COMD 3663 Dynamic Web 1

let const var

Declaring and Defining Variables: `var`

The `var` statement declares a variable, OPTIONALLY initializing it to a value. You don't have to immediately assign a value to a `var` (or `let`) variable. Uninitialized declaration of `var`:

```
var str
```

Initialized variable using `var`:

```
var str = 'Hello, world!'
```

Syntax:

```
var name1 = value1
```

var Declarations

var *declarations*, wherever they occur, are processed before any code is executed. This is called hoisting.

Variable Scope

1. A variable is either in the local scope, within the body of a function, or in the global scope, outside of a function and available everywhere in the application.
2. Unless you need a variable to be available everywhere in an application, it is always *preferrable* to **declare** and initialize it in the local scope.

Assigning Values to Undeclared Variables

Assigning values to undeclared variables *implicitly* makes them global variables. Because it makes them global variables, and because developers might not realize that they exist as such (i.e., they inadvertently ommitted `var`), undeclared variables can result in global variable leaks. This can cause unpredicatable code behavior, and the undeclared variables can be very hard to track down.

var hoisting

Now I'll give an example of hoisting in the JS Playground I have created.

Explicit Strict Mode

With ES5, which was introduced in 2009, the concept of strict mode was introduced.

"strict mode" was introduced in response to JavaScript's *lenience* and and acceptance of sloppy or incorrect code, such as *undeclared* global variables. It was not recognized by all browsers, however, so cross-browser compatibility *testing* was necessary. On the other hand, normal code and code in strict mode could co-exist, so that was not a problem. These days, even though Javascript is being compiled down to ES5 for cross-browser compatibility, we are already entering ECMAScript 2020. Pretty much all developers have embraced ECMAScript 2015 (ES6) and beyond.

`let` and `const`

In response to `var`'s "sloppiness", ES6 introduced `let` and `const`.

let

`let` is really neat. The `let` statement declares a block scope local variable, optionally (immediately) initializing it to a value.

Syntax:

```
let variable1 = value1;
```
`variable1`: the name of the variable that is being declared. Each character of the name must be a *legal* JavaScript identifier.
`value1`: with the *declaration* of the variable, you may optionally *immediately* specify its initial value to any legal JavaScript expression.

Legal JavaScript Identifiers in Naming Variables

According to Mathias Byens, member of the ECMAScript technical committee TC39, which oversees the evolution of the Ecma262 standard (in other words, JavaScript), states on his blog in a post entitled Valid JavaScript variable names in ES2015:

> **ES2015 updates the grammar for identifiers.**

He gets into great detail about what identifiers (characters) can be used in naming variables, most of which you can ignore for this class. BUT he does provide a very helpful tool in this post (I have included the *link* to it in the **Related Resources** section at the end of this lecture as well). It is a variable name validator! There are a whole bunch of other really cool validator tools included there which we probably will revisit when appropriate.

let vs var: recap

1. `let` allows you to declare variables *limited* to the local scope of a block statement (or expression on which it is used, and we'll talk about that when we get to expressions!), but the `var` **keyword** declares variables *globally* or *locally* to a whole function, *ignoring* block scope.
2. Unlike `var`, which is *immediately* **initialized** to a value, `let` is *initialized* to a value **only when** a **parser** evaluates it.
3. Unlike `var`, `let` does not create properties of the Window object when declared *globally*. We'll be getting into the `Window` object later in the course. Two such properties are `localStorage` and `sessionStorage`, for example.
4. `let` cannot be re-declared. On the other hand, `var` can.

Why the keyword name let was chosen

The MDN docs on `let` link to an in-depth explanation of the reason the keyword name `let` was chosen. It is on stackoverflow, and states the following:

> **Let is a mathematical statement that was adopted by early programming languages like Scheme and Basic. Variables are considered low level entities not suitable for higher levels of abstraction, thus the desire of many language designers to introduce similar but more powerful concepts like in Clojure, F#, Scala, where let might mean a value, or a variable that can be assigned, but not changed, which in turn lets the compiler catch more programming errors and optimize code better.**
>
> **JavaScript has had var from the beginning, so they just needed another keyword, and just borrowed from dozens of other languages that use let already as a traditional keyword as close to var as possible, although in JavaScript let creates block scope local variable instead.**

const

const was introduced in ES6, and is very different from var. const works like what its keyword name sounds like. The value assigned to it is ***read only***. It cannot be re-assigned a value as with let and var. let and var allow you to ***re-assign* values** to them. const does not.

const Syntax

Like let, const is block scoped. It also can't be changed through re-assignment (let can), and can't be re-declared.

Syntax:

```
const name1 = value1;
```

name1: the name of the constant, which can be any legal identifier.
value1: the value of the constant, which can be any legal expression, including a function expression.

const in Detail

1. `const` declaration creates a constant whose scope can be `global` or `local blocked`, depending on where it is placed.
2. Global `consts` do NOT become properties of the `Window` object, as with `var` variables.
3. An *initializer* is required when declaring a `const`. In other words, you can't do `const name; name="Maria";`. You have to immediately assign a value to the const `name`. So `const name = "Maria";` instead.
4. `const` declaration is read only. It creates a read-only reference to the value. This does NOT mean that the value which is assigned to it is immutable, unchangeable. It just means that the `variable identifier` *cannot be* `re-assigned`. The `value` can change, just the `structure` of the value cannot.

Related Resources

- TC39: Specifying JavaScript
- Strict Mode: MDN
- Valid JavaScript variable names in ES2015: mathias bynens
- var: MDN
- let: MDN
- const: MDN
- Why was the name 'let' chosen for block-scoped variable declarations in JavaScript?: stackoverflow
- Redeclaring a javascript variable: stackoverflow