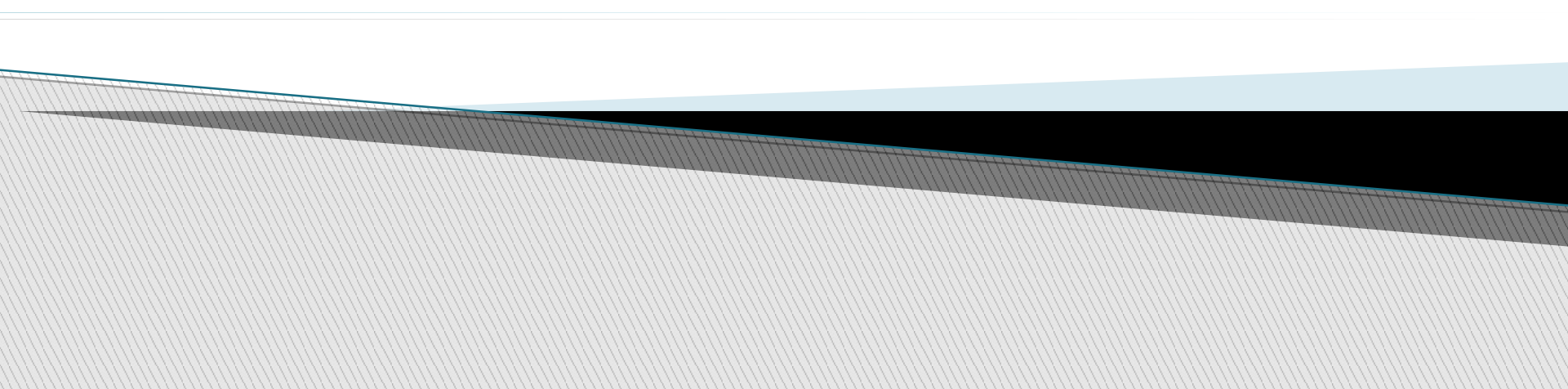


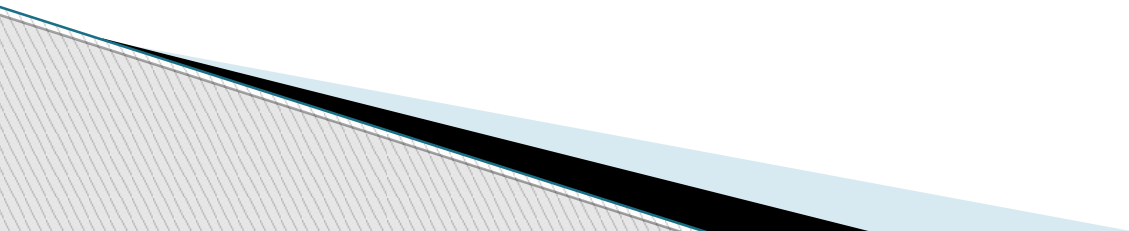
Linguagem de Programação para Web

JavaScript

Prof. Tales Bitelo Viegas



**A linguagem de
programação mais mal-
compreendida do mundo**



Razões

- ▶ O nome
- ▶ Implementações ruins
- ▶ O browser
- ▶ JavaScript é uma linguagem Funcional

Histórico

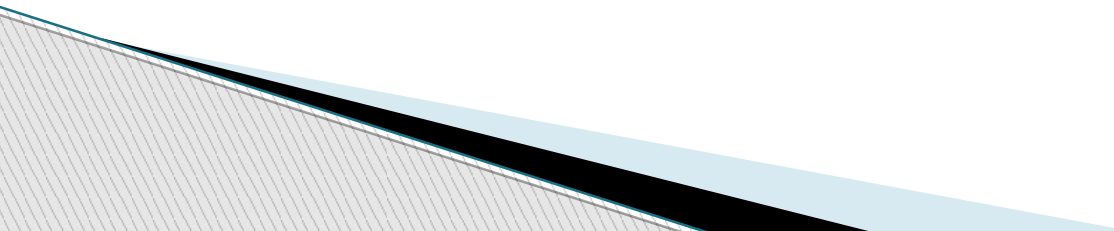
- ▶ 1992
 - Oak, Gosling at Sun & FirstPerson
- ▶ 1995
 - HotJava
 - LiveScript, Eich at Netscape
- ▶ 1996
 - Jscript at Microsoft
- ▶ 1998
 - ECMAScript
 - 2014 – ECMAScript 6 - POO



Não é uma linguagem de brinquedo

- ▶ É uma linguagem real
- ▶ Pequena, mas sofisticada
- ▶ Não é um sub-conjunto de Java

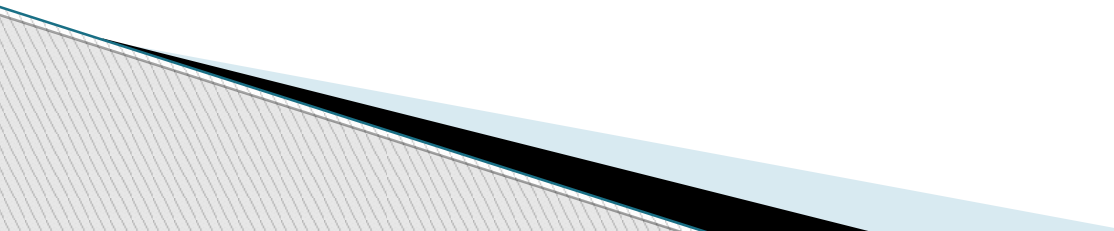
Ideias chave

- ▶ Entrega “Load and go”
 - ▶ Tipagem fraca
 - ▶ Objetos como containers gerais
 - ▶ Herança
 - ▶ Ligações através de variáveis globais
- 

Iniciando

- ▶ Interatividade via caixas de texto:
 - `confirm("Seu nome é Tales?");`
 - `prompt("Qual é o seu nome?");`
 - `alert("Oi Tales");`

Valores

- ▶ Numbers
 - ▶ Strings
 - ▶ Booleans
 - ▶ Objects
 - **null**
 - **undefined**
- 

Numbers

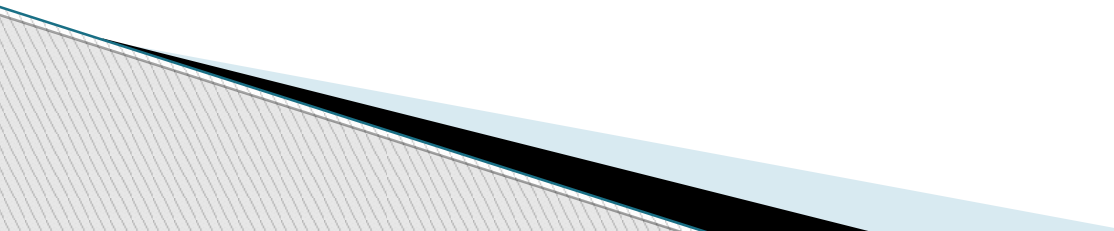
- ▶ Somente um tipo numérico
 - Sem inteiros
- ▶ 64-bit ponto flutuante
- ▶ IEEE-754 (aka “**Double**”)
- **$0.1 + 0.2 = 0.3000000000000000000004$**

NaN

- ▶ Número especial: Not a Number
- ▶ Resultado de operações erradas ou com undefined
- ▶ Qualquer operação aritmética com **NaN** nos operadores irá gerar um **NaN** como resultado
- **NaN** é diferente de qualquer coisa, incluindo **NaN**

Função Number

Number(value)

- ▶ Converte o valor em um Number.
 - ▶ Resulta em **NaN** se tiver um problema.
 - ▶ Similar ao operador +.
- 

Função parseInt

parseInt(value, 10)

- ▶ Converte o valor em um Number.
- ▶ Para no primeiro caracter não-dígito.
 - **parseInt("08")**
 - **parseInt("08", 10) === 8**

Math

- ▶ O objeto Math é modelado com base na classe Math do Java.

- ▶ http://www.w3schools.com/jsref/jsref_obj_math.asp

- ▶ Contém

Math.abs(x)	valor absoluto
Math.floor(x)	parte inteira
Math.log(x)	logaritmo
Math.max(x, ..., z)	máximo
Math.pow(x, y)	eleva a uma potência
Math.random()	número randômico
Math.round(x)	arredondamento
Math.sin(x)	seno
Math.sqrt(x)	raíz quadrada

Strings

- ▶ Sequencia de 0 ou mais caracteres de 16-bit
 - Codificação UCS-2, não apenas UTF-16
- ▶ Não possui um tipo caracter
 - Caracteres são representados com uma String de tamanho 1
- ▶ Strings são imutáveis
- ▶ Strings similares são iguais (==)
- ▶ String literais podem usar aspas simples ou duplas

String length

- ▶ `string.length`
- ▶ A propriedade **length** determina o número de caracteres de uma String.

Função String

String(value)

- ▶ Converte um valor para uma String
- ▶ http://www.w3schools.com/jsref/jsref_obj_string.asp

Métodos de um objeto

String

- `str.charAt(x)` Caracter na posição `x`
- `str.concat(str2)` Concatena `str` com `str2`
- `str.indexOf(str2)` Retorna a primeira posição onde `str2` está na string `str` (ou `-1`)
- `str.lastIndexOf(str2)` Retorna a última posição onde `str2` está na string `str` (ou `-1`)
- `str.match(re)` Verifica se `str` bate com a expressão regular `re`
- `str.replace(str2, str3)` Substitui todas as ocorrências de `str2` por `str3`

Métodos de um objeto

String

- `str.search(str2)` Procura a expressão/expressão regular `str2` em `str` e retorna o índice (ou -1)
- `str.slice(posIni, posFim)` Retorna uma string começando em `posIni` e terminando em `posFim`
- `str.split(separador)` Retorna um array com a string dividida em pedaços, separadas pelo separador
- `str.toLowerCase()` `str` em letras minúsculas
- `str.toUpperCase()` `str` em letras maiúsculas

Boolean

- **true**
- **false**

Função Boolean

Boolean(value)

- ▶ retorna **true** se o valor é verdadeiro
- ▶ retorna **false** se o valor é falso

null

- ▶ Um valor que não é nada

undefined

- ▶ Valor default para variáveis e parâmetros

Valores Falsos

- `false`
 - `null`
 - `undefined`
 - `""` (empty string)
 - `0`
 - `NaN`
- ▶ **Todos os outros valores (incluindo todos os objetos) são verdadeiros.**

`"0"`

`"false"`

Saída de Texto na Console

- ▶ `console.log(mensagem);`

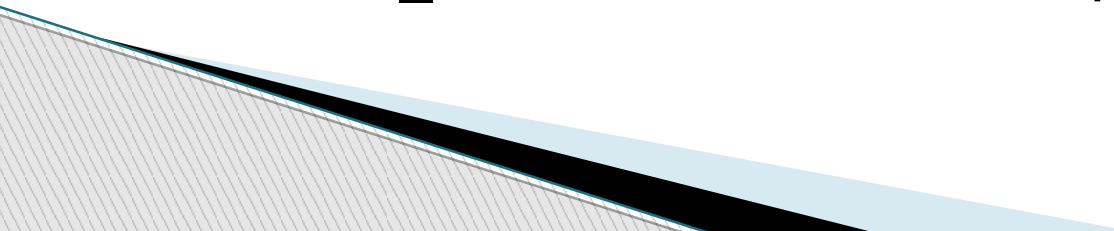
C

- ▶ JavaScript é sintaticamente uma linguagem da “família C”
- ▶ Difere do C principalmente no sistema de tipagem, o qual permite que funções sejam armazenadas como valores

Tipagem fraca

- ▶ Qualquer tipo pode ser armazenado em uma variável ou passado como parâmetro para qualquer função

Identificadores

- ▶ Começam com uma letra, ou `_`, ou `$`
 - ▶ Seguido por zero ou mais letras, números, `_` ou `$`
 - ▶ Por convenção, todas as variáveis, parâmetros, membros e funções começam com letras minúsculas
 - ▶ Exceto por construtores, que começam com letra maiúscula
 - ▶ Inicial `_` deve ser reservado para implementações
- 

Comentários

// Comentários de linha

/*

**comentários
de
bloco**

***/**



Operadores

- ▶ Aritméticos
 - + - * / %
- ▶ Comparação
 - == != < > <= >=
- ▶ Lógico
 - && || !
- ▶ Bit-a-Bit
 - & | ^ >> >>> <<
- ▶ Ternário
 - ? :

+

- ▶ Adição e concatenação
- ▶ Se os dois operandos forem números,
 - then
 - soma eles
 - else
 - converte todos em strings
 - concatena eles

'\$' + 3 + 4 = '\$34'

+

- ▶ Operador unário pode converter strings em números

`+"42" = 42`

- ▶ ou

`Number("42") = 42`

- ▶ ou

`parseInt("42", 10) = 42`

`+"3" + (+"4") = 7`



- ▶ Divisão de dois inteiros pode produzir um resultado não-inteiro

$$10 / 3 = 3.33333333333333335$$

`==` `!=`

- ▶ Igual e diferente
- ▶ Estes operadores irão coagir para tipos diferentes (`"1" == 1`)
- ▶ É melhor usar `===` e `!==`, os quais não fazem coeções de tipos.

&&

- ▶ Operador lógico AND
- ▶ Pode ser usado para evitar referências a null

```
if (a) {  
    return a.member;  
} else {  
    return a;  
}
```

- ▶ pode ser escrito como:
 return a && a.member;



- ▶ Operador lógico OR
- ▶ Pode ser usado para definir valores default.
`var last = input || nr_items;`
- ▶ (Se **input** é verdadeiro, então **last** é **input**, senão define **last** como **nr_items**.)

!

- ▶ Operador de negação

Comandos

- ▶ *expression*
- **if**
- **switch**
- **while**
- **do**
- **for**
- **break**
- **continue**
- **return**
- **try/throw**

Comando break

- ▶ Comandos podem ter labels.
- ▶ Comando Break podem referir a estes labels.

```
loop: for (;;) {
```

```
...
```

```
    if (...) {  
        break loop;  
    }
```

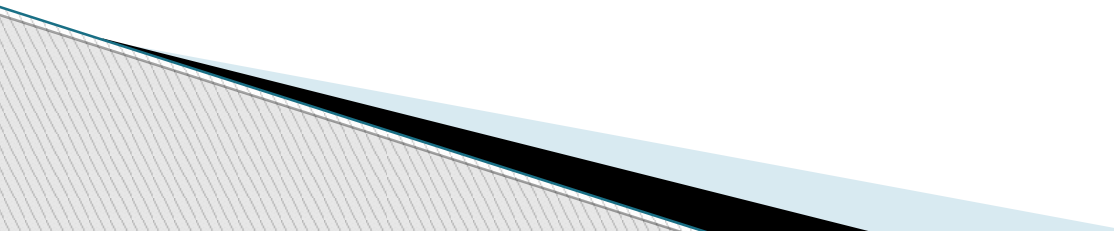
```
...
```

```
}
```



Comando For

```
for (var i = 0; i < array.length; i += 1) {  
  
    // comandos  
}
```



Comando For

- ▶ Pode iterar através de todos os membros de um objeto:

```
for (var name in object) {  
    if (object.hasOwnProperty(name)) {
```

```
        // name é a chave do membro corrente
```

```
        // object[name] é o valor da chave
```

```
    }
```

```
}
```

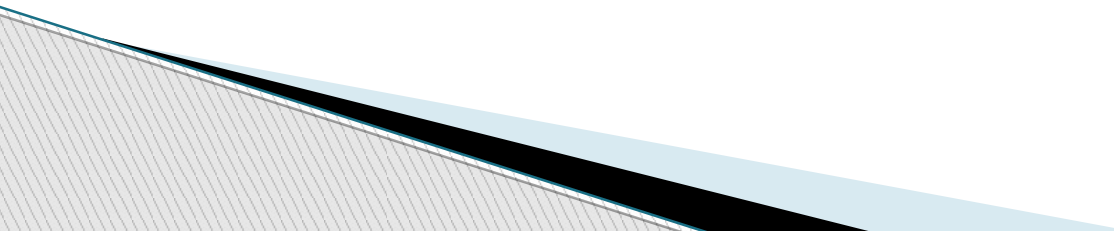


Comando Switch

- ▶ O valor do switch não necessita ser um number. Pode ser uma String
- ▶ Os valores dos “case” podem ser expressões.

Switch statement

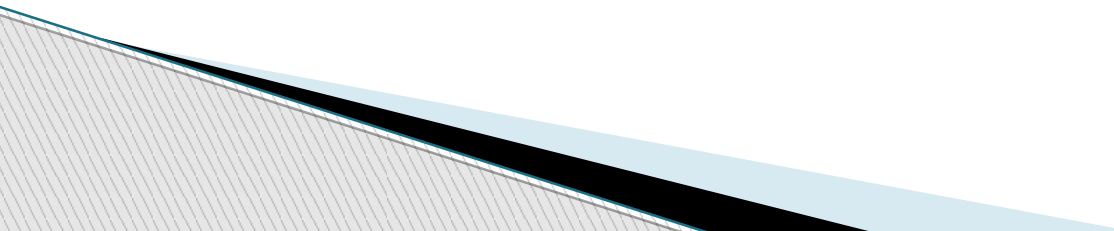
```
switch (expression) {  
  case ';' :  
  case ',' :  
  case '.' :  
    punctuation();  
    break;  
  default :  
    noneOfTheAbove();  
}
```



Comando Throw

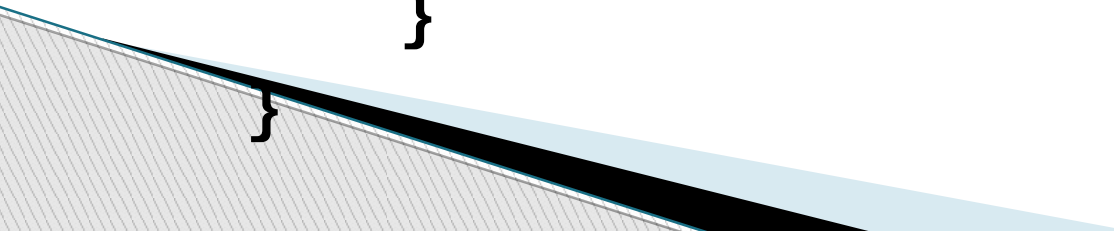
```
throw new Error(reason);
```

```
throw {  
    name: exceptionName,  
    message: reason  
};
```



Comando Try

```
try {  
    ...  
} catch (e) {  
    switch (e.name) {  
    case 'Error':  
        ...  
        break;  
    default:  
        throw e;  
    }  
}
```



Comando Try

- ▶ A implementação do JavaScript pode produzir estes nomes de exceções:

'Error'

'EvalError'

'RangeError'

'SyntaxError'

'TypeError'

'URIError'

Comando Function

```
function name(parameters) {  
    comandos;  
}
```

Parâmetros Default (ES6)

```
function teste(url, timeout=2000, callback =  
function(){} ) {  
    // O resto da função  
}
```

Apenas o parâmetro URL é necessário

Parâmetros REST (ES6)

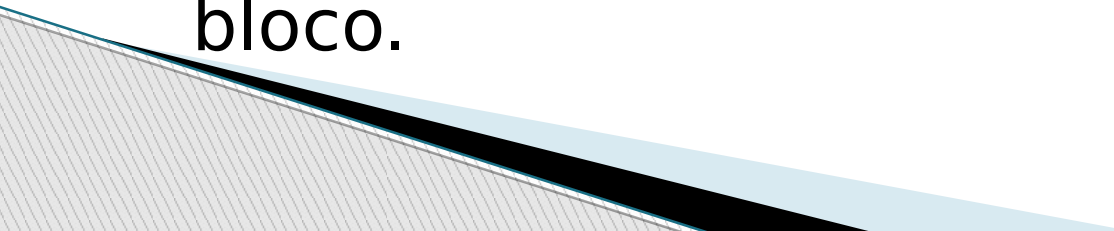
```
function sum(...numbers) {  
    var len = numbers.length;  
    var i=0;  
    var result = 0;  
    while(i < len){  
        result += numbers[i++];  
    }  
    return result;  
}
```


Comando Var

- ▶ Define variáveis de uma função.
- ▶ Tipos não são especificados.
- ▶ Valores iniciais são opcionais.

```
var name;  
var nrErrors = 0;  
var a, b, c;
```

Escopo

- ▶ Em JavaScript, {blocos} não tem escopo.
 - ▶ Apenas funções tem escopo.
 - ▶ Vars definidas em uma função não são visíveis fora desta função.
 - ▶ (ES6) Se declarar a variável com o comando `let` ao invés de `var`, o escopo é local ao bloco.
- 

(ES6) Constantes

- ▶ Declarar uma variável como const faz ela ser uma constante
- ▶ `const CONSTANTE=10;`

Comando Return

`return expression;`

▶ ou

`return;`

- ▶ Se não há *expression*, então o valor retornado é **undefined**.
- ▶ Exceto para construtores, no qual o valor default do retorno é **this**.

Exercício 1

- ▶ Implemente um programa em JavaScript que leia o tempo em segundos e escreve este tempo decomposto em horas, minutos e segundos.
- ▶ Por exemplo, se o tempo for 3788, o programa deve escrever 1 hora, 3 minutos e 8 segundos.

Exercício 2

- ▶ Faça um programa para informar a um usuário se o número que ele digitou é par ou ímpar

Exercício 3

- ▶ Faça um programa que peça para um usuário a altura e a largura de um quadrilátero (números inteiros)
- ▶ Após, imprima o quadrilátero na tela
- ▶ Ex:
 - Altura: 2
 - Largura: 5

XXXXXX

XXXXXX



Exercício 4

- ▶ Faça uma função que receba como parâmetros o nome e o sobrenome de uma pessoa e retorne o nome completo dela