Tópicos Especiais

Professor: Raphael Magalhães Hoed

Javascript

Januária – MG, 26/01/2017

O que é Javascript (1)?

O que é Javascript (2)?

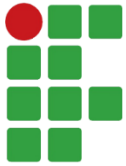
- Criada pela Netscape em 1995
- A princípio chamada de LiveScript
- Recebeu colaboração da Sun
- Foi projetada inicialmente para ser uma linguagem de Script.
- Não requer compilador.
- Não é uma linguagem computacionalmente autosuficiente.
- A organização Ecma padronizou a linguagem Javascript
- Javascript = Ecmascript

O que é Javascript (3)?

```
<form>
   <input type="button"
          value="Close"
          onClick="window.close();">
</form>
```

O que é Javascript (4)?

```
<form>
   <input type="button" value="mensagem" onClick="window.alert('Hello world!')";>
</form>
```



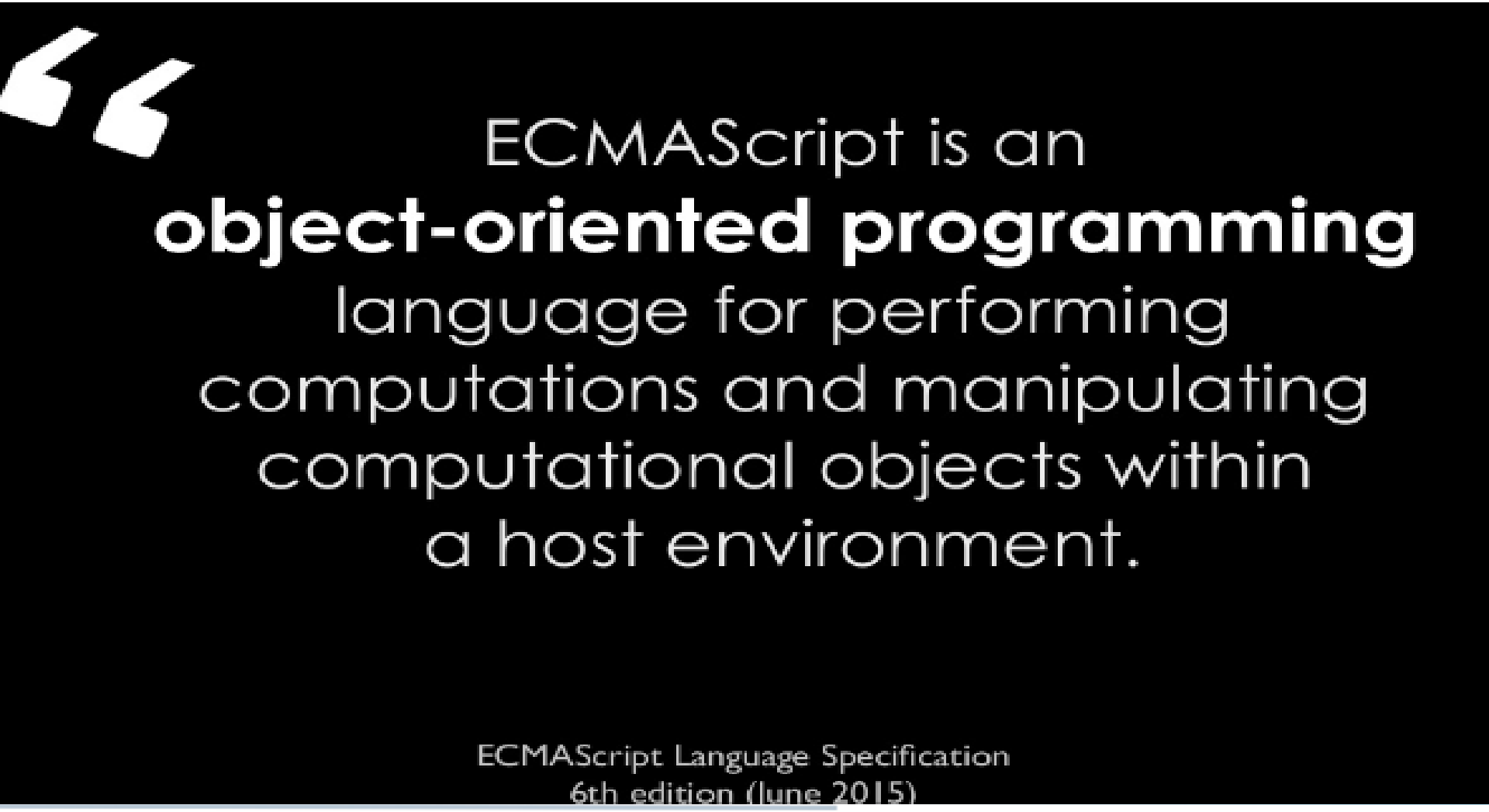
O que é Javascript (5)?

```
<script>
    function calcular(){
        numero1=document.getElementById("numero1").value;
        numero2=document.getElementById("numero2").value;
        numero1= parseFloat(numero1);
        numero2=parseFloat(numero2);
        resultado = numero1 + numero2;
        document.getElementById("resultado").value = resultado;
    }
</script>
<form>
  Número 1: <input type="text" id="numero1"> <br>
  Número 2:<input type="text" id="numero2">  <br>
  Resultado: <input type="text" id="resultado">
  <input type="button" value="calcular" onClick="calcular();">
</form>
```

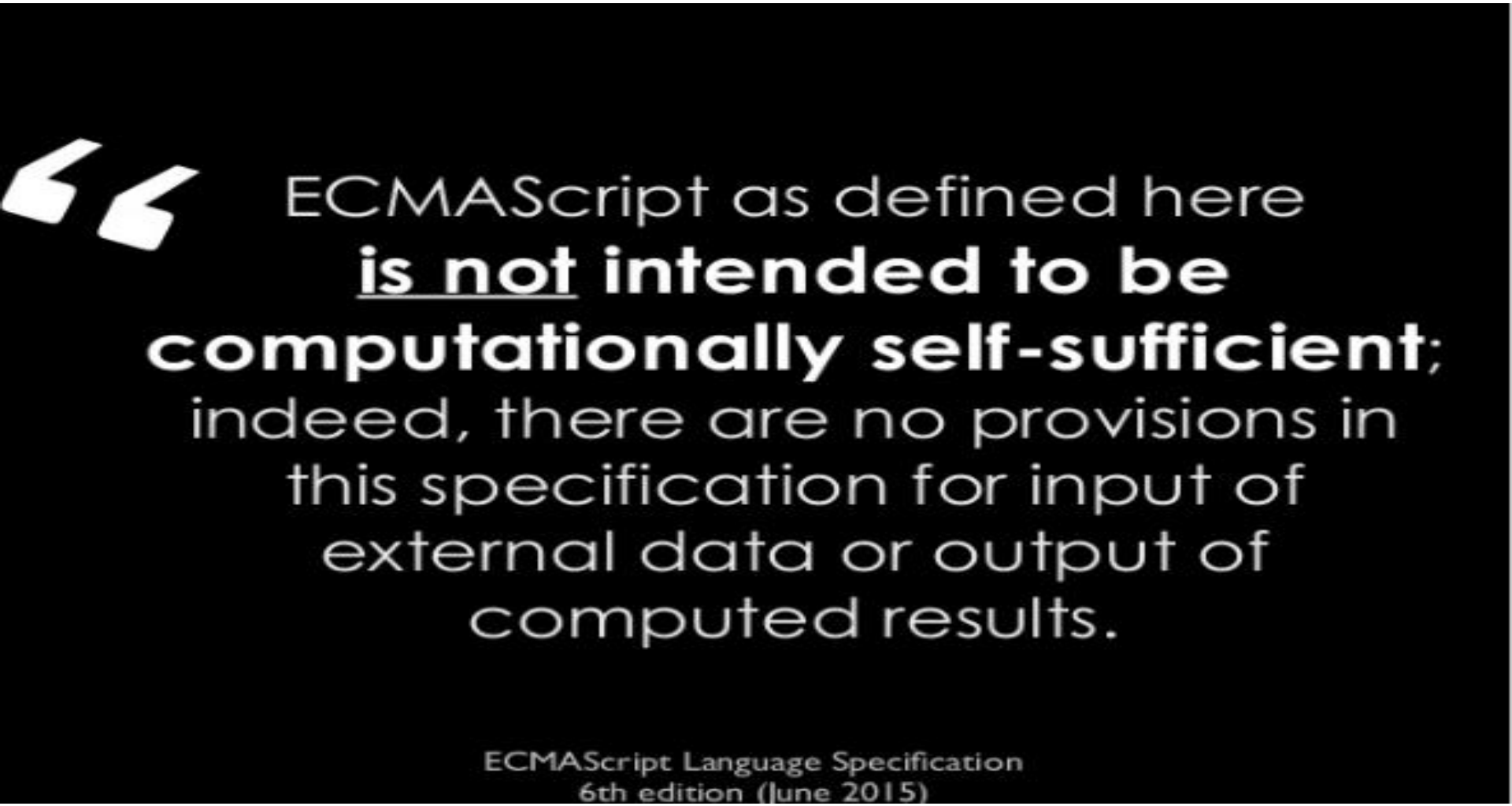
O que é Javascript (6)?

O que é Javascript (7)?

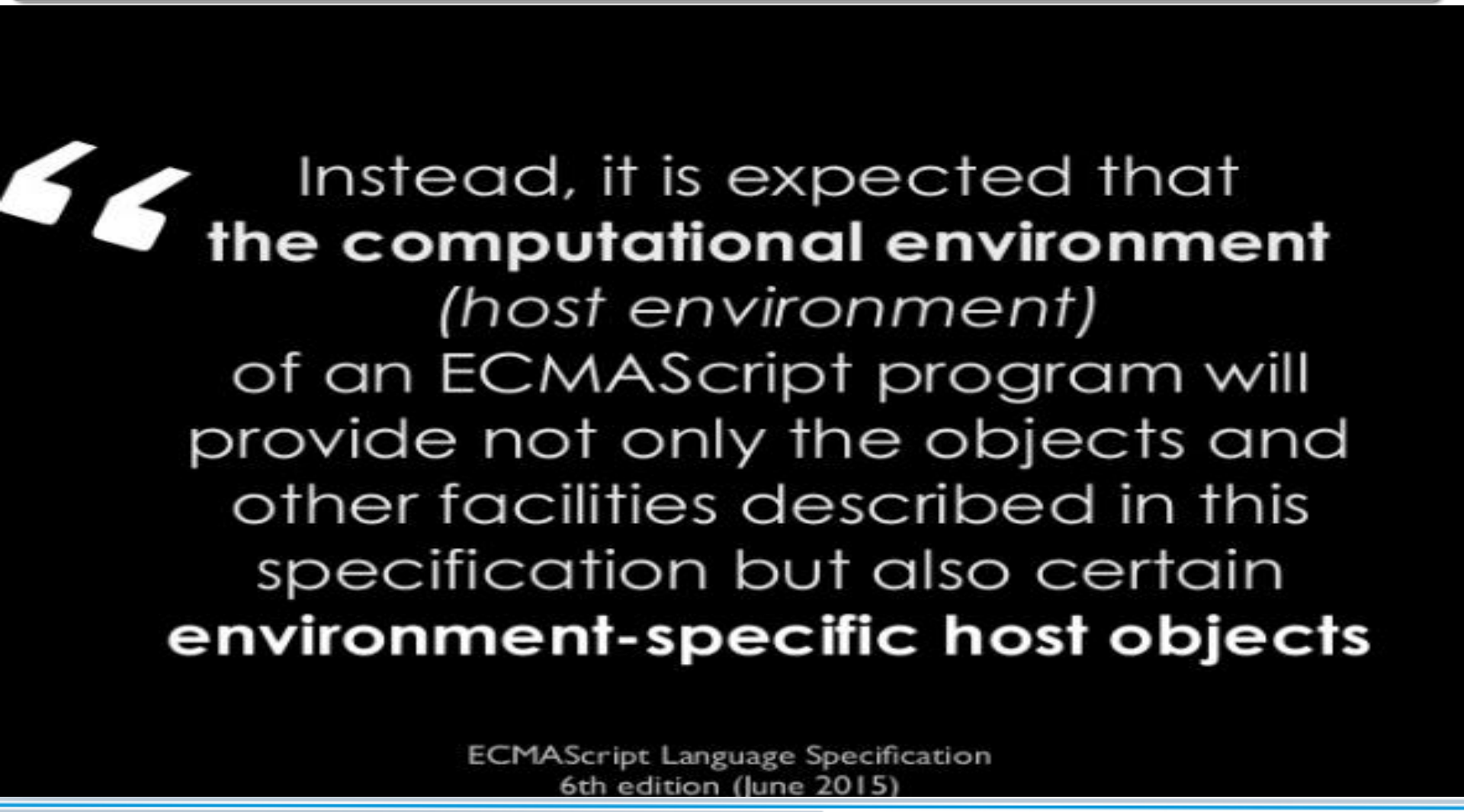
O que é Javascript (8)?



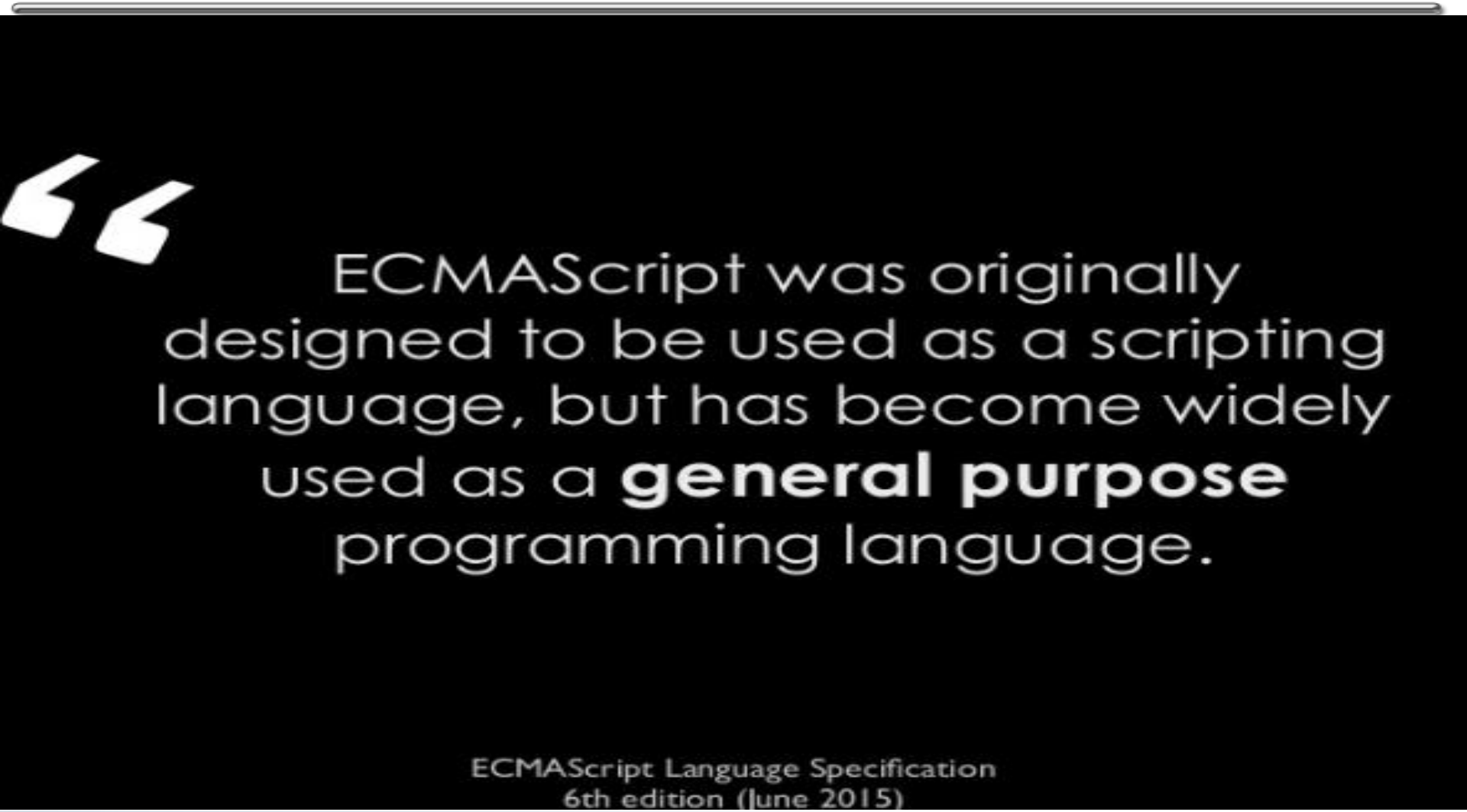
O que é Javascript (9)?



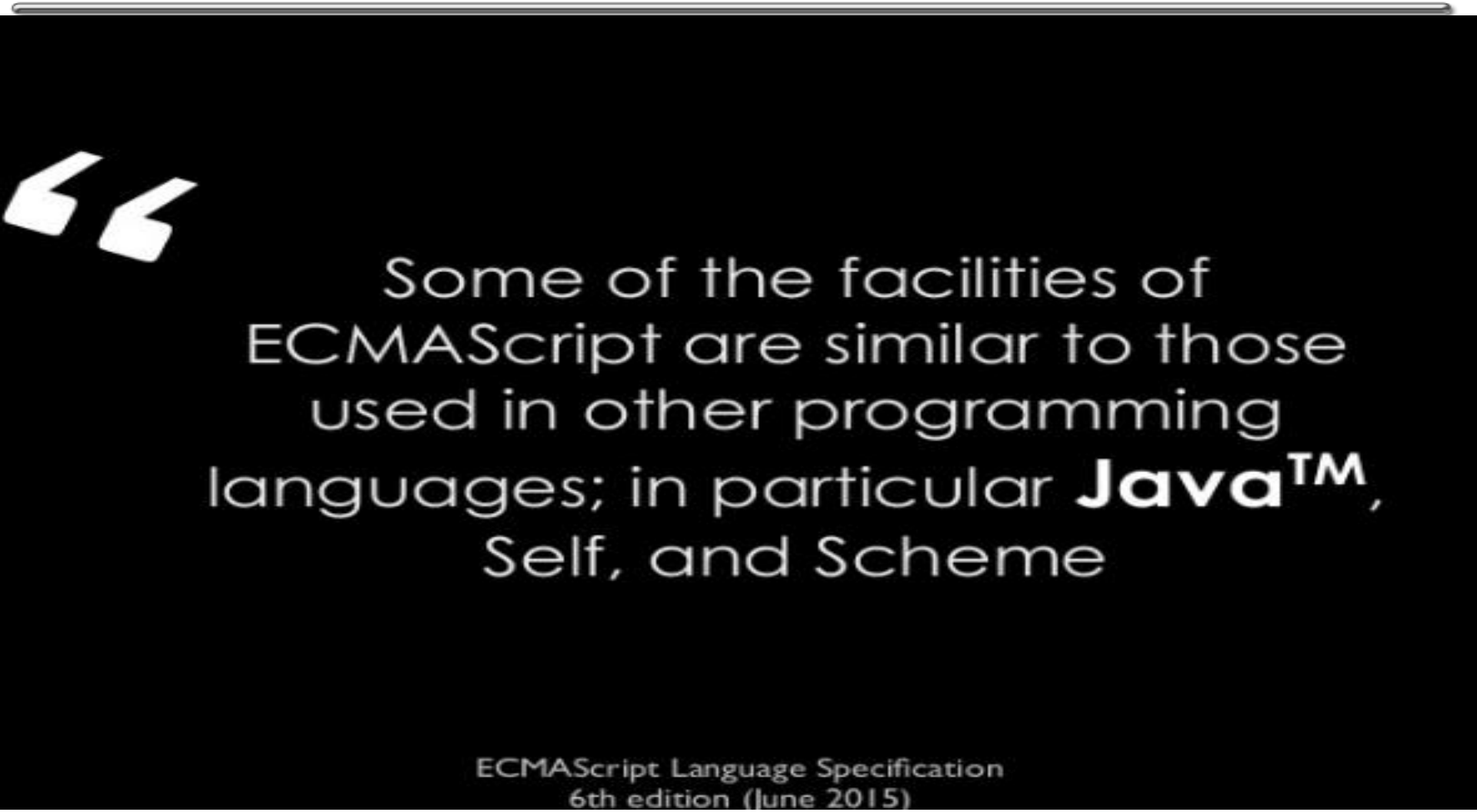
O que é Javascript (10)?



O que é Javascript (11)?



O que é Javascript (12)?



O que é Javascript (13)?

> A web browser provides an ECMAScript host environment for client-side computation including, for instance, objects that represent **windows**, **menus**, **pop-ups**, **dialog boxes**, **text areas**, **anchors**, **frames**, **history**, **cookies**, and **input/output**.

```
navigator.appName;
window.moveTo(100,100);
```

ECMAScript Language Specification
6th edition (June 2015)

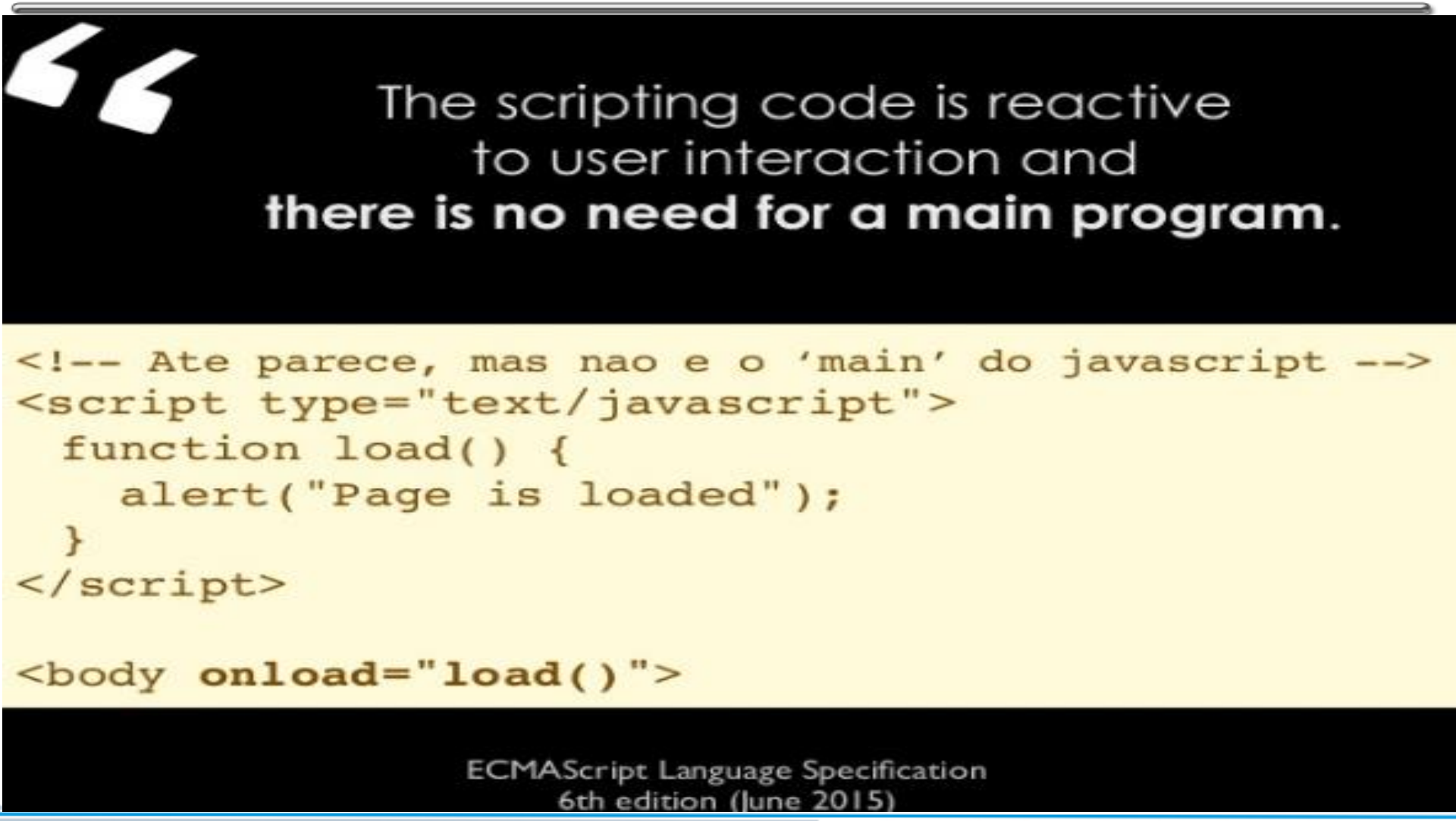
O que é Javascript (14)?

> Further, the host environment provides a means to attach scripting code to events such as **change of focus**, **page** and **image loading**, **unloading**, **error** and **abort**, **selection**, **form submission**, and **mouse actions**.

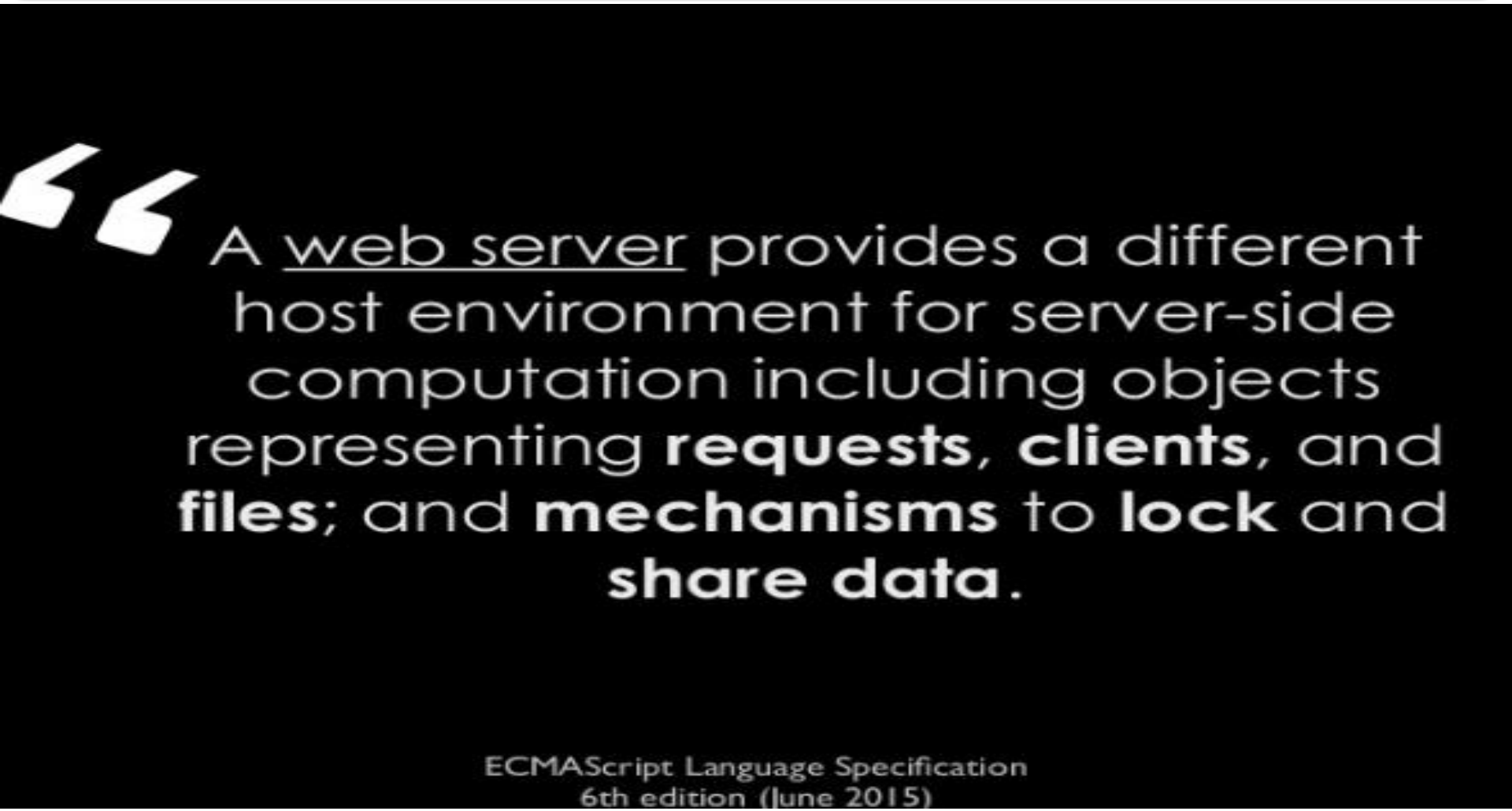
```
<button type="button" onclick="displayDate()">
    Display Date
</button>
```

ECMAScript Language Specification
6th edition (June 2015)

O que é Javascript (15)?

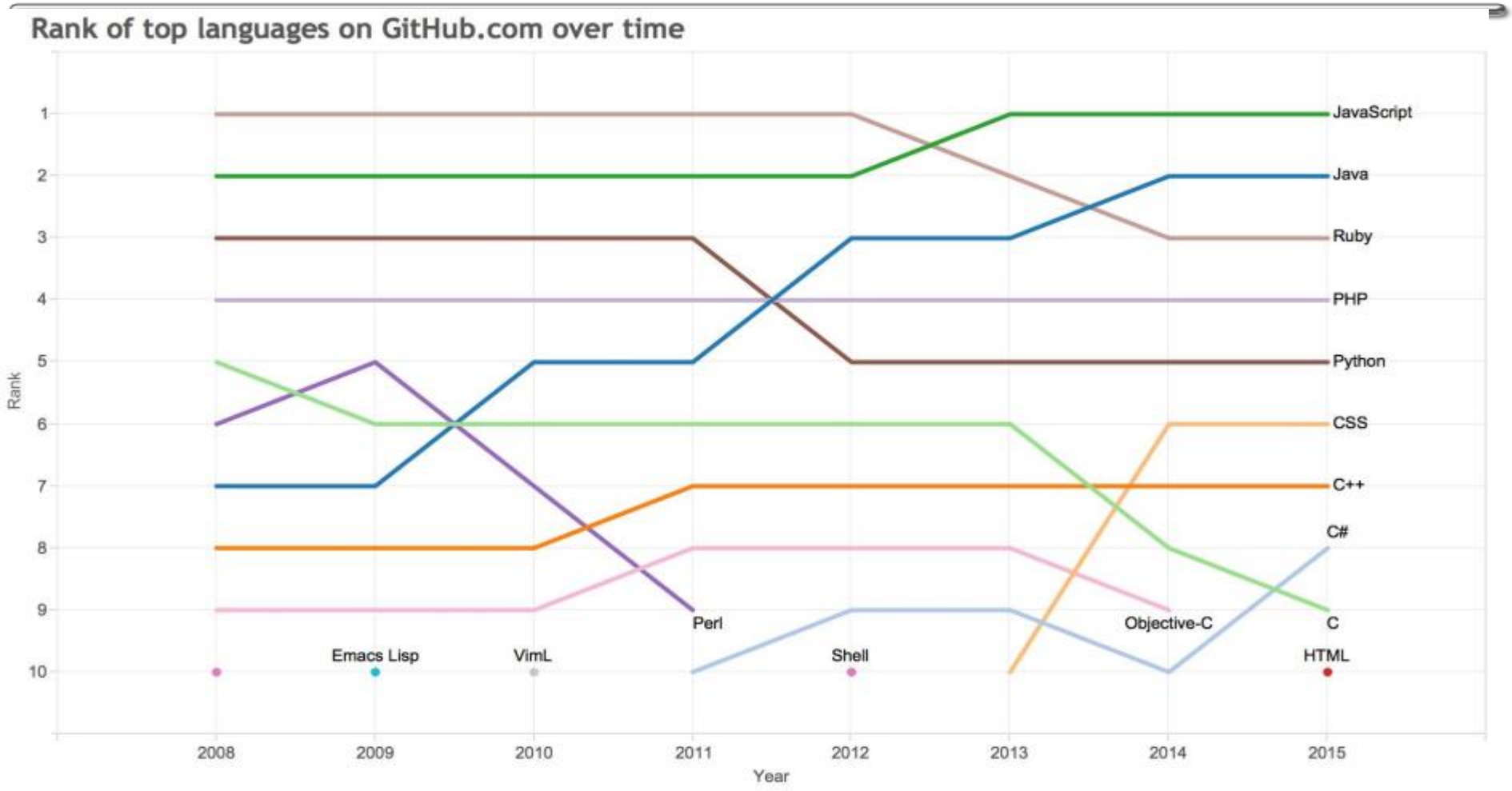


O que é Javascript (16)?



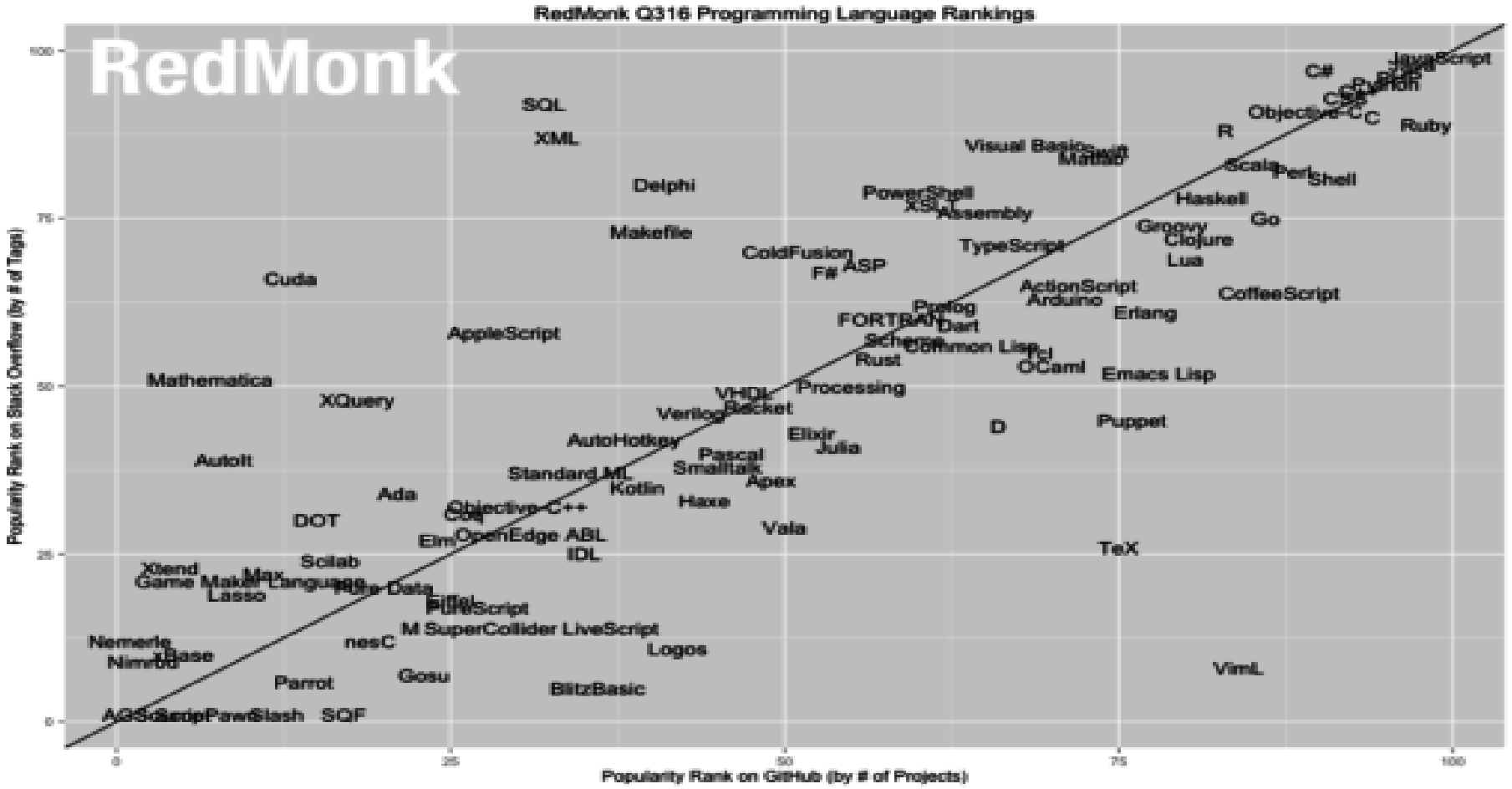
O que é Javascript (17)?

Linguagens mais populares no GITHUB (1)



Fonte: http://devblog.drall.com.br/linguagens-de-programacao-mais-comuns-do-github-atraves-dos-anos/

Linguagens mais populares no GITHUB (2)



Fonte: http://redmonk.com/sogrady/2016/07/20/language-rankings-6-16/

Objetos Javascript (1)

- Objetos da linguagem.
- Objetos fornecidos pelo Host Environment
- Objetos criados pelo usuário

Objetos Javascript (2)

- Objetos do navegador:

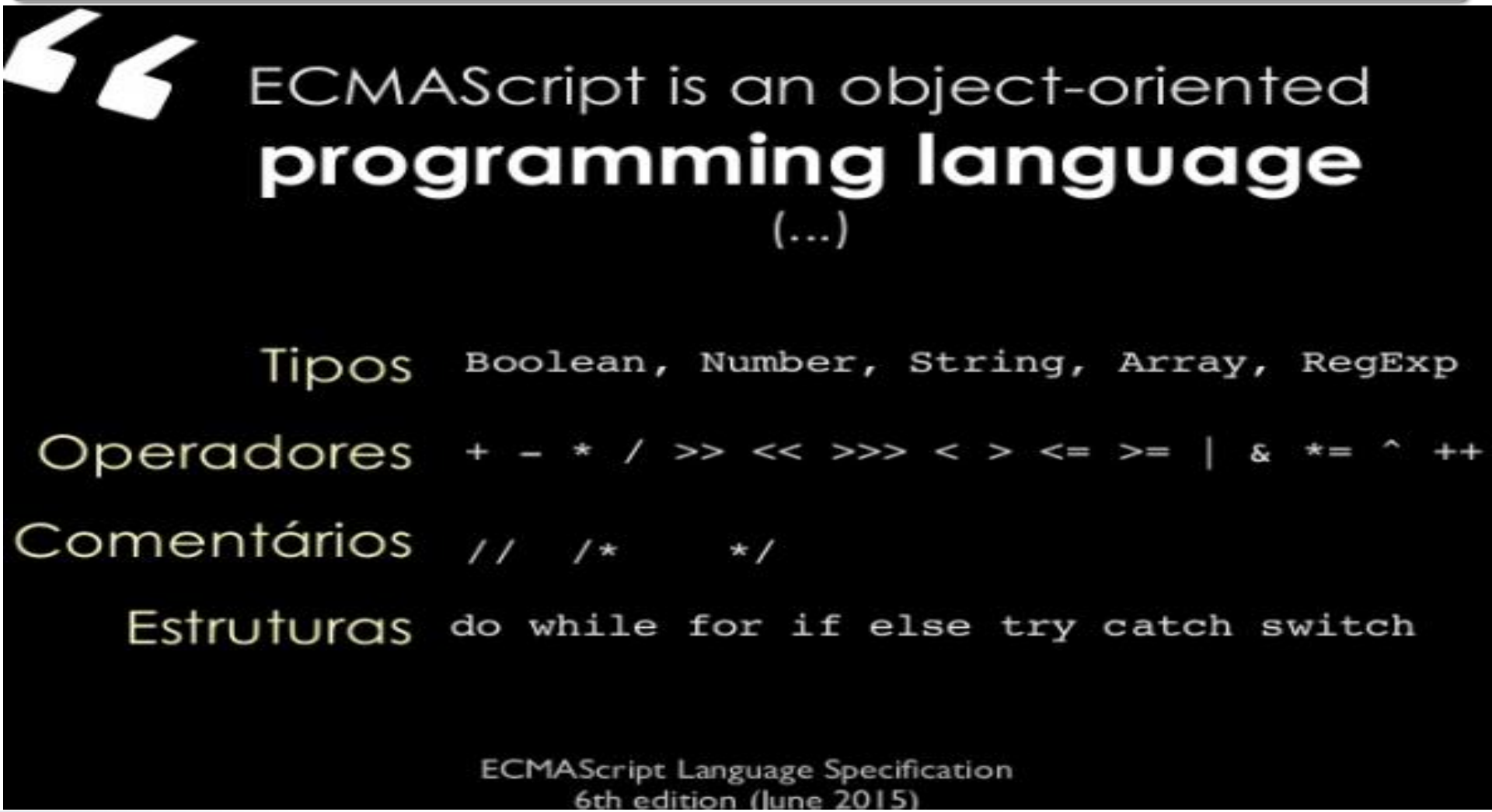
  - Document Object Model (DOM)

  - No Início, cada navegador criou seu DOM particular (IE e Netscape).

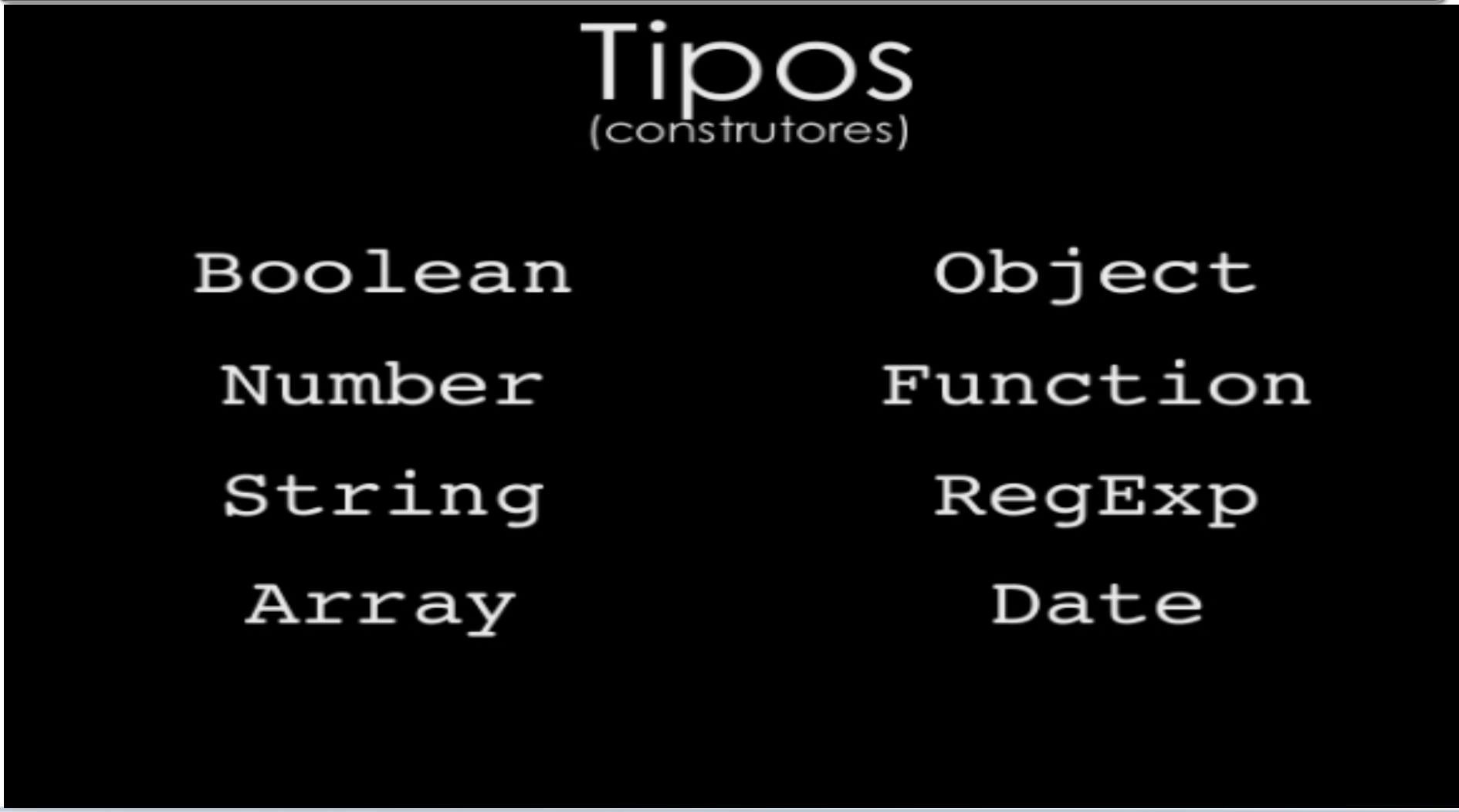
  - W3C iniciou trabalho de padronização do DOM

  - Ainda existem incompatibilidades.

Estrutura da linguagem (1)



Estrutura da linguagem (2)



Estrutura da linguagem (3)

```
<script>
    function mensagem(){
        var x;
        alert(x);
    }
</script>
<form>
   <input type="button" value="mensagem" onClick="mensagem()";>
</form>
```

- O que será exibido na caixa de alerta?

Estrutura da linguagem (4)

```
<script>
    function mensagem(){
        var x = null;
        alert(x);
    }
</script>
<form>
   <input type="button" value="mensagem" onClick="mensagem()";>
</form>
```

- O que será exibido na caixa de alerta?

Estrutura da linguagem (5)

Boolean

```
var x = true;
if(x) {
 alert('Verdadeiro');
}
```

Obs: 0 e null equivalem a false

ECMAScript Language Specification
6th edition (June 2015)

Estrutura da linguagem (6)

Number

```
var x = 10;
var y = 15;
alert(x+y);
```


```
var x = 10.1;
var y = 15.2;
alert(x+y);
```

ECMAScript Language Specification
6th edition (June 2015)

Estrutura da linguagem (7)

STRING

```
<form>
   <input type="button" value="mensagem" onClick="alert('RAPHAEL')";>
</form>
```

Estrutura da linguagem (8)

FUNCTION

```
<script>
    function mensagem(){
        alert('TESTE');
    }
</script>
<form>
    <input type="button" value="mensagem" onClick="mensagem()";>
</form>
```

Estrutura da linguagem (9)

CONSTRUTORES

```
<script>
        var x = false;
        alert(x.constructor);
</script>
```

Estrutura da linguagem (10)

CONSTRUTORES

```
<script>
        var x = "Raphael";
        alert(x.constructor);
</script>
```

Estrutura da linguagem (11)

CONSTRUTORES

```
<script>
        var x = 3.73;
        alert(x.constructor);
</script>
```

Estrutura da linguagem (12)

CONSTRUTORES

```
<script>
    var x = function (){alert("Professor Raphael")};
    alert(x.constructor);
</script>
```

Estrutura da linguagem (13)

CONSTRUTORES

```
var x = new Boolean(true);
if(x) { alert('Verdadeiro'); }
```

```
var x = new String("Alexandre");
alert(x);
```

```
var x = new Number(10);
var y = new Number(15);
alert(x+y);
```

Estrutura da linguagem (14)

OPERADOR typeof

```
<script>
    alert(typeof 1);
</script>
```

Estrutura da linguagem (15)

OPERADOR ++ e --

```
<script>
    var x = 2;
    ++x;
    alert (x);
</script>
```

```
<script>
    var x = 2;
    --x;
    alert (x);
</script>
```

Estrutura da linguagem (16)

instanceof

```
<script>
    var x = new Number(2);
    alert (x instanceof String);
</script>
```

Estrutura da linguagem (17)

Operadores ==, !=, ===, !==

```
<script>
    var x="2";
    if (x==2){
        alert("Verdadeiro!");
    }
</script>
```

```
<script>
    var x="2";
    if (x===2){
        alert("Verdadeiro!");
    }
</script>
```

```
<script>
    var x=2;
    if (x!=2){
        alert("Verdadeiro!");
    }
</script>
```

```
<script>
    var x="2";
    if (x!==2){
        alert("Verdadeiro!");
    }
</script>
```

Estrutura da linguagem (18)

Estrutura if/else

```
<script>
    var x=false;
    if (x){
        alert("Verdadeiro!");
    } else{
        alert("Falso!");
    }
</script>
```

Estrutura da linguagem (19)

Estrutura do/while

```
<script>
    var x=1;
    do{
        alert(x);
        x++;
    }while (x<5);
</script>
```

Estrutura da linguagem (20)

Estrutura while

```
<script>
    var x=1;
    while (x<5){
        alert(x);
        x++;
    }
</script>
```

Estrutura da linguagem (21)

Estrutura for

```
<script>
    for (x=1; x<5; x++){
        alert(x);
    }
</script>
```

Estrutura da linguagem (22)

Estrutura for/in

```
<script>
    var x = [1,3,5,7,9];
    for (var i in x){
        alert(x[i]);
    }
</script>
```

Estrutura da linguagem (23)

Estrutura with

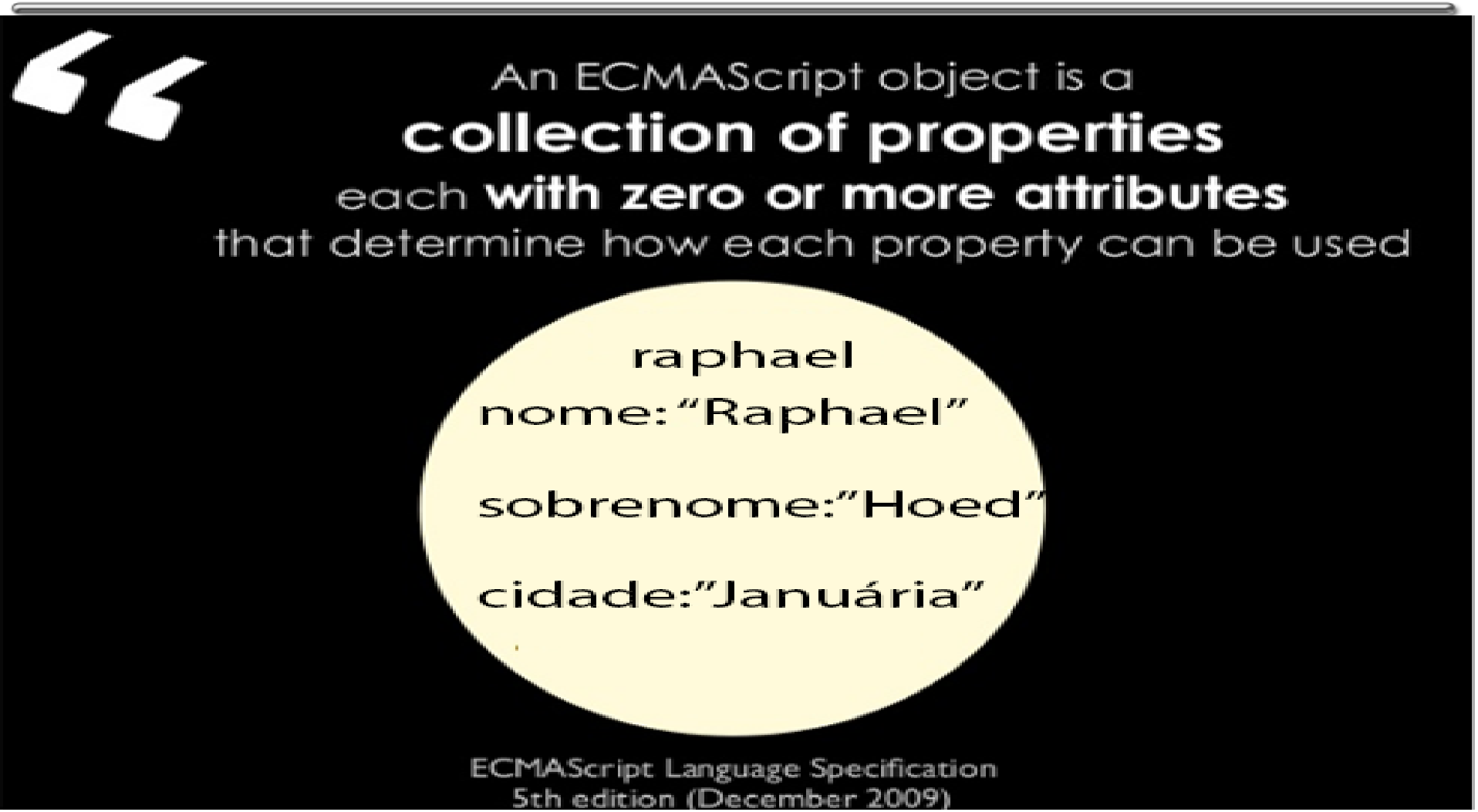
```
<script>
    var x = {
        a:1,b:3,c:5
    };
    with (x){
        alert(a);
        alert(b);
        alert(c);
    }
</script>
```

Estrutura da linguagem (24)

Estrutura switch/case

```
<script>
    var x = "Raphael";
    switch (x) {
        case "Alexandre":
        alert("Xandy");
        break;
        case "José":
        alert("Zé");
        break;
        case "Raphael":
        alert("Rapha");
        break;
    }
</script>
```

Estrutura da linguagem (25)



Estrutura da linguagem (26)

```
<script>
    var raphael = {
        nome:"Raphael",
        sobrenome:"Magalhães Hoed",
        cidade:"Januária"
    };
    alert(raphael.nome + " " + raphael.sobrenome +". Mora em " + raphael.cidade)
</script>
```

Ou:

```
<script>
    var raphael = new String;
    raphael.nome="Raphael";
    raphael.sobrenome="Magalhães Hoed";
    raphael.cidade="Januária";
    alert(raphael.nome + " " + raphael.sobrenome +". Mora em " + raphael.cidade)
</script>
```

Estrutura da linguagem (27)

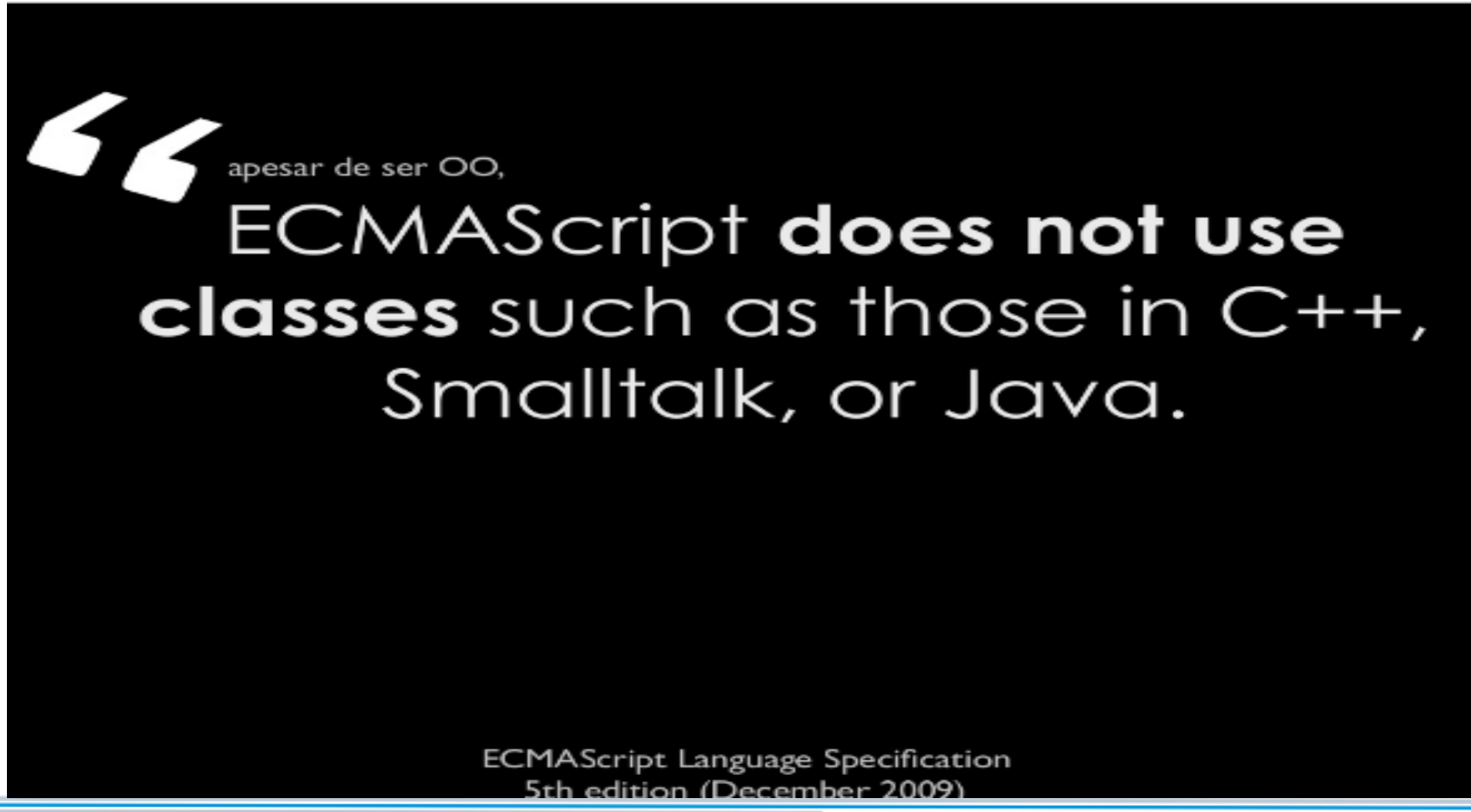
```
<script>
  function insereDados(){
    var raphael = {
        nome:"Raphael",
        sobrenome:"Magalhães Hoed",
        cidade:"Januária",
        escreve:function(){
            document.getElementById("dados").value= "Nome: "+ raphael.nome +
            " "+ raphael.sobrenome + ". Cidade: " + raphael.cidade;
        }
    }
    raphael.escreve();
  }
</script>
<form>
    <input type="text" id="dados" size="100"> <br>
    <input type="button" value="Inserir dados" onclick="insereDados()">
</form>
```

Obs: Não existem métodos, apenas propriedades.

Estrutura da linguagem (28)

Qual a outra forma de criar o objeto raphael no Slide anterior? Façam!

Herança em Javascript (1)



Herança em Javascript (2)

Herança em Javascript (3)

"Classful"

objetos criados a partir de classes

```
hoje = new Date()
```

"Classless"

objetos criados a partir de clonagem...

```
hoje = new Date()
```

...ou por 'geração expontânea'

```
var x = {
  one: 1,
  two: 2
}
```

Herança em Javascript (4)

Herança em Javascript (5)

PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

```
<script>
    var conta = {saldo: 1000};
    alert(conta.saldo);
    alert(conta.limite);
</script>
```

Qual será o resultado?

Herança em Javascript (6)

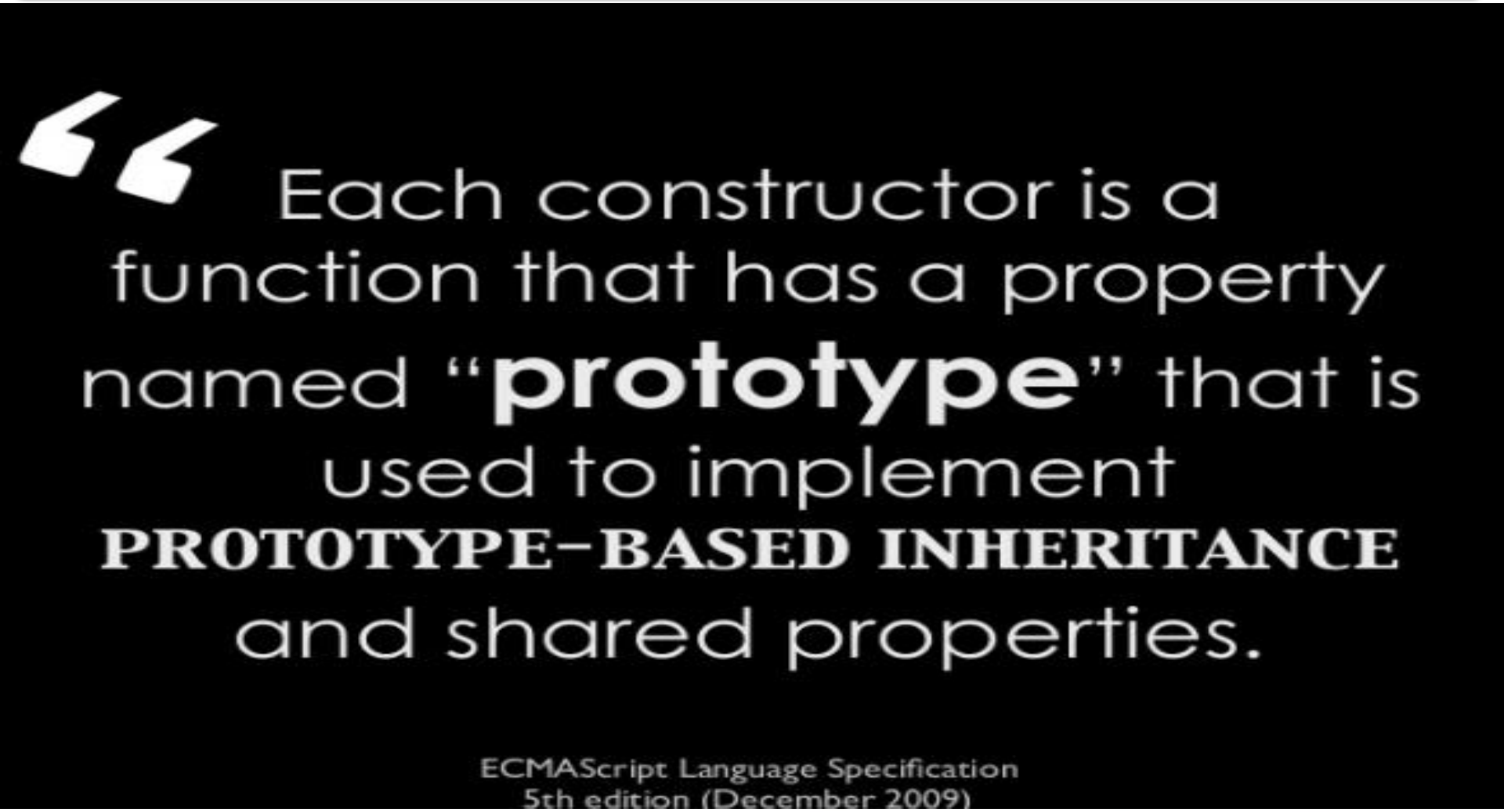
PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

```
<script>
    var conta = {saldo: 1000};

    var conta_especial= {limite: 500};
    alert(conta_especial.limite);
    alert(conta_especial.saldo);
</script>
```

Qual será o resultado?

Herança em Javascript (7)



Herança em Javascript (8)

PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

```
<script>
        function A(){}
        A.prototype.saldo = 1000;

        function B(){}
        B.prototype=new A();
        B.prototype.limite= 500;

        var objeto1= new B();
        alert(objeto1.saldo);
        alert(objeto1.limite);
</script>
```

Qual será o resultado?

Herança em Javascript (9)

PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

```
<script>
function A() {}
A.prototype.nome = "red";
A.prototype.digaNome = function () {
    alert(this.nome);
};

function B() {}
B.prototype = new A();

var objetoA = new A();
var objetoB = new B();
objetoA.nome = 'martins';
objetoB.nome = 'Christiano';
objetoA.digaNome();
objetoB.digaNome();

alert(objetoB instanceof A);
alert(objetoB instanceof B);
</script>
```

Qual será o resultado emitido por cada alert?

Herança em Javascript (10)

PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

```
<script>
    function A(){}
    A.prototype.numero = 10;
    function B(){}
    B.prototype = new A();
    var objeto1= new A();
    B.prototype.numero= objeto1.numero-3;
    function C(){}
    C.prototype = new B();
    var objeto2= new B();
    C.prototype.numero= objeto2.numero*2;
    function D(){}
    D.prototype= new C();
    var objeto3= new C();
    D.prototype.numero= objeto3.numero+22;
    var objeto4= new D();
    var objeto5= new C();
    var objeto6= new B();
    var objeto7= new A();
    alert(objeto4.numero+objeto5.numero+objeto6.numero+objeto7.numero);
</script>
```

Qual será o resultado?

Herança em Javascript (11)

PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

FAÇAM:

```
<script>
    Object.prototype.pO = 1;
    var A = function(){
        this.a=2;
    };
    A.prototype.pA=3;
    var B = function(){
        this.b = 4;
    }
    B.prototype = new A();
    B.prototype.pB = 5;
    x = new B();
    alert (x.b); // Diga o que será exibido
    alert(x.pB); // Diga o que será exibido
    alert(x.a); // Diga o que será exibido
    alert(x.pA); // Diga o que será exibido
    alert(x.pO); // Diga o que será exibido
</script>
```

Programação funcional com Javascript (1)

1. Bata bem todos os ingredientes (menos o fermento) da massa no liquidificador, aproximadamente 2 a 3 minutos
2. Acrescente o fermento e bata por mais uns 15 segundos
3. Coloque em uma fôrma redonda, untada com manteiga e polvilhada com farinha de trigo
4. Asse por cerca de 40 minutos em forno médio (180° graus), preaquecido

- Comunique assim que estiver pronto.
- Não deixe o bolo queimar.
- Não deixe que abram o forno antes da hora.

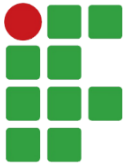
programação imperativa

programação declarativa

Programação funcional com Javascript (2)

```
alunos = [{nome: "Rodrigo",   idade: 30},
          {nome: "Patricia", idade: 20},
          {nome: "Marcos",    idade: 33},
          {nome: "Ricardo",   idade: 28}
         ]
```

Buscar alunos com menos de 30 anos.



Programação funcional com Javascript (3)

```
alunos = [{nome: "Rodrigo",   idade: 30},
          {nome: "Patricia",  idade: 20},
          {nome: "Marcos",    idade: 33},
          {nome: "Ricardo",   idade: 28}
         ]
```

```
menores = []

for (int i = 1; i <= 4; i++) {
   aluno = alunos[i];
   if (aluno.idade < 30) {
    menores.add(aluno);
   }
}

return menores;
```

Programação funcional com Javascript (4)

```
alunos = [{nome: "Rodrigo",   idade: 30},
          {nome: "Patricia",  idade: 20},
          {nome: "Marcos",    idade: 33},
          {nome: "Ricardo",   idade: 28}
         ]

menores = []

for (int i = 1; i <= 4; i++) {
  aluno = alunos[i];
  if (aluno.idade < 30)
   menores.add(aluno);
  }
}

return menores;
```

Programação funcional com Javascript (5)

```
SELECT * FROM alunos WHERE idade < 30
```

	Nome	Idade
1	Rodrigo	30
2	Patricia	20
3	Marcos	33
4	Ricardo	28

Programação funcional com Javascript (6)


```
SELECT * FROM alunos WHERE idade < 30
```

	Nome	Idade
1	Rodrigo	30
2	Patricia	20
3	Marcos	33
4	Ricardo	28

Programação funcional com Javascript (7)

```
<script>
    var numeros = [1,2,3,4,5];
    var dobro = [];
    for (var cont=0;cont<numeros.length;cont++){
        var novosNumeros = numeros[cont]*2;
        dobro.push(novosNumeros);
    }
    alert(dobro);
</script>
```

FUNÇÃO MAP

```
<script>
    var numeros = [1,2,3,4,5];
    var dobro = numeros.map(function(n){
        return n * 2
    }
    );
    alert (dobro);
</script>
```

O que a função map faz?

- Percorre o vetor da esquerda para a direita invocando uma função de retorno em cada elemento com parâmetros. Para cada chamada de retorno, o valor devolvido se torna o elemento do novo vetor. Depois que todos os elementos foram percorridos, map() retorna o novo vetor com todos os elementos “traduzidos”.

Programação funcional com Javascript (10)

```
<script>
    var fahrenheit = [0,32,45,50,75,80,99,120];
    var celsius = [];
    for (var cont=0;cont<fahrenheit.length;cont++){
        var conversao = (fahrenheit[cont]-32)*5/9;
        celsius.push(conversao);
    }
    alert(celsius);
</script>
```

Reescrever o código acima, que usa programação imperativa, usando a programação funcional e a função map.

Programação funcional com Javascript (11)

```
<script>
    var numeros = [1,2,3,4,5];
    var total = 0;

    for (var cont=0; cont<numeros.length; cont++){
        total += numeros[cont];
    }
    alert(total);
</script>
```

FUNÇÃO REDUCE

```
<script>
    var numeros = [1,2,3,4,5];
    var total = numeros.reduce(function(sum,n){
        return sum + n;
    });
    alert(total);
</script>
```

O que a função reduce faz?

A função reduce percorre o array da esquerda para a direita invocando uma função de retorno em cada elemento. O valor retornado é o valor acumulado passado de callback para callback. Depois de todos os elementos terem sido avaliados, reduce() retorna o valor acumulado/concatenado.

FUNCÕES DE PRIMEIRA CLASSE

- Lembrem-se: Em Javascript tudo é tido como objeto, inclusive as funções.
- Funções são objetos de primeira classe em Javascript, pois são tratadas como qualquer tipo de valor (podem possuir propriedades como outros objetos).

FUNCÕES DE ALTA ORDEM

- Aquelas que recebem uma ou mais funções como argumentos.
- Aquelas que devolvem outra função como valor de retorno.

FUNCÕES DE ALTA ORDEM

```
<script>
    function f(x){
        return x+3;
    }

    function twice(func,x){
        return func(func(x));
    }

    alert(twice(f,7));
</script>
```

Qual será o resultado?

CLOSURES

- *Uma closure ocorre normalmente quando uma função é declarada dentro do corpo de outra, e a função interior referencia variáveis locais da função exterior. Em tempo de execução, quando a função exterior é executada, então uma closure é formada, que consiste do código da função interior e referências para quaisquer variáveis no escopo da função exterior que a closure necessita.*

Fonte: https://pt.wikipedia.org/wiki/Clausura_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o)

CLOSURES

```
var digaSeuNome = function( nome ) {
    var msg = "Olá " + nome + ". Seja bem-vindo!";
    var exibeMensagem = function() {
        alert( msg );
    };

    exibeMensagem();
};

digaSeuNome("João");    // Olá João. Seja bem-vindo!
```

CLOSURES

```
var FabricaDeUsuarios = function( ) {
    var id = 0;
    var criaUsuario = function( nome ) {
        id++;
        return {
            id: id,
            nome: nome
        };
    };

    return criaUsuario;
};

var novoUsuario = FabricaDeUsuarios();
var joao = novoUsuario( "João" );
var jose = novoUsuario( "José" );

alert(joao.id);     // 1
alert(jose.id);     // 2
```

Função filter

```
<script>
        var numeros = [1,2,3,4,5];
            var maior4 = numeros.filter(function(n){
             return n >= 4;
        });
        alert(maior4);
</script>
```

O que a função filter faz?

Como map(), filter() percorre o array da esquerda para a direita invocando uma função de retorno em cada elemento. O valor retornado deve ser um booleano que indica se o elemento será mantido ou descartado. Depois de todos os elementos terem sido analisados, filter() retorna um novo array com todos os elementos que retornaram como verdadeiro.

Motivações para trabalhar com programação funcional

Código mais enxuto, mais elegante e sem efeitos colaterais.

```
function generateDate() {
    var date = new Date();
    generate(date);
}

generateDate(); // ???
```

- Não possui inputs e outputs bem definidos.
- Não recebe nada como parâmetro e retorna o que parece ser uma data processada mas não temos como ter certeza.
- Inputs/outputs ocultos podem gerar side effects (funções impuras)

Motivações para trabalhar com programação funcional

A função a seguir tem entradas e saídas bem definidas:

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

square(2); // 4
```

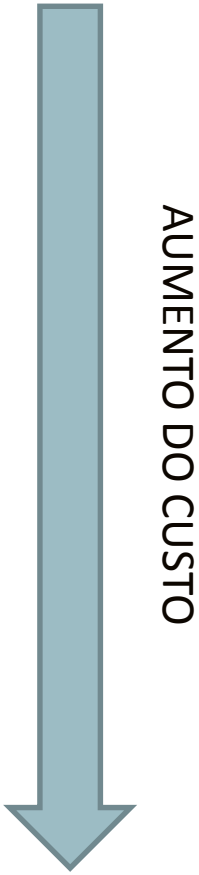
- Escrever funções puras e remover *side-effects* é a base da Programação Funcional.

TDD – Desenvolvimento Orientado a Testes

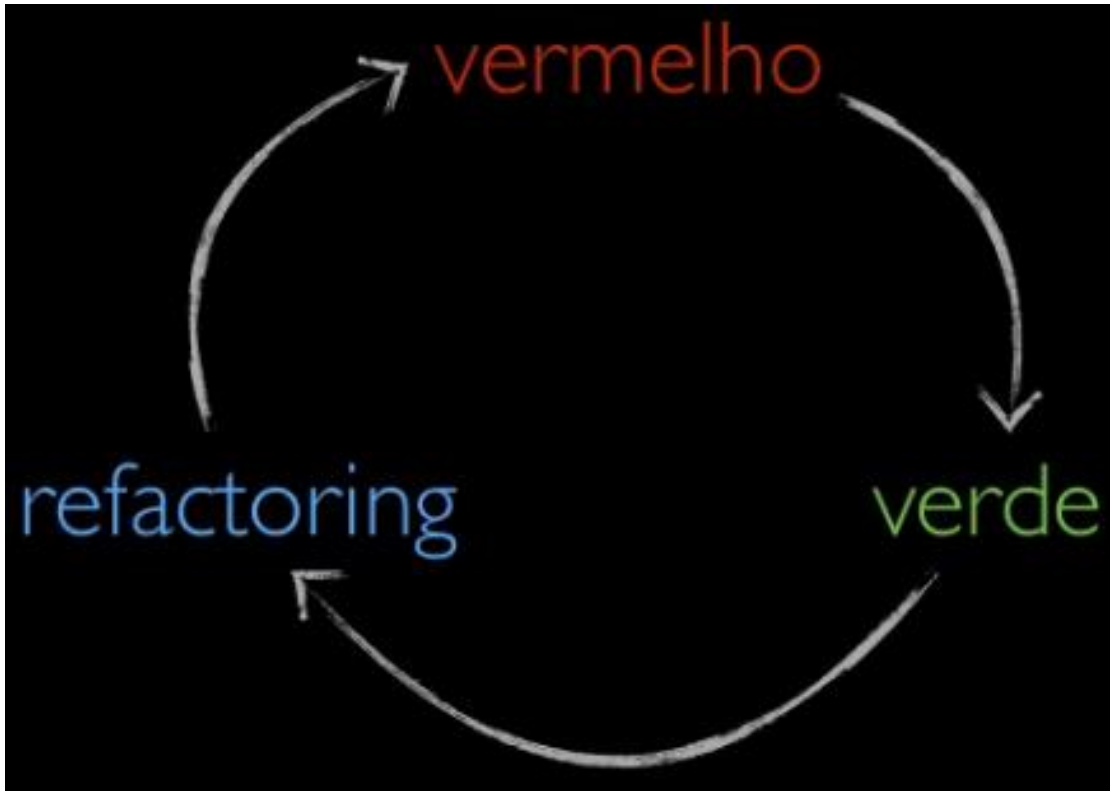
- Testes de software são importantes para garantir a qualidade do software.
- Você adquiriria um carro que foi projetado, montado mas não foi testado?
- Quanto mais tardiamente erros são descobertos, mais caros se tornam

TDD – Desenvolvimento Orientado a Testes

Etapa
Requisitos
Análise e modelagem
Codificação
Pós desenvolvimento
Produção (mão do cliente)



TDD – Desenvolvimento Orientado a Testes



TDD – Desenvolvimento Orientado a Testes

TDD – Desenvolvimento Orientado a Testes

Código a ser testado:

```
function soma(x,y){
        return x + y;
}
```

TDD – Desenvolvimento Orientado a Testes

Código de teste:

```
describe("Matematica", function(){

  it("adiciona um numero inteiro a outro",function(){
      expect(soma(3,3)).toEqual(6);
  });

});
```

TDD – Desenvolvimento Orientado a Testes

```
describe o que eu estou testando

  it tem que se comportar assim
  it tem que se comportar assado
  it tem que fazer isso
  it tem que fazer aquilo

  describe um caso especial
    it tem que tomar cuidado
        expect que alguma operação
        toEqual ao resultado esperado
```

TDD – Desenvolvimento Orientado a Testes

```
expect(true).toBe(true);
expect(false).not.toBe(true);
expect(12).toEqual(12);
expect("foo bar baz").toMatch(/bar/);
expect(soma).toBeDefined();
expect(null).toBeNull();
expect(true).toBeTruthy();
expect(false).toBeFalsy();
expect(["foo", "bar", "baz"]).toContain("bar");
expect(3).toBeLessThan(7);
expect(7).toBeGreaterThan(3);
expect(3.14).toBeCloseTo(3.24, 11);
expect(bar).toThrow();
expect(foo.setBar).toHaveBeenCalled();
expect(foo.setBar).toHaveBeenCalledWith(123);
```

- toEqual() é um matcher (o Jasmine possui vários matchers pré-definidos) que recebe como argumento o valor esperado.

TDD – Desenvolvimento Orientado a Testes

```
describe("Alguma Coisa", function() {} )
```

TDD – Desenvolvimento Orientado a Testes

```
describe("Alguma Coisa", function() {

  it("deveria fazer alguma coisa", function() {})

});
```

TDD – Desenvolvimento Orientado a Testes

```
describe("Alguma Coisa", function() {

  it("deveria fazer alguma coisa", function() {

    // ...

  });

});
```