ES-6

ECMAScript 6

ecma

- Formerly known as European Computer Manufacturers Association
- Developed general purpose, cross platform, vendor-neutral programming language (ES-5 and ES-6)

ES-5	ES-6*
All Modern Browsers	Safari (100%) Chrome (~97%) Firefox (~92%) IE 11 (0%) Microsoft Edge (~95)

* Statistics accurate on 08-Jan-2017 | compatibility chart

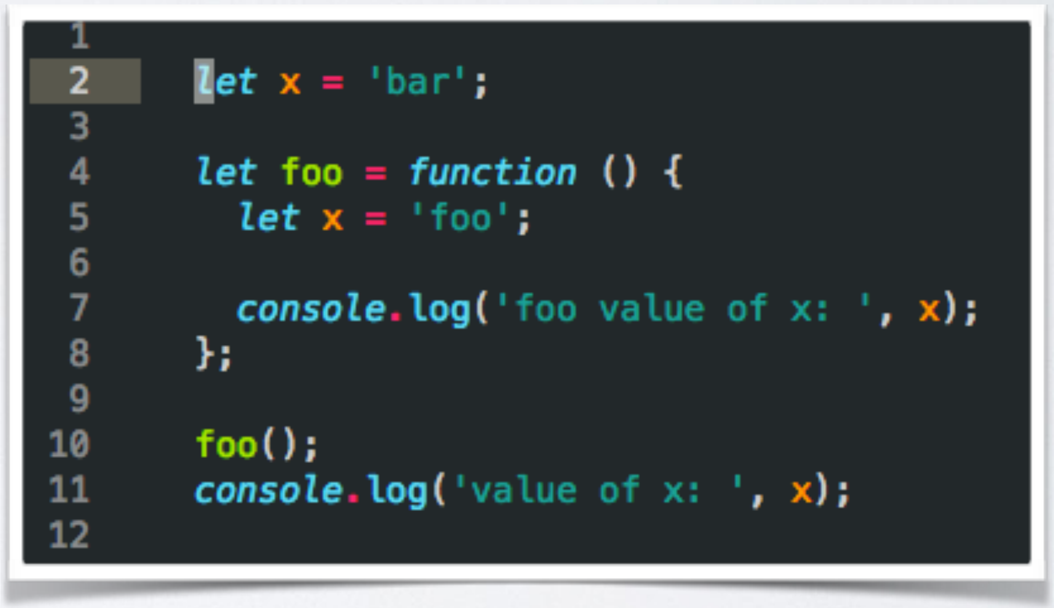
BABEL & ES-6

- Syntax transformation
- Transforms ES-6 into browser compatible ES-5
- Extensible
    - React JSX tranformer

VARIABLES

LET

- declares variable for local scope
- scope limited to block, statement, expression, etc

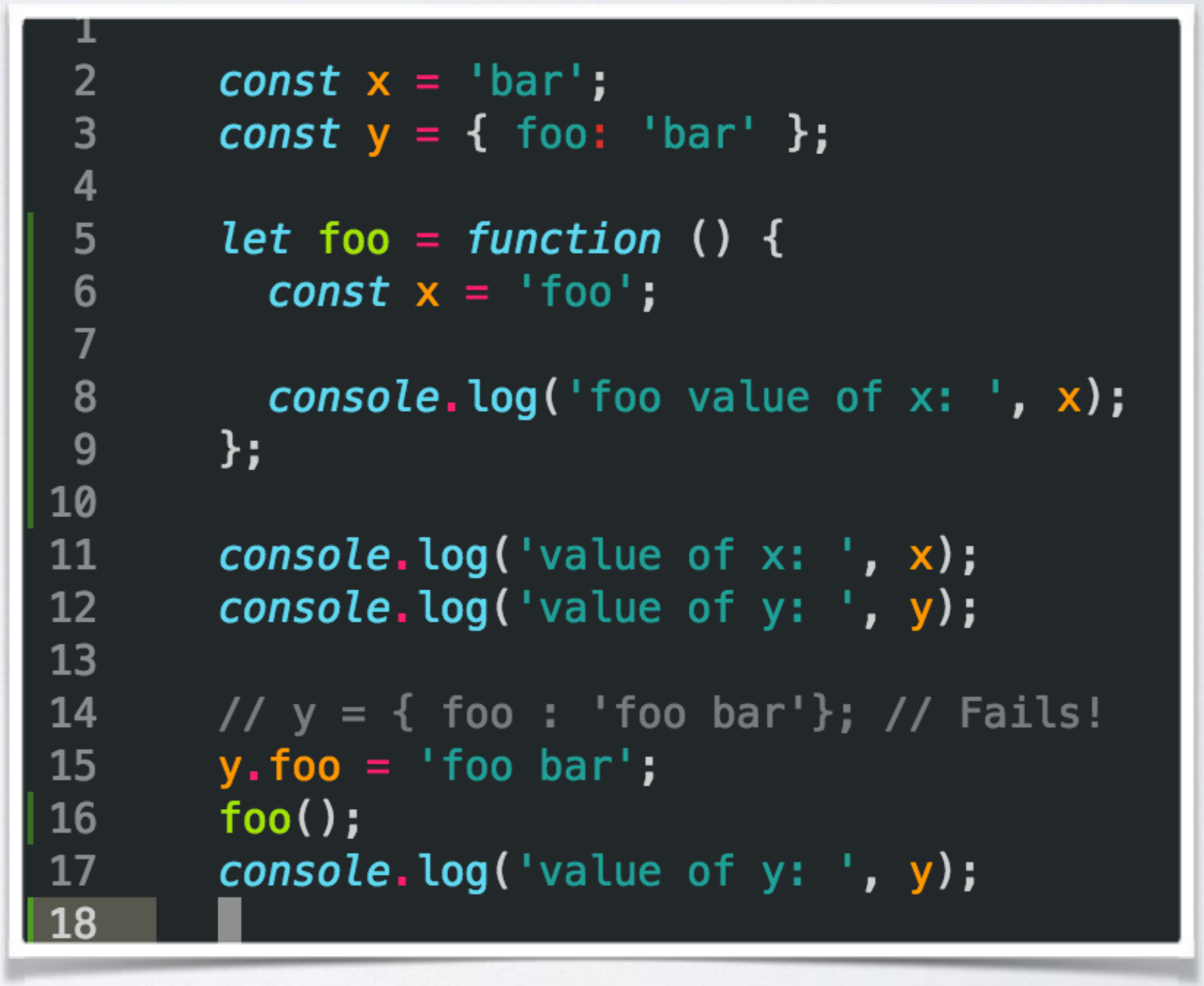


```

let x = 'bar';

let foo = function () {
  let x = 'foo';

  console.log('foo value of x: ', x);
};

foo();
console.log('value of x: ', x);

```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

CONST

- declares variable that cannot be reassigned
- scope works just like `let`
- must declare value in statement
- value remains mutable

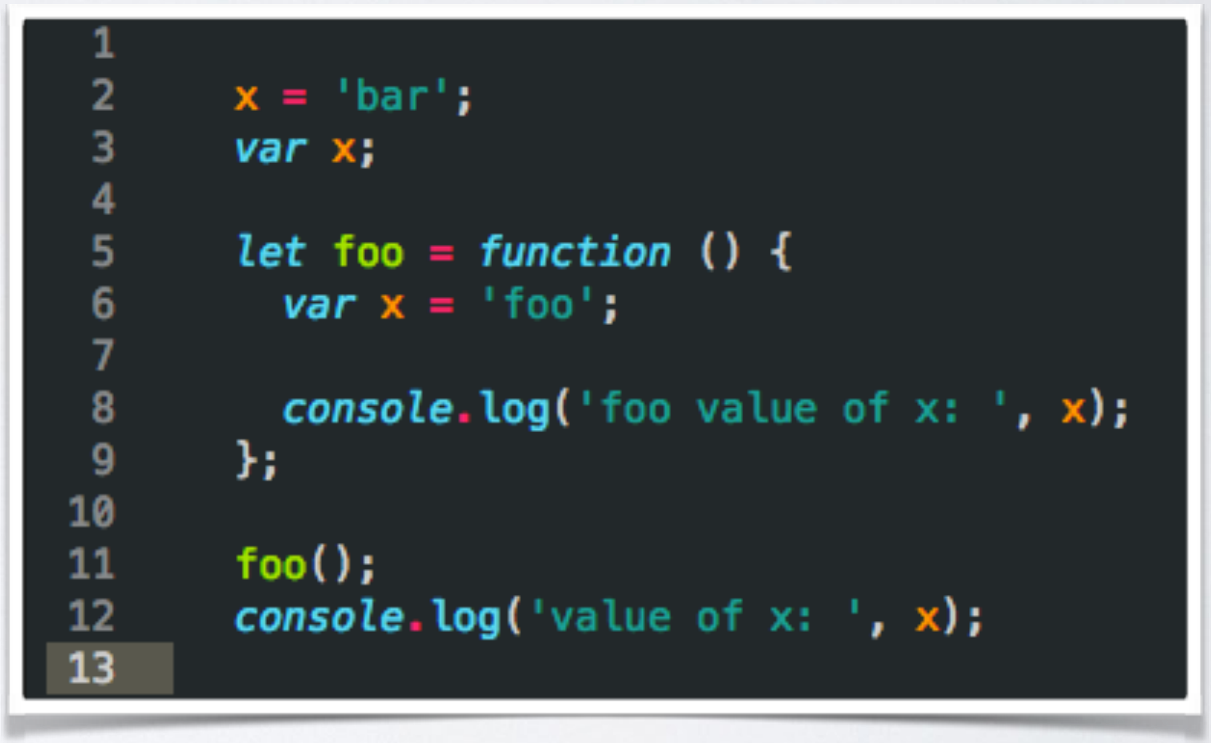


```
    const x = 'bar';
    const y = { foo: 'bar' };

    let foo = function () {
      const x = 'foo';

      console.log('foo value of x: ', x);
    };

    console.log('value of x: ', x);
    console.log('value of y: ', y);

    // y = { foo : 'foo bar'}; // Fails!
    y.foo = 'foo bar';
    foo();
    console.log('value of y: ', y);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const

VAR

- declares variable
- scope limited to execution context: enclosing function or global scope
- var hoisting preprocesses variable declaration

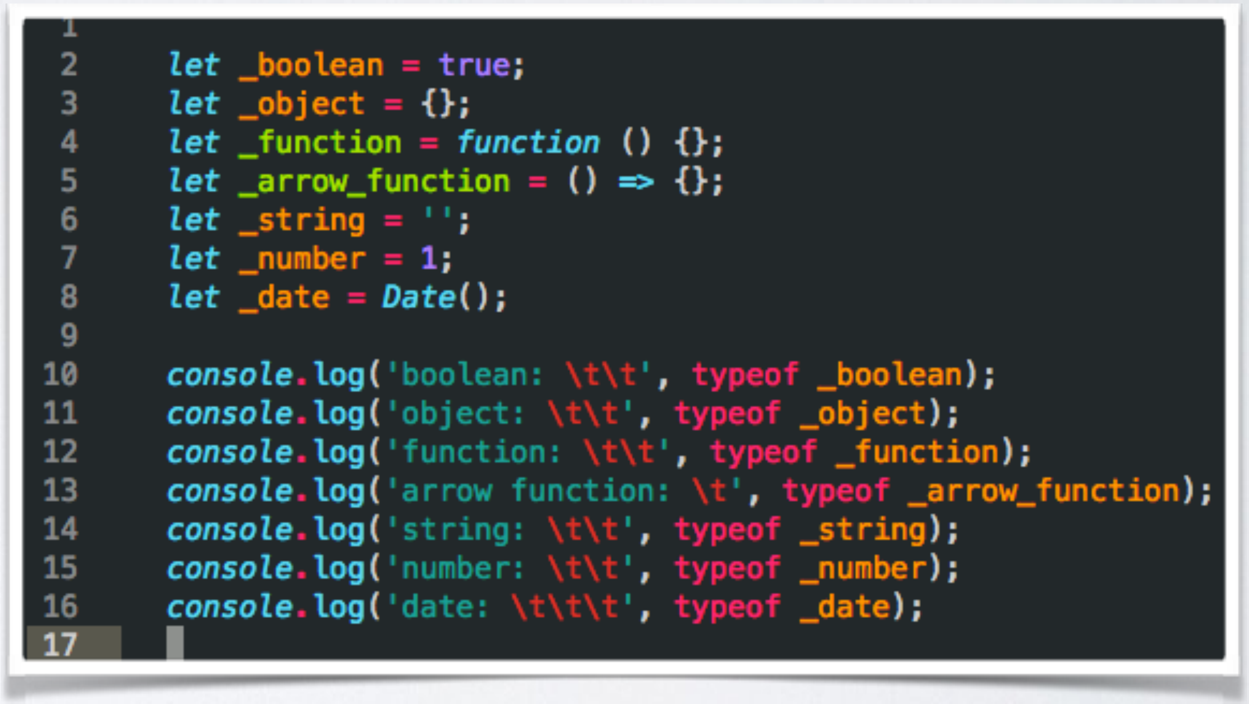


```

x = 'bar';
var x;

let foo = function () {
  var x = 'foo';

  console.log('foo value of x: ', x);
};

foo();
console.log('value of x: ', x);

```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

TYPES

- inferred from declaration
- boolean, object, function, string, number, date, null and several more

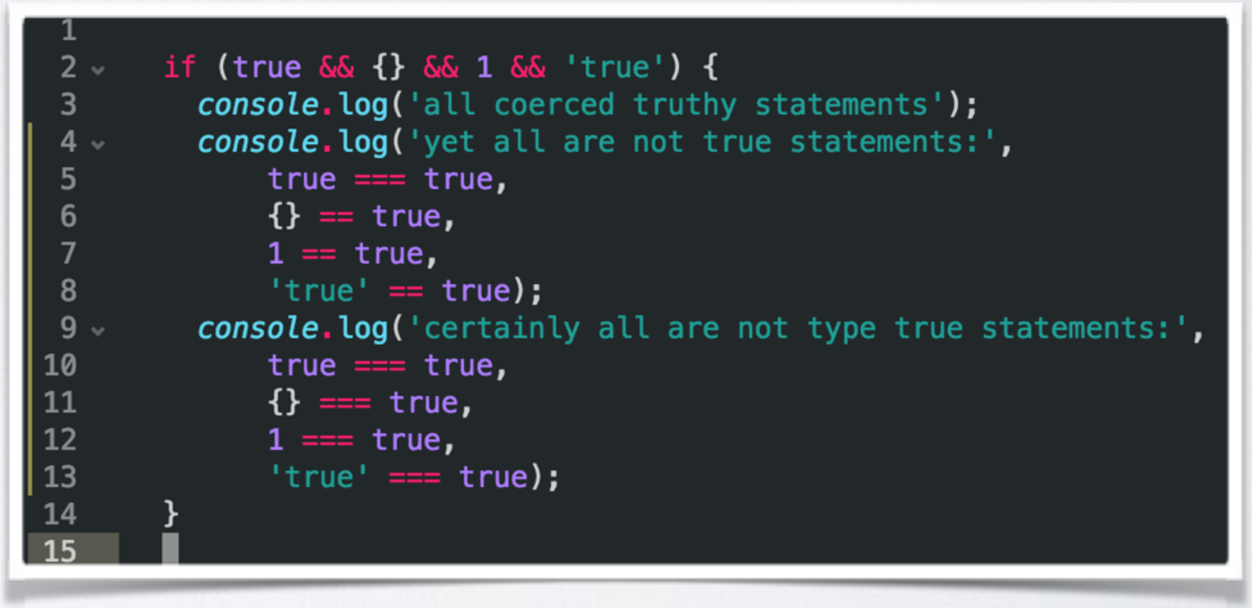


```
let _boolean = true;
let _object = {};
let _function = function () {};
let _arrow_function = () => {};
let _string = '';
let _number = 1;
let _date = Date();

console.log('boolean: \t\t', typeof _boolean);
console.log('object: \t\t', typeof _object);
console.log('function: \t\t', typeof _function);
console.log('arrow function: \t', typeof _arrow_function);
console.log('string: \t\t', typeof _string);
console.log('number: \t\t', typeof _number);
console.log('date: \t\t\t', typeof _date);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

COERCION

- falsy vs. truthy
- all values are considered true in a boolean context except:
  - `false, 0, '',  null, undefined, and NaN`

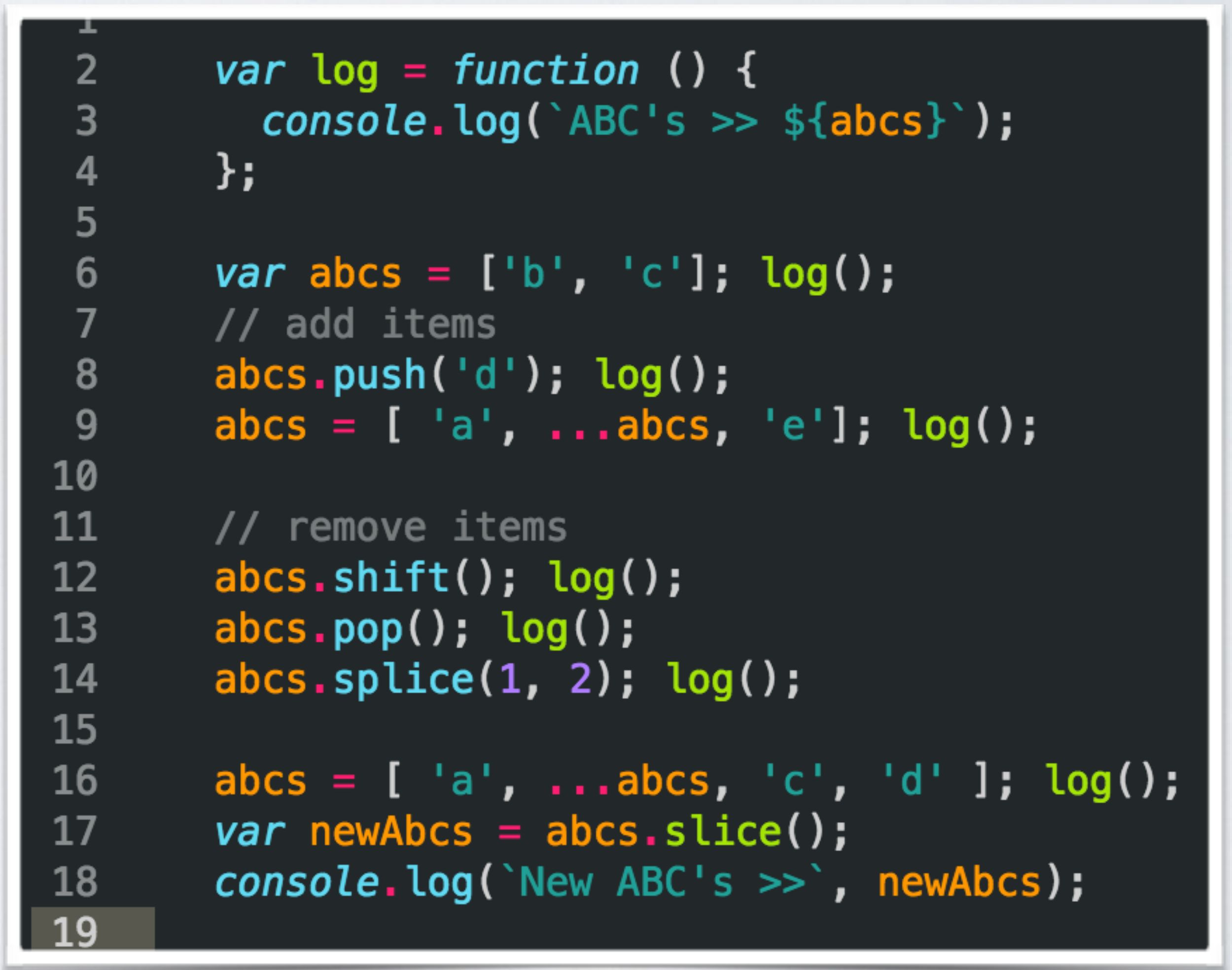


```
if (true && {} && 1 && 'true') {
  console.log('all coerced truthy statements');
  console.log('yet all are not true statements:',
      true === true,
      {} == true,
      1 == true,
      'true' == true);
  console.log('certainly all are not type true statements:',
      true === true,
      {} === true,
      1 === true,
      'true' === true);
}
```

ARRAYS

BASICS

- list-like objects
- add/remove items
- spread syntax (new)
- copy (slice)

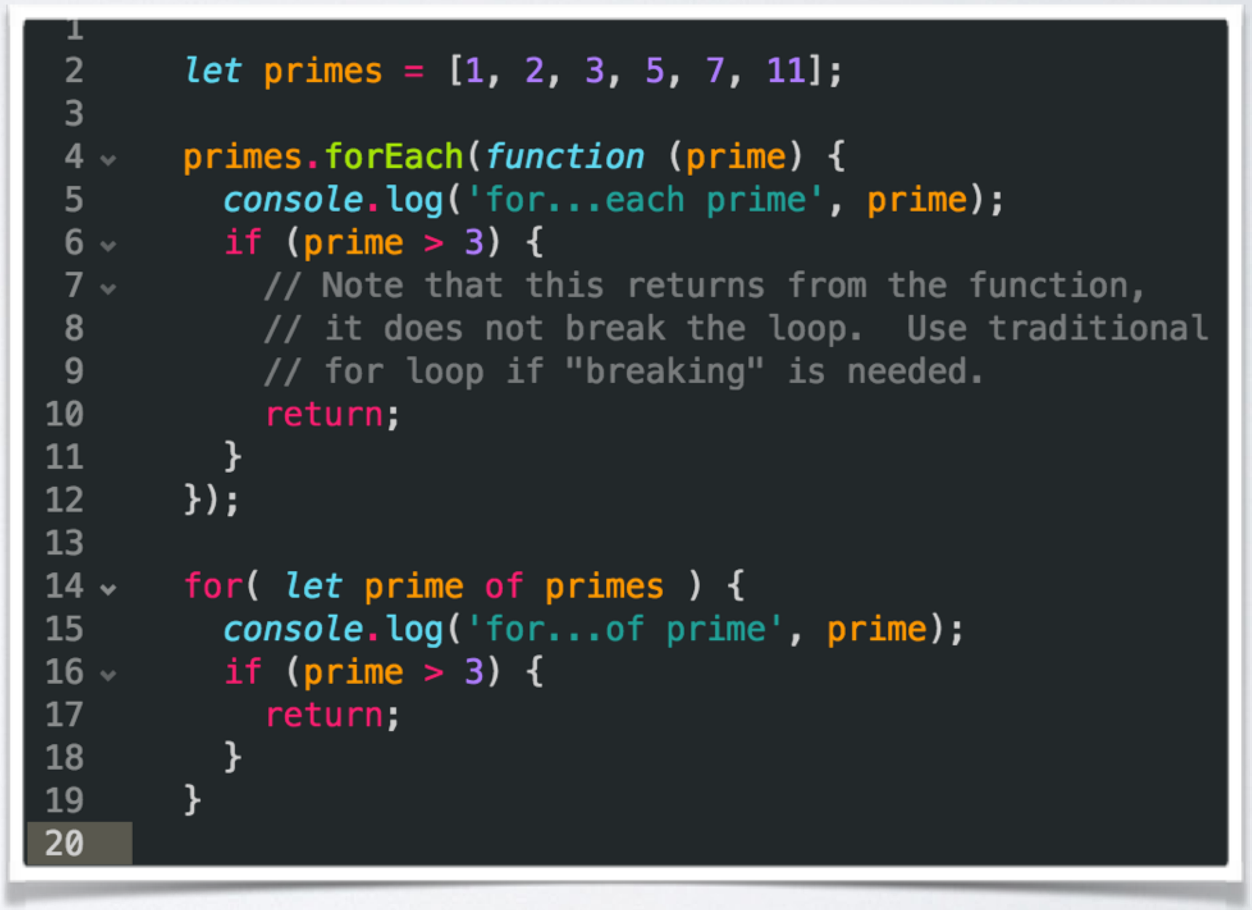


```
var log = function () {
  console.log(`ABC's >> ${abcs}`);
};

var abcs = ['b', 'c']; log();
// add items
abcs.push('d'); log();
abcs = [ 'a', ...abcs, 'e']; log();

// remove items
abcs.shift(); log();
abcs.pop(); log();
abcs.splice(1, 2); log();

abcs = [ 'a', ...abcs, 'c', 'd' ]; log();
var newAbcs = abcs.slice();
console.log(`New ABC's >>`, newAbcs);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

LOOPING

- for…each
- for…of

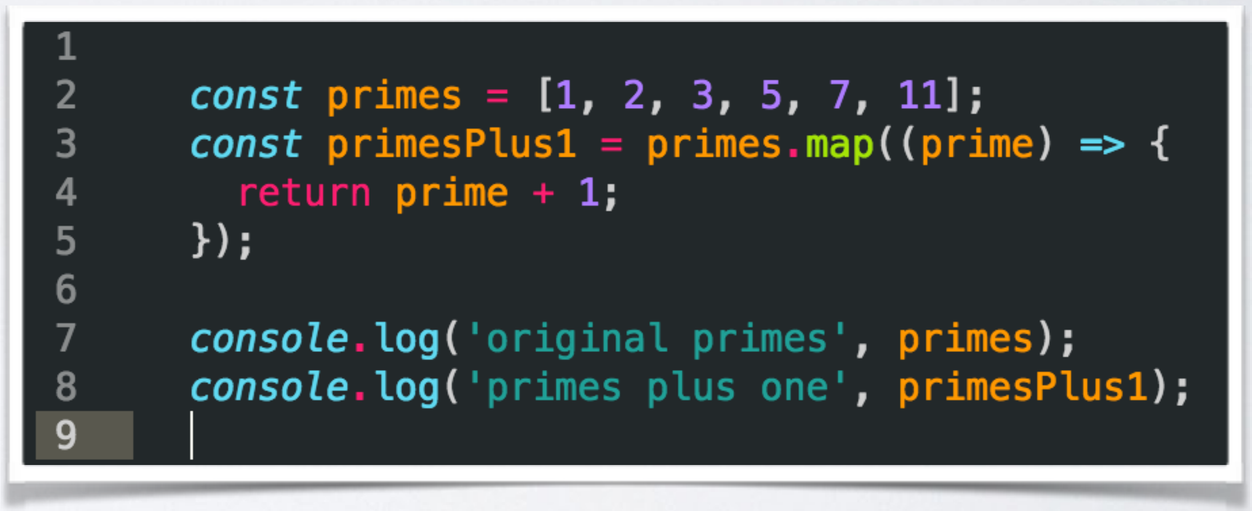


```
let primes = [1, 2, 3, 5, 7, 11];

primes.forEach(function (prime) {
  console.log('for...each prime', prime);
  if (prime > 3) {
    // Note that this returns from the function,
    // it does not break the loop.  Use traditional
    // for loop if "breaking" is needed.
    return;
  }
});

for( let prime of primes ) {
  console.log('for...of prime', prime);
  if (prime > 3) {
    return;
  }
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements

MAPPING

- maps (transforms) each value of an array into a new value
- functional paradigm



```
const primes = [1, 2, 3, 5, 7, 11];
const primesPlus1 = primes.map((prime) => {
  return prime + 1;
});

console.log('original primes', primes);
console.log('primes plus one', primesPlus1);
```

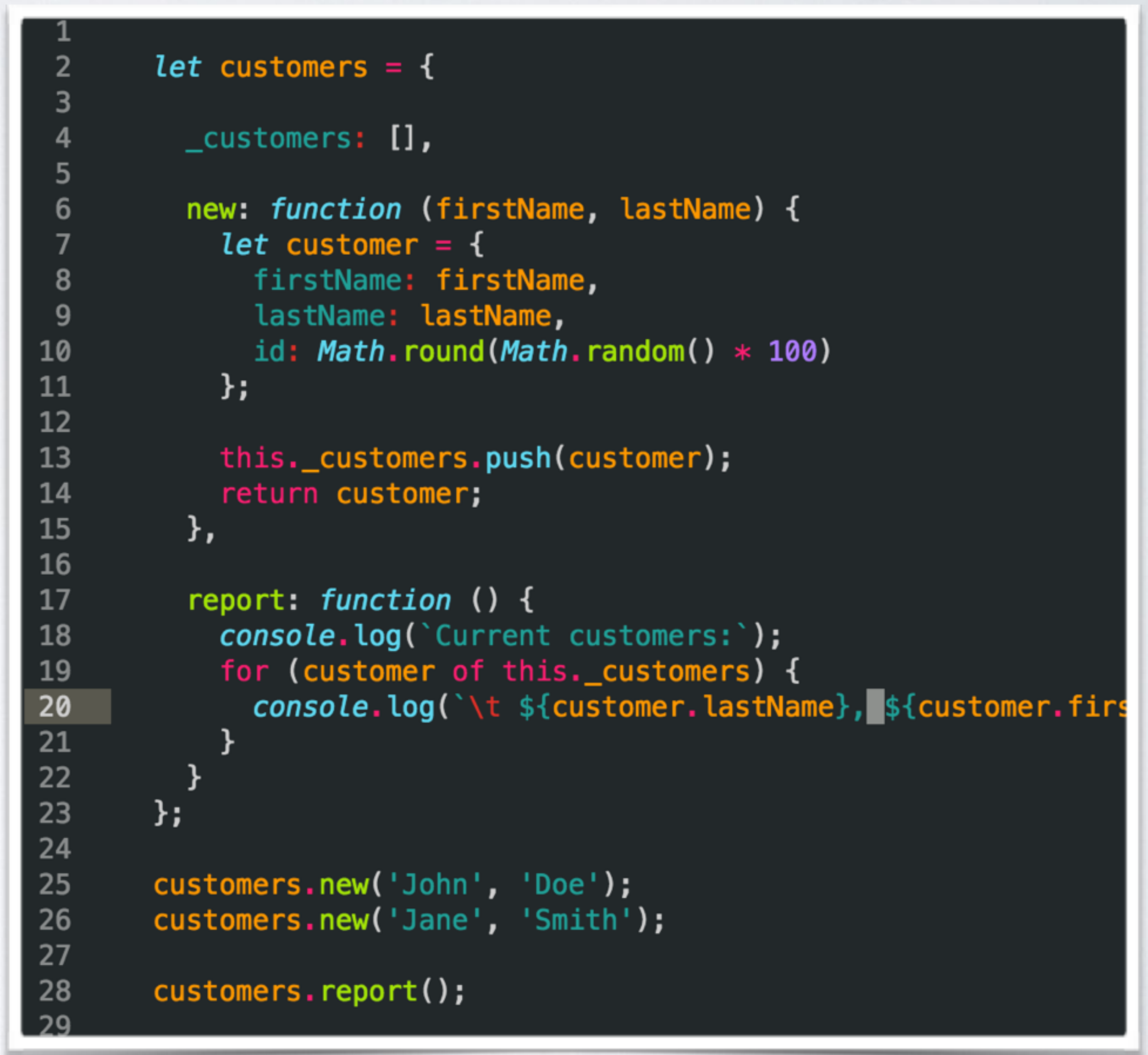
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements

OBJECTS

- key/value pairs
- keys must follow **`var`** naming standards
- value can be any ECMAScript datatype

BASICS

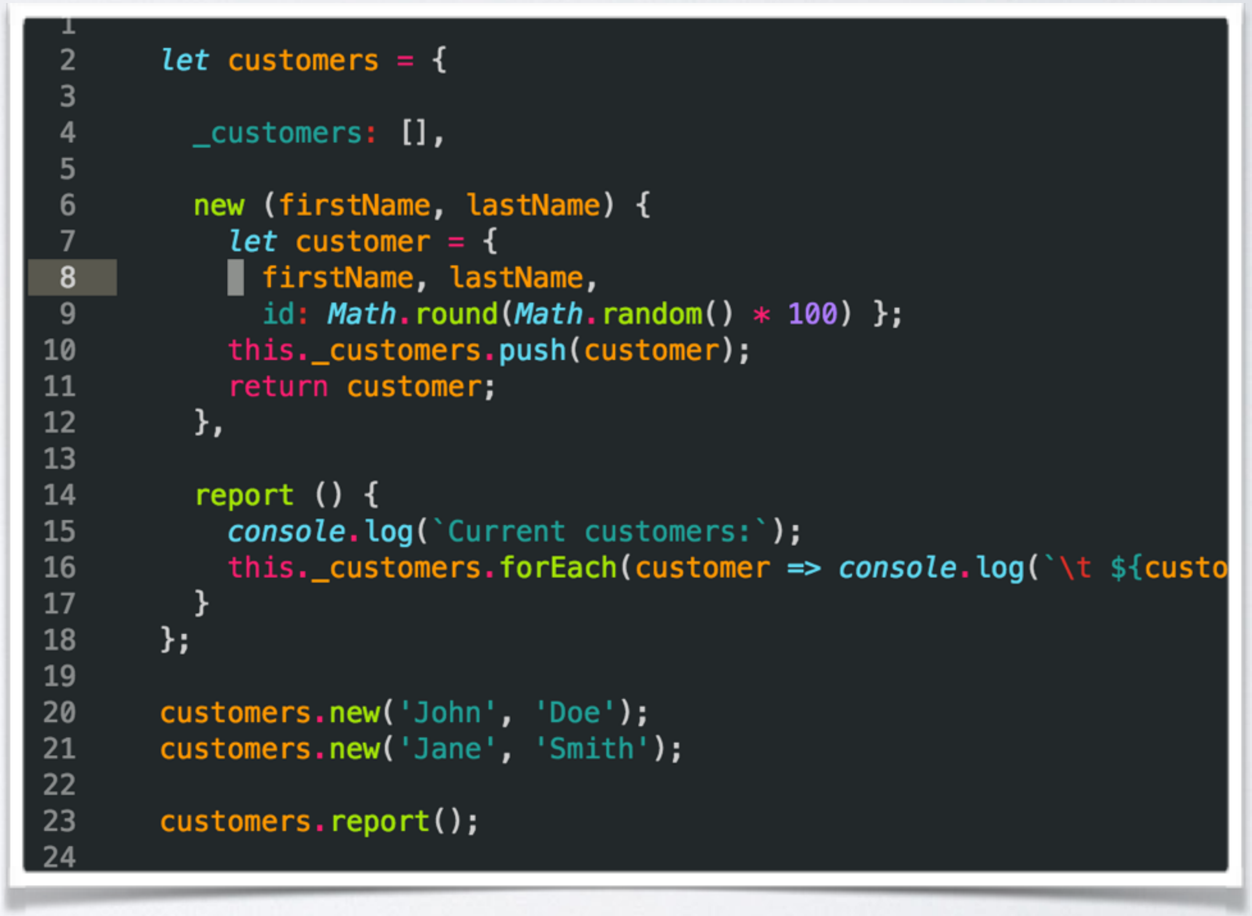
- consider an object holding the state of your customers
- '_' is a generally accepted indicator in ECMAScript for 'private' variables.



```

let customers = {

  _customers: [],

  new: function (firstName, lastName) {
    let customer = {
      firstName: firstName,
      lastName: lastName,
      id: Math.round(Math.random() * 100)
    };

    this._customers.push(customer);
    return customer;
  },

  report: function () {
    console.log(`Current customers:`);
    for (customer of this._customers) {
      console.log(`\t ${customer.lastName}, ${customer.firs
    }
  }
};

customers.new('John', 'Doe');
customers.new('Jane', 'Smith');

customers.report();

```

BASICS ES-6

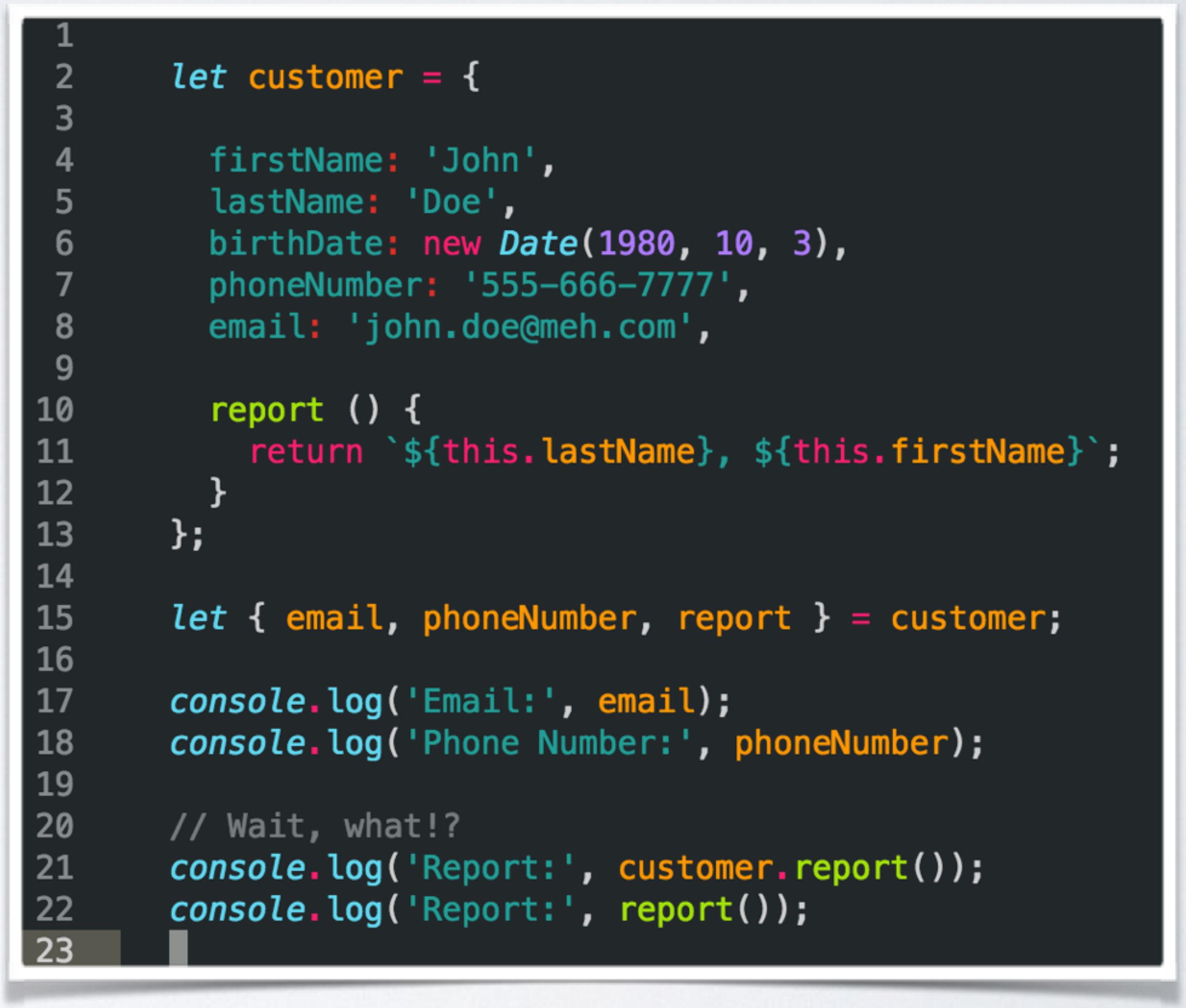
- implicit property definitions
- report function streamlined using for…each



```
let customers = {

  _customers: [],

  new (firstName, lastName) {
    let customer = {
      firstName, lastName,
      id: Math.round(Math.random() * 100) };
    this._customers.push(customer);
    return customer;
  },

  report () {
    console.log(`Current customers:`);
    this._customers.forEach(customer => console.log(`\t ${custo
  }
};

customers.new('John', 'Doe');
customers.new('Jane', 'Smith');

customers.report();
```

DESTRUCTURING ASSIGNMENT

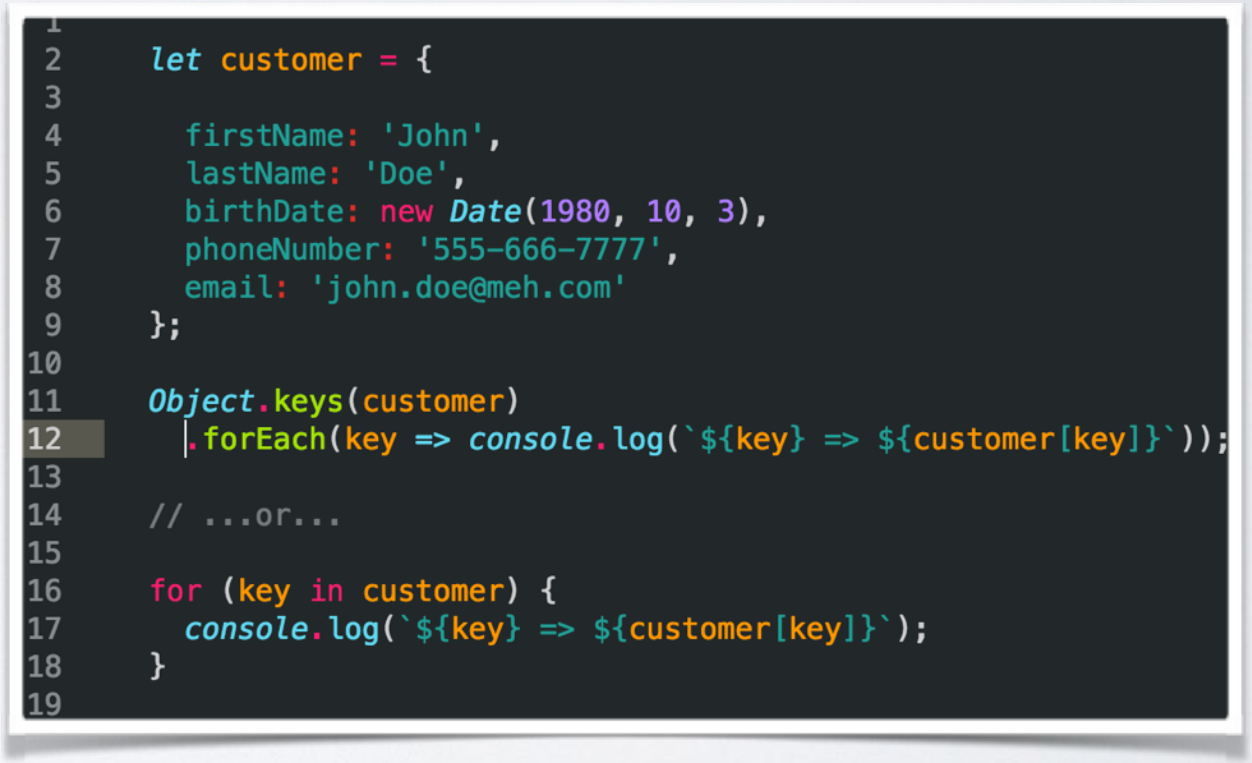
- flexible assignment of variables
- can assign functions…but



```javascript

let customer = {

  firstName: 'John',
  lastName: 'Doe',
  birthDate: new Date(1980, 10, 3),
  phoneNumber: '555-666-7777',
  email: 'john.doe@meh.com',

  report () {
    return `${this.lastName}, ${this.firstName}`;
  }
};

let { email, phoneNumber, report } = customer;

console.log('Email:', email);
console.log('Phone Number:', phoneNumber);

// Wait, what!?
console.log('Report:', customer.report());
console.log('Report:', report());

```

LOOPING

- occasionally you may want to process the properties of an object
  - `Object.keys()`
  - `for…in`
- order is arbitrary!



```javascript
let customer = {

  firstName: 'John',
  lastName: 'Doe',
  birthDate: new Date(1980, 10, 3),
  phoneNumber: '555-666-7777',
  email: 'john.doe@meh.com'
};

Object.keys(customer)
  .forEach(key => console.log(`${key} => ${customer[key]}`));

// ...or...

for (key in customer) {
  console.log(`${key} => ${customer[key]}`);
}
```

PROMISES

an object that represents a value which may be available now, or in the future, or never

STAGES

- *pending* - initial state, not fulfilled or rejected
- *fulfilled* - state meaning the operation completed successfully
- *rejected* - state meaning the operation has failed

HANDLERS

- *then* - appends fulfillment and rejection handlers to the promise, and returns a new promise resolving to the return value of the called handler, or to its original settled value if the promise was not handled

- *catch* - appends a rejection handler callback to the promise, and returns a new promise resolving to the return value of the callback if it is called, or to its original fulfillment value if the promise is instead fulfilled

REAL WORLD

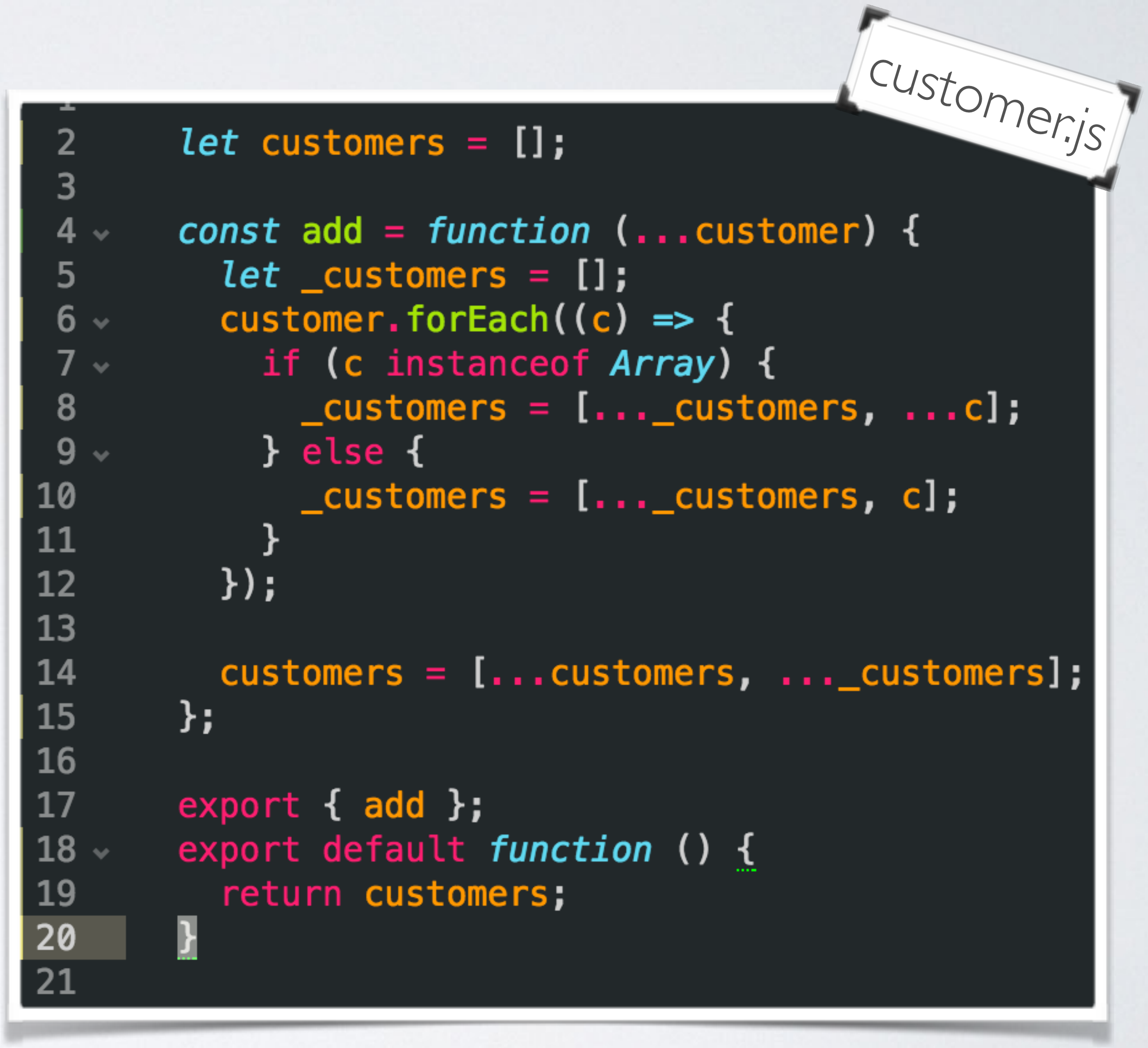
- initialization of the Promise is typically handled by some other library
- as a developer, you typically only care about what to do when the promise is fulfilled (`then`) or rejected (`catch`)

MODULES

- provide the backbone of modern ECMAScript applications
- 1st class support in Node and modern browsers
- promotes a ‘revealing’ pattern

EXPORT

- exports functions, objects and properties from a given file
- named exports
- default export

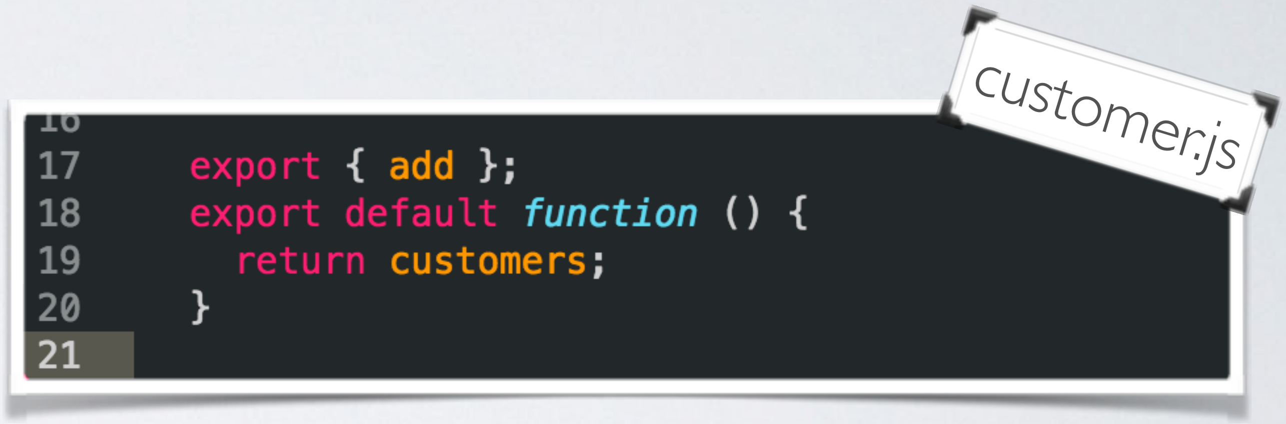


```javascript

let customers = [];

const add = function (...customer) {
  let _customers = [];
  customer.forEach((c) => {
    if (c instanceof Array) {
      _customers = [..._customers, ...c];
    } else {
      _customers = [..._customers, c];
    }
  });

  customers = [...customers, ..._customers];
};

export { add };
export default function () {
  return customers;
}

```

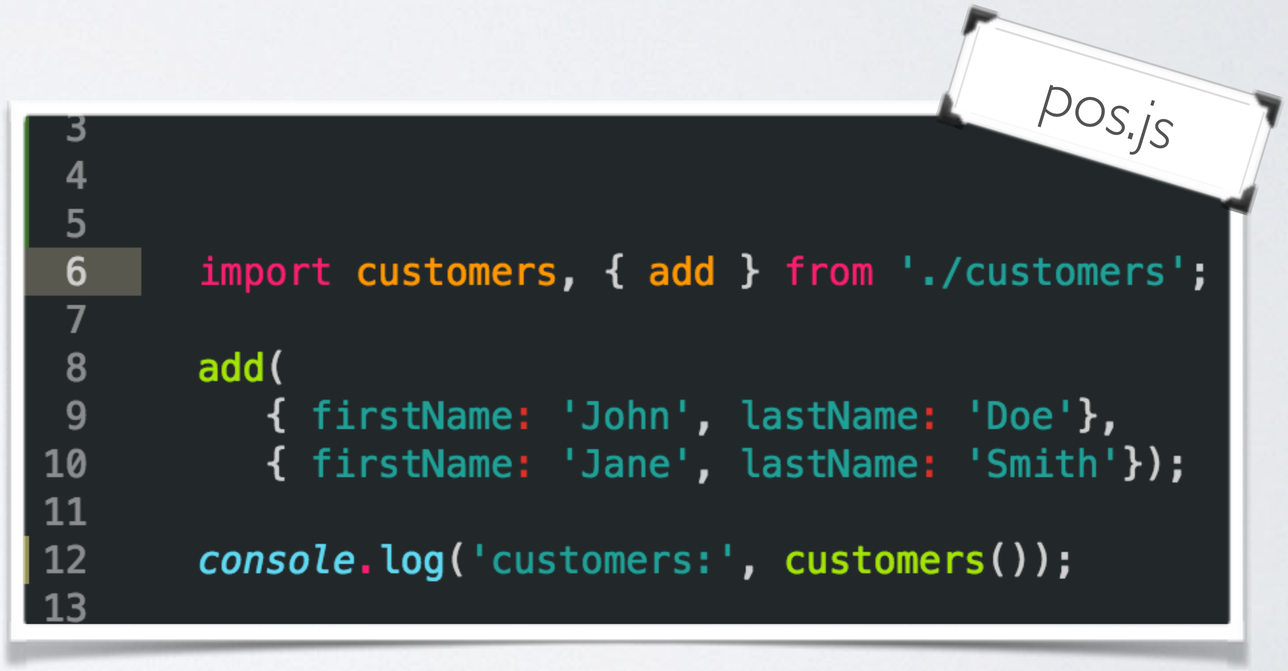
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export

IMPORT

- imports functions, objects and properties exported from another file
- named imports must match export name
- default import name is arbitrary (but typically matches module name)



```
    export { add };
    export default function () {
      return customers;
    }
```



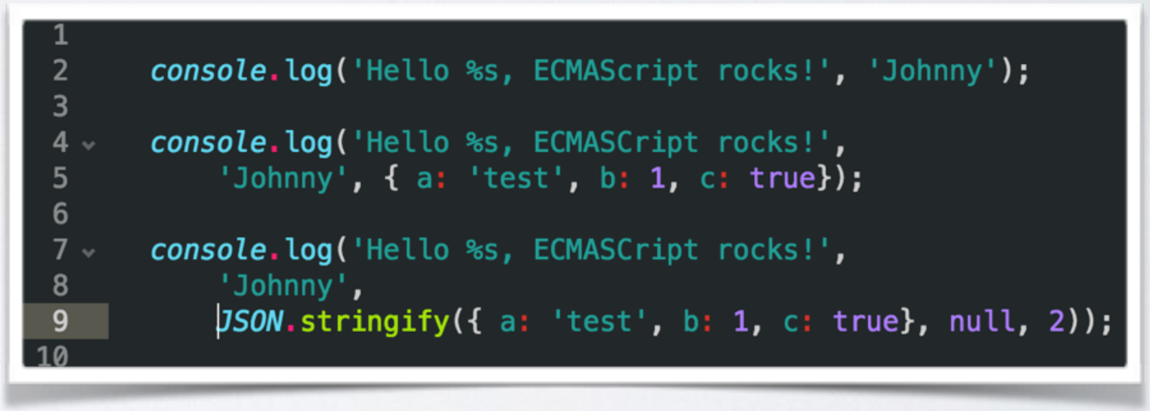
```


    import customers, { add } from './customers';

    add(
        { firstName: 'John', lastName: 'Doe'},
        { firstName: 'Jane', lastName: 'Smith'});

    console.log('customers:', customers());

```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

DEBUGGING

CONSOLE.LOG

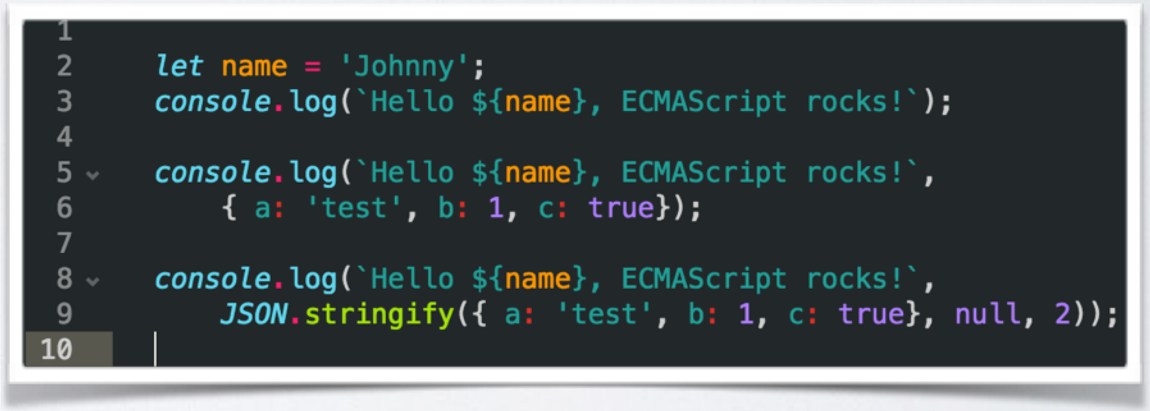
- you've seen this throughout this presentation
- not just another console logger
- group statements with console.group() and console.groupEnd()
- use with JSON.stringify() for well formatted objects



```
console.log('Hello %s, ECMAScript rocks!', 'Johnny');

console.log('Hello %s, ECMAScript rocks!',
    'Johnny', { a: 'test', b: 1, c: true});

console.log('Hello %s, ECMAScript rocks!',
    'Johnny',
    JSON.stringify({ a: 'test', b: 1, c: true}, null, 2));
```

STRING INTERPOLATION

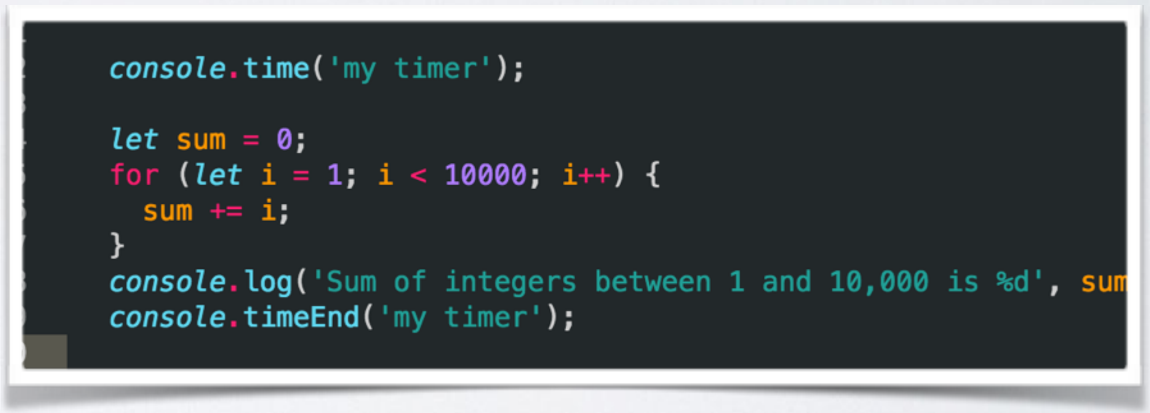
- while `console.log()` support string formatting, consider string interpolation instead



```
let name = 'Johnny';
console.log(`Hello ${name}, ECMAScript rocks!`);

console.log(`Hello ${name}, ECMAScript rocks!`,
    { a: 'test', b: 1, c: true});

console.log(`Hello ${name}, ECMAScript rocks!`,
    JSON.stringify({ a: 'test', b: 1, c: true}, null, 2));
```

TIMING

- starts and ends a timer with a timer name



```
console.time('my timer');

let sum = 0;
for (let i = 1; i < 10000; i++) {
  sum += i;
}
console.log('Sum of integers between 1 and 10,000 is %d', sum
console.timeEnd('my timer');
```