ECMAScript 2015

Not Your Grandpa's JavaScript

Court Ewing

Engineer @ Elastic

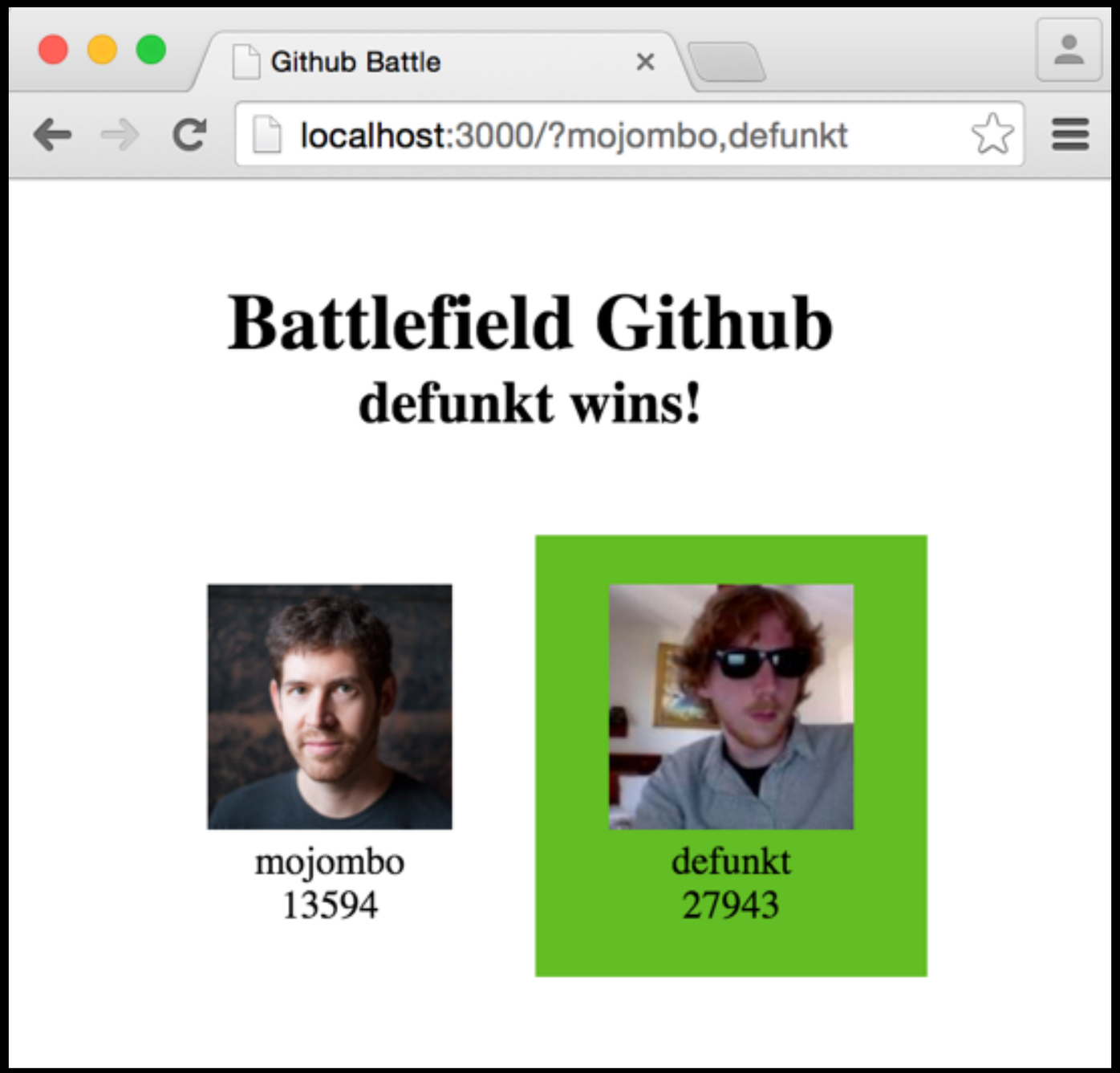
Organizer @ Pub Standards
@ Hack Lancaster

Volunteer @ Coder Dojo

es5 -> es2015

sep 2009 june 2015

Battlefield Github



Battlefield Github

get-json.js	Makes ajax calls
person.js	Represents one github user
battle.js	Determines winner of two people
render.js	Renders results to HTML
index.js	Ties it all together

get-json.js

```
import $ from './jquery';

export default function getJSON(url) {
  return new Promise(function(success, error) {
    $.getJSON(url, { success, error });
  });
}
```

Modules (import / export)

Promises

Short-property syntax

Modules

```
// my-functions.js
export function one() {
  // ...
}
export function two() {
  // ...
}

// some-other-file.js
import { one, two } from './my-functions';
one();
two();
```

Promise

```
var wait = new Promise(function(success) {
  setTimeout(function() {
    success();
  }, 5000);
});

wait.then(function() {
  console.log('5 seconds have passed');
});
```

Short property syntax

```
var foo = 'bar';
var no  = 'wai';

// es5
var myObject = { foo: foo, no: no };

// es2015
var myObject = { foo, no };

// both
myObject.foo; // bar
myObject.no;  // wai
```

Battlefield Github

~~get-json.js~~	~~Makes ajax calls~~
person.js	Represents one github user
battle.js	Determines winner of two people
render.js	Renders results to HTML
index.js	Ties it all together

person.js

```
export class Person {
  constructor(name) { /* ... */ }

  static endpoint(path) { /* ... */ }

  get url() { /* ... */ }

  rank() { /* ... */ }
}
```

Classes:

constructor, static, getter, method

Class (constructor)

```
class Person {
  constructor(name) {
    this.name = name;
    this.repos = [];
  }
}

var person = new Person('Court');
person.name; // Court
```

Class (static)

```
class Person {
  static endpoint(path) {
    return 'https://github.com' + path;
  }
}

Person.endpoint('/epixa'); // https://github.com/epixa
```

Class (getter)

```
class Person {
  get url() {
    return Person.endpoint('/' + this.name);
  }
}

var person = new Person('epixa');
person.url; // https://github.com/epixa
```

Class (method)

```
class Person {
  rank() {
    return this.repos.reduce(function(prev, current) {
      current.watchers + prev
    }, 0);
  }
}

var person = new Person('epixa');
person.rank(); // 16
```

Battlefield Github

~~get-json.js~~	~~Makes ajax calls~~
~~person.js~~	~~Represents one github user~~
battle.js	Determines winner of two people
render.js	Renders results to HTML
index.js	Ties it all together

battle.js

```
export function battle(a, b) {
  var loadUsers = [ a.load(), b.load() ];

  return Promise.all(loadUsers)
    .then(function(a, b) {
      if (a.ties(b)) {
        return null;
      }
      return a.beats(b) ? a : b;
    });
}
```

Promise.all

Promise.all

```
function waitForRandom() {
  var ms = Math.round(Math.random() * 1000);
  return new Promise(function(success) {
    setTimeout(function() {
      success();
    }, ms);
  });
}

Promise.all([ waitForRandom(), waitForRandom() ])
  .then(function() {
    console.log("Waited for both random timeouts");
  });
```

Battlefield Github

~~get-json.js~~	~~Makes ajax calls~~
~~person.js~~	~~Represents one github user~~
~~battle.js~~	~~Determines winner of two people~~
render.js	Renders results to HTML
index.js	Ties it all together

Battlefield Github

~~get-json.js~~	~~Makes ajax calls~~
~~person.js~~	~~Represents one github user~~
~~battle.js~~	~~Determines winner of two people~~
~~render.js~~	~~Renders results to HTML~~
index.js	Ties it all together

index.js

```
import { Person } from './person';
import { render } from './render';
import { battle } from './battle';

const people = location.search
  .substr(1).split(',')
  .map(name => new Person(name));

const [ personA, personB ] = people;

battle(personA, personB)
  .then(winner => render(personA, personB, winner));
```

Const, arrow functions, destructuring

const, why?

```
// the problem:
var index = [];

for (var index = 0; index < 100; index++) {
  index; // 0, 1, 2, 3, ...
}

index; // 99, whoops!
```

const

```
// const will prevent overrides
const index = [];

for (var index = 0; index < 100; index++) {
  // ????
}
```

let

```
const index = [];

for (let index = 0; index < 100; index++) {
  index; // 0, 1, 2, 3, ...
}

index; // [], huzah!
```

Arrow functions, why?

```
const person = {
  age: 20,
  agesOfFriends: [ 40, 30, 20 ],
  averageDifference() {
    const totalDiff = this.agesOfFriends
      .map(function(age) {
        return age - this.age; // wrong scope!
      })
      .reduce(function(a, b) {
        return a + b;
      }, 0);

    return totalDiff - this.agesOfFriends.length;
  }
}
```

var self = this :(

```
const person = {
  age: 20,
  agesOfFriends: [ 40, 30, 20 ],
  averageDifference() {
    const self = this; // blegh, gross.

    const totalDiff = this.agesOfFriends
      .map(function(age) {
        return age - self.age; // it works, at least
      })
      .reduce(function(a, b) {
        return a + b;
      }, 0);

    return totalDiff - this.agesOfFriends.length;
  }
}
```

Arrow functions FTW

```
const person = {
  age: 20,
  agesOfFriends: [ 40, 30, 20 ],
  averageDifference() {
    const totalDiff = this.agesOfFriends
      .map(age => age - this.age) // works!
      .reduce((a, b) => a + b, 0);

    return totalDiff - this.agesOfFriends.length;
  }
}
```

Destructuring

```
// for accessing elements of an array:

const myArr = [ 'bob', 'sue' ];
let bob, sue;

// the old way:
bob = myArr[0];
sue = myArr[1];

// the destructuring way:
[ bob, sue ] = myArr;
```

Destructuring

```
// for accessing properties of an object:

const myObj = { foo: 'bar', no: 'wai' };
let foo, no;

// the old way:
foo = myObj.foo;
no  = myObj.no;

// the destructuring way:
{ foo, no } = myObj;
```

Destructuring

```
// or how about some context?
// a common way to pass params to a function:
myFunc({ foo: 'bar', no: 'wai' });

// the old way:
function myFunc(params) {
  params.foo; // bar
  params.no;  // wai
}

// the destructuring way:
function myFunc({ foo, no }) {
  foo; // bar
  no;  // wai
}
```

Battlefield Github

~~get-json.js~~	~~Makes ajax calls~~
~~person.js~~	~~Represents one github user~~
~~battle.js~~	~~Determines winner of two people~~
~~render.js~~	~~Renders results to HTML~~
~~index.js~~	~~Ties it all together~~

Features we covered

Classes
Modules
Arrow functions
Destructuring
Const + Let
Promises
Short property syntax

And Features We Didn't...

Generators
Spread operator
Optional parameters
Unicode
Map + Set
Reflection
Tail calls

Template strings
Iterators
Symbols
Proxies
Extensible native objects
Math/Number enhancements
Binary and Octal literals

Compatibility

Edge	80%
Firefox	72%
Chrome	63%
Node	53%
IE11	20%

Babel

71% of the time,
it works every time

github.com/epixa/cposc2015