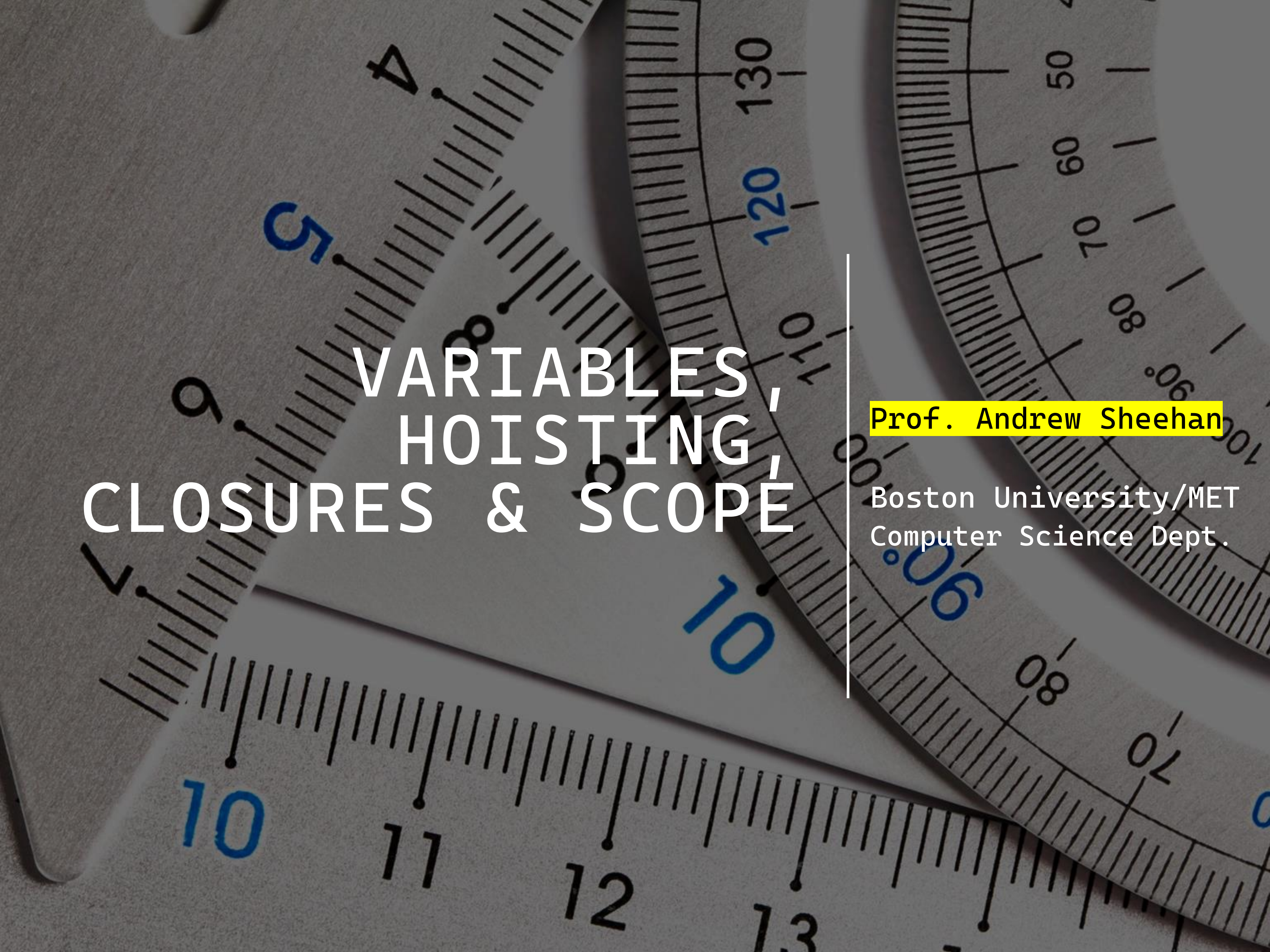


VARIABLES, HOISTING, CLOSURES & SCOPE

Prof. Andrew Sheehan

Boston University/MET
Computer Science Dept.

WHAT IS A VARIABLE?

Purpose:

Stores a value in a computer program

Goes by ‘identifier’ as well

`‘var’ means ‘can change’`

VAR == CHANGEABLE

RULES

1. Must start with a letter (a ~ z A ~ Z), underscore(_), or dollar($) sign.

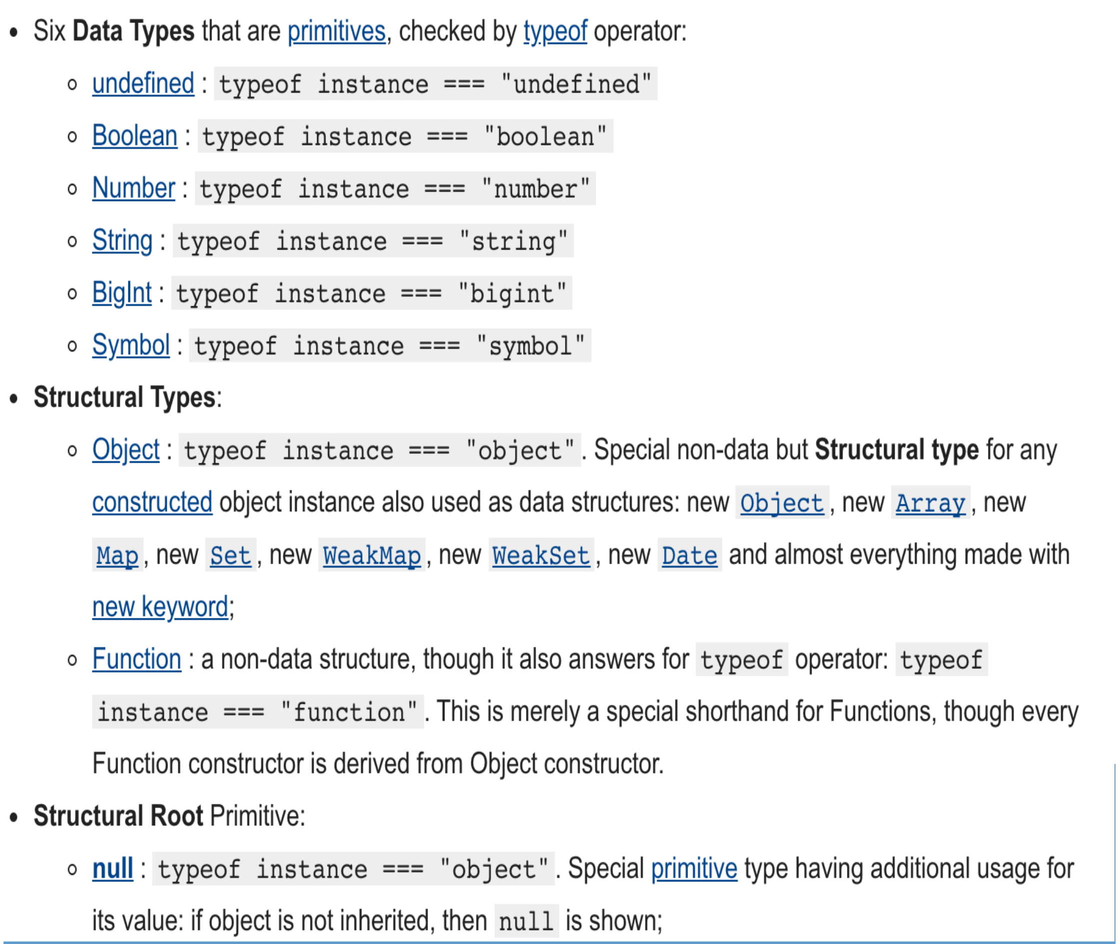
2. You cannot start with a number (0-9) for the beginning of your variable name. But, after the 1st legal character, you can.

3. Variable names are case sensitive.

A DYNAMIC LANGUAGE

NOT STRONGLY TYPED

DATA TYPES



- Six **Data Types** that are primitives, checked by typeof operator:
  - undefined : `typeof instance === "undefined"`
  - Boolean : `typeof instance === "boolean"`
  - Number : `typeof instance === "number"`
  - String : `typeof instance === "string"`
  - BigInt : `typeof instance === "bigint"`
  - Symbol : `typeof instance === "symbol"`
- **Structural Types**:
  - Object : `typeof instance === "object"`. Special non-data but **Structural type** for any constructed object instance also used as data structures: new `Object`, new `Array`, new `Map`, new `Set`, new `WeakMap`, new `WeakSet`, new `Date` and almost everything made with new keyword;
  - Function : a non-data structure, though it also answers for `typeof` operator: `typeof instance === "function"`. This is merely a special shorthand for Functions, though every Function constructor is derived from Object constructor.
- **Structural Root** Primitive:
  - **null** : `typeof instance === "object"`. Special primitive type having additional usage for its value: if object is not inherited, then `null` is shown;

CONST ~ VAR ~ LET

const 🟩

Assigned?
Can’t change
(best)

Block scope
(better)

let 🟩

⚠ var

Avoid using
(not great)

The worst thing you can do with variables is not using var, let or const at all.

ALWAYS USE VAR, LET OR CONST

Difference Between `var`, `let`, and `const`

JavaScript has three different keywords to declare a variable, which adds an extra layer of intricacy to the language. The differences between the three are based on scope, hoisting, and reassignment.

Keyword	Scope	Hoisting	Can Be Reassigned	Can Be Redeclared
`var`	Function scope	Yes	Yes	Yes
`let`	Block scope	No	Yes	No
`const`	Block scope	No	No	No

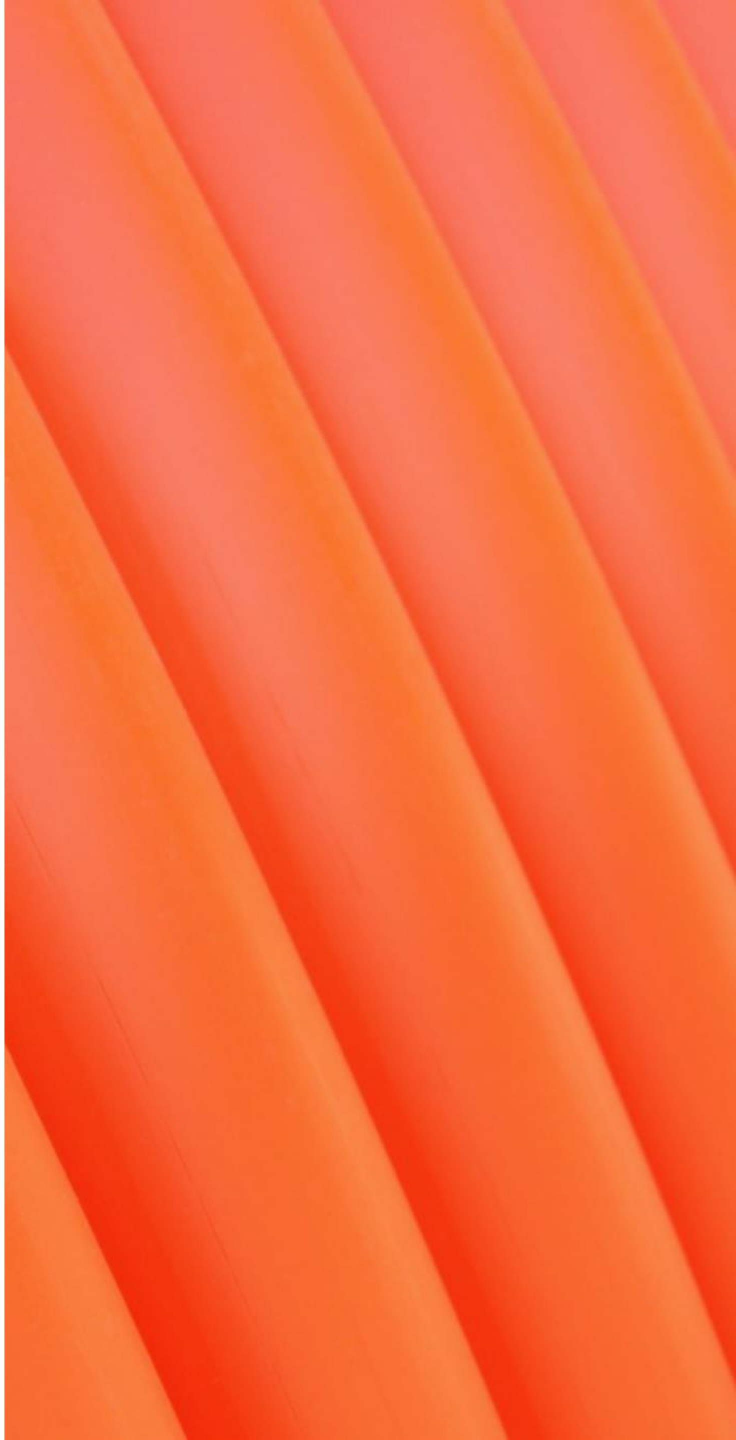
You may be wondering which of the three you should use in your own programs. A commonly accepted practice is to use `const` as much as possible, and `let` in the case of loops and reassignment. Generally, `var` can be avoided outside of working on legacy code.

TYPES OF SCOPE

Mentioned this two types:local and global

1. Inside a block or function (local)

2. Outside a block (global)



FUNCTION PARAMETERS

Do not use var, let or const within your parameter definition

```
function foo(let a=1,var f=8.01) {
  /* do something in function body… */
}
```

if we meet the let and var, we need to remove them

GLOBAL SCOPE

Outside any class, function or object

GLOBAL SCOPE

```
function check (A = 0) {
    return A > MAX;
}
```

When you use a variable before you assign something to it, the default value is

```
let name
Undefined
```

undefined

NULL

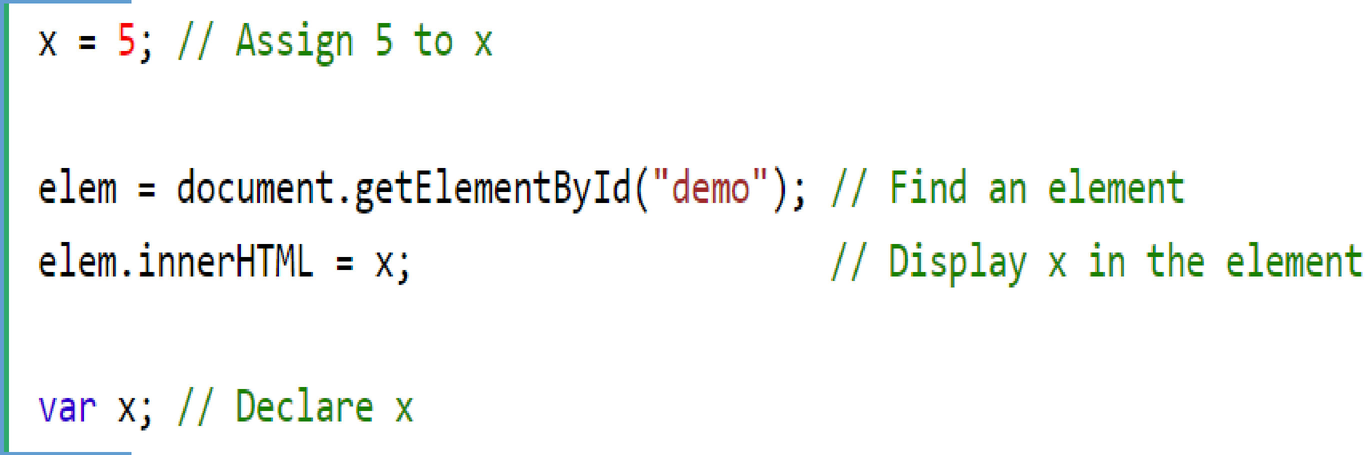
A place where unicorns live *and null, as well*

HOISTING

All undeclared variables are global variables

```
//Notice lack of var, let or const
message = `happy`;
```

HOIST EXAMPLE

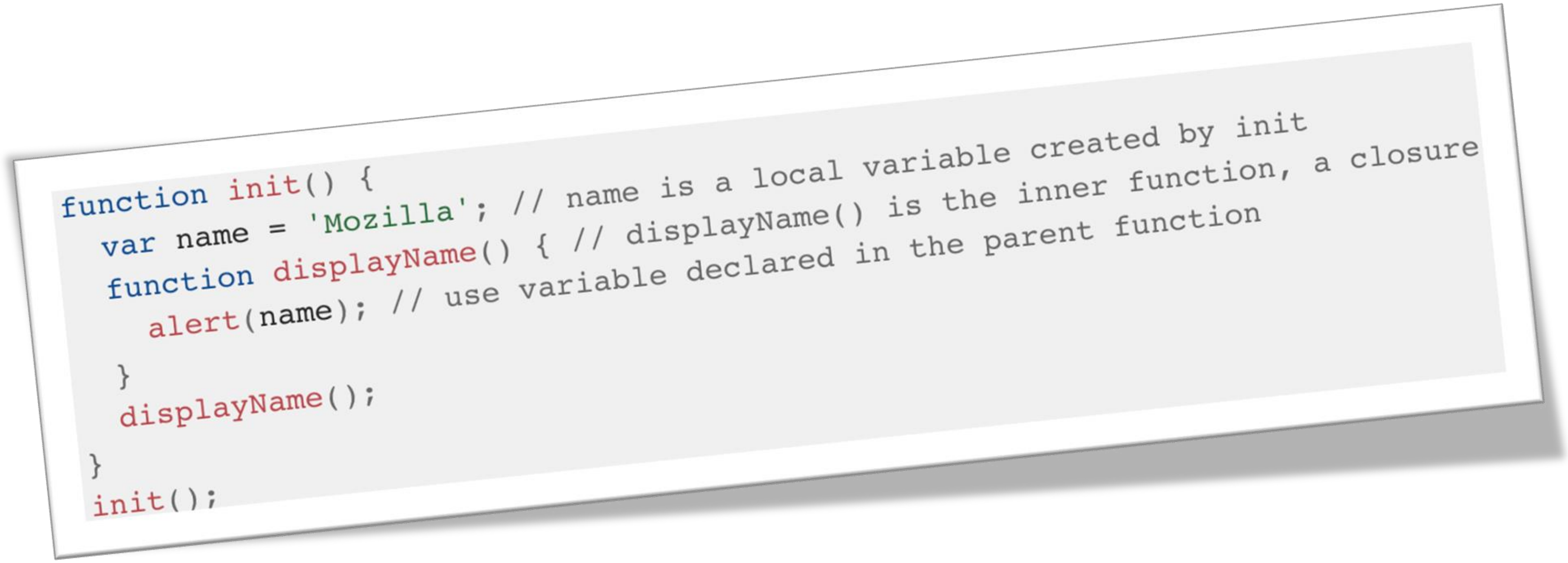


```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

CLOSURE

function
with
function

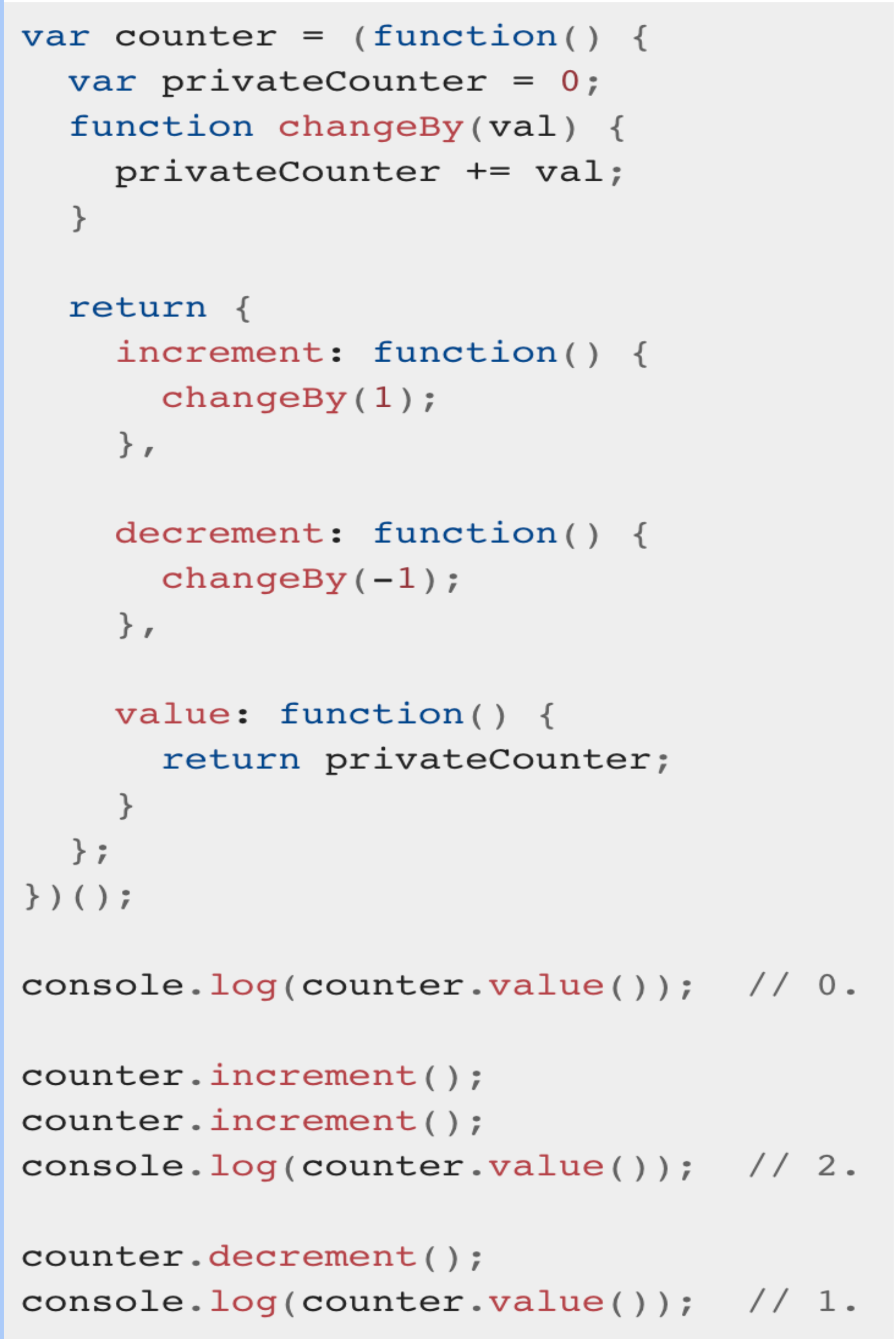
A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.



```
function init() {
  var name = 'Mozilla'; // name is a local variable created by init
  function displayName() { // displayName() is the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
init();
```

CLOSURE EXAMPLE

MORE CLOSURE EXAMPLES



```
var counter = (function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }

  return {
    increment: function() {
      changeBy(1);
    },

    decrement: function() {
      changeBy(-1);
    },

    value: function() {
      return privateCounter;
    }
  };
})();

console.log(counter.value());  // 0.

counter.increment();
counter.increment();
console.log(counter.value());  // 2.

counter.decrement();
console.log(counter.value());  // 1.
```