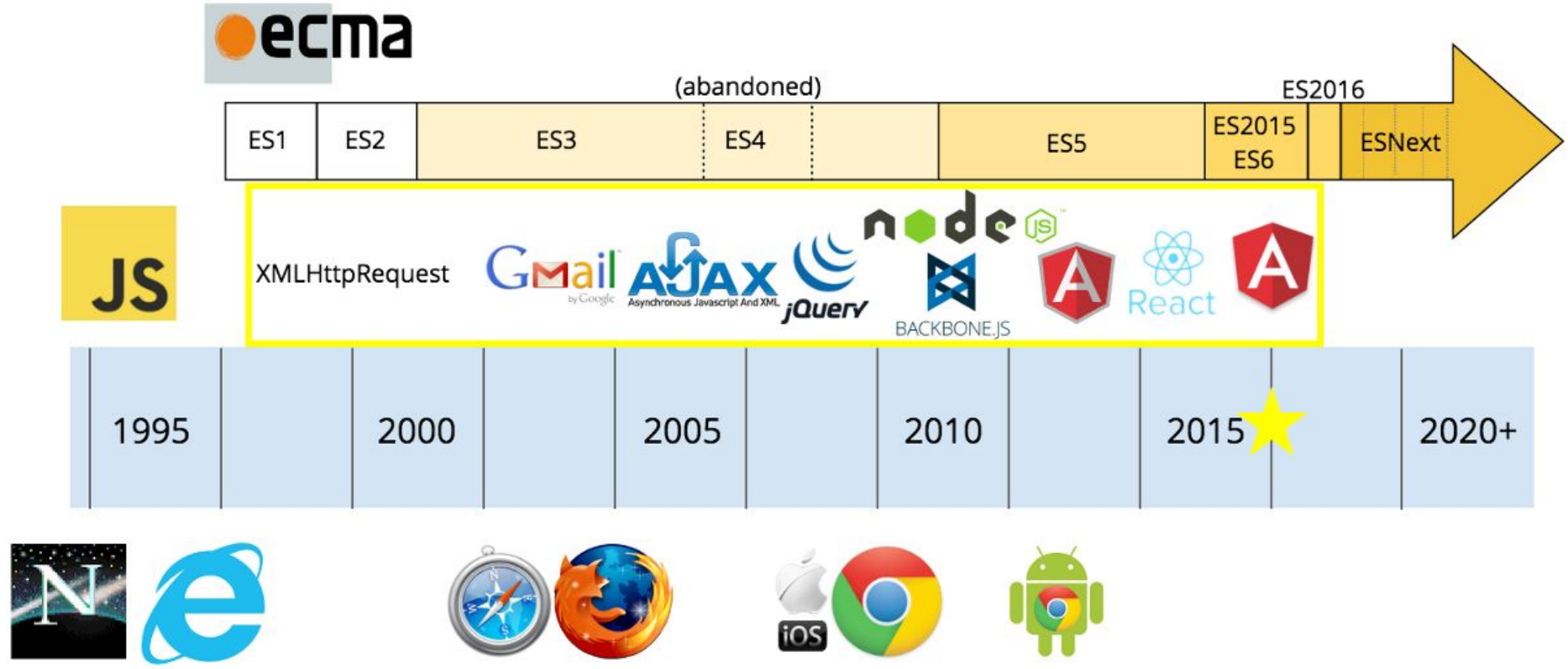
JavaScript:
Lo bueno, lo malo y lo feo

Agenda

- Introducción y motivación
- Características y sintaxis
- Sistema de tipos
- Paradigmas de programación
- Conclusiones

Historia

- Es un lenguaje de programación creado en 1995 por Brendan Eich (NetScape).
- Nombrado inicialmente como “LiveScript”, luego renombrado a “JavaScript” por estrategia de negocio.
- Enviado a European Computer Manufacturers Association (ECMA) International en 1997 para formar un estándar. De ahí se formó el estándar ECMA 262, que es la especificación del lenguaje “ECMAScript” (ES).

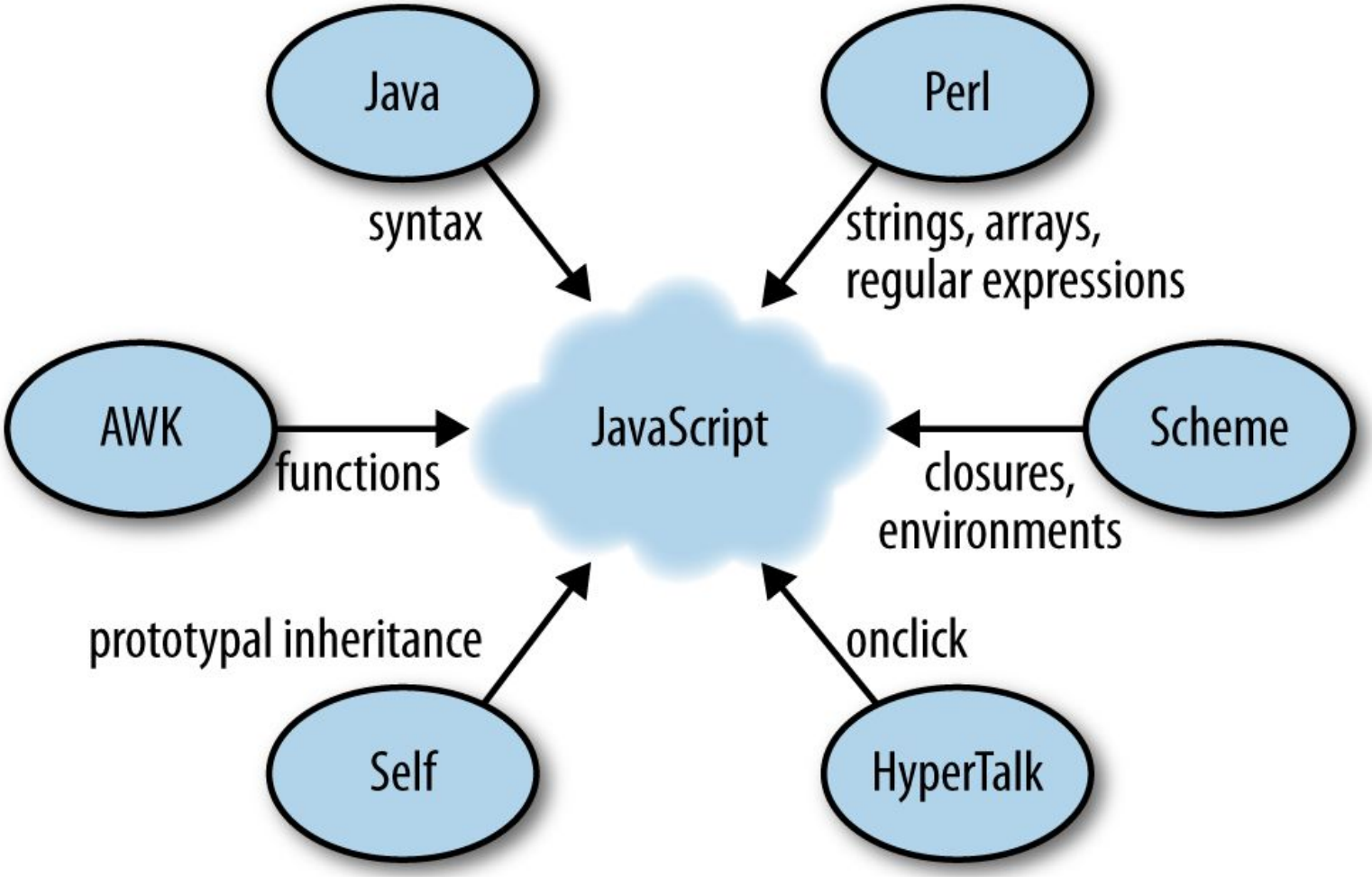

ecma
(abandoned)
ES2016
ES1
ES2
ES3
ES4
ES5
ES2015
ES6
ESNext
JS
XMLHttpRequest
Gmail
by Google
AJAX
Asynchronous Javascript And XML
jQuery
node
BACKBONE.JS
React
1995
2000
2005
2010
2015
2020+
iOS


enyo
Ractive.js
Sammy.js
GWT
dojo
canjs
mochikit
jQuery
ember
Knockout.
ext js
METEOR


I'M FRANKENSTEIN'S MONSTER
YOU DON'T SEEM LIKE A MONSTER
I'M A JAVASCRIPT DEVELOPER

Características

- Interpretado
- Multiparadigma / Scripting
- Dinámicamente tipado
- Prototipado
- Liviano
- Expresivo


Java
syntax
Perl
strings, arrays,
regular expressions
AWK
functions
JavaScript
Scheme
closures,
environments
prototypal inheritance
Self
onclick
HyperTalk

Sintaxis

```
if (a > 0) {
  console.log('Es mayor que cero');
} else {
  var mensaje = a < 0 ? 'menor' : 'igual';
  console.log('Es ', mensaje, ' a cero');
}

for (var i=0; i<10; i++) {
  console.log(i);
}
```

Sintaxis

```
var pi = 3.1416;

var cuadrado = function (x) {
    return x * x;
}

function area(radio) {
    return pi * cuadrado(radio);
}

var perimetro = radio => 2 * pi * radio;
```

Sistema de Tipos

Tipos

- undefined
- null
- boolean
- number
- string
- object
- symbol

undefined

```
var a;
var b = null;

b = undefined;

console.log(a); // undefined
console.log(b); // undefined


var c;

c; // undefined
d; // ReferenceError: d is not defined
```

number

```
42
42.0
42.

42.3
42.300

0.5
.5
```

```
1E3
1.1E6
2e-5

0xf3
0Xf3

0o363
0O363

0b11110011
0B11110011
```

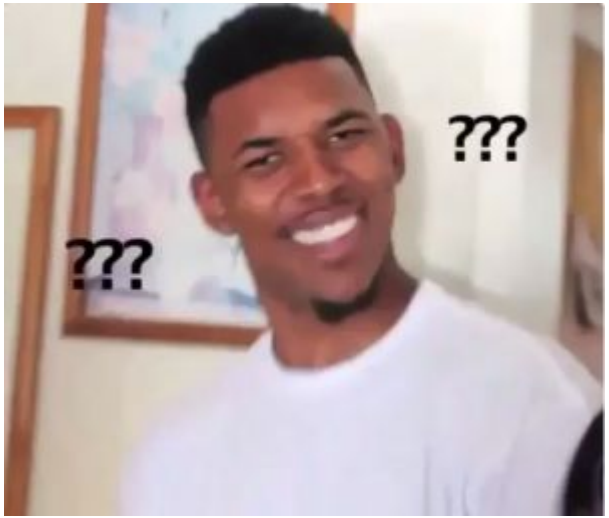
```
-Infinity
Infinity
+0
-0
NaN
```

object

```
var obj = {
  nombre: 'Prueba',
  ejemplo: true,
  contador: 42,
  saludar: function() {
    console.log('Hola!');
  },
  proximo: {
    nombre: 'Otro'
  },
  multiplos: [1, 2, 3, 5]
};
```

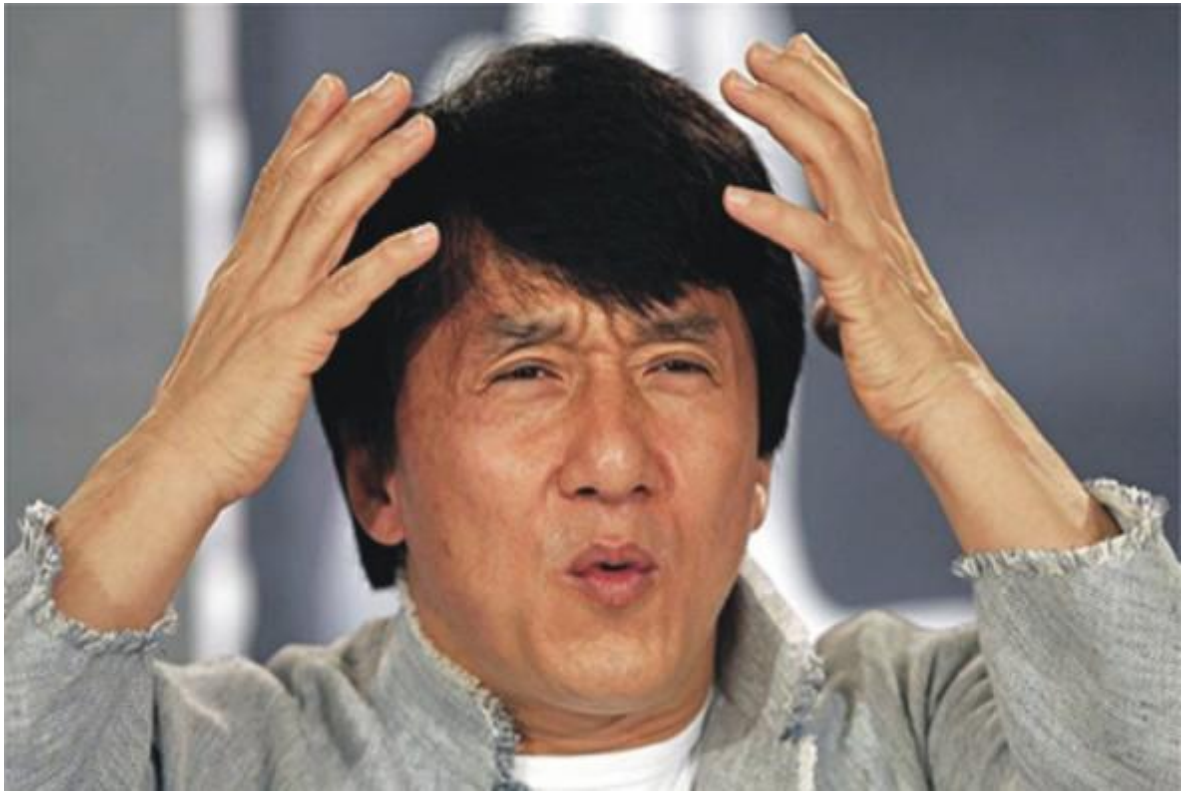
typeof

```
typeof 123.45          // "number"
typeof "hola"          // "string"
typeof true            // "boolean"
typeof undefined       // "undefined"
typeof Symbol()        // "symbol"
typeof {}              // "object"
typeof []              // "object"
typeof NaN             // "number"
typeof function(){}    // "function"
typeof null            // "object"
```



```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```

```
Math.max()
  -Infinity
Math.min()
   Infinity
typeof NaN
   "number"
0.1 + 0.2 == 0.3
   false
```



a == b

a === b

```
true == 1   // true
true === 1  // false

Number(true)    // 1
```

```
0 == ''     // true
0 == '0'    // true

'' == '0'   // false
```

```
[] == ![]    // true

Boolean(null)   // false
null == false   // false
```

```
9 + "1"          // "91"
91 - "1"         // 90
true + true      // 2


[] + []          // ''
[] + {}          // '[object Object]'
{} + []          // 0
```

Scope

- ¿Léxico o Dinámico? Léxico!
- Scope a nivel funciones (mediante `var`).
- Scope a nivel bloques (mediante `let` o `const`). Agregado en 2015.

this

```
function foo() {
  console.log(this);
}
var a = 1;
var obj = {
  a: 2,
  foo: foo
};
var obj2 = {
  a: 3
};
```

1 - Ligadura por defecto

```
foo();
```

2 - Ligadura implícita

```
obj.foo();
```

3 - Ligadura explícita

```
foo.call(obj2);
```

4 - Ligadura mediante new

```
var b = new foo();
```

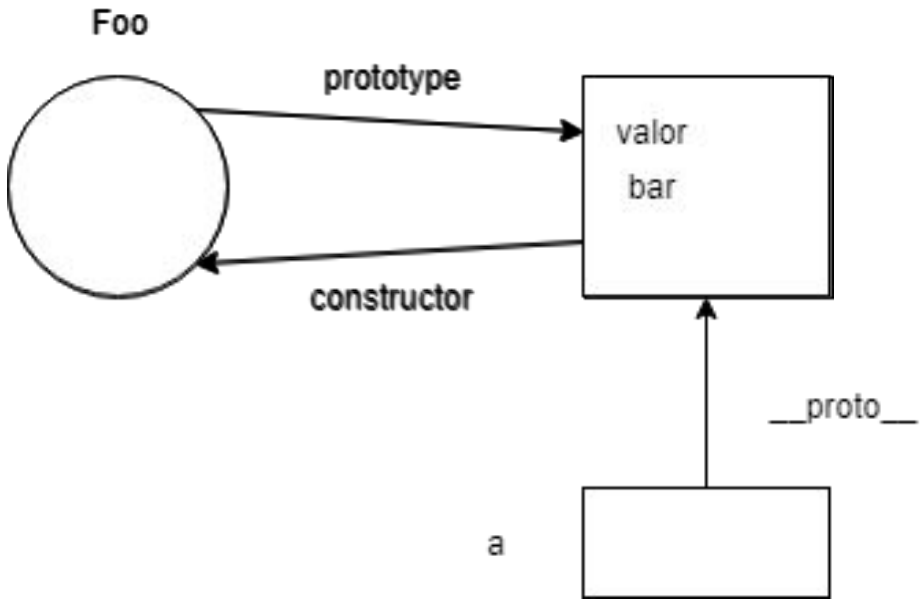

JavaScript
FOR
KIDS


JavaScript
FOR
KIDS

Paradigma Orientado a Objetos

Prototype

```
function Foo() {}
console.log(Foo.prototype); // {}
// agregamos propiedades al prototipo
Foo.prototype.valor = 42;
Foo.prototype.bar = function() {
  console.log('bar');
};
console.log(Foo.prototype);
// { valor: 42, bar: [Function] }
console.log(Foo.prototype.constructor);
// [Function: Foo]

 var a = new Foo();
 a.toString();
```



Clases

```
function Animal(nombre) {
  this.nombre = nombre;
}

Animal.prototype.saludar = function () {
  console.log('Hola, soy ', this.nombre);
};

var gato = new Animal('Garfield');
var perro = new Animal('Oddie');

perro.saludar();                // Hola, soy Oddie
gato.saludar();                 // Hola, soy Garfield
```

Clases

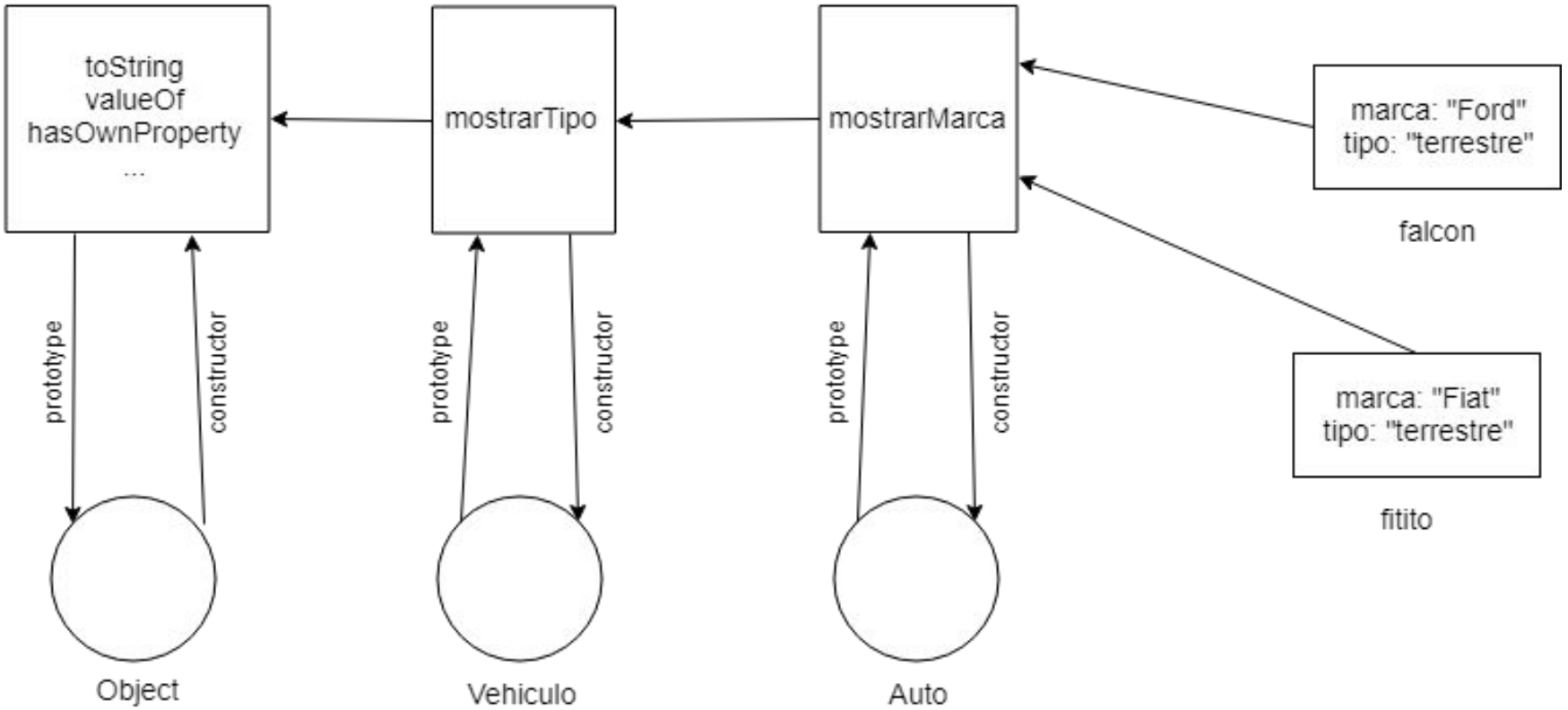
```
class Animal {
  constructor(nombre) {
    this.nombre = nombre;
  }
  saludar() {
    console.log('Hola, soy ', this.nombre);
  }
}

var gato = new Animal('Garfield');
var perro = new Animal('Oddie');

perro.saludar();                // Hola, soy Oddie
gato.saludar();                 // Hola, soy Garfield
```

Herencia

```
function Vehiculo(tipo) {
  this.tipo = tipo;
}

Vehiculo.prototype.mostrarTipo = function() {
  console.log(this.tipo);
};


 function Auto(marca) {
   Vehiculo.call(this, 'terrestre');
   this.marca = marca;
 }
```

```
Auto.prototype = Object.create(Vehiculo.prototype);
Auto.prototype.constructor = Auto;

Auto.prototype.mostrarMarca = function() {
  console.log(this.marca);
};


 var fitito = new Auto('Fiat');
 var falcon = new Auto('Ford');
 fitito.mostrarMarca(); // Fiat
 fitito.mostrarTipo(); // terrestre
 falcon.mostrarMarca(); // Ford
 falcon.mostrarTipo(); // terrestre
```

Herencia


toString
valueOf
hasOwnProperty
...
mostrarTipo
mostrarMarca
marca: "Ford"
tipo: "terrestre"
falcon
marca: "Fiat"
tipo: "terrestre"
fitito
prototype
constructor
prototype
constructor
prototype
constructor
Object
Vehiculo
Auto

Herencia

```
class Vehiculo {
  constructor(tipo) {
    this.tipo = tipo;
  }
  mostrarTipo() {
    console.log(this.tipo);
  }
}
```

```
class Auto extends Vehiculo {
  constructor(marca) {
    super('terrestre');
    this.marca = marca;
  }
  mostrarMarca() {
    console.log(this.marca);
  }
}
```

Encapsulamiento

```
function Persona(nombre, saldo) {
  var saldoPrivado = saldo;
  this.nombre = nombre;
  this.obtenerSaldo = function() {
    return saldoPrivado;
  };
  this.actualizarSaldo = function(nuevoSaldo) {
    saldoPrivado = nuevoSaldo;
  };
}
var pepe = new Persona('Jose', 25);
```

Polimorfismo

```
class Animal {
  mover() {}
}
class Pez extends Animal {
  mover() {
    console.log('Soy pez y estoy nadando...');
  }
}
class Ave extends Animal {
  mover() {
    console.log('Soy ave y estoy volando...');
  }
}
```

```
var animales = [new Pez(), new Ave()];

for (let animal of animales) {
  animal.mover();
}

// Soy pez y estoy nadando...
// Soy ave y estoy volando...
```

Módulos

- Patrones de diseño (ej: Singleton)
- Estándares hechos por la comunidad (AMD y CommonJS)
- `import` / `export` en ES6 (2015)

Paradigma Funcional

Recursividad

```
// Función recursiva con función como declaración
function sumatoria(n) {
  return n > 0 ? n + sumatoria(n - 1) : 0;
}

// Función recursiva con función como expresión
var factorial = function(n) {
  return n > 0 ? n * factorial(n - 1) : 1;
};

// Recursión cruzada
var esPar = num => (num === 0 ? true : esImpar(num - 1));
var esImpar = num => (num === 0 ? false : esPar(num - 1));
```

Funciones puras e impuras

```
// Funciones puras
function sumar(a, b) {
  return a + b;
}

function calcularPrecio(cantidad, costo) {
  return cantidad * costo;
}
```

```
// Funciones impuras
function obtenerDiaActual() {
  return new Date().getDate();
}

function obtenerTextoPorId(id) {
  return document.getElementById(id).textContent;
}

const PI = Math.PI;
function calcularArea(radio) {
  return PI * radio * radio;
}
```

Funciones de primera clase

```
function saludar() {
  console.log('hola');
}
var despedirse = () => console.log('chau');

console.log(saludar); // [Function: saludar]
console.log(despedirse); // [Function: despedirse]

console.log(saludar.toString());
// function saludar() { console.log('hola'); }
console.log(despedirse.toString());
// () => console.log('chau')
```

Funciones de alto orden

```
function ejecutar(fn) {
  fn();
}
var saludar = () => console.log('hola mundo!');

console.log(typeof saludar); // function
ejecutar(saludar); // hola mundo!
```

Funciones de alto orden

```
function generador(prefijo) {
  return texto => console.log(prefijo + ' ' + texto);
}
var imprimir = generador('hola');

console.log(typeof imprimir); // function
imprimir('che!'); // hola che!
imprimir('a todos'); // hola a todos
```

Otros conceptos del PF

- Inmutabilidad
- Semántica de valores
- Evaluación ansiosa y perezosa

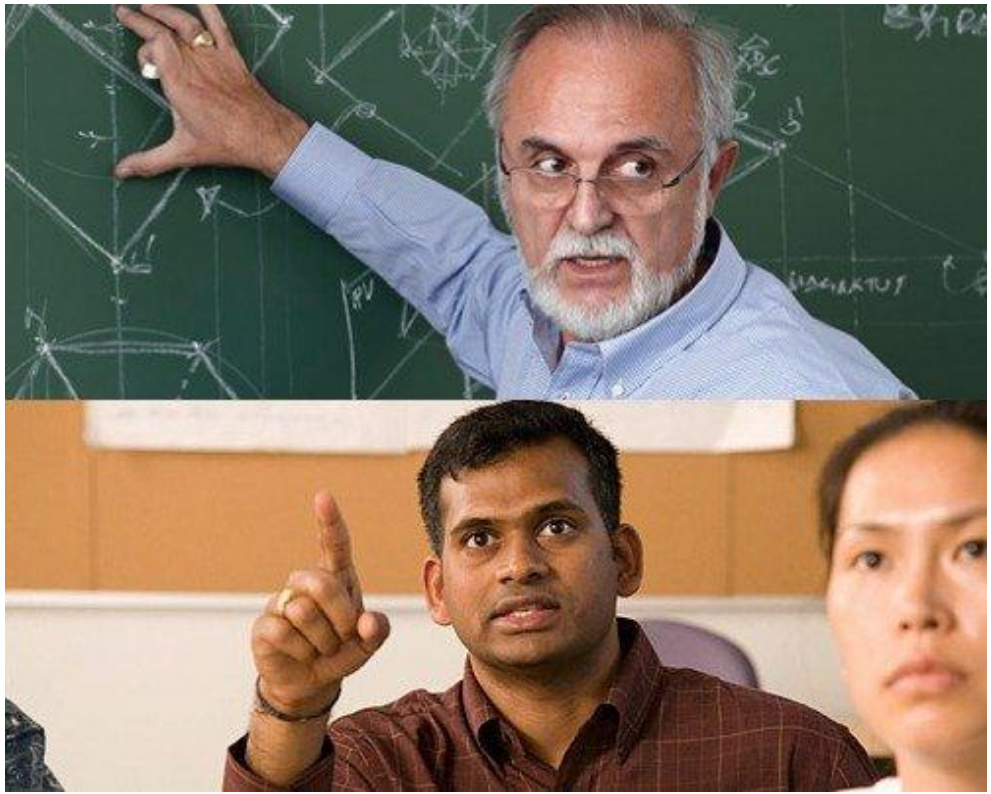
Conclusiones

“El lenguaje es sólamente una herramienta”

Más conceptos a analizar

- Novedades en ES6: Template literals, spread operator, rest operator, default values, destructuring.
- Meta-Programación: Symbol, Proxy, Iterators, Generators.
- Concurrencia: Cómo funciona JS (single thread) y cómo funciona el event loop.
- Asincronía: Callbacks, Promises, Observables, Generators, `async` y `await`.

¿Preguntas?



¡Muchas Gracias!