JavaScript Complete Guide & Cheat Sheet

1. BASICS & SYNTAX

Variables & Data Types

```
// Variable Declaration
var x = 5;           // Function-scoped (avoid)
let y = 10;          // Block-scoped (preferred)
const z = 15;        // Block-scoped, immutable (preferred)

// Data Types
const string = "Hello";
const number = 42;
const boolean = true;
const array = [1, 2, 3];
const object = { name: "John", age: 30 };
const nullValue = null;
const undefinedValue = undefined;
const symbol = Symbol("id");
const bigint = 123n;
```

Operators

```
// Arithmetic
const sum = 5 + 3;       // 8
const diff = 5 - 3;      // 2
const product = 5 * 3;   // 15
const quotient = 5 / 3;  // 1.666...
const remainder = 5 % 3; // 2
const power = 2 ** 3;    // 8

// Comparison
5 == "5"      // true (loose equality)
5 === "5"     // false (strict equality)
5 !== "5"     // true
5 > 3         // true
5 >= 5        // true

// Logical
true && false   // false (AND)
true || false   // true (OR)
!true           // false (NOT)

// Assignment
x += 5;  // x = x + 5
x -= 3;  // x = x - 3
```

```
x *= 2;  // x = x * 2
x /= 4;  // x = x / 4
x++;     // x = x + 1
x--;     // x = x - 1
```

Type Checking

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

// Better type checking
Array.isArray([1,2,3])                    // true
Object.prototype.toString.call([1,2,3])   // "[object Array]"
obj instanceof ClassName                   // true/false
Number.isInteger(42)                      // true
```

---

2. STRINGS

```
// String Methods
const str = "JavaScript";

str.length           // 10
str.charAt(0)        // "J"
str.charCodeAt(0)    // 74
str.toUpperCase()    // "JAVASCRIPT"
str.toLowerCase()    // "javascript"
str.indexOf("Script")      // 4
str.lastIndexOf("a")       // 0
str.includes("Script")     // true
str.startsWith("Java")     // true
str.endsWith("Script")     // true
str.slice(0, 4)            // "Java"
str.substring(0, 4)        // "Java"
str.substr(0, 4)           // "Java"
str.replace("Java", "Type")  // "TypeScript"
str.replaceAll("a", "A")   // "JAv AScript"
str.split("S")             // ["Java", "cript"]
str.trim()                 // removes whitespace
str.padStart(15, "*")      // "****JavaScript"
str.padEnd(15, "*")        // "JavaScript****"
```

```
str.repeat(2)              // "JavaScriptJavaScript"

// Template Literals
const name = "John";
const greeting = `Hello, ${name}!`;
const multiline = `Line 1
Line 2
Line 3`;
```

---

3. NUMBERS & MATH

```
// Number Methods
const num = 42.567;

num.toFixed(2)             // "42.57"
num.toPrecision(4)         // "42.57"
num.toString()             // "42.567"
Number.parseInt("42px")    // 42
Number.parseFloat("42.5px") // 42.5
isNaN(NaN)                 // true
isFinite(100)              // true
Number.isInteger(42)       // true
Number.isSafeInteger(42)   // true

// Math Object
Math.abs(-5)          // 5
Math.round(4.6)       // 5
Math.floor(4.9)       // 4
Math.ceil(4.1)        // 5
Math.max(1, 5, 3)     // 5
Math.min(1, 5, 3)     // 1
Math.pow(2, 3)        // 8
Math.sqrt(16)         // 4
Math.random()         // 0 to 1
Math.sin(Math.PI/2)   // 1
Math.cos(0)           // 1

// Random Number Range
function randomInt(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

---

4. ARRAYS

```
// Array Creation
const arr1 = [1, 2, 3];
const arr2 = new Array(1, 2, 3);
const arr3 = Array.from("hello");  // ['h','e','l','l','o']
const arr4 = Array.of(1, 2, 3);    // [1, 2, 3]

// Array Methods
const numbers = [1, 2, 3, 4, 5];
const fruits = ["apple", "banana", "cherry"];

// Mutating Methods
fruits.push("orange")              // Adds to end
fruits.pop()                       // Removes from end
fruits.unshift("mango")            // Adds to start
fruits.shift()                     // Removes from start
fruits.splice(1, 1, "berry")       // Remove/insert at index
fruits.reverse()                   // Reverses array
fruits.sort()                      // Sorts array
fruits.sort((a, b) => a.localeCompare(b))  // Custom sort

// Non-Mutating Methods
numbers.map(x => x * 2)            // [2, 4, 6, 8, 10]
numbers.filter(x => x > 2)         // [3, 4, 5]
numbers.reduce((sum, x) => sum + x, 0)     // 15
numbers.find(x => x > 3)           // 4
numbers.findIndex(x => x > 3)      // 3
numbers.some(x => x > 4)           // true
numbers.every(x => x > 0)          // true
numbers.includes(3)                // true
numbers.indexOf(3)                 // 2
numbers.lastIndexOf(3)             // 2
numbers.join(", ")                 // "1, 2, 3, 4, 5"
numbers.slice(1, 4)                // [2, 3, 4]
numbers.concat([6, 7])             // [1, 2, 3, 4, 5, 6, 7]
numbers.flat()                     // Flattens nested arrays
numbers.flatMap(x => [x, x*2])     // Map then flatten

// Array Spread & Rest
const arr = [1, 2, 3];
const newArr = [...arr, 4, 5];     // [1, 2, 3, 4, 5]
const [first, second, ...rest] = arr;  // Destructuring
```

---

5. OBJECTS

```
// Object Creation
const obj1 = { name: "John", age: 30 };
const obj2 = new Object();
const obj3 = Object.create(null);

// Accessing Properties
obj1.name               // "John"
obj1["age"]             // 30
const key = "name";
obj1[key]               // "John"

// Methods
Object.keys(obj1)                   // ["name", "age"]
Object.values(obj1)                 // ["John", 30]
Object.entries(obj1)                // [["name", "John"], ["age", 30]]
Object.assign({}, obj1, {city: "NYC"})  // Merge objects
Object.freeze(obj1)                 // Prevents modification
Object.seal(obj1)                   // Can't add/remove props
Object.defineProperty(obj1, "id", {value: 1, writable: false})

// Destructuring
const { name, age } = obj1;
const { name: fullName, age: years = 25 } = obj1;

// Spread Operator
const newObj = { ...obj1, city: "NYC" };

// Property Check
"name" in obj1                      // true
obj1.hasOwnProperty("name")         // true

// Delete Property
delete obj1.age;
```

---

6. FUNCTIONS

```
// Function Declaration
function greet(name) {
  return `Hello, ${name}!`;
}

// Function Expression
const add = function(a, b) {
```

```
  return a + b;
};

// Arrow Function
const multiply = (a, b) => a * b;
const square = x => x * x;
const sayHi = () => "Hi!";

// Default Parameters
function greetUser(name = "Guest") {
  return `Hello, ${name}!`;
}

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

// Named Parameters
function createUser({ name, age, city = "Unknown" }) {
  return { name, age, city };
}

// Higher-Order Functions
function multiplyBy(factor) {
  return number => number * factor;
}
const double = multiplyBy(2);
double(5);  // 10

// Callback Functions
function processArray(arr, callback) {
  return arr.map(callback);
}
processArray([1, 2, 3], x => x * 2);  // [2, 4, 6]

// Immediately Invoked Function Expression (IIFE)
(function() {
  const privateVar = "secret";
})();

// Generator Function
function* generatorFunc() {
  yield 1;
  yield 2;
  yield 3;
```

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

---

7. SCOPE & CLOSURES

```
// Global Scope
var globalVar = "global";

// Function Scope
function myFunc() {
  var funcVar = "function";  // Function-scoped
  let blockVar = "block";    // Block-scoped
  const constantVar = "const"; // Block-scoped
}

// Block Scope (if, for, while)
if (true) {
  let blockScoped = "only in block";
  var functionScoped = "in entire function";
}

// Closures
function outer() {
  const message = "Hello";

  function inner() {
    console.log(message);  // Accesses outer scope
  }

  return inner;
}

const myFunc = outer();
myFunc();  // "Hello"

// Practical Closure Example
function createCounter() {
  let count = 0;

  return {
    increment: () => ++count,
    decrement: () => --count,
    getCount: () => count
```

```
  };
}

const counter = createCounter();
counter.increment();  // 1
counter.increment();  // 2
counter.decrement();  // 1

// Module Pattern
const calculator = (function() {
  const add = (a, b) => a + b;
  const subtract = (a, b) => a - b;

  return {
    add,
    subtract
  };
})();

calculator.add(5, 3);  // 8
```

---

8. THIS KEYWORD

```
// Global Context
console.log(this);  // window (browser) or global (Node.js)

// Object Method
const obj = {
  name: "John",
  greet: function() {
    console.log(this.name);  // "John"
  }
};

obj.greet();

// Constructor Function
function Person(name) {
  this.name = name;
}
const person = new Person("John");

// Arrow Functions (inherit this)
const user = {
```

```
  name: "John",
  greet: function() {
    const inner = () => console.log(this.name);  // "John"
    inner();
  }
};

// call() - Explicit this
function introduce(greeting) {
  console.log(`${greeting}, I'm ${this.name}`);
}
introduce.call({ name: "John" }, "Hi");

// apply() - Like call, but args as array
introduce.apply({ name: "John" }, ["Hello"]);

// bind() - Create new function with bound this
const introduceBound = introduce.bind({ name: "John" });
introduceBound("Hey");
```

---

9. CLASSES & OOP

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

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

  static info() {
    console.log("This is the Animal class");
  }

  get age() {
    return this._age;
  }

  set age(value) {
    this._age = value;
  }
}
```

```
// Instance
const dog = new Animal("Dog");
dog.speak();  // "Dog makes a sound"
Animal.info();  // "This is the Animal class"

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

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

const myDog = new Dog("Buddy", "Labrador");
myDog.speak();  // "Buddy barks"

// Static Methods
class MathUtil {
  static add(a, b) {
    return a + b;
  }
}
MathUtil.add(5, 3);  // 8

// Private Fields
class BankAccount {
  #balance = 0;  // Private field

  deposit(amount) {
    this.#balance += amount;
  }

  getBalance() {
    return this.#balance;
  }
}

const account = new BankAccount();
account.deposit(100);
account.getBalance();  // 100
account.#balance;      // Error: Private field
```

10. ASYNC & PROMISES

```
// Promise Creation
const promise = new Promise((resolve, reject) => {
  const success = true;

  if (success) {
    resolve("Operation successful!");
  } else {
    reject("Operation failed!");
  }
});

// Promise Handling
promise
  .then(result => console.log(result))
  .catch(error => console.error(error))
  .finally(() => console.log("Done"));

// Promise Methods
Promise.all([promise1, promise2])      // All must resolve
Promise.race([promise1, promise2])     // First to resolve wins
Promise.allSettled([promise1, promise2])  // All settle (resolve or reject)
Promise.any([promise1, promise2])      // First to resolve

// Async/Await
async function fetchData() {
  try {
    const response = await fetch("https://api.example.com/data");
    const data = await response.json();
    return data;
  } catch (error) {
    console.error("Error:", error);
  } finally {
    console.log("Request completed");
  }
}

fetchData();

// Async Function Returns Promise
const result = fetchData().then(data => {
  console.log("Data:", data);
});
```

```javascript
// Multiple Awaits
async function processData() {
  const user = await fetchUser(1);
  const posts = await fetchPosts(user.id);
  const comments = await fetchComments(posts[0].id);
  return { user, posts, comments };
}

// Parallel Awaits
async function loadAll() {
  const [user, posts, comments] = await Promise.all([
    fetchUser(1),
    fetchPosts(1),
    fetchComments(1)
  ]);
  return { user, posts, comments };
}
```

---

11. ERROR HANDLING

```javascript
// Try-Catch-Finally
try {
  const result = riskyOperation();
  console.log(result);
} catch (error) {
  console.error("Error caught:", error.message);
} finally {
  console.log("Cleanup code runs regardless");
}

// Throwing Errors
function validateAge(age) {
  if (age < 0) {
    throw new Error("Age cannot be negative");
  }
  return age;
}

// Error Types
new Error("Generic error");
new TypeError("Type is wrong");
new ReferenceError("Variable not defined");
new SyntaxError("Syntax is wrong");
```

```
new RangeError("Value out of range");
new URIError("Invalid URI");

// Custom Error Class
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}

throw new ValidationError("Invalid input");

// Error Object Properties
try {
  throw new Error("Something went wrong");
} catch (error) {
  console.log(error.message);  // "Something went wrong"
  console.log(error.name);     // "Error"
  console.log(error.stack);    // Stack trace
}
```

---

12. DOM MANIPULATION

```
// Selecting Elements
document.getElementById("myId")
document.getElementsByClassName("myClass")
document.getElementsByTagName("div")
document.querySelector(".myClass")         // First match
document.querySelectorAll(".myClass")      // All matches

// Element Properties
element.textContent = "New text";
element.innerHTML = "<b>Bold</b>";
element.outerHTML = "<div>Replace element</div>";
element.className = "newClass";
element.classList.add("class1");
element.classList.remove("class1");
element.classList.toggle("class1");
element.classList.contains("class1");

// Attributes
element.getAttribute("data-value");
element.setAttribute("data-value", "123");
```

```
element.removeAttribute("data-value");
element.hasAttribute("data-value");
element.attributes;  // NamedNodeMap

// Styles
element.style.color = "red";
element.style.backgroundColor = "blue";
element.style.marginTop = "20px";
const styles = window.getComputedStyle(element);

// Creating & Removing Elements
const newDiv = document.createElement("div");
newDiv.textContent = "Hello";
element.appendChild(newDiv);
element.insertBefore(newDiv, element.firstChild);
element.removeChild(newDiv);
element.replaceChild(newDiv, oldElement);
element.remove();

// Element Properties
element.id
element.className
element.innerHTML
element.textContent
element.value          // For input
element.checked        // For checkbox
element.selected       // For option

// Traversing DOM
element.parentElement
element.parentNode
element.children
element.childNodes
element.firstChild
element.lastChild
element.nextSibling
element.previousSibling
element.nextElementSibling
element.previousElementSibling
```

---

13. EVENTS

```
// Adding Event Listeners
element.addEventListener("click", function(event) {
```

```
  console.log("Clicked!");
});

element.addEventListener("click", (event) => {
  event.preventDefault();      // Stop default action
  event.stopPropagation();     // Stop bubbling
  event.stopImmediatePropagation();  // Stop all
});

// Removing Event Listeners
function handleClick() {
  console.log("Clicked");
}
element.addEventListener("click", handleClick);
element.removeEventListener("click", handleClick);

// Common Events
"click"          // Mouse click
"dblclick"       // Double click
"mouseenter"     // Mouse over element
"mouseleave"     // Mouse leaves element
"mousemove"      // Mouse moves
"mousedown"      // Mouse button pressed
"mouseup"        // Mouse button released
"keydown"        // Key pressed
"keyup"          // Key released
"keypress"       // Character input
"change"         // Value changed
"input"          // Input in form
"submit"         // Form submitted
"focus"          // Element focused
"blur"           // Element loses focus
"scroll"         // Page scrolled
"resize"         // Window resized
"load"           // Page loaded
"unload"         // Page unloading

// Event Object
element.addEventListener("click", (event) => {
  event.type;           // "click"
  event.target;         // Element that triggered event
  event.currentTarget;  // Element with listener
  event.clientX;        // Mouse X position
  event.clientY;        // Mouse Y position
  event.screenX;        // Screen X position
  event.screenY;        // Screen Y position
```

```
  event.key;             // Key pressed
  event.code;            // Key code
  event.preventDefault();  // Stop default action
  event.stopPropagation(); // Stop bubbling
});

// Form Events
const form = document.querySelector("form");
form.addEventListener("submit", (event) => {
  event.preventDefault();
  const formData = new FormData(form);
  console.log(formData);
});

// Event Delegation
document.addEventListener("click", (event) => {
  if (event.target.matches(".button")) {
    console.log("Button clicked!");
  }
});
```

---

14. JSON

```
// JSON.stringify()
const obj = { name: "John", age: 30 };
const jsonString = JSON.stringify(obj);
// "{"name":"John","age":30}"

// With formatting
JSON.stringify(obj, null, 2);
// {
//   "name": "John",
//   "age": 30
// }

// With replacer function
JSON.stringify(obj, (key, value) => {
  if (typeof value === "number") {
    return value * 2;
  }
  return value;
});

// JSON.parse()
```

```
const jsonStr = '{"name":"John","age":30}';
const parsed = JSON.parse(jsonStr);
// { name: "John", age: 30 }

// With reviver function
JSON.parse(jsonStr, (key, value) => {
  if (key === "age") {
    return value + 10;
  }
  return value;
});
```

---

15. REGULAR EXPRESSIONS

```
// Creating Regex
const regex1 = /pattern/flags;
const regex2 = new RegExp("pattern", "flags");

// Flags
// g - global (find all)
// i - case insensitive
// m - multiline
// s - dot matches newline
// u - unicode
// y - sticky

// Testing Match
/hello/i.test("Hello World");  // true
"Hello World".match(/hello/i);  // ["Hello"]
"Hello World".match(/l/g);      // ["l", "l"]

// Searching
"Hello World".search(/World/);  // 6
"Hello World".indexOf("World"); // 6

// Replacing
"Hello World".replace(/World/, "JavaScript");  // "Hello JavaScript"
"Hello World".replaceAll(/l/g, "L");           // "HeLLo WorLd"

// Splitting
"apple,banana,cherry".split(/,/);  // ["apple", "banana", "cherry"]

// Extracting
const regex = /(\w+)@(\w+)/;
```

```
const match = "john@example".match(regex);
// match[0] = "john@example"
// match[1] = "john"
// match[2] = "example"

// Common Patterns
/^hello/          // Starts with "hello"
/world$/          // Ends with "world"
/h.llo/           // Any character: h_llo
/[abc]/           // Character class: a, b, or c
/[^abc]/          // Not a, b, or c
/[0-9]/           // Any digit
/\d/              // Any digit
/\D/              // Non-digit
/\w/              // Word character
/\W/              // Non-word character
/\s/              // Whitespace
/\S/              // Non-whitespace
/a{3}/            // Exactly 3 a's
/a{2,4}/          // 2-4 a's
/a{2,}/           // 2 or more a's
/a+/              // 1 or more a's
/a*/              // 0 or more a's
/a?/              // 0 or 1 a's
/a|b/             // a or b
/(abc)/           // Group

// Examples
/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/  // Email
/^(\+\d{1,3}[- ]?)?\d{10}$/                          // Phone
/^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$/                 // Hex color
```

---

16. STORAGE APIs

```
// Local Storage (Persistent)
localStorage.setItem("key", "value");
localStorage.getItem("key");          // "value"
localStorage.removeItem("key");
localStorage.clear();
localStorage.length;                  // Number of items

// Session Storage (Browser session only)
sessionStorage.setItem("key", "value");
sessionStorage.getItem("key");
```

```
sessionStorage.removeItem("key");
sessionStorage.clear();

// Storing Objects
const user = { name: "John", age: 30 };
localStorage.setItem("user", JSON.stringify(user));
const retrievedUser = JSON.parse(localStorage.getItem("user"));

// Cookies
document.cookie = "name=John; path=/; max-age=3600";
document.cookie = "name=John; expires=Fri, 31 Dec 2025 23:59:59 UTC";
const cookies = document.cookie;

// Getting Cookie Value
function getCookie(name) {
  const nameEQ = name + "=";
  const cookies = document.cookie.split(";");
  for (let cookie of cookies) {
    cookie = cookie.trim();
    if (cookie.startsWith(nameEQ)) {
      return cookie.substring(nameEQ.length);
    }
  }
  return null;
}
```

---

17. FETCH API & HTTP

```
// Basic Fetch
fetch("https://api.example.com/data")
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));

// Async/Await Fetch
async function getData() {
  try {
    const response = await fetch("https://api.example.com/data");
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const data = await response.json();
    return data;
  } catch (error) {
```

```
    console.error("Fetch error:", error);
  }
}

// POST Request
fetch("https://api.example.com/users", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ name: "John", age: 30 })
})
.then(response => response.json())
.then(data => console.log(data));

// PUT Request
fetch("https://api.example.com/users/1", {
  method: "PUT",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ name: "Jane", age: 25 })
});

// DELETE Request
fetch("https://api.example.com/users/1", {
  method: "DELETE"
});

// With Headers
fetch("https://api.example.com/data", {
  headers: {
    "Authorization": "Bearer token123",
    "Content-Type": "application/json"
  }
});

// Response Methods
response.json()           // Parse as JSON
response.text()           // Parse as text
response.blob()           // Parse as blob
response.arrayBuffer()    // Parse as ArrayBuffer
response.formData()       // Parse as FormData
response.clone()          // Clone response

// Response Properties
```

```
response.status           // Status code
response.statusText       // Status message
response.ok               // true if 200-299
response.headers          // Headers object
response.url              // Final URL
```

---

18. USEFUL SHORTCUTS & TIPS

```
// Destructuring
const { x, y } = { x: 1, y: 2 };
const [a, b] = [1, 2];
const [first, ...rest] = [1, 2, 3];

// Optional Chaining
obj?.prop?.nested?.value
array?.[0]
func?.()

// Nullish Coalescing
value ?? defaultValue  // Use default if null/undefined
value || defaultValue  // Use default if falsy

// Spread Operator
const arr = [1, 2, 3];
const newArr = [...arr, 4, 5];
const obj = { a: 1, b: 2 };
const newObj = { ...obj, c: 3 };

// Short-hand Properties
const name = "John";
const age = 30;
const person = { name, age };  // { name: "John", age: 30 }

// Short-hand Methods
const obj = {
  greet() {
    return "Hello";
  }
};

// Ternary Operator
condition ? trueValue : falseValue

// Logical AND/OR
```

```
const value = obj?.prop || "default";
const result = condition && executeFunction();

// Template Literals
const str = `Hello, ${name}!`;

// Array Methods Chaining
[1, 2, 3, 4, 5]
  .filter(x => x > 2)
  .map(x => x * 2)
  .reduce((sum, x) => sum + x, 0);

// Checking Existence
if (obj?.prop) { }
if (Array.isArray(arr)) { }
if (typeof func === "function") { }

// Converting to Boolean
Boolean(value)
!!value
```

---

19. PERFORMANCE TIPS

```
// Debouncing
function debounce(func, delay) {
  let timeoutId;
  return function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func(...args), delay);
  };
}

const handleResize = debounce(() => {
  console.log("Window resized");
}, 300);

window.addEventListener("resize", handleResize);

// Throttling
function throttle(func, limit) {
  let inThrottle;
  return function(...args) {
    if (!inThrottle) {
      func(...args);
```

```
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  };
}

const handleScroll = throttle(() => {
  console.log("Page scrolled");
}, 1000);

window.addEventListener("scroll", handleScroll);

// Memoization
function memoize(func) {
  const cache = {};
  return function(...args) {
    const key = JSON.stringify(args);
    if (key in cache) {
      return cache[key];
    }
    const result = func(...args);
    cache[key] = result;
    return result;
  };
}

const fibonacci = memoize(n => {
  if (n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
});
```

---

20. COMMON GOTCHAS & BEST PRACTICES

```
// 1. Avoid Global Variables
// Bad
globalVar = "value";
// Good
const localVar = "value";

// 2. Use const by default, let when needed
// Bad
var x = 10;
// Good
const x = 10;
```

```javascript
// 3. Use strict equality
// Bad
if (x == y) { }
// Good
if (x === y) { }

// 4. Don't modify parameters
// Bad
function greet(obj) {
  obj.name = "Modified";
}
// Good
function greet(obj) {
  const newObj = { ...obj, name: "Modified" };
  return newObj;
}

// 5. Handle errors properly
// Bad
try { /* code */ } catch (e) { }
// Good
try { /* code */ } catch (error) {
  console.error("Error:", error.message);
}

// 6. Use async/await instead of callbacks
// Bad
function getData(callback) {
  fetch("/api/data").then(response => {
    response.json().then(data => {
      callback(data);
    });
  });
}

// Good
async function getData() {
  const response = await fetch("/api/data");
  return await response.json();
}

// 7. Avoid side effects
// Bad
function calculateTotal(items) {
  items.forEach(item => item.price *= 1.1);
```

```
  return items.reduce((sum, item) => sum + item.price, 0);
}

// Good
function calculateTotal(items) {
  return items.reduce((sum, item) => sum + (item.price * 1.1), 0);
}

// 8. Use meaningful names
// Bad
const x = arr.filter(a => a > 10).map(a => a * 2);

// Good
const expensiveItems = prices
  .filter(price => price > 10)
  .map(price => price * 2);

// 9. Keep functions small and focused
// Bad
function processData(data) {
  // 50 lines of code doing multiple things
}

// Good
function validateData(data) { /* validation */ }
function transformData(data) { /* transformation */ }
function saveData(data) { /* saving */ }
```