KYLE SIMPSON GETIFY@GMAIL.COM

JAVASCRIPT: FUTURE

ES2017 (ES.Next)

- Async Functions
- Object.values() / Object.entries()
- String Padding (no more left-pad!)
- Trailing Commas,,,
- Shared Memory

Async Functions

ES2017

```
1  function printOrders(name) {
2      lookupPerson( name, function onPersonRecord(person){
3          lookupOrders( person.id, function onOrders(orders){
4              for (let order of orders) {
5                  // ..
6              }
7          } );
8      } );
9  }
10
11 printOrders( "Kyle Simpson" );
```

async functions: callback concurrency

```
1  function printOrders(name) {
2      lookupPerson( name )
3      .then( function onPersonRecord(person){
4          return lookupOrders( person.id );
5      } )
6      .then( function onOrders(orders){
7          for (let order of orders) {
8              // ..
9          }
10     } );
11 }
12
13 printOrders( "Kyle Simpson" );
```

async functions: promise concurrency

```
1  function *printOrders(name) {
2      var person = yield lookupPerson( name );
3      var orders = yield lookupOrders( person.id );
4      for (let order of orders) {
5          // ..
6      }
7  }
8
9  runner(
10     printOrders( "Kyle Simpson" )
11 );
```

async functions: generator+promise concurrency

```
1 async function printOrders(name) {
2     var person = await lookupPerson( name );
3     var orders = await lookupOrders( person.id );
4     for (let order of orders) {
5         // ..
6     }
7 }
8
9 printOrders( "Kyle Simpson" );
```

async functions: await+promise concurrency

```
1  async function printOrders(name) {
2      try {
3          var person = await lookupPerson( name );
4      }
5      catch (promiseRejection) {
6          // ..
7      }
8
9      var orders = await lookupOrders( person.id );
10     for (let order of orders) {
11         // ..
12     }
13 }
14
15 printOrders( "Kyle Simpson" );
```

async functions: sync errors!

```
1  async function printOrders(name) {
2      var person = await lookupPerson( name );
3      var orders = await lookupOrders( person.id );
4      var oldOrders =
5          await lookupOldOrders( person.id );
6
7      for (let order of [...orders, ...oldOrders]) {
8          // ..
9      }
10 }
11
12 printOrders( "Kyle Simpson" );
```

async functions: overly sequential

```
1  async function printOrders(name) {
2      var person = await lookupPerson( name );
3      var [orders,oldOrders] = await Promise.all( [
4          lookupOrders( person.id );
5          lookupOldOrders( person.id )
6      ] );
7
8      for (let order of [...orders, ...oldOrders]) {
9          // ..
10     }
11 }
12
13 printOrders( "Kyle Simpson" );
```

async functions: promises to the rescue!

```
1  async function lookupPerson( search ) {
2      var id =
3          await lookupId( "person", search );
4
5      var { fullName: name, email } =
6          await lookupPersonDetails( id );
7
8      return { id, name, email };
9  }
10
11 async function printOrders(name) {
12     var person = await lookupPerson( name );
13
14     // ..
15 }
```

async functions: async promises

Object.values() / Object.entries()

ES2017

```
1  var arr = [1,2,3];
2
3  for (let v of arr) {
4      console.log( v );
5  }
6  // 1 2 3
7
8  var obj = { a:1, b:2, c:3 };
9
10 for (let k in obj) {
11     console.log( obj[k] );
12 }
13 // 1 2 3
```

object iterators: no for-of

```
1  var obj = { a:1, b:2, c:3 };
2
3  obj[Symbol.iterator] = function*(){
4      for (let key of Object.keys( this )) {
5          yield this[key];
6      }
7  };
8
9  for (var v of obj) {
10     console.log( v );
11 }
12 // 1 2 3
```

object iterators: manual

```
1 var obj = { a:1, b:2, c:3 };
2
3 for (let v of Object.values( obj )) {
4     console.log( v );
5 }
6 // 1 2 3
```

object iterators: values()

```
1 var obj = { a:1, b:2, c:3 };
2
3 for (let [k,v] of Object.entries( obj )) {
4     console.log( `${k}:${v}` );
5 }
6 // a:1 b:2 c:3
```

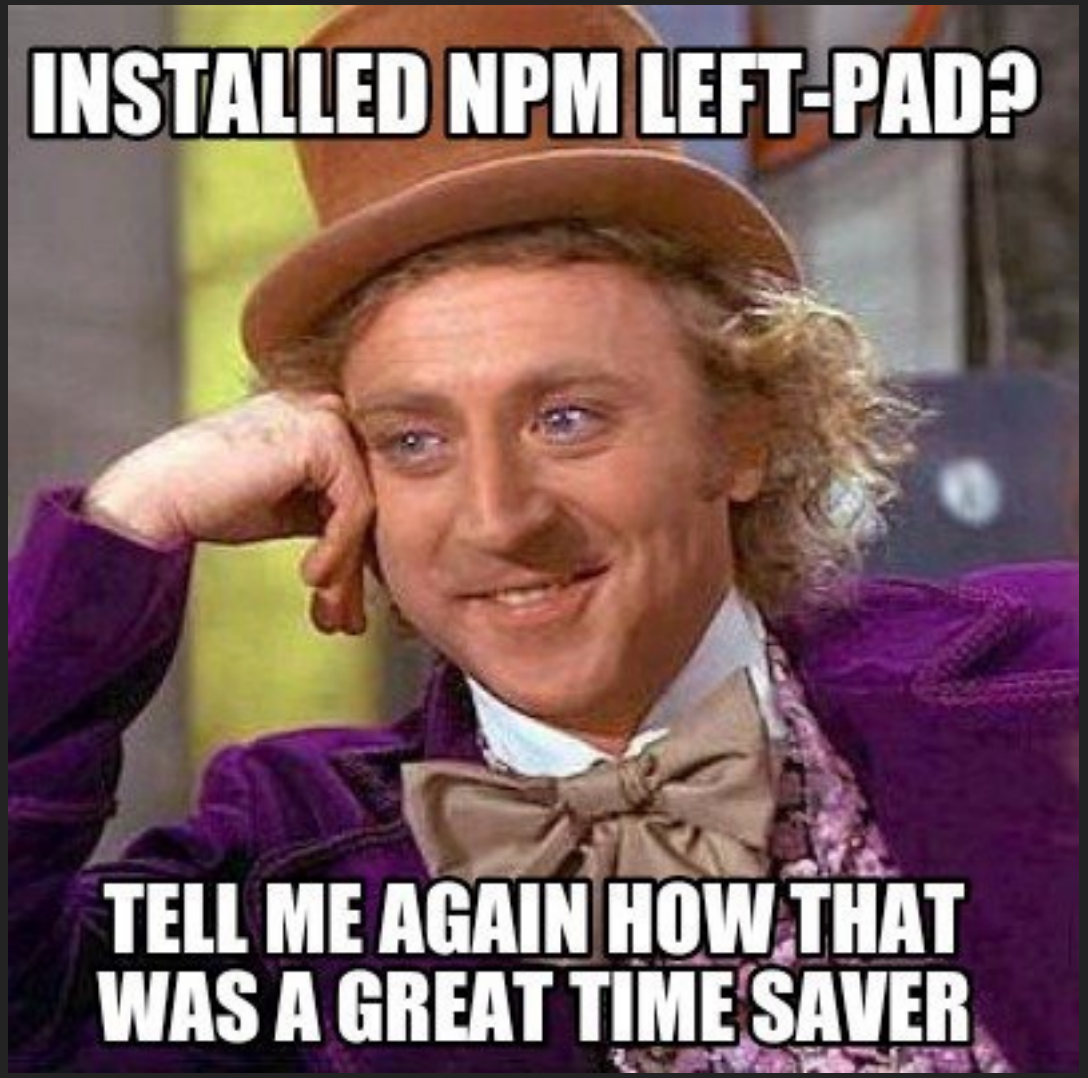
object iterators: entries()

```
1  var obj = { a:1, b:2, c:3 };
2
3  Object.keys( obj );
4  // ["a","b","c"]
5  Object.values( obj );
6  // [1,2,3]
7  Object.entries( obj );
8  // [["a",1],["b",2],["c",3]]
9
10 var arr = [1,2,3];
11
12 [...arr.keys()];
13 // [0,1,2]
14 [...arr.values()];  // <--- BE CAREFUL!
15 // [1,2,3]
16 [...arr.entries()];
17 // [[0,1],[1,2],[2,3]]
```

object iterators: arrays not iterators

String Padding

ES2017



string padding: facepalm

```
1 "abc".padStart( 5 );
2 // "  abc"
3
4 "abc".padStart( 5, "-" );
5 // "--abc"
6
7 "abc".padStart( 5, "012345" );
8 // "01abc"
```

string padding: left aka start

```
1 "abc".padEnd( 5 );
2 // "abc  "
3
4 "abc".padEnd( 5, "-" );
5 // "abc--"
6
7 "abc".padEnd( 5, "012345" );
8 // "abc01"
```

string padding: right aka end

Trailing Commas

(in function signatures and calls)

ES2017

```
1  var arr = [
2      1,
3      2,
4      3,
5  ];
6
7  var obj = {
8      a: 1,
9      b: 2,
10     c: 3,
11 };
```

trailing commas: git diffs ftw

```
1  function foo(
2      a,
3      b,
4      c,
5  ) {
6      // ..
7  }
8
9  foo(
10     1,
11     2,
12     3,
13 );
```

trailing commas: functions get love too

```
1 var
2         a = 1,
3         b = 2,
4         c = 3,
5 ;
```

trailing commas: no love for statements

Shared Memory

(with web workers)

ES2017

```
1  var shared = new SharedArrayBuffer( 40 );
2  var local = new Int32Array( shared );
3  var worker = new Worker( "worker.js" );
4
5  worker.postMessage( {shared} );
```

```
1  self.onMessage = function({data: {shared}}) {
2      var remote = new Int32Array( shared );
3
4      // ..
5  };
```

shared memory: typed array buffers

```
1  var shared = new SharedArrayBuffer( 40 );
2  var local = new Int32Array( shared );
3  var worker = new Worker( "worker.js" );
4
5  worker.postMessage( {shared} );
6
7  var meaningOfLife = 42;
8  // store value at index `5`
9  Atomics.store( local, 5, meaningOfLife );
```

```
1  self.onMessage = function({data: {shared}}) {
2      var remote = new Int32Array( shared );
3
4      // essentially a mutex
5      Atomics.wait( remote, 5, 0 );
6      var meaningOfLife = Atomics.load( remote, 5 );
7
8      console.log( meaningOfLife );
9      // 42
10 };
```

shared memory: atomics

ES.Beyond

- Rest/Spread Properties (...)
- Async Iteration
- do { Expression }s
- Null Propagation Operator (?.)

Rest/Spread Properties

(stage 3)

```
1 function lookupRecord(id,...otherParams) {
2     return db.lookup(
3         "people-records", id, ...otherParams
4     );
5 }
```

```
1 function getSomeData() {
2     var someVals = [2,3];
3
4     return [1,...someVals,4];
5 }
6
7 var [
8     a,
9     b,
10    ...otherVals
11 ] = getSomeData();
12
13 otherVals;
14 // [3,4]
```

Finder

rest/spread properties: recall array gather/spread

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

rest/spread properties: property gather (aka rest)

```
1  function getSomeData() {
2      var someVals = { b: 2, c: 3 };
3
4      return { a: 1, ...someVals, d: 4 };
5  }
6
7  var {
8      a,
9      b,
10     ...otherProps
11 } = getSomeData();
12
13 otherProps;
14 // { c: 3, d: 4 }
```

rest/spread properties: property spread

Async Iteration

(stage 3)

```
1  async function printOrders(name) {
2      var person = await lookupPerson( name );
3
4      for (let order of getOrders( person.id )) {
5          // ..
6      }
7  }
```

async iteration: sync vs async

```
1 async function printOrders(name) {
2     var person = await lookupPerson( name );
3
4     do {
5         let { value: order, done } =
6             await getNextOrder( person.id );
7         // ..
8     } while (!done);
9 }
```

async iteration: getting one at a time

```
1  async function printOrders(name) {
2      var person = await lookupPerson( name );
3
4      for await (let order of getOrders( person.id )) {
5          // ..
6      }
7  }
```

async iteration: for...await

```
1  async function *getOrders(personId) {
2      var cursor = db.query( "orders", personId );
3      do {
4          let record = await cursor.next();
5          yield record;
6      } while (!cursor.done);
7  }
8
9  async function printOrders(name) {
10     var person = await lookupPerson( name );
11
12     for await (let order of getOrders( person.id )) {
13         // ..
14     }
15 }
```

async iteration: async generators

Do Expressions

(stage 1)

```
1 var x = (function(){
2     function foo(v) {
3         return v * 2;
4     }
5
6     var a = 3;
7
8     return foo( a * 7 );
9 })();
```

do expressions: statements in IIFEs

```
1 var x = do {
2     function foo(v) {
3         return v * 2;
4     }
5
6     var a = 3;
7
8     foo( a * 7 );
9 };
```

do expressions: statements in do{ }

```
1 var x;
2
3 try { x = foo( 10 ); }
4 catch (err) { x = err; }
```

```
1 var x = do {
2     try { foo( 10 ); }
3     catch (err) { err; }
4 };
```

do expressions: capturing statement completion

```
1 bar( do {
2         try { foo( 10 ); }
3         catch (err) { 10; }
4 } );
```

```
1 bar( do {
2         switch (mode) {
3             case 1: 10; break;
4             case 2: 15; break;
5             case 3: 20; break;
6             default: 100;
7         }
8 } );
```

do expressions: avoiding temporary variables

Null Propagation Operator

(stage 1)

```
1  var city = person.addresses[0].city;
2  // many oops
3
4  var person;
5  var city = person ?
6      person.addresses ?
7          person.addresses[0] ?
8              person.addresses[0].city :
9              undefined :
10         undefined :
11     undefined;
```

null propagation operator: property access hazards

```
1 var city = person?.addresses?.[0]?.city;
```

null propagation operator: conditional access

ES.Beyond'er

- Observables
- class: Decorators, Public/Private Fields
- Promise Extensions
- RegExp: Dot-All, Look-Behind, Named Capture
- SIMD
- ...more

github.com/tc39/proposals

THANKS!!!!

KYLE SIMPSON GETIFY@GMAIL.COM

JAVASCRIPT: FUTURE