

fun.js

por

David Avellaneda

@Davsket



davsket? who's that guy?

about:me

about:me

about:me

Nombre: David Avellaneda

Twitter: @davsket

Github: davsket

Profesión: Desarrollador Front-end o UI Developer

monoku.

Co-fundador de Monoku



monomi
Tiendas Online



Col

Cambia de país ▼

Sobre monomi

Blog

Beneficios

| Planes

| Galería

| FAQ

| Pasarelas de Pago

Ingresar

Regístrate

La forma más fácil de vender
desde tu propia tienda virtual

Abre tu tienda ahora >

Carolina Serrano
www.coronadepapel.com

y Monomi!

BOG.JS

Co-organizador de BogotáJS



y también de Col.js



JSCONF COLOMBIA

MEDELLIN / OCTOBER 15 - 17TH 2015



RAQUEL VELEZ
NPM



JENN SCHIFFER
BOCOUP



ALEX SEXTON
STRIPE

y también de JSConf.co

enter the fun!

pero antes...

tener en cuenta

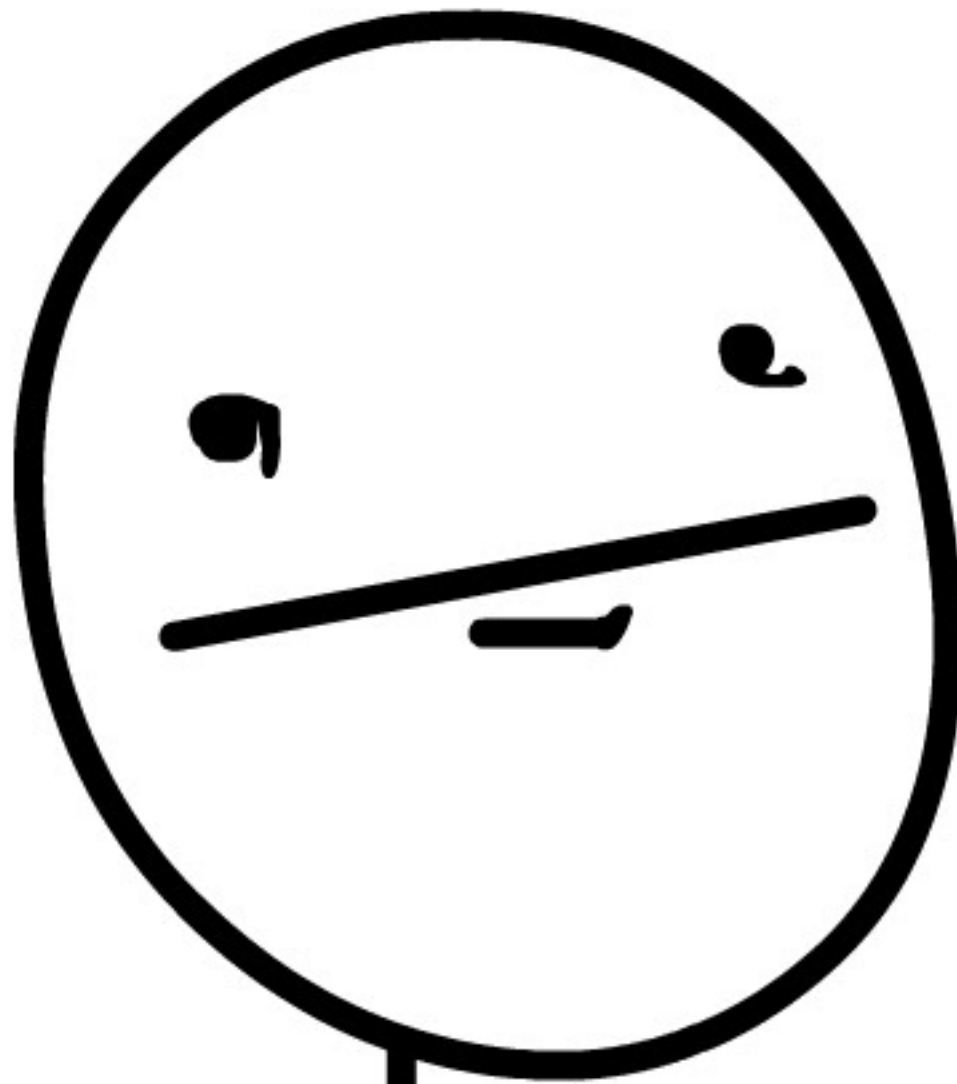
- Voy a usar chrome como navegador
- Voy a usar la consola del navegador para ejecutar los ejemplos
- Voy a usar mucho la función `console.log()`
- Voy a tratar de mostrar JavaScript puro sin jQuery y `cía`.

enter the fun!

la base de todo lo cool de JavaScript
las fun.. fun.. funciones

“Las funciones en JavaScript son objetos de primera clase "first-class objects".”

–David Avellaneda, 2013



POKER FACE

first-class objects

1. Las funciones son objetos
2. Las funciones tienen propiedades
3. Pueden asignarse a una variable
4. Pueden pasarse como argumento a otra función
5. Pueden retornarse

Las funciones son objetos

```
function a() {}  
typeof a;  
// function  
a instanceof Object;  
// true
```


Las funciones tienen propiedades

```
function a() {}  
a.name;  
// 'a'  
a.constructor;  
// function Function()...
```

Pueden asignarse a una variable

```
var a = function() {  
    return 4  
}  
var b = a;  
a(); // 4  
b(); // 4
```

Pueden pasarse como argumento a otra función

```
function a ( ) {  
    return 3;  
}  
function b ( arg ) {  
    return arg();  
}  
b(a); // 3
```

Pueden retornarse

```
function b( n ){  
    return function( m ){  
        return n * m;  
    }  
}  
b(2)(3);  
// 6
```



recursión

recursión



Web

Images

Videos

Books

Apps

More ▾

Search tools

About 69,100 results (0.44 seconds)

Did you mean: [recursión](#)

Recursion - Wikipedia, the free encyclopedia

en.wikipedia.org/wiki/Recursion ▾

Recursion is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other, the nested ...

recursión: fibonacci

n	0	1	2	3	4	5	6	7	...
fib	1	1	2	3	5	8	13	21	...

recursión: fibonacci

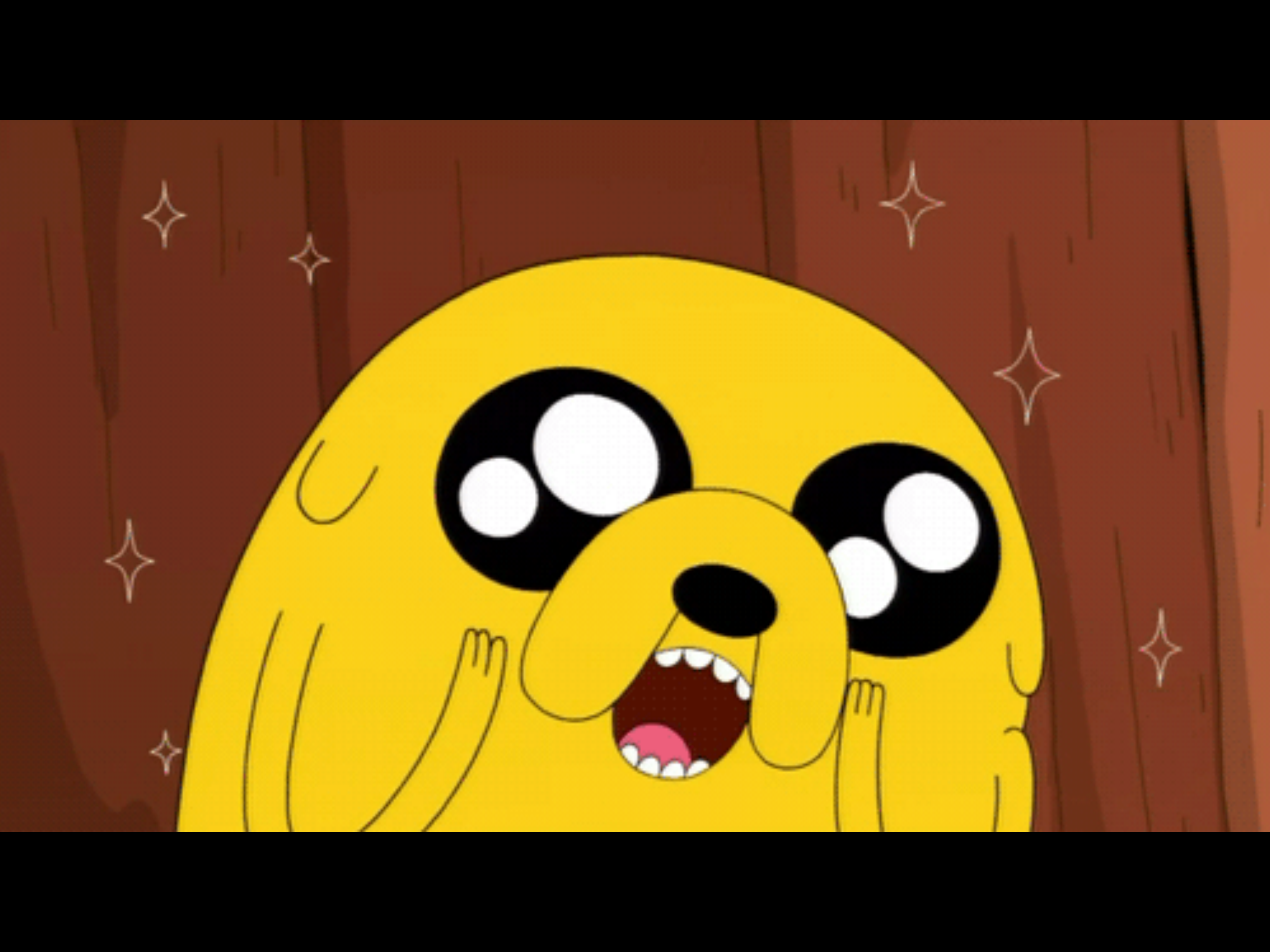
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

JavaScript: fibonacci

```
function fib( n ){  
    if( n<2 ) return 1;  
    return fib(n-1) + fib(n-2);  
}  
fib(7);  
// 21 !!
```

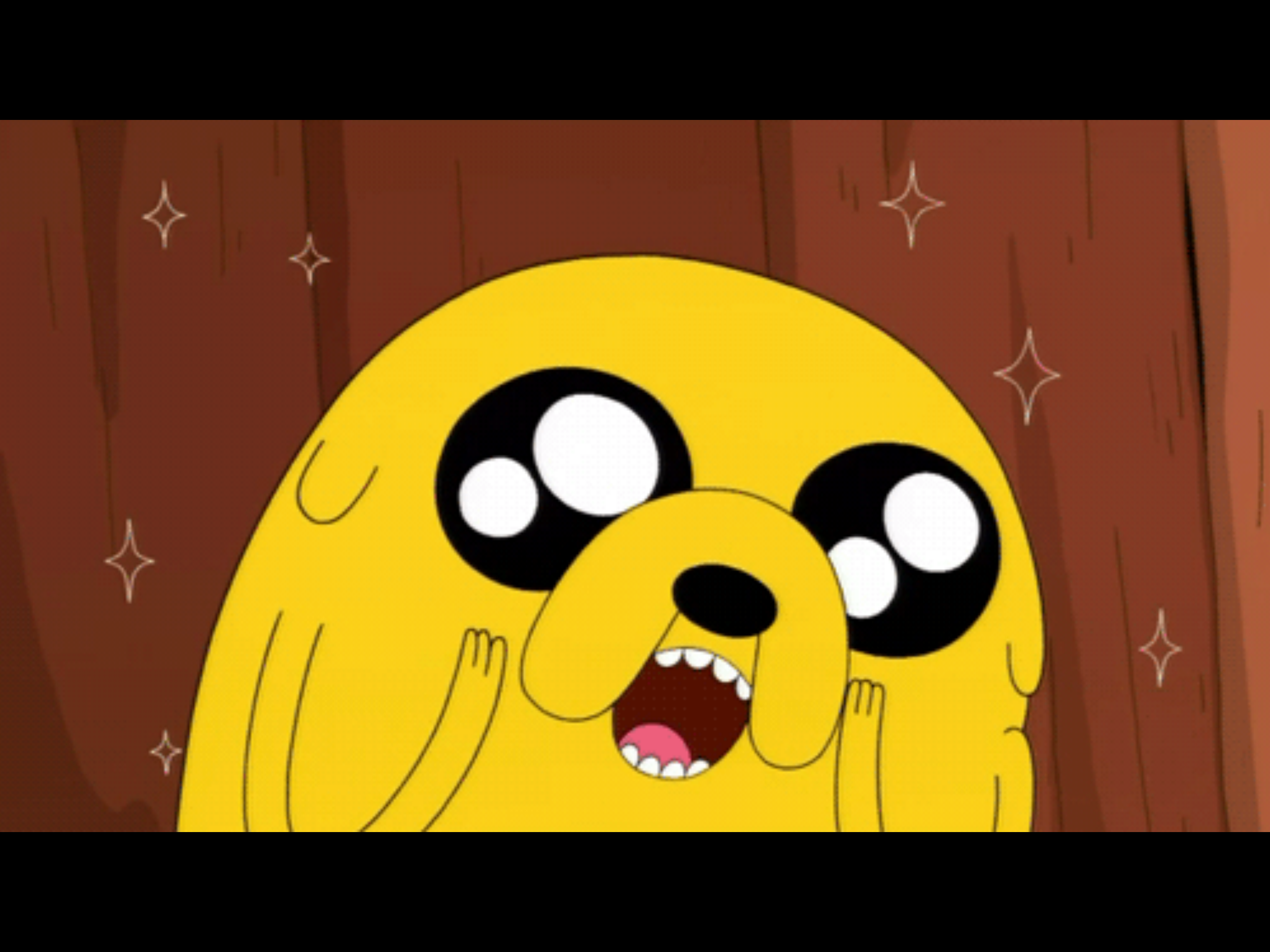

enter the fun!

las funciones tienen funciones!



las funciones de las funciones

- call
- apply
- bind



fun.call

fun.call

Llama a la función pasándole el this y los argumentos separados por coma.

```
fun.call( thisArg[, arg1[, arg2[, ...]]] )
```

```
var obj = {  
  a: 3,  
  f: function( n ){  
    return this.a * n;  
  }  
};
```

```
obj.f( 2 ); // 6  
var obj2 = {  
  a: 8  
};
```

```
obj.f.call(obj2, 2); // 16
```


fun.call: caso real

ES5 (Todos e IE8+) facilita unas funciones en los arreglos, una de ella es forEach, con la cual iteramos por cada item de un arreglo.

```
var arr = [1,2,3,4];  
arr.forEach(function( item ){  
    console.log( item );  
})  
// un log por cada item de arr
```

fun.call: caso real

El navegador tiene la función:
document.querySelectorAll, que
devuelve los elementos que coinciden
con un selector **CSS**.

```
var links = document.querySelectorAll( 'a' );
```

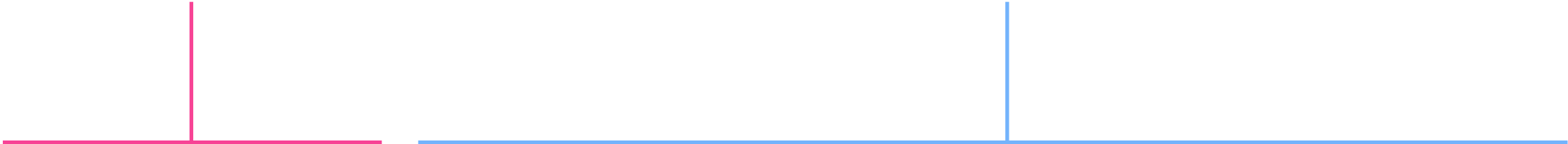
Pero links no es un arreglo, es un
NodeList y no tiene función ForEach

fun.call: caso real

como evitar hacer un ciclo for...

la función foreach
de un arreglo

la llamo sobre el
Nodelist con mi
función



```
[ ].forEach.call( links, function( link ){  
    console.log( link );  
} )
```

fun.apply

fun.apply

Llama a la función pasándole el this y un arreglo de argumentos.

```
fun.apply( thisArg[, argsArray] )
```

fun.apply

```
function a( a, b, c, d ){  
    console.log( a*a, b*b, c*c, d*d );  
}  
var arr = [ 1, 2, 3, 4 ];  
a.apply(null, arr);  
// 1, 4, 9, 16
```

fun.apply: caso real

El objeto global **Math** tiene funciones matemáticas como max, min, sin, cos... Podemos aprovechar el poder de apply, para invocar las funciones **max** y **min** y ahorrarnos código.

```
var arr = [9,23,55,322,11,3,59,12,122];  
Math.max.apply(null, arr); // 322  
Math.min.apply(null, arr); // 3
```




FEEL LIKE A SIR

fun.bind

fun.bind

Crea una nueva función, la cual, tiene asignado el this y sus argumentos.

```
fun.bind(thisArg[, arg1[, arg2[, ...]]])
```

```
var obj = {  
  a: 4,  
  f: function( n ){  
    return this.a * n;  
  }  
};
```

```
// Si por error hago esto:  
var f = obj.f;  
f(3); // NaN
```

Porque **this.a** es indefinido, **f** se está ejecutando fuera de **obj**.

```
var obj = {  
  a: 4,  
  f: function( n ){  
    return this.a * n;  
  }  
};
```

```
var f = obj.f.bind( {a:6} );  
f(1); // 6
```

```
var f5 = obj.f.bind( {a:6}, 5 );  
f5();  
// 30 <= (this.a = 6) * (n = 5)
```

fun.bind: caso real

Al modularizar el código de tus aplicaciones..

```
var app = {  
  msg: 'hola platzi!!',  
  init: function() {  
    document.addEventListener(  
      'click', this.sayHi.bind( this ));  
  },  
  sayHi: function() {  
    alert( this.msg );  
  }  
}  
app.init(); // click en la pag!
```


fun.bind: caso real 2

Otro posible uso: si abren el navegador, en la consola encontrarán dos funciones: **\$** y **\$\$** (no son jQuery).

La primera es un shortcut para **document.querySelector** que devuelve el primer elemento que coincida con el selector CSS que se le pase.

La segunda invoca **document.querySelectorAll** que devuelve **TODOS** los que coincidan.

fun.bind: caso real 2-1

cómo hacer nuestros queries más fácil
(sin jQuery)

```
var $ = document.querySelector.bind( document );  
var $$ = document.querySelectorAll.bind( document );
```

// ahora sí

\$('a'); // el primer link

\$\$ ('p'); // todos los párrafos!

fun.bind: caso real 2-1

cómo hacer nuestros queries más fácil
(sin jQuery)

Aplicando lo anterior

```
[ ].forEach.call( $$('p'), function( p ) {  
    console.log( p.innerHTML );  
})
```

Que claro, con jQuery sería tan solo

```
$('p').each( function( i, p ) {  
    console.log( p.innerHTML );  
})
```

fun fun fun!



yo se...

creando un microquery

```
function q( selector ){  
    return [].slice.call(  
        document.querySelectorAll( selector ) )  
}
```

fun.bind: caso real 3-1

haciendo un **microquery**!

```
q( 'p' )  
  .filter( function( p ) {  
    return p.innerHTML.match( /lorem/i )  
  } )  
  .slice( 2, 6 )  
  .forEach( console.log.bind( console ) )
```

Gracias!

follow me: @davsket