@saadjamilakhtar

Understanding

JAVASCRIPT SCOPES AND CLOSURES

SWIPE

01

SCOPE IN JAVASCRIPT

Scope determines the accessibility and visibility of variables and functions in JavaScript. We have two types of scope: ***global*** (for things declared outside functions) **and** ***local*** (for things declared inside functions)

```
let globalVar = 'I am global';
function foo() {
    let localVar = 'I am local';
    console.log(localVar);   // I am local
    console.log(globalVar);  // I am global
}
foo();
console.log(globalVar);  // I am global
console.log(localVar);    // error: localVar undefined
```

02

Hoisting

JavaScript exhibits a behavior where variable and function declarations are moved to the top of their containing scope during compilation, making them accessible before they appear in the code.

```
console.log(hoistedVariable);    // Output: undefined

var hoistedVariable = 'I am hoisted';

console.log(hoistedVariable);   // Output: I am hoisted
```

@saadjamilakhtar

03

LEXICAL SCOPING

JavaScript employs lexical (static) scoping, meaning that the variables are resolved by their source code position, not runtime location. This lets inner functions access outer function variables, forming a scope hierarchy.

```
function outerFunction() {
    let outerVariable = 'I exist in the outer realm';
    function innerFunction() {
        console.log(outerVariable);
    }
    innerFunction();
}
outerFunction();    // Output: I exist in the outer realm
```

04

CLOSURES IN JAVASCRIPT

A closure is created when an inner function references variables from its outer function, even after the outer function has finished executing.

```
function createGreeting(name) {
    let greeting = 'Hello, ';
    return function() {
        console.log(greeting + name);
    };
}
let greetJohn = createGreeting('John');
greetJohn();      // Output: Hello, John
```

05

PRACTICAL USAGE OF CLOSURES

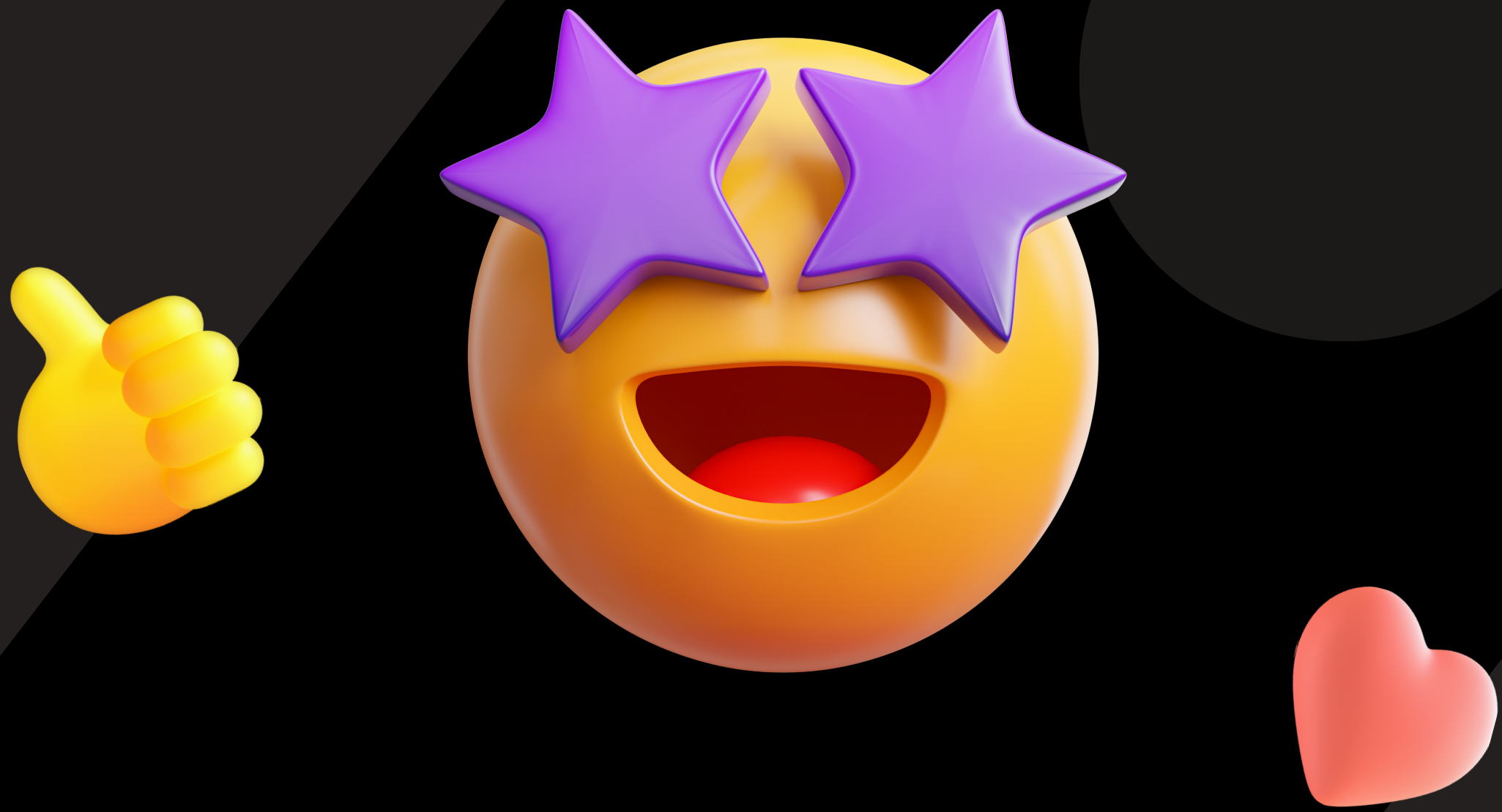
Closures have practical applications in JavaScript. They can be used to create private variables and encapsulated functions. By enclosing variables within a "closure", we can limit their accessibility from the outside and thus ultimately can enhance data privacy and prevent unintended modifications.
Let's check an example

@saadjamilakhtar

06

PRACTICAL USAGE OF CLOSURES

```
function createCounter() {
    let count = 1;
    return {
        increment:  function() { count++; },
        decrement:  function() { count--; },
        getCount:   function() { return count; }
    };
}
let counter = createCounter();
counter.increment();    // count: 2
counter.decrement();    // count: 1
```



Was this post helpful ?
Your thoughts?