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

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

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:

```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'
```

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):

```JavaScript
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'
```

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

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

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

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

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

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

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:

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

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

```JavaScript
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**.
