You Don't Know JS Yet: Scope & Closures

In JavaScript, scope and closures are fundamental concepts that govern the accessibility of variables and functions within a program. Understanding these concepts is crucial for writing maintainable and efficient code.

You Don't Know JS Yet: Scope & Closures by Kyle Simpson

4.5 out of 5

Language	: English
File size	: 2431 KB
Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 281 pages
Lending	: Enabled

Lexical Scope

JavaScript uses lexical scope, which means that the scope of a variable is determined by its physical location in the code. A variable is accessible within the block in which it is defined, as well as any inner blocks (blocks nested within it).

```
function outer(){const outerVariable = 1; function inner(){const innerVa
```

In this example, the `outerVariable` is defined in the `outer` function and is accessible within both the `outer` and `inner` functions. However, the `innerVariable` is only accessible within the `inner` function.

Variable Environments

Each function in JavaScript has its own variable environment, which is a collection of all the variables that are accessible within that function. The variable environment is created when the function is invoked and is destroyed when the function returns.

When a variable is accessed within a function, the JavaScript engine searches for the variable in the following order:

1. The current function's variable environment
2. The parent function's variable environment (and so on)
3. The global variable environment

If the variable is not found in any of these environments, a `ReferenceError` is thrown.

Function Closures

A function closure is a function that has access to the variable environments of the functions in which it was created. This allows functions to retain access to variables even after the functions that defined them have returned.

```
function outer(){const outerVariable = 1; return function inner(){consol
```

In this example, the `innerFunction` is a closure because it has access to the variable environment of the `outer` function, even though the `outer` function has returned. This is possible because the

`innerFunction` was created within the `outer` function and therefore has access to its variable environment.

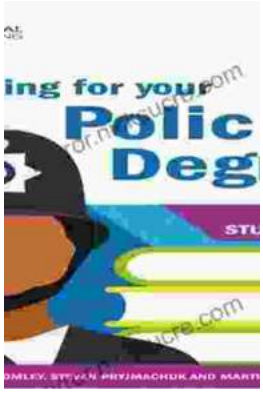
Applications of Closures

Closures have a wide range of applications in JavaScript, including:

- **Encapsulation:** Closures can be used to create private variables and methods, which helps to improve code organization and reduce the risk of data leaks.
- **Event handling:** Closures can be used to capture the state of a function at the time it was created, which is useful for event handling.
- **Partial application:** Closures can be used to create functions that are partially applied, which can simplify code and make it more reusable.

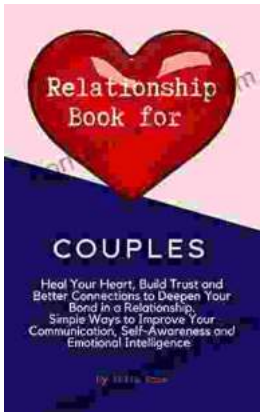
Understanding scope and closures is essential for writing efficient and maintainable JavaScript code. These concepts govern the accessibility of variables and functions within a program and are used in a wide range of applications.

By mastering these concepts, you will be able to write code that is more organized, reusable, and robust.



Studying for Your Policing Degree: Critical Study Skills You Need to Succeed

Pursuing a policing degree is a commendable step towards a fulfilling career in law enforcement. However, to excel in this demanding field, it is imperative...



Heal Your Heart, Build Trust, & Better Connections To Deepen Your Bond

In this article, we will cover tips on how to heal your heart, build trust, and better connections to deepen your bond. Heal Your Heart If...