UNDERHANDED JAVASCRIPT

HOW TO BE A COMPLETE ARSEHOLE WITH BAD JAVASCRIPT

@chewxy
#assholejavascript
AGENDA

• A bunch of small examples
• (Hopefully) One big example
• Throughout: How to be an arsehole.
NAMED FUNCTION EXPRESSION

What does this return?

```javascript
var foo = function bar() {
    console.log("baz")
};

bar();
```
NAMED FUNCTION EXPRESSION

Why?

• It’s a named Function Expression (the other type is a Function Declaration, which cannot be unnamed)
• The name can only be referenced from inside the function.
NAMED FUNCTION EXPRESSION

Why?

- It’s a named Function Expression (the other type is a Function Declaration, which cannot be unnamed)
- The name can only be referenced from inside the function.

Useful When:

- Making recursive function expressions.
- This will lead to a lot of undefined:

```javascript
var foo = function(x) {
  while (x < 10) {
    console.log(foo())
    x++
  }
}

foo(1)
```
- Replacing old code that has arguments.callee in them
IMMEDIATELY INVOKED
FUNCTION EXPRESSIONS

What do these return?

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>`function foo(x) {</td>
<td>`function foo(x) {</td>
<td>`(function foo(x) {</td>
</tr>
<tr>
<td>console.log(arguments)</td>
<td>console.log(arguments)</td>
<td>console.log(arguments)</td>
</tr>
<tr>
<td>return x</td>
<td>return x</td>
<td>return x</td>
</tr>
<tr>
<td>}</td>
<td>}(1, 2, 3, 4, 5)</td>
<td>}(1, 2, 3, 4, 5)</td>
</tr>
</tbody>
</table>

foo(1, 2, 3, 4, 5)

Answer: Depends on where you run it
IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>function foo(x) { console.log(arguments) return x }</code></td>
<td><code>function foo(x) { console.log(arguments) return x (1, 2, 3, 4, 5)</code></td>
<td><code>(function foo(x) { console.log(arguments) return x (1, 2, 3, 4, 5)</code>)</td>
</tr>
</tbody>
</table>

foo(1, 2, 3, 4, 5)

**Even more fun:**

`function foo(x) { return arguments }`  
`foo()`
# IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

## Results:

<table>
<thead>
<tr>
<th></th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chrome/Firefox</strong></td>
<td>[1, 2, 3, 4, 5]</td>
<td>5</td>
<td>[1, 2, 3, 4, 5]</td>
</tr>
<tr>
<td><strong>Node.js REPL</strong></td>
<td>[1, 2, 3, 4, 5]</td>
<td>[1, 2, 3, 4, 5]</td>
<td>[1, 2, 3, 4, 5]</td>
</tr>
</tbody>
</table>

@chewxy
#assholejavascript
IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

Results:

<table>
<thead>
<tr>
<th></th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome/Firefox</td>
<td>[1, 2, 3, 4, 5]</td>
<td>5</td>
<td>[1, 2, 3, 4, 5]</td>
</tr>
<tr>
<td>Node.js REPL</td>
<td>[1, 2, 3, 4, 5]</td>
<td>[1, 2, 3, 4, 5]</td>
<td>[1, 2, 3, 4, 5]</td>
</tr>
</tbody>
</table>

But Node.js runs on V8??!!
IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

Why?

• Example 2 is not an IIFE. It just looks like one.
• Automatic semicolon insertion happens
• 5 is the result of parsing the group expression that follows.
• Basically this:

```javascript
function foo(x){
    // code
};

(1, 2, 3, 4, 5) // group expression
```
IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

Why?

- Older versions of Node.js REPL wraps every command in (), making Example 2 an IIFE.

- foo() is basically this:
  
  (foo)()

- Fixed in latest Node.js :( *

---

* I like shenanigans. It gives people reason to consult me.
IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

Being an arsehole - because this looks like an IIFE, no one will give it a double look:

```javascript
function NotActuallyGoingToBeCalled() {
    // blablabla
    // make this code block as long as possible
    // just to confuse other people
}(function ActualFunctionThatGivesEffect() {
    // The name of the function is optional.
    // do evil things here
})();
```

Even more arseholish move: Give the evil IIFE the same name as the function declaration.
QUICK QUIZ

Remember that Function Expressions have optional names to facilitate recursion.

Remember that only Function Expressions can be immediately invoked.

What does this return? Why?

(function foo(foo) {
    console.log(foo())
})(function() { return 1 })
THE COMMA TRICK

What does this return?

```javascript
arr = [1, 2, 3, 4][1, 2, 3]
```
THE COMMA TRICK

Why?

```javascript
arr = [1, 2, 3, 4][1, 2, 3]
```

Array Literal

Array Accessor
THE COMMA TRICK

¿Por qué?

arr = [1, 2, 3, 4][1, 2, 3]

Array Literal

Array Accessor

[((1, 2), 3)]
THE COMMA TRICK

为什么？

```
arr = [1, 2, 3, 4][1, 2, 3]
```

Array Literal

Array Accessor

```
[(((1, 2), 3))]
```

Evaluate ((1, 2), 3)
Evaluate (1, 2)
Evaluate 1 ➔ Discard
Evaluate 2 ➔ Return
Evaluate (2, 3)
Evaluate 2 ➔ Discard
Evaluate 3 ➔ Return
STATEMENTS!

These are statements:

• if – else
• for loop
• while loop
• switch statements
• Block statements delimited by { and }

@chewxy
#assholejavascript
TRUTHY AND FALSEY

Everyone knows this:

```javascript
var x = "true"
if (x) {
    console.log(x == true)
} else {
    console.log("x is not a truthy value")
}
```

But why?
TRUTHY AND FALSEY

And then trip up on this!:

```javascript
var x = [0]
if (x) {
  console.log([0] == 0)
} else {
  console.log("[0] is not a truthy value")
}

But why?
```

@chewxy
#assholejavascript
TRUTHY AND FALSEY

Why?

• Truthy and Falsey values are coerced to Booleans when in statements
• Values are coerced to Numbers when in relational (equality) expressions
• Doesn’t mean you should NOT use ==. Instead, use with care
QUICK QUIZ

What do these return?

```javascript
{}
[0] == true

{}
[1] == true

!{}
[0] == true

!{}
[1] == true
```
STATEMENTS!

These are statements:

• if – else
• for loop
• while loop
• switch statements
• Block statements delimited by { and }

@chewxy
#assholejavascript
TRUTHY AND FALSEY

Why?

• Truthy and Falsey values are coerced to Booleans when in statements
• Values are coerced to Numbers when in relational (equality) expressions
• Doesn’t mean you should NOT use ==. Instead, use with care

@chewxy
#assholejavascript
What does this return?

```javascript
var x = 1;
switch(x) {
  case '1':
    console.log("Hey I just met you");
    break;
  case '2':
    console.log("And this is craaaazy");
    break;
  case 1:
    console.log("But here's my number");
    break;
  case 2:
    console.log("So call me maybe");
    break;
}
```
JAVASCRIPT

Y U NO CONSISTENT

@chewxy
#assholejavascript
Did you know that V8 has block level variable scoping without ES 6 syntax?

```javascript
function foo() {
  var x = 1
  {
    function x(){
      return 2;
    }
  }
  return x
}

What happens when you call foo() and what is the value of x?
```

@chewxy
#assholejavascript
STATEMENTS!

These are statements:
• if – else
• for loop
• while loop
• switch statements
• **Block statements delimited by { and }**
Did you know that V8 has block level variable scoping without ES 6 syntax?

```javascript
function foo() {
  var x = 1
  {
    function x(){
      return 2;
    }
  }
  return x
}
```

What happens when you call `foo()` and what is the value of `x`?
LAST ONE (I PROMISE)

Did you know that V8 has block level variable scoping without ES 6 syntax?

Just kidding. It doesn’t. But it appears to have block level scoping
BUY MY BOOK

It’s helluva less inconsistent than Javascript. It’s flexible like Javascript: Only 2 titles to choose from!

https://leanpub.com/underhandedjavascript

https://leanpub.com/jsinterviewquestions