JavaScript Notes
Basic Concepts

Variables

Variable in programming is a named container that holds a value. You can think of it like a labeled box where you can store a piece of information, such as a number, word, or sentence.

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

EXAMPLE

CASE 1:

```
x = 5;
y = 6;
z = x + y;
```

CASE 2 - `var` :

Only use `var` if you MUST support old browsers.

```
var x = 5;
var y = 6;
var z = x + y;
```

CASE 3 - `let` :

Only use `let` if you can't use `const`

```
let x = 5;
let y = 6;
let z = x + y;
```

CASE 4 - `const` :

Always use `const` if the value should not be changed

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

Functions

SYNTAX:

A JavaScript function is declared using the `function` keyword, followed by the function name, and then parentheses that contain the parameter list. The function body is enclosed in curly braces `{ }`.

```
function greet(name) {
  console.log(`Hello, ${name}!`);
}
```

PARAMETERS:

Parameters are values passed to a function when it's called. In JavaScript, parameters are specified in the function declaration's parentheses. A function can have multiple parameters, separated by commas.

```
function add(x, y) {
  return x + y;
}
```

RETURN VALUE:

A function can return a value using the `return` statement.
The `return` statement stops the execution of the function and sends the specified value back to the caller.

```
function square(x) {
  return x * x;
}
```

Events

1. **Onclick**:

- The onclick property returns the click event handler code on the current element. It can be used to execute a specified function or piece of code in response to the user's click action.

2. **addEventListener**:

- The addEventListener method attaches an event handler to the specified element. Any number of event handlers can be added to a single element without overwriting existing event handlers.

Example:

```
// Using onclick
var button = document.getElementById("myButton");
button.onclick = function() {
  console.log("Button clicked");
};

// Using addEventListener
var button = document.getElementById("myButton");
button.addEventListener("click", function() {
  console.log("Button clicked");
});
```

DOM Manipulation

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a document as a tree of nodes, allowing JavaScript code to dynamically access and modify the document's content, structure, and style.

Here are few of the DOM Manipulation methods:

1) Selecting elements:

document.getElementById()

The `getElementById()` method is a part of the Document Object Model (DOM) and is used to retrieve an HTML element with a specified ID. It is a property of the `Document` interface and is available in JavaScript.

Syntax:

```
document.getElementById(element_ID)
```

Parameters:

- `element_ID`: A case-sensitive string representing the ID of the desired HTML element.

Return Value

- An `Element` object representing the HTML element with the specified `element_ID`, or `null` if no matching element is found in the document.

Example:

Suppose you have an HTML element with an ID “myPara”:

```
<p id="myPara">Some text here</p>
```

You can use `getElementById()` to retrieve this element and manipulate its properties or content:

```
const paraElement = document.getElementById("myPara");
paraElement.style.color = "blue";
```

document.querySelector()

The `querySelector()` method is a part of the Document Object Model (DOM) and is used to retrieve the first HTML element that matches a specified CSS selector. It is a property of the `Document` interface and is available in JavaScript.

Syntax:

```
document.querySelector(selector)
```

Parameters:

- `selector`: A string representing a CSS selector used to match the desired HTML element.

Return Value

- An `Element` object representing the first HTML element that matches the specified `selector`, or `null` if no matching element is found in the document.

Example:

Suppose you have the following HTML structure:

```
<div class="container">
  <p id="myPara">Some text here</p>
  <p class="highlight">Highlighted text</p>
```

```
  <span>Span element</span>
</div>
```

You can use `querySelector()` to retrieve specific elements:

```
// Retrieve an element by ID
const paraElement = document.querySelector("#myPara");
console.log(paraElement); // <p id="myPara">Some text here</p>

// Retrieve an element by class name
const highlightedPara = document.querySelector(".highlight");
console.log(highlightedPara); // <p class="highlight">Highlighted text</p>

// Retrieve an element by tag name
const spanElement = document.querySelector("span");
console.log(spanElement); // <span>Span element</span>

// Retrieve an element by attribute selector
const paraElementWithAttr = document.querySelector("p[data-test]");
console.log(paraElementWithAttr); // <p id="myPara" data-test>Some text
here</p>
```

2) Modifying Content

element.innerHTML

The `innerHTML` property is a part of the Document Object Model (DOM) and is used to get or set the HTML content of an HTML element. It is a property of the `Element` interface and is available in JavaScript.

Syntax:

```
element.innerHTML
```

Return Value:

- A string representing the HTML content of the element, including all child elements and text nodes.

  Example:

Suppose you have the following HTML structure:

```
<div id="myDiv">
  <p>This is a paragraph of text.</p>
</div>
```

You can use `innerHTML` to get or set the HTML content of the `div` element:

```
// Get the HTML content of the div element
const divElement = document.getElementById("myDiv");
console.log(divElement.innerHTML); // <p>This is a paragraph of text.</p>

// Set the HTML content of the div element
divElement.innerHTML = "<h1>This is a new heading.</h1>";
console.log(divElement.innerHTML); // <h1>This is a new heading.</h1>
```

element.textContent

The `textContent` property is a part of the Document Object Model (DOM) and is used to get or set the text content of an HTML element. It is a property of the `Element` interface and is available in JavaScript.

Syntax:

```
element.textContent
```

Return Value:

- A string representing the text content of the element, without any HTML markup.

  Example:

  Suppose you have the following HTML structure:

```
<div id="myDiv">
  <p>This is a paragraph of text.</p>
</div>
```

You can use `textContent` to get or set the text content of the `div` element:

```
// Get the text content of the div element
const divElement = document.getElementById("myDiv");
console.log(divElement.textContent); // This is a paragraph of text.

// Set the text content of the div element
divElement.textContent = "This is new text content.";
console.log(divElement.textContent); // This is new text content.
```

3) Changing Styles

element.style

element.style is a property of the HTML Element object in JavaScript, which represents the inline style of an HTML element. It is a live CSSStyleDeclaration object that contains a list of all styles properties for that element with values assigned only for the attributes that are defined in the element's inline style attribute.

Here are some key points to understand about `element.style`:

1. **Inline styles**: `element.style` only reflects the styles defined directly in the element's `style` attribute, such as `<div style="color: red;">`. It does not inherit styles from parent elements or external stylesheets.
2. **Live object**: The `element.style` property is a live object, meaning it is updated dynamically as the element's inline styles change.
3. **CSSStyleDeclaration**: `element.style` is an instance of the CSSStyleDeclaration interface, which provides methods and properties for working with CSS styles.
4. **Property access**: You can access individual style properties using dot notation, such as `element.style.color` or `element.style.fontSize`.
5. **Setter and getter**: You can set a new value for a style property using the setter syntax, like `element.style.color = 'blue'`. You can also retrieve the current value using the getter syntax, like `element.style.color`.
6. **Priority**: Inline styles (defined in `element.style`) have higher priority than external stylesheets or parent element styles. If a style property is defined both inline and in an external stylesheet, the inline style will override the external style.
7. **Limited support**: Some older browsers may not support the `element.style` property or may have limited functionality.

Example:

```
const div = document.getElementById('myDiv');
div.style.color = 'red'; // sets the text color to red
console.log(div.style.fontSize); // outputs the current font size
```

Arrays

An array is a **data structure** that stores a collection of elements, each identified by an index or key. Arrays are similar to lists in other programming languages and are used to store and manipulate data in a program.

There are several ways to create an array in JavaScript:

1. **Using the Array Constructor**: `const arr = new Array();`
2. **Using the Array Literal**: `const arr = [];`
3. **Using the Spread Operator**: `const arr = [...];`

Array Properties and Methods

Here are some common properties and methods of arrays in JavaScript:

Properties

- **length**: Returns the number of elements in the array.
- **prototype**: Allows you to add new properties and methods to the array.

Methods

- **push()**: Adds one or more elements to the end of the array.
- **pop()**: Removes the last element from the array.
- **shift()**: Removes the first element from the array.
- **unshift()**: Adds one or more elements to the beginning of the array.
- **splice()**: Adds or removes elements from the array.
- **slice()**: Returns a shallow copy of a portion of the array.
- **indexOf()**: Returns the index of the first occurrence of a specified value.
- **includes()**: Returns a boolean indicating whether the array includes a specified value.
- **forEach()**: Calls a function for each element in the array.
- **map()**: Creates a new array with the results of applying a function to each element in the array.
- **filter()**: Creates a new array with all elements that pass the test implemented by a provided function.
- **reduce()**: Applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single output value.

Example

```
// Create an array
const arr = [1, 2, 3, 4, 5];

// Push an element to the end of the array
arr.push(6);
console.log(arr); // Output: [1, 2, 3, 4, 5, 6]

// Pop an element from the end of the array
arr.pop();
console.log(arr); // Output: [1, 2, 3, 4, 5]
```

```
// Use the forEach method to iterate over the array
arr.forEach((element) => {
  console.log(element);
});

// Use the map method to create a new array with doubled values
const doubledArr = arr.map((element) => element * 2);
console.log(doubledArr); // Output: [2, 4, 6, 8, 10]

// Use the filter method to create a new array with even values
const evenArr = arr.filter((element) => element % 2 === 0);
console.log(evenArr); // Output: [2, 4]
```

Accessing Items in an Array in JavaScript

In JavaScript, you can access items in an array using their index or key.

Array indices start at `0`, so the first item in an array is at index `0`, the second item is at index `1`, and so on.

```
const colors = ['red', 'green', 'blue'];

console.log(colors[0]); // Output: "red"
console.log(colors[2]); // Output: "blue"
console.log(colors[1]); // Output: "green"
console.log(colors[-1]); // Output: "blue"
```

Loops

Loops are used to execute a block of code repeatedly for a specified number of times. They are useful when we need to perform a task multiple times.

- **For Loop**: A for loop is used to execute a block of code for a specified number of times. It consists of three parts: initialization, condition, and increment/decrement.

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

- **While Loop**: A while loop is used to execute a block of code as long as a specified condition is true.

```
let i = 0;
while (i < 5) {
```

```
  console.log(i);
  i++; // Output: 0 1 2 3 4
```

- **Do-While Loop**: A do-while loop is similar to a while loop, but it executes the code at least once before checking the condition.

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

- **ForEach Loop**: A forEach loop is a type of loop in JavaScript that is used to iterate over the elements of an array. It executes a provided function once for each array element.

```
let colors = ['red', 'green', 'blue'];
colors.forEach(function(color, index) {
  console.log(`Color at index ${index}: ${color}`);
});
```

- **Map:** The `map()` in JavaScript is a method of the Array prototype that creates a new array with the results of applying a provided function on every element in the calling array.

```
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(function(number) {
  return number * 2;
});
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

NOTE!

To learn further, refer these websites:

JavaScript official Documentation

https://developer.mozilla.org/en-US/docs/Web/JavaScript

W3Schools

https://www.w3schools.com/js/default.asp

JavaTpoint

https://www.javatpoint.com/javascript-tutorial