

SCOPE, HOISTING && CLOSURES EXPLAINED

Prof. Andrew Sheehan

Boston University/MET Computer Science Dept.

# **О**Ш H C C H

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state

A closure gives you access to an outer function's scope from its inner function(s)

Closures are created every time a function is created

```
function init() {
 var name = 'Mozilla';
  function displayName() // inner function (closure)
      console.log(name);
  displayName();
```

init();

#### LEXICAL SCOPE

How a parser resolves variable names when functions are nested.

Lexical scoping uses the location where a variable is declared

Scope means where something is accessible from its declaration point

#### var

Can be either function scope or global.

#### let,const

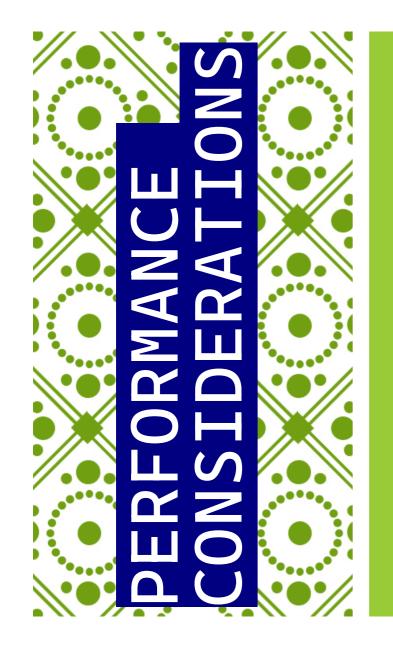
Not function scope, but block scope.

```
Local scope (within function body)
```

Enclosing scope (block,
function or module)

Global scope

# 刀 0



Do not create functions within other functions if closures are not needed.

It will negatively affect script performance both in terms of processing speed and memory consumption.

Refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

#### All undeclared variables will be placed into global scope

```
function hoist() {
 a = 20;
 var b = 100;
hoist();
console.log(a);
Accessible as a global variable outside hoist() function
Output: 20
console.log(b);
/*
Since it was declared, it is confined to the hoist() function scope.
We can't print it out outside the confines of the hoist() function.
Output: ReferenceError: b is not defined
*/
```

## < DZ

## Hoisting allows functions to be safely used in code before they are declared

```
sayHi("Andrew");
function sayHi(username = "Guest") {
  console.log(`Hi, ${username}`);
}
// The result of the code above is: "Hi, Andrew"
```

#### 'STRICT MODE'

By enabling strict mode, we opt into a restricted variant of JavaScript that will not tolerate the usage of variables before they are declared

```
'use strict';
console.log(hoist); // Output: ReferenceError: hoist is not defined
hoist = 'Hoisted';
```

## Hoisting allows functions to be safely used in code before they are declared

```
console.log(randomName); // undefined
var randomName; // undefined
randomName = "Frank";
Console.log(randonName); // Frank;
```

