



Linguagem de Programação para Internet

LUIZ CARLOS FELIX CARVALHO

Regras do Jogo

Primeiro momento (1 a 12/4)

- UNIUBE+ – 5 pontos
- Trabalho – 5 pontos
- Avaliação – 20 pontos

Segundo momento (20 a 29/5)

- UNIUBE+ – 5 pontos
- Trabalho – 5 pontos
- Avaliação – 20 pontos

Terceiro momento (19 a 25/6)

- UNIUBE+ – 5 pontos
- Trabalho – 5 pontos
- Avaliação – 20 pontos
- Simulado – 10 pontos
 - 18/6
- Segunda Chamada / Substitutiva / Recuperação
 - 24 a 28/6

Regras do Jogo

Primeiro momento

- Trabalho HTML – 5 pontos
- 2 Trabalhos avaliativos
 - 10 pontos
 - 10 pontos

Segundo momento

- Trabalho – 10 pontos
- Avaliação – 10 pontos
- 1 Trabalho Avaliativo – 5 pontos

Terceiro momento

- Projeto – 20 pontos
- 1 Trabalho Avaliativo - 5 pontos

JavaScript

REVISÃO

JavaScript

Linguagem de script orientada a objetos

Utilizada para tornar as páginas web interativas

Linguagem de programação que permite implementar funcionalidades mais complexas em páginas web

Linguagem para tratar requisições no lado servidor: Node.js

Portanto:

- Lado Cliente:
 - Objetos para controlar um navegador;
 - Objetos para controlar o DOM (Document Object Model).
 - DOM: interface de programação para a página HTML; representação da página de forma que linguagens possam alterar seu conteúdo.
- Lado Servidor:
 - Objetos relevantes à execução do JavaScript em um servidor

JavaScript

Ordem de execução

- Ordem em que o navegador encontra o código (de cima pra baixo)
- Se o script é carregado no início do documento e já realiza alterações no DOM, tem grandes chances de ter problemas

Linguagem de Script: interpretado x compilado

- JavaScript é uma linguagem interpretada
- O código é executado de cima para baixo e o resultado da execução do código é imediatamente retornado.
- Não há compilação!
- Linguagens compiladas: possuem etapa de compilação.
- Intérpretes JavaScript podem utilizar compilação *just-in-time*
 - Enquanto o script é utilizado, o código fonte é compilado em um binário mais rápido
 - Compilação em tempo de execução: ainda é considerada interpretada!

JavaScript

Código dinâmico x Código estático

- Dinâmico: as informações exibidas pelo sistema são alteradas através de determinadas circunstâncias
 - Ex.: um campo select que depende de uma seleção anterior para ser preenchido. Sequência de campos País, Estado e Cidade.
- Estático: não há alteração na exibição das informações.
 - Ex.: um artigo.
- JavaScript no lado cliente atual na dinamicidade dos elementos do DOM:
 - Preenchimento de campos select;
 - Preenchimento e dimensionamento de tabelas, através de uma consulta no banco de dados.

JavaScript

Adicionando JavaScript no HTML

- Interno

HTML

```
<script>  
  // O JavaScript fica aqui  
</script>
```

- Externo

HTML

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

JS

```
function createParagraph() {  
  let para = document.createElement("p");  
  para.textContent = "Você clicou no botão!";  
  document.body.appendChild(para);  
}
```


JavaScript

- Inline:

▼ JAVASCRIPT

```
1 function criarParagrafo() {  
2   let para = document.createElement("p");  
3   para.textContent = "Você clicou o botao!";  
4   document.body.appendChild(para);  
5 }
```

▼ HTML

```
1 <button onclick="criarParagrafo()">Me clique!</button>  
2
```

- Sem inline:
 - Quanto menos poluir o HTML melhor!

```
const botoes = document.querySelectorAll("button");  
  
for (var i = 0; i < botoes.length; i++) {  
  botoes[i].addEventListener("click", criarParagrafo);  
}
```

JavaScript

Carregamento dos Scripts

- HTML é carregado de cima pra baixo
- Seu código JavaScript pode manipular o DOM
- Se manipular o DOM e se a página não tiver sido carregada -> ERRO!

Possíveis soluções

- Carregar o script no fim da página
 - Mais lento
 - Aguarda todo o carregamento do DOM para carregar os scripts
- Interno: utilizar evento DOMContentLoaded
 - O script só será carregado após o DOM estar pronto
- Externo: utilizar atributo defer
 - Informa ao navegador para carregar o conteúdo HTML: script e HTML são carregados simultaneamente
 - Só funciona em carregamento externo

JavaScript

Exemplo de utilização do evento DOMContentLoaded

JS

```
document.addEventListener("DOMContentLoaded", function() {  
    ...  
});
```

Exemplo de utilização com atributo defer

JS

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

JavaScript

Async x defer

- Dois atributos utilizados no carregamento de scripts

HTML

```
<script async src="js/vendor/jquery.js"></script>  
  
<script async src="js/script2.js"></script>  
  
<script async src="js/script3.js"></script>
```

HTML

```
<script defer src="js/vendor/jquery.js"></script>  
  
<script defer src="js/script2.js"></script>  
  
<script defer src="js/script3.js"></script>
```

JavaScript

Async

- Scripts são baixados sem bloquear a renderização da página
- São executados imediatamente após o script terminar de ser disponibilizado
- Não tem garantia que os scripts carregados irão rodar em uma ordem específica
- Não irão impedir o carregamento do restante da página
- **Utilizar quando os scripts de uma página rodam de forma independente entre si e também não dependem de nenhum outro script.**

JavaScript

Defer

- Scripts irão rodar exatamente na ordem em que aparecem na página
- Serão executados assim que o script e o conteúdo for baixado
- Temos garantia de que um segundo script só será carregado depois que um primeiro for carregado
- Os scripts não irão rodar sem que antes todo o conteúdo da página seja carregado
 - Útil para manipular o DOM

JavaScript

Comentários

- Em linha

```
JS
// Eu sou um comentário
```

- De bloco

```
JS
/*
  Eu também sou
  um comentário
*/
```

JavaScript

Variáveis

- Containers para valores (números, textos etc.)
- Declarando

```
1  
2  var melhorTime;  
3  
4
```

- Inicializando

```
3  
4  melhorTime = "Corinthians";  
5
```

- Declarando e inicializando

```
1  
2  var melhorTime = "Corinthians";  
3
```


JavaScript

Variáveis: var x let

- Var (isso funciona!):

```
melhorTime = "Corinthians Paulista";  
  
var melhorTime = "Corinthians";
```

```
var melhorTime = "Corinthians Paulista";  
  
var melhorTime = "Corinthians";
```

- Com let isso gera erro!



The screenshot shows a code editor with two lines of TypeScript code: `let melhorTime: string = "Corinthians Paulista";` and `let melhorTime = "Corinthians";`. The second line is underlined with a red squiggly line, indicating an error. A tooltip is displayed over the error, showing the text: `Cannot redeclare block-scoped variable 'melhorTime'. ts(2451)`. Below the error message, there are two links: [View Problem](#) and [No quick fixes available](#).

JavaScript

Variáveis: var x let

- let:

```
let melhorTime = "Corinthians Paulista";  
  
melhorTime = "Corinthians";  
|
```

- Prefira o let em relação ao var

Resumindo:

- var: declara variáveis sem fazer verificações e não considera blocos de código;
- let: declara variáveis considerando verificações e considera blocos de código;
- const: semelhante ao let, porém não pode ter alteração do valor (constante);

JavaScript

Nomeação de variáveis:

- Utilize, de forma padrão, somente caracteres latinos (0-9, a-z, A-Z) e o caractere *underline* (_).
- Não use número no início do nome de variáveis: Isso não é permitido e irá causar um erro.
- Não utilize palavras reservadas como nome de variáveis, como, por exemplo, var, function, let e for: gera erro.

Boas práticas:

- Não utilize outros caracteres: podem causar erros ou ser difíceis de entender (internacional).
- Não use *underline* no início do nome de variáveis: isso é utilizado em certos construtores JavaScript para significar coisas específicas, então pode deixar as coisas confusas.
- Uma convenção segura e se ater é a chamada "lower camel case", onde você junta várias palavras, usando minúscula para a primeira palavra inteira e, em seguida, maiusculiza a primeira letra das palavras subsequentes. Ex.: melhorTime, vaiCorinthians, palmeirasNaoTemMundial.
- Faça nomes de variáveis intuitivos, para que descrevam o dado que ela contém. Não use letras ou números únicos, ou frases muito longas.
- As variáveis diferenciam letras maiúsculas e minúsculas — então minhaidade é uma variável diferente de minhaldade. Ou seja, JavaScript é *case sensitive*.

JavaScript

Tipos de variáveis:

- Em JavaScript não se declara os tipos;

Números

```
let numeroDez = 10;
```

Strings

```
let melhorTime = "Corinthians Paulista";
```

Booleans

```
let corinthiansEHOMelhor = true;
```

Arrays

```
let arrayExemplo = [ "Conteudo 1", "Conteudo 2" ];
```

JavaScript

Objetos

- Um objeto é uma estrutura de código que representa um objeto da vida real.

```
let melhorTime = { nome: "Corinthians", pais: "Brasil" };
```

JavaScript é tipada dinamicamente!

```
let numerosDeTitulosDoCorinthians = "158769";  
  
numerosDeTitulosDoCorinthians = 158769;
```

Obs.: utilize o método `typeof()` para verificar o tipo de uma variável

Ex.: `typeof(numerosDeTitulosDoCorinthians);`

JavaScript

Aritmética

- Tudo **Number**: Integer, Float, Double...

Operador	Nome	Propósito	Exemplo
<code>+</code>	Adição	Adiciona um número a outro.	<code>6 + 9</code>
<code>-</code>	Subtração	Subtrai o número da direita do número da esquerda.	<code>20 - 15</code>
<code>*</code>	Multiplicação	Multiplica um número pelo outro.	<code>3 * 7</code>
<code>/</code>	Divisão	Divide o número da esquerda pelo número da direita.	<code>10 / 5</code>
<code>%</code>	Restante (<i>Remainder</i> - as vezes chamado de modulo)	Retorna o resto da divisão em números inteiros do número da esquerda pelo número da direita.	<code>8 % 3</code> (retorna 2; como três cabe duas vezes em 8, deixando 2 como resto.)

JavaScript

Aritmética – Precedência de operadores

JS

```
num2 + num1 / 8 + 2;
```

Qual o resultado para num2 = 16 e num1 = 8?

= 16 + 8 / 8 + 2

= 16 + 1 + 2

= 19

JavaScript

Aritmética

- Incremento: `numeroTeste++`
- Decremento: `numeroTeste--`
- `++x` vs `x++`?
 - Se for uma operação simples (`++x` ou `x++`) não há diferença;
 - Se for utilizado em uma operação:
 - Utilizar o operador antes indica que deseja que o incremento/decremento ocorra antes da operação
 - Utilizar o operador depois indica que deseja que o incremento/decremento ocorra depois da operação

JavaScript

Aritmética

Operadores de Atribuição

Operator	Name	Purpose	Example	Shortcut for
<code>+=</code>	Atribuição de adição	Adiciona o valor à direita ao valor da variável à esquerda e, em seguida, retorna o novo valor da variável	<code>x = 3; x += 4;</code>	<code>x = 3; x = x + 4;</code>
<code>-=</code>	Atribuição de subtração	Subtrai o valor à direita do valor da variável à esquerda e retorna o novo valor da variável	<code>x = 6; x -= 3;</code>	<code>x = 6; x = x - 3;</code>
<code>*=</code>	Atribuição de multiplicação	Multiplica o valor da variável à esquerda pelo valor à direita e retorna o novo valor da variável	<code>x = 2; x *= 3;</code>	<code>x = 2; x = x * 3;</code>
<code>/=</code>	Atribuição de divisão	Divide o valor da variável à esquerda pelo valor à direita e retorna o novo valor da variável	<code>x = 10; x /= 5;</code>	<code>x = 10; x = x / 5;</code>

JavaScript

Aritmética: Operadores de Comparação

- == (igualdade)
- != (não-igualdade)
- > (maior que)
- >= (maior ou igual)
- < (menor)
- <= (menor ou igual)
- === (igualdade estrita)*
- !== (não-igualdade estrita)*

*Observam o tipo de dados

True or false: "1" == 1?

- True

True or false: "1" === 1?

- False

JavaScript

Strings: Textos

JS

```
var string = "The revolution will not be televised.";
```

Aspas simples ou Aspas dupla?

- Qualquer uma!

Cuidado:

- `var texto = "Esse texto está 'ok' ou não?";`
 - Sem erros
- `var texto = "Esse texto está 'ok ou não?";`
 - Sem erros
- `var texto = "Esse texto está "ok' ou não?";`
 - **Erro de sintaxe**
- `var texto = 'Esse texto está "ok" ou não?';`
 - Sem erros

JavaScript

Strings: Textos

- `let x = "Como fazer "funcionar" esse texto?"`

Caracter de escape!

- Caracter utilizado para que o caracter seguinte seja considerado como texto.
- `\` - Em JavaScript é a barra invertida
- `let x = "Como fazer \"funcionar\" esse texto?"`

String é um objeto

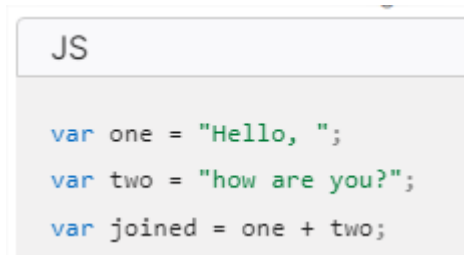
- Possui métodos
- Ex.: `minhaString.length` // tamanho de uma string

JavaScript

Strings: Textos

Concatenação de strings

- Operador +

A screenshot of a code editor with a tab labeled 'JS'. The code inside the editor demonstrates string concatenation using the '+' operator. It defines two variables, 'one' and 'two', and then concatenates them into a new variable 'joined'.

```
JS  
  
var one = "Hello, ";  
var two = "how are you?";  
var joined = one + two;
```

Números + Strings

- let x = "Quantidade de mundiais do Palmeiras: " + 0;
- Resultado?
 - String concatenada com o número
 - "Quantidade de mundiais do Palmeiras: 0"

JavaScript

Strings: Textos

Números + Strings (cont.)

- `var meuNumero = "19" + "67";`
- `typeof meuNumero; // String`

- `var meuNumero = "123";`
- `var meuNumero2 = Number(meuNumero);`
- `typeof meuNumero2; // Number`

- `var meuNumero = 123;`
- `var meuTexto = meuNumero.toString();`
- `typeof meuTexto; // String`

JavaScript

Strings: métodos mais utilizados

- Tamanho da string: `minhaString.length();`
- Caracter específico:
 - `minhaString[0];`
 - `minhaString.charAt(0);`
- Endereço de uma substring:
 - `let minhaString = "Vai Corinthians";`
 - `minhaString.indexOf("Corinthians"); // 4`
 - `minhaString.indexOf("Palmeiras"); // -1`
- Substring
 - `minhaString.slice(4); // Corinthians`
 - `minhaString.slice(2); // i Corinthians`
 - `minhaString.slice(4, 8); // Cori`
 - `minhaString.slice(4, 70); // Corinthians`

JavaScript

Strings: métodos mais utilizados

- Alterar “caixa” das letras: toUpperCase e toLowerCase
- Trocar pedaço de uma string
 - `var minhaString = “Vai Corinthians...”`
 - `minhaString.replace(“...”, “!!!”);`

CUIDADO: Todo método em string te retorna o resultado, NÃO altera o objeto string!!!!!!!!!!!!!!

JavaScript

Strings: métodos mais utilizados

- `var minhaString = "Vai Corinthians..."`
- `minhaString.replace("...", "!!!");`
- Qual o conteúdo de `minhaString`???
- Res.: `"Vai Corinthians..."`

CUIDADO: Todo método em string te retorna o resultado, NÃO altera o objeto string!!!!!!!!!!!!

JavaScript

Strings: métodos mais utilizados

- `var minhaString = "Vai Corinthians..."`
- `Let minhaString2 = minhaString.replace("...", "!!!");`
- Qual o conteúdo de `minhaString2`???
- Resp.: `"Vai Corinthians!!!"`

CUIDADO: Todo método em string te retorna o resultado, NÃO altera o objeto string!!!!!!!!!!!!

JavaScript - Funções

Bloco de código delimitado que é executado quando invocado.

Ideal para reutilização de código.

Declaração de uma função:

- consiste no uso da palavra-chave **function** (função)
- seguida pelo:
 - 1) nome da função,
 - 2) lista de argumentos entre parênteses e separados por vírgula e
 - 3) pelo código JavaScript que define a função, entre chaves { }.

Exemplo:

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

JavaScript - Funções

Observações:

- 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.
- Se você passar um objeto (ou seja, um valor não primitivo, tal como vetor (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.

```
function campeaoMundial( time ) {  
    time.titulosMundiais++;  
}  
  
var corinthians = { nome: "Corinthians", titulosMundiais: 1 };  
  
var mundiaisAntes2012 = corinthians.titulosMundiais;  
  
//2012  
campeaoMundial( corinthians );  
  
var mundiais2013 = corinthians.titulosMundiais;  
  
console.info( mundiaisAntes2012 ); // 1  
console.info( mundiais2013 ); // 2
```

JavaScript - Funções

Declaração por meio de expressão:

- Não utilizando nomeação:

```
var square = function (numero) {  
    return numero * numero;  
};  
var x = square(4); //x recebe o valor 16
```

- Utilizando nomeação:

```
var fatorial = function fac(n) {  
    return n < 2 ? 1 : n * fac(n - 1);  
};  
  
console.log(fatorial(3));
```

JavaScript - Funções

Invocando:

```
var a, b, c, d, e;  
a = fatorial(1); // a recebe o valor 1  
b = fatorial(2); // b recebe o valor 2  
c = fatorial(3); // c recebe o valor 6  
d = fatorial(4); // d recebe o valor 24  
e = fatorial(5); // e recebe o valor 120
```

Obs.: em JavaScript pode-se definir função dentro de blocos de código.

JavaScript - Funções

Escopo:

- Variáveis definidas dentro de uma função não podem ser acessadas externamente;
- O código dentro de uma função pode acessar as variáveis de fora da função;

JavaScript - Funções

Vamos pensar um pouco...

Qual o retorno para:

- `getScore();`
 - 5
- `getScore2();`
 - 53
- `getScore2(); //segunda chamada`
 - 83

```
88
89 let num1 = 20,
90     num2 = 3,
91     nome = "José";
92
93 // Esta função é definida no escopo global
94 function multiplica() {
95     return num1 * num2;
96 }
97
98 multiplica(); // Retorna 60
99
100 // Um exemplo de função aninhada
101 function getScore() {
102     let num1 = 2,
103         num2 = 3;
104
105     function add() {
106         return nome + " marcou " + (num1 + num2);
107     }
108
109     return add();
110 }
```

```
// Um exemplo de função aninhada
function getScore2() {

    num2 += 30;

    function add() {
        return nome + " marcou " + (num1 + num2);
    }

    return add();
}
```


JavaScript – HTML + JavaScript

Primeiro passo com manipulação do DOM

- **element.addEventListener(evento, comportamento);**
 - Adiciona a função de nome comportamento no evento especificado;
 - Ex.: select.addEventListener("change", function1);
- **document.getElementById(id)**
 - Obtém o elemento do DOM de id especificado

```
<label for="selectTime">Quantos mundiais tem o: </label>
<select id="selectTime">
  <option value="">--Selecione--</option>
  <option value="corinthians">Corinthians</option>
  <option value="palmeiras">Palmeiras</option>
</select>

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

```
<script>
  let select = document.getElementById( "selectTime" );
  let paragrafo = document.getElementById( "paragrafo" );

  select.addEventListener( "change", changeValue );

  function changeValue() {

    var time = select.value;

    if ( time === "corinthians" ) {

      paragrafo.textContent =
        "O Corinthians tem DOIS mundiais!!!";
    }
    else if ( time === "palmeiras" ) {

      paragrafo.textContent =
        "O Palmeiras NÃO tem mundial!!!";
    }
  }
</script>
```

JavaScript – HTML + JavaScript

Exemplo – visualização:

Quantos mundiais tem o:

Quantos mundiais tem o:

O Corinthians tem DOIS mundiais!!!

Quantos mundiais tem o:

O Palmeiras NÃO tem mundial!!!

JavaScript – HTML + JavaScript

Eventos

- São ações ou ocorrências que acontecem no sistema;
- Um tipo de sinal é disparado quando o evento ocorre;
- Promover a execução de algum método quando na ocorrência de tal evento.

Eventos são disparados na janela do navegador

Tende a estarem anexados a algum item específico nele

- Um único elemento
- Um conjunto de elementos
- HTML carregado na guia atual
- Toda a janela do navegador.

Exemplos:

- Clique do mouse
- Pressionar uma tecla
- Usuário fechando navegador
- Página finalizando carregamento

JavaScript – HTML + JavaScript

Cada evento possui um manipulador de evento (event handler)

- Bloco de código a ser executado quando o evento for disparado
- Registrar um event handler (event listeners, ouvintes)

Exemplo:

```
<button>Change color</button>
```

```
var btn = document.querySelector("button");

function random(number) {
    return Math.floor(Math.random() * (number + 1));
}

btn.onclick = function () {
    var rndCol =
        "rgb(" + random(255) + "," + random(255) + "," + random(255) + ")";
    document.body.style.backgroundColor = rndCol;
};
```

JavaScript – HTML + JavaScript

Manipulando eventos:

- `Element.onClick`: evento disparado no clique do elemento
- `Element.onFocus`: evento disparado quando o elemento ganha o foco
- `Element.onBlur`: evento disparado quando o elemento perde o foco
- `window.onkeypress`, `window.onkeydown`, `window.onkeyup`: eventos disparados na digitação das teclas do teclado
- `Element.onmouseover`: evento disparado quando o mouse passa pelo elemento
- `Element.onmouseout`: evento disparado quando o mouse sai da área do elemento

JavaScript – HTML + JavaScript

Manipulando eventos – inline:

```
<label for="selectTime2">Quantos mundiais tem o: </label>
<select id="selectTime2" onchange="changeValue2()">
  <option value="">--Selecione--</option>
  <option value="corinthians">Corinthians</option>
  <option value="palmeiras">Palmeiras</option>
</select>

<p id="paragrafo2"></p>
```

```
function changeValue2() {
  let select = document.getElementById( "selectTime2" );
  let paragrafo = document.getElementById( "paragrafo2" );

  var time = select.value;

  if ( time === "corinthians" ) {

    paragrafo.textContent =
      "O Corinthians tem DOIS mundiais!!!";
  }
  else if ( time === "palmeiras" ) {

    paragrafo.textContent =
      "O Palmeiras NÃO tem mundial!!!";
  }
}
```

JavaScript – HTML + JavaScript

Métodos utilizados para manipulação do DOM:

`document.createElement("<tag_elemento_HTML>")`

- Ex.: `document.createElement("a");`

`element.appendChild(element2);`

- Ex.: `elementTable.appendChild(elementRow);`

`document.createTextNode("Texto");`

- Ex.: `document.createTextNode("Cliente");`

`element.setAttribute('atributo', 'valor do atributo');`

- Ex.: `button.setAttribute('onClick', metodoDeComportamento);`

JavaScript – HTML + JavaScript

Exemplo de manipulação do DOM:

```
var tr = document.createElement("tr");
var col = document.createElement("td");

col.appendChild( document.createTextNode( "Texto da coluna" ) );

tr.appendChild( col );

document.getElementById( "tabela" ).appendChild( tr );
```

O que está acontecendo no exemplo?

JavaScript – Objetos

Objeto Literal

```
var pessoa = {};
```

```
pessoa.idade;  
pessoa.interesse[1];  
pessoa.bio();
```

```
var pessoa = {  
  nome: ["Bob", "Smith"],  
  idade: 32,  
  sexo: "masculino",  
  interesses: ["música", "esquiar"],  
  bio: function () {  
    alert(  
      this.nome[0] +  
        " " +  
      this.nome[1] +  
        " tem " +  
      this.idade +  
        " anos de idade. Ele gosta de " +  
      this.interesses[0] +  
        " e " +  
      this.interesses[1] +  
        " .",  
    );  
  },  
  saudacao: function () {  
    alert("Oi! Eu sou " + this.nome[0] + " .");  
  },  
};
```

JavaScript – Objetos

Objeto Literal

```
pessoa.nome.primeiro;  
pessoa.nome.ultimo;
```

```
var pessoa = {  
  nome : {  
    primeiro: 'Bob',  
    ultimo: 'Smith'  
  },  
  bio: function () {  
    alert(  
      this.nome[0] +  
        " " +  
      this.nome[1] +  
        " tem " +  
      this.idade +  
        " anos de idade. Ele gosta de " +  
      this.interesses[0] +  
        " e " +  
      this.interesses[1] +  
        " .",  
    );  
  },  
  saudacao: function () {  
    alert("Oi! Eu sou " + this.nome[0] + " .");  
  },  
};
```

JavaScript – Objetos

Objeto Literal

```
pessoa["nome"]["primeiro"];
```

```
var pessoa = {  
  nome : {  
    primeiro: 'Bob',  
    ultimo: 'Smith'  
  },  
  bio: function () {  
    alert(  
      this.nome[0] +  
        " " +  
      this.nome[1] +  
        " tem " +  
      this.idade +  
        " anos de idade. Ele gosta de " +  
      this.interesses[0] +  
        " e " +  
      this.interesses[1] +  
        " .",  
    );  
  },  
  saudacao: function () {  
    alert("Oi! Eu sou " + this.nome[0] + " .");  
  },  
};
```

JavaScript – Objetos

Objeto Literal

```
pessoa["idade"];
```

```
pessoa.idade = 45;  
pessoa["nome"]["ultimo"] = "Cratchit";
```

```
var pessoa = {  
  nome: ["Bob", "Smith"],  
  idade: 32,  
  sexo: "masculino",  
  interesses: ["música", "esquiar"],  
  bio: function () {  
    alert(  
      this.nome[0] +  
        " " +  
      this.nome[1] +  
        " tem " +  
      this.idade +  
        " anos de idade. Ele gosta de " +  
      this.interesses[0] +  
        " e " +  
      this.interesses[1] +  
        " .",  
    );  
  },  
  saudacao: function () {  
    alert("Oi! Eu sou " + this.nome[0] + " .");  
  },  
};
```

JavaScript – Objetos

Exercício:

- Através do console de um navegador e observando o exemplo anterior, crie um objeto com atributos e funções. Depois, invoque suas funções.

JavaScript – Objetos

Classes e Objetos

```
const giles = new Person("Giles");

giles.introduceSelf(); // Hi! I'm Giles
```

```
class Person {
  name;

  constructor(name) {
    this.name = name;
  }

  introduceSelf() {
    console.log(`Hi! I'm ${this.name}`);
  }
}
```

JavaScript – Objetos

Classes e Objetos

```
class Professor extends Person {
  teaches;

  constructor(name, teaches) {
    super(name);
    this.teaches = teaches;
  }

  introduceSelf() {
    console.log(
      `My name is ${this.name}, and I will be your ${this.teaches} professor.`
    );
  }

  grade(paper) {
    const grade = Math.floor(Math.random() * (5 - 1) + 1);
    console.log(grade);
  }
}
```

JavaScript – Objetos

Classes e Objetos

```
class Student extends Person {  
  #year;  
  
  constructor(name, year) {  
    super(name);  
    this.#year = year;  
  }  
  
  introduceSelf() {  
    console.log(`Hi! I'm ${this.name}, and I'm in year ${this.#year}.`);  
  }  
  
  canStudyArchery() {  
    return this.#year > 1;  
  }  
}
```


JavaScript – Objetos

Classes e Objetos

```
class Example {  
  somePublicMethod() {  
    this.#somePrivateMethod();  
  }  
  
  #somePrivateMethod() {  
    console.log("You called me?");  
  }  
}  
  
const myExample = new Example();  
  
myExample.somePublicMethod(); // 'You called me?'  
  
myExample.#somePrivateMethod(); // SyntaxError
```