Design and Implementation of Online Experiments

nodeGame.org

Stefano Balietti

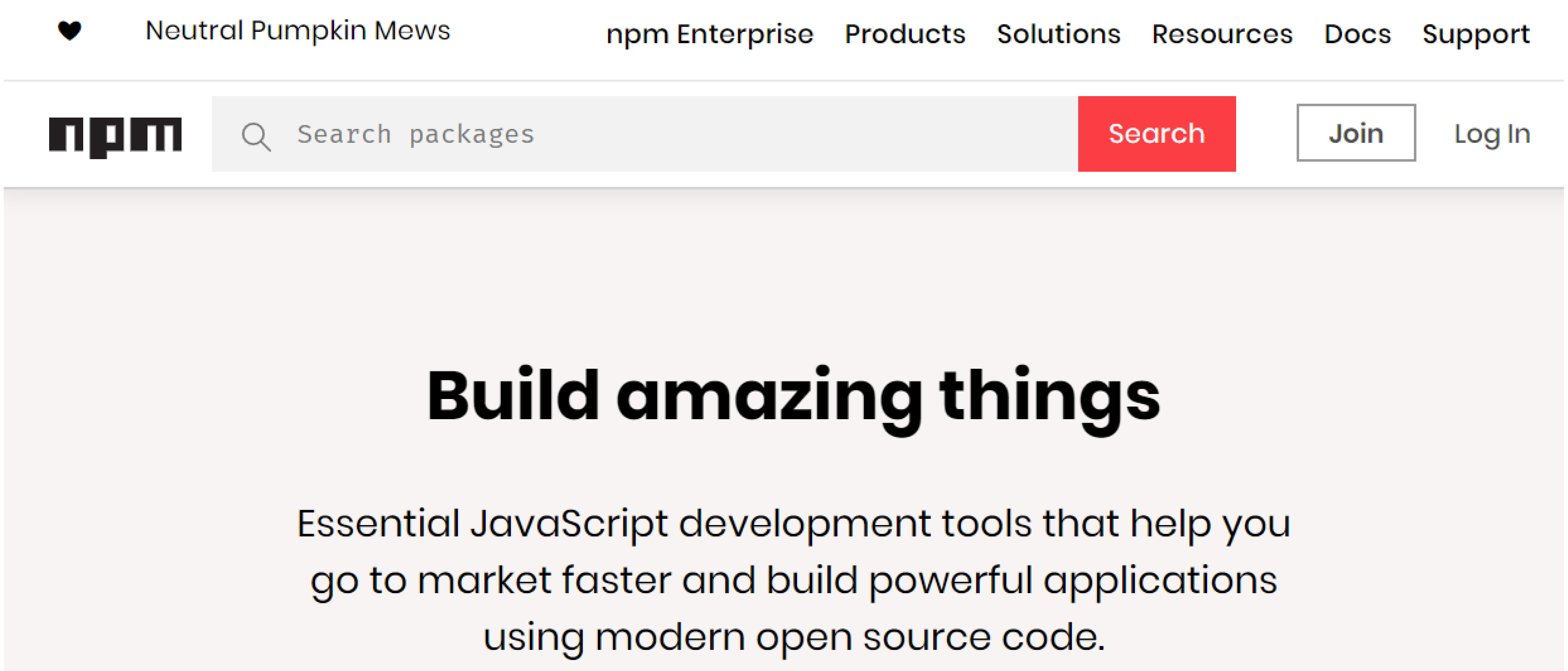
MZES and Heidelberg

Some Extra Concepts in JavaScript

@balietti
@nodegameorg
stefanobalietti.com@gmail.com

NPM: Node Package Manager

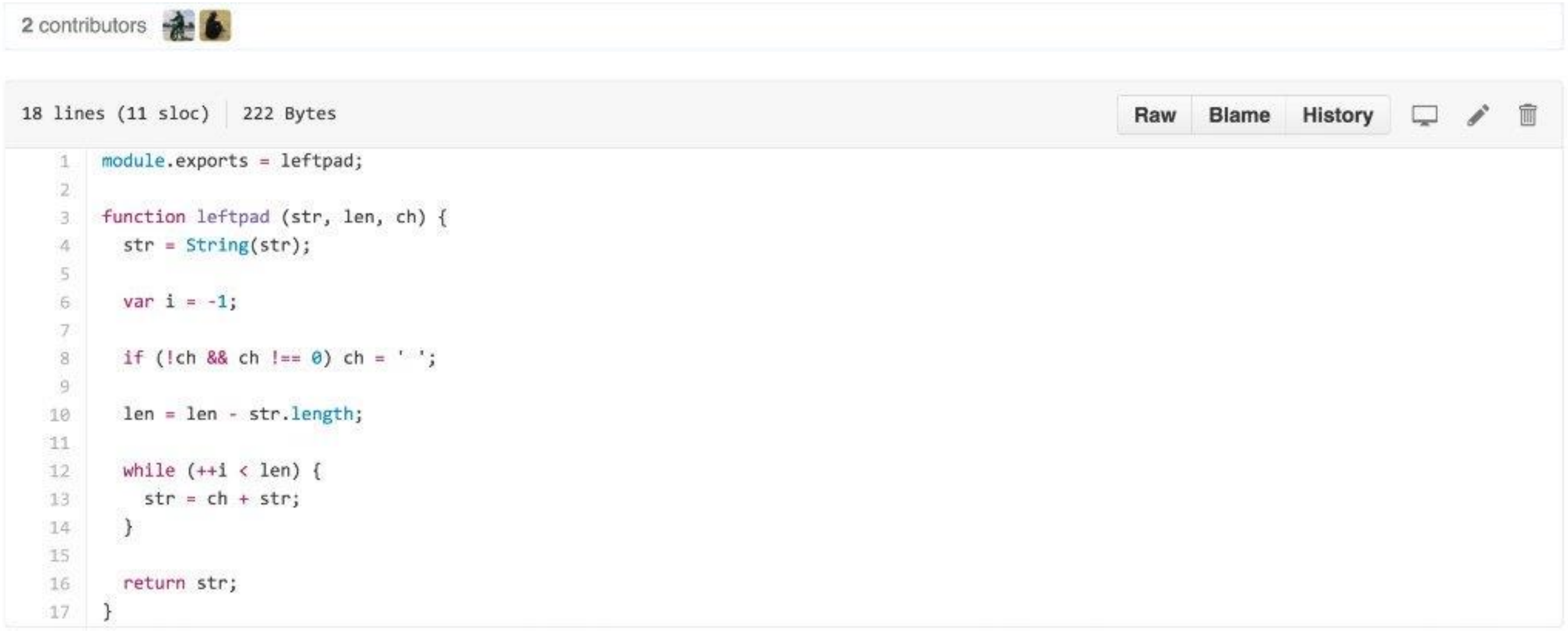
https://www.npmjs.com/



Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

The 11 Lines that Almost Broke the Internet



18 lines (11 sloc) | 222 Bytes

```
module.exports = leftpad;

function leftpad (str, len, ch) {
  str = String(str);

  var i = -1;

  if (!ch && ch !== 0) ch = ' ';

  len = len - str.length;

  while (++i < len) {
    str = ch + str;
  }

  return str;
}
```

https://www.sciencealert.com/how-a-programmer-almost-broke-the-internet-by-deleting-11-lines-of-code

NPM: Node Package Manager

```
npm install one-liner-joke
```

- Creates a node_modules/ folder inside the same directory.
- It contains the requested module and all its dependencies.
- We can now require it and use it in our programs.

```
const joker = require('one-liner-joker');

var randomJoke = joker.getRandomJoke();
console.log(randomJoke);
```

JS Functions

- JS functions are *objects*
- Can be passed as parameters to other functions
- Treat differently different input parameters
- Can have properties
- Can be executed with different contexts
- Two types exists: *declaration* and *expressions*
- Always remember the context of creation

JS Functions

Create an array of 10 functions returning the index in which they are inserted in the array.

Is the code below correct?

```
var i, len, obj;
len = 10, obj = [];

for (i = 0 ; i < len ; i++) {
  obj[i] = function() { return i; }
}
```

JS Functions

Create an array of 10 functions returning the index in which they are inserted in the array.

❓ Is the code below correct?

```
var i, len, obj;
len = 10, obj = [];

for (i = 0 ; i < len ; i++) {
   obj[i] = function() { return i; }
}
```

- **Creates 10 functions all returning the value 10, because they are all referencing variable i, which has value 10 when the loop ends**

JS Functions

We need a closure and a self-executing anonymous function!

```
for (i = 0 ; i < len ; i++) {
    obj[i] = (function(i) {
        return function() {
            return i;
        }
    })(i);
}
```

JS this

- The value of *this* is dynamic in JavaScript
- It is determined when function is *called*, not when it is declared.

```
function a() { return this.a; }
a(); // undefined

// Create a context.
var foo = { a: 1};

// call and apply set the this value
a.call(foo, 1, 2, 3); // 1;
a.apply(foo, [1, 2, 3]); // 1;
```

JS this

What will the following code print to console?

```
function A() {
    this.a = 1;
    (function() {
        console.log(this.a);
    })();
}
// Create a new object.
var a = new A();
```

JS this

What will the following code print to console?

```
function A() {
    this.a = 1;
    (function() {
        console.log(this.a);
    })();
}
// Create a new object.
var a = new A();
```

- **It will print *undefined***
- **How to adapt to print 1?**

JS this

What will the following code print to console?

```
function A() {
    this.a = 1;
    var that = this;
    (function() {
        console.log(that.a);
    })();
}
```

- **It will print *1***
- **The reference to `this` is stored in another variable**

JS this

Why is “this” solution better than using call or apply ?

```
function A() {
    this.a = 1;
    var that = this;
    (function() {
        console.log(that.a);
    })();
}
```

- **It will print *1***
- **The reference to `this` is stored in another variable**

JS this

Why is “this” solution better than using call or apply ?

```
function A() {
     this.a = 1;
     var that = this;
     (function() {
          console.log(that.a);
     })();
}
```

- **It will print *1***
- **The reference to `this` is stored in another variable**

Because you can reuse *that* multiple times!

JS this and Arrow function

However, ES6 has introduced the arrow function that accomplish the same goal without the need to introduce a new variable.

```
function A() {
    this.a = 1;
    (() => {
        console.log(this.a);
    })();
}
```

JS Inheritance

- In JS, each object inherits methods and properties from a parent object called **prototype**

- In turn, also the prototype object can have an own prototype, and all the properties are are inherited through the **prototype chain**

- It is possible to extend an object by extending its prototype or the prototype of its prototype...

- This pattern is called **prototypical inheritance**, and it is extremely powerful–if well understood

JS Prototypical Inheritance

```
function A() {
    this.a = 1;
}
A.prototype.printA = function() {
    console.log(this.a);
}
var a = new A();
a.printA(); // 1;
```

JS Prototypical Inheritance

```
function A() {
    this.a = 1;
}
A.prototype.printA = function() {
    console.log(this.a);
}
var a = new A();
a.printA(); // 1;
```

❓ **What is the difference with defining define the method *printA* inside the constructor? (this.printA = function …)**

JS Prototypical Inheritance

```
// Create a second object.
var a2 = new A();

// Assign property to method printA.
a2.printA.foo = 1;

// Property is also on object a
// because it is the prototype
// to be modified.
console.log(a.print.foo); // 1
```

JS Prototypical Inheritance

- Extending the prototype of the function leads to faster object creations because all the methods are already existing and only need to be referenced instead of being created

- However sometimes you need to have a clear separation between methods of objects of the same class

Looping in Objects (For In)

- Javascript does not guarantee clear separation between variables of the prototype and of the object itself

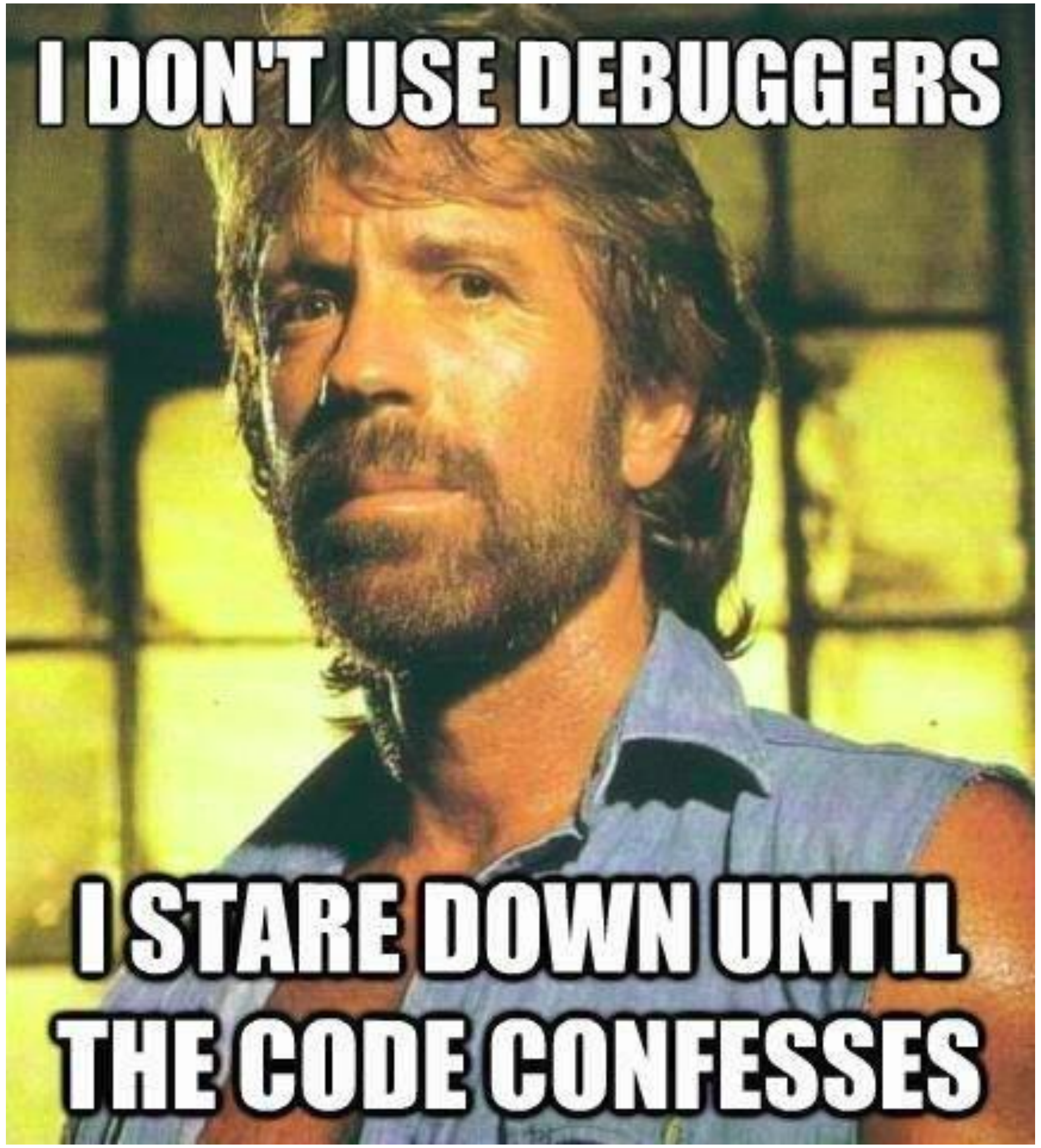
- Therefore, when looping through the properties of an object it is necessary to invoke the method *.hasOwnProperty*

Looping in Objects (For In)

```
var triangle = { a: 1, b: 2, c: 3 };
function ColoredTriangle() {
  this.color = "red";
}
ColoredTriangle.prototype = triangle;
var obj = new ColoredTriangle();

for (var prop in obj) {
  if (obj.hasOwnProperty(prop)) {
    console.log(obj[prop]);
  }
}
```

Debugging



Debugging

- Use the **debugger** keyword to stop and inspect your live code
- In the browser you need to keep the JavaScript console open
- In node.JS you need to call **node debug (node inspect)**:
- **node debug launcher.js**
- Useful Doc:

http://www.w3schools.com/js/js_debugging.asp

https://nodejs.org/api/debugger.html

Hands On 6: Debugging

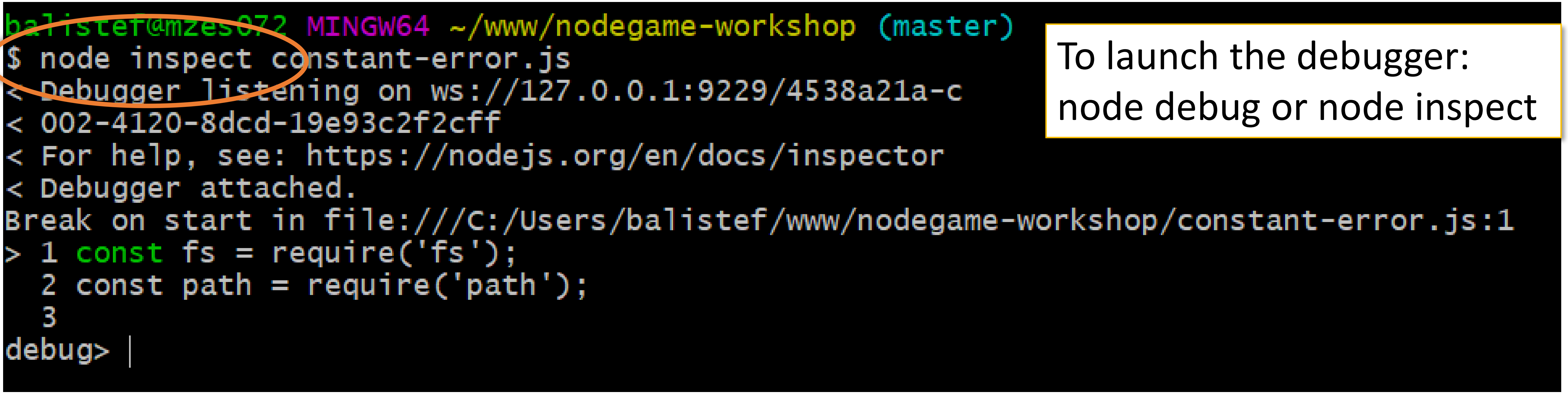
Save the lines below as "constant-error.js" and try to run it.

```
const fs = require('fs');
const path = require('path');

debugger;

// Assign a new property to the fs object.
fs.aNewProperty = 'some value';
// Reassign the fs object.
fs = 'a new life';
```

Hands On 6: Debugging



```
balistef@mzes072 MINGW64 ~/www/nodegame-workshop (master)
$ node inspect constant-error.js
< Debugger listening on ws://127.0.0.1:9229/4538a21a-c
< 002-4120-8dcd-19e93c2f2cff
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in file:///C:/Users/balistef/www/nodegame-workshop/constant-error.js:1
> 1 const fs = require('fs');
  2 const path = require('path');
  3
debug>
```

To launch the debugger:
node debug or node inspect

n: next line

s: step into a function call

Repl: enter into Read–eval–print loop

Linting Tool: JSHint

What is the option -g doing?

```
npm install -g jshint
```

```
jshint constant-error.js --show-non-errors
```

- You might be interested in one of the plugins for editors (vim, emacs, atom, sublime...)