

# If we execute this Javascript, what will the browser's console show?

```
var text = 'outside';  
  
function logIt(){  
    console.log(text);  
    var text = 'inside';  
};  
  
logIt();
```

JavaScript

## Gotchas

It's not "outside".

It's not "inside".

The script won't throw an error!

## Solution

The console will log undefined.

To understand this, we need to explain a few things about Javascript.

**Function-level scope.** Functions create new scopes in Javascript:

```
function setVar(){  
    // inside this function we have a new scope  
    // so this variable, declared in this function's scope, won't be available outside the function  
    var varInFunction = 'inside a function';  
}  
  
setVar();  
  
console.log(varInFunction); // throws 'ReferenceError: varInFunction is not defined'
```

JavaScript

Blocks like `if` statements and `for` loops do *not* create a new scope (this is also true of Python and recent versions of Ruby, but untrue of Java and C):

```
if (true) {  
    // this if statement doesn't create a new scope  
    // so varInIf is available in the global scope  
    var varInIf = 'inside an if statement';  
}  
  
console.log(varInIf); // logs 'inside an if statement'
```

JavaScript

**Declaration vs. assignment.** A variable *declaration* simply tells the interpreter that a variable exists. By default it initializes the variable to `undefined`:

```
var unicorn;  
  
console.log(unicorn); // logs undefined (NOT a ReferenceError)
```

JavaScript

A variable *assignment* assigns a value to the variable:

```
unicorn = 'Sparkles McGiggleton';
```

JavaScript

We can both *declare* and *assign* in the same line:

```
var centaur = 'Horsey McPersonhead';
```

JavaScript

**Hoisting.** In Javascript, variable *declarations* are "hoisted" to the top of the current scope. Variable *assignments*, however, are not.

So returning to the original problem:

```
var text = 'outside';  
  
function logIt(){  
    console.log(text);  
    var text = 'inside';  
};  
  
logIt();
```

JavaScript

The declaration (but not the assignment) of `text` gets hoisted to the top of `logIt()`. So our code gets interpreted as though it were:

```
var text = 'outside';

function logIt(){
  var text;
  console.log(text);
  text = 'inside';
};

logIt();
```

So we have a new variable `text` inside of `logIt()` that is initialized to `undefined`, which is what it holds when we hit our `log` statement.

## What We Learned

Remember: when you declare a variable in JavaScript (using "var"), that variable declaration is "hoisted" to the top of the current scope—meaning the top of the current function or the top of the script if the variable isn't in a function.

Hoisting can cause unexpected behavior, so a good way to keep things clear is to **always declare your variables at the top of the scope**.

## Ready for more?

**Check out our full course →**

---

Want more coding interview help?

Check out **interviewcake.com** for more advice, guides, and practice questions.