Strange and handy things in JavaScript

By Fred Bolder

Foreword

When you have experience with another programming language, you can be confused when programming in JavaScript. In this document there are many examples with results that you might not expect. This will help you to understand JavaScript better. There are also other useful examples.

To avoid unexpected results, you can do the following:

Don't use "var". Use "let" instead.
Don't use "==". Use "===" instead.
Don't use "!=". Use "!==" instead.
Avoid automatic type conversion. Use "Number", "parseInt", "toString" etc.

JavaScript examples

Adding a number to a string

```
let a = "Test";
let b = 1;
let c = 2;
let d = 3;
a = a + b + c.toString() + String(d);
console.log(a); // Output: "Test123"
```

Array methods

```
const numbers = [1, 10, 5, 3];

function add100(number) {
  return number + 100;
}

function greaterThan3(number) {
  return number > 3;
}

function doubleValueAtEvenIndex(number, index) {
  let output = number;

  if (index % 2 == 0) {
    output *= 2;
  }
  return output;
}

console.log(numbers.map(add100)); // Output: [ 101, 110, 105, 103 ]
console.log(numbers.filter(greaterThan3)); // Output: [ 10, 5 ]
console.log(numbers.find(greaterThan3)); // Output: 10
console.log(numbers.filter((number) => number > 5)); // Output: [ 10 ]
console.log(numbers.map(doubleValueAtEvenIndex)); // Output: [ 2, 10, 10, 3 ]

const nums = [1, 10, 5, 3, 1, 11, 10];

function uniqueNumber(number, index, arr) {
  return index === arr.indexOf(number);
}

console.log(nums.filter(uniqueNumber)); // Output: [ 1, 10, 5, 3, 11 ]
```

```
const arr = [2, 4, 10];
let initialValue = 0;
const sum = arr.reduce((accumulator, current, index) => {
  return accumulator + current;
  // The accumulator will be set to the returned value
}, initialValue);

initialValue = 1;
const product = arr.reduce((accumulator, current) => {
  return accumulator * current;
}, initialValue);

const min = arr.reduce((accumulator, current) => {
  return current < accumulator ? current : accumulator;
}); // No initial value
// When there is no initial value, the accumulator  will
// be set to the first number in the array and the first step
// will be skipped.

console.log(sum); // Output: 16
console.log(product); // Output: 80
console.log(min); // Output: 2


const arraySortExample = [4, 2, 3, 1, 11];
arraySortExample.sort(); // Default sorting is alphabetic
console.log(arraySortExample); // Output: [ 1, 11, 2, 3, 4 ]
arraySortExample.sort((a, b) => a - b);
console.log(arraySortExample); // Output: [ 1, 2, 3, 4, 11 ]
arraySortExample.sort((a, b) => b - a);
console.log(arraySortExample); // Output: [ 11, 4, 3, 2, 1 ]

console.log([6, 7, 8].every((n) => n > 5)); Output: true
console.log([5, 7, 8].every((n) => n > 5)); Output: false
console.log(["A", "A", "A"].every((s) => s === "A")); Output: true
console.log([5, 7, 8].some((n) => n > 5)); Output: true
console.log([3, 2, 1].some((n) => n > 5)); Output: false


function onlyNumbers(code) {
  return code.split("").every((n) => "1234567890".includes(n));
}

console.log(onlyNumbers("1234")); // true
console.log(onlyNumbers("2.5")); // false
```

Arrays

```
// Add or remove items
const a = [123, "A"]; // Mixed types
console.log(a); // Output: [ 123, 'A']
a.push(1); // Add item at the end
console.log(a); // Output: [ 123, 'A', 1 ]
a[0] = 321; // Change item
console.log(a); // Output: [ 321, 'A', 1 ]
a[3] = 2; // Index out of range, but no error
console.log(a); // Output: [ 321, 'A', 1, 2 ]
a[6] = "TEST";
console.log(a); // Output: [ 321, 'A', 1, 2, <2 empty items>, 'TEST' ]
console.log(a[5]); // Output: undefined
console.log(typeof a[5]); // Output: "undefined"
a.pop(); // Delete last item
console.log(a); // Output: [ 321, 'A', 1, 2, <2 empty items> ]
a.shift(); // Delete first item
console.log(a); // Output: [ 'A', 1, 2, <2 empty items> ]
a.unshift(9); // Add item at the start
console.log(a); // Output: [ 9, 'A', 1, 2, <2 empty items> ]
console.log(typeof a); // Output: "object"
console.log(Array.isArray(a)); // Output: true

// join and split
const names = ["John", "Mary", "Tom"];
let str = names.join(" ");
console.log(str); // Output: "John Mary Tom"
const arr = str.split(" ");
console.log(arr); // Output: [ 'John', 'Mary', 'Tom' ]

// Spread array
console.log(Math.max(...[3, 10, 5, 30, 22])); // Output: 30

// Array to object
console.log({ ...[1, 2, 3] }); // Output: { '0': 1, '1': 2, '2': 3 }
```

await

```
function process() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Resolved");
    }, 2000);
  });
}

async function start() {
  // await can only be used in an async function
  let result = await process();
  console.log(result); // Output: "Resolved"
}

start();
console.log("Busy"); // Output: "Busy"
```

Boolean

```
console.log(Boolean(1)); // Output: true
console.log(Boolean(0)); // Output: false
console.log(Boolean("true")); // Output: true
console.log(Boolean("false")); // Output: true
console.log(Boolean("0")); // Output: true
console.log(Boolean("")); // Output: false
console.log(Boolean(null)); // Output: false
console.log(Boolean(undefined)); // Output: false
```

Changing the type of a variable

```
let x = 5;
x = "Hello!";
console.log(x); // Output: "Hello!"
```

class

```
class Customer {
  constructor(firstName, lastName, email = "") {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
  }
  fullName() {
    return this.firstName + " " + this.lastName;
  }
}
const customer1 = new Customer("John", "Smith");
console.log(customer1);
// Output: Customer { firstName: 'John', lastName: 'Smith', email: '' }
console.log(customer1.fullName()); // Output: "John Smith"

class MathUtils {
  static add(a, b) {
    return a + b;
  }
  static sub(a, b) {
    return a - b;
  }
}
console.log(MathUtils.add(2, 3)); // Output: 5
```

```
class Animal {
  constructor(type, name, color) {
    this.type = type;
    this.name = name;
    this.color = color;
  }
  sleep() {
    return `${this.name} is sleeping`;
  }
}

class Bird extends Animal {
  constructor(name, color) {
    super("bird", name, color);
  }
  fly() {
    return `${this.name} is flying`;
  }
}

class Fish extends Animal {
  constructor(name, color) {
    super("fish", name, color);
  }
  swim() {
    return `${this.name} is swimming`;
  }
}

const bird1 = new Bird("Tweety", "blue");
console.log(bird1); // Output: Bird { type: 'bird', name: 'Tweety', color: 'blue' }
console.log(bird1.fly()); // Output: Tweety is flying
console.log(bird1.sleep()); // Output: Tweety is sleeping
const fish1 = new Fish("Nemo", "orange");
console.log(fish1.swim()); // Output: Tweety is flying
console.log(typeof fish1); // Output: object
console.log(fish1.constructor.name); // Output: Fish
```

Comparing

```
let a = 1;
let b = "1";
console.log(a == b); // Output: true
console.log(a === b); // Output: false
console.log(a != b); // Output: false
console.log(a !== b); // Output: true
```

Converting an object to an array

```
const person = {
  firstName: "Tom",
  lastName: "Smith",
  age: 40,
}
console.log(Object.keys(person)); // Output: [ 'firstName', 'lastName', 'age' ]
console.log(Object.values(person)); // Output: [ 'Tom', 'Smith', 40 ]
console.log(Object.entries(person));
// Output: [ [ 'firstName', 'Tom' ], [ 'lastName', 'Smith' ], [ 'age', 40 ] ]
console.log(person[Object.keys(person)[0]]); // Output: Tom
```

Copying an array

```
let arr = [1, 2, 3];
let arrCopy = [...arr];
console.log(arr); // Output: [ 1, 2, 3 ]
console.log(arrCopy); // Output: [ 1, 2, 3 ]
arrCopy[0] = 4;
console.log(arr); // Output: [ 1, 2, 3 ]
console.log(arrCopy); // Output: [ 4, 2, 3 ]
arrCopy = [...arrCopy, ...arr];
console.log(arrCopy); // Output: [ 4, 2, 3, 1, 2, 3 ]
arrCopy = arr.slice();
console.log(arrCopy); // Output: [ 1, 2, 3 ]
```

Copying an object

```
const user = {
   firstName: "Tom",
   lastName: "Smith",
};
console.log("Shallow copy");
const userCopy1 = { ...user }; // Can only be used for primitive values!
const userCopy2 = Object.assign({}, user); // Only for primitive values!
console.log(user); // Output: { firstName: 'Tom', lastName: 'Smith' }
console.log(userCopy1); // Output: { firstName: 'Tom', lastName: 'Smith' }
console.log(userCopy2); // Output: { firstName: 'Tom', lastName: 'Smith' }
userCopy1.firstName = "Mary";
userCopy2.firstName = "Fred";
console.log(user); // Output: { firstName: 'Tom', lastName: 'Smith' }
console.log(userCopy1); // Output: { firstName: 'Mary', lastName: 'Smith' }
console.log(userCopy2); // Output: { firstName: 'Fred', lastName: 'Smith' }
console.log("Deep copy");
delete user.lastName; // Delete for shorter log line :)
user.hobbies = [ "Music", "Tennis" ];
const userCopy3 = JSON.parse(JSON.stringify(user));
console.log(userCopy3);
// Output: { firstName: 'Tom', hobbies: [ 'Music', 'Tennis' ] }
const userCopy4 = JSON.parse(JSON.stringify(userCopy3));
userCopy4.hobbies[1] = "Reading";
console.log(userCopy3);
// Output: { firstName: 'Tom', hobbies: [ 'Music', 'Tennis' ] }
console.log(userCopy4);
// Output: { firstName: 'Tom', hobbies: [ 'Music', 'Reading' ] }
```

Data types

```
// Primitive data types are immutable
// Null, Undefined, Boolean, Number, BigInt, String, Symbol
// Objects are mutable
// Array, Object, Date, Function, RegExp
function add(a, b) {
  return a + b;
}
console.log(add(3, 4)); // Output: 7
console.log(typeof add); // Output: "function"
const arr = [1, 2, 3];
console.log(typeof arr); // Output: "object"
console.log(Array.isArray(arr)); Output: true
```

Date

```
const today = new Date();
console.log(today.getMonth()); // 0 = January, 1 = February etc.
console.log(today.getDay()); // 0 = Sunday, 1 = Monday etc.
console.log(today.getFullYear()); // Example: 2023
```

Destructuring

```
const fruits = ["Apple", "Banana", "Kiwi", "Mango"];
const [apple, banana, ...other] = fruits;
console.log(apple); // Output: "Apple"
console.log(banana); // Output: "Banana"
console.log(other); // Output: [ 'Kiwi', 'Mango' ]
const [, secondFruit] = fruits;
console.log(secondFruit); // Output: "Banana"
console.log(apple); // Output: "Apple"

const person = {
  firstName: "John",
  lastName: "Evans",
  email: "john.evans@mail.com",
  address: { street: "Street 20", city: "London" },
};
const { lastName, firstName, age = 34 } = person;
// The variable name must be the same as the property name.
// The order of the variables does not matter.
console.log(firstName); // Output: "John"
console.log(lastName); // Output: "Evans"
console.log(age); // Output: 34
const { email, ...more } = person;
console.log(email); // Output: "john.evans@mail.com"
console.log(more); // Output: { firstName: 'John', lastName: 'Evans' }
console.log(firstName); // Output: "John"
// Using other variable names
const { firstName: name1, lastName: name2 } = person;
console.log(name1); // Output: "John"
console.log(name2); // Output: "Evans"
// Nested object
const {
  firstName: first,
  address: { street, city },
} = person;
console.log(first); // Output: "John"
console.log(street); // Output: "Street 20"
console.log(city); // Output: "London"

const person1 = {
    firstName: "Fred",
    lastName: "Bolder"
}

const person2 = {
    firstName: "John",
    lastName: "Smith"
}

let {firstName, lastName} = person1;
console.log(firstName); // Output: "Fred"
({firstName, lastName} = person2); // Using the same variables
console.log(firstName); // Output: "John"
```

Expression in a string

```
let firstName = "Fred";
let message = `Hello ${firstName}`;
console.log(message); // Output: "Hello Fred"
```

for

```
let i = 10;
let s = "";
for (let i = 0; i < 3; i++) {
  if (i !== 0) {
    s += ", ";
  }
  s += i.toString();
}
console.log(s); // Output: "0, 1, 2"
console.log(i); // Output: 10

let a = [];
let n = 10;
for (let i = 0; i < 3; i++) {
  if (i === 0) {
    let n = 5;
    a.push(n);
  }
  a.push(n);
}
console.log(a); // Output [ 5, 10, 10, 10 ]
```

for in

```
const person = {
  firstName: "John",
  lastName: "Smith",
  Age: 20,
}

for (const key in person) {
  console.log(`${key} = ${person[key]}`);
}

// Output:
// firstName = John
// lastName = Smith
// Age = 20
```

for of

```
const names = [ "John", "Mary", "Fred" ];

for (const item of names) {
  console.log(item);
}

// Output:
// John
// Mary
// Fred
```

Function as argument

```
function calc(input, callback) {
  const output = [];
  for (let i = 0; i < input.length; i++) {
    output.push(callback(input[i]));
  }
  return output;
}

function double(number) {
  return number * 2;
}

function half(number) {
  return number / 2;
}

console.log(calc([1, 2, 3], double)); // Output: [ 2, 4, 6 ]
console.log(calc([1, 2, 3], half)); // Output: [ 0.5, 1, 1.5 ]
```

Functions

```
function add1(a, b) {
  return a + b;
}

const add2 = function (a, b) {
  return a + b;
};

const add3 = (a, b) => {
  return a + b;
};

const add4 = (a, b) => a + b;

console.log(add1(1, 2)); // Output: 3
console.log(add2(1, 2)); // Output: 3
console.log(add3(1, 2)); // Output: 3
console.log(add4(1, 2)); // Output: 3

function multiplyBy10(a) {
  return a * 10;
}

console.log(multiplyBy10(1)); // Output: 10
console.log(multiplyBy10(1, 2)); // Output: 10 (no error for extra argument)
```

indexOf

```
let s = "This is a test";
console.log(s.indexOf("test")); // Output: 10
console.log(s.indexOf("Test")); // Output: -1
console.log(s.indexOf("i", 4)); // Output: 5
console.log(s.lastIndexOf("t")); // Output: 13
console.log(s.lastIndexOf("t", 12)); // Output: 10
```

Methods

```
// It is better not to use arrow functions in an object, because of "this"
const myMath1 = {
  add: function (number1, number2) {
    return number1 + number2;
  },
};
console.log(myMath1.add(1, 2)); // Output: 3

const myMath2 = {
  add(number1, number2) {
    return number1 + number2;
  },
};
console.log(myMath2.add(3, 4)); // Output: 7

myMath2.subtract = function (number1, number2) {
  return number1 - number2;
};
console.log(myMath2.subtract(10, 2)); // Output: 8
```

NaN

```
let a = NaN;
console.log(typeof a); // Output: "number"
console.log(a === NaN); // Output: false
console.log(isNaN(a)); // Output: true
console.log(isNaN("test")); // Output: true
console.log(Number.isNaN("test")); // Output: false
console.log(isNaN("123test")); // Output: true
console.log(Number.isNaN("123test")); // Output: false
// isNaN tries to convert first, Number.isNaN not
```

Number type

```
let a = 5;
let b = 2.5;
let c = 10n;
console.log(typeof a); // Output: "number"
console.log(typeof b); // Output: "number"
console.log(typeof c); // Output: "bigint"
```

Object

```
const person1 = {
  firstName: "John",
  "last Name": "Smith", // Space between last and Name!
}
console.log(person1.firstName); // Output: "John"
let person2 = person1; // Pointer
person2.firstName = "Tom";
console.log(person1.firstName); // Output: "Tom"
console.log(person2.firstName); // Output: "Tom"
console.log(person2["last Name"]); // Output: "Smith"
console.log(Object.keys(person1)); // Output: [ 'firstName', 'last Name' ]
console.log(Object.keys(person1).length); // Output: 2
console.log(person1.hasOwnProperty("firstName")); // Output: true
console.log(person1.hasOwnProperty("lastName")); // Output: false
console.log(Boolean(person1["firstName"])); // Output: true
delete person1.firstName;
console.log(Object.keys(person1)); // Output: [ 'last Name' ]
person1.test = "ABC"; // Property test will be added!
console.log(Object.keys(person1)); // Output: [ 'last Name', 'test' ]
let p = "test2";
person1[p] = "DEF"; // Property test2 will be added
console.log(Object.keys(person1)); // Output: [ 'last Name', 'test', 'test2' ]


function CreateUser(firstName, lastName) {
  const obj = {
    firstName: firstName, // you can use the same names
    lastName: lastName,
  };
  return obj;
}

const user1 = CreateUser("John", "Smith");
console.log(user1); // Output: { firstName: 'John', lastName: 'Smith' }
for (const key in user1) {
  console.log(key, user1[key]);
}
// Output: "firstName John"
// Output: "lastName Smith"

const user1Entries = Object.entries(user1);
console.log(user1Entries);
// [ [ 'firstName', 'John' ], [ 'lastName', 'Smith' ] ]
for (let i = 0; i < user1Entries.length; i++) {
  console.log(user1Entries[i]);
}
// Output: [ 'firstName', 'John' ]
// Output: [ 'lastName', 'Smith' ]
// Read only
const object1 = {};
Object.defineProperty(object1, 'property1', {
  value: 10,
  writable: false,
  configurable: true, // if you want to change writable later
});
object1.property1 = 5; // Throws an error in strict mode
console.log(object1.property1); // Output: 10
Object.defineProperty(object1, 'property1', {
    writable: true,
  });
console.log(object1.property1); // Output: 10
object1.property1 = 6;
console.log(object1.property1); // Output: 6
```

Optional parameter(s)

```
const add = 0;
const substract = 1;

function calculate(number1, number2, mode = add) {
  let value;

  if (
    typeof number1 === "number" &&
    typeof number2 === "number" &&
    Number.isInteger(mode)
  ) {
    switch (mode) {
      case add:
        value = number1 + number2;
        break;
      case substract:
        value = number1 - number2;
        break;
      default:
        value = NaN;
        break;
    }
  } else {
    value = NaN;
  }
  return value;
}

console.log(calculate(1, 2)); // Output: 3
console.log(calculate(1, 2, add)); // Output: 3    if (input === number) {
      alert(`Suceess, the number was ${number}! Attempts: ${counter}`);
    }
console.log(calculate(1, 2, substract)); // Output: -1
console.log(calculate(1.5, 4.3, add)); // Output: 5.8
console.log(calculate("A", "B")); // Output: NaN

function greet(name = "stranger", greeting = "Hello")
{
    return `${greeting} ${name}!`;
}

console.log(greet("John", "Hi")); // Output: "Hi John!"
console.log(greet()); // Output: "Hello stranger!"
console.log(greet("Mary")); // Output: "Hello Mary!"
console.log(greet(undefined, "Hi")); // Output: "Hi stranger!"
```

```
function sum(...numbers) {
  let result = 0;

  for (let i = 0; i < numbers.length; i++) {
    result += numbers[i];
  }
  return result;
}

console.log(sum(1, 2, 3)); // Output: 6
console.log(sum(2, 2, 2, 2)); // Output: 8
```

parseInt vs Number

```
console.log(parseInt("123abc")); // Output: 123
console.log(parseInt("abc123")); // Output: NaN
console.log(Number("123abc")); // Output: NaN
console.log(Number("abc123")); // Output: NaN
```

Power

```
let a = Math.pow(2, 3); // Math.pow accepts only numbers
let b = 2n ** 3n; // ** accepts also bigint
console.log(a); // Output: 8
console.log(b); // Output: 8n
```

Promises

```
const promise = new Promise((resolve, reject) => {
  let error = false;

  // Time consuming code

  if (!error) {
    resolve("OK");
  } else {
    reject("Error");
  }
});

promise
  .then(function (result) {
    console.log(result);
  })
  .catch(function (error) {
    console.log(error);
  })
  .finally(function () {
    console.log("Ready");
  });
```

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Resolved 1");
  }, 2000);
});

const promise2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Resolved 2");
  }, 1000);
});

Promise.all([promise1, promise2])
  .then(function (result) {
    console.log(result); // Output: [ 'Resolved 1', 'Resolved 2' ]
  })
  .catch(function (error) {
    console.log(error);
  })
  .finally(function () {
    console.log("Ready"); // Output: "Ready"
  });
```

querySelector

```
// querySelector returns only the first matching node or null.
const ul = document.querySelector("ul");
const li = ul.querySelector("li");
// Change the first item in the first unordered list to "Item A"
li.innerText = "Item A";
```

querySelectorAll

```
// querySelectorAll returns a list (like an array) with all matching nodes.
const allLinks = document.querySelectorAll("a");
// Remove the underline of all links
allLinks.forEach((current, idx) => {
  current.style.textDecoration = "none"
});
```

randomInt

```
function randomInt(min, max) {
  let r;

  if (Number.isInteger(min) && Number.isInteger(max) && (max >= min)) {
    r = Math.floor(Math.random() * (max - min + 1)) + min;
  } else {
    r = NaN;
  }
  return r;
}

console.log(randomInt(5, 10).toString()); // Output: integer between 5 and 10
console.log(randomInt(-2, 2).toString()); // Output: integer between -2 and 2
console.log(randomInt(0, 2.5).toString()); // Output: NaN
console.log(randomInt("A", "C").toString()); // Output: NaN
```

Sanitize

```
export default class Utils {
  static sanitize(s, options = {}) {
    // Default only trim all
    let result = s;

    if (typeof options === "object") {
      if (options.hasOwnProperty("trim")) {
        if (typeof options.trim === "string") {
          switch (options.trim.trim().toLowerCase()) {
            case "all":
              result = result.trim();
              break;
            case "end":
              result = result.trimEnd();
              break;
            case "start":
              result = result.trimStart();
              break;
            default:
              break;
          }
        }
      } else {
        result = result.trim();
      }
      if (options.hasOwnProperty("case")) {
        if (typeof options.case === "string") {
          switch (options.case.trim().toLowerCase()) {
            case "capitalize":
              if (result.length > 0) {
                result =
                  result.charAt(0).toUpperCase() +
                  result.slice(1).toLowerCase();
              }
              break;
            case "lower":
              result = result.toLowerCase();
              break;
            case "upper":
              result = result.toUpperCase();
              break;
            default:
              break;
          }
        }
      }
    }
    return result;
  }
}
```

String index

```
let a = "Test";
let b = a[10]; // No error, but can lead to error
console.log(b); // Output: undefined
```

String type

```
let a = "Test";
let b = 'Test';
let c = `Test`;
let d = 'He said: "JavaScript is cool!"';
console.log(typeof a); // Output: "string"
console.log(typeof b); // Output: "string"
console.log(typeof c); // Output: "string"
console.log(typeof d); // Output: "string"
```

substring

```
let a = "example";
// first number is start index
// second number is end index (that character is not included)
console.log(a.substring(4, 6)); // Output: pl
console.log(a.substring(2)); // Output: ample
console.log(a.substring(-2)); // Output: example
console.log(a.slice(-2)); // Output: le
```

Subtracting a number from a string

```
let x = "20";
x = x - 5; // It is better to avoid this
console.log(x); // Output: 15
```

Swapping values

```
let a = 5;
let b = 10;
console.log(a, b); // Output: 5 10
[a, b] = [b, a];
console.log(a, b); // Output: 10 5
```

Ternary operator

```
let x = 5;
let s = x >= 0 ? "positive" : "negative";
console.log(s); // Output: "positive"
x = -3;
s = x >= 0 ? "positive" : "negative";
console.log(s); // Output: "negative"
```

this

```
function showName() {
  return this.firstName; // "this" is obj when calling obj.info2
}

const obj = {
  firstName: "John",
  info1() {
    return this.firstName;
  },
  info2: showName,
};

console.log(obj.info1()); // Output: "John"
console.log(obj.info2()); // Output: "John"
```

Type of null

```
let x = 5;
x = null;
console.log(typeof x); // Output: "object"
```

undefined vs null

```
let a = null;
let b;
console.log(a); // Output: null
console.log(b); // Output: undefined
console.log(typeof a); // Output: "object"
console.log(typeof b); // Output: "undefined"
console.log(a == b); // Output: true
console.log(a === b); // Output: false
a = a + 3; // no error (null is converted to 0)
console.log(a); // Output: 3
b = b + 3;
console.log(b); // Output: NaN
```

Simple examples of JavaScript in HTML file

For these simple examples, the JavaScript code is in the HTML file.
If you want to use an external js file, you can add the following line at the end in the body section of the HTML file.

```
<script src="./script.js"></script>
```

Random number

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Random number</title>
    <style>
      #number {
        font-size: 5em;
      }
    </style>
  </head>
  <header>
    <h1>Random number</h1>
  </header>
  <body>
    <button id="btGenerate">Generate</button>
    <p id="number"></p>

    <script>
      document
        .getElementById("btGenerate")
        .addEventListener("click", randomNumber);

      function randomNumber() {
        document.getElementById("number").innerHTML = Math.trunc(
          Math.random() * 6 + 1
        ).toString();
      }
    </script>
  </body>
</html>
```

Converter

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Converter</title>
  </head>
  <header>
    <h1>Converter</h1>
  </header>
  <body>
    <label for="input">Number</label>
    <input id="input" type="text" />
    <button id="btcmToInch">cm to inch</button>
    <button id="btInchToCm">inch to cm</button>
    <p id="result"></p>

    <script>
      document.getElementById("btcmToInch").addEventListener("click", () => {
        cmInch(0);
      });
      document.getElementById("btInchToCm").addEventListener("click", () => {
        cmInch(1);
      });

      function cmInch(mode) {
        let input = Number(document.getElementById("input").value);
        let output = "";
        if (Number.isNaN(input)) {
          output = "Invalid value for Number entered";
        } else {
          if (mode === 0) {
            output = (input / 2.54).toString();
          } else {
            output = (input * 2.54).toString();
          }
        }
        document.getElementById("result").innerHTML = output;
      }
    </script>
  </body>
</html>
```

Set CSS custom property

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Set CSS custom property</title>
    <style>
      :root {
        --primary-color: blue;
      }
      body {
        background-color: var(--primary-color);
      }
    </style>
  </head>
  <header>
    <h1>Set CSS custom property</h1>
  </header>
  <body>
    <button id="btBlue">Blue</button>
    <button id="btYellow">Yellow</button>

    <script>
      document.getElementById("btBlue").addEventListener("click", () => {
        changeColor(0);
      });
      document.getElementById("btYellow").addEventListener("click", () => {
        changeColor(1);
      });

      function changeColor(colorNumber) {
        let r = document.querySelector(":root");
        if (colorNumber === 0) {
          r.style.setProperty("--primary-color", "blue");
        }
        if (colorNumber === 1) {
          r.style.setProperty("--primary-color", "yellow");
        }
      }
    </script>
  </body>
</html>
```

Mouse events

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Mouse events</title>
    <style>
      #position {
        font-size: 1.5em;
      }
    </style>
  </head>
  <header>
    <h1>Mouse events</h1>
  </header>
  <body>
    <p id="position"></p>

    <script>
      document.addEventListener("mousemove", mouseMoved);

      function mouseMoved(event) {
        let s = "";
        s = `Position: ${event.clientX}, Y: ${event.clientY}`;
        document.getElementById("position").innerHTML = s;
      }
    </script>
  </body>
</html>
```

Modules example

calc.js

```
class myMath {
  static add(a, b) {
    return a + b;
  }

  static multiply(a, b) {
    return a * b;
  }
}

export { myMath };
```

script.js

```
// This is just an example. Normally you don't need modules for this.
// Type the command below in the terminal to use modules with Node
// npm init -y es6

import { myMath } from "./calc.js";

const btCalculate = document.getElementById("btCalculate");
const result = document.getElementById("result");
result.style.marginTop = "2rem";
result.style.fontSize = "2rem";

btCalculate.addEventListener("click", () => {
  const number1 = Number(document.getElementById("number1").value);
  const number2 = Number(document.getElementById("number2").value);
  const selector = document.getElementById("selector").value;
  switch (selector) {
    case "Add":
      result.innerText = myMath.add(number1, number2).toString();
      break;
    case "Multiply":
      result.innerText = myMath.multiply(number1, number2).toString();
      break;
    default:
      break;
  }
});
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Modules example</title>
  </head>
  <body>
    <header>
      <h1>Modules example</h1>
    </header>
    <main>
      <input id="number1" type="number" />
      <select id="selector">
        <option value="Add" selected="selected">Add</option>
        <option value="Multiply">Multiply</option>
      </select>
      <input id="number2" type="number" />
      <button id="btCalculate">Calculate</button>
      <div id="result"></div>
    </main>
    <script type="module" src="./script.js"></script>
  </body>
</html>
```

Useful links

for loops
https://thecodebarbarian.com/for-vs-for-each-vs-for-in-vs-for-of-in-javascript

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

Other

https://www.smashingmagazine.com/2023/02/discovering-primitive-objects-javascript-part1/

https://www.zhenghao.io/posts/javascript-variables

https://www.zhenghao.io/posts/javascript-memory
[[200~npm install webpack webpack-cli style-loader css-loader --save-dev~

https://frontendmasters.com/blog/vanilla-javascript-todomvc/