I just met you, and 'this' is crazy, but here's my NaN, so call(me) maybe?

JavaScript you so funny

Rachel Appel
http://rachelappel.com
rachel@rachelappel.com
@RachelAppel
What makes JavaScript fun?
(and by fun I mean....a little bit strange)

• JavaScript
• Blocks
• Functions
• Null
• Equality
• Truthy/Falsey
• Objects
• this!

• Eval
• parseInt
• NaN
• With
• JSLint/JSHint
• Arrays
• Switches
• dates
• more...
JavaScript

I Love it

I Love to make fun of it more
JavaScript
...is not Java

Java is to JavaScript as...
car is to carpet
ham is to hamster
iron is to irony
taco is to Tacoma
Alf is to Gandalf
pot is to potato
One is essentially a toy, designed for writing small pieces of code, and traditionally used and abused by inexperienced programmers.

The other is a scripting language for web browsers.
JavaScript....

has no class
it's so proto-typical

Finally, after many years, ES6 will support classes
Class comes with maturity
DEMO

• Prototypes
The DOM

...is a problem
  Browser incompatibilities
  Malformed HTML
DAYS

SINCE LAST NEW JAVASCRIPT FRAMEWORK
Syntax silly

{} and ;
Syntax silly

• Most often, a newline (\n) ends a statement unless...
  • The statement has an unclosed parenthesis ")", array literal, or object literal.
  • The line uses -- or ++
  • Any block statements such as for, while, do, if, or else, and there is no starting bracket "{"
  • After these constructs: break, continue, return, throw
Syntax Silliness...ASI Affects these:

- empty statement
- var statement
- expression statement
- do-while statement
- continue statement
- break statement
- return statement
- throw statement
ASI

return
"something";

(how the dev sees it)

return;
"something";

(how the parser sees it)
Increment/Decrement Operators

If the operator appears before the variable, the value is modified before the expression is evaluated. If the operator appears after the variable, the value is modified after the expression is evaluated.

In other words,
given \( j = ++k; \), the value of \( j \) is the original value of \( k \) plus one;
given \( j = k++; \), the value of \( j \) is the original value of \( k \), which is incremented after its value is assigned to \( j \).
DEMO

• Increment and Decrement
Putting the "fun" in functions

• Functions are first class
• Named and Anonymous
• Statement and Expression
• Immediate or delayed execution
Named Functions

**Statement**

```javascript
function echo(what) {
    return thing;
}
```

**Expression**

```javascript
var result = echo(what) {
    return thing;
}
```

```javascript
(function echo(what) {
    return thing;
})();
```

**Immediately Invoked function**
Anonymous Functions

**Statement**
```javascript
function () {
  return thing;
}
```

**Expression**
```javascript
var result = () {
  return thing;
}
```

```javascript
(function () {
  return thing;
})(thing);
```

**Immediately Invoked function**
Putting the "fun" in functions

```javascript
var result = functionStatement();
function functionStatement() {
    // Some code
    return value;
}
```

OK

```javascript
var result = functionExpression();
var functionExpression = function() {
    // Some code
    return value;
};
```

Nope
Nope
Nope
DEMO

• Fun with functions
Global

Window object
Global namespace pollution

<script src="1.js">
  var a, b, c
</script>
<bullet>
  <script src="2.js">
    var a, b, c
  </script>
  <script src="3.js">
    var a, b, c
  </script>
  <script src="4.js">
    var a, b, c
  </script>
DEMO

• Global Rachii
Scope & Closures

Block syntax but not block scope

Variables created in a block are available in the entire function
DEMO

• Closures
Arrays

• There is no such thing
• No dimensions
• No out of bounds errors
• typeof doesn't know the difference
DEMO

arrays
Dates

• Date constructor
• Date method
• Moment
• Date.js
DEMO

• Dating in JavaScript
Carry on

continue statement

"I've never seen a piece of code that was not improved by removing it"

-- Douglas Crockford
The switch
Auto fall through

```java
switch (expression) {
    case expression:
        // statements
        [break;]
    case expression:
        // statements
        [break;]
    default:
        // statements
        [break;]
}
```
All things being equal-ish. Maybe.
Let's get to the truth of the matter

<table>
<thead>
<tr>
<th>Truthy values</th>
<th>Falsey values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'false' (quoted false)</td>
<td>false</td>
</tr>
<tr>
<td>'0' (quoted zero)</td>
<td>0 (zero)</td>
</tr>
<tr>
<td>() (empty functions)</td>
<td>'' (empty string)</td>
</tr>
<tr>
<td>[] (empty arrays)</td>
<td>null</td>
</tr>
<tr>
<td>{} (empty objects)</td>
<td>undefined</td>
</tr>
<tr>
<td>All other values</td>
<td>NaN</td>
</tr>
</tbody>
</table>
JavaScript Equality Table

http://dorey.github.io/JavaScript-Equality-Table/
DEMO

Truthy and Falsey
Just what's up with this, anyway?
Sometimes when I'm writing JavaScript, I just want to throw my hands up and yell "This is BS!", but I can never remember what "this" is referring to.
Dynamic Concepts @DynCon... 1h
@RachelAppel - I object to that [pun intended]

Mitch Ruebush @mruebush 1h
@RachelAppel that is so proto-typical of JavaScript

Candice Olechna @Candice... 35m
@RachelAppel don't closure door on javascript!!!
You can force *this* to be an argument.
DEMO

this
Eval or Evil?

- Improper use is a potential security threat
- Code is slower
- A challenge to debug
DEMO

• Eval
Are you *with* me, or against me?

```javascript
with (objX) {
    // statements
}
```
New, new, don't do

• typed wrappers
  • new array
• Use {} or [] instead
DEMO

• A new look at performance
farceInt(fib,lie);

static int parseInt(String s)
static int parseInt(String s, int radix)
### parseInt's farce parsing

<table>
<thead>
<tr>
<th>If it begins with</th>
<th>parseInt uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X or 0x</td>
<td>16 (hexadecimal)</td>
</tr>
<tr>
<td>0</td>
<td>10 (decimal)</td>
</tr>
<tr>
<td>Everything else</td>
<td>10 (decimal)</td>
</tr>
</tbody>
</table>

```javascript
var result = parseInt(inputString, radix);
```
DEMO

• parseInt
NaN

NotANumber

It is a liar

typeof will tell you
NaN

NaN does not equal itself
Don't use equality operators with NaN
  (var == NaN) will not work
  (var === NaN) will not work
ES6 Number.isNaN or isNaN
You want to hear a JavaScript joke?
I'll callback later.
fs.readdir(source, function(err, files) {
    files.forEach(function(filename, fileIndex) {
        console.log(filename)
        gm(source + filename).size(function(err, values) {
            if (err) {
                console.log('Error identifying file size: ' + err)
            } else {
                console.log(filename + ' : ' + values)
                aspect = (values.width / values.height)
                widths.forEach(function(width, widthIndex) {
                    height = Math.round(width / aspect)
                    console.log('resizing ' + filename + 'to ' + height + 'x' + height)
                    this.resize(width, height).write(destination + 'w' + width + '_' + filename, function(err) {
                        if (err) console.log('Error writing file: ' + err)
                    }).bind(this))
                })
            }
        })
    })
})

The problem with callbacks
Seems legit

var add = function() {
    return arguments[0] + arguments[1];
};

console.log(add(4, 4)); // returns 8
How do you avoid all this crazy nonsense?

Linting to the rescue!

JSLint | JSHint | JSFiddle
JSLint
What is both this and that?

After clicking the button, each list item turns red as it becomes "this" during a loop:

```
(function () {
    // this is the whatIsThisButton
    var that = this;
    $('#whatIsThatSpan').text("That is: " + that.id);

    // the following each method loops through the <ul id="aListOfThis">
    $(
    '
    </ul>
    '</ul>

    // this is each <li> in the
```
Thank You!