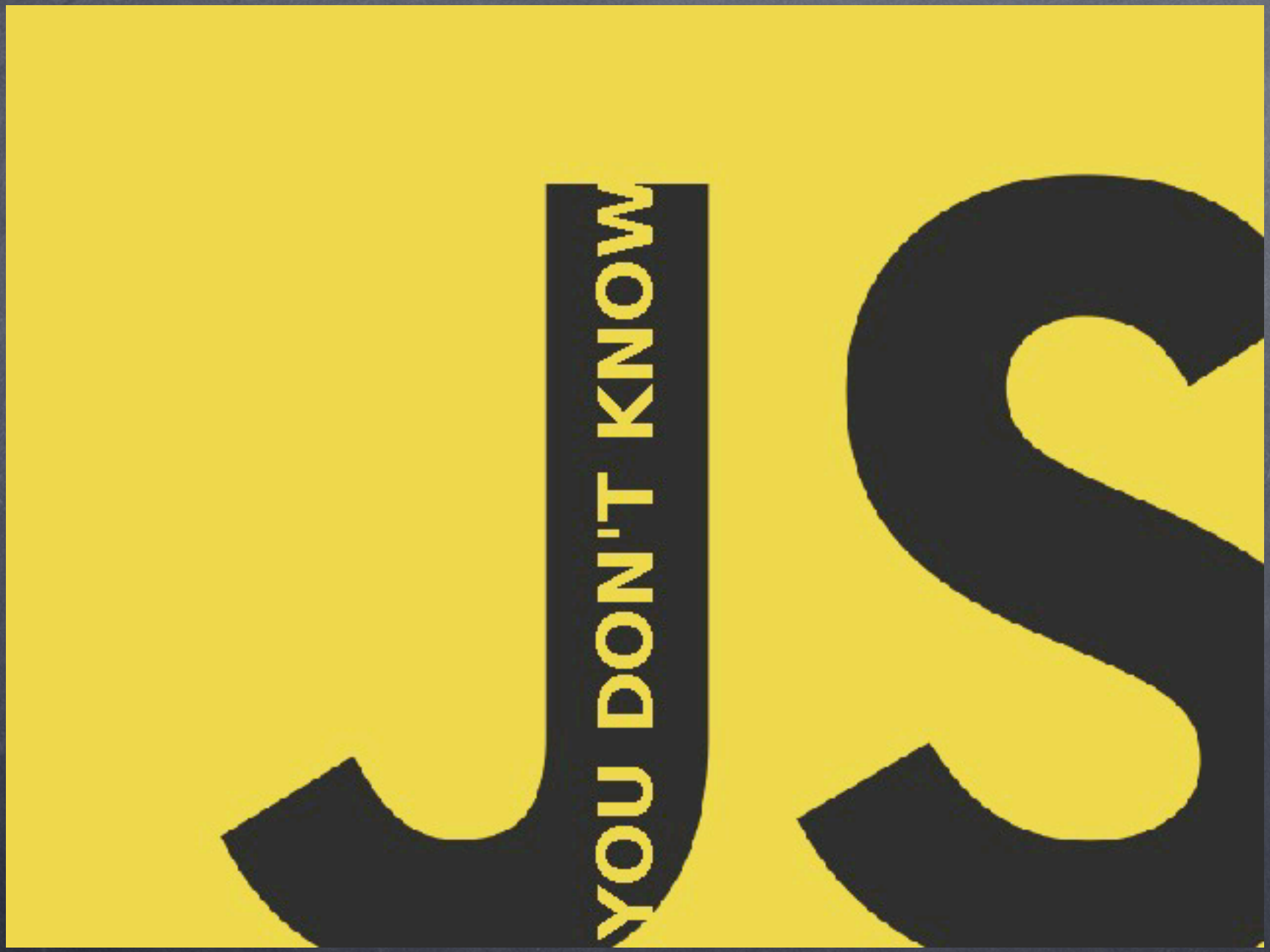
Kyle Simpson
@getify
http://getify.me

- LABjs
- grips
- asynquence

Kyle Simpson
@getify
http://getify.me

http://speakerdeck.com/getify



http://YouDontKnowJS.com

Agenda

Optimizing DOM & Events

- Event Management

Event Management

```
1  var mobsters = ["Lou","Frankie","Max"];
2  var $li;
3  var $hit_list = $("#hit_list");
4
5  for (var i=0; i<mobsters.length; i++) {
6      $li = $("<li></li>")
7      .text(mobsters[i]);
8
9      $hit_list.append($li);
10 }
```


Event Management

```
1 var $hit_list = $("#hit_list");
2
3 $hit_list.children("li")
4 .bind("click",function(evt){
5       var who = $(evt.target).text();
6       alert("You just shot " + who);
7 });
```

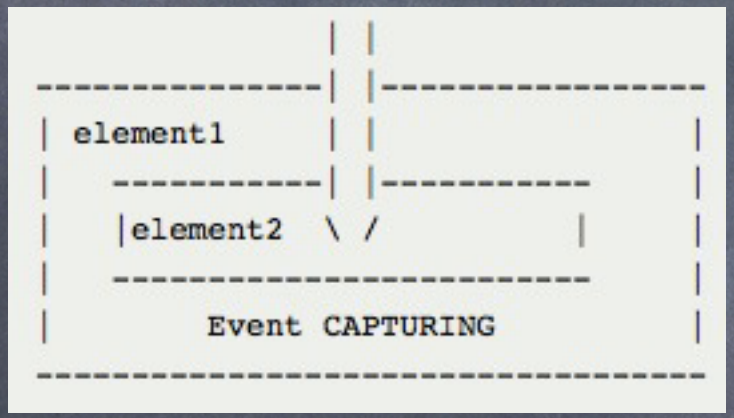
Event Management

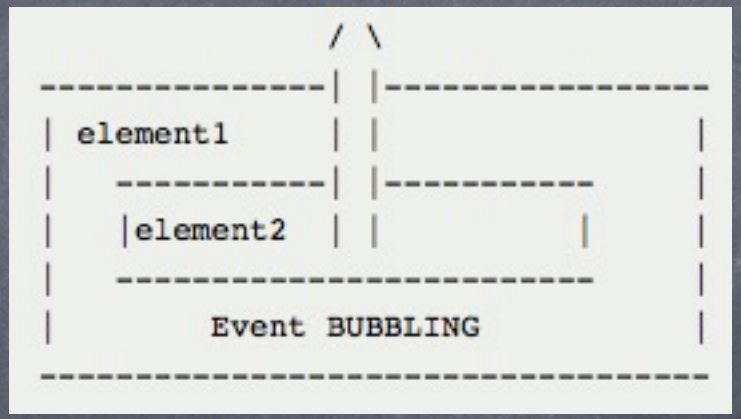
Event Propagation

Capturing

Bubbling

http://www.quirksmode.org/js/events_order.html





Event Management

```
1 var $hit_list = $("#hit_list");
2
3 $hit_list
4 .bind("click",function(evt){
5       var who = $(evt.target).text();
6       alert("You just shot " + who);
7 });
```

Event Management: delegation

```
1 var $hit_list = $("#hit_list");
2
3 $hit_list
4 .bind("click",function(evt){
5     if ($(evt.target).is("li")) {
6         var who = $(evt.target).text();
7         alert("You just shot " + who);
8     }
9 });
```

Event Management: delegation

```
1 var $hit_list = $("#hit_list");
2
3 $hit_list
4 .on("click","> li",function(evt){
5       var who = $(evt.target).text();
6       alert("You just shot " + who);
7 });
```

Event Management: delegation

focus change

Event Management: capturing

```
1 var $hit_list = $("#hit_list");
2
3 $hit_list
4 .bind("focus",function(evt){
5     alert("Watch out!");
6 });
```

Event Management: capturing

```
1  // cheat :)
2  $hit_list.find("li").append(
3      $("<input>")
4  );
5
6  $hit_list[0].addEventListener(
7      "focus",
8      function(evt){
9          console.log("Watch out!");
10     },
11     /*capturing=*/true
12 );
```

Event Management: capturing

http://fuelyourcoding.com/jquery-events-stop-misusing-return-false/

Event Management: canceling

```
1  $("#nav").on(
2      "click",
3      "> a",
4      function(evt){
5          // do something useful!
6
7          // prevent default
8          return false;
9      }
10 );
```

Event Management: canceling

```
$("#nav").on(
        "click",
        "> a",
        function(evt){
                // do something useful!

                // prevent default
                evt.preventDefault();
        }
);
```

Event Management: canceling

```
1  $("#nav").on(
2      "click",
3      "> a",
4      function(evt){
5          // do something useful!
6
7          // prevent propagations
8          evt.stopPropagation();
9          evt.stopImmediatePropagation();
10     }
11 );
```

Event Management: canceling

```
1 $("#nav")[0].addEventListener(
2       "click",
3       function(evt){
4           evt.preventDefault();
5           evt.stopPropagation();
6           evt.stopImmediatePropagation();
7       },
8       /*capturing=*/true
9 );
```

Event Management: canceling

```
1 $("#nav")[0].addEventListener(
2       "focus",
3       function(evt){
4           console.log("cancelable: " +
5               evt.cancelable // false :(
6           );
7       },
8       /*capturing=*/true
9 );
```

Event Management: canceling

(exercise #3: 20min)

```
1  $("#nav").on(
2      "disabled", // custom event!
3      "> a",
4      function(evt){
5          $(evt.target)
6          .addClass("disabled");
7      }
8  );
9
10 // later
11 $("#nav > a").trigger("disabled");
```

Event Management: custom DOM events

https://github.com/hij1nx/EventEmitter2
https://github.com/creationix/eventemitter-browser

```
1 var EVT = new EventEmitter2();
2
3 // later
4 EVT.on("foobar",function(){
5     console.log("foobar fired!");
6 });
7
8 // even later
9 EVT.emit("foobar");
```

Event Management: custom JS events

Agenda

Scope, Closures

- Nested Scope
- Hoisting
- Closure

Scope: where to look for things

JavaScript has
function scope only*

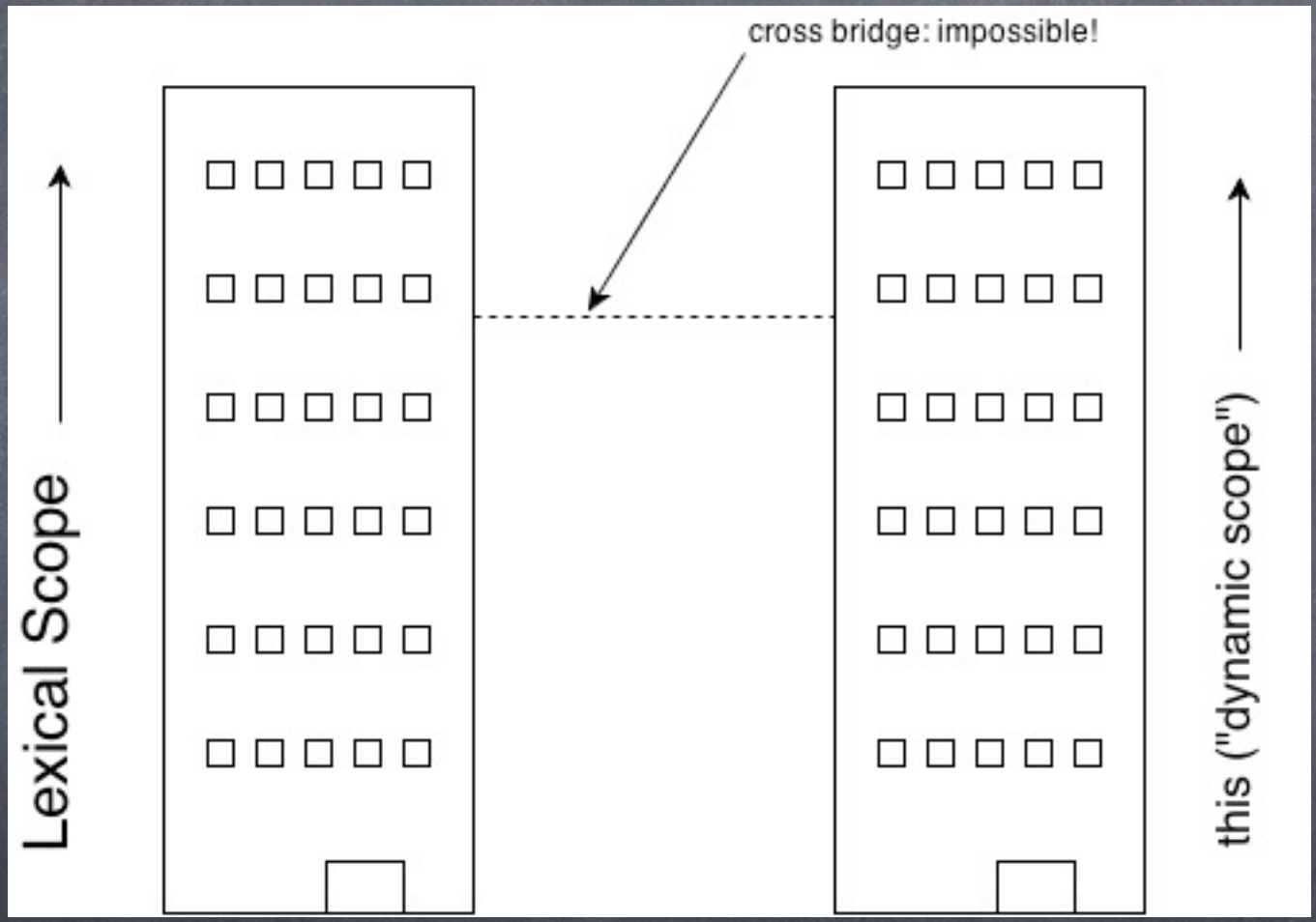
```
 1 var foo = "bar";
 2
 3 function bar() {
 4     var foo = "baz";
 5 }
 6
 7 function baz(foo) {
 8     foo = "bam";
 9     bam = "yay";
10 }
```

Scope

```
1  var foo = "bar";
2
3  function bar() {
4      var foo = "baz";
5
6      function baz(foo) {
7          foo = "bam";
8          bam = "yay";
9      }
10     baz();
11 }
12
13 bar();
14 foo;          // ???
15 bam;          // ???
16 baz();        // ???
```

```
1  var foo = "bar";
2
3  function bar() {
4      var foo = "baz";
5
6      function baz(foo) {
7          foo = "bam";
8          bam = "yay";
9      }
10     baz();
11 }
12
13 bar();
14 foo;          // "bar"
15 bam;          // "yay"
16 baz();        // Error!
```

Scope



Scope

```
1  var foo = function bar() {
2      var foo = "baz";
3
4      function baz(foo) {
5          foo = bar;
6          foo;      // function...
7      }
8      baz();
9  };
10
11 foo();
12 bar();          // Error!
```

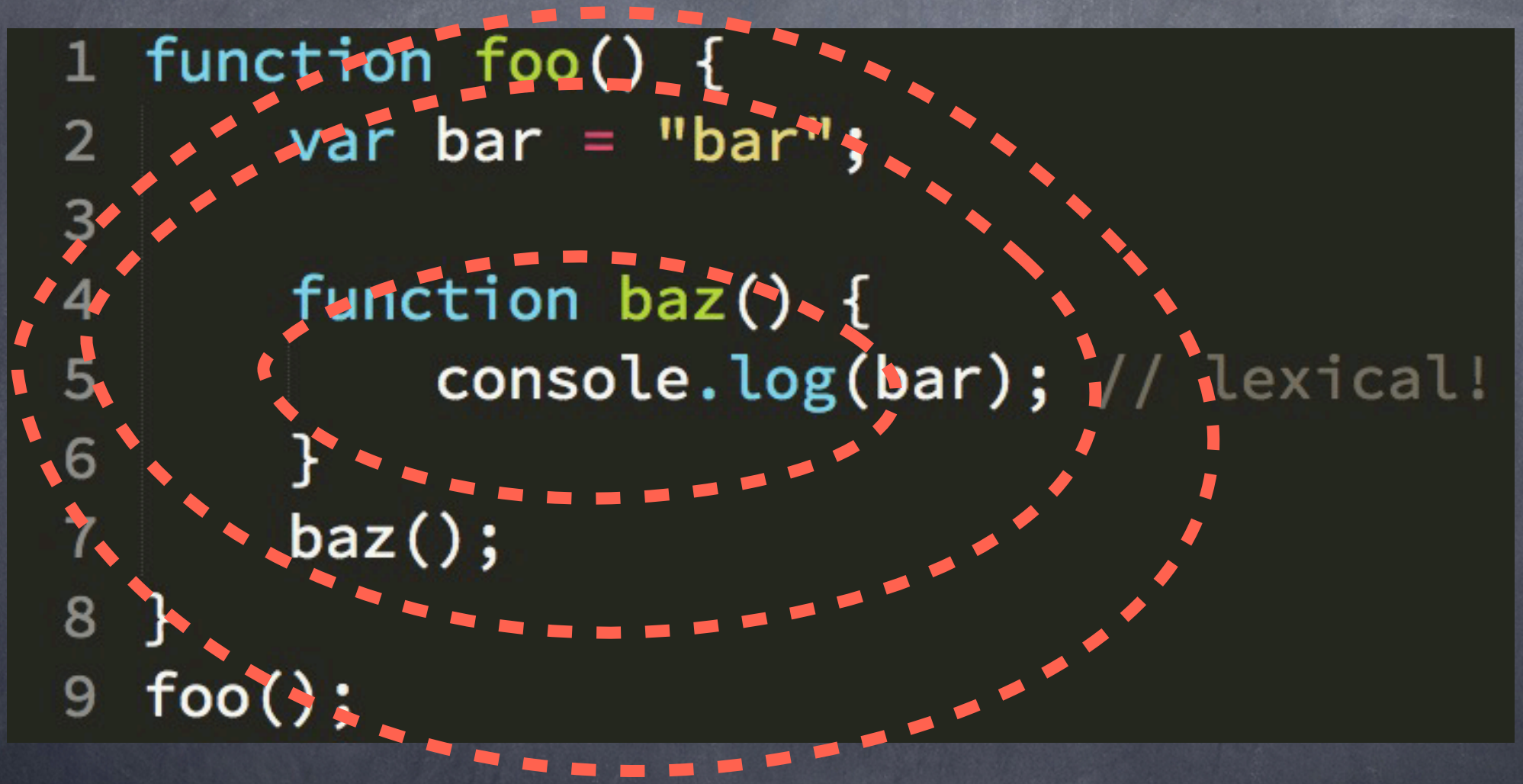
Scope: function scope?

```
1  var foo;
2
3  try {
4      foo.length;
5  }
6  catch (err) {
7      console.log(err); // TypeError
8  }
9
10 console.log(err); // ReferenceError
```

Scope: block scope?

lexical scope

dynamic scope



Scope: lexical

```
1 var bar = "bar";
2
3 function foo(str) {
4     eval(str);  // cheating!
5     console.log(bar); // 42
6 }
7
8 foo("var bar = 42;");
```

Scope: WAT!?

```
var obj = {
    a: 2,
    b: 3,
    c: 4
};

obj.a = obj.b + obj.c;
obj.c = obj.b - obj.a;

with (obj) {
    a = b + c;
    c = b - a;
    d = 3; // <-- Look!
}

obj.d; // undefined
d; // 3 -- oops!
```

Scope: WAT!?

IIFE

```
1  var foo = "foo";
2
3  (function(){
4
5      var foo = "foo2";
6      console.log(foo);    // "foo2"
7
8  })();
9
10 console.log(foo);    // "foo"
```

http://benalman.com/news/2010/11/immediately-invoked-function-expression/

Function Scope

```
1  var foo = "foo";
2
3  (function(bar){
4
5      var foo = bar;
6      console.log(foo);    // "foo"
7
8  })(foo);
9
10 console.log(foo);    // "foo"
```

Function Scope

Block Scope

let (ES6+)

Block Scope

```
1 function foo() {
2     var bar = "bar";
3     for (let i=0; i<bar.length; i++) {
4         console.log(bar.charAt(i));
5     }
6     console.log(i); // ReferenceError
7 }
8
9 foo();
```

Block Scope: let

```
1  function foo(bar) {
2      if (bar) {
3          let baz = bar;
4          if (baz) {
5              let bam = baz;
6          }
7          console.log(bam); // Error
8      }
9      console.log(baz); // Error
10 }
11
12 foo("bar");
```

Block Scope: let

```
1 function foo(bar) {
2     let (baz = bar) {
3         console.log(baz); // "bar"
4     }
5     console.log(baz); // Error
6 }
7
8 foo("bar");
```

Block Scope: let

```
1 function foo(bar) {
2     /*let*/ { let baz = bar;
3         console.log(baz); // "bar"
4     }
5     console.log(baz); // Error
6 }
7
8 foo("bar");
```

https://gist.github.com/getify/5285514

Block Scope: let

```
let (foo) {
    foo = "foo";
    console.log(foo); // "foo"
}

foo; // ReferenceError
```

https://github.com/getify/let-er

Block Scope: let

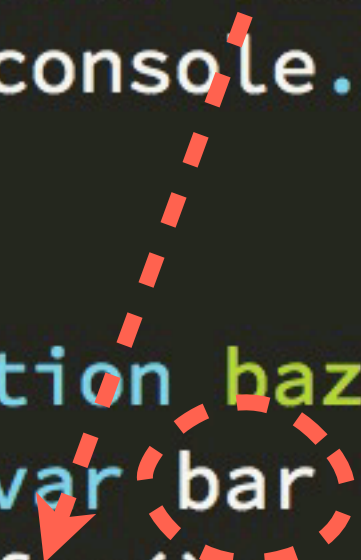
```
1 try{throw void 0}catch
2 /*let*/(foo) {
3     foo = "foo";
4     console.log(foo);
5 }
6
7 foo; // Reference Error!
```

https://github.com/getify/let-er

Block Scope: let

dynamic scope

```
1  // theoretical dynamic scoping
2  function foo() {
3      console.log(bar); // dynamic!
4  }
5
6  function baz() {
7      var bar = "bar";
8      foo();
9  }
10
11 baz();
```



Scope: dynamic!?!?

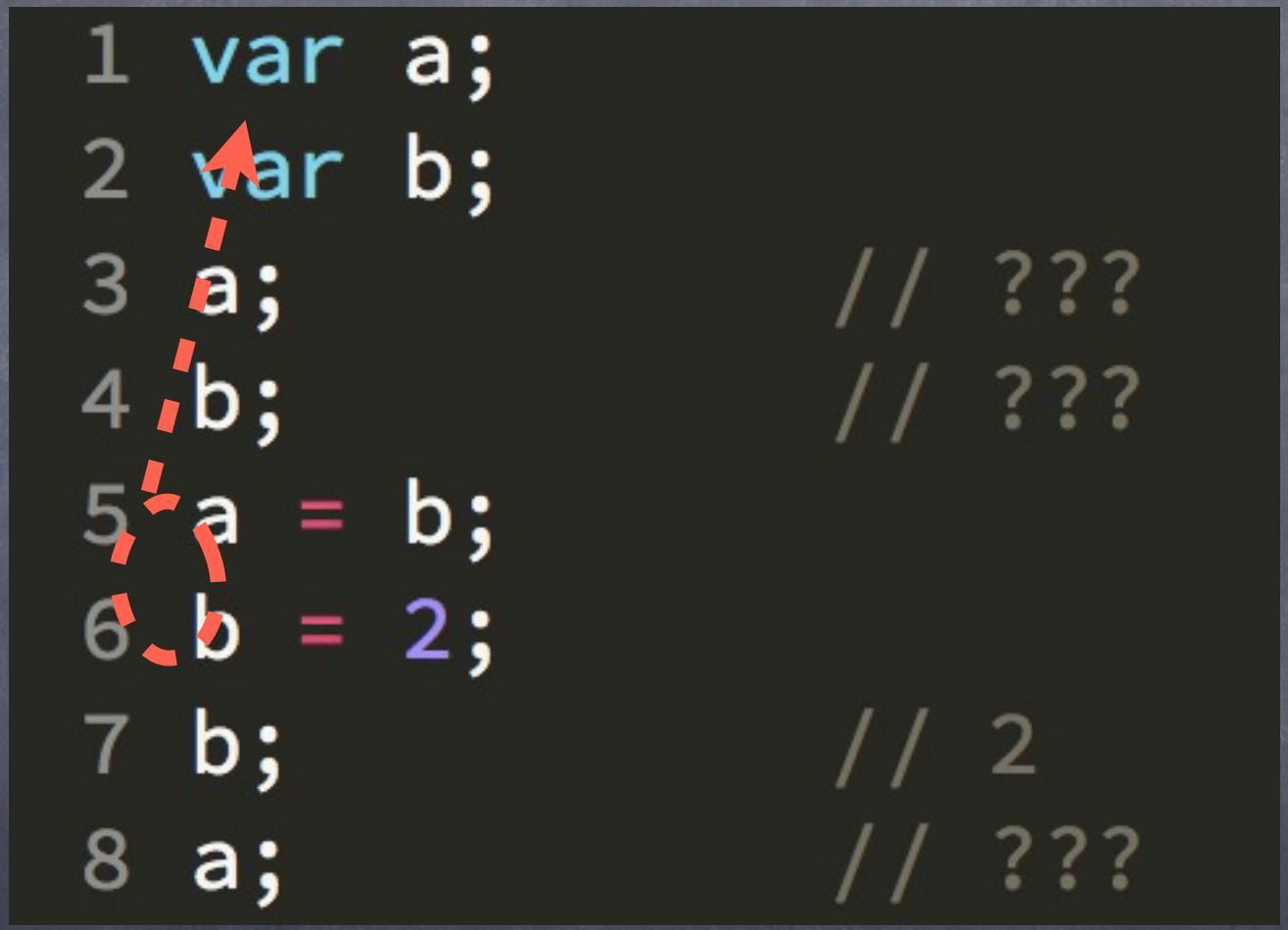
Quiz

1. What type of scoping rule(s) does JavaScript have? Exceptions?
2. What are the different ways you can create a new scope?
3. What's the difference between undeclared and undefined?

Scope: hoisting

```
1 a;                    // ???
2 b;                    // ???
3 var a = b;
4 var b = 2;
5 b;                    // 2
6 a;                    // ???
```

Scope: hoisting



Scope: hoisting

```
1  var a = b();
2  var c = d();
3  a;                    // ???
4  c;                    // ???
5
6  function b() {
7      return c;
8  }
9
10 var d = function() {
11     return b();
12 };
```

Scope: hoisting

```
 1 function b() {
 2     return c;
 3 }
 4 var a;
 5 var c;
 6 var d;
 7 a = b();
 8 c = d();
 9 a;                    // ???
10 c;                    // ???
11 d = function() {
12     return b();
13 };
```

Scope: hoisting

```
1  foo(); // "foo"
2
3  var foo = 2;
4
5  function foo() {
6      console.log("bar");
7  }
8
9  function foo() {
10     console.log("foo");
11 }
```

Hoisting: functions first

```
1  a(1);          // ???
2
3  function a(foo) {
4      if (foo > 20) return foo;
5      return b(foo+2);
6  }
7  function b(foo) {
8      return c(foo) + 1;
9  }
10 function c(foo) {
11     return a(foo*2);
12 }
```

Hoisting: recursion

```
1 function foo(bar) {
2     if (bar) {
3         console.log(baz); // ReferenceError
4         let baz = bar;
5     }
6 }
7
8 foo("bar");
```

Hoisting: let gotcha

Closure

Closure is when a function "remembers" its lexical scope even when the function is executed outside that lexical scope.

Closure

```
1  function foo() {
2      var bar = "bar";
3
4      function baz() {
5          console.log(bar);
6      }
7
8      bam(baz);
9  }
10
11 function bam(baz) {
12     baz();              // "bar"
13 }
14
15 foo();
```

```
1  function foo() {
2      var bar = "bar";
3
4      return function() {
5          console.log(bar);
6      };
7  }
8
9  function bam() {
10     foo()();                    // "bar"
11 }
12
13 bam();
```

Closure

```
1 function foo() {
2     var bar = "bar";
3
4     setTimeout(function() {
5         console.log(bar);
6     },1000);
7 }
8
9 foo();
```

Closure

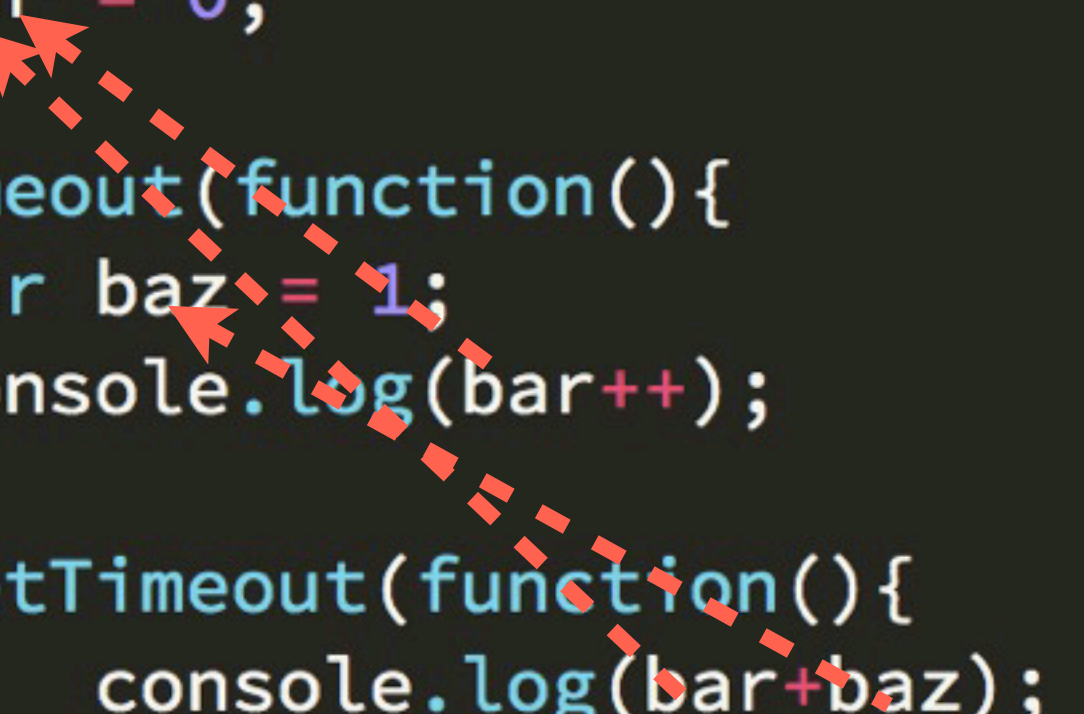
```
1 function foo() {
2     var bar = "bar";
3
4     $("#btn").click(function(evt) {
5         console.log(bar);
6     });
7 }
8
9 foo();
```

Closure

```
1  function foo() {
2      var bar = 0;
3
4      setTimeout(function(){
5          console.log(bar++);
6      },100);
7      setTimeout(function(){
8          console.log(bar++);
9      },200);
10 }
11
12 foo();  // 0 1
```

Closure: shared scope

```
1  function foo() {
2      var bar = 0;
3
4      setTimeout(function(){
5          var baz = 1;
6          console.log(bar++);
7
8          setTimeout(function(){
9              console.log(bar+baz);
10         },200);
11     },100);
12 }
13
14 foo();   // 0 2
```



Closure: nested scope

```
1 for (var i=1; i<=5; i++) {
2       setTimeout(function(){
3           console.log("i: " + i);
4       },i*1000);
5 }
```

Closure: loops

```
1 for (var i=1; i<=5; i++) {
2     (function(i){
3         setTimeout(function(){
4             console.log("i: " + i);
5         },i*1000);
6     })(i);
7 }
```

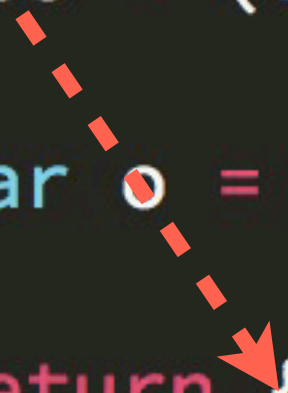
Code Me

Closure: loops

```
1 for (let i=1; i<5; i++) {
2       setTimeout(function(){
3           console.log("i: " + i);
4       },i*1000);
5 }
```

Closure: loops + block scope

```
1 var foo = (function(){
2
3     var o = { bar: "bar" };
4
5     return { obj: o };
6
7 })();
8
9 console.log(foo.obj.bar);    // "bar"
```



Closure?

```
1  var foo = (function(){
2
3      var o = { bar: "bar" };
4
5      return {
6          bar: function(){
7              console.log(o.bar);
8          }
9      };
10
11 })();
12
13 foo.bar();          // "bar"
```

Closure: classic module pattern

```
1  var foo = (function(){
2        var publicAPI = {
3            bar: function(){
4                publicAPI.baz();
5            },
6            baz: function(){
7                console.log("baz");
8            }
9        };
10       return publicAPI;
11 })();
12
13 foo.bar();          // "baz"
```

Closure: modified module pattern

```
define("foo",function(){

    var o = { bar: "bar" };

    return {
        bar: function(){
            console.log(o.bar);
        }
    };

});
```

Closure: modern module pattern

foo.js:

```
1 var o = { bar: "bar" };
2
3 export function bar() {
4     return o.bar;
5 }
```

```
1 import bar from "foo";
2
3 bar();  // "bar"
4
5 import * as foo from "foo";
6
7 foo.bar();  // "bar"
```

Closure: future/ES6+ module pattern

Quiz

1. What is a closure and how is it created?
2. How long does its scope stay around?
3. Why doesn't a function callback inside a loop behave as expected? How do we fix it?
4. How do you use a closure to create an encapsulated module? What's the benefits of that approach?

(exercise #5: 20min)

http://dmitrysoshnikov.com/ecmascript/javascript-the-core/

Kyle Simpson
@getify
http://getify.me

Thanks!

Questions?