

CommonJS (Node.js)

javascript

```javascript
// Exporting
module.exports = { name: 'John' };
exports.greet = function() { };

// Importing
const module = require('./module');
const { name } = require('./module');
```

Advanced Concepts

Closures

javascript

```javascript
function outer(x) {
    return function inner(y) {
        return x + y;  // Accesses outer's x
    };
}
const addFive = outer(5);
console.log(addFive(3));  // 8
```

This Binding

javascript

```javascript
// Function context
function func() { console.log(this); }

// Method context
obj.method();  // this = obj

// Arrow functions inherit this
const arrow = () => console.log(this);

// Explicit binding
func.call(thisArg, arg1, arg2);
func.apply(thisArg, [arg1, arg2]);
const bound = func.bind(thisArg);
```

Prototypes

javascript

```javascript
// Prototype chain
obj.__proto__  // [[Prototype]]
Object.getPrototypeOf(obj)
Object.setPrototypeOf(obj, proto)

// Constructor prototype
function Person(name) {
    this.name = name;
}
Person.prototype.greet = function() {
    return `Hello, I'm ${this.name}`;
};
```

Generators

javascript

```javascript
function* generator() {
    yield 1;
    yield 2;
    yield 3;
}

const gen = generator();
console.log(gen.next());  // { value: 1, done: false }
```

Proxy & Reflect

javascript

```javascript
const proxy = new Proxy(target, {
    get(target, prop) {
        console.log(`Getting ${prop}`);
        return Reflect.get(target, prop);
    },
    set(target, prop, value) {
        console.log(`Setting ${prop} = ${value}`);
        return Reflect.set(target, prop, value);
    }
});
```

Symbols

javascript

```javascript
const sym1 = Symbol('description');
const sym2 = Symbol.for('shared');

// Well-known symbols
Symbol.iterator
Symbol.hasInstance
Symbol.toPrimitive
```

Map & Set

javascript

```javascript
// Map
const map = new Map();
map.set('key', 'value');
map.get('key');
map.has('key');
map.delete('key');
map.size;

// Set
const set = new Set([1, 2, 3]);
set.add(4);
set.has(2);
set.delete(3);
set.size;
```

WeakMap & WeakSet

javascript

```javascript
// WeakMap (keys must be objects)
const wm = new WeakMap();
wm.set(obj, 'value');

// WeakSet (values must be objects)
const ws = new WeakSet();
ws.add(obj);
```

Performance Tips

1. **Use const/let instead of var**
2. **Cache DOM queries**
3. **Debounce/throttle event handlers**
4. **Use event delegation**
5. **Minimize reflows/repaints**
6. **Use Web Workers for heavy computations**
7. **Lazy load resources**
8. **Use requestAnimationFrame for animations**

Best Practices

1. **Use strict mode**: `'use strict';`

2. **Always declare variables**
3. **Use meaningful variable names**
4. **Keep functions small and focused**
5. **Handle errors properly**
6. **Comment complex logic**
7. **Use consistent code style**
8. **Avoid global variables**
9. **Prefer immutability**
10. **Test your code**