

JS

Sumário

1 - Introdução

1.1 - Origem

1.2 - Visão geral

1.3 - Por onde começar

1.4 - Hello World

2 - Amarrações

2.1 - Definições e declarações

2.2 - Tempos de amarração

2.3 - Closures

2.4 - Identificadores

2.5 - Ambiente e escopo de amarração

Sumário

3 - Valores e tipos de dados:

3.1 - Sistema de Tipos

3.2 - Tipos Compostos: Objeto

3.3 - Tipos Compostos: Array

4 - Variáveis e constantes:

4.1 - Armazenamento em memória principal

4.2 - I/O;

4.3 - Serialização

Sumário

5 - Expressões e comandos:

5.1 - Atribuição

5.2 - Comparação

5.3 - Aritmético

5.4 - Bit a bit

5.5 - Lógico

5.6 - String

5.7 - Condicional (Ternário)

5.8 - Unário

5.9 - Relacionais

5.10 -Curto Circuito

5.11 - Tipos de comandos e expressões

5.12 - Detalhes da linguagem

5.13 - Expressões

Sumário

6-Modularização:

- 6.1- Subprogramas e parâmetros
- 6.2 - Parâmetros Rest
- 6.2 - Pacotes e espaços de nome
- 6.3 - Arquivos separados

7 - Polimorfismo:

- 7.1 - Características
- 7.2 - Coerção
- 7.3 - Sobrecarga
- 7.4 - Paramétrico
- 7.5 - Inclusão
- 7.6 - Herança

Sumário

8 - Exceções

9 - Concorrência

9.1 - Processos e threads

9.2 - Semáforos

9.3 - Suporte avançado de JavaScript à programação concorrente

10 - Frameworks

10.1 - Frameworks

10.2 - Interação com HTML - JQuery

11 - Avaliação

12- Referências

1. Introdução

1.1 Origem

- Criada em 1995 por Brendan Eich enquanto trabalhava na Netscape Communications Corporation
- Microsoft portou a linguagem para seu navegador

1.1 Origem

- Recebeu nomes como LiveScript e Mocha
- Java e JavaScript são completamente diferentes
- Javascript é a linguagem de programação mais usada no mundo, lista divulgada pela firma de análise de mercado RedMonk durante o mês de junho.

1.2 Visão Geral

JavaScript[®] (às vezes abreviado para JS) é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe, mais conhecida como a linguagem de script para páginas Web, mas usada também em vários outros ambientes sem browser como node.js, Apache CouchDB e Adobe Acrobat.

É uma linguagem de script multi-paradigma, baseada em protótipo que é dinâmica, e suporta estilos de programação orientado a objetos, imperativo e funcional.

1.2 Visão Geral

- É uma linguagem de scripting multiplataforma orientada a objetos
- Facilmente incorporada a outros produtos e aplicações, como navegadores web
- É uma linguagem muito popular
- Muito utilizada no front-end junto com o HTML e CSS
- Multi paradigma
- Case-sensitive

1.2 Visão Geral

- No lado do cliente: Estende a linguagem básica através do fornecimento de objetos para controle do navegador e seu Modelo de Objeto de Documento (DOM).
- Por exemplo, extensões no lado do cliente permitem a uma aplicação colocar elementos em um formulário HTML e responder a eventos do usuário como cliques do mouse, entrada de dados em formulário e navegação na página.

1.2 Visão Geral

- Lado do servidor: Estende a linguagem básica fornecendo objetos relevantes à execução de JavaScript no servidor.
- Por exemplo, extensões no servidor permitem a uma aplicação se comunicar com um banco de dados relacional, disponibilizar continuidade de informação entre uma chamada e outra da aplicação ou realizar manipulações de arquivo em um servidor.

1.3 Por onde começar

Você pode escrever o código no arquivo .html ou criar um arquivo separado para isso, caso opte pelo arquivo externo, use .js como extensão.

- No mesmo arquivo

Insira seu código preferencialmente dentro da tag `</body>`

```
<script> //código aqui </script>
```

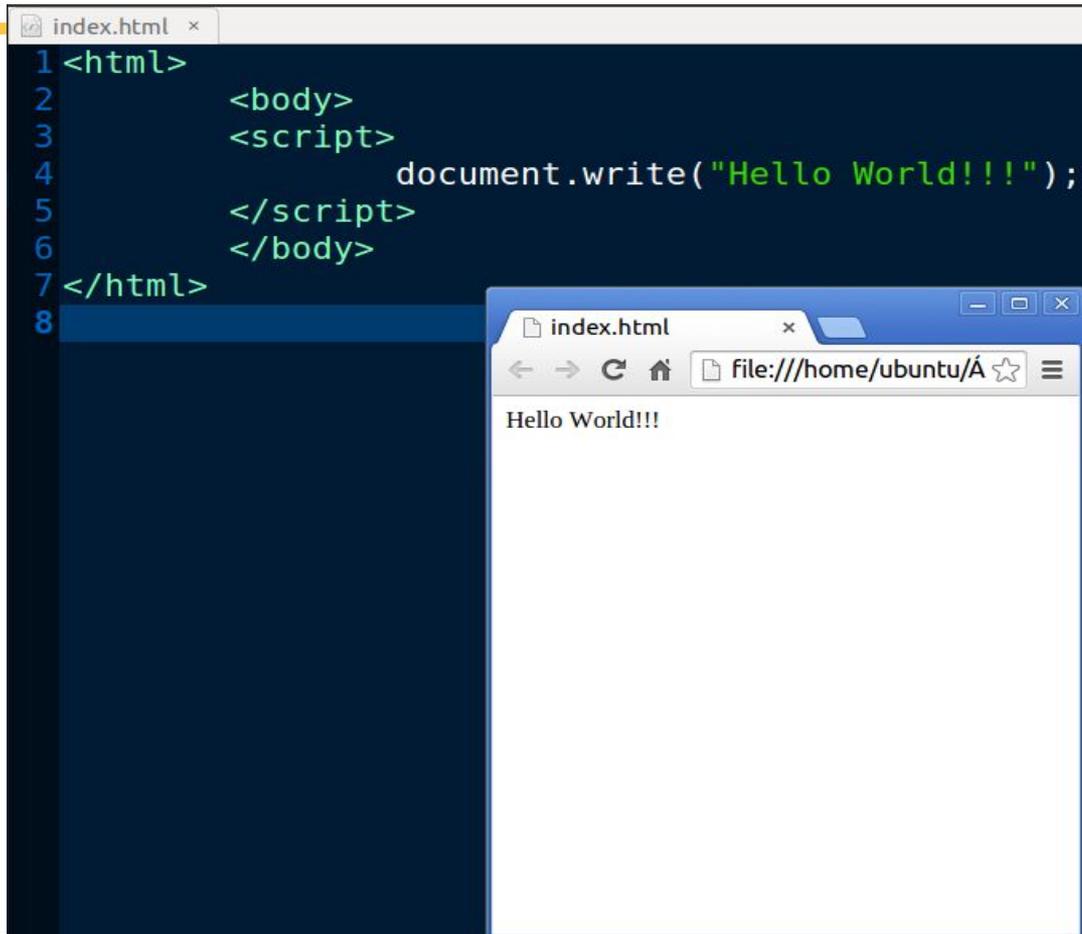
- Arquivo externo

Insira o link do seu arquivo javascript no atributo `src` da tag `<script>`

```
<script src="arquivo.js"></script>
```

1.4 Hello World

- Você pode usar:
`document.write();`
para escrever no documento.



The image shows a code editor window titled 'index.html' with the following code:

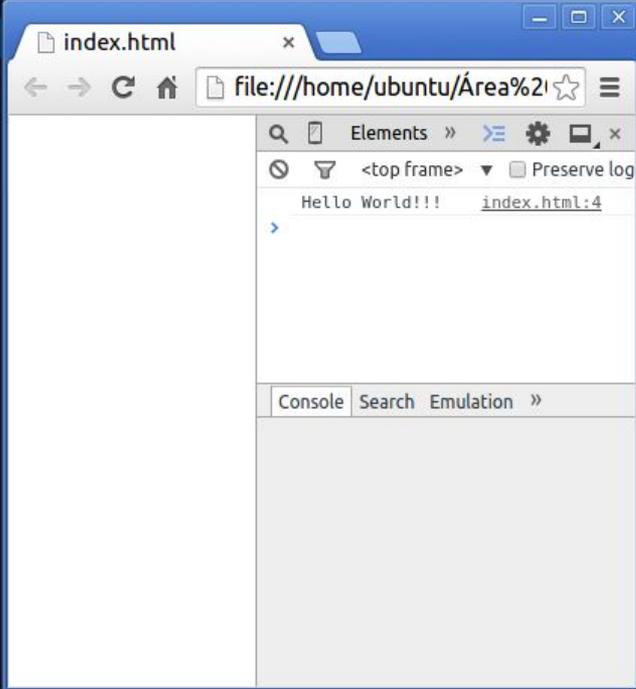
```
1 <html>
2     <body>
3     <script>
4         document.write("Hello World!!!");
5     </script>
6 </body>
7 </html>
8
```

Below the code editor is a browser window titled 'index.html' with the address bar showing 'file:///home/ubuntu/Á'. The browser displays the output 'Hello World!!!'.

1.4 Hello World

- Pode utilizar `console.log()`; para escrever no console

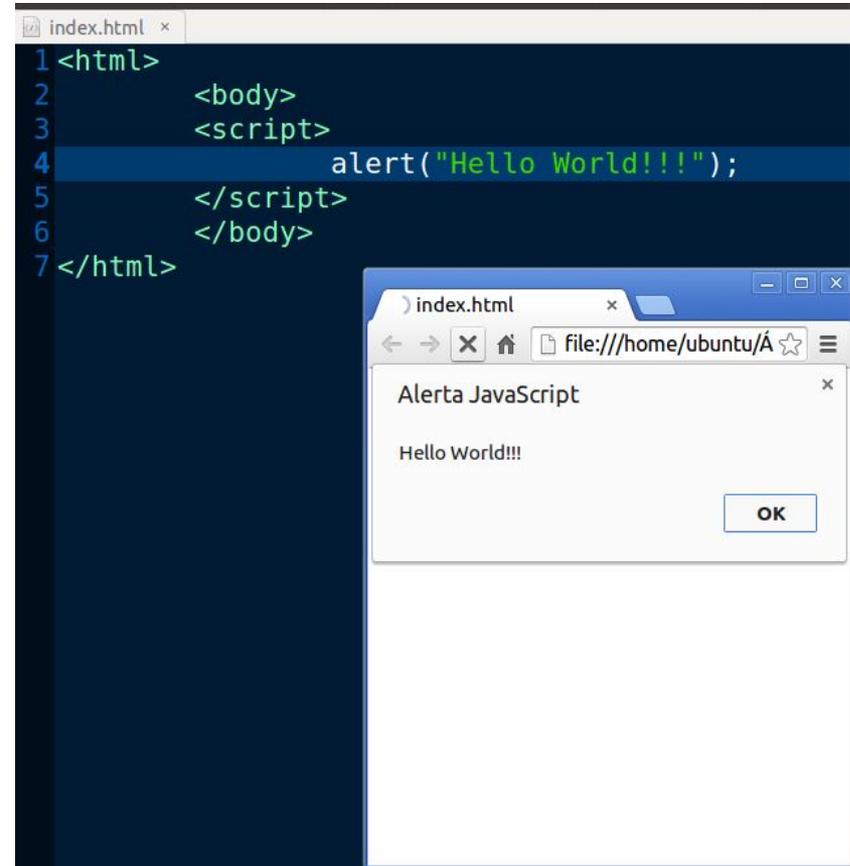
```
index.html x
1 <html>
2     <body>
3     <script>
4         console.log("Hello World!!!");
5     </script>
6     </body>
7 </html>
8
```



The screenshot shows a web browser window with the address bar displaying `file:///home/ubuntu/Área%20de%20trabalho/`. The browser's developer tools are open, showing the 'Console' tab with a single log entry: `Hello World!!!` from `index.html:4`. The 'Elements' panel is also visible, showing the document structure.

1.4 Hello World

- Pode utilizar `alert();` para criar um alerta



The image shows a code editor window with a dark theme. The code is as follows:

```
1 <html>
2     <body>
3     <script>
4         alert("Hello World!!!");
5     </script>
6     </body>
7 </html>
```

Below the code editor, a browser window is open. The address bar shows the file path: `file:///home/ubuntu/Á`. An alert dialog box is displayed in the foreground with the title "Alerta JavaScript" and the message "Hello World!!!". There is an "OK" button at the bottom right of the dialog.

2. Amarrações

2.1 Definições e Declarações

- Instruções são chamadas de declaração e são separadas por um ponto e vírgula (;)
- Espaços, tabulação e uma nova linha são chamados de espaços em branco.
- Se as instruções forem separadas por uma linha, você pode omitir o (;)
- label,break,continue e return funcionam da mesma forma que em java

2.1 Definições e Declarações

- Variáveis:

`var` - Declara uma variável, opcionalmente, inicializando-a com um valor.

`let` - Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor.

`const` - Declara uma constante apenas de leitura.

2.2 Tempos de amarração

- O tempo de vida das variáveis em JavaScript começa quando elas são declaradas.
- Variáveis locais são descartadas quando a função é completada.
- Variáveis globais são descartadas quando uma página é fechada.
- Amarração tardia (dinâmica)

2.3 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
- JavaScript utiliza closures: A cada novo contexto criado dentro (inner) de um contexto já existente tem acesso a todas as variáveis definidas no "de fora" (outer):

```
function x(a1) {           // "x" tem acesso a "a"
  var a2;
  function y(b1) {       // "y" tem acesso a "a" e "b"
    var b2;
    function z(c1) {    // "z" tem acesso a "a", "b", e "c"
      var c2;
```

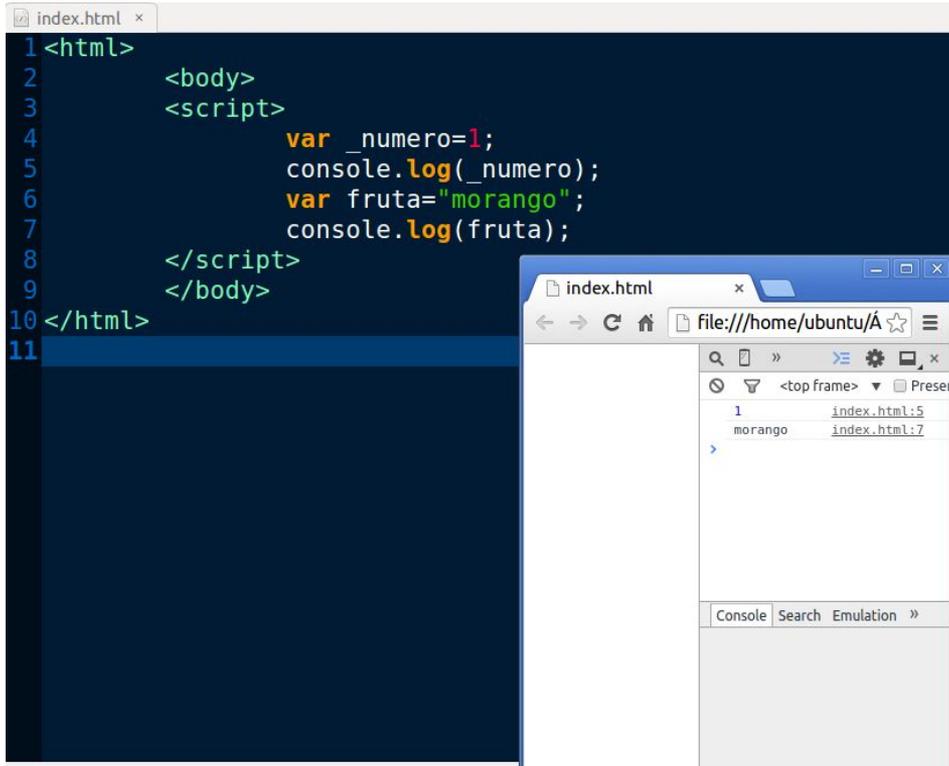
2.3 Closures

- Closures são feitas através de cópia da referência das variáveis e não dos valores, o que pode gerar problemas com efeitos colaterais
- O interpretador JavaScript mantém uma pilha de *execution context* conforme as funções vão sendo chamadas. E é essa pilha que vai fornecer para o interpretador a informação se o escopo da variável terminou ou não.

2.4 Identificadores

- A linguagem JavaScript faz distinção entre maiúsculas e minúsculas (case-sensitive)
- Composto por caracteres alfanuméricos e o caractere sublinhado (_)
- Começar por um caractere alfabético ou sublinhado
- O nome da variável não deve ser uma palavra reservada.

2.4 Identificadores



The image shows a code editor window on the left and a browser window on the right. The code editor displays the following HTML and JavaScript code:

```
1 <html>
2   <body>
3     <script>
4       var _numero=1;
5       console.log(_numero);
6       var fruta="morango";
7       console.log(fruta);
8     </script>
9   </body>
10 </html>
11
```

The browser window shows the file:///home/ubuntu/Á path and a console output:

```
1 index.html:5
morango index.html:7
```

The browser window also shows a console tab at the bottom with the text "Console Search Emulation »".

2.4 Identificadores

- Lista de palavras reservadas:

break	default	função	return	var
case	excluir	if	switch	void
catch	do	em	this	while
const	else	instanceof	throw	por
continue	finally	let	try	
debugger	for	novo	typeof	

2.4 Identificadores

- Lista de palavras reservadas futuras
- Na linguagem JavaScript, palavras-chave reservadas futuras não devem ser usadas como identificadores, mesmo que não tenham um significado especial na versão atual.

<code>CLASS</code>	<code>enum</code>	<code>export</code>
<code>extends</code>	<code>import</code>	<code>super</code>

2.5 Ambiente e escopo de amarração

- A linguagem JavaScript tem dois escopos: global e local.
- Variáveis definidas no interior de uma função não podem ser acessadas fora da função
- Funções definidas no escopo global podem acessar todas variáveis definidas no escopo global
- Funções aninhadas podem acessar todas as variáveis definidas na função externa e as variáveis na qual a função externa tem acesso.

2.5 Ambiente e escopo de amarração

// As seguintes variáveis são definidas no escopo

global

```
var num1 = 20,  
    num2 = 3,  
    nome = "Chamahk";
```

// Esta função é definida no escopo global

```
function multiplica() {  
    return num1 * num2;  
}
```

`multiplica();` // Retorna 60

// Um exemplo de função aninhada

```
function getScore () {  
    var num1 = 2,  
        num2 = 3;  
  
    function add() {  
        return nome + " scored " + (num1 + num2);  
    }  
  
    return add();  
}
```

`getScore();` // Retorna "Chamahk scored 5"

2.5 Ambiente e escopo de amarração

```
var variavel = 'fora';  
  
function escopo() {  
    variavel = 'dentro';  
  
    console.log(variavel); //dentro  
}  
  
escopo();  
  
console.log(variavel); //dentro
```

```
var variavel = 'fora';  
  
function escopo(variavel) {  
    variavel = 'dentro';  
  
    console.log(variavel); //dentro  
}  
  
escopo();  
  
console.log(variavel); //fora
```

```
function escopo() {  
    variavel = 'dentro';  
  
    console.log(variavel); //dentro  
}  
  
escopo();  
  
console.log(variavel); //não está  
definida
```

2.5 Ambiente e escopo de amarração

- Variáveis com escopo de bloqueio:

```
let x = 10;
var y = 10;
{
  let x = 5;
  var y = 5;
  {
    let x = 2;
    var y = 2;
    document.write("x: " + x + "<br/>");
    document.write("y: " + y + "<br/>");
    // Output:
    // x: 2
    // y: 2
  }
}
```

```
document.write("x: " + x + "<br/>");
document.write("y: " + y + "<br/>");
// Output:
// x: 5
// y: 2
}

document.write("x: " + x + "<br/>");
document.write("y: " + y + "<br/>");
// Output:
// x: 10
// y: 2
```

3. Valores e tipos de dados

3.1 Sistemas e tipos

- O JavaScript possui tipagem dinâmica
- As variáveis podem ser tipadas ou não tipadas
- Não precisa declarar o tipo explicitamente, o tipo é definido de acordo com o valor da variável

Tipos de Dados Primitivos

- String(cadeia de caracteres)

```
var string = 'Joao';
```

- Número `var numero = 4;`

- Booleano `var booleano = true;`

Tipos de dados compostos

Os tipos de dados compostos (de referência) são:

- Object (array, functions)

```
var vetor = [1, 'turquesa', false, 2];
```

```
var objeto = document.querySelector('h1');
```

```
var objeto = $('h1');
```

3.2 Tipos compostos: Objeto

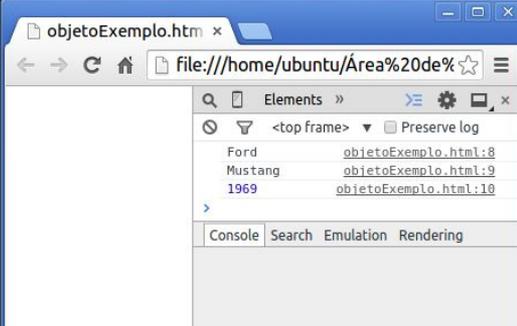
Objetos são coleções de propriedades e métodos. Métodos são funções de um objeto. Uma propriedade é um valor ou um conjunto de valores (no formato de uma matriz ou um objeto) que é membro de um objeto.

JavaScript suporta quatro tipos de objetos:

- Objetos intrínsecos, como **Array**.
- Objetos criados por você.
- Objetos de host, como **window (browser)** e **document**.
- Objetos ActiveX. (Arquivos externos)

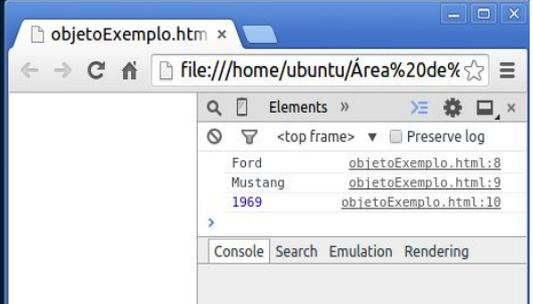
3.2 Tipos compostos: Objeto

```
objetoExemplo.html x
1 <html>
2   <body>
3     <script>
4       var meuCarro = new Object();
5       meuCarro.fabricacao = "Ford";
6       meuCarro.modelo = "Mustang";
7       meuCarro.ano = 1969;
8       console.log(meuCarro.fabricacao);
9       console.log(meuCarro.modelo);
10      console.log(meuCarro.ano);
11    </script>
12  </body>
13 </html>
14
```



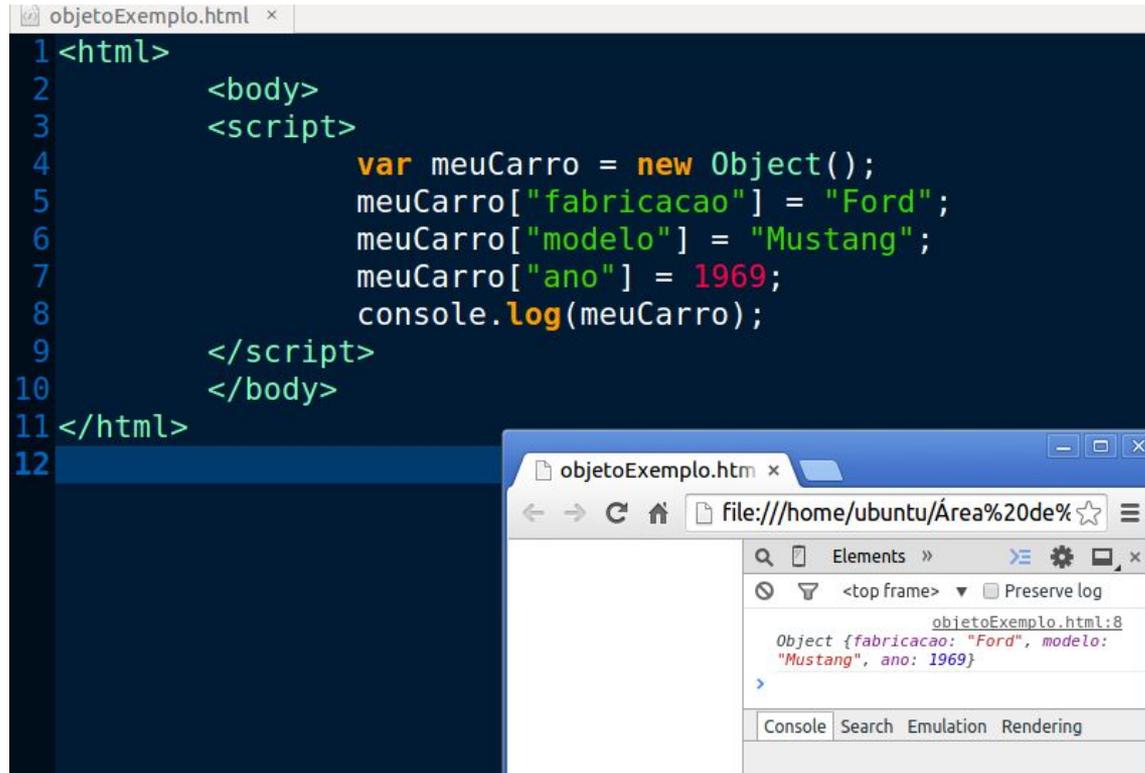
Value	File
Ford	objetoExemplo.html:8
Mustang	objetoExemplo.html:9
1969	objetoExemplo.html:10

```
objetoExemplo.html x
1 <html>
2   <body>
3     <script>
4       var meuCarro = new Object();
5       meuCarro["fabricacao"] = "Ford";
6       meuCarro["modelo"] = "Mustang";
7       meuCarro["ano"] = 1969;
8       console.log(meuCarro.fabricacao);
9       console.log(meuCarro.modelo);
10      console.log(meuCarro.ano);
11    </script>
12  </body>
13 </html>
14
```



Value	File
Ford	objetoExemplo.html:8
Mustang	objetoExemplo.html:9
1969	objetoExemplo.html:10

3.2 Tipos compostos: Objeto



The image shows a code editor window titled 'objetoExemplo.html' with the following code:

```
1 <html>
2   <body>
3     <script>
4       var meuCarro = new Object();
5       meuCarro["fabricacao"] = "Ford";
6       meuCarro["modelo"] = "Mustang";
7       meuCarro["ano"] = 1969;
8       console.log(meuCarro);
9     </script>
10  </body>
11 </html>
12
```

Below the code editor is a browser window titled 'objetoExemplo.htm' showing the output of the console log:

```
Object {fabricacao: "Ford", modelo: "Mustang", ano: 1969}
```

3.3 Tipos compostos: Array

- No JavaScript, um Array não reserva uma alocação contínua de memória de tamanho pré definido ele é simplesmente um objeto glorificado com um construtor único, com uma sintaxe literal e com um conjunto adicional de propriedades e de métodos herdados de um protótipo de Array.

3.3 Tipos compostos: Array

```
1  var a = [];  
2  a[0] = 8;  
3  var b = [];  
4  b[0] = 8;  
5  b[1] = 9;  
6  console.log(a.length);//1  
7  console.log(b.length);//2  
8  console.log(a[0]);//8  
9  console.log(b[0]);//8
```

```
1  var a = new Array(8);  
2  var b = new Array(8,9);  
3  console.log(a.length); //8  
4  console.log(b.length); //2  
5  console.log(a[0]); //undefined  
6  console.log( b[0]); //8
```

3.3 Tipos compostos: Array

- Um Array pode conter qualquer objeto do tipo primitivo. Dados de tipos diferentes podem coexistir no mesmo Array.

```
1  var a = new Array(8, "cachorro", true, "fruta");
2  console.log(a);
3  if(a[2]==true){
4      console.log("verdadeiro");
5  }
```

3.3 Tipos compostos: Array

Quando você usa *typeof* no JavaScript para um Array, ele retorna "object". **Como sei se meu objeto é do tipo Array?**

```
1  function isArray(o) {  
2    return Object.prototype.toString.call(o) === "[object Array]";  
3  }
```

3.3 Tipos compostos: Array

Métodos predefinidos:

- **concat** faz cópia simples do Array e adiciona os argumentos
- **join** cria um string do Array. Adiciona o argumento como cola entre cada membro do Array.
- **shift** remove e retorna o primeiro elemento
- **pop** remove e retorna o último elemento
- **unshift** anexa os argumentos na frente do Array
- **push** anexa os argumentos no final do Array

3.3 Tipos compostos: Array

Métodos predefinidos:

- **reverse** inverte o Array sem copiá-lo
- **slice** faz cópia simples da porção do Array delimitada por argumentos do índice
- **splice** remove elementos especificados do Array, e os substitui com argumentos opcionais
- **sort** classifica o Array sem copiá-lo, opcionalmente usando um argumento comparador
- **toString** chama *join* sem passar um argumento

4. Armazenamento

4.1 Armazenamento em memória principal

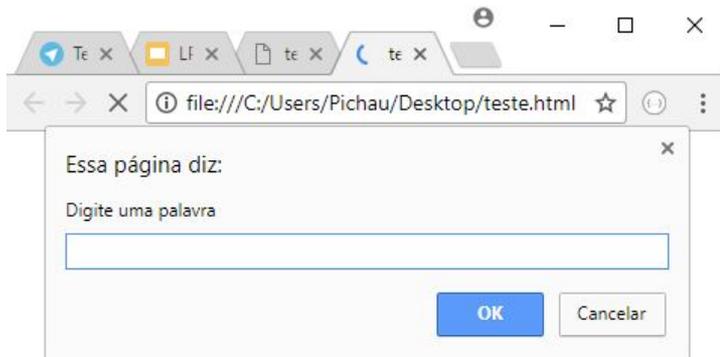
- Endereço da variável não é acessível
- Alocação dinâmica
- Alocação de memória por conta da LP
- Liberação de memória por conta da LP
- Presença do coletor de lixo

4.2 I/O

- Javascript interage com o usuário por meio de caixas de mensagem

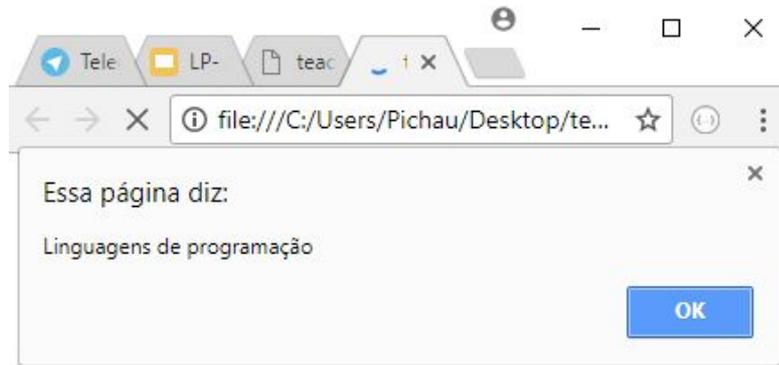
Entrada:

```
var palavra = prompt("Digite uma palavra", "");
```



Saída:

```
alert(palavra);
```



4.3 Serialização

- **Json (JavaScript Object Notation)**
- **JSON.stringify()** converte valores em JavaScript para uma String JSON.
- **JSON.parse()** converte valores em JSON em objeto JavaScript
- Leitura e escrita de arquivos não é possível no lado do cliente.
- Uso de ajax para leitura de arquivos

```
var comida =  
["strogonoff","frango","macarrao"];  
  
var bebida = ["refrigerante","suco"];  
  
var doces =  
["brigadeiro","beijinho","cajuzinho","bolo"];  
  
var matriz = [comida,bebida,doces];  
  
var resultado = JSON.stringify(matriz);  
  
console.log(resultado);  
//[["strogonoff","frango","macarrao"],["refri  
gerante","suco"],["brigadeiro","beijinho","c  
ajuzinho","bolo"]]
```

5. Expressões e comandos

5.1 Atribuição

- Um operador de atribuição atribui um valor ao operando à sua esquerda baseado no valor do operando à direita.
- O operador de atribuição básico é o igual (=), que atribui o valor do operando à direita ao operando à esquerda. Isto é, $x = y$ atribui o valor de y a x .
- Os outros operadores de atribuição são encurtamentos de operadores padrão, como mostrado na tabela a seguir.

Nome	Operador encurtado	Significado
Atribuição	$x = y$	$x = y$
Atribuição de adição	$x += y$	$x = x + y$
Atribuição de subtração	$x -= y$	$x = x - y$
Atribuição de multiplicação	$x *= y$	$x = x * y$
Atribuição de divisão	$x /= y$	$x = x / y$
Atribuição de resto	$x %= y$	$x = x \% y$
Atribuição exponencial	$x **= y$	$x = x ** y$
Atribuição bit-a-bit por deslocamento á esquerda	$x <<= y$	$x = x << y$
Atribuição bit-a-bit por deslocamento á direita	$x >>= y$	$x = x >> y$
Atribuição de bit-a-bit deslocamento á direita não assinado	$x >>>= y$	$x = x >>> y$
Atribuição AND bit-a-bit	$x \&= y$	$x = x \& y$
Atribuição XOR bit-a-bit	$x \^= y$	$x = x \^ y$
Atribuição OR bit-a-bit	$x = y$	$x = x y$

5.2 Comparação

- Um operador de comparação compara seus operandos e retorna um valor lógico baseado em se a comparação é verdadeira.
- Os operandos podem ser numéricos, strings, lógicos ou objetos.
- Strings são comparadas com base em ordenação lexicográfica utilizando valores Unicode.
- Na maioria dos casos, se dois operandos não são do mesmo tipo, o JavaScript tenta convertê-los para um tipo apropriado.

Operador	Descrição	Exemplos que retornam verdadeiro
Igual (==)	Retorna verdadeiro caso os operandos sejam iguais.	<code>3 == var1</code> <code>"3" == var1</code> <code>3 == '3'</code>
Não igual (!=)	Retorna verdadeiro caso os operandos não sejam iguais.	<code>var1 != 4</code> <code>var2 != "3"</code>
Estritamente igual (===)	Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo. Veja também <code>Object.is</code> e <code>igualdade em JS</code> .	<code>3 === var1</code>
Estritamente não igual (!==)	Retorna verdadeiro caso os operandos não sejam iguais e/ou não sejam do mesmo tipo.	<code>var1 !== "3"</code> <code>3 !== '3'</code>
Maior que (>)	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.	<code>var2 > var1</code> <code>"12" > 2</code>
Maior que ou igual (>=)	Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita.	<code>var2 >= var1</code> <code>var1 >= 3</code>
Menor que (<)	Retorna verdadeiro caso o operando da esquerda seja menor que o da direita.	<code>var1 < var2</code> <code>"12" < "2"</code>
Menor que ou igual (<=)	Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita.	<code>var1 <= var2</code> <code>var2 <= 5</code>

5.3 Aritmético

- Operadores aritméticos tomam valores numéricos (sejam literais ou variáveis) como seus operandos e retornam um único valor numérico.
- Os operadores aritméticos padrão são os de soma (+), subtração (-), multiplicação (*) e divisão (/).
- Estes operadores trabalham da mesma forma como na maioria das linguagens de programação quando utilizados com números de ponto flutuante (em particular, repare que divisão por zero produz um NaN).

```
1 console.log(1 / 2); /* imprime 0.5 */
2 console.log(1 / 2 == 1.0 / 2.0); /* isto também é verdadeiro */
```

Operador	Descrição	Exemplo
Módulo (%)	Operador binário. Retorna o inteiro restante da divisão dos dois operandos.	12 % 5 retorna 2.
Incremento (++)	Operador unário. Adiciona um ao seu operando. Se usado como operador prefixado (++x), retorna o valor de seu operando após a adição. Se usado como operador pósfixado (x++), retorna o valor de seu operando antes da adição.	Se x é 3, então ++x define x como 4 e retorna 4, enquanto x++ retorna 3 e, somente então, define x como 4.
Decremento (-)	Operador unário. Subtrai um de seu operando. O valor de retorno é análogo àquele do operador de incremento.	Se x é 3, então --x define x como 2 e retorna 2, enquanto x-- retorna 3 e, somente então, define x como 2.
Negação (-)	Operador unário. Retorna a negação de seu operando.	Se x é 3, então -x retorna -3.
Adição (+)	Operador unário. Tenta converter o operando em um número, sempre que possível.	+ "3" retorna 3. + true retorna 1.
Operador de exponenciação (**)	Calcula a base elevada à potência do expoente, que é, base ^{expoente}	2 ** 3 retorna 8. 10 ** -1 retorna 0.1

5.4 Bit a bit

- Operadores bit a bit tratam seus operandos como um conjunto de 32 bits (zeros e uns), em vez de tratá-los como números decimais, hexadecimais ou octais.
- Por exemplo, o número decimal nove possui uma representação binária 1001. Operadores bit a bit realizam suas operações nestas representações, mas retornam valores numéricos padrões do JavaScript.

Operador	Expressão	Descrição
AND	<code>a & b</code>	Retorna um 1 para cada posição em que os bits da posição correspondente de ambos operandos sejam uns.
OR	<code>a b</code>	Retorna um 0 para cada posição em que os bits da posição correspondente de ambos os operandos sejam zeros.
XOR	<code>a ^ b</code>	Retorna um 0 para cada posição em que os bits da posição correspondente são os mesmos. [Retorna um 1 para cada posição em que os bits da posição correspondente sejam diferentes.]
NOT	<code>~ a</code>	Inverte os bits do operando.
Deslocamento à esquerda	<code>a << b</code>	Desloca <code>a</code> em representação binária <code>b</code> bits à esquerda, preenchendo com zeros à direita.
Deslocamento à direita com propagação de sinal	<code>a >> b</code>	Desloca <code>a</code> em representação binária <code>b</code> bits à direita, descartando bits excedentes.
Deslocamento à direita com preenchimento zero	<code>a >>> b</code>	Desloca <code>a</code> em representação binária <code>b</code> bits à direita, descartando bits excedentes e preenchendo com zeros à esquerda.

5.5 Lógico

- Operadores lógicos são utilizados tipicamente com valores booleanos (lógicos); neste caso, retornam um valor booleano

Operador	Utilização	Descrição
AND lógico (<code>&&</code>)	<code>expr1</code> <code>&&</code> <code>expr2</code>	(E lógico) - Retorna <code>expr1</code> caso possa ser convertido para falso; senão, retorna <code>expr2</code> . Assim, quando utilizado com valores booleanos, <code>&&</code> retorna verdadeiro caso ambos operandos sejam verdadeiros; caso contrário, retorna falso.
OU lógico (<code> </code>)	<code>expr1</code> <code> </code> <code>expr2</code>	(OU lógico) - Retorna <code>expr1</code> caso possa ser convertido para verdadeiro; senão, retorna <code>expr2</code> . Assim, quando utilizado com valores booleanos, <code> </code> retorna verdadeiro caso ambos os operandos sejam verdadeiro; se ambos forem falsos, retorna falso.
NOT lógico (<code>!</code>)	<code>!expr</code>	(Negação lógica) Retorna falso caso o único operando possa ser convertido para verdadeiro; senão, retorna verdadeiro.

5.6 String

- O operador (+) pode ser usado na concatenação de strings, operando sobre strings.

```
console.log("minha " + "string"); // exibe a string "minha string".
```

5.7 Condicional Ternário

- É um tipo de operador com 3 operandos, de modo que o operador assume um valor de acordo com uma condição, sintaxe:

condicao ? valor1 : valor2

```
1 | var status = (idade >= 18) ? "adulto" : "menor de idade";
```

5.8 Unário

- Um operador unário é uma operação com apenas um operando, Ex: delete, typeof e void.

```
1 | delete nomeObjeto;  
2 | delete nomeObjeto.propriedade;  
3 | delete nomeObjeto[indice];  
4 | delete propriedade; // válido apenas dentro de uma declaração with
```

```
1 | typeof operando  
2 | typeof (operando)
```

5.9 Relacionais

- Um operador relacional compara seus operandos e retorna um valor booleano baseado em se a comparação é verdadeira. Ex: `in` e `instanceof`

```
1 | nomePropriedadeOuNumero in nomeObjeto
```

```
1 | nomeObjeto instanceof tipoObjeto
```

5.10 Curto Circuito

- JavaScript usa curto circuito nos operadores lógicos:
 - false && qualquercoisa é avaliado em curto-circuito como falso.
 - true || qualquercoisa é avaliado em curto-circuito como verdadeiro.
- Precedência de operadores em JS:

Tipo de operador	Operadores individuais
membro	· []
chamada / criação de instância	() new
negação / incremento	! ~ - + ++ -- typeof void delete
multiplicação / divisão	* / %
adição / subtração	+ -
deslocamento bit a bit	<< >> >>>
relacional	< <= > >= in instanceof
igualdade	== != === !==
E bit a bit	&
OU exclusivo bit a bit	^
OU bit a bit	
E lógico	&&
OU lógico	
condicional	?:
atribuição	= += -= *= /= %= <<= >>= >>>= &= ^= =

5.11 Tipos de comandos e expressões

- JS utiliza o “for”, “while” e o “do/while” , idênticos aos de C e Java:

```
1  = for (i = 0; i < 5; i++) {  
2      text += "The number is " + i + "<br>";  
3  }  
4  
5  = while (i < 10) {  
6      text += "The number is " + i;  
7      i++;  
8  }  
9  
10 = do {  
11     text += "The number is " + i;  
12     i++;  
13 }  
14 while (i < 10);
```

5.11 Tipos de comandos e expressões

- For-in: Executa uma ou mais instruções para cada propriedade de um objeto:

```
1   a = {"a" : "Athens" , "b" : "Belgrade", "c" : "Cairo"}
2
3   // Iterate over the properties.
4   var s = ""
5   for (var key in a) {
6       s += key + ": " + a[key];
7       s += "<br />";
8   }
9   document.write (s);
10
11  // Output:
12  // a: Athens
13  // b: Belgrade
14  // c: Cairo
```

5.11 Tipos de comandos e expressões

- O for-each repete uma variável específica sobre todos os valores das propriedades do objeto. Para cada propriedade distinta, uma declaração específica é executada.
- O loop for-of percorre objetos iteráveis (incluindo Array, Map, Set, o objeto arguments e assim por diante), chamando uma função personalizada com instruções a serem executadas para o valor de cada objeto distinto.

5.11 Tipos de comandos e expressões

- For-each/For-of

```
1 var sum = 0;
2 var obj = {prop1: 5, prop2: 13, prop3: 8};
3
4 for each (var item in obj) {
5     sum += item;
6 }
7
8 console.log(sum); // escreve no log "26", que é 5+13+8
```

```
1 let iterable = [10, 20, 30];
2
3 for (const value of iterable) {
4     console.log(value);
5 }
6 // 10
7 // 20
8 // 30
```

5.12 Detalhes da linguagem

- JS utiliza labels e os comandos “return”, “break” e “continue” de maneira similar a Java.
- Ponto e vírgula: Seu uso é opcional em JS com a finalidade de separar instruções, geralmente ele pode ser omitido se duas instruções forem escritas em linhas separadas.
- Em JavaScript, temos o ASI, Automatic Semicolon Insertion, ele é o responsável por inserir automaticamente “ ; ” em determinadas situações. Esse é assunto polêmico!

5.12 Detalhes da linguagem

- O “;” serve como um delimitador de statements. Mas devido ao ASI, o “\n” também irá funcionar como delimitador de statement, exceto nos seguintes casos:
 1. O statement possui um parêntese, array literal ou objeto literal não fechado ou acaba de qualquer outra forma a qual não seja um modo válido de finalizar um statement.
 2. A linha inteira é um -- ou ++ (neste caso, irá incrementar/decrementar o próximo token)
 3. É um for(), while(), do, if() ou else e não existe {
 4. A próxima linha começa com [, (, +, -, *, /, ,, ., ou qualquer outro operador binário que só pode ser encontrado entre dois tokens em uma única expressão.

5.12 Detalhes da linguagem

- Existe mais uma regra sobre ASI na linguagem que cobre casos especiais. Esses são chamados de “restricted productions”. Esta regra fala que, caso exista um `\n` logo após um `return`, `throw`, `break`, `continue` ou um `label` o `statement` sempre será finalizado, sem exceções.
- Existem outras regras do ASI que não estão no escopo deste trabalho.
- Contudo, é recomendado o uso do “;” sempre por uma questão de legibilidade.

5.13 Expressões

- Uma expressão consiste em qualquer unidade válida de código que é resolvida como um valor.
- Em JavaScript podemos categorizar as expressões da seguinte maneira:

Aritmética: é avaliada como um número, por exemplo 3.14159.

String: é avaliada como uma string de caracteres, por exemplo, "Fred" ou "234".

Lógica: é avaliada como verdadeira ou falsa. (Costuma envolver operadores lógicos).

Expressões primárias: Palavras reservadas e expressões gerais do JavaScript. (Ex: this)

Expressão lado esquerdo: atribuição à esquerda de valores. (Ex: New)

5.13 Expressões

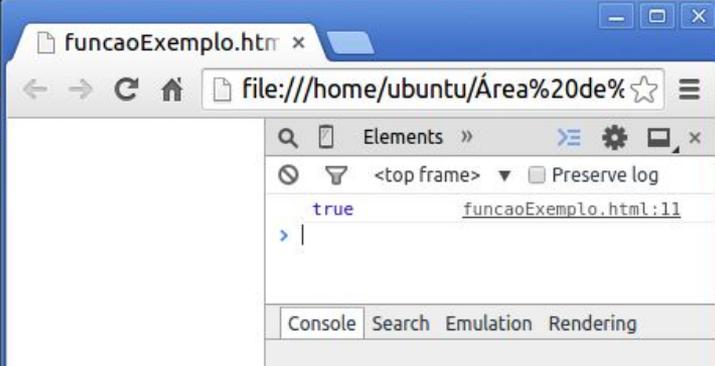
- É possível, em JS fazer agregação de valores para arrays e objects:

```
1 []  
2 [1+2, 3+4]  
3  
4 var p = {x:2.3, Y:-1.2}  
5 var q = {}  
6 q.x = 2.3; q.y = -1.2
```

6. Modularização

6.1 Subprogramas e parâmetros

```
1 <html>
2     <body>
3     <script>
4         var a=10;
5         function verificaNumeroPar(n)
6         {
7             if ( n%2==0 ) return true;
8             else return false;
9         }
10        console.log(verificaNumeroPar(a));
11    </script>
12    </body>
13 </html>
14
```



The screenshot shows a web browser window with the file path `file:///home/ubuntu/Área%20de%`. The browser's developer tools are open, showing the console with the output `true` from `funcaoExemplo.html:11`. The console also includes tabs for 'Elements', 'Search', 'Emulation', and 'Rendering'.

6.1 Subprogramas e parâmetros

Usando modularização na criação de objetos

```
funcaoObjetoExemplo.html x
1 <html>
2   <body>
3     <script>
4       var p1=new Pessoa("Isabel","21 anos", "feminino");
5       var carro= new Carro("Volkswagen","Fusca","1950",p1);
6       function Pessoa(nome, idade, sexo) {
7         this.nome = nome;
8         this.idade = idade;
9         this.sexo = sexo;
10      }
11      function Carro(fabricacao, modelo, ano, proprietario) {
12        this.fabricacao = fabricacao;
13        this.modelo = modelo;
14        this.ano = ano;
15        this.proprietario = proprietario;
16        this.mostraCarro = mostraCarro;
17      }
18      function mostraCarro(){
19        var resultado = "Um belo " + this.modelo + " " + this.fabricacao+ " " +
20        this.ano+"Dono:"+this.proprietario.nome;
21        console.log(resultado);
22      };
23      carro.mostraCarro();
24    </script>
25  </body>
26 </html>
```



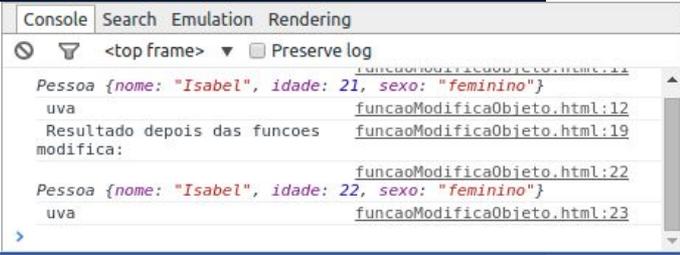
```
Console Search Emulation Rendering
<top frame> Preserve log
Um belo Fusca Volkswagen 1950 funcaoObjetoExemplo.html:20
Dono:Isabel
```

6.1 Subprogramas e parâmetros

- Passagem por cópia:
- Parâmetros primitivos (como um número) são passados para as funções por **valor**; o valor é passado para a função, mas se a função altera o valor do parâmetro, esta mudança não reflete globalmente ou na função chamada.
- Passagem por referência:
- Se você passar um objeto (ou seja, um valor não primitivo, tal como Array ou um objeto definido por você) como um parâmetro e a função alterar as propriedades do objeto, essa mudança é visível fora da função, conforme mostrado no exemplo a seguir:

6.1 Subprogramas e parâmetros

```
funcaoModificaObjeto.html x
1 <html>
2   <body>
3   <script>
4     var f="uva";
5     var p1=new Pessoa("Isabel",21, "feminino");
6     function Pessoa(nome, idade, sexo) {
7       this.nome = nome;
8       this.idade = idade;
9       this.sexo = sexo;
10    }
11    console.log(p1);
12    console.log(f);
13    function modificaIdade(pessoa){
14      pessoa.idade++;
15    }
16    function modificaFruta(fruta){
17      fruta="banana";
18    }
19    console.log("Resultado depois das funcoes modifica:");
20    modificaIdade(p1);
21    modificaFruta(f);
22    console.log(p1);
23    console.log(f);
24  </script>
25 </body>
```



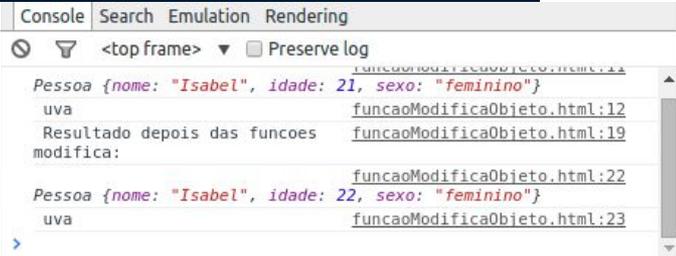
Console	Search	Emulation	Rendering
<input type="checkbox"/> Preserve log	<top frame>		
funcaoModificaObjeto.html:11			
Pessoa {nome: "Isabel", idade: 21, sexo: "feminino"}			
uva funcaoModificaObjeto.html:12			
Resultado depois das funcoes funcaoModificaObjeto.html:19			
modifica:			
funcaoModificaObjeto.html:22			
Pessoa {nome: "Isabel", idade: 22, sexo: "feminino"}			
uva funcaoModificaObjeto.html:23			

6.1 Subprogramas e parâmetros

- Funções também podem ser criadas por uma **expressão de função**. Tal função pode ser **anônima**; ele não tem que ter um nome.
- No entanto, um nome pode ser fornecido com uma expressão de função e pode ser utilizado no interior da função para se referir a si mesma.

6.1 Subprogramas e parâmetros

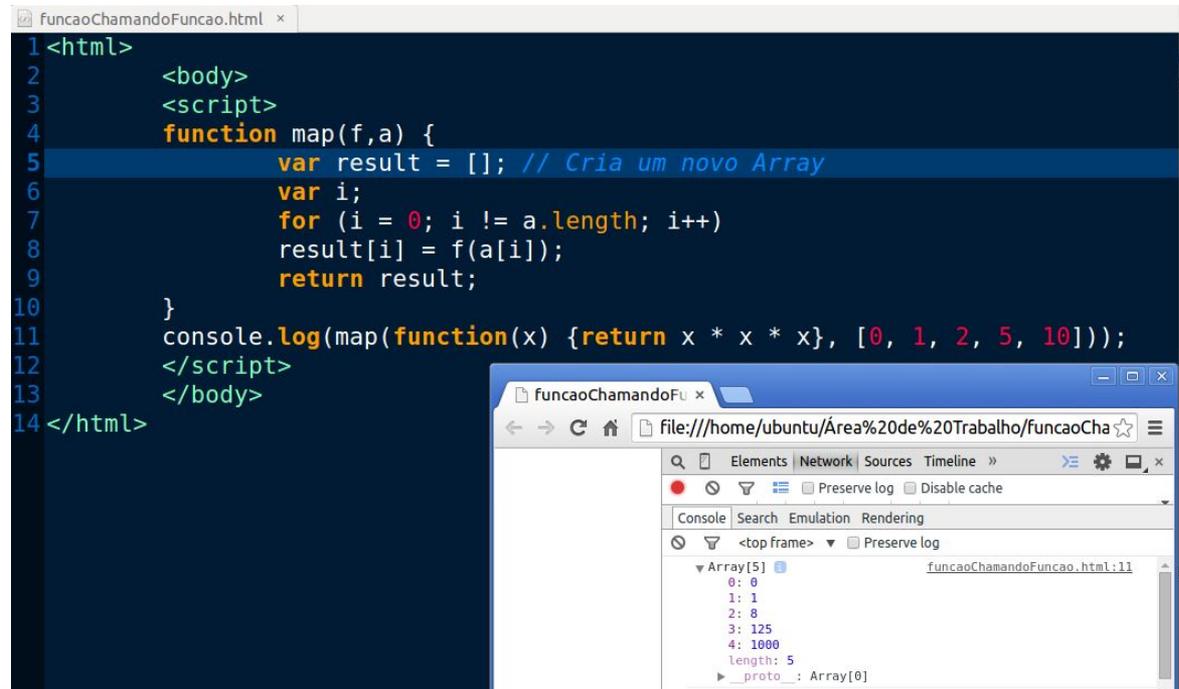
```
1 <html>
2   <body>
3     <script>
4       var f="uva";
5       var p1=new Pessoa("Isabel",21, "feminino");
6       function Pessoa(nome, idade, sexo) {
7         this.nome = nome;
8         this.idade = idade;
9         this.sexo = sexo;
10      }
11      console.log(p1);
12      console.log(f);
13      var modificaIdade=function(pessoa){
14        pessoa.idade++;
15      }
16      function modificaFruta(fruta){
17        fruta="banana";
18      }
19      console.log("Resultado depois das funcoes modifica:");
20      modificaIdade(p1);
21      modificaFruta(f);
22      console.log(p1);
23      console.log(f);
24    </script>
```



Console	Search	Emulation	Rendering
<top frame> <input type="checkbox"/> Preserve log			
Pessoa {nome: "Isabel", idade: 21, sexo: "feminino"} uva			
Resultado depois das funcoes modifica: Pessoa {nome: "Isabel", idade: 22, sexo: "feminino"} uva			

6.1 Subprogramas e parâmetros

- As expressões de função são convenientes ao passar uma função como um argumento para outra função.



```
1 <html>
2   <body>
3   <script>
4     function map(f,a) {
5       var result = []; // Cria um novo Array
6       var i;
7       for (i = 0; i != a.length; i++)
8         result[i] = f(a[i]);
9       return result;
10    }
11    console.log(map(function(x) {return x * x * x}, [0, 1, 2, 5, 10]));
12  </script>
13  </body>
14 </html>
```

The screenshot shows a browser window with the file `funcaoChamandoFu.html`. The browser's developer tools are open, showing the console output of the JavaScript code. The console displays an array of five elements: `0, 1, 8, 125, 1000`. The array object also shows `length: 5` and `__proto__: Array[0]`.

6.1 Subprogramas e parâmetros

- Recursividade:

```
funcaoRecursiva.html x
1 <html>
2   <script>
3     function fatorial(n){
4       if ((n == 0) || (n == 1))
5         return 1;
6       else
7         return (n * fatorial(n - 1));
8     }
9   </script>
10 </html>
```

6.2 Parâmetros Rest

- É possível ter uma quantidade de parâmetros variável:

ECMAScript 5

```
function f(x, y) {  
  var a = Array.prototype.slice.call(arguments, 2);  
  return (x + y) * a.length;  
};  
f(1, 2, "hello", true, 7) === 9;
```

ECMAScript 6

```
function f(x, y, ...a) {  
  return (x + y) * a.length  
}  
f(1, 2, "hello", true, 7) === 9
```

6.3 Pacotes e espaço de nome

- Em JavaScript, um espaço de nomes é simplesmente outro objeto contendo métodos, propriedades e objetos.
- A ideia por trás de criar um espaço de nomes em JavaScript é simples: cria-se um objeto global e todas as variáveis, métodos e chamadas de função tornam-se propriedades daquele objeto. O uso de espaços de nomes também reduz a chance de conflitos de nomes em uma aplicação, já que os objetos de cada aplicação são propriedades de um objeto global definido pela aplicação

6.3 Pacotes e espaço de nome

```
1  var AppLabirintorium = function() {  
2      return {  
3          metodo01 : function() {  
4              alert("Método número 01");  
5          },  
6          metodo02 : function() {  
7              alert("Método número 02");  
8          }  
9      };  
10 }();
```

Para utilizar o namespace é simples, basta seguir a hierarquia de objetos

```
1  AppLabirintorium.metodo01();  
2  AppLabirintorium.metodo02();
```

6.4 Arquivos separados

- Vantagens de JavaScript Externo
- A colocação de scripts em arquivos externos possui algumas vantagens:
 - Separa HTML e código
 - Isso torna o HTML e o JavaScript mais fáceis de ler e manter
 - Arquivos de JavaScript em cache podem acelerar cargas de página
- Para adicionar vários arquivos de script a uma página - use várias tags de script:

```
<script src="/js/myScript1.js"></script>
```

```
<script src="myScript1.js"></script>
```

6.4 Arquivos separados

- O processo de minificação de arquivos, portanto, tem o objetivo de remover caracteres dispensáveis de forma que o arquivo final tenha um tamanho menor que o arquivo original, mas o mesmo resultado.
- No caso do JavaScript, ainda existem outras possíveis otimizações, como a substituição de nomes grandes de variáveis ou funções por nomes mais curtos. Além disso, o processo pode ser acompanhado de um "embaralhamento" do código, que tem o objetivo de ofuscar o funcionamento do código, mas sem interferir em seu resultado principal.
- Você pode dividir o código em dezenas de arquivos js, e compilar um arquivo único, minificado, para ser entregue ao browser. Em geral, quanto menos arquivos você entrega ao browser, mais rápida fica a app.

7. Polimorfismo

7.1 Características

Os métodos não podem ser sobrescritos

Amarração tardia

Não possui herança múltipla

7.2 Coerção

- JavaScript é uma linguagem fracamente tipada.
- O tipo de uma variável é o tipo do valor dela.
- Em JavaScript, você pode executar operações em valores de tipos diferentes sem gerar uma exceção.
- Regras de coerção:
 - número + string => número vira string.
 - booleano + string => booleano vira string.
 - número + booleano => booleano vira número.

```
var x = 2000;  
var y = "Hello";
```

```
x = x + y;  
document.write(x);
```

```
// Output:  
// 2000Hello
```

```
var x = true;  
var y = "Hello";
```

```
x = x + y;  
document.write(x);
```

```
// Output:  
// trueHello
```

```
var x = 2000;  
var y = true;
```

```
x = x + y;  
document.write(x);
```

```
// Output:  
// 2001
```

7.3 Sobrecarga

- Sobrecarga de operador

```
var a = 2;
```

```
var b = 4;
```

```
console.log(a+b); //6
```

```
var a = 'Hello';
```

```
var b = 'World';
```

```
console.log(a+b); //HelloWorld
```

7.4 Paramétrico

- Se manifesta na declaração de tipos, visto que o identificador `var` declara tipos primitivo e objetos
- É possível utilizar diversos tipos em uma mesma variável

```
var Cachorro = {  
    raça: "Yorkshire",  
    cor: "Caramelo",  
};
```

```
var qtd = 1;  
var dono = "João";  
var vet = [dono,qtd,Cachorro];  
console.log(vet);  
//Output  
//[“João, 1, {...}”]
```

7.5 Inclusão

ECMAScript 5

```
var Shape = function (id, x, y) {  
  this.id = id;  
  this.move(x, y);  
};  
Shape.prototype.move = function (x, y) {  
  this.x = x;  
  this.y = y;  
};
```

ECMAScript 6

```
class Shape {  
  constructor (id, x, y) {  
    this.id = id  
    this.move(x, y)  
  }  
  move (x, y) {  
    this.x = x  
    this.y = y  
  }  
}
```

7.5 Inclusão

ECMAScript 5

```
var Rectangle = function (id, x, y, width, height) {  
  Shape.call(this, id, x, y);  
  this.width = width;  
  this.height = height;  
};  
Rectangle.prototype = Object.create(Shape.prototype);  
Rectangle.prototype.constructor = Rectangle;  
var Circle = function (id, x, y, radius) {  
  Shape.call(this, id, x, y);  
  this.radius = radius;  
};  
Circle.prototype = Object.create(Shape.prototype);  
Circle.prototype.constructor = Circle;
```

ECMAScript 6

```
class Rectangle extends Shape {  
  constructor (id, x, y, width, height) {  
    super(id, x, y)  
    this.width = width  
    this.height = height  
  }  
}  
class Circle extends Shape {  
  constructor (id, x, y, radius) {  
    super(id, x, y)  
    this.radius = radius  
  }  
}
```

7.6 Herança

- JavaScript adotou programação baseada em **protótipos** é um estilo de programação orientada a objetos na qual não temos presença de classes.
- A reutilização de comportamento (equivalente à herança das linguagens baseadas em classes) é realizada através de um processo de decorar (ou expandir) objetos existentes que servem como *protótipos*. Este modelo também é conhecido como **sem classes**, **orientado a protótipo**, ou **programação baseada em exemplares**.

7.6 Herança

- O modelo baseado em protótipos provê herança dinâmica, que é o quando uma herança pode variar para objetos individuais.
- Funções podem ser propriedades de objetos, sendo executadas como objetos tipados livres

8. Excessões

8.1 Exceções

- De maneira similar a JAVA, JavaScript também manipula exceções através dos comandos `throw`, `try`, `catch` e `finally`:

```
1  try {  
2      throw "myException"; // lança uma exceção  
3  }  
4  catch (e) {  
5      // declarações de lidar com as exceções  
6      logMyErrors(e); // passar a exceção para o manipulador de erro  
7  }
```

8.1 Exceções

- A declaração `throw` lança uma exceção definida pelo usuário. A execução da função atual vai parar (as instruções após o `throw` não serão executadas), e o controle será passado para o primeiro bloco `catch` na pilha de chamadas. Se nenhum bloco `catch` existe entre as funções "chamadoras", o programa vai terminar.
- A declaração `try...catch` é composta por um bloco `try`, que contém uma ou mais declarações, e zero ou mais blocos `catch`, contendo declarações que especificam o que fazer se uma exceção é lançada no bloco `try`.
- O bloco `finally` contém instruções para executar após os blocos `try` e `catch`, mas antes das declarações seguinte a declaração `try...catch`. O bloco `finally` é executado com ou sem o lançamento de uma exceção

8.1 Exceções

```
1 openMyFile();
2 try {
3     writeMyFile(theData); //Isso pode lançar um erro
4 } catch(e) {
5     handleError(e); // Se temos um erro temos que lidar com ele
6 } finally {
7     closeMyFile(); // feche sempre o recurso
8 }
```

9. Concorrência

9.1 Processos e threads

- Se seu aplicativo deve executar computação suficiente para causar um atraso notável, você deve permitir que o documento seja carregado completamente antes.
- Executando essa computação, e você deve certificar-se de notificar o usuário de que a computação está em andamento e que o navegador não está pendurado
- Se for possível quebrar o seu cálculo em subtarefas discretas, você pode usar métodos como `setTimeout()`
- e `setInterval()` para executar as subtarefas em segundo plano enquanto atualiza um indicador de progresso que exibe feedback para o usuário.

9.2 Semáforos

- `setTimeout()` e `setInterval()` permitem que você registre uma função a ser invocada uma vez ou repetidamente após um determinado período de tempo decorrido.
- O método `setTimeout()` do objeto `Window` agenda uma função a ser executada após um número especificado de milissegundos decorridos.
- `SetTimeout()` retorna um valor que pode ser passado para `clearTimeout()` para cancelar a execução da função agendada.
- `SetInterval()` é como `setTimeout()`, exceto que a função agendada especificada é chamada repetidamente em intervalos do número especificado de milissegundos.

9.2 Semáforos

```
1 <script>
2 var myVar;
3 function runLevels(level)
4 {
5   switch(level) {
6     case 0: // Red light
7       console.log("Red: Level = " + level);
8       myVar = setTimeout(function(){
9         runLevels(level+1); }, 3000);
10      break;
11     case 1: // Green light
12       console.log("Green: Level = " + level);
13       myVar = setTimeout(function(){
14         runLevels(level+1); }, 3000);
15
16       break;
17     case 2: // Yellow light
18       console.log("Yellow: Level = " + level);
19       myVar = setTimeout(function(){
20         runLevels(0); }, 1000);
21       break;
22     default: // break
23       clearTimeout(myVar);
24   }
25 };
26 </script>
```

9.3 Suporte avançado de Javascript à programação concorrente

Quando uma função assíncrona é chamada, ela retorna uma Promise. Quando a função assíncrona retorna um valor, a Promise será resolvida com o valor retornado. Quando a função assíncrona lança uma exceção ou algum valor, a Promise será rejeitada com o valor lançado.

Uma função assíncrona pode conter uma expressão `await`, que pausa a execução da função assíncrona e espera pela resolução da Promise passada, e depois retoma a execução da função assíncrona e retorna o valor resolvido.

9.3 Suporte avançado de Javascript à programação concorrente

```
1  async function
2  pegarDadosProcessados(url) {
3  let v;
4  try {
5      v = await baixarDados(url);
6  } catch(e) {
7      v = await baixarDadosReservas(url);
8  }
9  return processarDadosNoWorker(v);
10 }
```

9.3 Suporte avançado de Javascript à programação concorrente

No event loop cada mensagem é processada completamente antes de outra mensagem ser processada.

```
1 while(queue.waitForMessage()){  
2   queue.processNextMessage();  
3 }
```

9.3 Suporte avançado de Javascript

à programação concorrente

- O JavaScript é um ambiente de sequência única, isso significa que não é possível executar vários scripts ao mesmo tempo.
- A especificação Web Workers (link em inglês) define uma API para geração de scripts de segundo plano no seu aplicativo da web. O Web Workers permite executar tarefas como disparar scripts de longa duração para executar tarefas muito dispendiosas, mas sem bloquear a interface de usuário ou outros scripts para manipular interações com o usuário.
- O Workers utiliza a transmissão de mensagem do tipo sequência para obter paralelismo. É perfeito para manter a interface atualizada, com desempenho e responsiva para os usuários

10. Frameworks

10.1 Frameworks

- Um framework, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica
- JavaScript deve muito da sua fama na comunidade de programação aos seus frameworks
- Grande empresas utilizam frameworks JavaScript como por exemplo Google, Facebook, Microsoft e GitHub.

10.1 Frameworks

- Angular.Js:

É um Framework MV, usado pelo google, para desenvolvimento do front-end de aplicações web, ou seja, que rodam dentro do navegador do cliente. Sua filosofia parte de que uma programação declarativa é muito mais importante que uma programação imperativa quando se trata de desenvolvimento web.

- Electron.Js:

É um framework usado pela GitHub que permite desenvolver aplicações para desktop GUI usando componentes front end e back end originalmente criados para aplicações web. O editor de texto ATOM foi desenvolvido utilizando o Electron.

10.1 Frameworks

- Node.js:

Node.js é um projeto de código aberto projetado para auxiliar na escrita de programas em JavaScript que conversem com redes, sistema de arquivos ou outro I/O, tão utilizado que existem frameworks que usam o Node.js ou seja, Frameworks de Frameworks

- Vue.js:

Vue (pronuncia-se /vju:/, como view, em inglês) é um framework progressivo para a construção de interfaces de usuário, considerado o framework de desenvolvimento front end mais usado em 2016

10.2 Interação com HTML - JQuery

- **jQuery** é uma biblioteca de funções JavaScript que interage com o HTML
- Lançada em dezembro de 2006 no BarCamp, de Nova York, por John Resig.
- Redução de código
- Client-side
- Usada por cerca de 77% dos 10 mil sites mais visitados do mundo, jQuery é a mais popular das bibliotecas JavaScript

10.2 Interação com HTML - JQuery

```
<html>
  <head>
    <title>Js</title>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
  </head>
  <body>
    <h1>João, Maria, José</h1>

    <script type="text/javascript">

      //Captura a string em h1
      var string = $('h1').text();

      //Imprime no console a string
      console.log(string);

      //Separa os nomes usando ',' como separador
      var nomes = string.split(',');

      //Imprime no console os nomes
      console.log(nomes);

      //cria a estrutura de uma lista
      $('body').append('<ul></ul>');

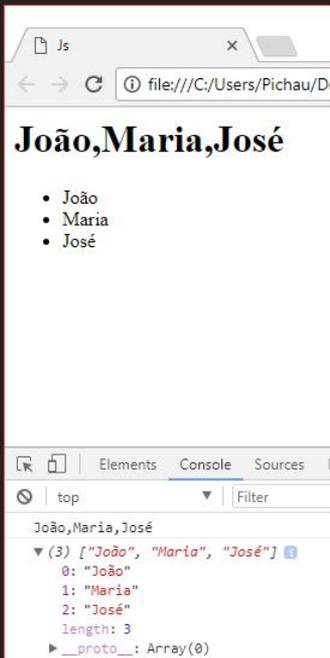
      var printar = '';

      for(i=0; i<nomes.length ; i++){
        var printar = printar + '<li>'+nomes[i]+'</li>';
      };

      //Insere em ul os itens da lista
      $('ul').append($(printar));

    </script>

  </body>
</html>
```



10.2 Interação com HTML - JQuery

- Each

```
<html>
  <head>
    <title>Js</title>
    <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
  </head>
  <body>
    <h1>Resultado</h1>
    <ul>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
    </ul>

    <script type="text/javascript">
      var i=0;
      $( "li" ).each(function() {
        i++;
        this.append(i);
      });
    </script>
  </body>
</html>
```



Resultado

- 1
- 2
- 3
- 4
- 5

- É possível utilizar elementos do html no javascript abrindo um leque para diversas possibilidades

10.2 Interação com HTML - JQuery

```
<html>
<head>
  <title>Js</title>
  <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>

  <style>
    .vermelho{
      background: red;
    }
    .estilo{
      width: 100px;
      margin: 100px;
      padding: 32px 0;
      text-align: center;
      color: white;
    }
    section{
      display: flex;
    }
  </style>

</head>
<body>
<h1>Brincando com classes e ids com Javascript</h1>
  <section>
    <!-- <div class="verde">Verde</div -->
    <div class="vermelho">Vermelho</div>
    <div id="bolinha">Bolinha Vermelha</div>
  </section>
  <script type="text/javascript">

//Adiciona o background azul
$('#vermelho').css('background', 'blue');

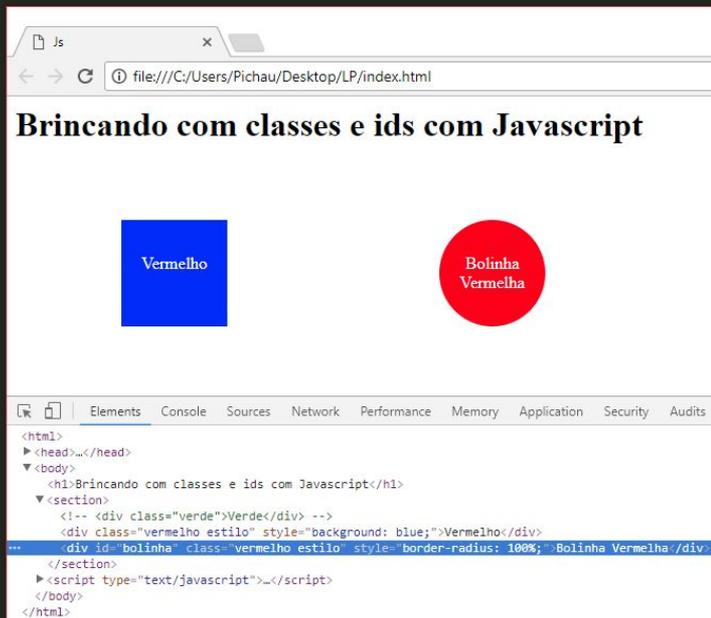
//Adiciona borda arredondada
$('#bolinha').css('border-radius', '100%');

//Adiciona classe no elemento de id bolinha
$('#bolinha').addClass('vermelho');

//Adiciona a classe estilo para todas as divs
$('div').addClass('estilo');

</script>

</body>
</html>
```



11. Avaliação

11.1 Avaliação da linguagem

Crítérios gerais	C	C++	Java	JavaScript
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Não
Aprendizado	Não	Não	Não	Sim
Eficiência	Sim	Sim	Parcial	Não
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Multiparadigma
E evolutibilidade	Não	Parcial	Sim	Sim

11.1 Avaliação da linguagem

Crítérios gerais	C	C++	Java	JavaScript
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Sim
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos Primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	JSON

11.1 Avaliação da linguagem

Crítérios gerais	C	C++	Java	JavaScript
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável, por valor e por cópia de referência
Encapsulamento e proteção	Parcial	Sim	Sim	Não
Sistema de tipos	Não	Parcial	Sim	Não
Verificação de Tipos	Estática	Estática/Dinâmica	Estática/Dinâmica	Dinâmica
Polimorfismo	Coersão e sobrecarga	Todos	Todos	Todos
Exceções	Não	Parcial	Sim	Sim
Concorrência	Não(Biblioteca de funções)	Não(Biblioteca de funções)	Sim	Sim

12. Referências

12 Referências

<http://www.criarweb.com/manual/javascript/>

<https://msdn.microsoft.com/pt-br/library/67defydd%28v=vs.94%29.aspx>

<https://canaltech.com.br/internet/O-que-e-e-como-funciona-a-linguagem-JavaScript/>

<https://www.tecmundo.com.br/software/119217-javascript-linguagens-populares-momento-veja.htm>

<http://br.ccm.net/faq/2680-javascript-introducao-a-linguagem-javascript>

<https://tableless.com.br/modularizacao-em-javascript/>

12 Referências

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objeto

https://www.w3schools.com/js/js_loop_for.asp

https://www.w3schools.com/js/js_loop_while.asp

<http://loopinfinito.com.br/2013/10/22/mamilos-pontos-e-virgulas-em-js/>

<https://felipenmoura.com/articles/escopo-this-e-that/>

<https://www.showmetech.com.br/as-linguagens-de-programacao-mais-usadas-de-2017-ate-julho/>

12 Referências

<https://rodrigorgs.github.io/aulas/mata56/aula15-concorrencia>

<https://imasters.com.br/artigo/21197/javascript/entendendo-arrays-no-javascript/?t=1519021197&source=single>

<https://www.showmetech.com.br/as-linguagens-de-programacao-mais-usadas-de-2017-ate-julho/>

<https://medium.com/weyes/entendendo-o-uso-de-escopo-no-javascript-3669172ca5ba>

<https://pt.stackoverflow.com/questions/1859/como-funcionam-closures-em-javascript>

12 Referências

<http://nodebr.com/entendendo-o-node/>

<https://risingstars2016.js.org/#framework>

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/throw>

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Values,_variables,_and_literals