JavaScript Foundation Mastery Cheatsheet

1. Variables & Data Types

Variable Declaration

javascript

```javascript
// var - function scoped, can be redeclared
var name = "John";

// let - block scoped, can be reassigned
let age = 25;

// const - block scoped, cannot be reassigned
const PI = 3.14159;
```

Data Types

javascript

```javascript
// Primitive Types
let str = "Hello World";          // String
let num = 42;                     // Number
let bigInt = 9007199254740991n;   // BigInt
let bool = true;                  // Boolean
let undef = undefined;            // Undefined
let nul = null;                   // Null
let sym = Symbol("id");           // Symbol

// Reference Types
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 "text"         // "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
```

2. Operators

Arithmetic Operators

javascript

```javascript
10 + 5    // 15 (Addition)
10 - 5    // 5  (Subtraction)
10 * 5    // 50 (Multiplication)
10 / 5    // 2  (Division)
10 % 3    // 1  (Modulus/Remainder)
2 ** 3    // 8  (Exponentiation)
x++       // Post-increment
++x       // Pre-increment
x--       // Post-decrement
--x       // Pre-decrement
```

Comparison Operators

javascript

```javascript
5 == "5"    // true  (loose equality)
5 === "5"   // false (strict equality)
5 != "5"    // false (loose inequality)
5 !== "5"   // true  (strict inequality)
5 > 3       // true
5 < 3       // false
5 >= 5      // true
5 <= 4      // false
```

Logical Operators

javascript

```javascript
true && false   // false (AND)
true || false   // true  (OR)
!true           // false (NOT)

// Short-circuit evaluation
let result = value || "default";    // OR assignment
condition && doSomething();         // Conditional execution
```

Assignment Operators

javascript

```javascript
x = 5      // Assignment
x += 3     // x = x + 3
x -= 2     // x = x - 2
x *= 4     // x = x * 4
x /= 2     // x = x / 2
x %= 3     // x = x % 3
x **= 2    // x = x ** 2
```

3. Control Flow

Conditional Statements

```javascript
// if...else
if (condition) {
    // code if true
} else if (anotherCondition) {
    // code if another true
} else {
    // code if all false
}

// Ternary operator
let result = condition ? valueIfTrue : valueIfFalse;

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

Loops

```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...in (iterates over object keys)
for (let key in object) {
    console.log(key, object[key]);
}

// for...of (iterates over iterable values)
for (let value of array) {
    console.log(value);
}

// Loop control
break;      // Exit loop
continue;   // Skip to next iteration
```

4. Functions

Function Declaration & Expression

javascript

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

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

// Arrow function
const greet = (name) => `Hello, ${name}!`;
const add = (a, b) => a + b;
const log = () => console.log("Hi");
const square = x => x * x;  // Single param, no parentheses
```

Function Features

javascript

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

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

// Destructuring parameters
function displayUser({ name, age }) {
    console.log(`${name} is ${age} years old`);
}

// Higher-order functions
function createMultiplier(multiplier) {
    return function(x) {
        return x * multiplier;
    };
}

// IIFE (Immediately Invoked Function Expression)
(function() {
    console.log("Executed immediately!");
})();
```

5. Arrays

Array Creation & Access

javascript

```javascript
const arr = [1, 2, 3, 4, 5];
const arr2 = new Array(5);         // Empty array with 5 slots
const arr3 = Array.of(1, 2, 3);     // [1, 2, 3]
const arr4 = Array.from("hello");   // ['h', 'e', 'l', 'l', 'o']

// Accessing elements
arr[0]           // First element
arr[arr.length - 1]  // Last element
```

Array Methods - Mutating

javascript

```javascript
arr.push(6)        // Add to end
arr.pop()          // Remove from end
arr.unshift(0)     // Add to beginning
arr.shift()        // Remove from beginning
arr.splice(1, 2)   // Remove 2 elements starting at index 1
arr.splice(1, 0, 'a', 'b')  // Insert at index 1
arr.reverse()      // Reverse array
arr.sort()         // Sort array
arr.sort((a, b) => a - b)  // Numeric sort
arr.fill(0)        // Fill with value
```

Array Methods - Non-Mutating

javascript

```javascript
// Transforming
arr.map(x => x * 2)        // Transform each element
arr.filter(x => x > 2)     // Filter elements
arr.reduce((acc, x) => acc + x, 0)  // Reduce to single value

// Finding
arr.find(x => x > 3)       // First element matching condition
arr.findIndex(x => x > 3)  // Index of first match
arr.indexOf(3)             // Index of value
arr.includes(3)            // Check if includes value
arr.some(x => x > 3)       // Any element matches?
arr.every(x => x > 0)      // All elements match?

// Creating new arrays
arr.slice(1, 3)            // Extract portion
arr.concat([6, 7])         // Combine arrays
[...arr1, ...arr2]         // Spread to combine
arr.flat()                 // Flatten nested arrays
arr.flatMap(x => [x, x * 2]) // Map and flatten

// Other
arr.join(', ')             // Join to string
arr.toString()             // Convert to string
arr.forEach(x => console.log(x))  // Iterate (no return)
```

6. Objects

Object Creation & Access

javascript

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

// Accessing properties
person.name        // Dot notation
person['name']     // Bracket notation
person?.address?.city  // Optional chaining

// Adding/modifying properties
person.email = "john@example.com";
person['phone'] = "123-456-7890";

// Deleting properties
delete person.phone;
```

Object Methods

javascript

```javascript
// Object static methods
Object.keys(obj)           // Array of keys
Object.values(obj)         // Array of values
Object.entries(obj)        // Array of [key, value] pairs
Object.assign({}, obj1, obj2)  // Merge objects
Object.freeze(obj)         // Make immutable
Object.seal(obj)           // Prevent adding/deleting properties
Object.create(proto)       // Create with prototype

// Property descriptors
Object.defineProperty(obj, 'prop', {
  value: 42,
  writable: true,
  enumerable: true,
  configurable: true
});

// Check properties
obj.hasOwnProperty('prop')  // Own property check
'prop' in obj               // Property exists (including inherited)
```

Destructuring

javascript

```javascript
// Object destructuring
const { name, age } = person;
const { name: userName, age: userAge } = person;  // Rename
const { name, ...rest } = person;  // Rest properties

// Array destructuring
const [first, second] = arr;
const [first, , third] = arr;  // Skip elements
const [first, ...rest] = arr;  // Rest elements

// Nested destructuring
const { address: { city } } = person;

// Default values
const { name = "Anonymous", age = 0 } = person;
```

7. DOM Manipulation

Selecting Elements

javascript

```javascript
// Single element
document.getElementById('id')
document.querySelector('.class')    // First match
document.querySelector('#id')
document.querySelector('tag')

// Multiple elements
document.getElementsByClassName('class')
document.getElementsByTagName('tag')
document.querySelectorAll('.class')  // All matches

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

Modifying Elements

```javascript
// Content
element.textContent = "Text"        // Plain text
element.innerHTML = "<b>HTML</b>"   // HTML content

// Attributes
element.getAttribute('attr')
element.setAttribute('attr', 'value')
element.removeAttribute('attr')
element.hasAttribute('attr')

// Classes
element.classList.add('class')
element.classList.remove('class')
element.classList.toggle('class')
element.classList.contains('class')
element.className = "class1 class2"

// Styles
element.style.color = "red"
element.style.backgroundColor = "blue"
element.style.cssText = "color: red; background: blue;"

// Creating & removing elements
const newDiv = document.createElement('div')
parent.appendChild(newDiv)
parent.insertBefore(newDiv, referenceNode)
parent.removeChild(child)
element.remove()  // Remove self
```

8. Events

Event Handling

```javascript
// addEventListener (preferred)
element.addEventListener('click', function(e) {
    console.log('Clicked!', e);
});

// Multiple listeners
element.addEventListener('click', handler1);
element.addEventListener('click', handler2);

// Remove listener
element.removeEventListener('click', handler);

// Event object
element.addEventListener('click', function(e) {
    e.preventDefault();     // Prevent default action
    e.stopPropagation();   // Stop bubbling
    e.target;              // Element that triggered event
    e.currentTarget;       // Element with listener
});
```

Common Events

```javascript
// Mouse events
'click', 'dblclick', 'mousedown', 'mouseup',
'mouseover', 'mouseout', 'mousemove'

// Keyboard events
'keydown', 'keyup', 'keypress'

// Form events
'submit', 'change', 'input', 'focus', 'blur'

// Window events
'load', 'resize', 'scroll', 'unload'

// Touch events
'touchstart', 'touchmove', 'touchend'
```

Event Delegation

javascript

```javascript
// Handle events on parent for dynamic children
document.querySelector('.parent').addEventListener('click', function(e) {
    if (e.target.matches('.child')) {
        console.log('Child clicked!');
    }
});
```

9. Asynchronous JavaScript

Callbacks

javascript

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

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

Promises

javascript

```javascript
// Creating promises
const promise = new Promise((resolve, reject) => {
    if (success) {
        resolve(result);
    } else {
        reject(error);
    }
});

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

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

Async/Await

javascript

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

// Using async function
fetchData().then(data => console.log(data));

// Parallel execution
const [result1, result2] = await Promise.all([
    asyncOperation1(),
    asyncOperation2()
]);
```

Fetch API

javascript

```javascript
// GET request
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));

// POST request
fetch('https://api.example.com/data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ key: 'value' })
})
.then(response => response.json())
.then(data => console.log(data));

// With async/await
async function postData() {
  const response = await fetch(url, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });
  return response.json();
}
```

10. ES6+ Features

Template Literals

javascript

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

// Tagged templates
function tag(strings, ...values) {
    return strings[0] + values[0];
}
const result = tag`Hello ${name}`;
```

Spread & Rest

javascript

```javascript
// Spread in arrays
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];       // [1, 2, 3, 4, 5]
const arrCopy = [...arr1];          // Shallow copy

// Spread in objects
const obj1 = { a: 1, b: 2 };
const obj2 = { ...obj1, c: 3 };     // { a: 1, b: 2, c: 3 }
const objCopy = { ...obj1 };        // Shallow copy

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

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

Classes

javascript

```javascript
class Person {
  // Constructor
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  // Methods
  greet() {
    return `Hello, I'm ${this.name}`;
  }

  // Static methods
  static species() {
    return 'Homo sapiens';
  }

  // Getters and setters
  get birthYear() {
    return new Date().getFullYear() - this.age;
  }

  set birthYear(year) {
    this.age = new Date().getFullYear() - year;
  }
}

// Inheritance
class Student extends Person {
  constructor(name, age, grade) {
    super(name, age);  // Call parent constructor
    this.grade = grade;
  }

  study() {
    return `${this.name} is studying`;
  }
}

// Usage
const student = new Student("Alice", 20, "A");
console.log(student.greet());
console.log(student.study());
```

Modules

javascript

```javascript
// Exporting (in module.js)
export const PI = 3.14159;
export function add(a, b) { return a + b; }
export default class Calculator { }

// Importing
import Calculator from './module.js';        // Default import
import { PI, add } from './module.js';       // Named imports
import * as math from './module.js';         // Import all
import { add as addition } from './module.js'; // Rename import
```

Other ES6+ Features

javascript

```javascript
// Optional chaining
const city = user?.address?.city ?? 'Unknown';

// Nullish coalescing
const value = input ?? defaultValue;  // Only null/undefined

// Dynamic property names
const prop = 'name';
const obj = {
    [prop]: 'John',
    [`${prop}Length`]: 4
};

// Symbol
const sym = Symbol('id');
const obj = {
    [sym]: 'unique value'
};

// Map & Set
const map = new Map();
map.set('key', 'value');
map.get('key');
map.has('key');
map.delete('key');

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

11. Error Handling

Try-Catch-Finally

javascript

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

// Throwing errors
throw new Error('Something went wrong');
throw new TypeError('Wrong type');
throw new RangeError('Out of range');

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

12. Best Practices

Code Style

javascript

```javascript
// Use meaningful variable names
const userAge = 25;  // Good
const a = 25;        // Bad

// Use const by default, let when needed
const PI = 3.14159;
let counter = 0;

// Prefer arrow functions for callbacks
arr.map(x => x * 2);

// Use template literals for string concatenation
const message = `Hello, ${name}!`;

// Use destructuring
const { name, age } = user;

// Use default parameters
function greet(name = 'Guest') { }
```

Performance Tips

```javascript
// Cache DOM queries
const element = document.querySelector('.class');

// Use event delegation for dynamic elements
document.addEventListener('click', e => {
  if (e.target.matches('.button')) {
    // Handle click
  }
});

// Debounce expensive operations
function debounce(func, delay) {
  let timeoutId;
  return function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func.apply(this, args), delay);
  };
}

// Use requestAnimationFrame for animations
function animate() {
  // Update animation
  requestAnimationFrame(animate);
}
```

Common Patterns

javascript

```
// Module pattern
const module = (function() {
    let private = 0;

    return {
        public: function() {
            return private++;
        }
    };
})();

// Singleton pattern
const singleton = (function() {
    let instance;

    function createInstance() {
        return { /* object */ };
    }

    return {
        getInstance: function() {
            if (!instance) {
                instance = createInstance();
            }
            return instance;
        }
    };
})();

// Observer pattern
class EventEmitter {
    constructor() {
        this.events = {};
    }

    on(event, listener) {
        if (!this.events[event]) {
            this.events[event] = [];
        }
        this.events[event].push(listener);
    }

    emit(event, ...args) {
        if (this.events[event]) {
            this.events[event].forEach(listener => listener(...args));
        }
    }
}
```

Quick Reference

Type Conversion

```javascript
// To String
String(123)        // "123"
123 + ""           // "123"
123..toString()    // "123"

// To Number
Number("123")      // 123
+"123"             // 123
parseInt("123")    // 123
parseFloat("3.14") // 3.14

// To Boolean
Boolean(1)         // true
!!value            // true/false
```

Truthy/Falsy Values

```javascript
// Falsy values
false, 0, -0, 0n, "", null, undefined, NaN

// Everything else is truthy
true, {}, [], 42, "0", "false", new Date(), -42, 12n, 3.14, -3.14, Infinity
```

Useful One-Liners

javascript

```javascript
// Array unique values
const unique = [...new Set(arr)];

// Array shuffle
const shuffled = arr.sort(() => Math.random() - 0.5);

// Get random element
const random = arr[Math.floor(Math.random() * arr.length)];

// Object to array of key-value pairs
const pairs = Object.entries(obj);

// Array to object
const obj = Object.fromEntries(pairs);

// Deep clone (simple objects)
const clone = JSON.parse(JSON.stringify(obj));

// Generate range
const range = Array.from({ length: 10 }, (_, i) => i);

// Check if array
const isArray = Array.isArray(value);

// Format number with commas
const formatted = num.toLocaleString();
```

Remember: Practice is key! Use this cheatsheet as a reference while building projects to solidify your understanding of JavaScript fundamentals.