Scholastic scope the thief quiz Full Version

scholastic scope the thief quiz is a popular assessment tool designed to evaluate students' comprehension and analytical skills regarding the classic short story "The Thief" by Ann Petry. This quiz is commonly used in classrooms to encourage critical thinking, reinforce reading comprehension, and prepare students for discussions and exams related to the story. In this article, we will explore the significance of the quiz, its structure, sample questions, tips for success, and how educators can effectively incorporate it into their teaching strategies.

Understanding the Purpose of the Scholastic Scope The Thief Quiz

Why Use Quizzes in Literature Education?

Quizzes serve as an effective method to assess students' understanding of literary texts. They help teachers identify areas where students excel or struggle, ensuring targeted instruction. Specifically, for stories like "The Thief," quizzes can:

- Test comprehension of plot, characters, and themes
- Encourage close reading and analytical thinking
- Reinforce vocabulary and literary devices used in the story
- Prepare students for essays, discussions, and exams

The Role of Scholastic Scope in Literary Quizzes

Scholastic Scope, a renowned educational magazine, offers curated quizzes aligned with the story's themes and learning objectives. Their "The Thief" quiz is designed to challenge students while providing immediate feedback, making it a valuable resource for both teachers and learners.

Structure of the Scholastic Scope The Thief Quiz

Types of Questions Included

The quiz typically encompasses various question formats to assess different skills:

- **Multiple Choice Questions (MCQs):** Test knowledge of plot details, vocabulary, and literary elements.
- **True or False:** Evaluate understanding of specific facts or concepts from the story.

- **Short Answer Questions:** Encourage students to express their interpretations and analyze characters or themes.
- **Discussion Prompts:** Some quizzes include open-ended questions to foster deeper engagement and critical thinking.

Sample Questions from the Quiz

To give an idea of what students might encounter, here are some typical questions:

- What motivates the main character in "The Thief"?
- Describe the setting of the story and how it influences the plot.
- Identify and explain one literary device used by Ann Petry.
- True or False: The story's main conflict revolves around economic hardship.
- In your own words, summarize the resolution of the story.

Tips for Students Preparing for the "The Thief" Quiz

Effective Study Strategies

Preparing for the quiz requires active engagement with the story. Consider the following strategies:

- **Read Carefully:** Pay attention to details, character actions, and dialogue.
- **Annotate:** Mark important passages and note literary devices or themes.
- **Review Vocabulary:** Understand key words and phrases used in the story.
- **Summarize:** Practice summarizing sections to reinforce comprehension.
- **Discuss:** Talk about the story with classmates or teachers to deepen understanding.

Practice with Sample Questions

Familiarize yourself with the quiz format by practicing with sample questions. This can boost confidence and improve performance.

How Educators Can Incorporate the Quiz Effectively

Using the Quiz as a Teaching Tool

Teachers can leverage the quiz in various ways:

- As an initial assessment to gauge prior knowledge
- As a formative assessment during reading to monitor comprehension
- As a summative assessment after completing the story
- To facilitate group discussions and peer learning

Enhancing Student Engagement

Incorporate interactive activities such as:

- Group quiz competitions
- Creative rewriting of parts of the story
- Role-playing character perspectives

Providing Feedback and Support

Review quiz results with students, highlighting strengths and areas for improvement. Offer additional resources or mini-lessons on challenging topics to reinforce learning.

Additional Resources for "The Thief" Study and Quizzes

Online Practice Quizzes

Many educational websites and platforms offer free practice quizzes based on "The Thief." These resources often include explanations for answers, aiding understanding.

Study Guides and Summaries

Utilize comprehensive study guides that break down the story's themes, characters, and literary devices, providing context for quiz questions.

Discussion Forums and Literature Groups

Participate in online or classroom discussion groups to share insights and clarify doubts about the story.

Conclusion

The scholastic scope the thief quiz is an invaluable resource for reinforcing students' comprehension of Ann Petry's compelling story. By understanding its structure, preparing strategically, and engaging actively with the material, students can enhance their literary analysis skills and perform

well on assessments. Educators, meanwhile, can utilize the quiz to gauge student progress and foster a lively, interactive learning environment. Ultimately, mastering "The Thief" through such quizzes not only improves reading skills but also deepens appreciation for storytelling and literary craft.

Remember: Regular practice and active engagement are key to excelling in literary quizzes. Use available resources wisely, discuss ideas openly, and always strive to understand the deeper meanings behind the story.

Scholastic Scope The Thief Quiz: A Comprehensive Guide to Engaging Readers and Enhancing Literary Comprehension

Introduction

Scholastic Scope The Thief Quiz has emerged as a popular educational tool designed to deepen students' understanding of the classic short story, The Thief by Ann Petry. As teachers and students navigate the complexities of literature, quizzes like these serve as vital instruments not only to assess comprehension but also to foster critical thinking and analytical skills. This article explores the purpose and structure of the quiz, its role in literary education, and how educators can leverage it to promote a richer engagement with the text.

Understanding the Purpose of the Scholastic Scope The Thief Quiz

Promoting Comprehension and Critical Thinking

The primary goal of the Scholastic Scope The Thief Quiz is to evaluate students? grasp of the story?s key themes, characters, and plot developments. Beyond simple recall, the quiz is structured to challenge students to analyze motives, interpret symbolism, and understand the social context of the narrative.

For instance, questions may ask students to identify the motivations behind the protagonist?s actions or to interpret the significance of specific symbols used in the story. This encourages learners not only to remember details but also to think critically about the underlying messages and implications.

Alignment with Educational Standards

The quiz is designed to align with common core standards and literature curricula, ensuring that it complements classroom instruction effectively. It emphasizes skills such as textual analysis, vocabulary development, and the ability to cite evidence from the text, all of which are fundamental to literary studies.

Reinforcement of Literary Elements

Moreover, the quiz reinforces understanding of essential literary elements?such as themes, character development, setting, conflict, and resolution?helping students to recognize these components in context and develop a holistic understanding of the story.

Structure and Content of the Scholastic Scope The Thief Quiz

Types of Questions Included

The quiz comprises various question formats aimed at assessing different levels of comprehension:

- Multiple Choice Questions: These test students? ability to recall factual details, interpret characters? motives, and understand plot points.

- Short Answer Questions: Designed to prompt students to explain themes or analyze specific scenes in their own words.

- Vocabulary Questions: Focused on understanding key terms within context, aiding vocabulary development.

- Analytical Questions: These may require students to compare characters, interpret symbols, or discuss the story?s social message.

Sample Questions Breakdown

For example, a multiple-choice question might ask:

What motivates the protagonist in the story?

A) Desire for wealth

B) A need for acceptance

C) Revenge against an unfair system

D) Curiosity

An analytical question could be:

Explain how the story?s setting influences the protagonist?s decisions.

By incorporating diverse question types, the quiz ensures a comprehensive assessment of students? understanding.

Answer Key and Explanations

Most versions of the quiz provide an answer key with detailed explanations, helping teachers to facilitate discussions and clarify misunderstandings. These explanations often include references to specific parts of the text, encouraging students to revisit the story for deeper insight.

The Role of the Quiz in Classroom Learning

Assessing Student Progress

The Scholastic Scope The Thief Quiz serves as an effective formative assessment tool, allowing educators to gauge how well students have grasped the story?s core elements. It can be used after reading assignments or as part of broader unit assessments to inform instruction.

Promoting Classroom Discussions

Questions from the quiz can spark meaningful classroom conversations about themes such as morality, social injustice, or the importance of empathy. Teachers might use quiz questions as prompts for group discussions or debates, encouraging students to articulate their interpretations and hear diverse perspectives.

Enhancing Literary Skills

Regular engagement with quizzes like this helps students develop skills such as close reading, textual evidence citation, and analytical writing. These skills are transferable across literary works and are vital for success in standardized testing and higher-level coursework.

Strategies for Educators Using the Scholastic Scope The Thief Quiz

Pre-Reading Activities

Before administering the quiz, teachers can engage students with pre-reading discussions about the story?s themes or background context. This primes students for comprehension and helps them connect personal experiences to the narrative.

Guided Reading and Annotation

Encouraging students to annotate the story as they read can enhance understanding. When students identify key passages related to quiz questions, they are better prepared to answer accurately and thoughtfully.

Post-Quiz Analysis and Reflection

Following the quiz, educators should review questions and answers collectively, discussing why certain choices are correct or incorrect. Reflection exercises can deepen comprehension and foster critical thinking.

Differentiated Instruction

Given the diversity in student abilities, teachers can modify the quiz by providing additional hints, offering alternative formats (such as oral questions), or extending questions for advanced learners. This ensures equitable engagement with the material.

Why Use the Scholastic Scope The Thief Quiz?

Engaging Students with Literature

Quizzes like these are more than assessments; they are tools to stimulate curiosity and engagement. When students see assessments as opportunities for exploration rather than just evaluation, their motivation and interest increase.

Reinforcing Learning Objectives

The quiz helps reinforce essential learning objectives?understanding themes, analyzing characters, and interpreting symbolism?all in a structured format that guides both teaching and learning.

Preparing for Standardized Tests

Many standardized tests include questions about literature that require analytical reading and comprehension. Familiarity with quiz formats and question types can boost student confidence and performance.

Conclusion

Scholastic Scope The Thief Quiz exemplifies how well-designed assessments can serve as valuable educational resources, fostering a deeper appreciation and understanding of literature. By

combining diverse question formats with thoughtful analysis, the quiz challenges students to think critically and engage actively with the story. For educators, integrating such quizzes into their teaching repertoire offers a strategic way to assess comprehension, promote meaningful discussions, and develop essential literary skills. As students learn to interpret and analyze stories like The Thief, they cultivate skills that extend beyond the classroom, enriching their overall educational journey and fostering a lifelong love for literature.

QuestionAnswer What is the main theme of 'The Thief' in Scholastic Scope? The main theme centers around honesty and the consequences of theft, exploring how stealing affects relationships and personal integrity. Who is the protagonist in 'The Thief' story in Scholastic Scope? The protagonist is typically a young character who faces a moral dilemma involving theft, often depicted as trying to choose between right and wrong. What lesson does 'The Thief' story aim to teach readers? It teaches the importance of honesty, integrity, and understanding the impact of one's actions on others. How does the story 'The Thief' in Scholastic Scope develop its main conflict? The conflict develops as the protagonist struggles with the temptation to steal and the consequences that follow after their actions are discovered. What are some common questions on the 'The Thief' quiz in Scholastic Scope? Common questions include identifying key events, understanding character motivations, and analyzing the story's theme and moral lessons. How can students prepare for the 'The Thief' quiz in Scholastic Scope? Students should review the story's plot, characters, and themes, and practice answering comprehension and analytical questions related to the story. Are there any vocabulary words from 'The Thief' story that are commonly tested in the quiz? Yes, vocabulary words related to morality, theft, and character traits are often tested to ensure comprehension of the story's context. What types of questions are typically included in the 'The Thief' quiz for Scholastic Scope? The quiz usually includes multiple-choice questions, short answer prompts, and comprehension questions about the story's details and moral lesson. Where can I find practice quizzes or resources to prepare for the 'The Thief' Scholastic Scope quiz? Practice quizzes and resources are often available in the student editions of Scholastic Scope or through teacher-provided study guides and online educational platforms.

Related keywords: Scholastic Scope, The Thief, quiz, literature quiz, reading comprehension, story analysis, vocabulary quiz, educational worksheet, classroom activity, teaching resources

You don t know js scope closures

you don t know js scope closures is a phrase that many JavaScript developers have encountered, often accompanied by confusion or frustration. Understanding scope and closures is fundamental to mastering JavaScript, yet these concepts can be notoriously tricky for beginners and even intermediate programmers. Mastering scope and closures not only helps in writing cleaner,

more efficient code but also prevents bugs related to variable lifetime and accessibility. In this comprehensive guide, we will explore JavaScript scope and closures in depth, breaking down complex ideas into understandable concepts, and providing practical examples to solidify your understanding.

Understanding JavaScript Scope

What is Scope?

Scope in JavaScript determines the visibility and lifetime of variables. It defines where a variable is accessible in the code, preventing conflicts and helping organize code effectively. JavaScript primarily uses lexical scope, meaning the scope of variables is determined by their position in the source code.

Types of Scope in JavaScript

JavaScript has several types of scope:

- **Global Scope**: Variables declared outside any function or block. Accessible throughout the entire script.
- **Function Scope**: Variables declared inside a function are only accessible within that function.
- **Block Scope**: Introduced with ES6, variables declared with let or const inside blocks ({}) are limited to that block.

Variable Declaration Keywords and Their Scope

JavaScript provides three main keywords for variable declaration:

- **var**: Function-scoped. Variables declared with var are hoisted and accessible throughout the entire function, regardless of block boundaries.
- **let**: Block-scoped. Variables are limited to the block in which they are declared.
- **const**: Also block-scoped. Used for variables that shouldn't be reassigned.

Understanding Closures in JavaScript

What is a Closure?

A closure is a function that retains access to variables from its lexical scope even after the outer function has finished executing. Essentially, closures "close over" variables, preserving their state

across multiple invocations.

How Do Closures Work?

When a function is created inside another function, it forms a closure capturing the variables from its outer scope. Even if the outer function has completed, the inner function still has access to those variables, thanks to JavaScript's lexical scoping and closure mechanisms.

Practical Example of a Closure

```javascript

function outerFunction() {

let count = 0; // 'count' is in outerFunction's scope

return function innerFunction() {

count++;

console.log(`Count: ${count}`);

};

}

const counter = outerFunction();

counter(); // Count: 1

counter(); // Count: 2

```

In this example, the inner function retains access to the count variable even after outerFunction has finished executing.

Common Use Cases for Closures

Closures are powerful and are used in various situations:

- **Data Encapsulation**: Hiding variables and exposing only necessary functions.
- **Function Factories**: Creating functions with preset parameters.
- **Event Handlers**: Maintaining state across asynchronous events.
- **Currying and Partial Application**: Pre-filling some arguments of a function.

Common Pitfalls and Misunderstandings

Variables in Closures are References, Not Values

One common misunderstanding is that closures capture the value of variables at the time of closure creation. In reality, they capture references to variables, meaning if the variable changes later, the closure sees the updated value.

Example:

```javascript

function createFunctions() {

const functions = [];

for (var i = 0; i
functions.push(function() {

console.log(i);

});

}

return functions;

}

const funcs = createFunctions();

funcs[0](); // Outputs: 3

funcs[1](); // Outputs: 3

funcs[2](); // Outputs: 3
```

```

Solution:

Use IIFE or block scope:

```javascript

function createFunctions() {

const functions = [];

for (let i = 0; i
(function(index) {

functions.push(function() {

console.log(index);

});

})(i);

}

return functions;

}

const funcs = createFunctions();

funcs[0](); // Outputs: 0

funcs[1](); // Outputs: 1

funcs[2](); // Outputs: 2

```

Or, using ES6:
```

```javascript

function createFunctions() {

const functions = [];

for (let i = 0; i
functions.push(function() {

console.log(i);

});

}

return functions;

}

```

Understanding Variable Lifetimes

Variables declared inside a closure remain alive as long as the closure exists. This can lead to memory leaks if not managed carefully, especially when closures hold references to large objects or DOM elements.

Best Practices for Working with Scope and Closures

Limit the Scope of Variables

Declare variables in the narrowest scope possible to avoid unintended side effects and improve code clarity.

Use let and const Instead of var

These keywords provide block scope, reducing bugs related to variable hoisting and scope leakage.

Be Mindful of Closure Variables in Loops

When creating functions inside loops, ensure each function captures the intended variable value,

often by using an IIFE or block scope.

Manage Memory Usage

Avoid holding onto closures longer than necessary, especially when they reference large objects or DOM nodes.

Practical Examples and Use Cases

Counter with Closure

```javascript

function createCounter() {

let count = 0;

return {

increment() {

count++;

return count;

},

decrement() {

count--;

return count;

},

getCount() {

return count;

}
```

```
};

}

const counter = createCounter();

console.log(counter.increment()); // 1

console.log(counter.increment()); // 2

console.log(counter.getCount()); // 2

console.log(counter.decrement()); // 1
```

This pattern encapsulates state in a closure, preventing external code from modifying the internal variable directly.

Private Variables in JavaScript

Using closures, you can emulate private variables:

```javascript
function Person(name) {

let _name = name; // private variable

return {

getName() {

return _name;

},

setName(newName) {

_name = newName;
```

```
}

};

}

const person = Person('Alice');

console.log(person.getName()); // Alice

person.setName('Bob');

console.log(person.getName()); // Bob
```

Summary: Demystifying Scope and Closures

Understanding scope and closures is crucial for writing robust, efficient JavaScript code. Scope defines where variables are accessible, while closures allow functions to remember and access variables from their creation context, even after that context has finished executing. Recognizing how variables are captured?by reference rather than by value?and managing their lifetime effectively can prevent common bugs and enable powerful programming patterns. Remember to leverage block scope with let and const, be cautious with variable references in closures, and utilize these concepts to write encapsulated, maintainable code.

With practice and careful attention, the mysteries of "you don?t know JS scope closures" will become clear, empowering you to write cleaner, more effective JavaScript applications.

You Don?t Know JS Scope Closures: A Deep Dive into JavaScript's Power and Pitfalls

Understanding you don?t know js scope closures is fundamental for writing robust, efficient, and bug-free JavaScript code. Despite being core concepts taught early in JavaScript education, scope and closures can often be sources of confusion for both newcomers and seasoned developers. Mastering these topics unlocks a deeper understanding of how JavaScript manages data and execution context, enabling you to write cleaner, more predictable code.

In this comprehensive guide, we'll examine JavaScript's scope system, unravel closures, explore practical examples, and discuss common pitfalls. Whether you're aiming to refine your skills or deepen your knowledge, this article provides the clarity and insights you need.

What Is Scope in JavaScript?

The Concept of Scope

In programming, scope determines the visibility and lifetime of variables. In JavaScript, scope defines where a variable is accessible within the code. JavaScript has two primary types:

- Global Scope: Variables declared outside any function or block are globally accessible throughout the code.
- Local Scope: Variables declared within a function or block are only accessible inside that region.

Function Scope

Before ES6, JavaScript only had function scope. Variables declared with `var` are scoped to the nearest enclosing function. For example:

```javascript

function example() {

var message = "Hello, World!";

console.log(message); // Accessible here

}

console.log(message); // Error: message is not defined

```

Block Scope (ES6+)

With ES6, `let` and `const` introduced block scope, meaning variables declared within `{}` are confined to that block:

```javascript

if (true) {

let count = 10;
```

```
}

console.log(count); // Error: count is not defined
```

Understanding these differences is crucial because they influence how closures capture variables.

Closures: The Hidden Power of JavaScript

Defining Closures

A closure is a function that remembers and has access to variables from its lexical scope even when invoked outside that scope. Essentially, closures "close over" variables from their surrounding context.

In simpler terms: When a function is defined inside another function, it retains access to the outer function's variables even after the outer function has finished executing.

Why Are Closures Important?

Closures enable powerful patterns such as data encapsulation, function factories, and callback functions. They allow functions to "remember" state without relying on global variables.

How Closures Work: An Illustrated Example

```javascript
function outer() {

let count = 0;

function inner() {

count++;

console.log(`Count is: ${count}`);

}

return inner;
```

```
}

const counter = outer();

counter(); // Count is: 1

counter(); // Count is: 2
```

In this example:

- `outer()` defines a variable `count` and an inner function `inner()`.
- When `outer()` is invoked, it returns `inner()`, which retains access to `count`.
- Even after `outer()` has finished executing, the `counter` function still "remembers" `count`.

This demonstrates a closure: `inner` has access to `count`, which persists across multiple invocations.

Common Use Cases for Closures

Data Encapsulation and Privacy

Closures allow you to simulate private variables:

```javascript
function createCounter() {

let count = 0;

return {

increment() {

count++;

return count;

},
```

```
getCount() {

return count;

}

};

}

const myCounter = createCounter();

console.log(myCounter.increment()); // 1

console.log(myCounter.getCount()); // 1
```

Here, `count` is not accessible directly from outside, only through the provided methods.

Function Factories

Generate customized functions:

```javascript
function makeGreeting(greeting) {

return function(name) {

console.log(`${greeting}, ${name}!`);

};

}

const sayHello = makeGreeting("Hello");

sayHello("Alice"); // Hello, Alice!
```

Maintaining State in Asynchronous Operations

Closures are invaluable in event handling, timers, or asynchronous code:

```javascript

for (var i = 1; i
setTimeout(function() {

console.log(i);

}, 1000);

}

// Outputs: 4, 4, 4 after 1 second, due to closure capturing `i`.

```

To fix this, you can use an IIFE or `let`:

```javascript

for (let i = 1; i
setTimeout(function() {

console.log(i);

}, 1000);

}

// Outputs: 1, 2, 3

```

Common Pitfalls and Misunderstandings

- Loop Variables and Closures

Using `var` in loops causes closures to capture the same variable, leading to unexpected behavior:

```javascript

for (var i = 1; i
setTimeout(function() {

console.log(i);

}, 100);

}

// Output: 4, 4, 4

```

Solution: Use `let` (block scope) or an IIFE:

```javascript

for (let i = 1; i
setTimeout(function() {

console.log(i);

}, 100);

}

```

- Memory Leaks

Closures keep references to variables, preventing garbage collection if not handled carefully. Be cautious with long-lived closures.

- Overusing Closures

While closures are powerful, overuse can make code harder to read and debug. Use them judiciously.

Advanced Topics: Scope Chains and Lexical Scope

Scope Chain

When a variable is accessed, JavaScript looks in the current scope; if not found, it moves outward through parent scopes until it reaches the global scope.

```javascript

function outer() {

let outerVar = 'outer';

function inner() {

let innerVar = 'inner';

console.log(outerVar); // Accessible via scope chain

}

inner();

}

```

Lexical Scope

JavaScript's scope is lexical, meaning the scope of variables is determined by their position in the source code, not the execution context.

Best Practices for Working with Scope and Closures

- Use `let` and `const` instead of `var` to avoid scope-related bugs.
- Be mindful of closure pitfalls in loops; prefer `let` for loop counters.
- Keep closures lightweight to avoid memory leaks.

- Use IIFEs when you need to create a new scope in ES5.
- Document closures clearly, especially when they manipulate shared state.
- Avoid unnecessary closures to improve performance and readability.

Summary

You don?t know js scope closures because these concepts often seem subtle yet are incredibly powerful. Grasping scope helps you understand where variables live, while closures give you tools to preserve state, create private data, and write modular code. Remember:

- Scope determines variable visibility.
- Closures are functions that remember their lexical environment.
- Proper understanding prevents bugs related to variable capture, memory leaks, and asynchronous behavior.
- Use modern JavaScript features (`let`, `const`, arrow functions) to write clearer, safer code involving closures.

By mastering scope and closures, you elevate your JavaScript skills, enabling you to write smarter, cleaner, and more maintainable code?fundamentals that are essential for any serious JavaScript developer.

Final Thoughts

JavaScript's scope and closures are not merely language features?they are foundational concepts that shape how your code behaves. Embrace their nuances, practice with real-world examples, and you'll find yourself writing more predictable and powerful JavaScript applications.

QuestionAnswer What is the difference between scope and closure in JavaScript? Scope defines the accessibility of variables in different parts of the code, while a closure is a function that retains access to its outer scope variables even after the outer function has finished executing. How do closures help in maintaining private variables? Closures allow functions to capture variables from their outer scope, enabling the creation of private variables that are not accessible from outside the closure, thus supporting encapsulation. Why does JavaScript have function scope instead of block scope? Traditional JavaScript uses function scope because variables declared with var are scoped to their enclosing function, not blocks. ES6 introduced let and const to provide block scope, but var remains function-scoped for backward compatibility. Can closures cause memory leaks in JavaScript? Yes, if closures retain references to large objects or DOM elements, they can prevent garbage collection, leading to memory leaks. Proper management and cleanup are essential when using closures extensively. How does variable hoisting affect closures and scope? Variable hoisting moves declarations to the top of their scope, which can lead to unexpected behavior within closures,

especially if variables are accessed before they are initialized. Understanding hoisting is crucial for predictable closures. What is a common use case for closures in JavaScript? Closures are commonly used to create private variables, function factories, callback functions, and to preserve state in asynchronous operations. How can I avoid common pitfalls with scope and closures? Use let and const for block scope, be cautious with variable declarations inside loops, and be aware of closure behavior to prevent unintended references. Employing IIFEs (Immediately Invoked Function Expressions) can also help manage scope. What are some examples of scope chain in JavaScript? The scope chain is the hierarchy of nested scopes that JavaScript searches through when resolving variable references. For example, a variable inside a function can access variables in its own scope, its outer scope, and the global scope. How do closures impact performance in JavaScript applications? While closures are powerful, excessive or unnecessary use can lead to increased memory consumption because variables captured in closures remain in memory. Optimizing closure usage can improve application performance.

Related keywords: JavaScript, scope, closures, understanding scope, closure functions, variable scope, lexical scope, function scope, scope chain, JavaScript closures

Read the full article online: you don t know js scope closures