KYLE SIMPSON GETIFY@GMAIL.COM

ES6: THE RIGHT PARTS

ES6 / ES2015

- Arrow Functions (=>)
- Object Literal Extensions
- Block Scoping (let, const)
- Default Parameters
- Rest/Spread Operator (...)
- Destructuring
- Interpolated String Literals (`..`)
- Symbols
- Iterators + Generators

Arrow Functions

ES6 / ES2015

```
1  function two() {
2      return 2;
3  }
4
5  function identity(v) {
6      return v;
7  }
8
9  function sum(x,y) {
10     return x + y;
11 }
```

arrow functions

```
() => 2

v => v

(x,y) => x + y
```

arrow functions: terse syntax!

```
1  () => 2
2  _ => 2
3  x => 2
4
5  v => v
6  (...v) => v
7  (v = 2) => v
8  ([v]) => v
9  ({v}) => v
10
11 (x,y) => x + y
12
13 v => ({ prop: v })
14 v => {
15     try { return v() } catch (err) {}
16 }
```

arrow functions: lots of syntax...

```
1 var v = (f,g) => f()
2
3 var v = (f,g) => f(), g
4
5 var v = ((f,g) => f)(), g
6
7 var v = (f,g) => (f(), g)
```

arrow functions: operator precedence fun

```
var f1 = v => v > 10 ? v % 2 == 0 ? v + 1 : v : 10
f1( 12 )
// 13

var f2 = v => (v.push( 10 ), v)
f2( [8,9] )
// [8,9,10]

var f3 = (v,m) => (m = v.indexOf( 10 ), v[ ~m ? m : 0])
f3( [8,9] )
// 8

var f4 = v => m => () => v + m
f4( 8 )( 9 )()
// 17
```

arrow functions: syntactic tricks

```
1 var ids = records.map( r => r.id )
2
3 var ids = records.map( function recordId(r){
4     return r.id
5 } )
```

arrow functions: debugging, readability

EXERCISE 0

Object Literal Extensions

ES6 / ES2015

```
1 var foo = 2;
2
3 var obj = {
4     foo: foo
5 };
6
7 obj.foo;     // 2
```

```
1 var foo = 2;
2
3 var obj = {
4     foo
5 };
6
7 obj.foo;     // 2
```

object literals: concise properties

```
1 var obj = {
2     foo: function foo() {
3         // ..
4     }
5 };
```

```
1 var obj = {
2     foo() {
3         // ..
4     }
5 };
```

object literals: concise methods

```
1 var prop = "abc";
2
3 var obj = {
4 };
5
6 obj[prop] = 1;
```

```
1 var prop = "abc";
2
3 var obj = {
4     [prop]: 1
5 };
```

object literals: computed properties

```
1 var prop = "abc";
2
3 var obj = {
4     [prop]() { /* .. */ },
5     *[prop+"Gen"]() { /* .. */ }
6 };
```

object literals: computed methods/generators

```
1  var obj = {
2      __v: 10,
3      get prop() {
4          return this.__v;
5      },
6      set prop(v) {
7          this.__v = v;
8      }
9  };
10
11 obj.prop;              // 10
12 obj.prop = 42;
13 obj.prop;              // 42
```

object literals: getters/setters (ES5)

Block Scoping

ES6 / ES2015

```
1 function diff(x,y) {
2     if (x > y) {
3         var tmp = x;
4         x = y;
5         y = tmp;
6     }
7
8     return y - x;
9 }
```

block scoping

```
1 function diff(x,y) {
2     if (x > y) {
3         let tmp = x;
4         x = y;
5         y = tmp;
6     }
7
8     return y - x;
9 }
```

block scoping: let

```
1 function repeat(fn,n) {
2     var result;
3
4     for (var i = 0; i < n; i++) {
5         result = fn( result, i );
6     }
7
8     return result;
9 }
```

block scoping: not all vars...

```
1 function repeat(fn,n) {
2     var result;
3
4     for (let i = 0; i < n; i++) {
5         result = fn( result, i );
6     }
7
8     return result;
9 }
```

block scoping: let + var

```
1  function lookupRecord(searchStr) {
2      try {
3          var id = getRecord( searchStr );
4      }
5      catch (err) {
6          var id = -1;
7      }
8
9      return id;
10 }
```

block scoping: sometimes, var > let

```
1  function formatStr(str) {
2      { let prefix, rest;
3          prefix = str.slice( 0, 3 );
4          rest = str.slice( 3 );
5          str = prefix.toUpperCase() + rest;
6      }
7
8      if (/^FOO:/.test( str )) {
9          return str;
10     }
11
12     return str.slice( 4 );
13 }
```

block scoping: explicit let

```
1 var x = 2;
2 ++x;
3 // 3
4
5 const y = 2;
6 ++y;
7 // Error!
```

block scoping: const

```
1 var a = [0,2,3];
2 ++a[0];
3 a;
4 // [1,2,3]
5
6 const b = [0,2,3];
7 ++b[0];
8 b;
9 // [1,2,3]    <--- whoa!
```

block scoping: not so const...ant

MORE CODE

EXERCISE 1

Default Parameters

ES6 / ES2015

```
1 function lookupRecord(id) {
2     // id = id || -1
3     id = id !== undefined ? id : -1;
4
5     // ..
6 }
```

default: imperative

```
1 function lookupRecord(id = -1) {
2     // ..
3 }
```

default: declarative syntax

```
1 function req(name) {
2     throw new Error( name + " is required!" );
3 }
4
5
6 function lookupRecord(id = req( "id" )) {
7     // ..
8 }
```

default: required parameter

MORE CODE

Rest/Spread Operator

ES6 / ES2015

```
1  function lookupRecord(id) {
2      var otherParams = [].slice.call( arguments, 1 );
3
4      // ..
5  }
```

rest: imperative

```
1 function lookupRecord(id) {
2     var otherParams = [].slice.call( arguments, 1 );
3     otherParams.shift(
4         "people-records", id.toUpperCase()
5     );
6     return db.lookup.apply( null, otherParams );
7 }
```

spread: imperative

```
1 function lookupRecord(id,...otherParams) {
2     // ..
3 }
```

rest: aka "gather"

gather: declarative

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

spread: declarative

MORE CODE

EXERCISE 2

Destructuring

decomposing a structure into
its individual parts

ES6 / ES2015

```
1  var tmp = getSomeRecords();
2
3  var first = tmp[0];
4  var second = tmp[1];
5
6  var firstName = first.name;
7  var firstEmail = first.email !== undefined ?
8      first.email :
9      "nobody@none.tld";
10
11 var secondName = second.name;
12 var secondEmail = second.email !== undefined ?
13     second.email :
14     "nobody@none.tld";
```

destructuring: imperative

```
1  var [
2      {
3          name: firstName,
4          email: firstEmail = "nobody@none.tld"
5      },
6      {
7          name: secondName,
8          email: secondEmail = "nobody@none.tld"
9      }
10 ] = getSomeRecords();
```

destructuring: declarative

```
1  function lookupRecord(store = "person-records",id = -1) {
2      // ..
3  }
4
5  function lookupRecord({
6      store = "person-records",
7      id = -1
8  }) {
9      // ..
10 }
11
12 lookupRecord({id: 42});
```

destructuring: named arguments

MORE CODE

EXERCISE 3

Interpolated String Literals

ES6 / ES2015

```
1  var name = "Kyle Simpson";
2  var email = "getify@gmail.com";
3  var title = "Teacher";
4
5  var msg = "Welcome to this class! Your " +
6      title + " is " + name + ", contact: " +
7      email + ".";
8
9  // Welcome to this class! Your Teacher is
10 // Kyle Simpson, contact: getify@gmail.com.
```

string interpolation: imperative

```
1  var name = "Kyle Simpson";
2  var email = "getify@gmail.com";
3  var title = "Teacher";
4
5  var msg = `Welcome to this class! Your
6  ${title} is ${name}, contact: ${email}.`;
7
8  // Welcome to this class! Your
9  // Teacher is Kyle Simpson, contact: getify@gmail.com.
```

string interpolation: declarative

```
1 var amount = 12.3;
2
3 var msg =
4     formatCurrency
5 `The total for your
6 order is ${amount}`;
7
8 // The total for your
9 // order is $12.30
```

string interpolation: tagged

```
1  function formatCurrency(strings,...values) {
2      var str = "";
3      for (let i = 0; i < strings.length; i++) {
4          if (i > 0) {
5              if (typeof values[i-1] == "number") {
6                  str += `$${values[i-1].toFixed(2)}`;
7              }
8              else {
9                  str += values[i-1];
10             }
11         }
12         str += strings[i]
13     }
14     return str;
15 }
```

string interpolation: tagged

EXERCISE 4

Symbols

```
1  var x = Symbol();
2  var y = Symbol();
3  var z = Symbol("some description");
4
5  x === y;          // false
6
7  x.toString();     // "Symbol()"
8  z.toString();     // "Symbol(some description)"
9
10 x + "";           // TypeError!
```

symbols: primitives

```
1  var p = Symbol("some secret prop");
2
3  var obj = {
4      [p]: 42
5  };
6
7  obj.p;                          // undefined
8  obj["some secret prop"];        // undefined
9  Object.keys(obj);               // []
10
11 Object.getOwnPropertyNames(obj);
12 // []
13
14 Object.getOwnPropertySymbols(obj);
15 // [ Symbol(some secret prop) ]
```

symbols: ~~private~~ obscured properties

```
1  Symbol.toStringTag;
2  Symbol.isConcatSpreadable;
3  Symbol.species;
4  Symbol.toPrimitive;
5  Symbol.iterator;
6  // ...
7
8  var obj = {
9      [Symbol.toStringTag]: "hello!"
10 };
11
12 obj.toString();          // "[object hello!]"
```

symbols: well known (WKS)

Iterators + Generators

ES6 / ES2015

```
1  var str = "Hello";
2  var world = ["W","o","r","l","d"];
3
4  var it1 = str[Symbol.iterator]();
5  var it2 = world[Symbol.iterator]();
6
7  it1.next();      // { value: "H", done: false }
8  it1.next();      // { value: "e", done: false }
9  it1.next();      // { value: "l", done: false }
10 it1.next();      // { value: "l", done: false }
11 it1.next();      // { value: "o", done: false }
12 it1.next();      // { value: undefined, done: true }
13
14 it2.next();      // { value: "W", done: false }
15 // ..
```

iterators: built-in iterators

```
1  var str = "Hello";
2
3  for (
4      let it = str[Symbol.iterator](), v, result;
5      (result = it.next()) && !result.done &&
6          (v = result.value || true);
7  ) {
8      console.log(v);
9  }
10 // "H" "e" "l" "l" "o"
```

iterators: imperative iteration

```
1  var str = "Hello";
2  var it = str[Symbol.iterator]();
3
4  for (let v of it) {
5      console.log(v);
6  }
7  // "H" "e" "l" "l" "o"
8
9  for (let v of str) {
10     console.log(v);
11 }
12 // "H" "e" "l" "l" "o"
```

iterators: declarative iteration

```
1 var str = "Hello";
2
3 var letters = [...str];
4 letters;
5 // ["H","e","l","l","o"]
```

iterators: declarative iteration

```
1  var obj = {
2      a: 1,
3      b: 2,
4      c: 3
5  };
6
7  for (let v of obj) {
8      console.log(v);
9  }
10 // TypeError!
```

iterators: objects not iterables

```
1  var obj = {
2      a: 1,
3      b: 2,
4      c: 3,
5      [Symbol.iterator]: function(){
6          var keys = Object.keys(this);
7          var index = 0;
8          return {
9              next: () =>
10                 (index < keys.length) ?
11                     { done: false, value: this[keys[index++]] } :
12                     { done: true, value: undefined }
13         };
14     }
15 };
16
17 [...obj];
18 // [1,2,3]
```

iterators: imperative iterator

```
1  function *main() {
2      yield 1;
3      yield 2;
4      yield 3;
5      return 4;
6  }
7
8  var it = main();
9
10 it.next();       // { value: 1, done: false }
11 it.next();       // { value: 2, done: false }
12 it.next();       // { value: 3, done: false }
13 it.next();       // { value: 4, done: true }
14
15 [...main()];
16 // [1,2,3]
```

iterators: generators

```
1  var obj = {
2      a: 1,
3      b: 2,
4      c: 3,
5      *[Symbol.iterator](){
6          for (let key of Object.keys(this)) {
7              yield this[key];
8          }
9      }
10 };
11
12 [...obj];
13 // [1,2,3]
```

iterators: declarative iterator

EXERCISE 5

ES2016

- Exponentiation Operator (**)
- Array .includes(..)

Exponentiation Operator

ES2016

```
1 var x = Math.pow( 3, 4 );
2 // 81
3
4 var y = 3 ** 4;
5 // 81
```

exponent syntax

Array .includes(..)

ES2016

```
1  var arr = [1,2,3,4,5];
2
3  ~arr.indexOf( 2 );
4  // -2    <-- truthy
5
6  arr.includes( 2 );
7  // true
8
9  arr.includes( 6 );
10 // false
```

includes API > syntax

THANKS!!!!

KYLE SIMPSON GETIFY@GMAIL.COM

ES6: THE RIGHT PARTS