JavaScript Cheatsheet

Abyan Majid, 2023

Concept	Code Snippet
If-Else If-Else	`if (condition1) {` `  // code to be executed if condition1 is true` `} else if (condition2) {` `  // code to be executed if condition2 is true` `} else {` `  // code to be executed if both condition1 and condition2 are false` `}`
Switch	`switch (expression) {` `  case value1:` `  // code to be executed when expression equals value1` `  break;` `  case value2:` `  // code to be executed when expression equals value2` `  break;` `  // include additional cases as needed` `  default:` `  // code to be executed if expression doesn't match any case` `}`
While	`while (condition) {` `  // code to be executed while the condition is true` `}`
Do-While	`do {` `  // code to be ran at least once and then while the condition is true` `} while (condition);`
For	`// For loop` `for (initialization; condition; increment) {` `  // code to be executed for each iteration` `}`
For-In	`// For-in loop; used for iterating over properties of an object` `const object = { a: 1, b: 2, c: 3 };` `for (const key in object) {` `  console.log(key + ' = ' + object[key]);` `}`
For-Of	`// For-of loop; use for iterating over a collection` `const array = [1, 2, 3, 4, 5];` `for (const value of array) {` `  console.log(value);` `}`
Object	`const objectName = {` `  property1: value1,` `  property2: value2,` `  method1: function() {` `  // code for method1`

	`    }` `};`
Function	`function functionName(parameter1, parameter2) {` `    // function body` `    return result;` `}`
Arrow Function	`const functionName = (parameter1, parameter2) => {` `    // function body` `    return result;` `};`
Class	`class ClassName {` `  constructor(a, b, c) {` `    this.a = a;` `    this.b = b;` `    this.c = c;` `  }` `  some_method() {` `    // do stuff` `    return result` `  }` `}` `const newInstance = new ClassName(1, 2, 3)`
Inheritance (`extends`, `super`)	`class Pet {` `    constructor(name, age) {` `        this.name = name;` `        this.age = age;` `    }` `    eat() {` `        return `${this.name} is eating!` ` }` `}`  `class Cat extends Pet {` ` // Using `super` to prevent definition of variables already present in parent` ` constructor(name, age, livesLeft = 9) {` ` super(name, age)` ` this.livesLeft = livesLeft;` ` }` ` meow() {` ` return "MEOW!"` ` }` `}`  `class Dog extends Pet {` ` // Not adding any new variables, so no need to define constructor` ` bark() {` ` return "WOOF!"` ` }` `}`
Destructuring	`const person = {`

Objects	``` name: 'John Doe', age: 30, address: { street: '123 Main St', city: 'Anytown' } }; // Destructuring properties const { name, age } = person; // Destructuring nested properties const { address: { street, city } } = person;```
Destructuring Arrays	```const colors = ['red', 'green', 'blue']; // Destructuring elements const [firstColor, secondColor] = colors; // Skip elements const [, , thirdColor] = colors;```
Destructuring Function Parameters	```function displayUser({ name, age }) { console.log(`Name: ${name}, Age: ${age}`); } const user = { name: 'Johan', age: 25 }; displayUser(user); // Name: Alice, Age: 25```
Destructuring from `this`	```class ClassName { constructor(a, b, c) { this.a = a; this.b = b; this.c = c; } some_method() { const { a, b, c } = this; } }```
DOM GET-Element Methods	```// GET element by ID const elementById = document.getElementById('id'); // GET element by CLASS const elementsByClassName = document.getElementsByClassName('class'); // GET element by TAG NAME const elementsByTagName = document.getElementsByTagName('tag');```
DOM Query Selector	```// Single element query selector const element = document.querySelector('.class or #id or tag'); // Multiple elements query selector const elements = document.querySelectorAll('.class, #id, tag');```
Create New Promise	```const promise = new Promise((resolve, reject) => { if (/* condition */) {```

	`` resolve('Success'); } else { reject('Error'); } }); ```
Promises with Then-Catch	``` promise .then((result) => { // do stuff }) .catch((error) => { // do stuff }); ```
Promises with Async Function	``` const asyncFunction = async () => { try { await <code> // use await to wait for this line to finish // do stuff } catch (error) { // do stuff } }; ```