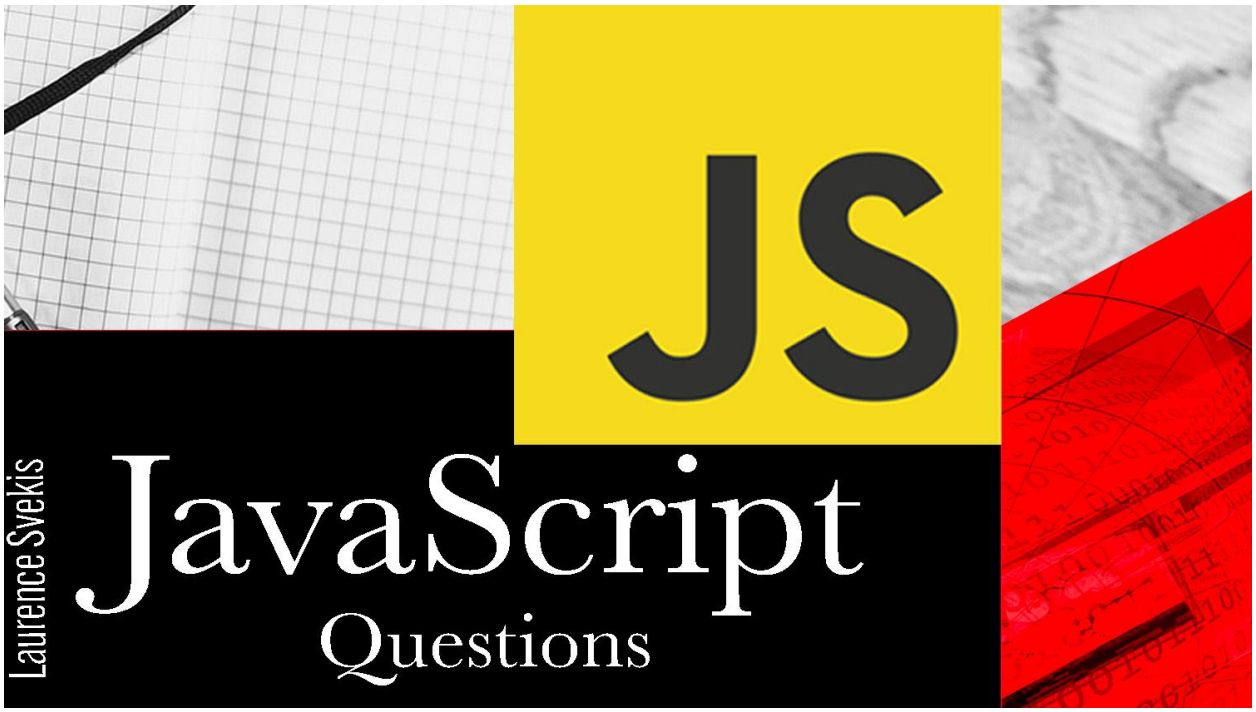
10 Top JavaScript Questions with Examples of Code

JavaScript Coding Questions with Example Code



What is the difference between let and var in JavaScript?

var is a keyword used to declare a variable in JavaScript, whereas let is a relatively new keyword that was introduced in ES6. The main difference between the two is in their scoping rules. Variables declared with var have function-level scope, meaning they are accessible anywhere within the function in which they are declared, even if they are declared within a block. Variables declared with let, on the other hand, have block-level scope, meaning they are only accessible within the block in which they are declared.

Example:

```
function myFunction() {
  var x = 10;
  if (true) {
    var x = 20;
    console.log(x); // Output: 20
  }
  console.log(x); // Output: 20
}
```

```
function myFunction() {
  let x = 10;
  if (true) {
    let x = 20;
    console.log(x); // Output: 20
  }
  console.log(x); // Output: 10
}
```

What is the this keyword in JavaScript?

The this keyword in JavaScript refers to the object that the function is a method of. In other words, it refers to the object that the function is bound to. The value of this depends on how the function is called. When a function is called in the global scope, this refers to the global object (window in a web browser). When a function is called as a method of an object, this refers to that object. When a function is called with the new keyword, this refers to the newly created object.

Example:

```
let person = {
```

```
  firstName: "John",
  lastName: "Doe",
  fullName: function() {
    console.log(this.firstName + " " + this.lastName);
  }
};

person.fullName(); // Output: John Doe
```

What is the difference between == and === in JavaScript?

== is an equality operator that compares two values for equality, but it does type coercion. This means that it tries to convert the values to a common type before comparing them. === is a strict equality operator that compares two values for equality without type coercion. It checks if the values are of the same type and have the same value.

Example:

```
console.log(5 == "5"); // Output: true
console.log(5 === "5"); // Output: false
```

What is a closure in JavaScript?

A closure in JavaScript is created when a function returns another function that has access to the variables in the outer function, even after the outer function has returned. This allows the inner function to access and manipulate the variables in the outer function, even if the outer function is no longer executing.

Example:

```
function outerFunction() {
  let count = 0;
  return function innerFunction() {
    count++;
    console.log(count);
  }
}

let counter = outerFunction();
counter(); // Output: 1
counter(); // Output: 2
counter(); // Output: 3
```

What is the forEach() method in JavaScript?

The forEach() method in JavaScript is a method of the Array object that allows you to loop through the elements of an array and perform a function on each element. It takes a callback function as its argument, which is executed for each element in the array. The callback function takes three arguments: the current element, the index of the current element, and the array itself.

Example:

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(number) {
  console.log(number);
});
// Output:
// 1
// 2
// 3
// 4
// 5

numbers.forEach(function(number, index) {
  console.log(index + ": " + number);
});
```

```
// Output:
// 0: 1
// 1: 2
// 2: 3
// 3: 4
// 4: 5
```

What is the difference between null and undefined in JavaScript?

null and undefined are both values in JavaScript that represent the absence of a value. null is a value that represents a deliberate non-value, while undefined is a value that represents an unintentional non-value. null is usually assigned by the programmer to indicate that a variable has no value, while undefined is usually the default value of a variable that has not been initialized.

Example:

```
let x;
console.log(x); // Output: undefined

let y = null;
```

```
console.log(y); // Output: null
```

What is the difference between synchronous and asynchronous code in JavaScript?

Synchronous code is code that executes in a sequence, one line after another, and the program waits for each line to finish before moving on to the next. Asynchronous code, on the other hand, allows the program to continue executing while a certain task is being performed in the background. Asynchronous code uses callbacks, promises, or async/await to handle the results of the task when it is finished.

Example:

```
// Synchronous code
console.log("Line 1");
console.log("Line 2");
console.log("Line 3");

// Output:
// Line 1
// Line 2
// Line 3
```

```
// Asynchronous code using setTimeout()
console.log("Line 1");
setTimeout(function() {
  console.log("Line 2");
}, 2000);
console.log("Line 3");

// Output:
// Line 1
// Line 3
// Line 2 (after 2 seconds)
```

What is a promise in JavaScript?

A promise in JavaScript is an object that represents a value that may not be available yet, but will be resolved at some point in the future. It is used to handle asynchronous code and provides a way to chain asynchronous operations together. A promise can be in one of three states: pending (initial state), fulfilled (resolved successfully), or rejected (resolved with an error).

Example:

```
let promise = new Promise(function(resolve, reject) {
```

```
  setTimeout(function() {
    resolve("Success!");
  }, 2000);
});

promise.then(function(result) {
  console.log(result); // Output: Success!
}).catch(function(error) {
  console.log(error);
});
```

What is the difference between slice() and splice() in JavaScript?

slice() and splice() are both methods that can be used to manipulate arrays in JavaScript. The main difference between the two is that slice() returns a new array with a portion of the original array, while splice() modifies the original array by adding or removing elements.

Example:

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

```
let slicedNumbers = numbers.slice(1, 3);
console.log(slicedNumbers); // Output: [2, 3]

let splicedNumbers = numbers.splice(1, 2);
console.log(splicedNumbers); // Output: [2, 3]
console.log(numbers); // Output: [1, 4]
```

What is hoisting in JavaScript?

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their respective scopes before code execution. This means that variables and functions can be used before they are declared, as long as they are declared in the same scope.

Example:

```
console.log(x); // Output: undefined
var x = 5;

// The code above is equivalent to:
var x;
console.log(x); // Output: undefined
x = 5;
```

```
// Hoisting also applies to function declarations
sayHello();

function sayHello() {
  console.log("Hello!");
}
```

In the example above, the variable x is declared after it is used, but it still works because of hoisting. Similarly, the function sayHello() is called before it is declared, but it still works because of hoisting.

Note that hoisting only applies to declarations, not to initializations. In the example above, the variable x is hoisted but its value is still undefined until it is assigned the value of 5.