levi nine
Technology Services

SCOPE & CLOSURE

Beograd, 25.10.2019.

AGENDA SLIDE

01 Scope and nesting

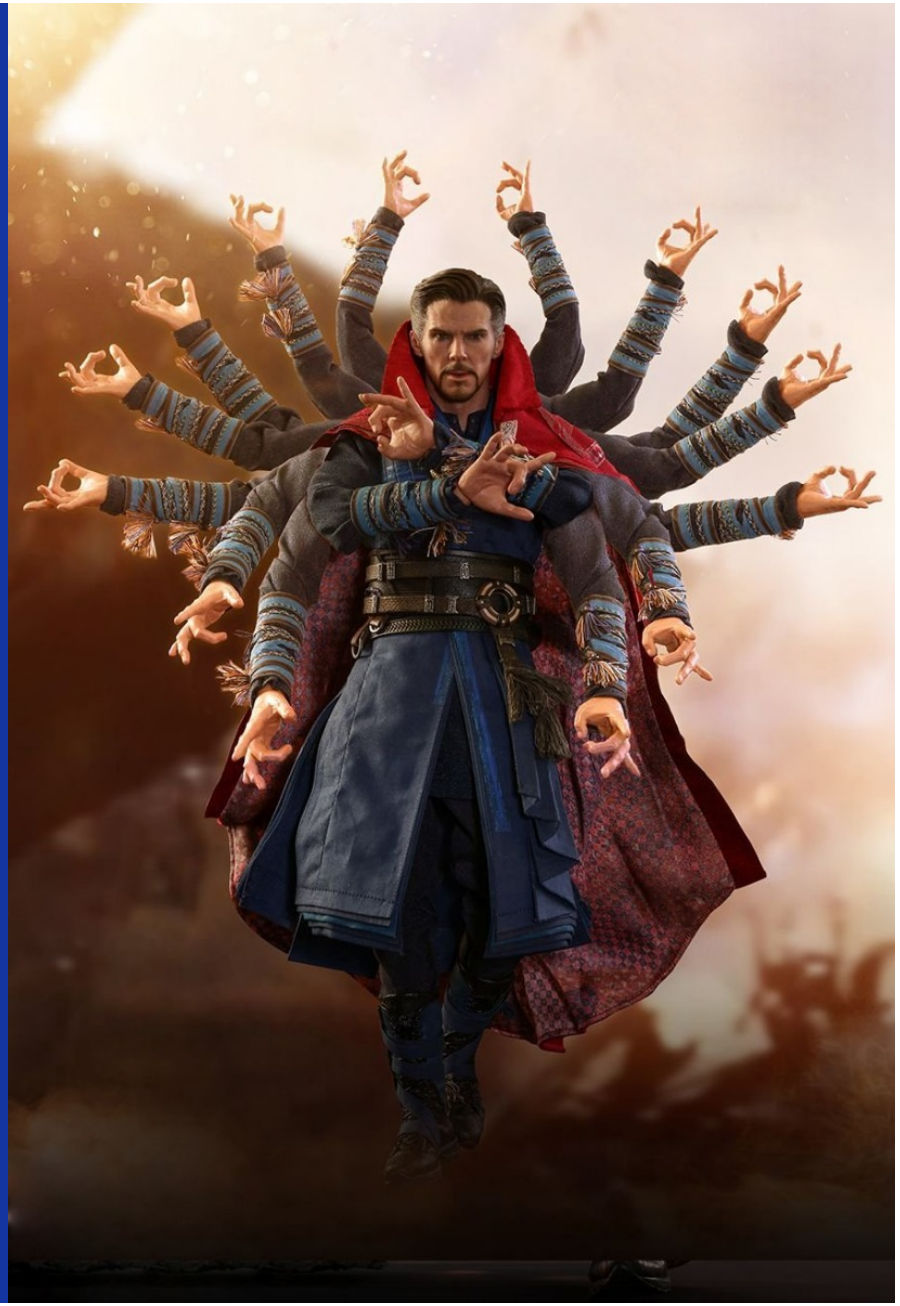
02 Hoisting

03 Closure

04 Modules

01

SCOPE & NESTING



WHAT IS A SCOPE?

- where to look for things
- the set of rules that govern how the engine can look up a variable by its identifier name and find it, either in the current scope, or in any of the nested scopes it's contained within.

LEXICAL SCOPE

- FIRST STAGE OF PARSING

```
function foo(a) {

    var b = a * 2;

    function bar(c) {
        console.log( a, b, c );
    }

    bar(b * 3);
}

foo( 2 );
```

DYNAMIC SCOPE

```
function foo() {
    console.log( a ); // 3  (not 2!)
}

function bar() {
    var a = 3;
    foo();
}

var a = 2;

bar();
```

FUNCTION SCOPING

- JavaScript has function-based scope. That is, each function you declare creates a bubble for itself, but no other structures create their own scope bubbles.

```
var a = 2;

function foo() { // <-- insert this

    var a = 3;
    console.log( a ); // 3

} // <-- and this
foo(); // <-- and this

console.log( a ); // 2
```

IIFE SCOPING

```
var a = 2;

(function IIFE(){

    var a = 3;
    console.log( a ); // 3

})();

console.log( a ); // 2
```

BLOCK SCOPING

- Var for loop - The variable actually scopes itself to the enclosing scope (function or global).
- Let - The let keyword attaches the variable declaration to the scope of whatever block (commonly a { .. } pair) it's contained in
- Const - ES6 introduces const, which also creates a block-scoped variable, but whose value is fixed (constant). Any attempt to change that value at a later time results in an error.

VAR- FOR LOOP CASE

```
for (var i=0; i<10; i++) {
    console.log( i );
}

console.log( i ); // 10 ??????

------------------------------

if(true) {
  var foo = 'bar';
}

console.log( foo ); // 'bar' ??????
```

LET

- Let - The let keyword attaches the variable declaration to the scope of whatever block (commonly a { .. } pair) it's contained in

```
var foo = true;

if (foo) {
    let bar = foo * 2;
    bar = something( bar );
    console.log( bar );
}

console.log( bar ); // ReferenceError
```

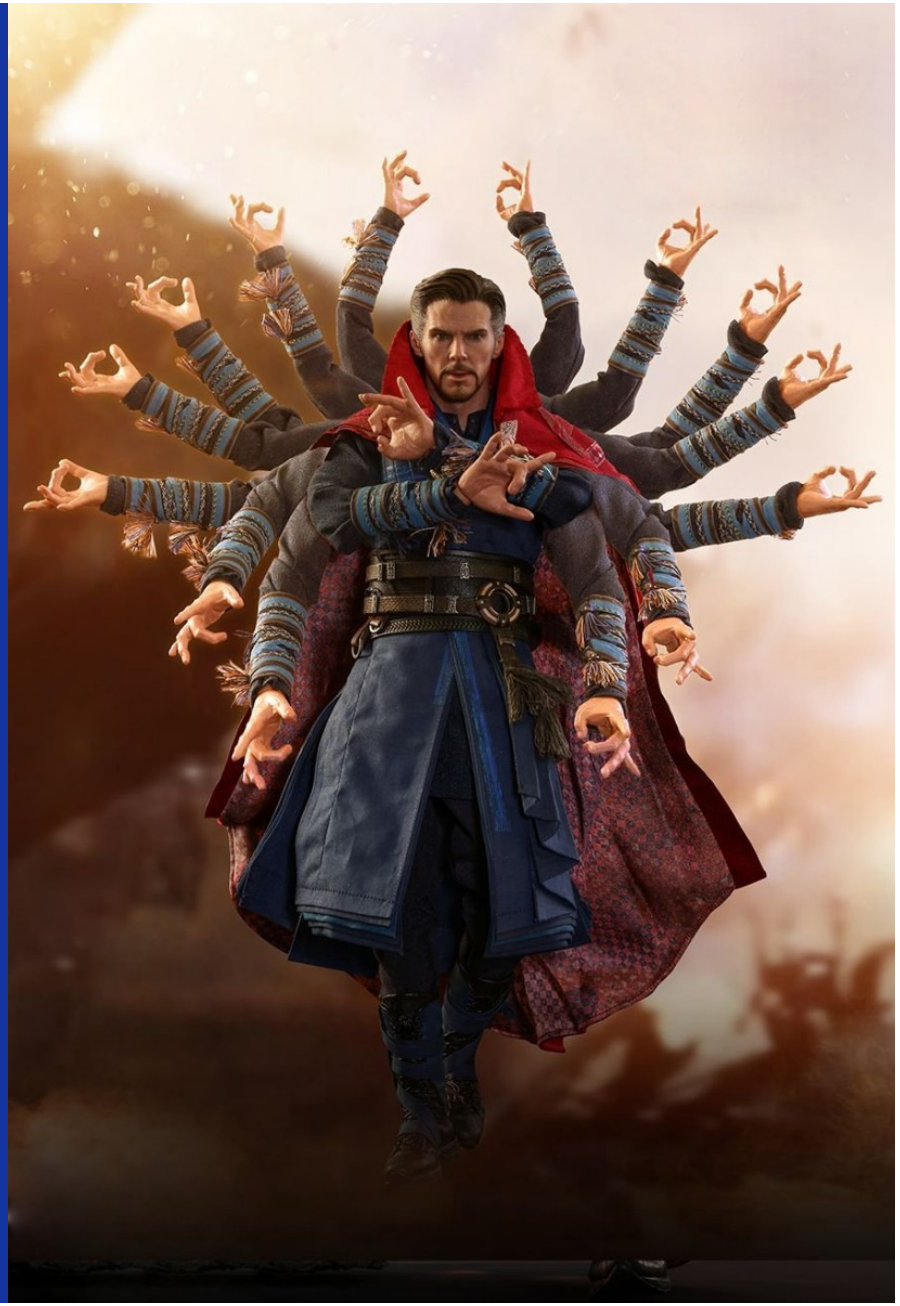
CONST

- ES6 introduces const, which also creates a block-scoped variable, but whose value is fixed (constant). Any attempt to change that value at a later time results in an error.

```
var foo = true;

if (foo) {
    var a = 2;
    const b = 3; // block-scoped to the containing `if`

    a = 3; // just fine!
    b = 4; // error!
}

console.log( a ); // 3
console.log( b ); // ReferenceError!
```

02

HOISTING



HOISTING

- All the code you see in a JavaScript program is interpreted line-by-line, top-down in order, as the program executes. While that is substantially true.

```
a = 2;

var a;

console.log( a );
```

HOISTING

```
foo();

function foo() {
    console.log( a ); // undefined

    var a = 2;
}

----------------------------------------

foo(); // not ReferenceError, but TypeError!

var foo = function bar() {
    // ...
};
```

HOISTING

```
foo(); // 1

var foo;

function foo() {
    console.log( 1 );
}

foo = function() {
    console.log( 2 );
};

----------------------------------------

foo(); // 3

function foo() {
    console.log( 1 );
}

var foo = function() {
    console.log( 2 );
};

function foo() {
    console.log( 3 );
}
```

HOISTING

- Variable and function declarations are “moved” from where they appear in the flow of the code to the top of the code. This gives rise to the name hoisting.

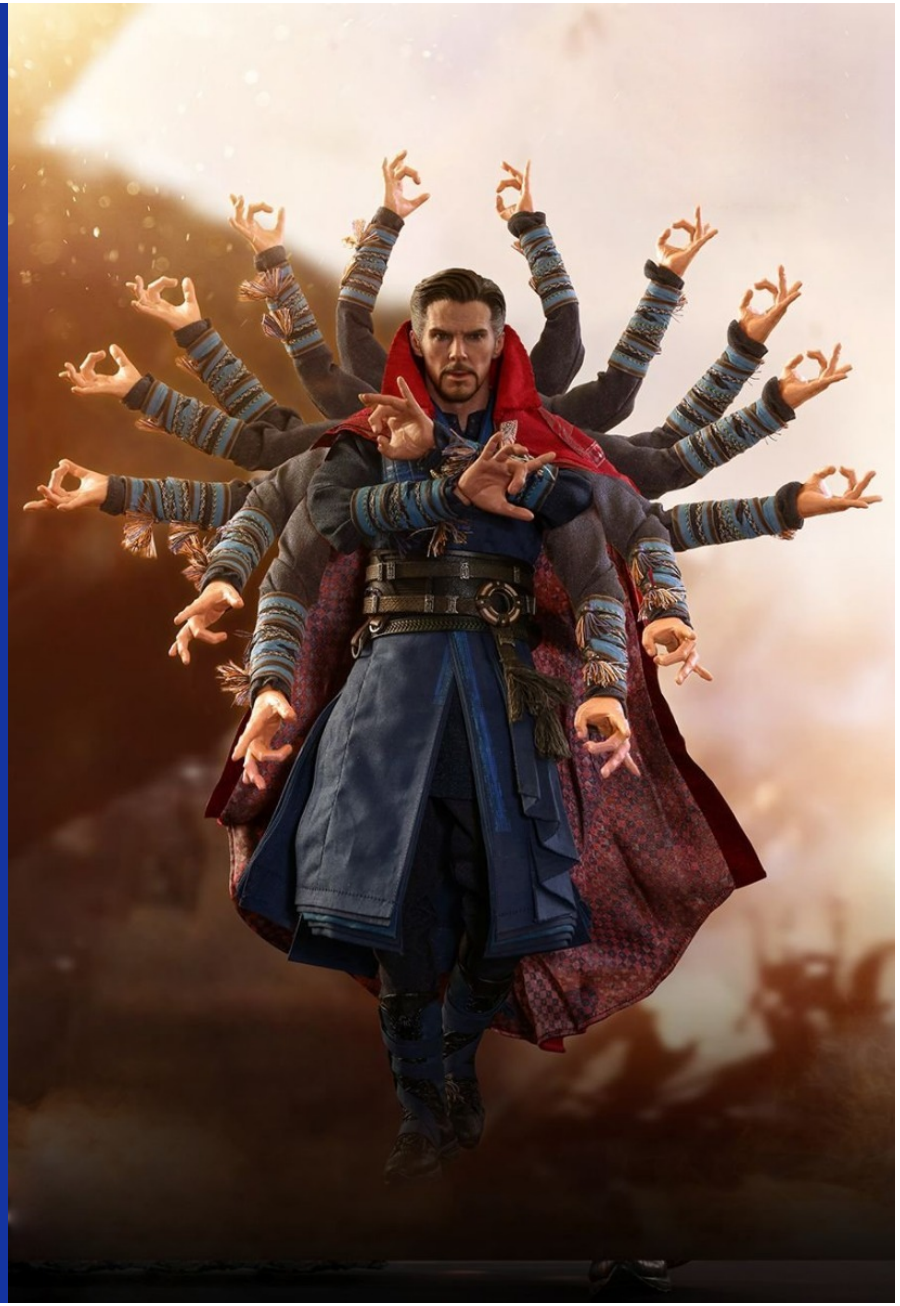
HOISTING

- Example

```
foo(); // will print 'foo' to console
bar(); // will throw an error - bar is not a function

function foo() {
    console.log('foo');
}

var bar = function () {
    console.log('bar');
}
```

03

CLOSURE



CLOSURE

- Closure is when a function is able to remember and access its lexical scope even when that function is executing outside its lexical scope.

```
function foo() {
    var a = 2;

    function bar() {
        console.log( a );
    }

    return bar;
}

var baz = foo();

baz(); // 2 -- Whoa, closure was just observed, man.
```

CLOSURE

```
function foo() {
    var a = 2;

    function baz() {
        console.log( a ); // 2
    }

    bar( baz );
}

function bar(fn) {
    fn(); // look ma, I saw closure!
}
```

CLOSURE

```
var fn;

function foo() {
    var a = 2;

    function baz() {
        console.log( a );
    }

    fn = baz; // assign `baz` to global variable
}

function bar() {
    fn(); // look ma, I saw closure!
}

foo();

bar(); // 2
```

CLOSURE

```
function wait(message) {

    setTimeout( function timer(){
        console.log( message );
    }, 1000 );

}

wait( "Hello, closure!" );
```

CLOSURE

```
function setupBot(name,selector) {
    $( selector ).click( function activator(){
        console.log( "Activating: " + name );
    } );
}

setupBot( "Closure Bot 1", "#bot_1" );
setupBot( "Closure Bot 2", "#bot_2" );
```

CLOSURE

```
var a = 2;

(function IIFE(){
    console.log( a );
})();
```

CLOSURE

```
for (var i=1; i<=5; i++) {
    setTimeout( function timer(){
        console.log( i );
    }, i*1000 );
}
```

CLOSURE

```
for (var i=1; i<=5; i++) {
    (function(j){
        setTimeout( function timer(){
            console.log( j );
        }, j*1000 );
    })( i );
}
```

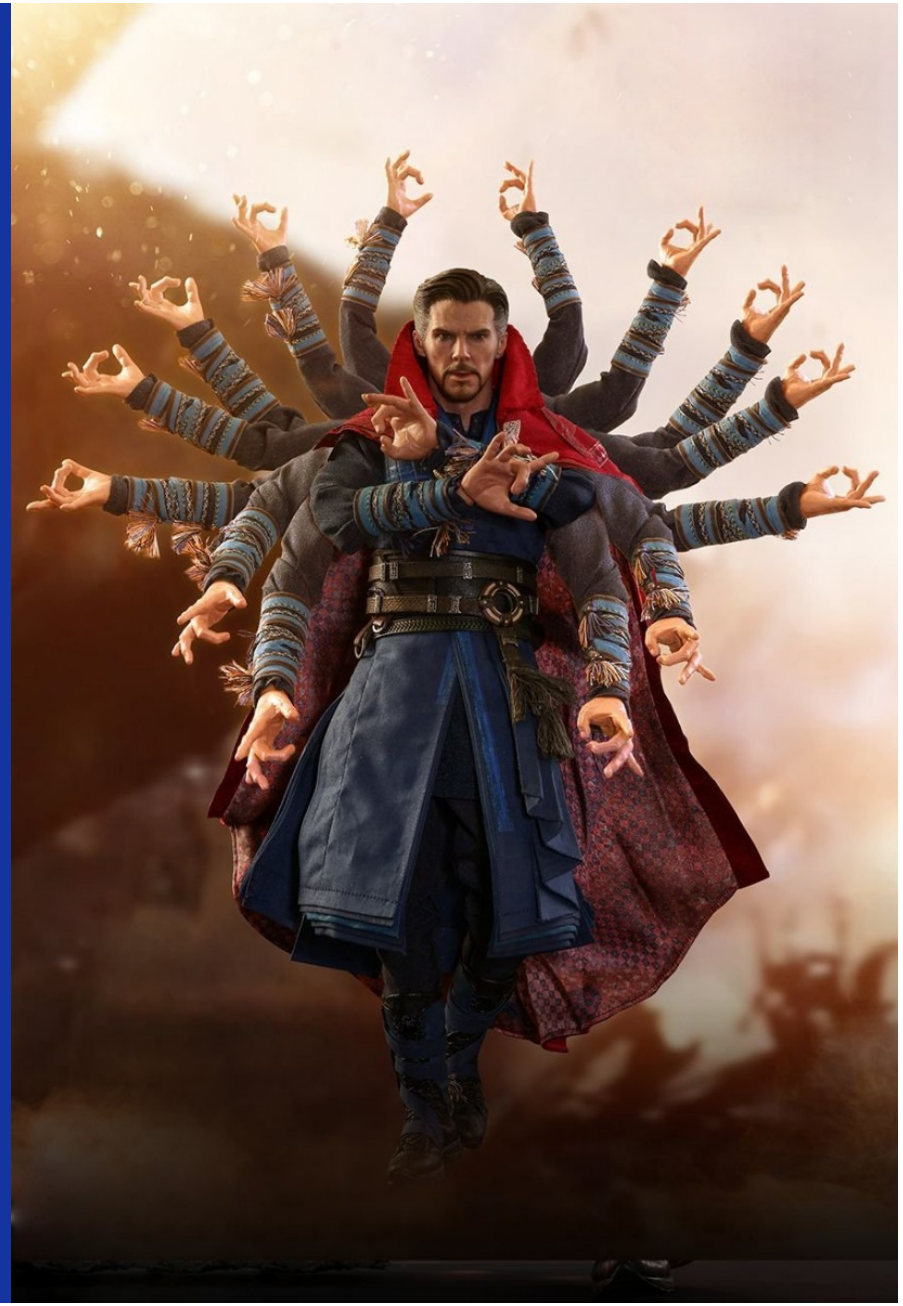
CLOSURE

```
for (let i=1; i<=5; i++) {
    setTimeout( function timer(){
        console.log( i );
    }, i*1000 );
}
```

04

MODULES

Fun part.



MODULES

```
function CoolModule() {
    var something = "cool";
    var another = [1, 2, 3];

    function doSomething() {
        console.log( something );
    }

    function doAnother() {
        console.log( another.join( " ! " ) );
    }

    return {
        doSomething: doSomething,
        doAnother: doAnother
    };
}

var foo = CoolModule();

foo.doSomething(); // cool
foo.doAnother(); // 1 ! 2 ! 3
```

MODULES

```
var foo = (function CoolModule(id) {
    function change() {
        // modifying the public API
        publicAPI.identify = identify2;
    }

    function identify1() {
        console.log( id );
    }

    function identify2() {
        console.log( id.toUpperCase() );
    }

    var publicAPI = {
        change: change,
        identify: identify1
    };

    return publicAPI;
})( "foo module" );

foo.identify(); // foo module
foo.change();
foo.identify(); // FOO MODULE
```

LITERATURE

- Eloquent JavaScript: https://eloquentjavascript.net
- You Don't Know JS: https://github.com/getify/You-Dont-Know-JS/tree/1st-ed

THANK YOU

levi nine

Technology Services