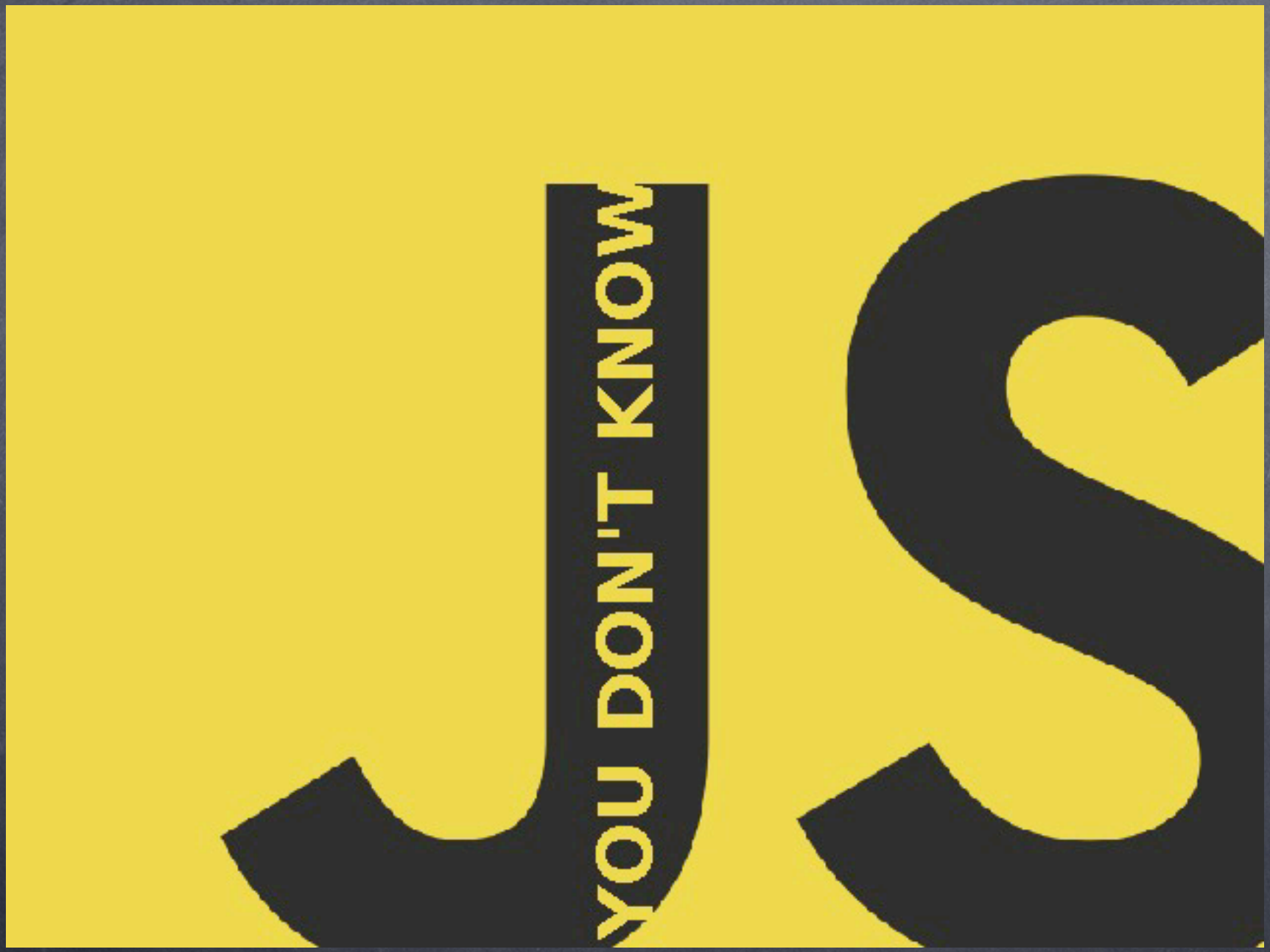
Kyle Simpson
@getify
http://getify.me

- LABjs
- grips
- asynquence

Kyle Simpson
@getify
http://getify.me

http://speakerdeck.com/getify



http://YouDontKnowJS.com

Agenda

Async Patterns

- Callbacks
- Generators/Coroutines
- Promises

Callbacks

```
1 setTimeout(function(){
2     console.log("callback!");
3 },1000);
```

Async Patterns: callbacks

```
1 setTimeout(function(){
2       console.log("one");
3       setTimeout(function(){
4           console.log("two");
5           setTimeout(function(){
6               console.log("three");
7           },1000);
8       },1000);
9 },1000);
```

Async Patterns: "callback hell"

```
1  function one(cb) {
2      console.log("one");
3      setTimeout(cb,1000);
4  }
5  function two(cb) {
6      console.log("two");
7      setTimeout(cb,1000);
8  }
9  function three() {
10     console.log("three");
11 }
12
13 one(function(){
14     two(three);
15 });
```

Async Patterns: "callback hell"

```
1 // line 1
2 setTimeout(function(){
3       // line 3
4       // line 4
5 },1000);
6 // line 2
```

Async Patterns: callbacks

Inversion of Control

Async Patterns: "callback hell"

```
 1 function trySomething(ok,err) {
 2     setTimeout(function(){
 3         var num = Math.random();
 4         if (num > 0.5) ok(num);
 5         else err(num);
 6     },1000);
 7 }
 8
 9 trySomething(
10     function(num){
11         console.log("Success: " + num);
12     },
13     function(num){
14         console.log("Sorry: " + num);
15     }
16 );
```

Async Patterns: separate callbacks

```
1  function trySomething(cb) {
2      setTimeout(function(){
3          var num = Math.random();
4          if (num > 0.5) cb(null,num);
5          else cb("Too low!");
6      },1000);
7  }
8
9  trySomething(function(err,num){
10     if (err) {
11         console.log(err);
12     }
13     else {
14         console.log("Number: " + num)
15     }
16 });
```

Async Patterns: "error-first style"

```
1  function getData(d,cb) {
2      setTimeout(function(){ cb(d); },1000);
3  }
4
5  getData(10,function(num1){
6      var x = 1 + num1;
7      getData(30,function(num2){
8          var y = 1 + num2;
9          getData(
10             "Meaning of life: " + (x + y),
11             function(answer){
12                 console.log(answer);
13                 // Meaning of life: 42
14             }
15         );
16     });
17 });
```

Async Patterns: nested-callback tasks

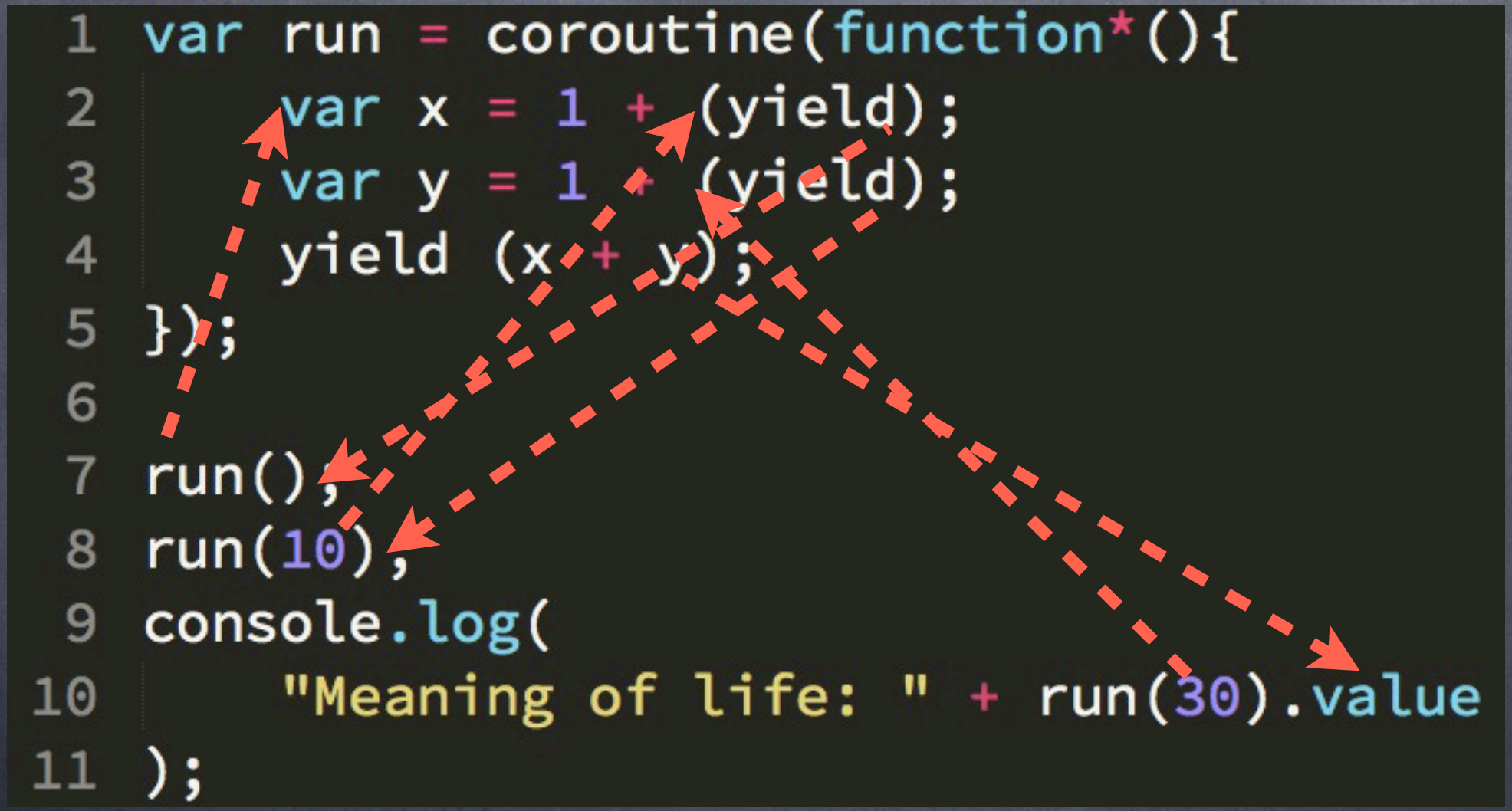
Generators (yield)

```
1 function* gen() {
2     console.log("Hello");
3     yield;
4     console.log("World");
5 }
6
7 var it = gen();
8 it.next(); // Hello
9 it.next(); // World
```

Async Patterns: generators

```
1 function coroutine(g) {
2     var it = g();
3     return function(){
4         return it.next.apply(it,arguments);
5     };
6 }
```

Async Patterns: generator coroutines



Async Patterns: generator messages

```
1  function getData(d) {
2      setTimeout(function(){ run(d); },1000);
3  }
4
5  var run = coroutine(function*(){
6      var x = 1 + (yield getData(10));
7      var y = 1 + (yield getData(30));
8      var answer = (yield getData(
9          "Meaning of life: " + (x + y)
10     ));
11     console.log(answer);
12     // Meaning of life: 42
13 });
14
15 run();
```

Async Patterns: yield tasks

Promises

"continuation events"

```
1  function delay(num) {
2      return new Promise(function(resolve,reject){
3          setTimeout(resolve,num);
4      });
5  }
6
7  delay(100)
8  .then(function(){
9      return delay(50);
10 })
11 .then(function(){
12     return delay(200);
13 })
14 .then(function(){
15     console.log("all done!");
16 });
```

Async Patterns: (native) promise tasks

```
1  function getData(d) {
2      return new Promise(function(resolve,reject){
3          setTimeout(function(){ resolve(d); },1000);
4      });
5  }
6
7  var x;
8
9  getData(10)
10 .then(function(num1){
11     x = 1 + num1;
12     return getData(30);
13 })
14 .then(function(num2){
15     var y = 1 + num2;
16     return getData("Meaning of life: " + (x + y));
17 })
18 .then(function(answer){
19     console.log(answer);
20     // Meaning of life: 42
21 });
```

Async Patterns: (native) promise tasks

sequence = automatically chained promises

https://github.com/getify/asynquence

```
ASQ()
.then(function(done){
    setTimeout(done,1000);
})
.gate(
    function(done){
        setTimeout(done,1000);
    },
    function(done){
        setTimeout(done,1000);
    }
)
.then(function(done){
    console.log("2 seconds passed!");
});
```

Async Patterns: sequences & gates

```
1  function getData(d) {
2      return ASQ(function(done){
3          setTimeout(function(){ done(d); },1000);
4      });
5  }
6
7  ASQ()
8  .waterfall(
9      function(done){ getData(10).pipe(done); },
10     function(done){ getData(30).pipe(done); }
11 )
12 .seq(function(num1,num2){
13     var x = 1 + num1;
14     var y = 1 + num2;
15     return getData("Meaning of life: " + (x + y));
16 })
17 .val(function(answer){
18     console.log(answer);
19     // Meaning of life: 42
20 });
```

Async Patterns: sequence tasks

generators + promises

Async Patterns: async generators

yield promise

```
1  function getData(d) {
2      return ASQ(function(done){
3          setTimeout(function(){ done(d); },1000);
4      });
5  }
6
7  ASQ()
8  .runner(function*(){
9      var x = 1 + (yield getData(10));
10     var y = 1 + (yield getData(30));
11     var answer = yield (getData(
12         "Meaning of life: " + (x + y)
13     ));
14     yield answer;
15 })
16 .val(function(answer){
17     console.log(answer);
18     // Meaning of life: 42
19 });
```

Async Patterns: generator+sequence tasks

Quiz

1. What is "callback hell"? Why do callbacks suffer from "inversion of control"?
2. How do you pause a generator? How do you resume it?
3. What is a Promise? How does it solve inversion of control issues?
4. How do we combine generators and promises for flow control?

(exercise #13: 15min)

Take a deep breath

Kyle Simpson
@getify
http://getify.me

Thanks!

Questions?