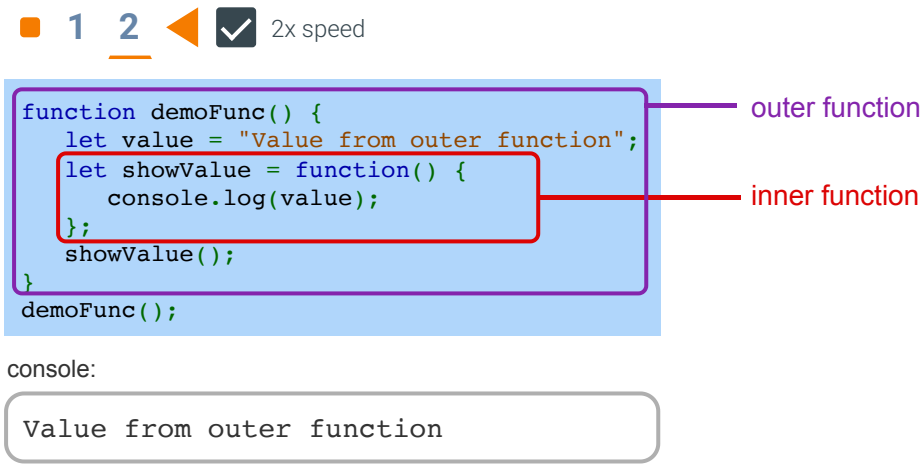
8.5 Inner functions, outer functions, and function scope

Inner and outer functions

An ***inner function*** (***nested function***) is a function declared inside another function. An ***outer function*** is a function containing an inner function. An inner function can access variables declared in the outer function.

PARTICIPATION ACTIVITY 8.5.1: Inner and outer functions.



Despite being declared outside of the showValue() function, value can still be accessed from within showValue().

Captions

1. showValue() is an inner function declared within the demoFunc() function. demoFunc() is the outer function.
2. Despite being declared outside of the showValue() function, value can still be accessed from within showValue().

Feedback?

PARTICIPATION ACTIVITY 8.5.2: Inner and outer functions.

```
function logzyBooks() {
   console.log("zyBooks");
}

function logSum(x, y, z) {
   const sum = x + y + z;
   let actualLogger = function() {
      console.log(sum);
   }
   actualLogger();
}
```

If unable to drag and drop, refresh the page.

---

actualLogger()	**Inner function** `actualLogger()` is declared inside the `logSum()` function and is therefore an inner function.	**Correct**
logSum()	**Outer function** `logSum()` contains the nested `actualLogger()` function, so `logSum()` is an outer function.	**Correct**
logzyBooks()	**Neither an inner nor outer function** The `logzyBooks()` function does not contain any nested functions and is not nested inside another function. `logzyBooks()` is therefore not an inner or outer function.	**Correct**

Reset

Feedback?

A function declaration or function expression can be used to declare an inner function

The examples above declare an inner function using a function expression. Inner functions can also be declared using function declarations, as shown in the example below.

Ex: Array filtering

Inner functions are commonly used for array filtering. An Array object's ***filter()*** method takes a filter function as an argument, calls the filter function for each array element, and returns a new array consisting only of elements for which the filter function returns true.

PARTICIPATION ACTIVITY 8.5.3: Filtering an array of grades to get only passing grades.

1 2 3 2x speed

```
function getPassingGrades(grades) {
   function isPassing(number) {
      return number >= 73;
   }

   return grades.filter(isPassing);
}

const grades = getPassingGrades([73.1, 86.4, 62.1, 59.6, 88.8, 99.9]);
console.log("Passing grades: " + grades);
```

Console:

```
Passing grades: 73.1,86.4,88.8,99.9
```

The filter function is called for each element. The returned array consists only of the grades >= 73.

Captions

1. Outer function getPassingGrades() declares isPassing() as an inner function. isPassing() returns true only if the number passed as an argument is >= 73.
2. getPassingGrades() is called with an array of grades. The array's filter() method is called with the inner function passed as an argument.
3. The filter function is called for each element. The returned array consists only of the grades >= 73.

Feedback?

PARTICIPATION ACTIVITY 8.5.4: Array filtering using inner functions.

Consider the following code.

```
const strings = ["one", "two words", "three", "four five"];

function getSingleWords(stringArray) {
   const noSpace = function(element) {
      return element.indexOf(" ") === -1;
   };

   return stringArray.filter(noSpace);
}

function getStartingWith(stringArray, startString) {
   function startsWith(string) {
      return string.indexOf(startString) === 0;
   }

   return stringArray.filter(startsWith);
}
```

1) What does `getSingleWords(strings);` return?

- `["one", "two words", "three", "four five"]`
- ◉ `["one", "three"]`
- `"one"`

Correct

The filter function returns true when `indexOf()` does not find a space in the array element. Strings `"one"` and `"three"` are the 2 strings without a space.

2) Which inner function uses a variable from the outer function?

- `noSpace()`
- ◉ `startsWith()`
- neither

Correct

`startsWith()` uses `startString`, which is an argument passed to the `getStartingWith()` outer function.

3) What does `getStartingWith(strings, "t");` return?

- ◉ `["two words", "three"]`
- `["three"]`
- `[]`

Correct

The filter function returns true when `indexOf()` finds the `startString` at the beginning of the array element. The strings `"two words"` and `"three"` begin with "t" and are included in the filtered array.

Feedback?

Scope objects

To store a collection of variables for a particular scope, JavaScript implementations commonly use a ***scope object***: An object that stores a collection of variable names and

corresponding values. Ex: Whenever a function with local variables is executed, the JavaScript runtime creates a scope object that stores the function's local variables.

Scope objects are behind-the-scenes objects used to implement the JavaScript runtime and are not accessible in JavaScript code.

PARTICIPATION ACTIVITY 8.5.5: Scope objects.

1 2 3 4 5 6 2x speed

```
function logXYZ() {
   let x = Math.random();
   if (x < 0.5) {
      let y = x * 100;
      console.log(y);
   }

   let z = x * 1000;
   console.log(z);
}
```

Call to logXYZ()

Function scope object:

x	0.25
z	250

if block scope object:

y	25

console:

```
25
250
```

The function scope object is used to store and lookup values for x and z.

Captions

1. When logXYZ() is called, a scope object for the function is created. Variables x and z are scoped to the entire function and are stored in the scope object.
2. The scope object includes the variable values. x is assigned with the random number 0.25 on the first line, and z is initially undefined.
3. A new block scope object is created when execution enters the if block. Variable y is declared with let and is scoped to the if block.
4. The block scope object is used by the runtime to lookup y's value for the console.log() call.
5. When execution leaves the if block, the block scope object with y's value is removed. The variable y is out of scope and no longer available.
6. The function scope object is used to store and lookup values for x and z.

Feedback?

Avoid mixing 'var' and 'let' in practice

Examples in this material, like the one above, may mix the use of ***`var`*** *and* ***`let`*** *to illustrate technical concepts. While* ***`var`*** *and* ***`let`*** *can be used together, good practice is to avoid mixed usage and instead use only one of the two.*

PARTICIPATION ACTIVITY 8.5.6: Scope objects.

1) Function `logXYZ()` always creates _____.

- a single scope object to hold all of the function's variables
- two scope objects: one for `x` and `z` and the other for `y`
- one scope object for variables `x` and `z`, but also creates a second scope object for `y` if execution enters the if block

Correct

The scope object for `x` and `z` is always created. But the scope object for `y` is only created if execution enters the if block.

2) Calling function `logXYZ()` six times means that at least six distinct scope objects are created by the JavaScript runtime.

- True
- False

Correct

A new scope object is created for each function call. So six calls implies at least six distinct scope objects.

3) Consider the altered version of function `logXYZ()` below, which logs `x`'s value in the if block.

```
function logXYZ()
{
   let x =
Math.random();
   if (x < 0.5) {
      let y = x *
100;

console.log(x);

console.log(y);
   }

   let z = x *
1000;

console.log(z);
}
```

For the statement `console.log(x);`, how many scope objects will be checked to find the value for `x`?

- 1
- 2 (selected)

Correct

Looking up `x`'s value starts with the most recent block scope, which doesn't contain `x`. The `logXYZ()` function's scope object, which contains `x`, is checked next. So two scope objects are checked.

Feedback?

Scope chain

A ***scope chain*** is a linked list of scope objects used by the JavaScript runtime to store and lookup variable values when executing code. When a variable is needed, a search begins at the scope object at the beginning of the scope chain. If the variable is found, the corresponding value is used. Otherwise, the next object in the scope chain is searched. If the search reaches a null object at the end of the scope chain, the variable is not found and a ReferenceError is thrown.

The scope chain always contains the global scope object. Additional scope objects are prepended to the list as code executes. Ex: Calling a function prepends a new scope object for that function's local variables. A block scope object is prepended when execution enters a nested block.

The animation below illustrates how the scope chain works by using the same variable name in the function's block scope and the nested scope.

PARTICIPATION ACTIVITY 8.5.7: Scope chain.

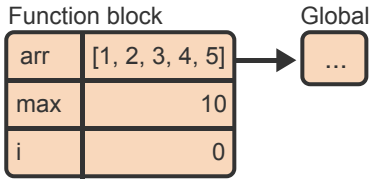
1 2 3 4 5 6 7 8 2x speed

```
function shuffle(arr) {
   const max = arr.length * 2;
   let i = 0;
   while (i < max) {
      // max redeclared in new block
      const max = arr.length;

      // Create 2 random indices
      const i1 = Math.floor(Math.random() * max);
      const i2 = Math.floor(Math.random() * max);

      // Swap two array elements
      const temp = arr[i1];
      arr[i1] = arr[i2];
      arr[i2] = temp;

      ++i;
   }
   return arr;
}
shuffle([1, 2, 3, 4, 5]);
```

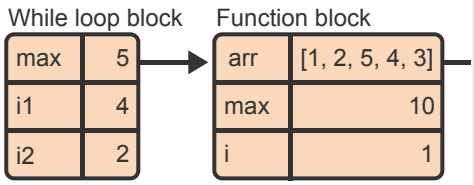
Execution context scope chain...

... just before the call to shuffle:

... just before entering the loop for the



... at the end of the first loop iteration:



When using the arr and i variables, the scope chain is searched from the beginning. Both vari scoped to the function block, and are found in the second scope chain object.

Captions

1. Just before calling shuffle(), code is executing in the global scope. The scope chain has one object containing global variables.
2. Entering the function's body prepends a scope object to the chain. Function-block-scoped variables max and i are included, along with parameter arr.
3. Entering the while loop's block for an iteration prepends a new block scope object.
4. const gives a variable block scope, so the redeclaration puts max into the block scope object with a value of 5.
5. When retrieving the value of max, the scope chain is searched. Since max is found in the first object, no additional objects are searched.
6. i1 and i2 are also in the object at the front of the scope chain.
7. When using the arr and i variables, the scope chain is searched from the beginning. Both variables are scoped to the function block, and are found in the second scope chain object.
8. The scope object for the last loop iteration is removed after exiting the loop. The scope chain again has two objects just before returning.

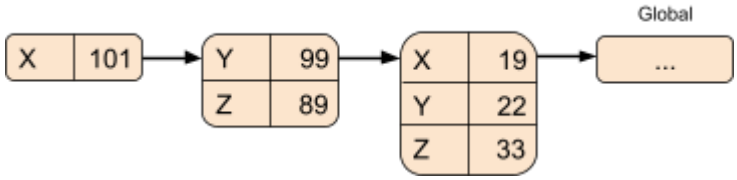
Feedback?

Scope objects reference the next/outer scope object

Arrows in the animation above illustrate the links between scope objects. Each scope object has a link to the next scope object in the list. The link is commonly referred to as the "outer" link.

PARTICIPATION ACTIVITY 8.5.8: Scope chain.

Suppose the following scope chain is being used when executing code.



1) The statement `console.log(X)` will _____.

- log 101 to the console
- log 19 to the console
- throw a ReferenceError

Correct

The lookup for `X`'s value begins at the front of the scope chain. `X` exists in the first scope object and has a value of 101.

2) The statement `Y = 42;` will _____.

- add `Y` to the object at the front of the scope chain, with a value of 42
- replace `Y`'s value of 99 with 42 in the second scope object
- replace `Y`'s value of 22 with 42 in the third scope object

Correct

Since `Y` is not redeclared, `Y` is searched for starting at the beginning of the scope chain. `Y` is found in the second scope object, and the value of 99 is replaced with 42.

3) The scope chain with more than 2 scope objects implies that at least 1 function call was made.

- True
- False

Correct

No function calls are made in the following code, which could execute in the global scope and create a scope chain with multiple scope objects:

```
{
   let X = 19;
   let Y = 22;
   let Z = 33;
   if (Z > Y) {
      const Y = 99;
      const Z = 89;
      if (Y + Z > 100) {
         let X = Y + 2;
         console.log(X);
      }
   }
}
```

Feedback?

Exploring further:

- Nested functions and closures (MDN)

How was this section? Provide section feedback