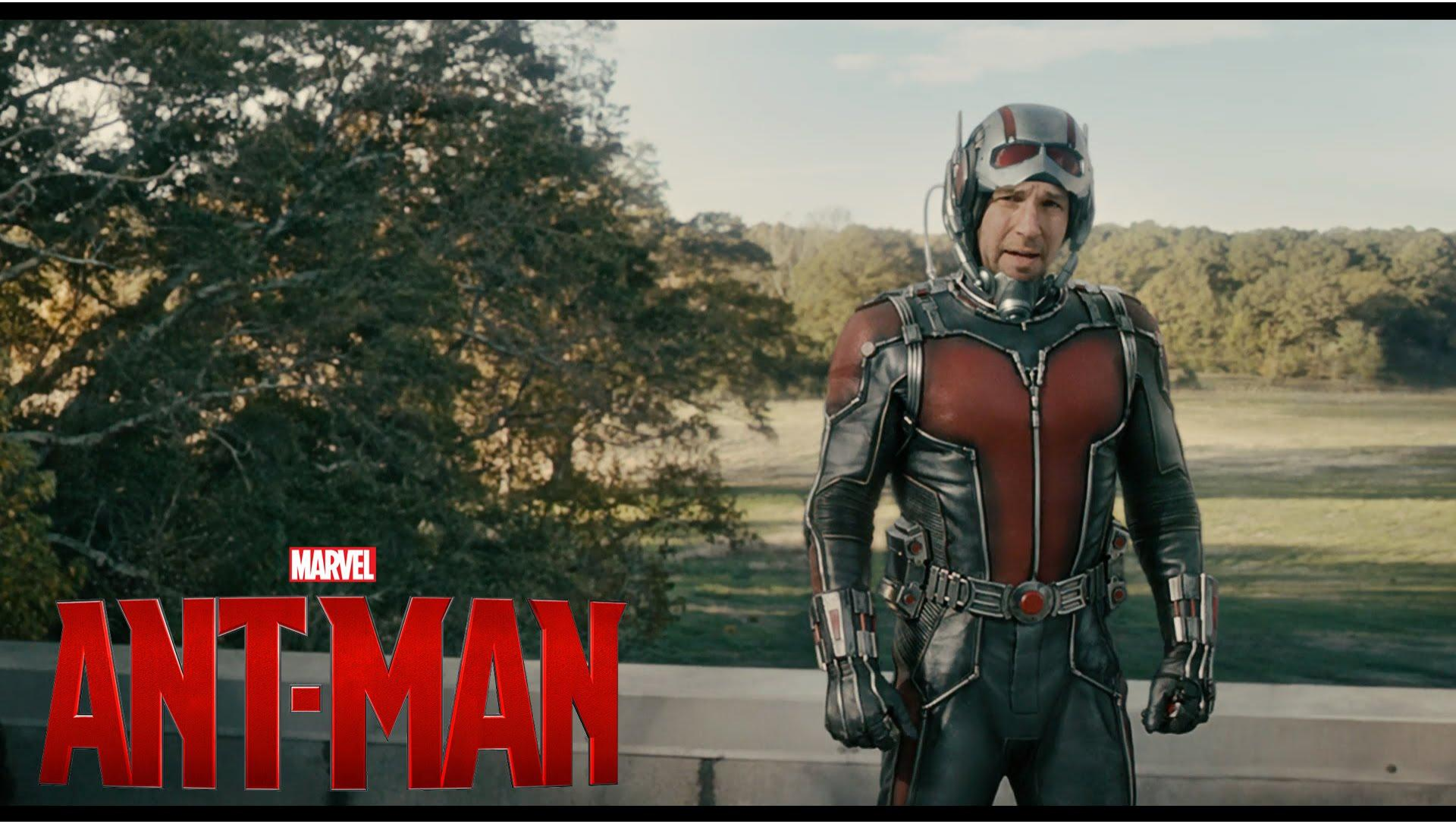
Lecture III


SPIDER-MAN


MARVEL
ANT-MAN

Compiler-man

Example

```
var a = 2;
var b = 2;
console.log(a + b);
```

Recap of the previous lecture

Scope in ES3 World

The Scope of a Variable

The scope of a variable are the locations where it is accessible.

For example:

```
function foo() {
   var x;
}
```

Scope - is a logical boundaries in which a variable (or expression) has its meaning. For example, a global variable, a local variable, etc, which generally reflects a logical range of a variable lifetime.

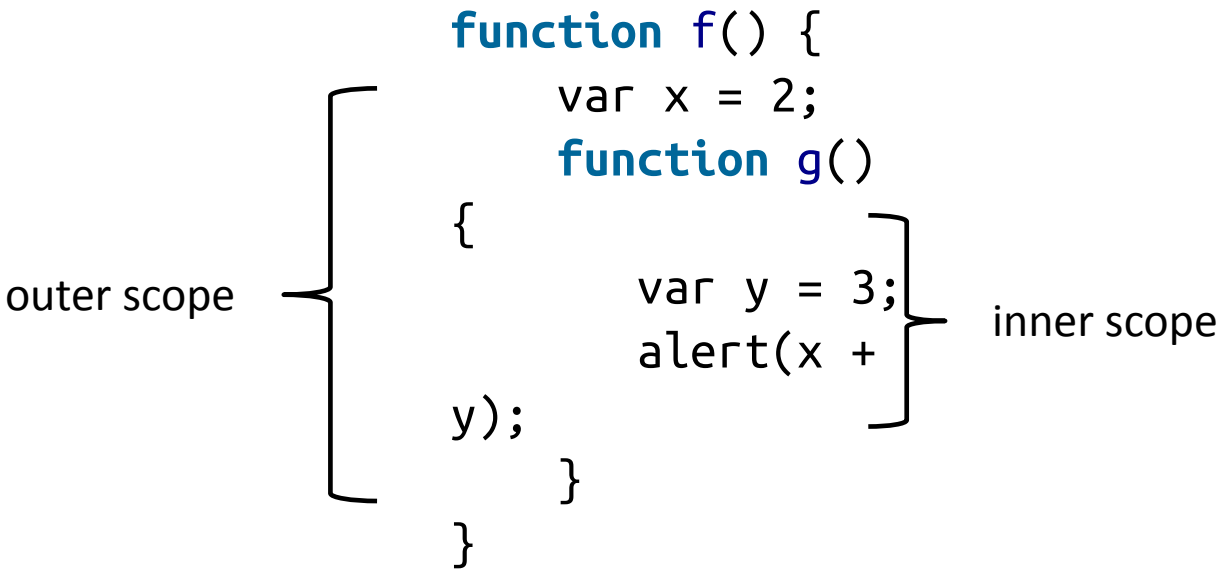
Scope - is a logical boundaries in which a variable (or expression) has its meaning.

For example, a global variable, a local variable, etc, which generally reflects a logical range of a variable lifetime.

Scope can be nested

```
function f() {
    var x = 2;
    function g()
{
        var y = 3;
        alert(x +
y);
    }
}
```

Nested Scope



Shadowing

```
var scope = "global ";        // A global variable
function outer() {
        var scope = “outer”;      // A outer variable
        function inner() {
                var scope = “inner";
                document.write(scope);      // Prints "inner"
        }
        inner();
}
outer();
```

Shadowing

```
var scope = "global ";        // A global variable
function outer() {
        var scope = “outer”;      // A outer variable
        function inner() {
                document.write(scope);      // Prints "outer"
        }
        inner();
}
outer();
```

Shadowing

```
var scope = "global ";        // A global variable
function outer() {
        function inner() {
                document.write(scope);      // Prints "global"
        }
        inner();
}
outer();
```

```
function X() {
        var a = 3, b = 5;
        function foo () {
                var b = 7, c = 11;
                a += b + c;
        };
        foo();
        alert("a = " + a + "; b = " + b);
}
X();
```

```
var x = function(){
   alert(x.toString());
}
x();
```

Hoisting

JavaScript *hoists* all variable declarations, it moves them to the beginning of their direct scopes.

Hoisting

```
foo(); // undefined ("foo"
and "bar" exist)

function foo() {
    alert(bar);
}

var bar;
```

```
function foo() {
    alert(bar);
}

var bar;


foo(); // undefined
```

Hoisting

```
var scope = "global ";
function f( ) {
        alert(scope);
        var scope = "local";
        alert(scope);
}
 f( );
```

```
var scope = "global ";
function f( ) {
        var scope;
        alert(scope);
        scope = "local";
        alert(scope);
}
 f( );
```

Only functions introduce new scopes

No block scope

```
function f() {
    { // block starts
        var foo = 4;
    } // block ends
    console.log(foo);
// 4
}
```

No block scope!

```
function test(o) {
      var i = 0;      // i is defined throughout function
      if (typeof o == "object") {
            var j = 0;    // j is defined everywhere, not just block

            for(var k=0; k < 10; k++) {    // k is defined everywhere, not just loop
                document.write(k);
            }

            document.write(k);   // k is still defined: prints 10
      }
      document.write(j);    // j is defined, but may not be initialized
}
```

Hoisting

```
var foo = 1;

function bar() {
     if (!foo) {
          var foo = 10;
     }
     alert(foo);
}

bar();
```

Code in global scope

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Untitled Page</title>
<script type="text/javascript" src="script.js"></script>

</head>
<body>
<script type="text/javascript">
    var a = 5;
    var b = 2;

    function sum(x, y) {
        return x + y;
    }

    function mul(x, y) {
        return x * y;
    }
</script>
</body>
</html>
```

script.js

```
var gloabalVar = 5;
function square(x) {
    return x * x;
}
```

Global namespace

Every variable is global unless it's in a function and is declared with **var**

```
function f(){
    x = “global variable”;
}
f();
```

Global object (WAT)

```
function f(){
    x = “global variable”; // var is missed
}
f();
this.x === “global variable”;
window.x ===  “global variable”; // true for browsers
```

window vs global

```
(function (glob) {
    // glob points to global object
}(typeof window !== 'undefined' ? window :
global));
```

Global variables are evil

- They are less robust, behave less predictably, and are less reusable.
- Name clashes. Your code, built-ins, analytics code, social media buttons use the same global scope.

Globals

```
// antipattern
function sum(x, y) {
     // implied global
     result = x + y;
     return result;
}


// antipattern
function foo() {
     var a = b = 0;
     // ...
}

// preferred
function foo() {
     var a, b;
     // ...
     a = b = 0; // both local
}
```

```
<script>
    if (("a" in window) == false) {
       var a = 1;
    }
    alert(a);
</script>
```

Working with global

```
window.foo

if (window.foo) { … }
if (“foo” in window) { … }
if (typeof “foo” !== “undefined”) {
… }
```

Namespaces

```
if (window.myNamespace == null){
    window.myNamespace = {};
}

window.myNamespace.myFunction = function(/* params*/ ) {
    /* code here */
};
```

immediately invoked function expression
in next lecture

REFERENCES

JavaScript: The Definitive Guide, Six Edition
by David Flanagan

Speaking JavaScript: An In-Depth Guide for Programmers
by Dr. Axel Rauschmayer

http://dmitrysoshnikov.com

http://learn.javascript.ru