

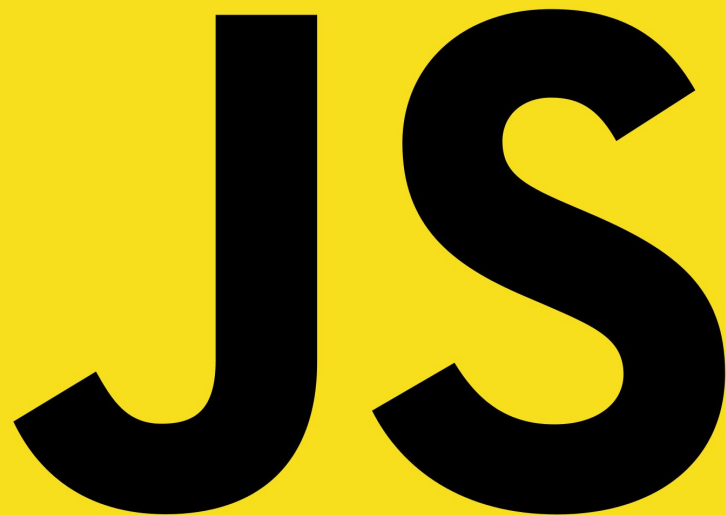
Javascript e Infraestruturas MMM e BRB

...

Tópicos de Javascript e Infraestrutura BRB na plataforma MMM

Javascript - Tópicos

1. Variáveis;
2. Objetos;
3. Funções;
4. Herança;

A large, bold, black 'JS' logo is centered on a bright yellow background. The letters are thick and stylized, with the 'J' having a curved bottom and the 'S' being a simple, rounded shape.

1. Variáveis

- Referência;
- Não é tipado;
- Cuidado com os escopos;



Escopo de Variáveis

Possui 2 escopos:

GLOBAL

- Declarada fora de uma função;
- Acessível e modificável em todo o programa;
- **Setar um valor a uma variável não declarada;**

LOCAL

- Declarada dentro de uma função;
- Ela é criada e destruída sempre que a função é executada;
- Não pode ser acessada fora da função;
- **Pode ter o mesmo nome de uma global, mas é totalmente diferente!**

Exemplo 1

```
var msg = "O aplicativo ficou pronto,";
```

```
function completaMensagem() {  
    var msg = " agora não falta mais nada.";  
}
```

```
completaMensagem();  
msg += " agora falta o chope.";
```

```
console.log(msg);
```

Escopo de Variáveis

Hoisting

- As variáveis são SEMPRE declaradas no início do escopo ao qual pertencem;
- Valor padrão: undefined;
- Inicialização das variáveis não é hasteada.



Exemplo 2

```
function() {  
    var x = 5;  
    elem = document.getElementById("demo");  
    elem.innerHTML = x;  
}
```

Exemplo 3

```
function() {  
    x = 5;  
    elem = document.getElementById("demo");  
    elem.innerHTML = x;  
    var x;  
}
```

Exemplo 4

```
function() {  
    var x = 5;  
    var y = 7;  
    elem = document.getElementById("demo");  
    elem.innerHTML = x + " " + y;  
}
```

Exemplo 5

```
function() {  
    var x = 5;  
    elem = document.getElementById("demo");  
    elem.innerHTML = x + " " + y;  
    var y = 7;  
}
```

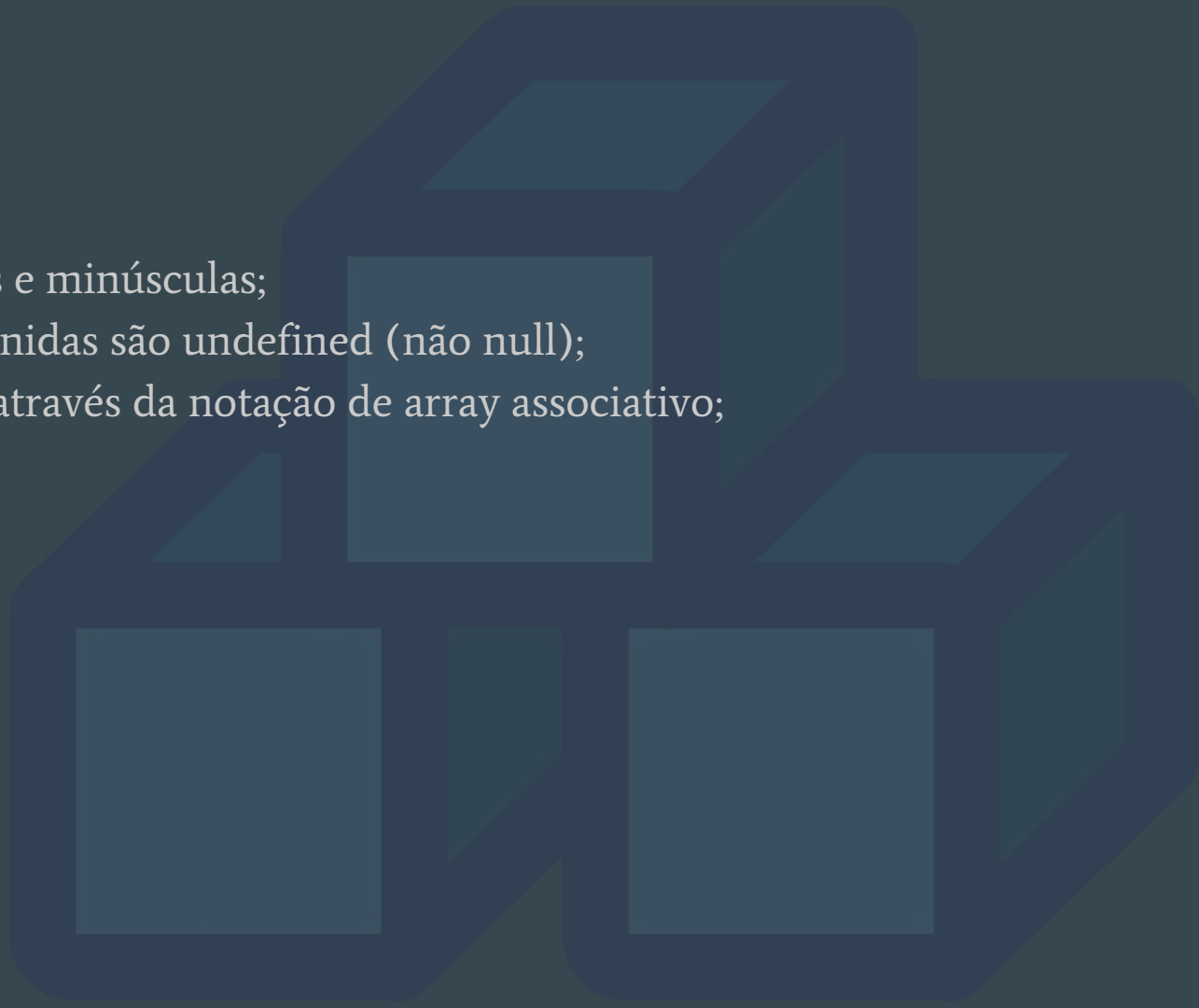

2. Objetos

- JS segue um paradigma orientado a objeto;
- Objeto é uma coleção de propriedades (pares chave/valor);
- Um valor de propriedade pode ser uma função (método);
- Conjunto de objetos pré-definidos;
- Você pode definir seus próprios objetos;



Propriedades

- Diferencia maiúsculas e minúsculas;
- Propriedades não definidas são undefined (não null);
- Podem ser acessados através da notação de array associativo;



Exemplo 6

```
objeto['propriedade'] = 'teste';  
console.log (objeto.propriedade);
```

Exemplo 7

```
var nome = 'propriedade';  
  
objeto[nome] = 'teste2';  
  
console.log (objeto.propriedade);
```

Objetos Padrão

- Praticamente tudo no JS é um objeto;
- Todos os principais tipos primitivos herdam de Objeto;
- Até funções são objetos!
- Objetos disponíveis dependem do ambiente (Ecmascript, Web API...);



Criando Objetos

Formas de criar novos objetos:

1. Notação literal;
2. Construtores;
3. `Object.create()`;



Exemplo 8 - Notação Literal

```
var objeto1 = {  
  propriedade1: 'valor1',  
  propriedade2: 2,  
  propriedade3: {  
    prop1OutroObj: 'valor',  
    prop2OutroObj: false  
  },  
  propriedade4: function () {  
    return true;  
  }  
};
```

```
var objeto2 = {};  
objeto2['propriedade1'] = 'valor1';
```

Exemplo 9 - Construtor

```
function Carro(marca, modelo, ano, dono) {  
    this.marca = marca;  
    this.modelo = modelo;  
    this.ano = ano;  
    this.dono = dono;  
}
```

```
function Pessoa(nome, idade, sexo) {  
    this.nome = nome;  
    this.idade = idade;  
    this.sexo = sexo;  
}
```

```
var dono = new Pessoa("José", 18, "M");  
var meucarro = new Carro("Fiat", "Uno",  
2000, dono);
```

3. Funções

- Toda função no javascript é um objeto Function;
- Todas as funções retornam algo:
 - Caso tenha uma instrução *return*, retorna o que foi definido;
 - Caso não tenha um *return*, retorna o valor padrão, que é *undefined*;
 - A exceção para a regra acima é para os casos de uma função construtora chamada através do operador *new*. Neste caso, o valor retornado é o objeto apontado por *this*.
- É permitido invocar uma função com uma quantidade qualquer de parâmetros;

Criando Funções

Formas de criar funções:

DECLARAÇÃO DE FUNÇÃO

- A função possui um nome;
- Pode ser usado antes da própria declaração;

EXPRESSÃO DE FUNÇÃO

- Função anônima;
- Não pode ser utilizado antes da declaração;

Exemplo 10 - Declarando uma função

```
function minhaFuncao1(objeto) {  
    objeto.make = "Toyota";  
}  
  
var meucarro = {make: "Honda", model:  
"Accord", year: 1998};  
  
minhaFuncao1(meucarro);  
console.log(meucarro.make);
```

Exemplo 11 - Function Expression

```
var minhaFuncao1 = function (objeto) {  
    objeto.make = "Toyota";  
};
```

```
var meucarro = {make: "Honda", model:  
"Accord", year: 1998};
```

```
minhaFuncao1(meucarro);  
console.log(meucarro.make);
```

//IIFE - Immediately Invokable Function
Expression

```
(function(zacarias) {  
    zacarias.alert('teste');  
})(this);
```

Array de Argumentos

- Dentro de uma função, é possível utilizar o array *arguments* para acessar os argumentos repassados ao invocar a função;
- É possível descobrir a quantidade de argumentos através de *arguments.length*;
- É possível acessar qualquer argumento através do índice (*arguments[i]*);
- Útil para os casos em que não se sabe a quantidade de argumentos que podem ser passados para a função;

0 1 2 3 4 5 6 7 8 9



This

Muda de acordo com o contexto:

- **Contexto global:** *this* refere-se ao objeto global. Nos navegadores, refere-se ao objeto *window*;
- **Função com chamada simples** (sem modo estrito): Por padrão, refere-se ao objeto de contexto global;
- **Função com chamada simples** (modo estrito): Por padrão, fica como *undefined*;
- **Função chamada como método de objeto:** Refere-se ao objeto que possui o método;
- **Função chamada como construtor:** Refere-se ao novo objeto que está sendo construído;
- **Função de callback de evento DOM:** Refere-se ao objeto que disparou o evento;

This

Algumas alternativas:

- `Function.prototype.call();`
- `Function.prototype.apply();`
- `Function.prototype.bind();`
- *'that'*;



Funções Aninhadas e *Closures*

- É possível aninhar uma função dentro da outra;
- A função aninhada (interna) é acessível apenas para a função que a contém (exterior).
- *Closures* são funções aninhadas que referenciam a variáveis livres de seu ambiente, além das próprias variáveis internas e globais;
- A função interior contém o escopo da função exterior;
- A função interna pode usar os argumentos e variáveis da função externa, enquanto a função externa não pode usar os argumentos e variáveis da função interna.

Exemplo 12

```
function makeFunc() {  
    var name = "Mozilla";  
    function displayName() {  
        alert(name);  
    }  
    return displayName;  
}
```

```
var myFunc = makeFunc();  
myFunc();
```


Exemplo 13

```
function fora(x) {  
    function dentro(y) {  
        return x + y;  
    }  
    return dentro;  
}
```

`fn_inside = fora(3); // Pense nisso como:`

Receba uma função que adicionará 3 ao que
quer que você repasse para ela

`result = fn_inside(5); // retorna 8`

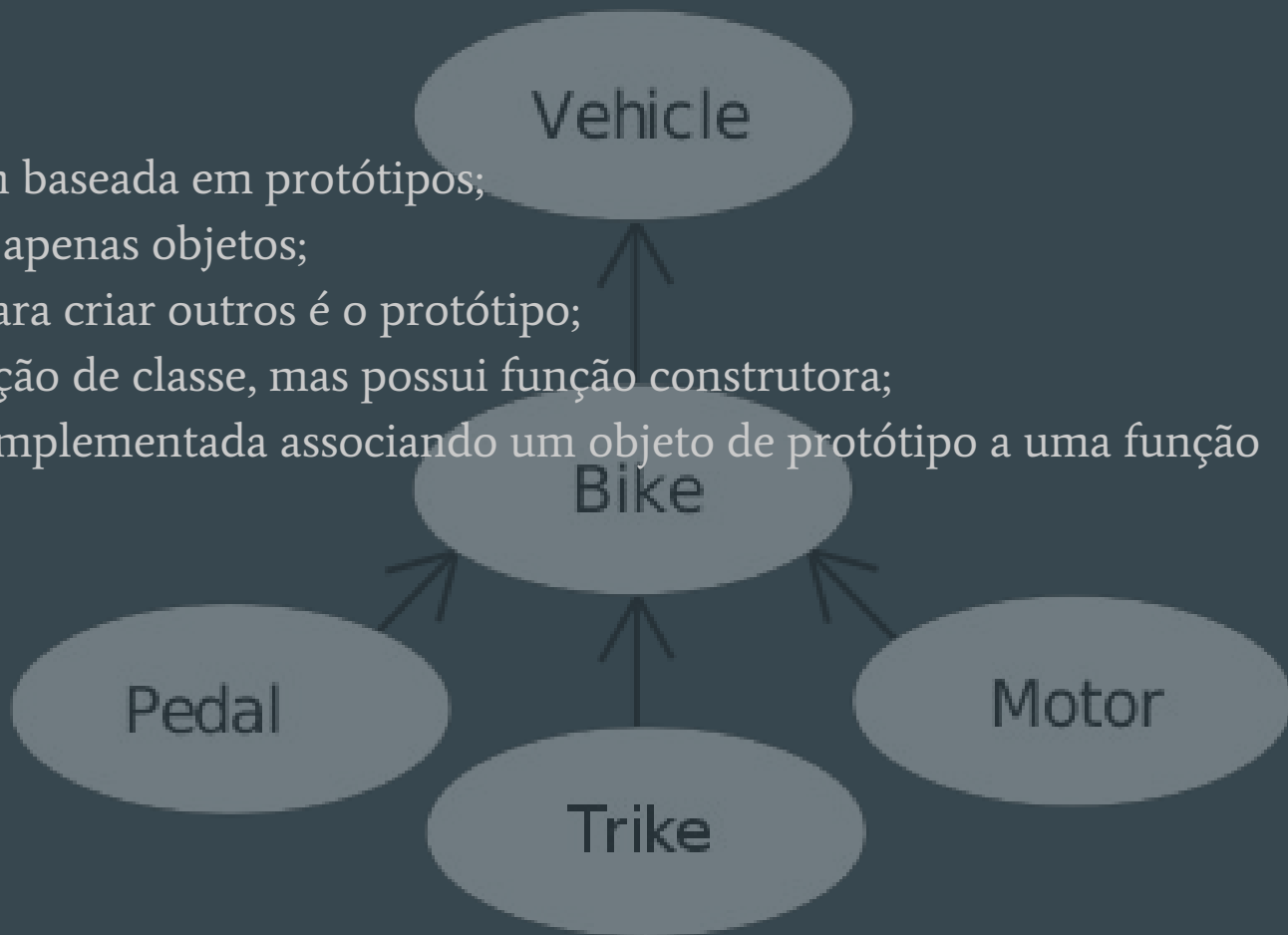
`result1 = fora(3)(5); // retorna 8`

Exemplo 14

```
var Counter = (function() {  
    var privateCounter = 0;  
    function changeBy(val) {  
        privateCounter += val;  
    }  
    return {  
        increment: function() {  
            changeBy(1);  
        },  
        decrement: function() {  
            changeBy(-1);  
        },  
        value: function() {  
            return privateCounter;  
        }  
    }  
})();  
  
alert(Counter.value()); /* Alerts 0 */  
Counter.increment();  
Counter.increment();  
alert(Counter.value()); /* Alerts 2 */  
Counter.decrement();  
alert(Counter.value()); /* Alerts 1 */
```

Herança

- JS é uma linguagem baseada em protótipos;
- Não possui classes, apenas objetos;
- O objeto modelo para criar outros é o protótipo;
- Não possui declaração de classe, mas possui função construtora;
- A herança no JS é implementada associando um objeto de protótipo a uma função construtora;



Herança

Baseados em classes (Java)	Baseados em protótipos (JavaScript)
Classes e instâncias são entidades distintas.	Todos os objetos são instâncias.
Define uma classe com uma definição de classe; instancia uma classe com o método constructor.	Define e cria um conjunto de objetos com funções construtoras.
Cria um único objeto com o operador new.	Faz o mesmo.
Constrói uma hierarquia de objetos usando definição de classe para definir subclasses de classes existentes.	Constrói uma hierarquia de objetos, atribuindo um objeto como o protótipo associado com uma função de construtor.
Herda propriedade seguindo a cadeia de classe.	Herda propriedade seguindo a cadeia de protótipo.
Definição de classe especifica todas as propriedades de todas as instâncias de uma classe. Não é possível adicionar propriedades dinamicamente em tempo de execução.	Função construtoras ou protótipo especifica um conjunto <i>inicial</i> de propriedades. Pode adicionar ou remover propriedades de forma dinâmica para objetos individuais ou para todo o conjunto de objetos.

Exemplo 15

```
function Empregado(nome, departamento) {
    this.nome = nome;
    this.departamento = departamento;
    this.empresa = 'Banrisul';
}
Empregado.prototype.mostrarNome = function () {
    return 'O nome do empregado é: ' + this.nome;
};

function Gerente(nome, departamento) {
    Empregado.call(this, nome, departamento);
    this.gerencia = true;
}
Gerente.prototype = Object.create(Empregado.prototype);

var empregado = new Gerente('Zacarias', 'Almoxarifado');
window.alert(empregado.mostrarNome()); //Zacarias
window.alert(empregado.empresa); //Banrisul
```

Referências

- Escopo de Variáveis:
[https://msdn.microsoft.com/pt-br/library/bzt2dkta\(v=vs.94\).aspx](https://msdn.microsoft.com/pt-br/library/bzt2dkta(v=vs.94).aspx)
- Objetos:
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objeto
- Closures:
<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Closures>
- This:
<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/this>
- <http://pt.slideshare.net/brbruno/javascript-avanado-9524463>
- <https://jsfiddle.net/>