




JavaScript - JS  
piffer.lucas  
@gmail.com

Aula 1 - Projeto  
Lucas Piffer



# JavaScript

- Como chamar um JavaScript de uma página WEB;
- JavaScript Outputs;
- Sintaxe;
- Statements;
- Operadores;
- Tipo dos Dados;
- Funções

# JavaScript

- Objetos;
- Escopos;
- Eventos;
- Strings e Strings methods
- Numbers e Numbers methods
- Math;
- Dates;
- Arrays;
- Booleans;

# JavaScript

- Switch, loop for, loop while;
- Conversão de tipos;
- RegExp;
- Errors;
- Strict mode;
- Ajax;

# Como chamar um JavaScript...

index.html

...

<script>

```
function startMyApp(){  
    console.log('Olá Mundo.');
```

```
}
```

```
window.onload = startMyApp;
```

</script>

</body>

# JavaScript Outputs

```
console.log('Info 1', 'Info 2'); /* A informação aparecerá dentro do console do browser, basta ativar  
usando a tecla F12 */
```

```
document.getElementById("demo").innerHTML = 5 + 6; /* A tag com o ID  
Demo receberá o resultado da soma 5 + 6 */
```

```
<button onclick="document.write(5 + 6)">Try it</button> ; /* Deleta o HTML existente e  
apresenta a informação resultante da operação, nesse caso o valor 11 */
```

# JavaScript Sintaxe

Instruções em um script JavaScript são separadas por ;

```
var x = 5;
```

```
var y = 6;
```

```
var z = x + y;
```

# JavaScript Sintaxe

Existem 2 tipos de valores:

Valores literais: `10.5`, `"Lucas"`, `1000`

Valores variáveis: `var` `x`; `x` = `6`;



# JavaScript Sintaxe

Atribuição:

Atribuição ocorre através do uso do =

```
x = 6;
```

# JavaScript Sintaxe

## Expressões

Expressão é a combinação de valores que avalia para um valor.

`4 * 10` avalia para `40`;

`“lucas” + “piffer”` avalia para `“lucaspiffer”`

# JavaScript Sintaxe

## Keywords

Identificam ações a serem performadas:

o uso da palavra `var` diz para o browser que uma nova variável será criada.

# JavaScript Sintaxe

## Identifiers

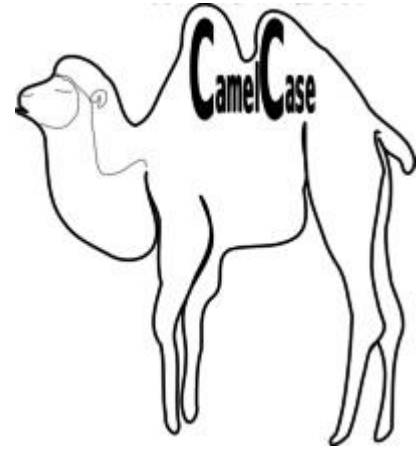
No JavaScript um identifier é o nome atribuído uma variável.

- Número não são permitidos como primeiro carácter.

# JavaScript Sintaxe

caseSensitive identifiers

- `userName = "Madalena";`
- `lastName = "Esptein";`



# JavaScript Statements

Statements são instruções a serem executadas pelo browser;

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Sempre prefira o uso de ; ao término de Statements;

# JavaScript Operadores

Operador	Definição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
++	Incremento
--	Decremento

# Tipos de dados

```
var length = 16;           // Number
var lastName = "Johnson"; // String
var cars = ["Saab", "Volvo", "BMW"]; // Array
var x = {firstName:"John", lastName:"Doe"}; // Object
```



# Tipos de dados

```
function C(){}  
function D(){}  
var o = new C();  
// true, porque: Object.getPrototypeOf(o) === C.prototype  
o instanceof C;
```

[https://developer.mozilla.org/pt-](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/instanceof)

[BR/docs/Web/JavaScript/Reference/Operators/instanceof](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/instanceof)

# Funções

```
function myFunction(p1, p2) {  
    return p1 * p2;           // The function returns the product of p1 and p2  
}
```

-----

```
var x = toCelsius(32);  
  
var text = "The temperature is " + x + " Celsius";
```

# Objeto

```
var car = {type:"Fiat", model:"500",  
color:"white"};
```

```
console.log(car.type);
```

```
console.log(car["type"]);
```

# Escopo

```
var a = "a"; /* global */
```

```
function myFunction(){  
  
}
```

---

```
function myFunction(){  
  
    var a = "a"; /* local */  
  
}
```

# Escopo - automaticamente global

```
function myFunction(){  
    a = "a";  
}
```

```
console.log(a);
```

# Escopo

Local variables are deleted when the function is completed.

Global variables are deleted when you close the page.

# Eventos (events)

Eventos são “coisas” que acontecem. Um evento pode ser algo feito pelo browser ou algo feito pelo usuário.

```
<button onclick='getElementById("demo").innerHTML=Date() '>
```

The time is?

```
</button>
```

# Strings e Strings methods

```
var txt = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
```

```
var sln = txt.length;
```

```
var str = "Please locate where 'locate' occurs!";
```

```
var pos = str.indexOf("locate");
```

```
var str = "Please locate where 'locate' occurs!";
```

```
var pos = str.search("locate");
```

```
var str = "Apple, Banana, Kiwi";
```

```
var res = str.slice(7,13);
```



# Number e Number methods

```
var x = 34.00;    // A number with decimals  
var y = 34;       // A number without decimals
```

```
var x = 123e5;    // 12300000  
var y = 123e-5;   // 0.00123
```

```
var x = 9.656;
```

```
x.toFixed(0);      // returns 10  
x.toFixed(2);      // returns 9.66
```

```
Global Methods: parseInt("10");    // returns 10
```

```
Global Methods: parseInt("years 10"); // returns NaN
```

# Math object

```
Math.random();
```

```
Math.min(0, 150, 30, 20, -8, -200);
```

```
Math.max(0, 150, 30, 20, -8, -200);
```

```
Math.round(4.7);           // returns 5
```

```
Math.round(4.4);           // returns 4
```

```
Math.ceil(4.4);
```

# Dates

Data em javascript podem ser escritos por extenso:

**Tue Jul 12 2016 12:02:18 GMT-0300 (E. South America Standard Time)**

Ou como um número (January 1, 1970, 00:00:00)

**1468335738747**

```
<p id="demo"></p>
```

```
<script>
```

```
    document.getElementById("demo").innerHTML = Date();
```

```
</script>
```

# Creating Date object

```
new Date();
```

```
#Tue Jul 12 2016 12:07:25 GMT-0300 (E. South America Standard Time)
```

```
new Date(milliseconds);
```

```
#new Date(1000000000000) => Sat Sep 08 2001 22:46:40 GMT-0300 (E. South America Standard Time)
```

```
new Date(dateString);
```

```
#new Date("January 20, 2016") => Wed Jan 20 2016 00:00:00 GMT-0200 (E. South America Daylight Time)
```

```
new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

```
new Date(2016, 6, 12, 00, 00, 00, 0000); => Tue Jul 12 2016 00:00:00 GMT-0300 (E. South America Standard Time)
```

```
new Date(2016, 0, 12, 00, 00, 00, 0000); => Tue Jan 12 2016 00:00:00 GMT-0200 (E. South America Daylight Time)
```

# Arrays

```
var cars = [ "Saab", "Volvo", "BMW" ];
```

```
var cars = new Array("Saab", "Volvo", "BMW");
```

```
var name = cars[0];
```

```
var x = cars.length;    // The length property returns the number of elements
```

```
var y = cars.sort();    // The sort() method sorts arrays
```

# Booleans

```
Boolean(10 > 9)      // returns true
```

```
var x = 0;
```

```
Boolean(x);          // returns false
```

```
var x = null;
```

```
Boolean(x);          // returns false
```

# Switch, loop for, loop while

```
switch(expression) {
```

```
    case n:
```

```
        code block
```

```
        break;
```

```
    case n:
```

```
        code block
```

```
        break;
```

```
    default:
```

```
        default code block
```

```
}
```

```
for (i = 0; i < cars.length; i++)  
{  
  
    text += cars[i] + "<br>";  
  
}
```

```
while (condition) {
```

```
    code block to be executed
```

```
}
```

# Conversão de tipos

```
typeof "John"           // Returns string
typeof 3.14              // Returns number
typeof NaN               // Returns number
typeof false             // Returns boolean
typeof [1,2,3,4]         // Returns object
typeof {name:'John', age:34} // Returns object
typeof new Date()        // Returns object
typeof function () {}    // Returns function
typeof myCar              // Returns undefined (if myCar is not declared)
typeof null
```



# Conversão de tipos

```
String(false)           // returns "false"
```

```
String(true)            // returns "true"
```

```
Date().toString()
```

```
Number("3.14")          // returns 3.14
```

```
Number(" ")              // returns 0
```

```
Number("")               // returns 0
```

# RegExp

# Errors

The **try** statement lets you test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

The **finally** statement lets you execute code, after try and catch, regardless of the result.

# Errors

```
<script>
try {
  adddler("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
</body>

</html>
```

# Strict Mode

## **The "use strict" Directive**

The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5). It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.

The purpose of "use strict" is to indicate that the code should be executed in "strict mode".

With strict mode, you can not, for example, use undeclared variables.

# Strict Mode

```
"use strict";
```

```
x = 3.14;           // This will cause an error (x is not defined)
```

```
"use strict";
```

```
function x(p1, p2) {};
```

```
delete x;           // This will cause an error
```

```
"use strict";
```

```
eval ("var x = 2");
```

```
alert (x);           // This will cause an error
```

# Ajax

AJAX is a developer's dream, because you can:

Update a web page without reloading the page

Request data from a server - after the page has loaded

Receive data from a server - after the page has loaded

Send data to a server - in the background

# Ajax

```
function loadDoc() {  
  
    var xhttp = new XMLHttpRequest();  
  
    xhttp.onreadystatechange = function() {  
  
        if (xhttp.readyState == 4 && xhttp.status == 200) {  
  
            document.getElementById("demo").innerHTML = xhttp.responseText;  
  
        }  
  
    };  
  
    xhttp.open("GET", "ajax_info.txt", true);  
  
    xhttp.send();  
}
```

```
<button type="button" onclick="loadDoc()"  
>Change Content</button>
```