FOR EMPLOYERS

JOBS COMPANIES ARTICLES SALARIES COURSES MY ITEMS

Top 50 JavaScript Interview Questions With Example Answers

Review these common JavaScript interview questions and answers and practice your coding fundamentals with this guide to ace your next interview.

Written by Akshay Kumar
Published on Aug. 15, 2023

Share Save

well-versed in the fundamentals but you also should have some grasp on how to

COMMON JAVASCRIPT INTERVIEW QUESTIONS

1. What are the different data types in JavaScript?
2. What is hoisting in JavaScript?
3. What is the difference between null and undefined?
4. What are closures in JavaScript?
5. What is a callback function in JavaScript?
6. What are promises in JavaScript?
7. What is the purpose of the setTimeout() function in Javascript?
8. How can you check if an array includes a certain value?
9. How can you remove duplicates in an array?
10. What is the purpose of async and await in JavaScript?

Below are some tips for preparing for the interview along with some common questions and answers to help you ace your next interview.

JavaScript Interview Questions and Answers With Examples

JavaScript interview questions range from the basics like explaining the different data types in JavaScript to more complicated concepts like generator functions and async and await. Each question will have answers and examples you can use to prepare for

Share Save

JAVASCRIPT FUNDAMENTALS

1. What is JavaScript?

A high-level, interpreted programming language called JavaScript makes it possible to create interactive web pages and online apps with dynamic functionality. Commonly referred to as the universal language, Javascript is primarily used by developers for front-end and back-end work.

2. What are the different data types in JavaScript?

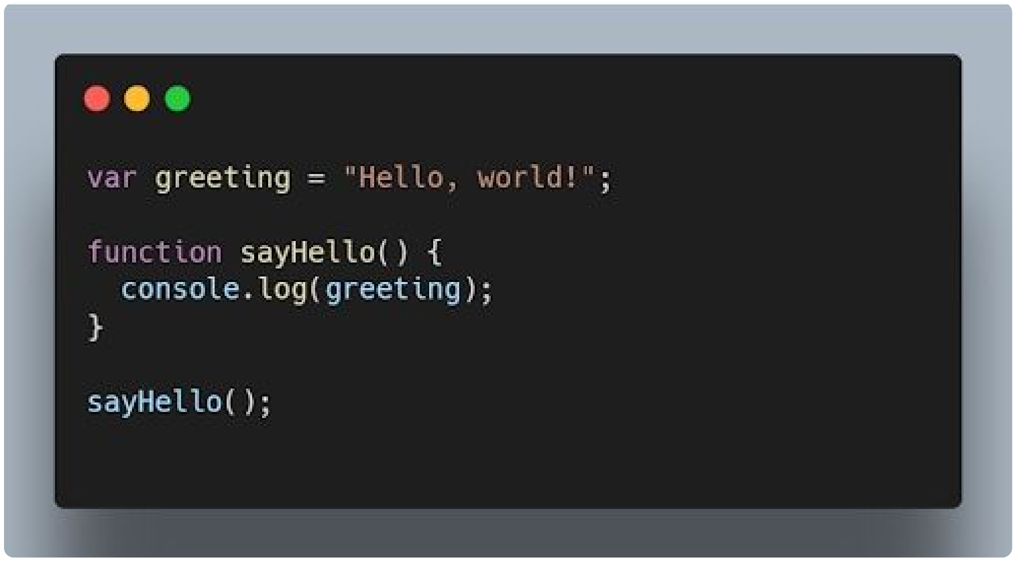
JavaScript has six primitive data types:

- Number
- String
- Boolean
- Null
- Undefined
- Symbol

It also has two compound data types:

the top of their scope. This means that variables and functions can be used before they are declared, as long as they are declared before they are used in a function.

For example, the following code will print `"Hello, world!"` even though the greeting variable is not declared until after the `console.log()` statement.



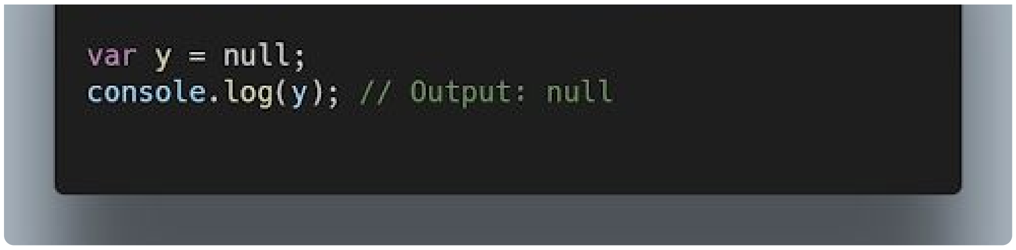
```
var greeting = "Hello, world!";

function sayHello() {
  console.log(greeting);
}

sayHello();
```

JavaScript code printing "Hello World" using hoisting. | Image: Akshay Kumar

4. What is the difference between null and undefined?

`null` is an assignment value that represents no value or an empty value, while `undefined` is a variable that has been declared but not assigned a value.

Share Save



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

JavaScript code outputting null and undefined values. | Image: Akshay Kumar

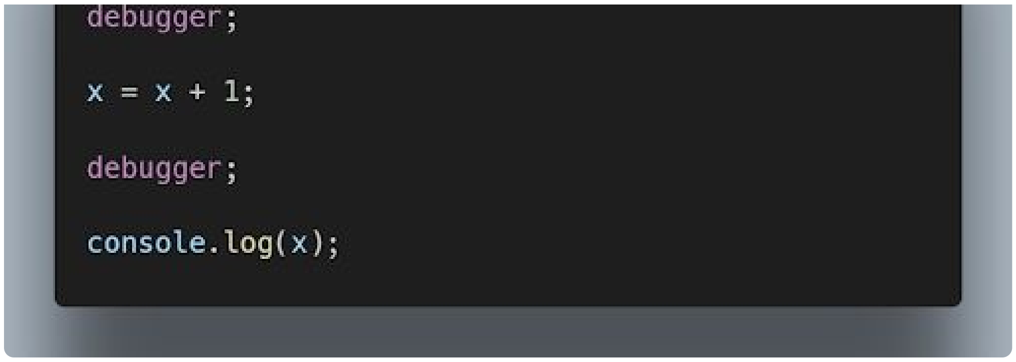
5. Why do we use the word “debugger” in JavaScript?

The word `“debugger”` is used in JavaScript to refer to a tool that can be used to step through JavaScript code line by line. This can be helpful for debugging JavaScript code, which is the process of finding and fixing errors in JavaScript code. To use the `debugger`, you need to open the JavaScript console in your browser. Then, you can use `debugger` commands to comb through your code line by line.

It's essential to know debugging techniques as well as the more general ideas behind code optimization and speed improvement. In addition to operating smoothly, efficient code significantly enhances the user experience.

For example, the following code will print the value of the x variable at each step of the `debugger`.

Share Save

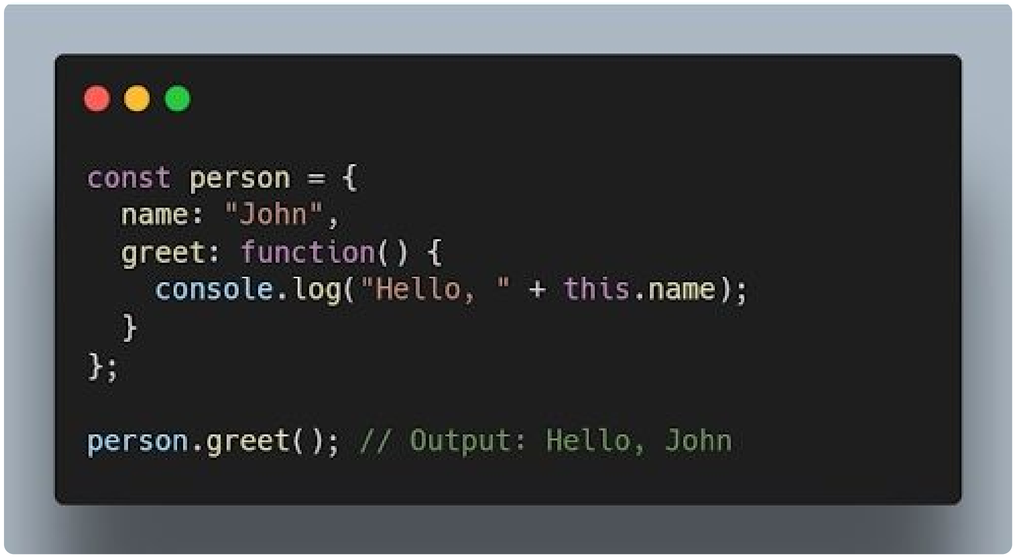


```
debugger;

x = x + 1;

debugger;

console.log(x);
```

JavaScript debugger code printing the value of x at each step. | Image: Akshay Kumar

6. What is the purpose of the "this" keyword in JavaScript?

The `this` keyword refers to the object that is executing the current function or method. It allows access to object properties and methods within the context of that object.

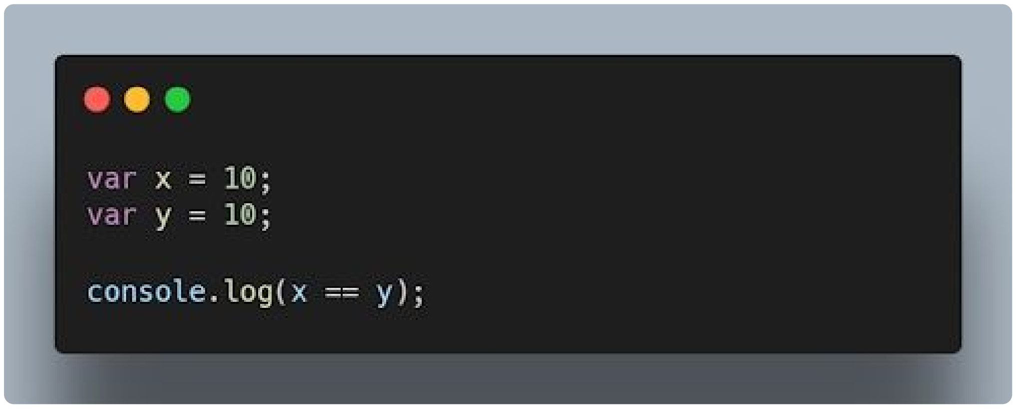


```
const person = {
  name: "John",
  greet: function() {
    console.log("Hello, " + this.name);
  }
};

person.greet(); // Output: Hello, John
```

JavaScript code using the this keyword to output person name. | Image: Akshay Kumar

Share Save

For example, the following code will return true, because the values of the x and y variables are equal.



```
var x = 10;
var y = 10;

console.log(x == y);
```

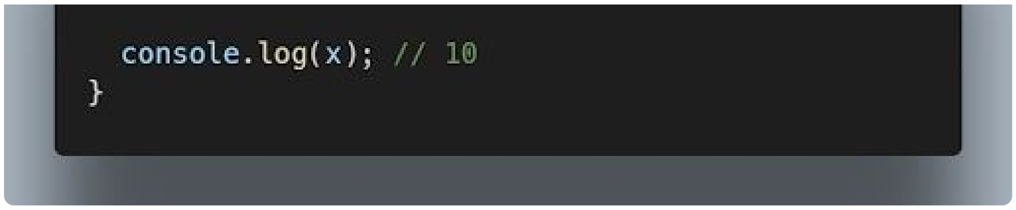
JavaScript equality operator code comparing x and y variables. | Image: Akshay Kumar

8. What is the difference between “var” and “let” keywords in JavaScript?

The `var` and `let` keywords are both used to declare variables in JavaScript. However, there are some key differences between the two keywords.

The `var` keyword declares a global variable, which means that the variable can be accessed from anywhere in the code. The `let` keyword declares a local variable, which means that the variable can only be accessed within the block of code where it is declared.

Share Save

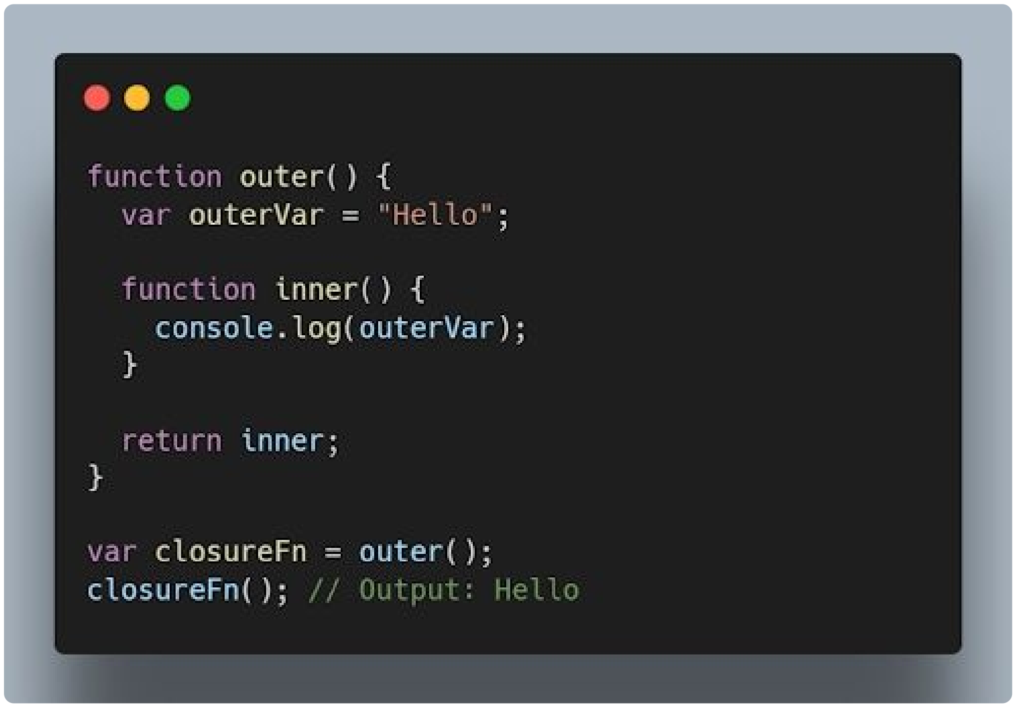


```
  console.log(x); // 10
}
```

JavaScript let keyword example. | Image: Akshay Kumar

9. What are closures in JavaScript?

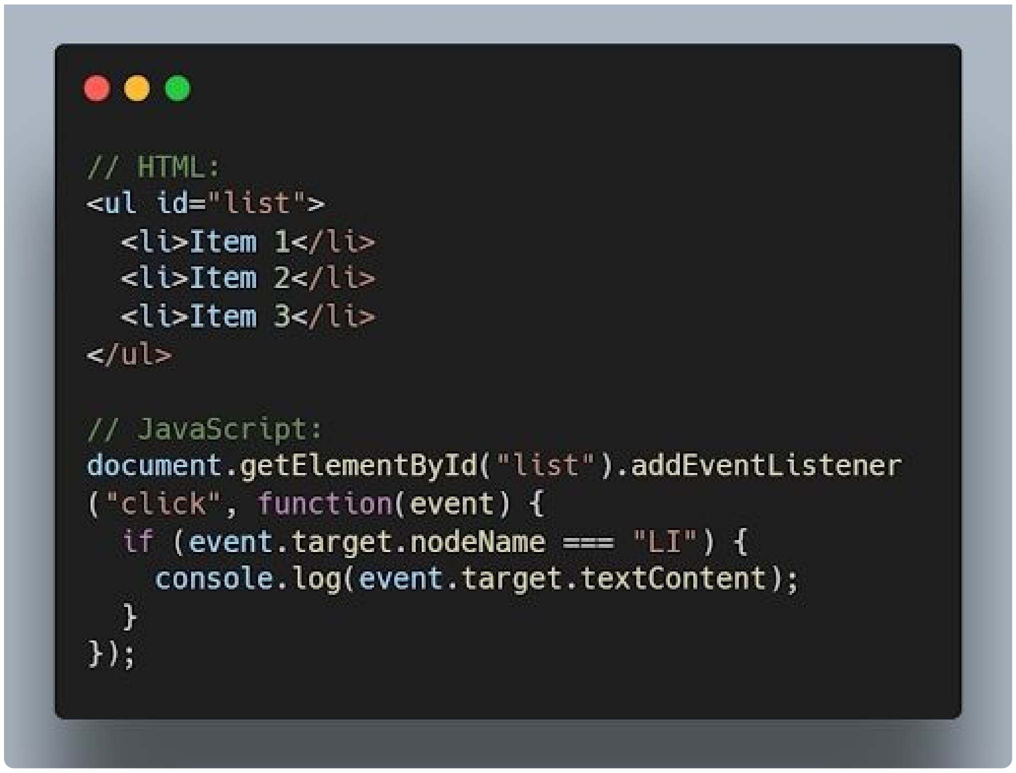
Closures (`closureFn`) are functions that have access to variables from an outer function even after the outer function has finished executing. They "remember" the environment in which they were created.



```
function outer() {
  var outerVar = "Hello";

  function inner() {
    console.log(outerVar);
  }

  return inner;
}

var closureFn = outer();
closureFn(); // Output: Hello
```

JavaScript closure code. | Image: Akshay Kumar

Share Save



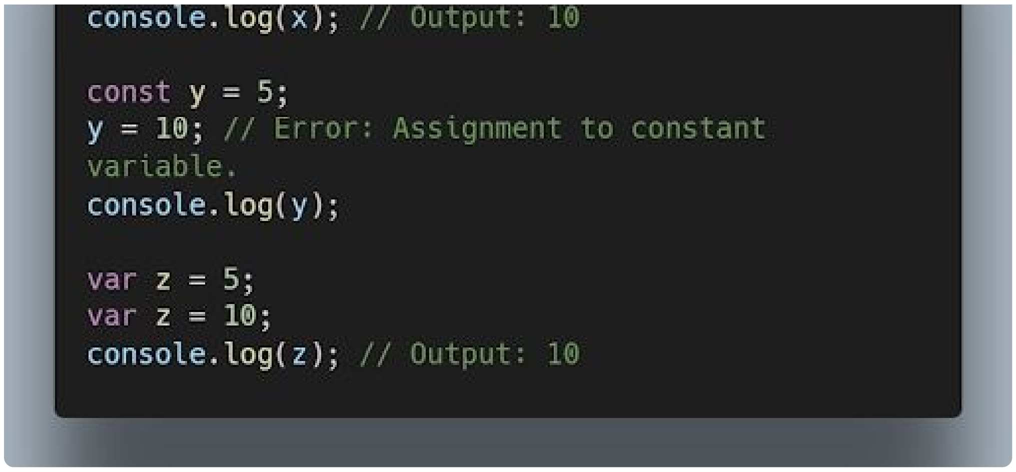
```
// HTML:
<ul id="list">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>

// JavaScript:
document.getElementById("list").addEventListener
("click", function(event) {
  if (event.target.nodeName === "LI") {
    console.log(event.target.textContent);
  }
});
```

JavaScript event delegation code example. | Image: Akshay Kumar

11. What is the difference between “let”, “const”, and “var”?

`let` and `const` were introduced in ES6 and have block scope. `let` is reassignable, and `const` is non-reassignable. `var` is function-scoped and can be redeclared and reassigned throughout the function.

Share Save



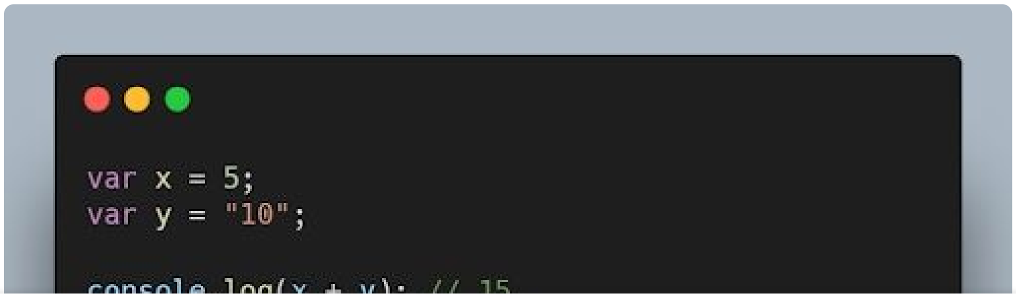
```
console.log(x); // Output: 10

const y = 5;
y = 10; // Error: Assignment to constant
variable.
console.log(y);

var z = 5;
var z = 10;
console.log(z); // Output: 10
```

JavaScript let, const and var keywords with outputs. | Image: Akshay Kumar

12. What is implicit type coercion in JavaScript?

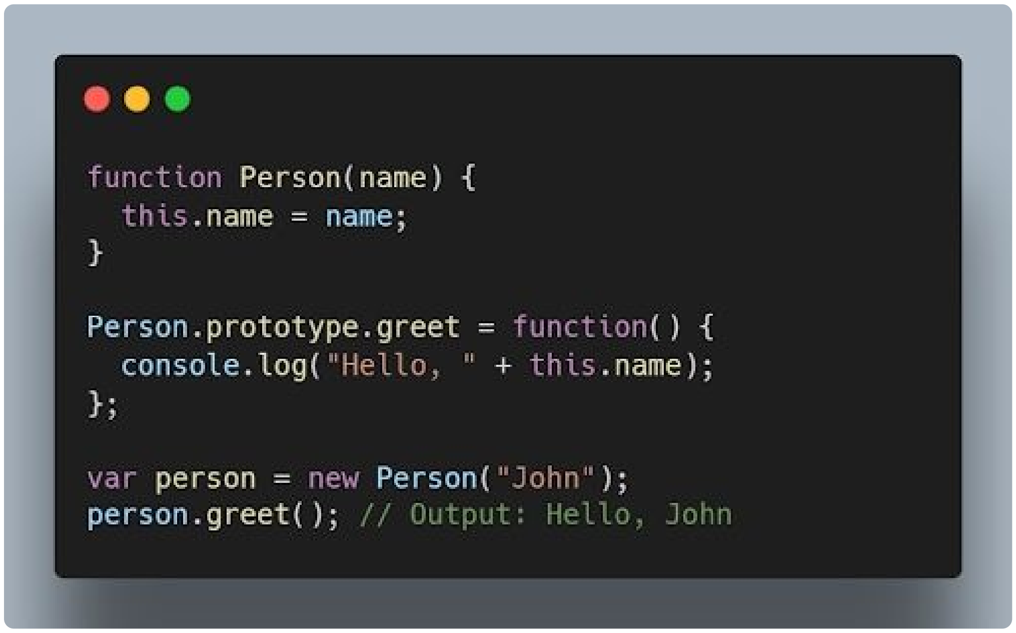
Implicit type coercion is a JavaScript concept that refers to the process of converting a value from one type to another. If you try to add a string to a number, JavaScript will implicitly coerce the string to a number before performing the addition operation.

For example, the following code will add the string `"10"` to the number `5`. This is because JavaScript will implicitly coerce the string `"10"` to a number before performing the addition operation.



```
var x = 5;
var y = "10";

console.log(x + y); // 15
```

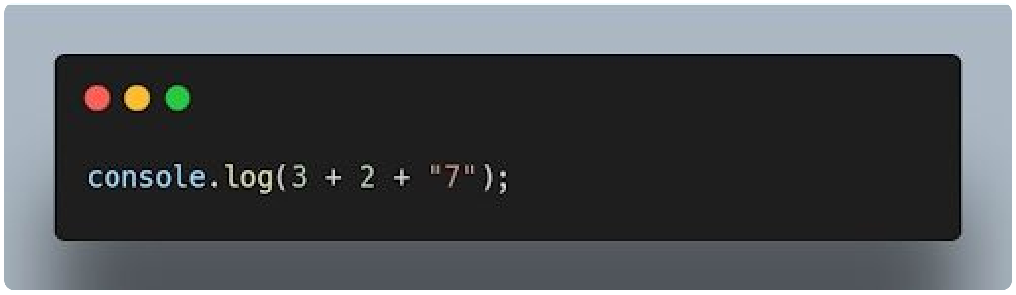
JavaScript object has a prototype, which provides properties and methods that can be accessed by that object.



```
function Person(name) {
  this.name = name;
}

Person.prototype.greet = function() {
  console.log("Hello, " + this.name);
};

var person = new Person("John");
person.greet(); // Output: Hello, John
```

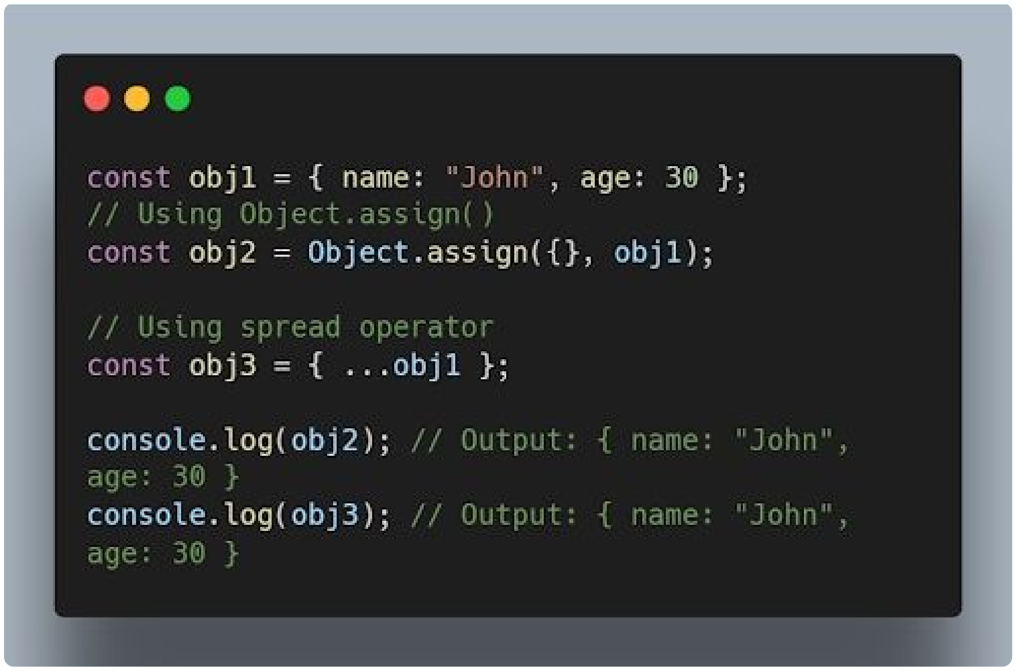
JavaScript prototype code example. | Image: Akshay Kumar

14. What is the output of the following code?



```
console.log(3 + 2 + "7");
```

JavaScript console.log code. | Image: Akshay Kumar



```
const obj1 = { name: "John", age: 30 };
// Using Object.assign()
const obj2 = Object.assign({}, obj1);

// Using spread operator
const obj3 = { ...obj1 };

console.log(obj2); // Output: { name: "John",
age: 30 }
console.log(obj3); // Output: { name: "John",
age: 30 }
```

JavaScript code for cloning an object using object.assign() and ... operators. | Image: Akshay Kumar

MORE ON JAVASCRIPT

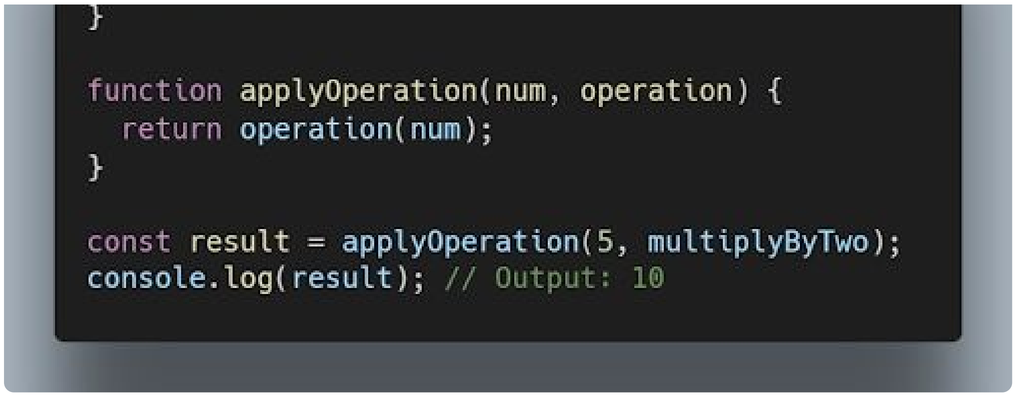
JavaScript Question Mark (?) Operator Explained

INTERMEDIATE CONCEPTS

16. What are higher-order functions in JavaScript?

Higher order functions are functions that can accept other functions as arguments or return functions as their results. They enable powerful functional programming patterns in JavaScript.

Share Save

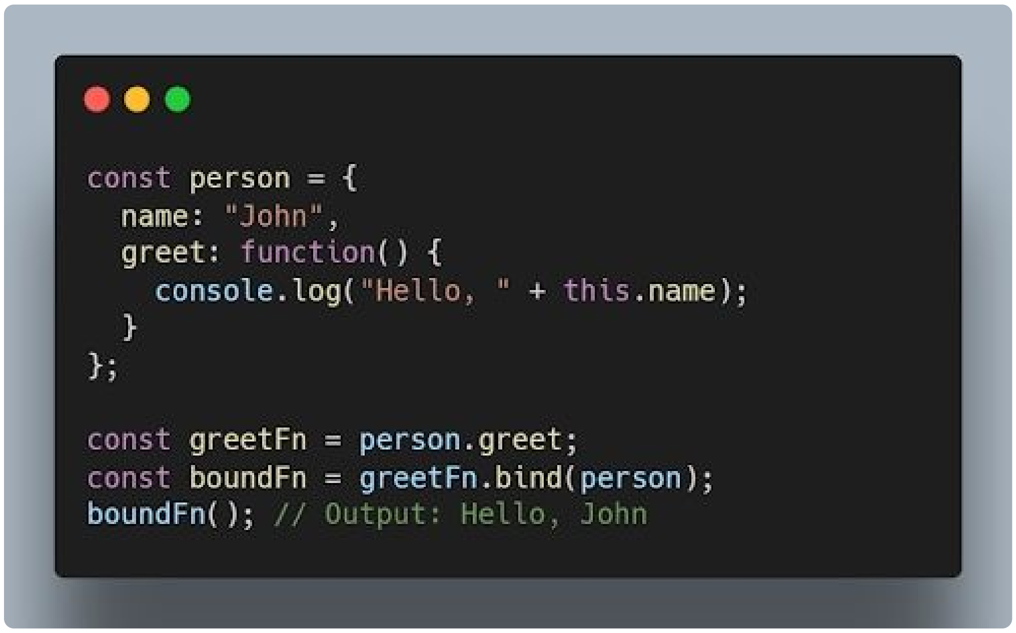


```
}

function applyOperation(num, operation) {
  return operation(num);
}

const result = applyOperation(5, multiplyByTwo);
console.log(result); // Output: 10
```

JavaScript higher order functions code. | Image: Akshay Kumar

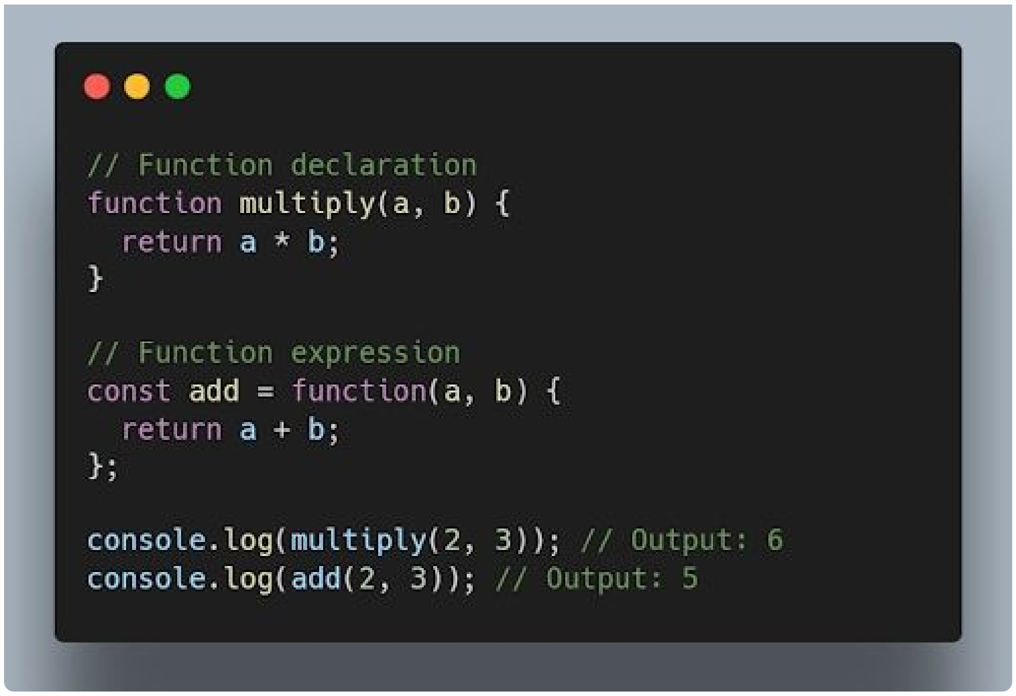
17. What is the purpose of the bind() method in JavaScript?

The `bind()` method is used to create a new function with a specified `this` value and an initial set of arguments. It allows you to set the context of a function permanently.



```
const person = {
  name: "John",
  greet: function() {
    console.log("Hello, " + this.name);
  }
};

const greetFn = person.greet;
const boundFn = greetFn.bind(person);
boundFn(); // Output: Hello, John
```

JavaScript bound() method code example. | Image: Akshay Kumar



```
// Function declaration
function multiply(a, b) {
  return a * b;
}

// Function expression
const add = function(a, b) {
  return a + b;
};

console.log(multiply(2, 3)); // Output: 6
console.log(add(2, 3)); // Output: 5
```

JavaScript code showing differences between declaration and expression. | Image: Akshay Kumar

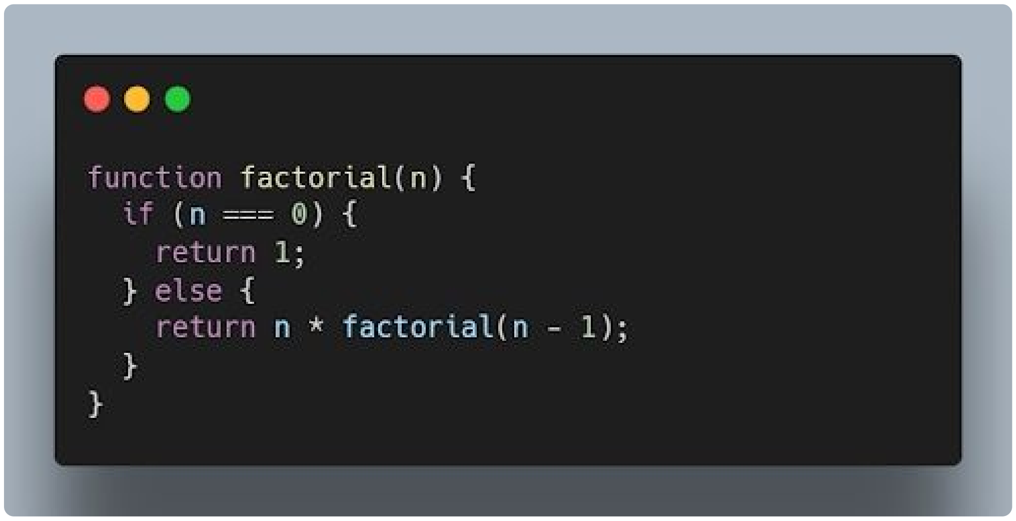
19. What are the different types of errors in JavaScript?

JavaScript can throw a variety of errors, including:

- **Syntax errors:** These errors occur when the JavaScript code is not syntactically correct.
- **Runtime errors:** These errors occur when the JavaScript code is executed and there is a problem.
- **Logical errors:** These errors occur when the JavaScript code does not do what it is supposed to do.

Share Save

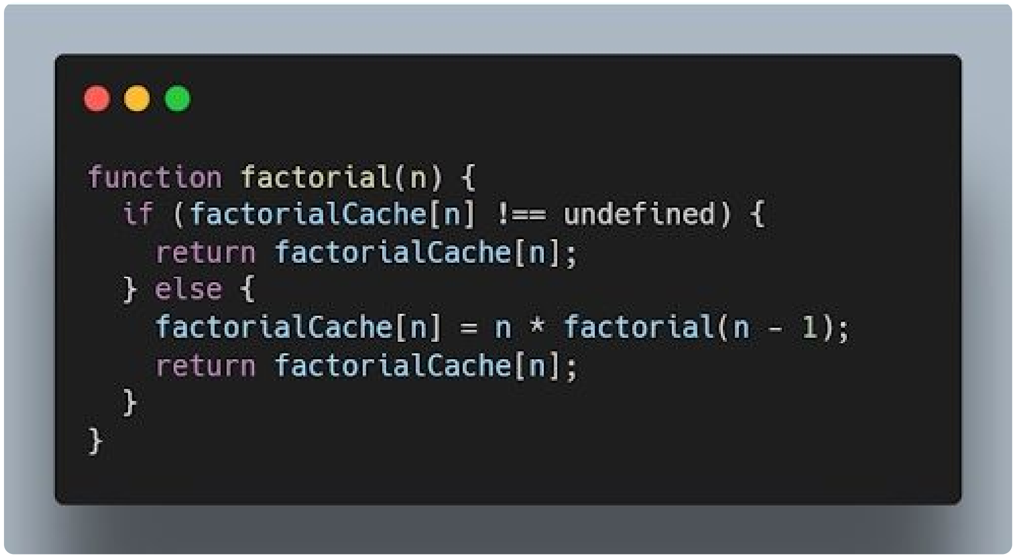
For example, the following code calculates the factorial of a number. The factorial of a number is the product of all the positive integers from one to the number.



```
function factorial(n) {
  if (n === 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

JavaScript code to calculate the factorial of all positive integers from one to a number. | Image: Akshay Kumar

This code can be memoized as follows:

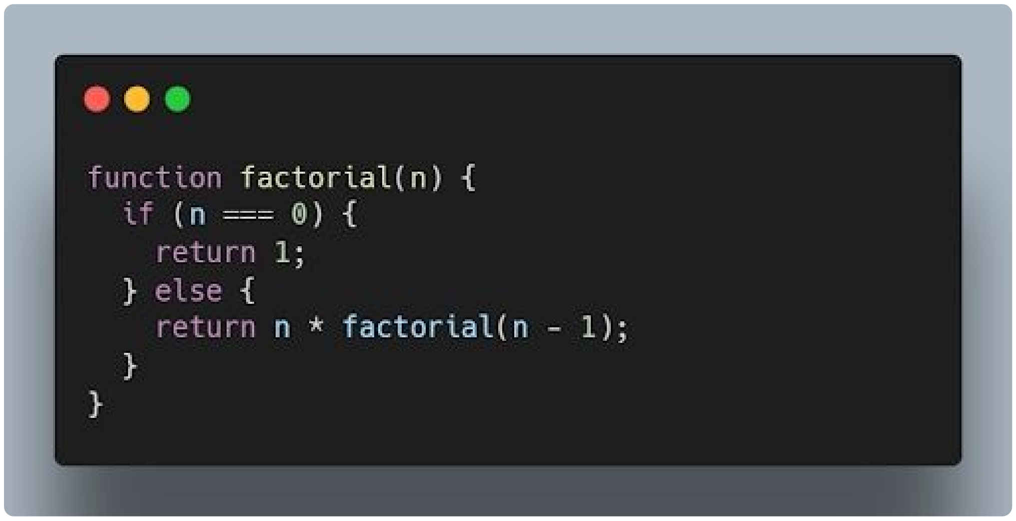


```
function factorial(n) {
  if (factorialCache[n] !== undefined) {
    return factorialCache[n];
  } else {
    factorialCache[n] = n * factorial(n - 1);
    return factorialCache[n];
  }
}
```

Share Save

21. What is recursion in JavaScript?

The following code shows how to use recursion to calculate the factorial of a number:



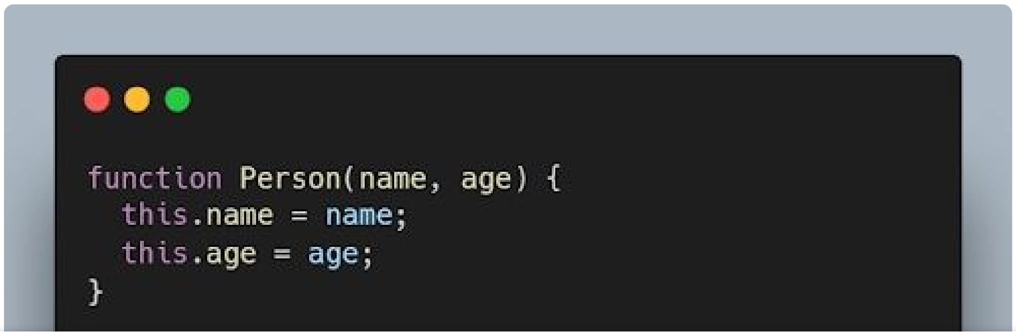
```
function factorial(n) {
  if (n === 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

JavaScript recursion code to solve for factorial of a number. | Image: Akshay Kumar

22. What is the use of a constructor function in JavaScript?

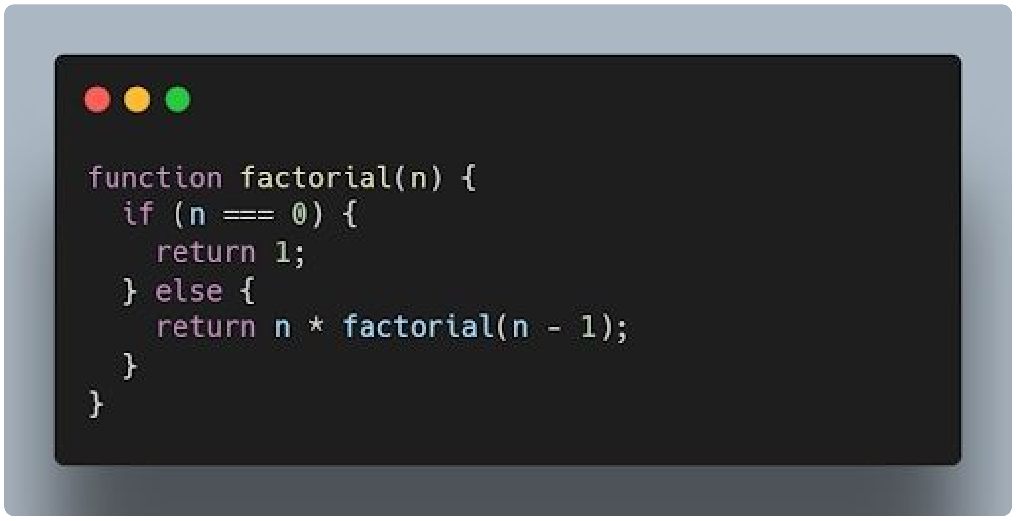
A constructor function is a special type of function that is used to create objects. Constructor functions are used to define the properties and methods of an object.

The following code shows how to create a constructor function:



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

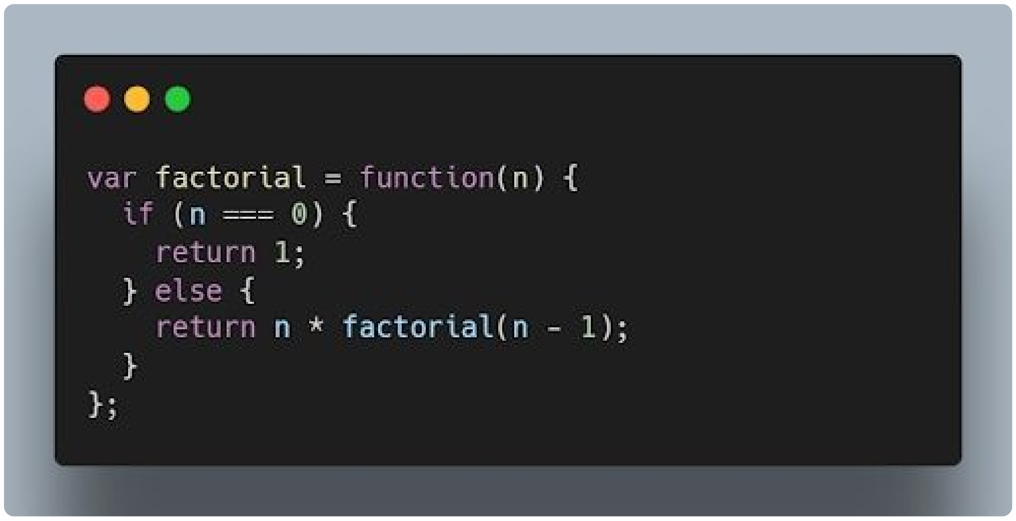
The following code shows an example of a function declaration. This code defines a function named factorial. The factorial function calculates the factorial of a number.



```
function factorial(n) {
  if (n === 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

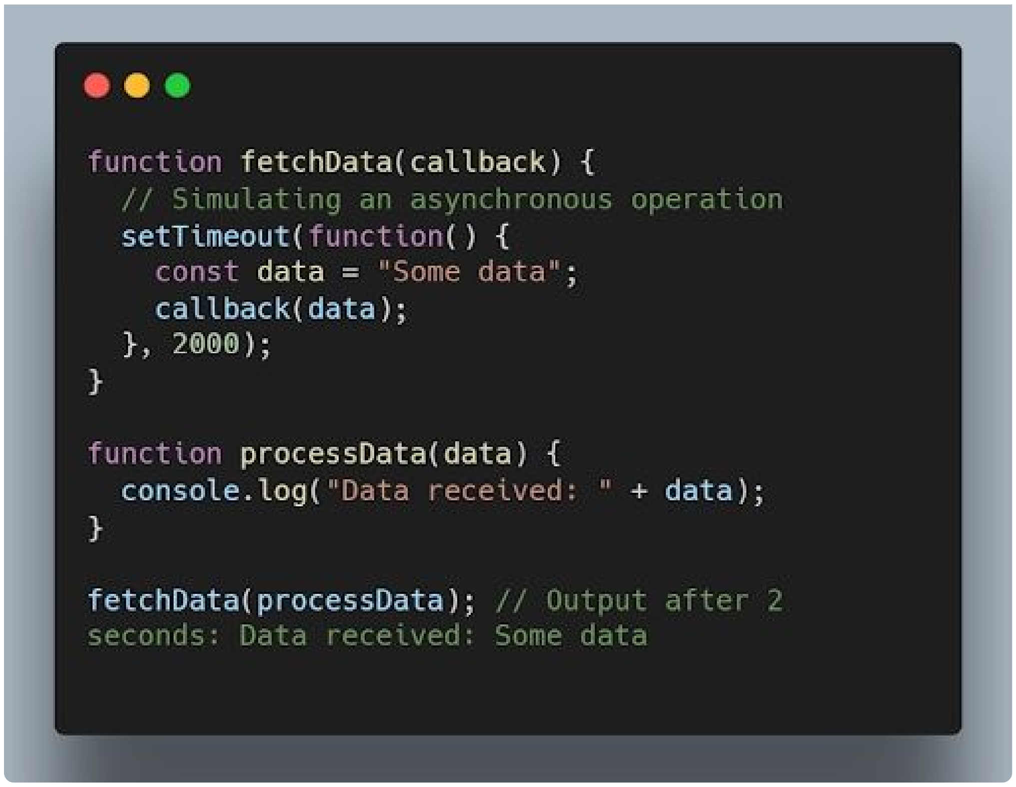
JavaScript function declaration code for a factorial function. | Image: Akshay Kumar

The following code shows an example of a function expression:



```
var factorial = function(n) {
  if (n === 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
};
```

JavaScript function expression for factorial code. | Image: Akshay Kumar



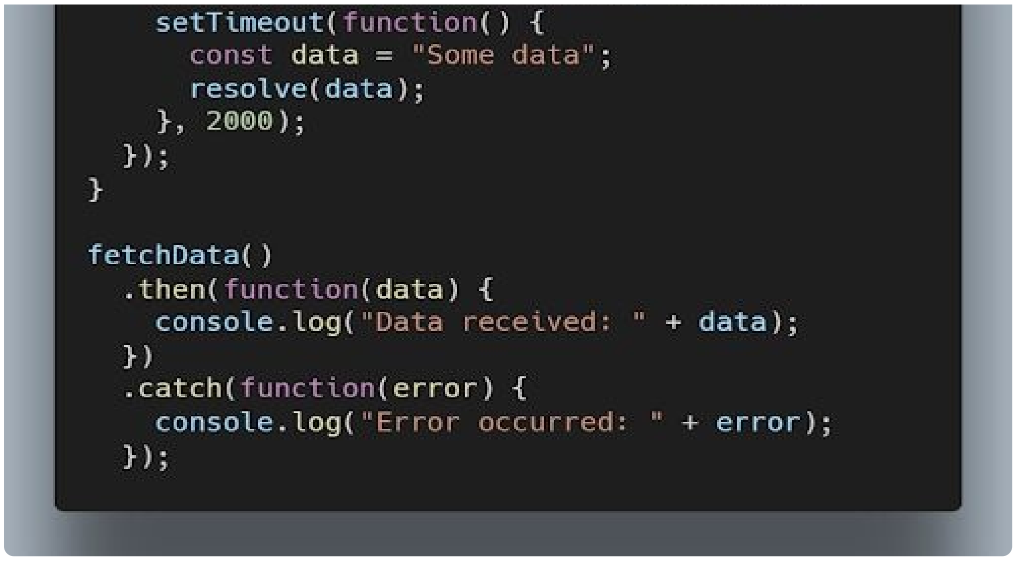
```javascript
function fetchData(callback) {
  // Simulating an asynchronous operation
  setTimeout(function() {
    const data = "Some data";
    callback(data);
  }, 2000);
}

function processData(data) {
  console.log("Data received: " + data);
}

fetchData(processData); // Output after 2
seconds: Data received: Some data
```

JavaScript code for the callback function. | Image: Akshay Kumar

25. What are promises in JavaScript?

Promises are objects used for asynchronous operations. They represent the eventual completion or failure of an asynchronous operation and allow chaining and handling of success or error cases.

Share Save



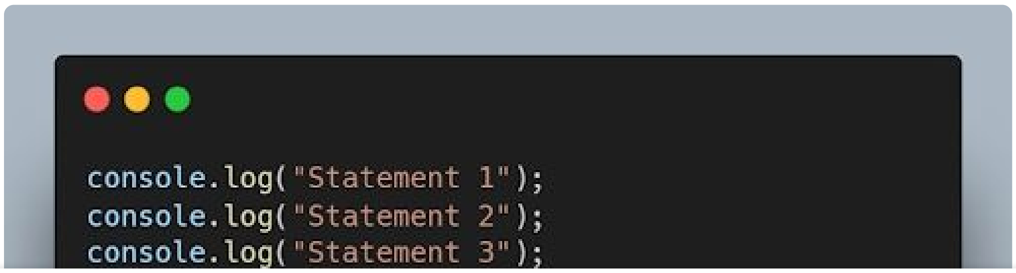
```
      setTimeout(function() {
        const data = "Some data";
        resolve(data);
      }, 2000);
    });
  }

  fetchData()
    .then(function(data) {
      console.log("Data received: " + data);
    })
    .catch(function(error) {
      console.log("Error occurred: " + error);
    });
```

JavaScript promises code example. | Image: Akshay Kumar

26. What is the difference between synchronous and asynchronous programming?

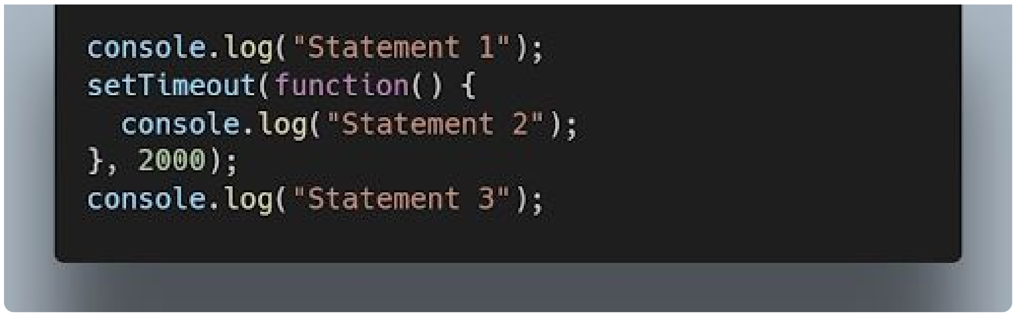
In synchronous programming, the program execution occurs sequentially, and each statement blocks the execution until it is completed. In asynchronous programming, multiple tasks can be executed concurrently, and the program doesn't wait for a task to finish before moving to the next one.

Synchronous coding example:



```
console.log("Statement 1");
console.log("Statement 2");
console.log("Statement 3");
```

Share Save

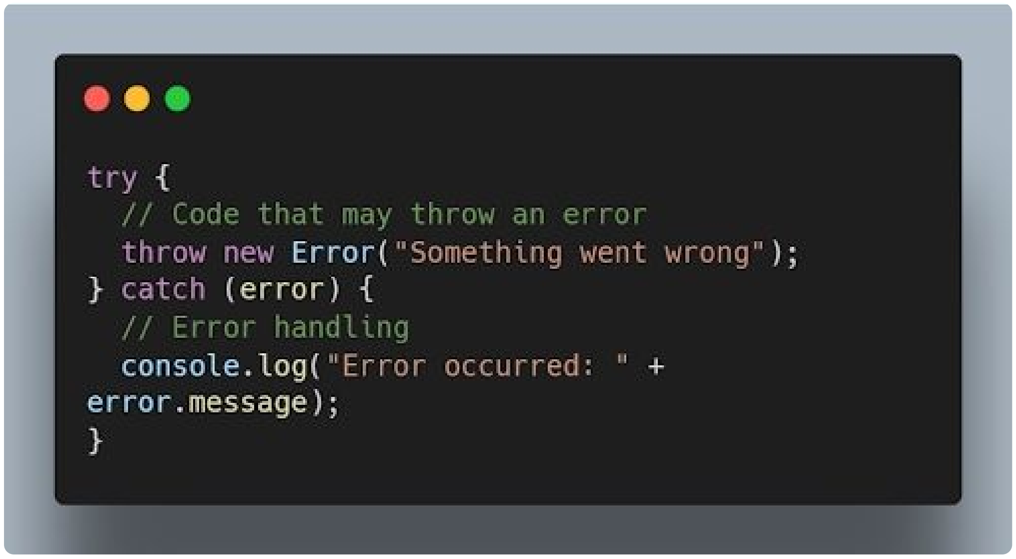


```
console.log("Statement 1");
setTimeout(function() {
  console.log("Statement 2");
}, 2000);
console.log("Statement 3");
```

Asynchronous JavaScript code example. | Image: Akshay Kumar

27. How do you handle errors in JavaScript?

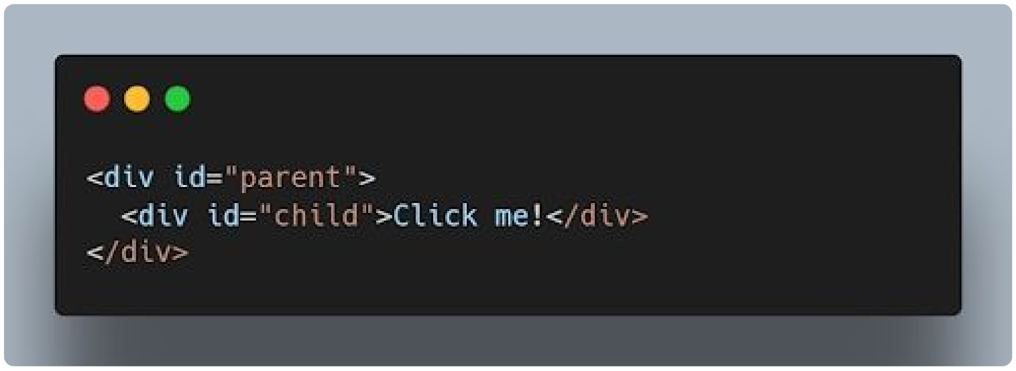
Errors in JavaScript can be handled using `try`-`catch` blocks. The `try` block contains the code that may throw an error, and the `catch` block handles the error and provides an alternative execution path.



```
try {
  // Code that may throw an error
  throw new Error("Something went wrong");
} catch (error) {
  // Error handling
  console.log("Error occurred: " +
error.message);
}
```

JavaScript try-catch blocks of code. | Image: Akshay Kumar

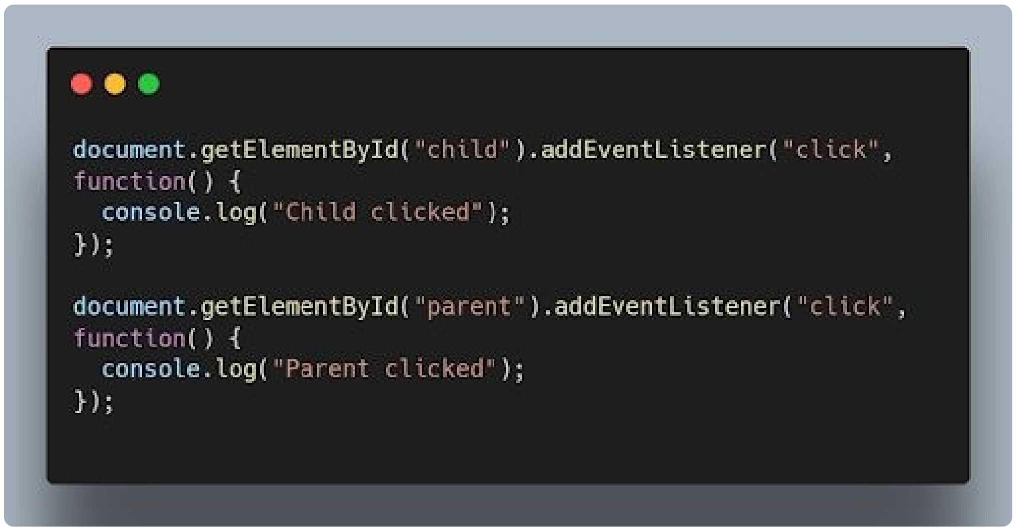
Example:



```
<div id="parent">
  <div id="child">Click me!</div>
</div>
```

JavaScript code using event bubbling. | Image: Akshay Kumar

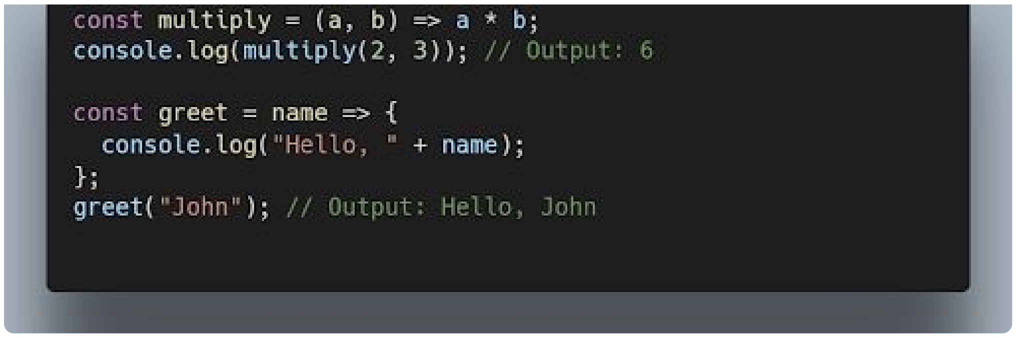
When you click on the child element, both the child and parent event handlers will be triggered, and the output will be:



```
document.getElementById("child").addEventListener("click",
function() {
  console.log("Child clicked");
});

document.getElementById("parent").addEventListener("click",
function() {
  console.log("Parent clicked");
});
```

JavaScript code output after clicking on the child element with event bubbling. | Image: Akshay Kumar

29. What are arrow functions in JavaScript?

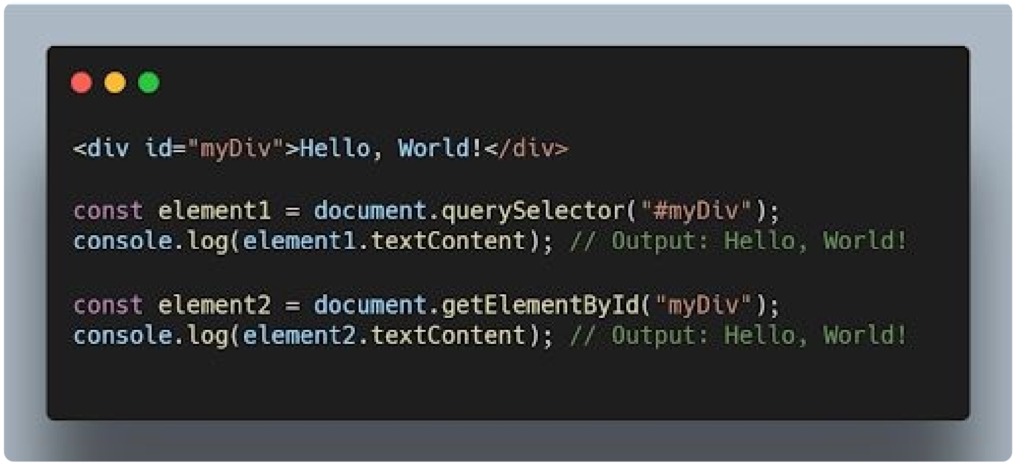


```
const multiply = (a, b) => a * b;
console.log(multiply(2, 3)); // Output: 6

const greet = name => {
  console.log("Hello, " + name);
};
greet("John"); // Output: Hello, John
```

JavaScript arrow functions code example. | Image: Akshay Kumar

30. What is the difference between querySelector and getElementById?

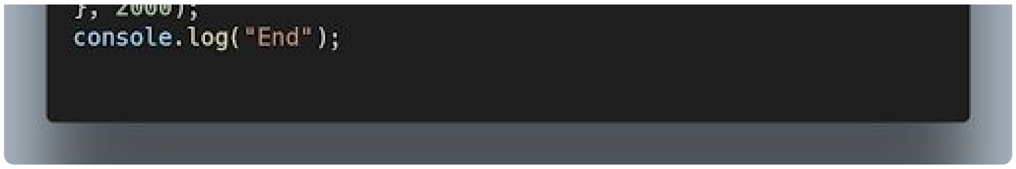
`querySelector` is a more versatile method that allows you to select elements using CSS-like selectors, while `getElementById` specifically selects an element with the specified ID.



```
<div id="myDiv">Hello, World!</div>

const element1 = document.querySelector("#myDiv");
console.log(element1.textContent); // Output: Hello, World!

const element2 = document.getElementById("myDiv");
console.log(element2.textContent); // Output: Hello, World!
```

JavaScript code comparing querySelector and getElementByID methods. | Image: Akshay Kumar

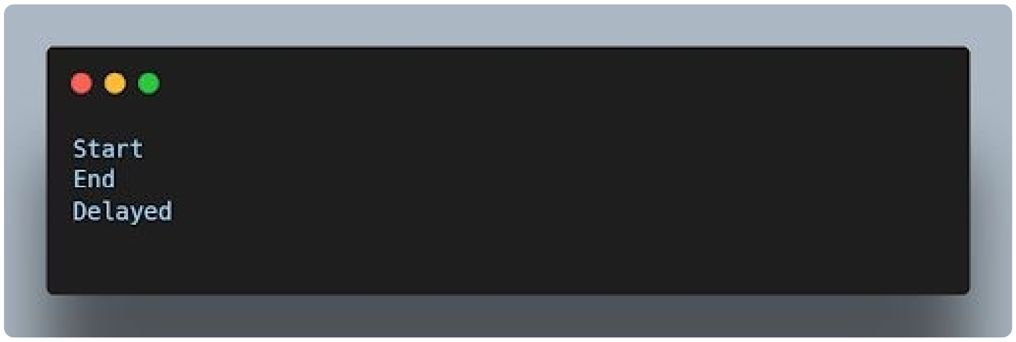
Share Save



```
}, 2000);
console.log("End");
```

JavaScript setTimeout() function code. | Image: Akshay Kumar

Output after two seconds:

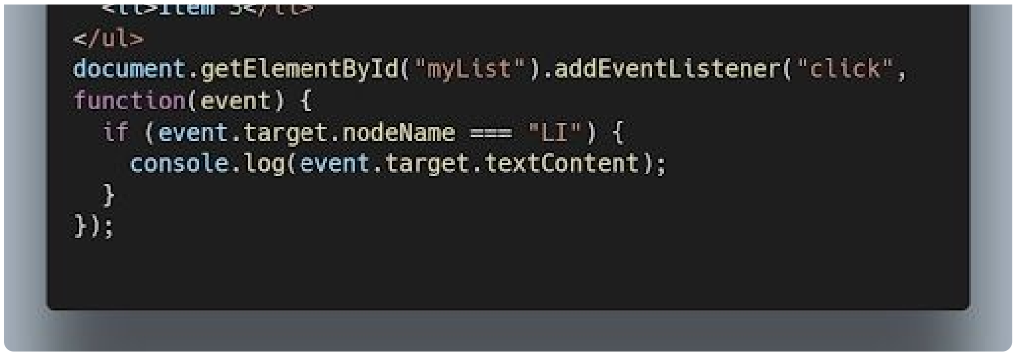


```
Start
End
Delayed
```

JavaScript setTimeout code output after two seconds. | Image: Akshay Kumar

32. What is event delegation and why is it useful?

Event delegation is a technique where you attach a single event listener to a parent element to handle events occurring on its child elements. It's useful for dynamically created elements or when you have a large number of elements.

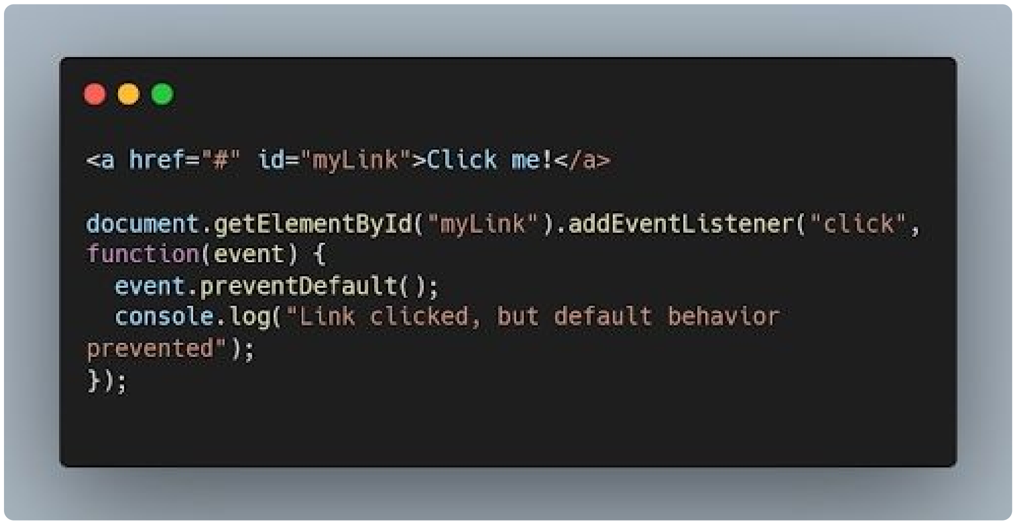


```
  <li>Item 3</li>
</ul>
document.getElementById("myList").addEventListener("click",
function(event) {
  if (event.target.nodeName === "LI") {
    console.log(event.target.textContent);
  }
});
```

JavaScript event delegation code example. | Image: Akshay Kumar

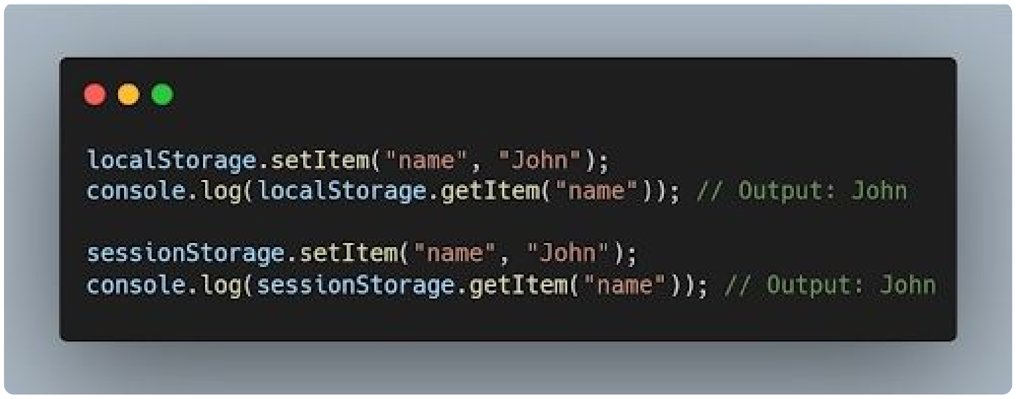
33. How can you prevent the default behavior of an event in JavaScript?

You can use the `preventDefault()` method on the event object within an event handler to prevent the default behavior associated with that event.



```
<a href="#" id="myLink">Click me!</a>

document.getElementById("myLink").addEventListener("click",
function(event) {
  event.preventDefault();
  console.log("Link clicked, but default behavior
prevented");
});
```

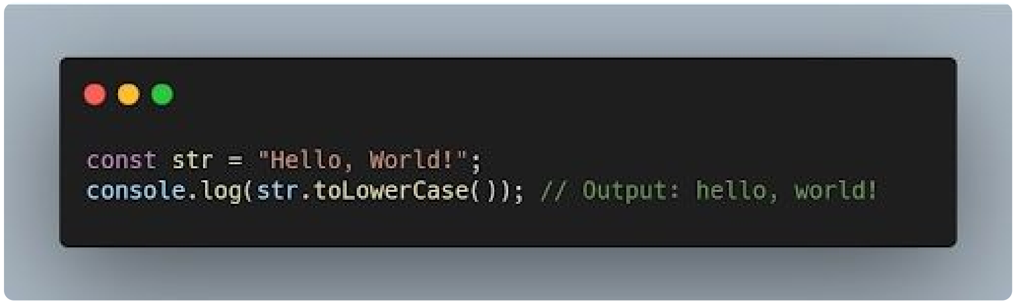
JavaScript preventDefault() method code example. | Image: Akshay Kumar within the same tab or window.


```
localStorage.setItem("name", "John");
console.log(localStorage.getItem("name")); // Output: John

sessionStorage.setItem("name", "John");
console.log(sessionStorage.getItem("name")); // Output: John
```

JavaScript localStorage and sessionStorage code comparisons. | Image: Akshay Kumar

35. How can you convert a string to lowercase in JavaScript?

You can use the `toLowerCase()` method to convert a string to lowercase in JavaScript.

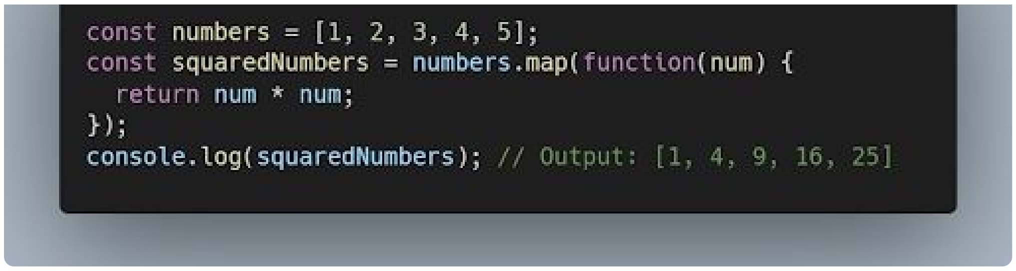

```
const str = "Hello, World!";
console.log(str.toLowerCase()); // Output: hello, world!
```

JavaScript toLowerCase() code example. | Image: Akshay Kumar

ADVANCED CONCEPTS

Share Save

computation on each element. It returns a new array with the results of the



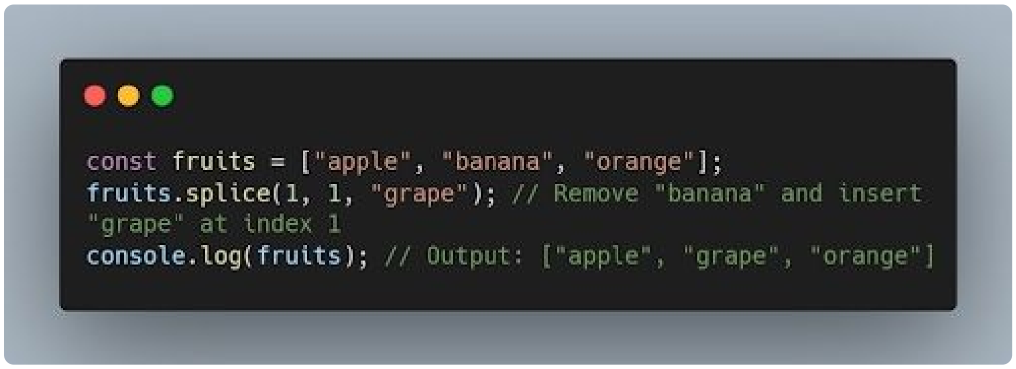
```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(function(num) {
  return num * num;
});
console.log(squaredNumbers); // Output: [1, 4, 9, 16, 25]
```

JavaScript map() function code example. | Image: Akshay Kumar

37. What is the difference between splice() and slice()?

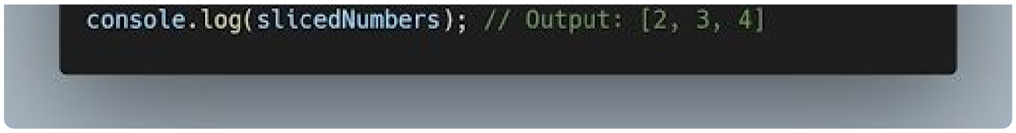
- `splice()` is used to modify an array by adding, removing, or replacing elements at a specific position.
- `slice()` is used to create a new array that contains a portion of an existing array, specified by the starting and ending indices.

Example of `splice()`:



```
const fruits = ["apple", "banana", "orange"];
fruits.splice(1, 1, "grape"); // Remove "banana" and insert
"grape" at index 1
console.log(fruits); // Output: ["apple", "grape", "orange"]
```

JavaScript splice() function code example. | Image: Akshay Kumar

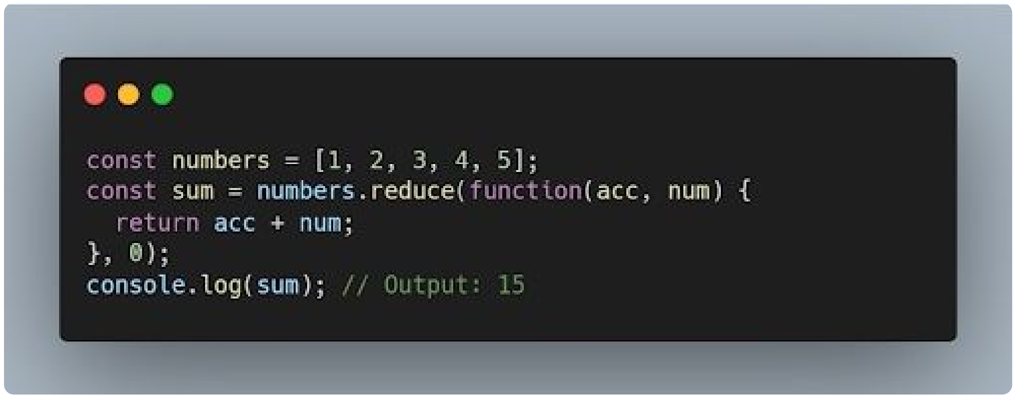


```
console.log(slicedNumbers); // Output: [2, 3, 4]
```

JavaScript slice() code example. | Image: Akshay Kumar

38. What is the purpose of the reduce() function in JavaScript?

The reduce() function is used to reduce an array to a single value by applying a function to each element and accumulating the result.



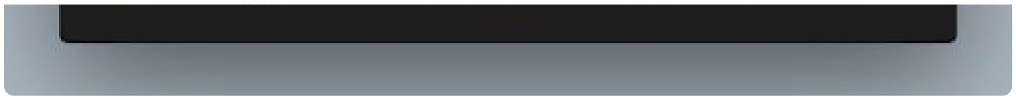
```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce(function(acc, num) {
  return acc + num;
}, 0);
console.log(sum); // Output: 15
```

JavaScript reduce() function example. | Image: Akshay Kumar

39. How can you check if an array includes a certain value in JavaScript?

You can use the `includes()` method to check if an array includes a specific value. It returns true if the value is found, and false otherwise.

Share Save


JavaScript includes() method code on an array. | Image: Akshay Kumar

40. What is the difference between prototype and instance properties in JavaScript?

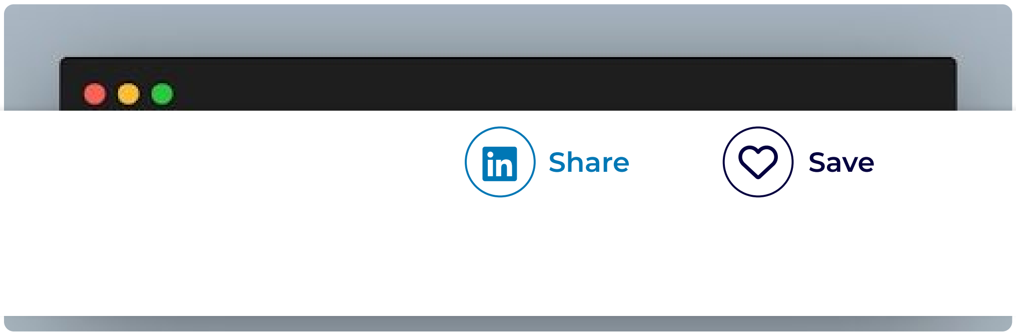
A prototype property is a property that is defined on the prototype object of a constructor function. Instance properties are properties that are defined on individual objects that are created by a constructor function.

Prototype properties are shared by all objects that are created by a constructor function. Instance properties are not shared by other objects.

41. What is the difference between an array and an object in JavaScript?

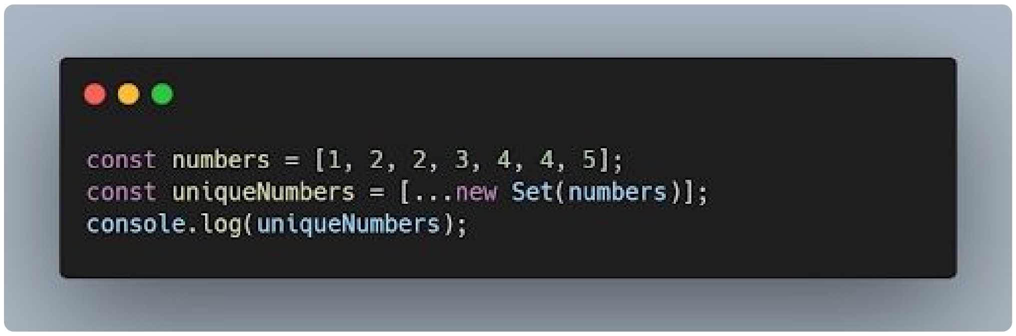
An array is a data structure that can store a collection of values. An object is a data structure that can store a collection of properties.

Arrays are indexed by numbers. Objects are indexed by strings. Arrays can only store primitive data types and objects. Objects can store primitive data types, objects and arrays.



One way to remove duplicates from an array is by using the `Set` object or by using the filter() method with the `indexOf()` method.

Example:



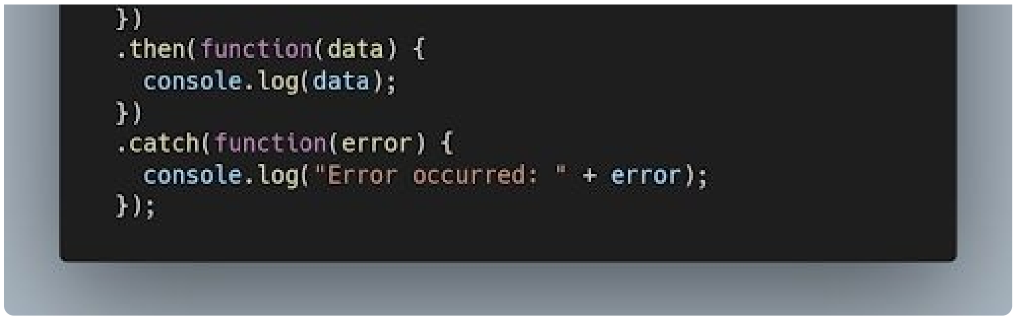
```
const numbers = [1, 2, 2, 3, 4, 4, 5];
const uniqueNumbers = [...new Set(numbers)];
console.log(uniqueNumbers);
```

JavaScript code example removing duplicates using the filter() method. | Image: Akshay Kumar

43. What is the purpose of the fetch() function in JavaScript?

The `fetch()` function is used to make asynchronous HTTP requests in JavaScript. It returns a `Promise` that resolves to the response from the server.

Example:



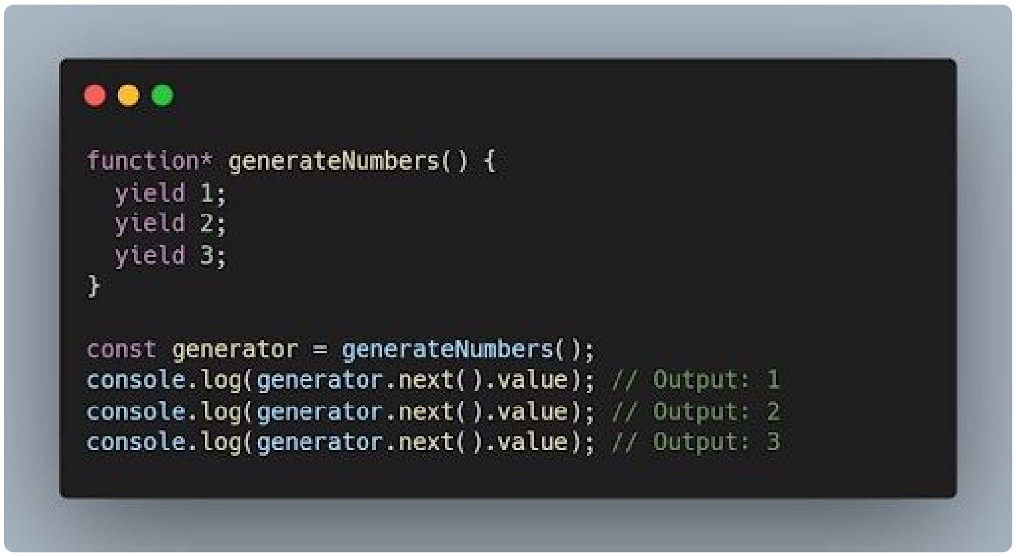
```
  })
  .then(function(data) {
    console.log(data);
  })
  .catch(function(error) {
    console.log("Error occurred: " + error);
  });
```

JavaScript fetch() code function example. | Image: Akshay Kumar

44. What is a generator function in JavaScript?

A generator function is a special type of function that can be paused and resumed during its execution. It allows generating a sequence of values over time, using the `yield` keyword.

Example:



```
function* generateNumbers() {
  yield 1;
  yield 2;
  yield 3;
}

const generator = generateNumbers();
console.log(generator.next().value); // Output: 1
console.log(generator.next().value); // Output: 2
console.log(generator.next().value); // Output: 3
```

Share Save

- **Click**: The click event occurs when a user clicks on an HTML element.
- **Mouseover**: The mouseover event occurs when a user's mouse pointer moves over an HTML element.
- **Keydown**: The keydown event occurs when a user presses a key on the keyboard.
- **Keyup**: The keyup event occurs when a user releases a key on the keyboard.
- **Change**: The change event occurs when a user changes the value of an HTML input element.

46. What are the different ways to access an HTML element in JavaScript?

There are three main ways to access an HTML element in JavaScript:

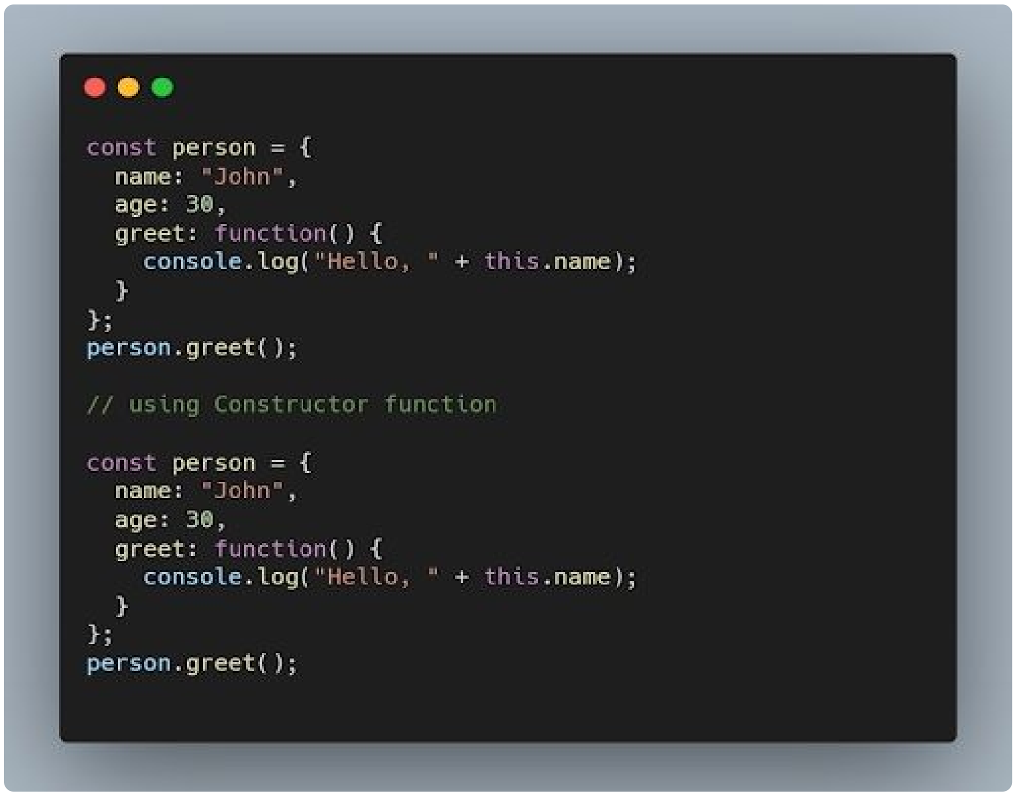
1. **Using the `getElementById()` method:** The `getElementById()` method takes a string as an argument and returns the HTML element with the specified ID.
2. **Using the `getElementsByTagName()` method:** The `getElementsByTagName()` method takes a string as an argument and returns an array of all the HTML elements with the specified tag name.
3. **Using the `querySelector()` method**: The `querySelector()` method takes a CSS selector as an argument and returns the first HTML element that matches the selector.

that they can be accessed from anywhere in the code.

48. What are the different ways to create objects in JavaScript?

There are multiple ways to create objects in JavaScript, including object literals, constructor functions, the `Object.create()` method and the class syntax introduced in ECMAScript 2015 (ES6).

Example using object literals:



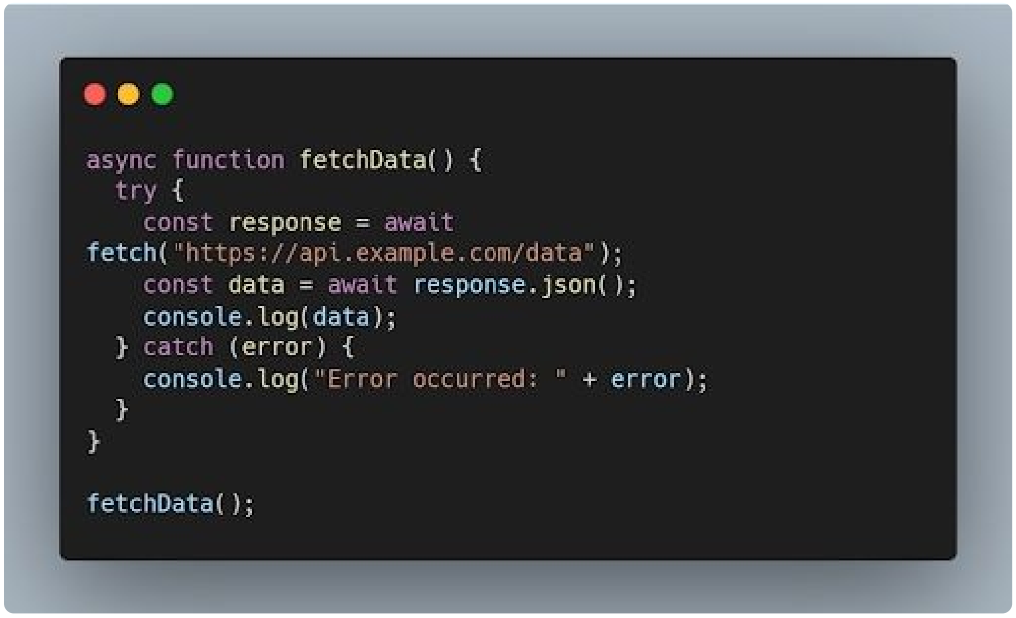
```
const person = {
  name: "John",
  age: 30,
  greet: function() {
    console.log("Hello, " + this.name);
  }
};
person.greet();

// using Constructor function

const person = {
  name: "John",
  age: 30,
  greet: function() {
    console.log("Hello, " + this.name);
  }
};
person.greet();
```

JavaScript object literals code example. | Image: Akshay Kumar

50. What is the purpose of the async and await keywords in JavaScript?

The `async` and `await` keywords are used for handling asynchronous operations in a more synchronous-like manner. The `async` keyword is used to define an asynchronous function, and the `await` keyword is used to pause the execution of an async function until a promise is fulfilled or rejected.

Example:



```
async function fetchData() {
  try {
    const response = await
fetch("https://api.example.com/data");
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.log("Error occurred: " + error);
  }
}

fetchData();
```

JavaScript async and await function code example. | Image: Akshay Kumar

An introduction to the top JavaScript interview questions and answers. | Video: Simplilearn

MORE ON JAVASCRIPT
JavaScript PreventExtensions Method Explained

How to Prepare for a JavaScript Interview

In order to ace a JavaScript interview, you need to be ready for anything. It's important to practice your code, but you should also be able to clearly explain how different functions work, have real world experience working in JavaScript and understand how to debug.

7 WAYS TO PREPARE FOR A JAVASCRIPT INTERVIEW

- Review JavaScript fundamentals

Share Save

7. Mock interviews.

Fortunately, there are some basic steps you can take to be prepared and stand out from other applicants.

1. REVIEW JAVASCRIPT FUNDAMENTALS

Make sure you are well-versed in the foundational concepts of JavaScript, such as data types, variables, operators, control structures, functions and objects.

2. MASTER KEY CONCEPTS

It's also important to study up on key JavaScript topics like promises, asynchronous programming, hoisting, scope, closures, prototypes and ES6 features. You should understand how each of these works.

3. STUDY COMMON INTERVIEW TOPICS

Take the time to review JavaScript interview questions that are regularly asked, including those about closures, prototypes, callbacks, promises, AJAX (asynchronous JavaScript and XML), error handling and module systems. Most interviews follow a code issues. Learn how to read error messages and review basic debugging techniques.

5. PRACTICE CODING

To develop your coding abilities, complete coding tasks and challenges. Concentrate on standard JavaScript data structures and algorithms such arrays, strings, objects, recursion and sorting techniques.

Online resources like LeetCode, CodeChef and HackerRank can be used to practice coding and get ready for interviews. These websites provide a wide variety of coding puzzles and challenges covering a range of subjects and levels of complexity. They are great resources for developing your coding expertise, problem-solving skills, and algorithmic thinking, all of which are necessary for acing technical interviews.

6. BUILD PROJECTS

Take on modest JavaScript projects to get experience and show that you can create useful applications. Showing off your portfolio at the interview is also beneficial. In addition, developers can also work on JavaScript projects to obtain practical experience and show that they are capable of building effective applications. A diversified portfolio can be quite helpful when applying for jobs. Platforms like LeetCode, CodeChef, HackerRank and others enable users to develop projects gradually, starting with minor ones and eventually progressing to more ambitious

7. MOCK INTERVIEWS

It's not just mastering the technical aspect of JavaScript, it's about your body language and how you explain your answers. Many companies are also assessing your ability to work within a team and pair program. The better you can explain your actions and thought process, the more likely you'll be to win over the interviewer.

RECENT CAREER DEVELOPMENT ARTICLES

7 Principles of Lean Software Development

How Location Fingerprinting Can Thwart Fraud

5 Tips to Get You Through Any Tough Situation

Career Development Expert Contributors Interviewing

Javascript Javascript Algorithms Job Interviews

Software Engineering Software Engineering Perspectives

built in EXPERT CONTRIBUTOR NETWORK

Share Save

person accounts of problem-solving on the road to innovation.

FOR EMPLOYERS

JOBS COMPANIES ARTICLES SALARIES COURSES MY ITEMS

Great Companies Need Great People. **That's Where We Come In.**

RECRUIT WITH US

Built In is the online community for startups and tech companies. Find startup jobs, tech news and events.

About

Our Story

Careers

Our Staff Writers

Content Descriptions

Company News

Get Involved

Share Save

FOR EMPLOYERS

JOBS COMPANIES ARTICLES SALARIES COURSES MY ITEMS

Share Feedback

Report a Bug

Tech A-Z

Browse Jobs

Tech Hubs

Built In Austin

Built In Boston

Built In Chicago

Built In Colorado

Built In LA

Built In NYC

Built In San Francisco

Built In Seattle

See All Tech Hubs

Share Save

FOR EMPLOYERS

JOBS COMPANIES ARTICLES SALARIES COURSES MY ITEMS

Share Save