JavaScript Syntax and DOM Manipulation

Variables

```
let name = "John";
const PI = 3.14;
var age = 30;
// No output here since we are just declaring variables
```

Data Types

```
let number = 25; //number
let text = "Hello"; //string
let isActive = true; //boolean
let person = {
name: "John",
age: 25
};
let colors = ["red", "green", "blue"];
let nothing = null;
let notDefined;
// No output since these are variable declarations
```

Conditionals (if-else)

```
let age = 20;
if (age >= 18) {
console.log("You are an adult.");
} else {
console.log("You are a minor.");
}
// Output: You are an adult.
```

Loops

```
// For loop
for (let i = 0; i < 5; i++) {
console.log(i);
}
// Output: 0, 1, 2, 3, 4
```

```
// While loop
let i = 0;
while (i < 5) {
console.log(i);
i++;
}
// Output: 0, 1, 2, 3, 4

// For...of loop (for arrays)
let fruits = ["apple", "banana", "cherry"];
for (let fruit of fruits) {
console.log(fruit);
}
// Output: apple, banana, cherry
```

Functions

```
// Function declaration
function greet(name) {
return "Hello " + name;
}
console.log(greet("John"));
// Output: Hello John

// Arrow function
const sum = (a, b) => a + b;
console.log(sum(5, 3));
// Output: 8
```

Objects

```
let person = {
name: "John",
age: 30,
greet() {
return "Hello " + [this.name](<http://this.name/>);
}
};
console.log(person.greet()); // Output: Hello John
```

Array Methods

```
let numbers = [1, 2, 3, 4, 5];
```

```
// Push
numbers.push(6); // Adds 6 to the array
console.log(numbers); // Output: [1, 2, 3, 4, 5, 6]

// Map
let doubled = numbers.map(x => x * 2);
console.log(doubled); // Output: [2, 4, 6, 8, 10, 12]

// Filter
let even = numbers.filter(x => x % 2 === 0);
console.log(even); // Output: [2, 4, 6]
```

Template Literals

```
let name = "John";
let greeting = `Hello, ${name}! Welcome to JavaScript.`;
console.log(greeting);
// Output: Hello, John! Welcome to JavaScript.
```

Switch Statement

```
let day = "Monday";

switch (day) {
  case "Monday":
    console.log("Start of the week");
    break;
  case "Friday":
    console.log("Almost weekend");
    break;
  default:
    console.log("Regular day");
}
// Output: Start of the week
```

Try-Catch (Error Handling)

```
try {
  let result = x / 0;
// x is not defined, so it will throw an error
} catch (error) {
  console.log("Error occurred: " + error.message);
```

```
}
// Output: Error occurred: x is not defined
```

Async Functions (Promise)

```
let fetchData = async () => {
  try {
    let response = await fetch('<https://api.example.com/data>'); //
Simulated example
    let data = await response.json();
    console.log(data); // Output: JSON data from the API
  } catch (error) {
    console.log("Error fetching data", error); // Output: Error
fetching data <error object>
  }
};
fetchData();
```

DOM Manipulation

```
// Select element
let button = document.querySelector('button');

// Add event listener
button.addEventListener('click', function() {
  alert("Button clicked!"); // When button is clicked, an alert will
show with "Button clicked!"
});
```

Ternary Operator

```
let age = 18;
let message = (age >= 18) ? "Adult" : "Minor";
console.log(message);
// Output: Adult
```

Spread Operator

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];

let merged = [...arr1, ...arr2];
console.log(merged);
```

```
// Output: [1, 2, 3, 4, 5, 6]
```

Destructuring

```
// Object destructuring
let person = { name: "John", age: 30 };
let { name, age } = person;
console.log(name, age);
// Output: John 30

// Array destructuring
let fruits = ["apple", "banana", "cherry"];
let [firstFruit, secondFruit] = fruits;
console.log(firstFruit, secondFruit);
// Output: apple banana
```

Advanced: More on JavaScript

SetTimeout and SetInterval

```
// setTimeout: Executes after a specified time
setTimeout(() => {
  console.log("This will be displayed after 2 seconds");
}, 2000);
// Output after 2 seconds


// setInterval: Executes repeatedly with a fixed interval
let count = 0;
let intervalId = setInterval(() => {
  console.log(`Count: ${count}`);
  count++;
// Output: Count: 0, 1, 2, ...
  if (count > 5) clearInterval(intervalId);
}, 1000);
// Stops after count reaches 5
```

Short-Circuit Evaluation (Logical Operators)

```
let isLoggedIn = false;
let user = isLoggedIn && "User logged in";
```

```
console.log(user); // Output: false

let name = null;
let displayName = name || "Guest";
console.log(displayName);
// Output: Guest
```

Object Methods (Object.keys, Object.values, Object.entries)

```
let user = {
  name: "Alice",
  age: 25,
  city: "New York"
};

console.log(Object.keys(user));
// Output: ["name", "age", "city"]
console.log(Object.values(user));
// Output: ["Alice", 25, "New York"]
console.log(Object.entries(user));
// Output: [["name", "Alice"], ["age", 25], ["city", "New York"]]
```

Nullish Coalescing Operator (??)

```
let username = null;
let defaultUsername = "Guest";
let nameToDisplay = username ?? defaultUsername;
console.log(nameToDisplay);
// Output: Guest (since username is null)
```

Optional Chaining (?.)

```
let user = {
  name: "Alice",
  address: {
    city: "New York"
  }
};

console.log(user?.address?.city);
// Output: New York
console.log(user?.contact?.phone);
// Output: undefined (no error)
```

Rest Parameters

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

console.log(sum(1, 2, 3, 4)); // Output: 10
```

Default Parameters

```
function greet(name = "Guest") {
  return `Hello, ${name}`;
}

console.log(greet("John"));
// Output: Hello, John
console.log(greet());
// Output: Hello, Guest
```

Array Reduce

```
let numbers = [1, 2, 3, 4, 5];

let sum = numbers.reduce((accumulator, currentValue) => accumulator +
currentValue, 0);
console.log(sum);
// Output: 15
```

Class Syntax

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

  greet() {
    return `Hello, my name is ${this.name} and I'm ${this.age} years
old.`;
  }
}
```

```
let john = new Person("John", 30);
console.log(john.greet());
// Output: Hello, my name is John and I'm 30 years old.
```

Array Find and FindIndex

```
let people = [
  { name: "Alice", age: 25 },
  { name: "Bob", age: 30 },
  { name: "Charlie", age: 35 }
];

let person = people.find(person => person.age === 30);
console.log(person);
// Output: { name: "Bob", age: 30 }

let index = people.findIndex(person => person.age === 35);
console.log(index);
// Output: 2
```

Promises

```
let fetchData = new Promise((resolve, reject) => {
  let success = true;
  if (success) {
    resolve("Data fetched successfully");
  } else {
    reject("Error fetching data");
  }
});

fetchData
  .then(result => console.log(result))
// Output: Data fetched successfully
  .catch(error => console.log(error));
// Output: Error fetching data (if rejected)
```

Promise.all

```
let promise1 = Promise.resolve(10);
let promise2 = Promise.resolve(20);
let promise3 = Promise.resolve(30);
```

```
Promise.all([promise1, promise2, promise3])
  .then(results => console.log(results)); // Output: [10, 20, 30]
```

Modules (Import/Export)

Exporting from a module (file: module.js)

```
export const name = "John";
export function greet() {
  return "Hello, John!";
}
```

Importing in another file

```
import { name, greet } from './module.js';

console.log(name);
// Output: John
console.log(greet());
// Output: Hello, John!
```

Fetching Data

```
fetch('<https://jsonplaceholder.typicode.com/posts>')
  .then(response => response.json())
  .then(data => console.log(data))
// Output: Array of posts from the API
  .catch(error => console.log("Error:", error));
```

Chaining Promises

```
fetch('<https://jsonplaceholder.typicode.com/posts>')
  .then(response => response.json())
  .then(posts => {
    console.log(posts[0]);
// Output: First post
    return fetch('<https://jsonplaceholder.typicode.com/users>');
  })
  .then(response => response.json())
  .then(users => console.log(users[0]))
// Output: First user
  .catch(error => console.log("Error:", error));
```

DOM

Selecting DOM Elements

getElementById

```
let title = document.getElementById('main-title');
console.log(title);
// Output: <h1 id="main-title">...</h1>
```

getElementsByClassName

```
let items = document.getElementsByClassName('item');
console.log(items);
// Output: HTMLCollection of elements with class 'item'
```

querySelector and querySelectorAll

```
// querySelector: selects the first match
let firstItem = document.querySelector('.item');
console.log(firstItem);
// Output: First element with class 'item'

// querySelectorAll: selects all matches (NodeList)
let allItems = document.querySelectorAll('.item');
console.log(allItems);
// Output: NodeList of all elements with class 'item'
```

Modifying DOM Content

textContent vs innerHTML

```
let paragraph = document.querySelector('p');

// Modify textContent (only text)
paragraph.textContent = "This is new text content.";
console.log(paragraph.textContent);
// Output: This is new text content.

// Modify innerHTML (HTML content including tags)
paragraph.innerHTML = "This is <strong>bold</strong> text.";
```

```
console.log(paragraph.innerHTML);
// Output: This is <strong>bold</strong> text.
```

Changing Attributes

```
let image = document.querySelector('img');

// Get the current 'src' attribute
console.log(image.getAttribute('src'));
// Output: URL of the image

// Set a new 'src' attribute
image.setAttribute('src', 'new-image.jpg');
console.log(image.getAttribute('src'));
// Output: new-image.jpg
```

Modifying Styles

```
let box = document.querySelector('.box');

// Change styles
box.style.backgroundColor = "blue";
box.style.width = "200px";
box.style.height = "200px";
```

Adding and Removing Classes

```
let element = document.querySelector('.my-element');

// Add a class
element.classList.add('new-class');

// Remove a class
element.classList.remove('my-element');

// Toggle a class (adds if not present, removes if present)
element.classList.toggle('active');
```

Creating and Appending Elements

```
let newDiv = document.createElement('div');
newDiv.textContent = "I'm a new div";
```

```
// Append to the body or any other element
document.body.appendChild(newDiv);
```

Removing Elements

```
let toBeRemoved = document.querySelector('.remove-me');
toBeRemoved.remove();
// Removes the element from the DOM
```

Event Listeners

You can add event listeners to elements to perform actions when certain events (clicks, keypresses, etc.) occur.

click event

```
let button = document.querySelector('button');

// Add a click event listener
button.addEventListener('click', () => {
  console.log('Button clicked!');
  button.textContent = "Clicked!";
// Change button text on click
});
```

keyup event

```
let input = document.querySelector('input');

// Add a keyup event listener
input.addEventListener('keyup', (event) => {
  console.log(`You pressed: ${event.key}`);
// Logs the key pressed
});
```

Event Delegation

Instead of attaching an event to every child, you can use event delegation by attaching it to a parent.

```
let list = document.querySelector('ul');
```

```
// Event delegation
list.addEventListener('click', (event) => {
  if (event.target.tagName === 'LI') {
    console.log('List item clicked:', event.target.textContent);
  }
});
```

Prevent Default Action

*You can prevent default browser behavior using **event.preventDefault()**. This is useful for links, form submissions, etc.*

```
let link = document.querySelector('a');

// Prevent the link from navigating to a new page
link.addEventListener('click', (event) => {
  event.preventDefault();
  console.log("Link clicked, but default action prevented!");
});
```

Form Input and Submission

You can listen for form submissions and capture input values.

```
let form = document.querySelector('form');

// Listen for form submission
form.addEventListener('submit', (event) => {
  event.preventDefault(); // Prevent form from submitting

  // Capture input values
  let username = document.querySelector('#username').value;
  let email = document.querySelector('#email').value;

  console.log(`Username: ${username}, Email: ${email}`);
});
```

Element Visibility

You can toggle the visibility of an element dynamically.

```
let box = document.querySelector('.box');

// Hide the element
box.style.display = 'none';

// Show the element
box.style.display = 'block';
```

Scrolling to Elements

You can programmatically scroll to a particular element on the page.

```
let section = document.querySelector('#section');

// Scroll to the element
section.scrollIntoView({ behavior: 'smooth' });
```

Animating with CSS Transitions

You can trigger CSS animations through JavaScript.

```
let animatedBox = document.querySelector('.animate');

// Start animation by adding a class
animatedBox.classList.add('slide-in');

// CSS for slide-in class:
/*
.slide-in {
  transition: transform 0.5s ease;
  transform: translateX(100px);
}
*/
```

Handling Multiple Events with One Listener

You can handle different event types using one listener and event properties.

```
let element = document.querySelector('.interactive');
```

```
element.addEventListener('mouseover', () => {
  console.log("Mouse over the element.");
});

element.addEventListener('mouseout', () => {
  console.log("Mouse left the element.");
});
```

Modifying Element Dimensions

```
let box = document.querySelector('.box');

// Change height and width dynamically

box.style.width = '300px';
box.style.height = '150px';

console.log(box.style.width); // Output: 300px
console.log(box.style.height); // Output: 150px
```