CO BYSTE MĚLI VĚDĚT O JAVASCRIPTU?

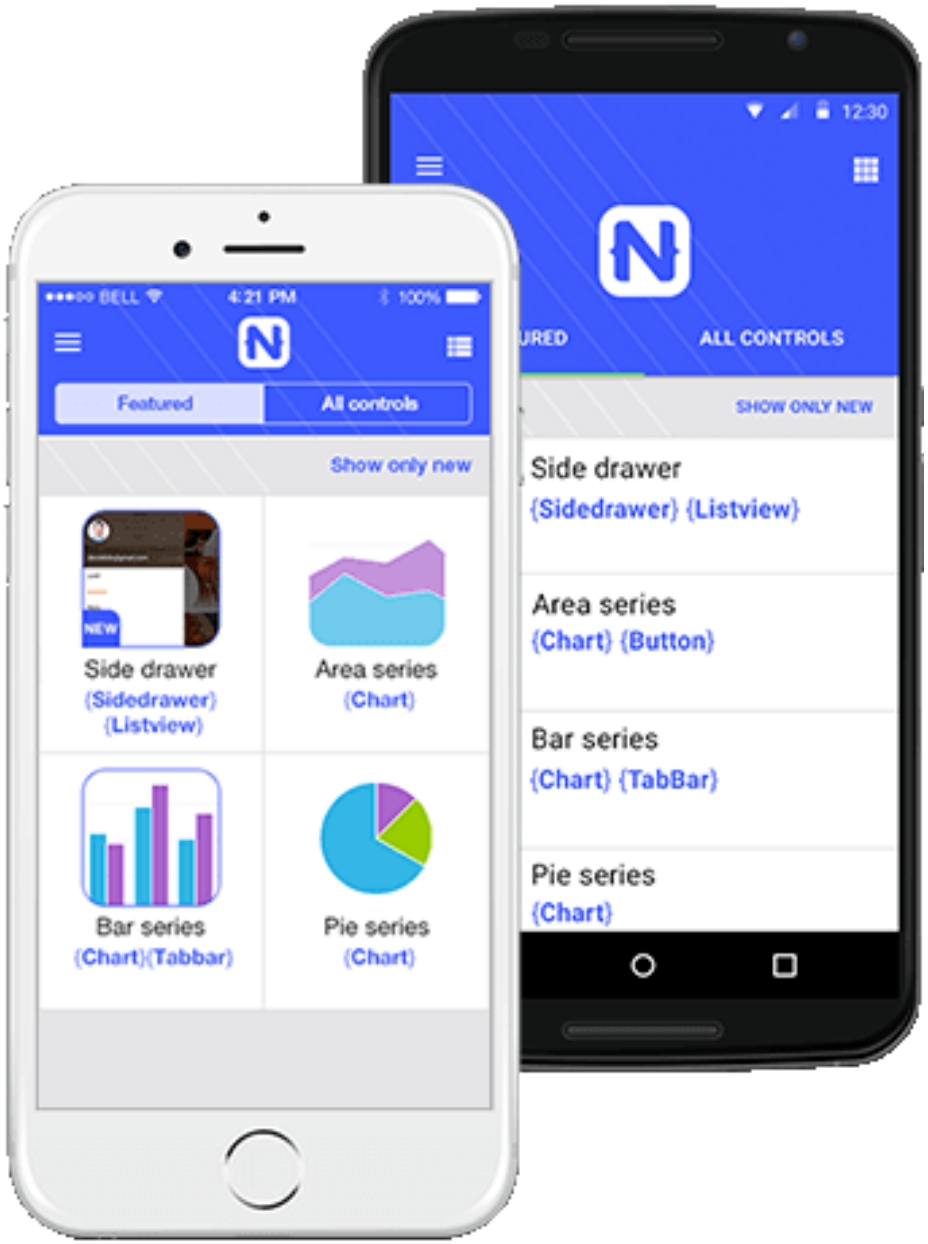
Milan Lempera @milanlempera
Víťa Plšek @winsik

Angular.cz

Příklady si můžete otevřít zde:

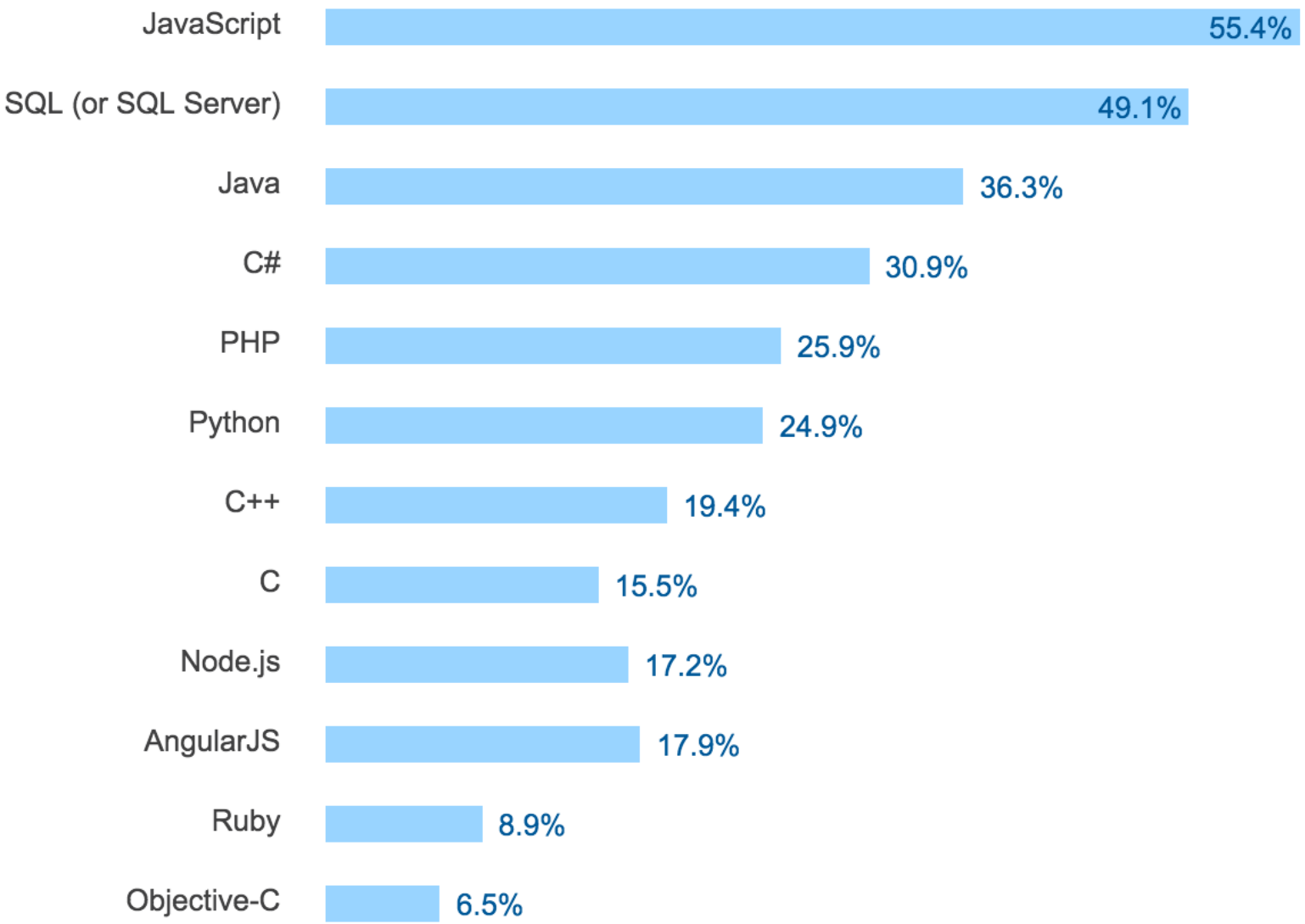
ANGULAR.CZ/INFS2

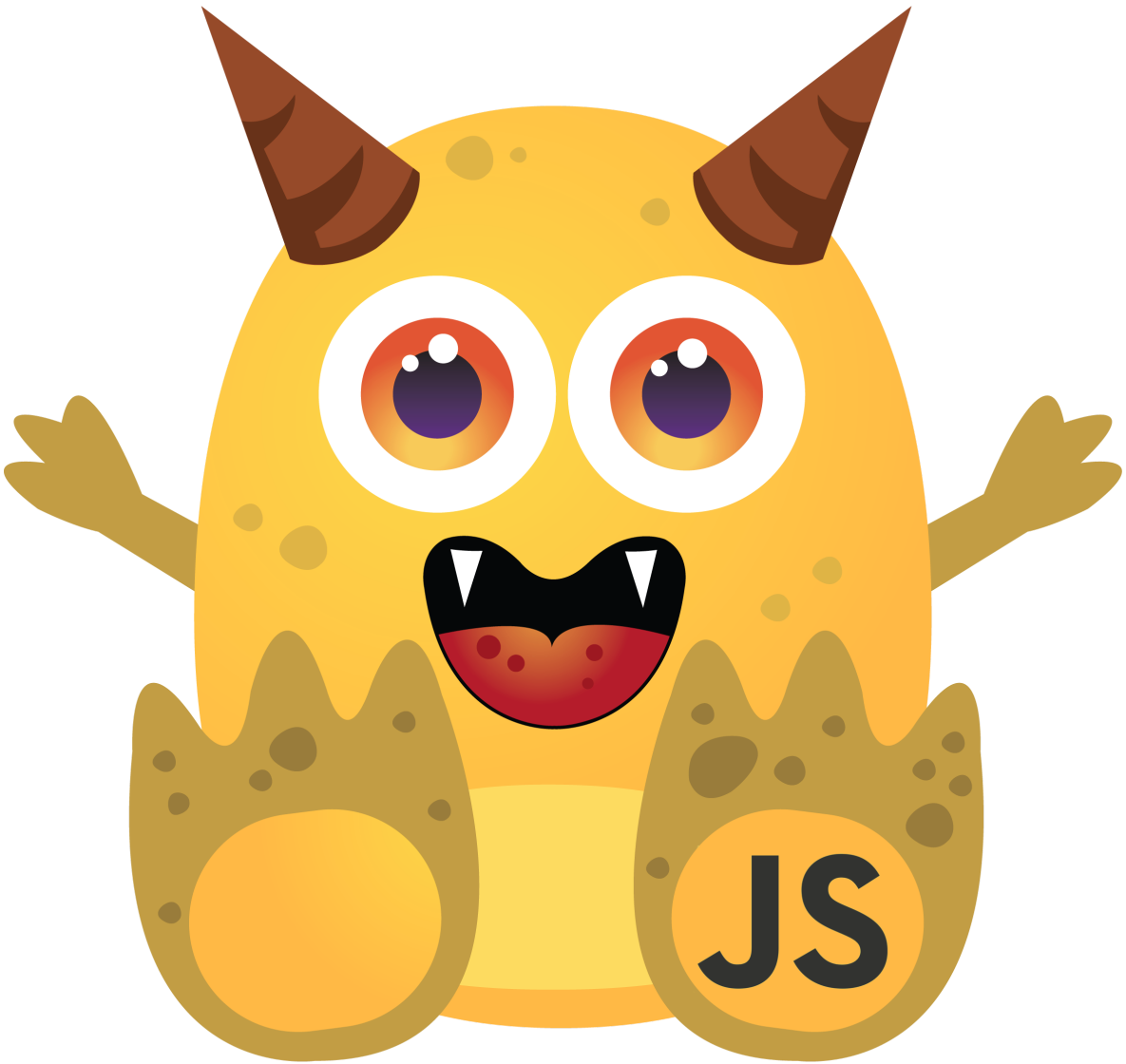
PROČ JAVASCRIPT?



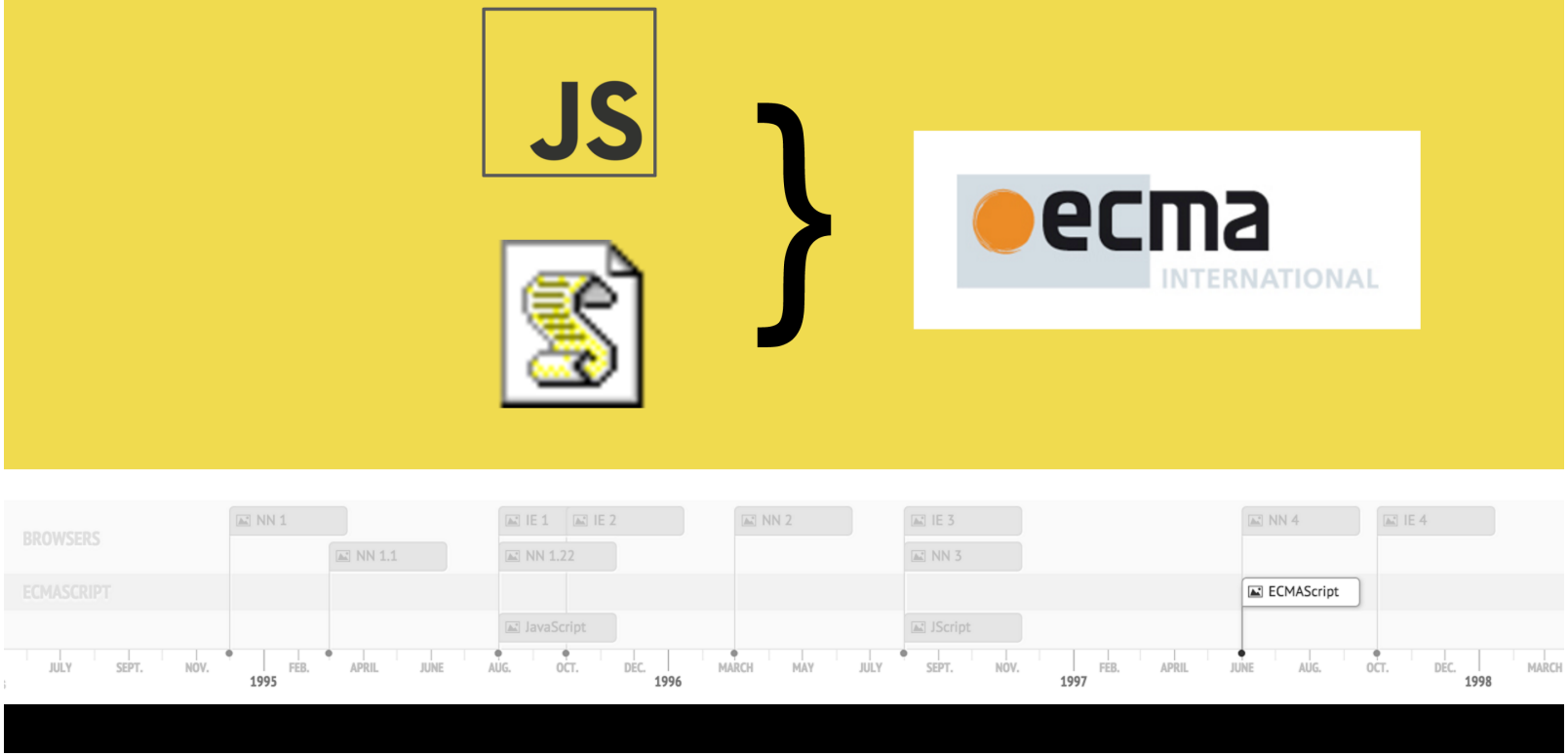
MOST POPULAR TECHNOLOGIES

2016 | 2015 | 2014 | 2013

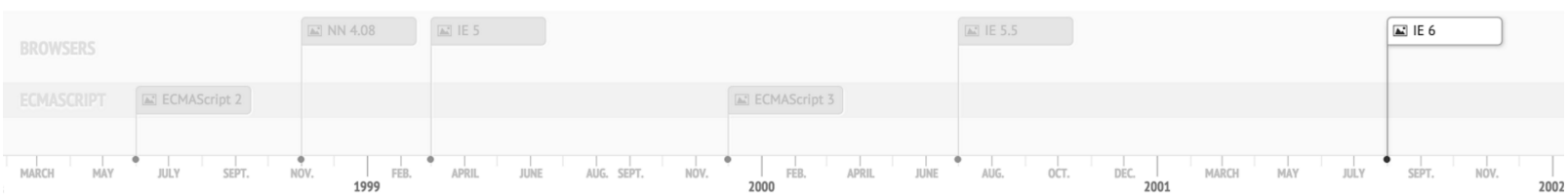



JS

Starověk


JS
}
ecma
INTERNATIONAL
BROWSERS
ECMASCRIPT
NN 1
NN 1.1
IE 1
IE 2
NN 1.22
NN 2
IE 3
NN 3
NN 4
IE 4
ECMAScript
JavaScript
JScript
JULY
SEPT.
NOV.
FEB.
APRIL
JUNE
AUG.
OCT.
DEC.
MARCH
MAY
1995
1996
1997
1998

Středověk

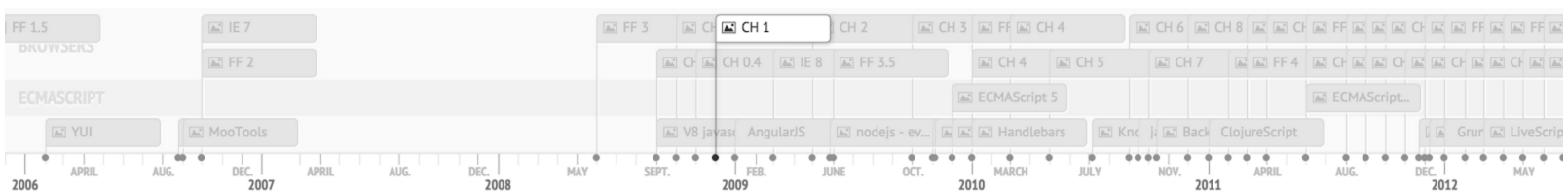

BROWSERS
NN 4.08
IE 5
IE 5.5
IE 6
ECMASCRIPT
ECMAScript 2
ECMAScript 3
MARCH MAY JULY SEPT. NOV. FEB. APRIL JUNE AUG. SEPT. NOV. FEB. APRIL JUNE AUG. OCT. DEC. MARCH MAY JULY SEPT. NOV.
1999
2000
2001
2002

Doba temna

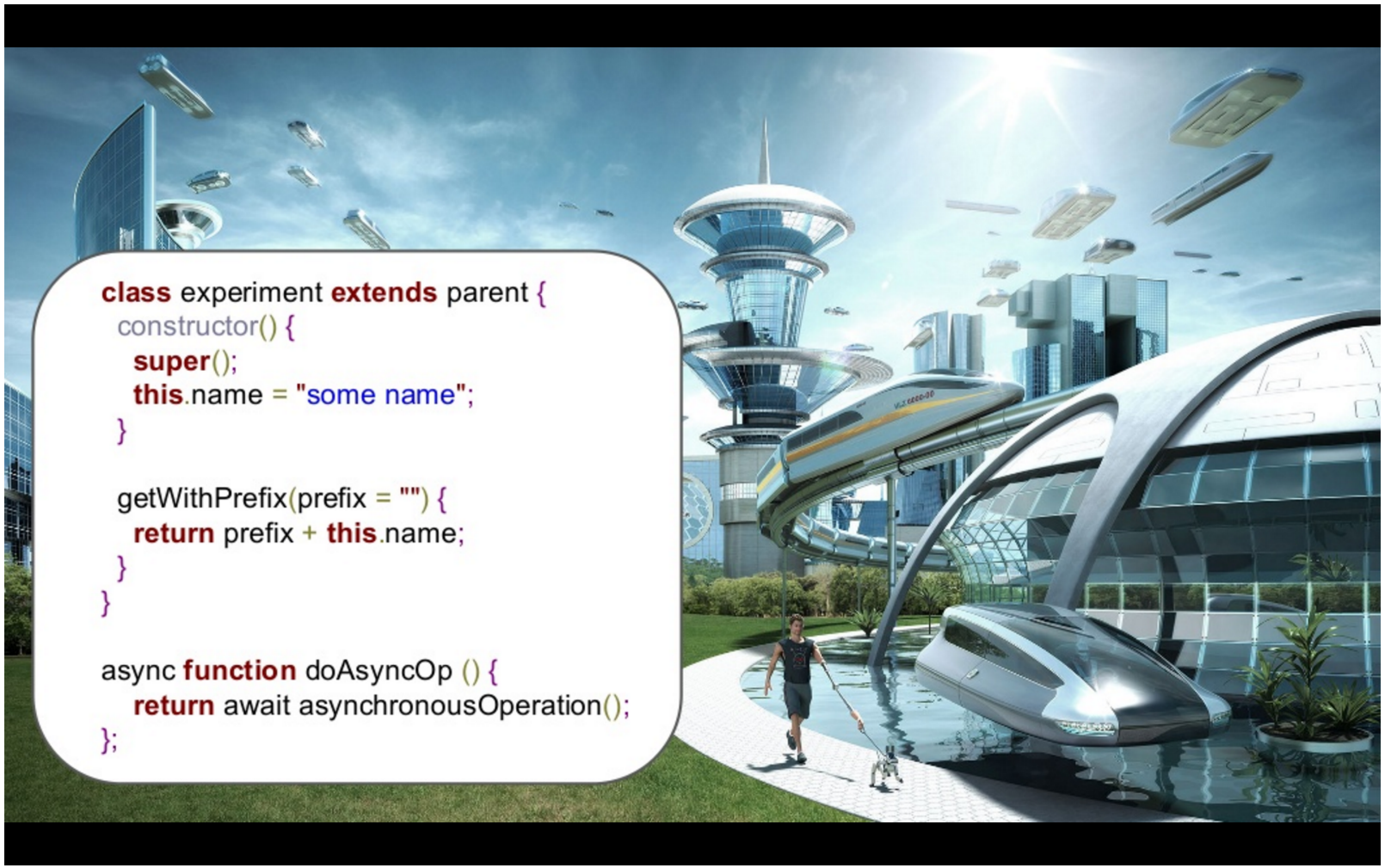
jQuery
write less, do more.

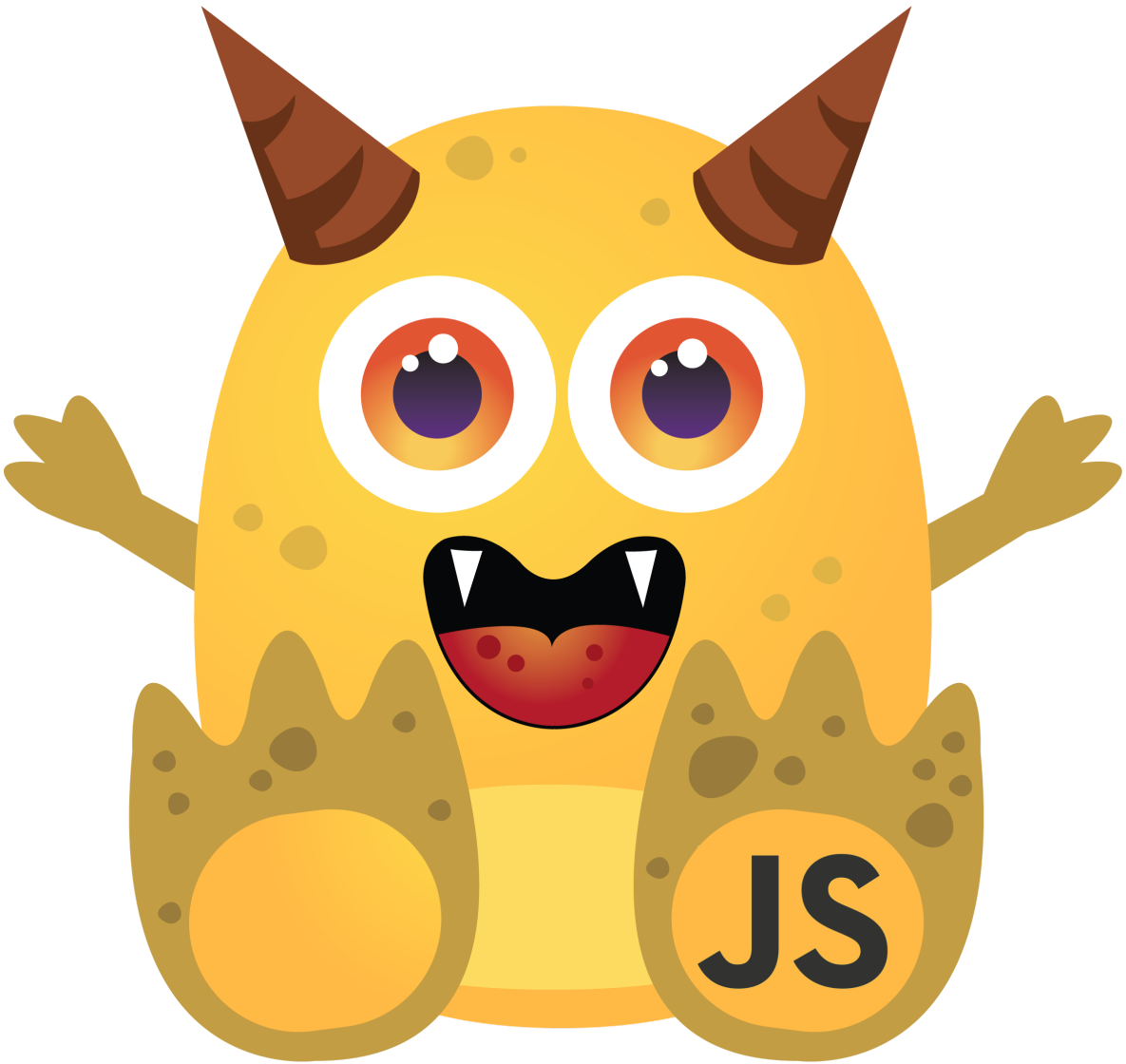
Novověk


chrome!
Firefox!
GLUE


FF 1.5
IE 7
FF 2
CH 1
CH 2
CH 3
CH 4
CH 0.4
IE 8
FF 3.5
CH 5
CH 7
FF 4
ECMAScript 5
YUI
MooTools
AngularJS
Handlebars
ClojureScript
2006
2007
2008
2009
2010
2011
2012

Století páry


class experiment extends parent {
constructor() {
super();
this.name = "some name";
}
getWithPrefix(prefix = "") {
return prefix + this.name;
}
}
async function doAsyncOp () {
return await asynchronousOperation();
};


JS

STRICT MODE

volitelně aktivovatelná omezenější varianta JS

```
'use strict';   // strict mode pro celý soubor

a = 11;         // ReferenceError: a is not defined


function sum(a, a, c) {   // SyntaxError:
  //    Strict mode function may not have duplicate parameter names

  return a + b + c;
}

var someNumber = 010;  // SyntaxError:
  //    Octal literals are not allowed in strict mode.
```

V čem se JS liší od ostatních jazyků?

FUNCTION

- funkce jsou objektovým typem
- každá funkce je instancí typu Function

function statement

```
function hello() {
    console.log('Hello!');
}

hello();
```

function expression

```
var hello2 = function() {
  console.log('Hello!');
};

hello2();
```

FUNCTIONS ARE FIRST-CLASS OBJECTS

```
var hello = function() {
  console.log('Hello!');
};

hello();    // 'Hello!'


var helloAlias = hello;
helloAlias();   // 'Hello!'

anotherFunction(hello);
```

```
var getFunction = function() {
  return function() {
    console.log('Hello function');
  };
};


var hello = getFunction();

hello();  // 'Hello function'
```

FUNCTION - NEEXISTUJE PŘETĚŽOVÁNÍ

```
function test(a, b) {
  ...
}

function test(a, b, c) {  // tato definice "přepíše" původní
  ...
}
```

FUNCTION - ARGUMENTS

```
function test(a, b) {
  console.log('a:', a);
  console.log('b:', b);
  console.log('arguments:', arguments);
}

test(1, 2);
//  a: 1
//  b: 2
//  arguments: [1, 2]
```

arguments je array-like objekt obsahující parametry předané funkci při volání

FUNCTION - PARAMETRY

```
test(1);
//  a: 1
//  b: undefined
//  arguments: [1]

test(1, 2, 3);
//  a: 1
//  b: 2
//  arguments: [1, 2, 3]
```

FUNCTION - DEFAULTNÍ HODNOTY

```
function test(a) {

  a = a || "defaultValue";


}
```

Boolean	true	false
String	libovolný neprázdný řetězec	""
Number	libovolné nenulové číslo	0, NaN
Object	libovolný objekt	null
Undefined	-	undefined

FUNCTION - DEFAULTNÍ HODNOTY

```
function test(a, b) {

  a = a || "defaultValue";

  b = typeof b !== 'undefined' ?  b : 1;

}
```

== vs ===

DEFAULTNÍ PARAMETRY

```
function pow(x, y = 2) {
  return Math.pow(x, y);
}

pow(3);     // 9

pow(3, 3);  // 27


var getDefault = function() { return 42;}

var fce = function(a, b = getDefault()) {
  // funkce getDefault je volána jen, pokud b je undefined
}
```

ES6

ARROW FUNCTIONS

```
var array = [1, 2, 3, 4, 5, 6, 7, 8, 9];

// ES5
array.map(function(number) {
  return 2 * number;
})


// ES6  - expression body
array.map(number => 2 * number);



// ES6  - statement body
array.map((number, index) => {
  if (index % 2 === 0) {
    return 2 * number;
  }

  return 3 * number;
});
```

OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

JS MÁ OBJEKTY

ALE NEMÁ TŘÍDY

OBJECT LITERAL

```
var dog = {
  name: 'Teggi',
  speak: function() {
    return "Haf";
  }
};
```

- vytváří instanci objektu
- jde o anonymní objekt

KONSTRUKTOR

```
var Dog = function(name) {
  this.name = name;

  this.speak = function() {
    return "Haf";
  };
}

var dog = new Dog("Teggi");
```

- konstrukční funkce
- název vždy velkým písmenem (konvence)
- voláme s operátorem new

KONSTRUKTOR

```
var Dog = function(name) {
  this.name = name;
  this.speak = function() {
    return "Haf";
  };
}
```

```
var dog1 = new Dog("Teggi");
var dog2 = new Dog("Rek");
var dog3 = new Dog("Sára");
var dog4 = new Dog("Zára");
var dog5 = new Dog("Dag");
...
```

každá instance bude mít vlastní metodu speak

```
dog1.speak === dog2.speak;   // false
dog1.speak === dog3.speak;   // false
dog1.speak === dog4.speak;   // false
dog1.speak === dog5.speak;   // false
```

PROTOTYPE

- vlastnost každé funkce
- obsahuje vlastnosti dostupné každé instanci odvozené z této funkce
- vlastnosti prototypu jsou společné všem instancím

KONSTRUKTOR + PROTOTYPE

```
var Dog = function(name) {
  this.name = name;
};

Dog.prototype.speak = function() {
  return "Haf";
};
```

```
var dog1 = new Dog("Teggi");
var dog2 = new Dog("Rek");
var dog3 = new Dog("Sára");

dog1.speak();     // "Haf"

dog1.speak === dog2.speak;   // true
dog1.speak === dog3.speak;   // true
```

PROTOTYPE CHAIN

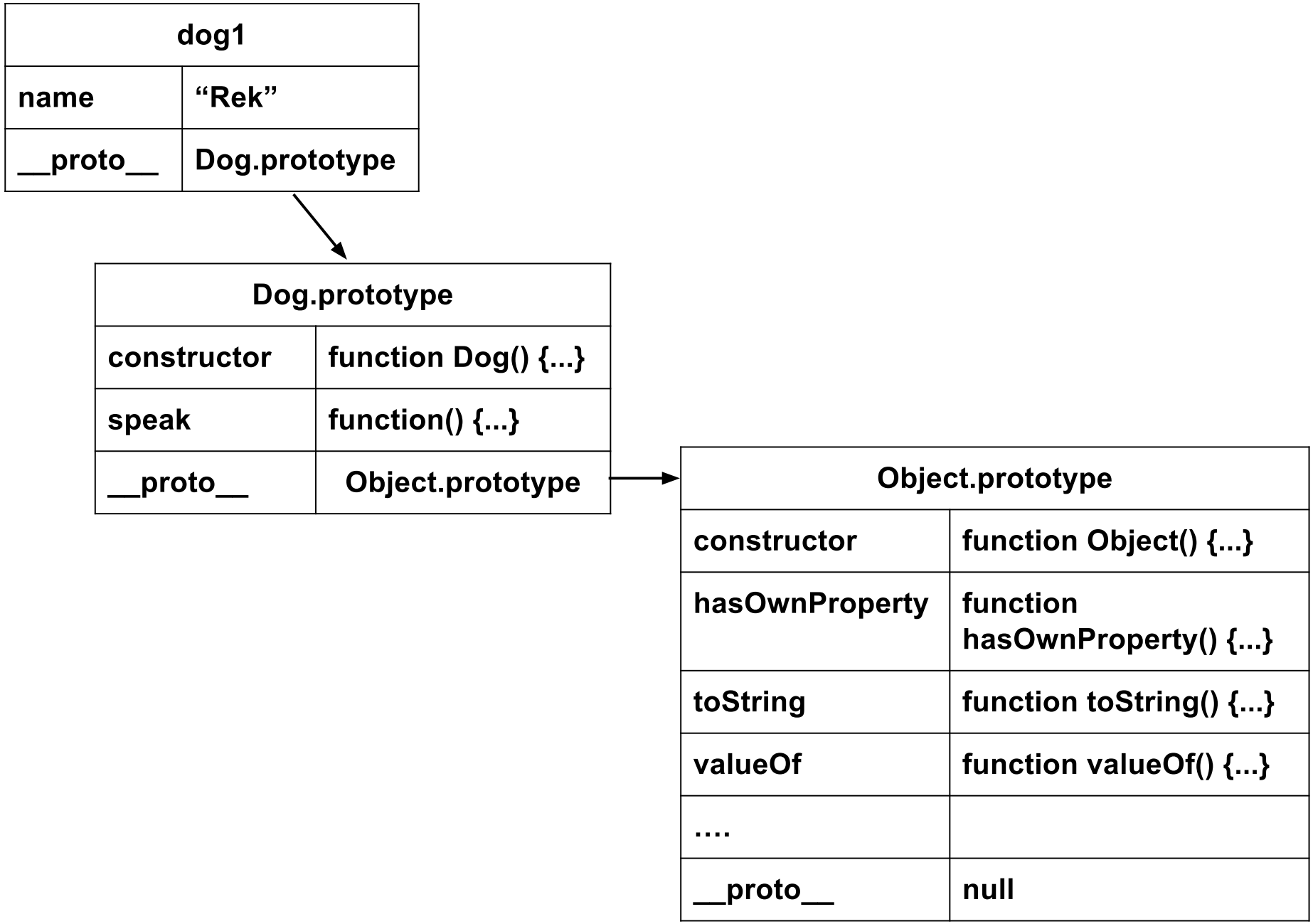
```
var Dog = function(name) {
  this.name = name;
};

Dog.prototype.speak = function() {
  return "Haf";
};

var dog1 = new Dog("Rek");
dog1 instanceof Dog   // true
dog1 instanceof Object // true

dog1            // Dog {name: "Rek"}

var p1 = Object.getPrototypeOf(dog1)    // Dog.prototype
var p2 = Object.getPrototypeOf(p2)      // Object.prototype
var p3 = Object.getPrototypeOf(p3)      // null
```


dog1
name
“Rek”
__proto__
Dog.prototype
Dog.prototype
constructor
function Dog() {...}
speak
function() {...}
__proto__
Object.prototype
Object.prototype
constructor
function Object() {...}
hasOwnProperty
function hasOwnProperty() {...}
toString
function toString() {...}
valueOf
function valueOf() {...}
….
__proto__
null

CLASS

```
class Dog {

  constructor(name) {
    this.name = name
  }

  speak() {
    return "Haf";
  }
}

var dog1 = new Dog("Rek");
```

class je jen syntax sugar pro konstrukční funkce a prototype!

DĚDIČNOST

- v JS používáme prototypovou dědičnost
- zřetězení prototypů od potomka k rodiči/rodičům

DĚDIČNOST

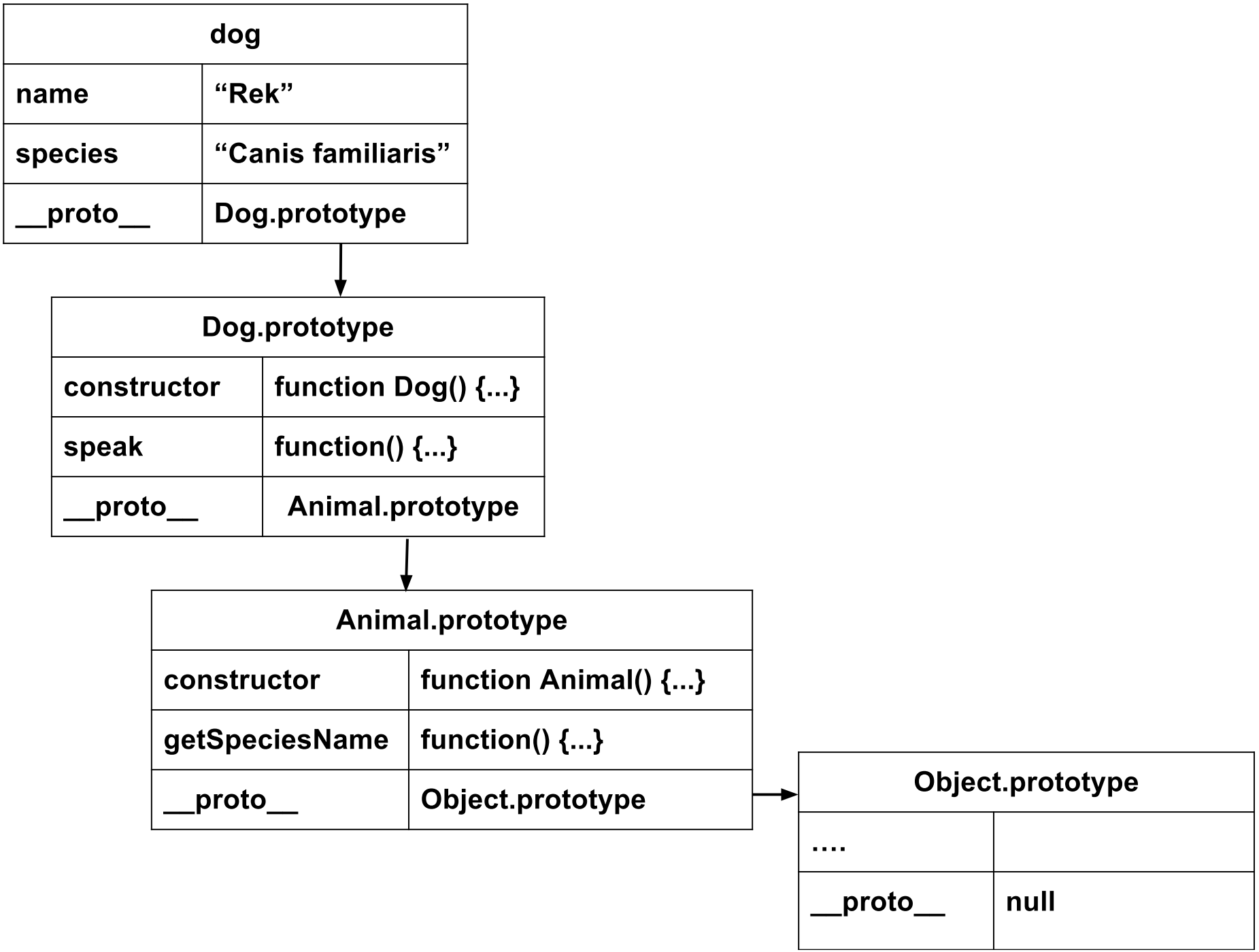
```
var Animal = function(species) {
  this.species = species;
}

Animal.prototype.getSpeciesName = function() {
  return this.species;
}
```

```
var Dog = function(name) {
  Animal.call(this, "Canis familiaris");
  this.name = name;
};

Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;

Dog.prototype.speak = function() {
  return "Haf";
};
```


dog
name
“Rek”
species
“Canis familiaris”
__proto__
Dog.prototype
Dog.prototype
constructor
function Dog() {...}
speak
function() {...}
__proto__
Animal.prototype
Animal.prototype
constructor
function Animal() {...}
getSpeciesName
function() {...}
__proto__
Object.prototype
Object.prototype
....
__proto__
null

EXTENDS

```
class Animal {

  constructor(species) {
    this.species = species;
  }

  getSpeciesName() {
    return this.species;
  }
}
```

ES6

```
class Dog extends Animal {

  constructor(name) {
    super();
    this.name = name;
  }

  speak() {
    return "Haf";
  }
}
```

class a extends je jen syntax sugar pro konstrukční funkce a prototypovou dědičnost!

MODIFIKÁTORY PŘÍSTUPU

JS NEMÁ

- v JS je vše public
- vše můžete přepsat

  i vlasnosti nativních objektů

OBJECT + CLOSURE

```
var dogFactory = function(dogName) {
  var name = dogName;

  function speak() {
    return "Haf";
  };

  return {
    getName: function() {
      return name;
    },
    speak: speak
  }
}

var dog = dogFactory("Teggi");
```

- není to constructor, ale factory
- volá se bez new, nefunguje instanceOf
- využívá closure

CLOSURE

```
var genFrom10 = generator(10);
genFrom10()   // 10
genFrom10()   // 11
genFrom10()   // 12
```

```
var generator = function(initialNumber) {
  var current = initialNumber;

  return function () {
    return current++;
  };

};
```

OBJECT + CLOSURE

```
var dogFactory = function(dogName) {
  var name = dogName;

  function speak() {
    return "Haf";
  };

  return {
    getName: function() {
      return name;
    },
    speak: speak
  }
}

var dog = dogFactory("Teggi");
```

Nepoužívejte closure jako náhradu private.

CLOSURE A ITERACE

```
function printInSecond() {

  var names = [ 'Rek', 'Jessie', 'Teggi', 'Bára', 'Rita'];

  for (var i = 0; i < names.length; i++) {
    var name = names[i];
    setTimeout(function() {
      console.log(name);
    }, 1000);
  }
}

printInSecond();

//  Rita
//  Rita
//  Rita
//  ...
```

CLOSURE A ITERACE

```
function printInSecond() {
  var names, i, name;  // hoisting

  names = [ 'Rek', 'Jessie', 'Teggi', 'Bára', 'Rita'];

  for (i = 0; i < names.length; i++) {
    name = names[i];
    setTimeout(function() {
      console.log(name);
    }, 1000);
  }
}

printInSecond();

//  Rita
//  Rita
//  Rita
//  ...
```

CLOSURE A ITERACE

```
function makeLogTimeout(name) {
  setTimeout(function() {
      console.log(name);
    }, 1000);
 }

function printInSecond() {
  ...

  for (var i = 0; i < names.length; i++) {
    var name = names[i];
    setTimeout(makeLogTimeout(name), 1000);
  }
}

printInSecond();

//  Rek
//  Jessie
//  Teggi
//  ...
```

LET & CONST

```
function letTest() {
  let x = 31;

  if (true) {
    let x = 71;       // uvnitř bloku je x jiná proměnná
    console.log(x);   // 71
  }

  const AAA = 12;
  AAA = 1;            // SyntaxError
}
```

ES6

LET A ITERACE

```
function printInSecond() {

  var names = [ 'Rek', 'Jessie', 'Teggi', 'Bára', 'Rita'];

  for (var i = 0; i < names.length; i++) {
    let name = names[i];
    setTimeout(function() {
      console.log(name);
    }, 1000);
  }
}

printInSecond();

//  Rek
//  Jessie
//  Teggi
//  ...
```

THIS V JS

THIS A OBJEKT

```
var object = {
  prop: 42,
  method: function() {
    console.log(this.prop);
  }
};

console.log(object.method());    // 42
```

THIS A OBJEKT

```
function independent() {
  console.log(this.prop);
}

var object = {
  prop: 42,
};

object.method = independent;

console.log(object.method());    // 42
```

This je objekt na kterém je funkce volaná

- Vazba na this není danná definicí, ale pouze voláním.
- Proto nezáleží na tom, jestli funkce vznikla v rámci objektu, nebo jak k němu byla přiřazena.

THIS V ASYNCHRONNÍCH OPERACÍCH

```
var object = {
  prop: 42,
  method: function() {
    console.log(this.prop);
  }
};

setTimeout(object.method, 1000);

// po jedné vteřině vypíše "undefined"
```

```
setTimeout(function() {
  object.method();
}, 1000);
```

```
setTimeout(object.method.bind(object), 1000);

//  fce.bind(thisArg[, arg1[, arg2[, ...]]])  : Function
```

FUNKCE CALL A APPLY

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

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

```
function showThisAndArgs() {
  console.log("this:", this);
  console.log("arguments:", arguments);
}

showThisAndArgs(1, 2, 3)

showThisAndArgs.call({a:1}, 1, 2, 3, 4)

showThisAndArgs.apply({b:1}, [1, 2, 3, 4])
```

FUNKCE APPLY

```
function originalFunction(has, some, args) {
    ...
}
```

```
function myFunction(has, some, args) {
  console.log(arguments)

  originalFunction(has, some, args);
}
```

```
function myFunction() {
  console.log(arguments)

  originalFunction.apply(null, arguments);
}
```

FUNKCE BIND

```
function myLog(message, data) {
  console.log("myLog:", message, data);
}

var myLog = console.log.bind(console, "myLog:");

myLog('Ahoj Ostravo!!!');    // myLog: Ahoj Ostravo!!!
```

DATOVÉ TYPY

Primitivní

- Undefined
- Null
- Boolean
- Number
- String

Objektové

- Object
- Array
- Date
- Regexp
- Function

PROBLÉMY S ČÍSLY

OPERÁTORY A DYNAMICKÉ TYPY

Operátor sčítání / operátor zřetězení

```
"Angular" + ".cz" = "Angular.cz"

35 + 7 = 42
```

Význam operátoru závisí na kontextu

```
 1  + "1" = "11"
"1" +  1  = "11"
```

Pokud je jeden z operandů string, jedná se o zřetězení

NUMBER - SPECIÁLNÍ HODNOTY

```
"Ostrava " + 2016   // "Ostrava 2016"
```

```
"Ostrava " - 2016   // NaN
"Ostrava " * 2016   // NaN
"Ostrava " / 2016   // NaN
```

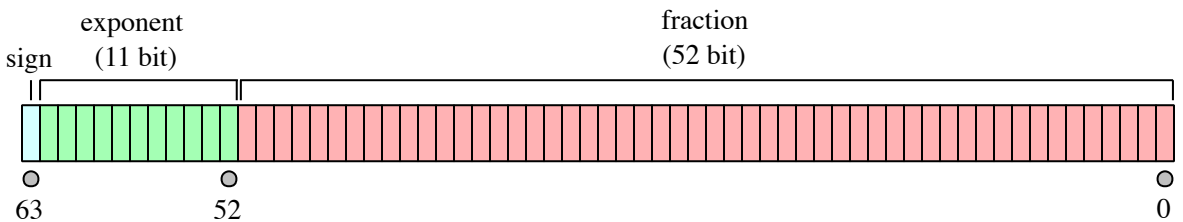
NaN - Not a Number.

```
10 / 0  // Infinity
```

Dělení nulou nevyvolá žádnou chybu.

NUMBER

IEEE 754



$$\text{sign} * \text{fraction} * 2^{\text{exponent}}$$

$$2 => +1 * 1 * 2^{1}$$

$$3 => +1 * 1.5 * 2^{1}$$

$$0.3 => +1 * 1.2000000476837158 * 2^{-2}$$

$$= 0.30000001192092896$$

```
var number = 0.2 + 0.1;   //  0.30000....

number.toFixed(2);        // "0.30"
number.toExponential(2);  // "3.00e-1"
```

```
number === 0.3                    // false

(number - 0.3) < Number.EPSILON   // true
```

ES6

```
(function () {

  /* tady je kód */

})();
```

IIFE

Immediately-invoked function expression

```
(function () {

  /* tady je kód */

})();
```

využití scope k izolaci kódu

IIFE

Immediately-invoked function expression

```
// tento kód špiní globální prostor
var a = 1;    // window.a
```

IIFE

Immediately-invoked function expression

```
(function(App){

  var Display = function () {
    ...
  }

  Display.prototype.show = function () {
    ...
  }

  App.services = {};
  App.services.Display = Display;

}(AppNamespace));
```

MODULY

```
//------ lib.js ------

  export const PI = 3.14159;

  export function square(x) {
      return x * x;
  }
```


```
//------ main.js ------

  import { PI, square } from 'lib';

  console.log(PI);           // 3.14159
  console.log(square(11));   // 121
```

jQuery Prototype Underscore
Lodash

POTŘEBUJEME JEŠTĚ JQUERY?

- $('#id');
- $(el).parent();
- $(el).attr('tabindex');
- $(el).html();
- $(el).text();
- $(el).on(e, fce);
- $.parseJSON(string);
- $.trim(string);

- document.querySelector('#id');
- el.parentNode
- el.getAttribute('tabindex');
- el.innerHTML
- el.textContent
- el.addEventListener(e, fce);
- JSON.parse(string);
- string.trim();

POTŘEBUJEME JEŠTĚ ...?

youMightNotNeedjquery.com

reindex.io/blog/you-might-not-need-underscore

github.com/cht8687/You-Dont-Need-Lodash-Underscore

ES6 / ES2015

- Částečná podpora v prohlížečích
- transpilery

JAK ZAČÍT?

cli

- nodejs

na webu

- console v browseru
- jsbin.com,webpackbin.com

KNIHOVNY NEBO FRAMEWORK?

Děkujeme za pozornost

ANGULAR.CZ/INFS2