A Guide to JavaScript Concepts

This document covers the fundamental concepts of JavaScript (JS), the programming language of the web. It's used to create dynamic and interactive content.

1. JavaScript Basics: Syntax and Inclusion

JavaScript is a high-level, interpreted programming language that enables dynamic behavior on websites.

Syntax

- JavaScript is made of **statements**, separated by semicolons (;).
- Whitespace (spaces, tabs, new lines) is largely ignored.
- It is case-sensitive (myVariable is different from myvariable).

Comments

```
// This is a single-line comment

/*
  This is a
  multi-line comment.
*/
```

How to Add JS to HTML

There are two primary ways to include JavaScript:

1. External Script (Best Practice)
   A separate .js file is linked in the HTML, usually just before the closing </body> tag.
   HTML:
   ```
   <!-- ... body content ... -->
   <script src="app.js"></script>
   </body>
   </html>
   ```

   JavaScript (app.js):
   ```
   console.log("Hello from external file!");
   ```

2. Internal Script
   JavaScript is placed inside a <script> tag within the HTML, also typically at the end of the <body>.
   HTML:

```
<!-- ... body content ... -->
<script>
  console.log("Hello from internal script!");
</script>
</body>
</html>
```

2. Variables and Data Types

Variables are containers for storing data values.

Declaring Variables

- let: Declares a block-scoped variable that can be reassigned. (Modern standard)
- const: Declares a block-scoped variable that **cannot** be reassigned. (Modern standard)
- var: The old way to declare variables. It has function-scope, not block-scope, which can be confusing. (Avoid using var).

```
let age = 30;
age = 31; // This is allowed

const name = "Alice";
// name = "Bob"; // This would cause an error

var score = 100; // Avoid this
```

Data Types

Primitive Types

- **String:** Text, enclosed in single (') or double (") quotes.
  ```
  let greeting = "Hello, world!";
  ```
- **Number:** Both integers and floating-point (decimal) numbers.
  ```
  let count = 10;
  let price = 19.99;
  ```
- **Boolean:** Logical values, either true or false.
  ```
  let isComplete = false;
  ```
- **Undefined:** A variable that has been declared but not assigned a value.
  ```
  let myVar; // myVar is undefined
  ```

- **Null:** Represents the intentional absence of any object value.
  ```
  let noValue = null;
  ```

Non-Primitive Types (Objects)

- **Object:** A collection of key-value pairs.
  ```
  let person = {
    firstName: "John",
    lastName: "Doe",
    age: 50
  };
  ```

- **Array:** A special type of object used to store ordered lists.
  ```
  let colors = ["red", "green", "blue"];
  ```

3. Operators

Arithmetic Operators

+ (Add), - (Subtract), * (Multiply), / (Divide), % (Modulus/Remainder)

```
let total = 10 + 5; // 15
let remainder = 10 % 3; // 1
```

Comparison Operators

- ===: Strict equality (checks value AND type) - **Always use this**
- !==: Strict inequality
- ==: Loose equality (performs type coercion) - **Avoid this**
- !=: Loose inequality
- > (Greater than), < (Less than), >= (Greater/equal), <= (Less/equal)

```
10 === 10;   // true
10 === "10"; // false (different types)

10 == "10";  // true (type coercion, avoid)
```

Logical Operators

&& (AND), || (OR), ! (NOT)

```
let isSunny = true;
let isWarm = true;
```

```
if (isSunny && isWarm) {
  // Go to the beach
}
```

4. Control Flow

if / else if / else

Executes code based on a condition.

```
let time = 14;

if (time < 12) {
  console.log("Good morning");
} else if (time < 18) {
  console.log("Good afternoon");
} else {
  console.log("Good evening");
}
```

switch Statement

Used to perform different actions based on different conditions (a "cleaner" if/else for specific cases).

```
let fruit = "Apple";

switch (fruit) {
  case "Banana":
    console.log("Bananas are yellow.");
    break;
  case "Apple":
    console.log("Apples are red or green.");
    break;
  default:
    console.log("I don't know that fruit.");
}
```

Ternary Operator

A shortcut for an if/else statement.

```
let userAge = 20;
let canVote = (userAge >= 18) ? "Yes" : "No";
// canVote will be "Yes"
```

5. Loops

Used to execute a block of code repeatedly.

for Loop

The most common loop.

```
for (let i = 0; i < 5; i++) {
  // Runs 5 times (i=0, 1, 2, 3, 4)
  console.log(i);
}
```

while Loop

Runs as long as a condition is true.

```
let count = 0;
while (count < 5) {
  console.log(count);
  count++;
}
```

for...of Loop

Iterates over iterable values (like Arrays).

```
let fruits = ["apple", "banana", "cherry"];

for (let fruit of fruits) {
  console.log(fruit);
}
```

for...in Loop

Iterates over the *keys* (properties) of an Object.

```
let car = { make: "Ford", model: "Mustang" };

for (let key in car) {
  console.log(key); // "make", "model"
  console.log(car[key]); // "Ford", "Mustang"
}
```

6. Functions

A block of code designed to perform a particular task.

Function Declaration

Hoisted to the top of their scope (can be called before they are defined).

```
function greet(name) {
  return "Hello, " + name + "!";
}

let greeting = greet("Alice"); // "Hello, Alice!"
```

Function Expression

Not hoisted. Often used to assign a function to a variable.

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

let sum = add(5, 10); // 15
```

Arrow Functions (ES6+)

A modern, more concise syntax for functions.

```
// A concise arrow function with an implicit return
const multiply = (a, b) => a * b;

// A block-body arrow function (needs explicit 'return')
const subtract = (a, b) => {
  return a - b;
};
```

7. Data Structures (Deeper Dive)

Objects

Used to store collections of related data.

```
let user = {
  username: "coder123",
  email: "coder@example.com",
  isActive: true,
  // Objects can have methods (functions)
  greet: function() {
    console.log("Hello, my name is " + this.username);
  }
};

// Accessing properties
console.log(user.username); // "coder123"
console.log(user["email"]); // "coder@example.com"

// Calling a method
user.greet(); // "Hello, my name is coder123"

// Converting to/from JSON (JavaScript Object Notation)
let jsonString = JSON.stringify(user); // Converts object to string
let userObject = JSON.parse(jsonString); // Converts string back to object
```

Arrays

Used to store ordered lists.

```
let numbers = [10, 20, 30, 40];

// Access by index (0-based)
console.log(numbers[0]); // 10

// Add to the end
numbers.push(50); // [10, 20, 30, 40, 50]
```

```
// Iterate with forEach
numbers.forEach(function(number) {
  console.log(number);
});

// Create a new array with .map()
let doubled = numbers.map(function(number) {
  return number * 2;
});
// doubled is [20, 40, 60, 80, 100]

// Create a new array with .filter()
let over30 = numbers.filter(function(number) {
  return number > 30;
});
// over30 is [40, 50]
```

8. The DOM (Document Object Model)

The DOM is a programming interface for HTML. It represents the page as a tree of objects, allowing JavaScript to read and change the page's content, structure, and styles.

Selecting Elements

- document.getElementById('id'): Selects one element by its ID.
- document.querySelector('selector'): Selects the *first* element that matches a CSS selector. (Very versatile)
- document.querySelectorAll('selector'): Selects *all* elements that match a CSS selector (returns a NodeList).

```
// Select the element with id="title"
let title = document.getElementById("title");

// Select the first element with class="item"
let firstItem = document.querySelector(".item");

// Select all <p> elements
let paragraphs = document.querySelectorAll("p");
```

Manipulating Elements

```
let heading = document.querySelector("h1");
```

```
// Change text content
heading.textContent = "New Title";

// Change styles (use camelCase for CSS properties)
heading.style.color = "blue";
heading.style.backgroundColor = "yellow";

// Add/remove CSS classes
heading.classList.add("highlight");
heading.classList.remove("old-class");

// Change attributes
let link = document.querySelector("a");
link.setAttribute("href", "[https://www.google.com](https://www.google.com)");
```

Creating and Appending Elements

```
// 1. Create a new element
let newPara = document.createElement("p");
newPara.textContent = "This is a new paragraph.";

// 2. Find where to append it
let container = document.querySelector(".container");

// 3. Append it
container.appendChild(newPara);
```

9. Events

Events are actions that happen in the browser, such as a user clicking a button, submitting a form, or pressing a key. JavaScript can "listen" for these events and run code in response.

Event Listeners

The addEventListener method is the modern, standard way to handle events.

```
// Get the button
let myButton = document.querySelector("#myButton");

// Add a 'click' event listener
myButton.addEventListener("click", function(event) {
  // This function runs when the button is clicked
  console.log("Button was clicked!");

  // The 'event' object contains info about the event
  // event.preventDefault(); // Often used in 'submit' events to stop the form from refreshing
});

// Other common events:
// 'submit' (on a form)
// 'mouseover' (when the mouse enters an element)
// 'keydown' (when a key is pressed)
```

10. Asynchronous JavaScript

By default, JavaScript runs one line at a time (synchronously). Asynchronous JavaScript allows tasks (like fetching data from a server) to run in the background without blocking the main thread.

Promises

A Promise is an object representing the eventual completion (or failure) of an asynchronous operation.

```
// .then() runs if the promise is successful
// .catch() runs if the promise fails

fetch("[https://api.example.com/data](https://api.example.com/data)")
  .then(response => response.json()) // Convert the response to JSON
  .then(data => {
    console.log(data); // Use the data
  })
  .catch(error => {
    console.error("Error fetching data:", error);
  });
```

async / await (ES2017+)

A modern, cleaner syntax for working with Promises. It makes asynchronous code look synchronous.

- async: Must be put before a function declaration to make it an async function.
- await: Can only be used inside an async function. It pauses the function execution until

the Promise resolves.

```
async function fetchData() {
  try {
    // Pause execution until fetch() completes
    let response = await fetch("[https://api.example.com/data](https://api.example.com/data)");

    // Pause execution until .json() completes
    let data = await response.json();

    console.log(data); // Use the data

  } catch (error) {
    console.error("Error fetching data:", error);
  }
}

// Call the async function
fetchData();
```

11. Modern ES6+ Features

Template Literals

Strings using back-ticks (`) that allow embedded expressions (${...}).

```
let userName = "Bob";
let message = `Welcome, ${userName}! Your score is ${10 * 5}.`;
// message is "Welcome, Bob! Your score is 50."
```

Destructuring

A shortcut to unpack values from arrays or properties from objects into distinct variables.

```
// Object Destructuring
let settings = { theme: "dark", fontSize: 16 };
let { theme, fontSize } = settings;
// theme is "dark", fontSize is 16

// Array Destructuring
let coords = [10, 20];
let [x, y] = coords;
// x is 10, y is 20
```

Spread / Rest Operator (...)

- **Spread:** Expands an iterable (like an array) into individual elements.
- **Rest:** Collects multiple elements into a single array.

```
// Spread (for cloning or merging)
let arr1 = [1, 2, 3];
let arr2 = [...arr1, 4, 5]; // [1, 2, 3, 4, 5]

let obj1 = { a: 1, b: 2 };
let obj2 = { ...obj1, c: 3 }; // { a: 1, b: 2, c: 3 }

// Rest (in function parameters)
function sum(...allNumbers) {
  // allNumbers will be an array [1, 2, 3]
  return allNumbers.reduce((acc, current) => acc + current, 0);
}
sum(1, 2, 3); // 6
```

Modules

Allow you to split your code into separate files.

helpers.js:

```
export const PI = 3.14;
export function add(a, b) {
  return a + b;
}
```

main.js:

```
import { PI, add } from './helpers.js';
// (Note: To use modules in a browser, your <script> tag needs type="module")
// <script type="module" src="main.js"></script>

console.log(PI); // 3.14
console.log(add(5, 5)); // 10
```