JavaScript Comprehensive Guide

1. Variables and Data Types

Theory
JavaScript has three ways to declare variables:
- ‘var‘: Function-scoped, hoisted
- ‘let‘: Block-scoped, not hoisted
- ‘const‘: Block-scoped, cannot be reassigned

Primary data types:
- Number: Both integers and floating-point numbers
- String: Text enclosed in quotes
- Boolean: true/false
- null: Intentional absence of value
- undefined: Unassigned value
- Symbol: Unique identifier
- BigInt: Large integers

Code Examples
‘‘‘javascript
// Variable declarations
let age = 25;
const name = "John";
var isStudent = true;

// Data type examples
let price = 99.99;   // Number
let message = "Hello";   // String
let isActive = false;   // Boolean
let user = null;   // Null
let score;   // Undefined
let id = Symbol("id");   // Symbol
let bigNumber = 9007199254740991n; // BigInt

// Type checking
console.log(typeof price);   // "number"
console.log(typeof message);   // "string"
‘‘‘

2. Arrays and Objects

Theory
Arrays are ordered collections of values, while objects are collections of key-value pairs.
- Arrays maintain order and are accessed by index
- Objects store properties and are accessed by keys
- Both are reference types

Code Examples
‘‘‘javascript
// Arrays
const fruits = ["apple", "banana", "orange"];
fruits.push("mango");   // Add to end
fruits.pop();   // Remove from end
fruits.unshift("grape");   // Add to start

```
fruits.shift();                // Remove from start

// Array methods
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map(num => num * 2);
const evenNumbers = numbers.filter(num => num % 2 === 0);
const sum = numbers.reduce((acc, curr) => acc + curr, 0);

// Objects
const person = {
    name: "Jane",
    age: 30,
    isEmployed: true
};

// Object operations
person.location = "New York";   // Add property
delete person.age;              // Delete property
const keys = Object.keys(person);
const values = Object.values(person);
```

3. Functions

Theory
Functions are first-class citizens in JavaScript:
- Can be assigned to variables
- Passed as arguments
- Returned from other functions
- Have their own scope
- Support different types of parameters

Code Examples
```javascript
// 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;

// Default parameters
function createUser(name, age = 18) {
    return { name, age };
}

// Rest parameters
function sum(...numbers) {
    return numbers.reduce((total, num) => total + num, 0);
```

```
}

// Callback function
function processArray(arr, callback) {
    return arr.map(callback);
}
```

4. Asynchronous JavaScript

Theory
JavaScript handles asynchronous operations through:
- Callbacks
- Promises
- Async/Await
These mechanisms help manage operations like API calls, file operations, and timers.

Code Examples
```javascript
// Promises
const fetchData = new Promise((resolve, reject) => {
    setTimeout(() => {
        const data = { id: 1, name: "User" };
        resolve(data);
        // reject("Error fetching data");
    }, 2000);
});

fetchData
    .then(data => console.log(data))
    .catch(error => console.error(error));

// Async/Await
async function getData() {
    try {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}

// Practical example: Loading user data
async function loadUserProfile(userId) {
    try {
        const user = await fetchUserData(userId);
        const posts = await fetchUserPosts(userId);
        return { user, posts };
    } catch (error) {
        throw new Error('Failed to load profile');
    }
}
```

5. DOM Manipulation

Theory
The Document Object Model (DOM) represents HTML as a tree structure:
- Elements can be selected, created, modified, and deleted
- Events can be handled
- Styles can be manipulated

Code Examples
```javascript
// Selecting elements
const element = document.getElementById('myId');
const elements = document.getElementsByClassName('myClass');
const queryElement = document.querySelector('.myClass');

// Creating elements
const div = document.createElement('div');
div.textContent = 'New Element';
div.classList.add('new-class');

// Modifying elements
element.innerHTML = '<span>Updated content</span>';
element.setAttribute('data-id', '123');
element.style.backgroundColor = 'blue';

// Event handling
element.addEventListener('click', (event) => {
    console.log('Element clicked!', event);
});

// Practical example: Dynamic list
function createTodoList(items) {
    const ul = document.createElement('ul');
    items.forEach(item => {
        const li = document.createElement('li');
        li.textContent = item;
        li.addEventListener('click', () => li.classList.toggle('done'));
        ul.appendChild(li);
    });
    document.body.appendChild(ul);
}
```

6. Error Handling

Theory
Error handling helps manage and recover from runtime errors:
- try/catch blocks catch and handle errors
- throw statement creates custom errors
- finally block executes regardless of errors

Code Examples
```javascript
// Basic error handling
try {
```

```
    // Code that might throw an error
    const result = someUndefinedFunction();
} catch (error) {
    console.error('An error occurred:', error.message);
} finally {
    console.log('This always runs');
}

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

// Practical example: Form validation
function validateUser(user) {
    try {
        if (!user.name) {
            throw new ValidationError('Name is required');
        }
        if (user.age < 18) {
            throw new ValidationError('Must be 18 or older');
        }
        return true;
    } catch (error) {
        console.error('Validation failed:', error.message);
        return false;
    }
}
```

7. Modern JavaScript Features

Theory
ES6+ introduced many new features:
- Template literals
- Destructuring
- Spread/rest operators
- Classes
- Modules
- Optional chaining

Code Examples
```javascript
// Template literals
const name = "World";
console.log(`Hello, ${name}!`);

// Destructuring
const { firstName, lastName } = person;
const [first, second, ...rest] = numbers;

// Spread operator
```

```
const newArray = [...array1, ...array2];
const newObject = { ...obj1, ...obj2 };

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

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

// Modules
export const helper = {
    formatDate(date) {
        return new Date(date).toLocaleDateString();
    }
};

// Optional chaining
const userCity = user?.address?.city;
```

Practice Exercises

1. Create a function that takes an array of numbers and returns the sum of even numbers only.
2. Implement a simple todo list with add, remove, and toggle completion functionality.
3. Create a Promise-based function that simulates an API call with random delay.
4. Implement a class for a basic shopping cart with methods to add, remove, and calculate total.