@aswin_a

CLOSURES & SCOPE

Demystifying Closures and Scope in JavaScript

FEATURING :
Aswin A

@aswin_a

Understanding Scope

Global Scope: Variables defined outside any function, accessible anywhere in the code.

Local Scope: Variables defined within a function, accessible only within that function.

```
let globalVar = "I am global";
function localScope() {
  let localVar = "I am local";
  console.log(globalVar); // Accessible
  console.log(localVar); // Accessible
}
localScope();
console.log(localVar);//Error: localVar not defined
```

Function Scope: Each function creates a new scope. Variables defined inside a function are not accessible outside of it.

```
function testFunction() {
  let functionVar = "Inside function";
  console.log(functionVar); // Accessible
}
testFunction();
console.log(functionVar);//Error:functionVar not defined
```

FEATURING:
Aswin A

Block Scope with let and const :

let and const are used to declare variables that are block-scoped. A block is defined by a pair of curly braces {}. Variables declared with let or const are only accessible within the block in which they are defined, including any nested blocks.

Using let :

```
if (true) {
  let x = 10;
  console.log(x); // Output: 10
}
console.log(x); // ReferenceError: x is not defined
```

Using const :

```
if (true) {
  const y = 20;
  console.log(y); // Output: 20
}
console.log(y); // ReferenceError: y is not defined
```

FEATURING:
Aswin A

@aswin_a

Understanding Closures

Definition : A closure is a function that retains access to its lexical scope even when invoked outside that scope. It is useful for creating private variables and functions.

```
function outerFunction() {
  let outerVar = "I am outer";
  function innerFunction() {
    console.log(outerVar); // Accessible
  }
  return innerFunction;
}
let closureFunction = outerFunction();
closureFunction(); // "I am outer"
```

Inner Function : Functions defined inside another function have access to the outer function's scope.

```
function counter() {
  let count = 0;
  return function() {
    count++;
    return count;
  }}
let increment = counter();
console.log(increment()); // 1
console.log(increment()); // 2
```

FEATURING:
Aswin A

Private Variables: Closures can create private variables that are not accessible from outside the function.

```
function createPerson(name) {
  let age = 0;

  return {
    getName: function() {
      return name;
    },
    getAge: function() {
      return age;
    },
    birthday: function() {
      age++;
    }
  };
}

let person = createPerson("John");
console.log(person.getName()); // John
console.log(person.getAge()); // 0
person.birthday();
console.log(person.getAge()); // 1
```

FEATURING:
Aswin A

Event Handlers : Commonly used in event handlers to retain access to the outer scope.

```
function setupClickHandler() {
  let count = 0;
  document.getElementById("myButton").addEventListener("click", function()
  {
    count++;
    console.log(`Button clicked ${count} times`);
  });
}
setupClickHandler();
```

Memory Implications : Closures can increase memory usage as they keep references to outer scope variables.Avoid excessive use of closures in performance-critical applications.

```
function createHeavyClosure() {
  let largeArray = new Array(1000000).fill("data");
  return function() {
    console.log(largeArray.length);
  };}
let heavyClosure = createHeavyClosure();
heavyClosure(); // 1000000
```

FEATURING:
Aswin A

Did you find it Useful ?

Leave a Comment !

Follow For More ...

@aswin_a