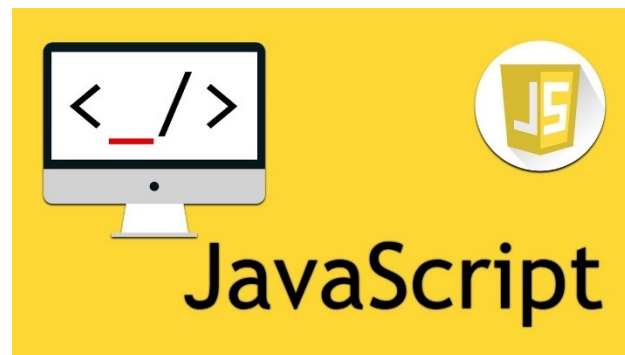




Programação Web





Introdução

JavaScript (JS)



- JavaScript é uma linguagem de programação para criar *scripts* (linguagem interpretada) multiplataforma – entendida em diferentes sistemas operativos e por diferentes plataformas de execução.
- Pode ser executada numa aplicação cliente, num web browser, numa aplicação servidor, num servidor web.
- Tal como a maioria das linguagens de programação, é detentora de todo um conjunto de elementos, como: declarações, operadores, estruturas de controlo e uma biblioteca padrão de objetos, como [Array](#), [Date](#), [Math](#), etc.

JS - Processamento



- Linguagem Interpretada
 - É processada aos blocos e interpretada à medida que é necessário converter as diversas estruturas para uma representação capaz de ser executada.
 - A **vantagem** é que a linguagem vai sendo processada a pedido (*on demand*). O código escrito cuja execução não é pedida, não consome tempo de processamento.
 - A principal **desvantagem** desta abordagem é que eventuais erros no código só são detetados quando este é executado, o que pode gerar erros inesperados de execução.

JS - Utilidade

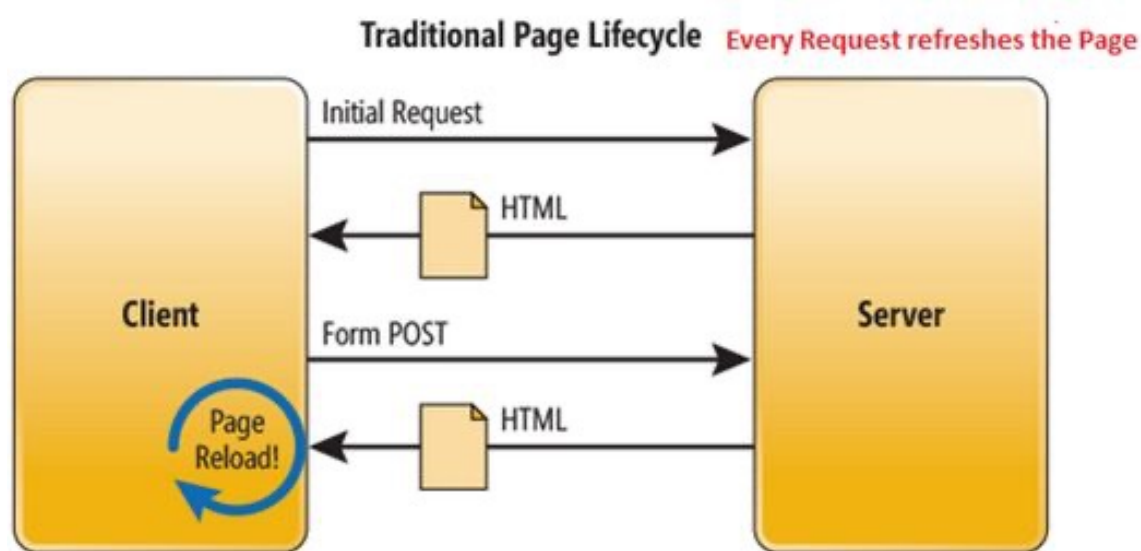


- Concebida originalmente para ser executada no web browser, no cliente, para fornecer conteúdos mais dinâmicos e um maior nível de interação com o utilizador, sem necessidade de obrigatoriamente se recorrer ao servidor.
- Pelo que um programa (*script*) em JavaScript permite:
 - controlar o web browser;
 - alterar o conteúdo do documento exibido, de modo dinâmico;
 - e ainda realizar comunicações síncronas ou assíncronas, com o servidor.

JS - Web Tradicional



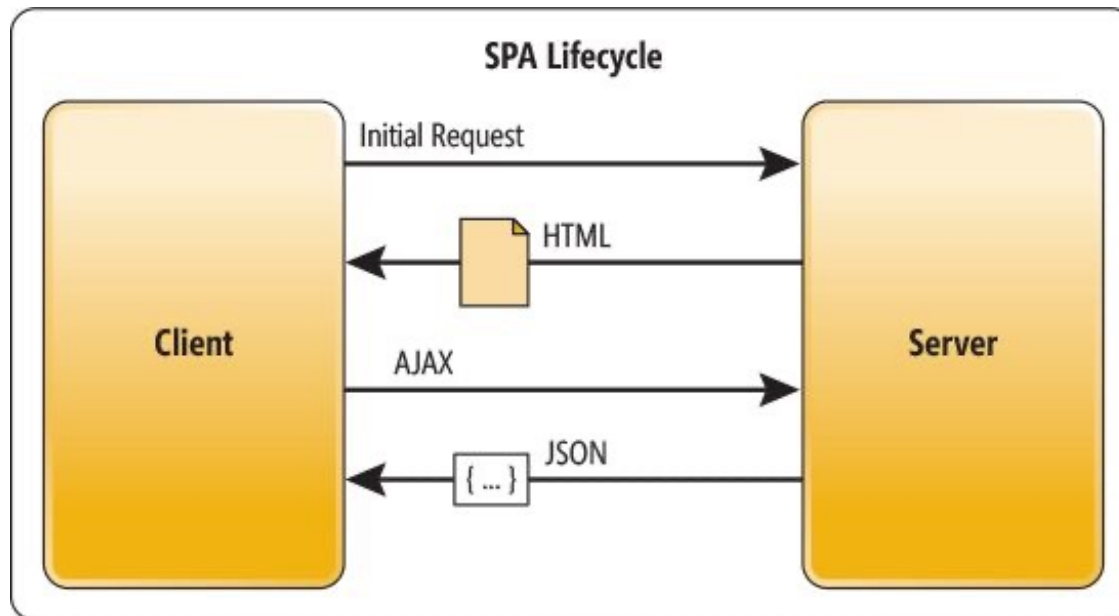
- Ciclo de Vida uma página web tradicional, sem recurso a JavaScript.



JS - Web Dinâmica



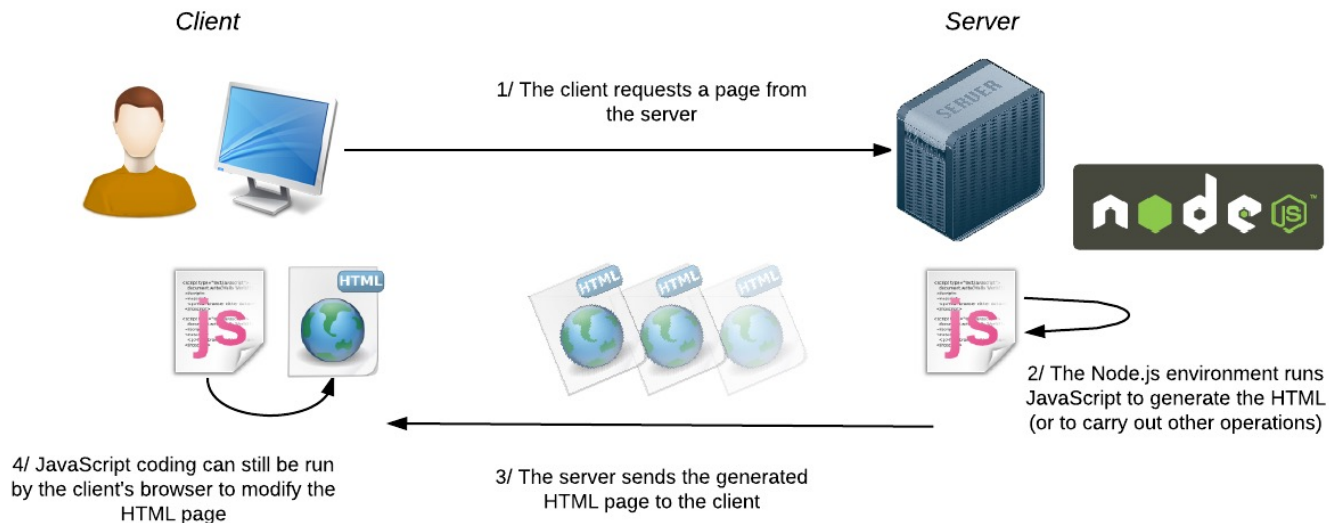
- Ciclo de Vida de uma página web, mais dinâmica, com recurso a JavaScript.



JS – Uso Alargado



- Devido à enorme versatilidade da linguagem JavaScript, esta passou a ser usada também do lado do servidor para gerar conteúdos a enviar ao cliente.
 - Exemplo de uso na plataforma Node.js





JS - JavaScript

Inclusão

Entrada e Saída de Dados

JS – Inclusão



- A inclusão de um programa em JavaScript numa página web pode ser feita de duas formas:
 - na própria página, dentro de um elemento `<script>`, que pode aparecer
 - no cabeçalho da página, elemento `<head>`
 - no corpo da página, elemento `<body>`
 - em ambos, cabeçalho e corpo
 - ou num ficheiro externo, normalmente com extensão “.js”, que será importado para a página web também através do elemento `<script>`

JS – Inclusão



- Exemplo de inclusão no cabeçalho da página:

```
<> index.htm > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>My Page</title>
5      <meta charset="utf-8">
6      <script>
7          // Comentário de uma linha: Programa em Javascript
8          /*
9              Comentário multi-linha:
10             Programa no cabeçalho da página web.
11             */
12      </script>
13  </head>
14  <body>
15
16      <!-- Conteúdo da página web -->
17
18  </body>
19  </html>
```

JS – Inclusão



- Exemplo de inclusão no corpo da página:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>My Page</title>
5      <meta charset="utf-8">
6  </head>
7  <body>
8
9      <!-- Conteúdo da página web -->
10
11      <script>
12          // Comentário de uma linha: Programa em Javascript
13          /*
14              Comentário multi-linha:
15              Programa no cabeçalho da página web.
16          */
17      </script>
18  </body>
19  </html>
20
```

JS – Inclusão



- Exemplo de inclusão num ficheiro externo e sua importação:

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>My Page</title>
5      <meta charset="utf-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7      <!-- Inclusão do CSS do Bootstrap -->
8      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.2/dist/css/bootstrap.min.css" rel="stylesheet">
9      <!-- Inclusão do programa Javascript do Bootstrap -->
10     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.2/dist/js/bootstrap.bundle.min.js"></script>
11 </head>
12 <body>
13     <!-- Conteúdo da página web -->
14     <h1>My Page</h1>
15
16     <!-- Inclusão do programa Javascript pessoal -->
17     <script src="prog.js"></script>
18 </body>
19 </html>
```

JS – Inclusão



- O uso de ficheiros externos para incluir os programas Javascript possui algumas vantagens:
 - separa o código de programação (parte comportamental) da parte de HTML (parte de estrutura e conteúdo), tornando-as partes independentes;
 - é mais fácil ler e manter as duas partes;
 - os ficheiros JavaScript, sendo externos e independentes dos ficheiros HTML, permanecem mais tempo na cache dos browsers, o que acelera o carregamento repetido das mesmas páginas.

JS – Entrada e Saída de Dados



- Um programa JavaScript pode ler e escrever dados de diferentes formas, dependendo do objetivo.
- Caso o objetivo seja para testes e depuração do programa:
- Saída de Dados:
 - na consola do browser: `console.log("Hello World!!!")`
 - numa janela de alerta: `alert("Hello World!!!")`
 - na própria página web: `window.write("Hello World!!!")`
- Entrada de Dados:
 - numa janela de input: `prompt("Insira o nome: ")`
 - caso seja necessário fazer a leitura de um número, é necessário proceder à conversão do texto de entrada para número:
 - `parseInt(prompt("Insira o número real: "))`
 - `parseFloat(prompt("Insira o número inteiro: "))`

JS – Entrada e Saída de Dados



- Caso o objetivo seja a interação efetiva com a página web, de uma forma integrada e harmoniosa com o seus conteúdos:

- Saída de Dados:

- num elemento da página web

```
document.getElementById("demo").innerHTML = "Hello World!!!"
```

- Entrada de Dados:

- A partir de um elemento <input>

```
var name = document.getElementById("name").value
```




JS - JavaScript

Programação

JS – Sintaxe



- A sintaxe consiste num conjunto de regras que determinam como os elementos devem ser escritos e combinados por forma a que as instruções possam ser lidas, entendidas e executadas.
- Embora não obrigatória, mas considerada como muito boa prática, a primeira regra que preside à escrita das instruções é terminar a escrita de cada linha com um ponto e vírgula (;).
- Outra regra importante, é que o JavaScript distingue entre letras maiúsculas e minúsculas, ou seja, é *case-sensitive*.
 - Exemplo: nome ≠ Nome ≠ NOME

JS – Variáveis



- As variáveis são usadas para armazenar valores de dados.
- O JS usa as palavras **let**, **var** e **const** para declarar variáveis. (Sintaxe: palavra_chave variavel;)
- Depois da declaração é possível atribuir valores às variáveis. (Sintaxe: variavel = valor;)
- Exemplos:
 - `var x; // variável global no documento`
 - `x = 3;`
 - `let y; // variável num bloco de código`
 - `y = 5;`
 - `const z = 10; // constante, a variável z não pode mudar de valor`

JS – Operadores



- O JS utiliza operadores aritméticos para proceder ao cálculo de valores.
- Operadores elementares: + - * /
- Outros:
 - ** (exponenciação)
 - % (módulo – resto da divisão inteira)
 - ++ e -- (incremento ou decremento do valor de uma variável)
- São usadas as regras matemáticas de precedência entre os operadores para efetuar o cálculo.
- Exemplos:
 - $2 + 2 \rightarrow 4$
 - $2 + 2 * 3 \rightarrow 8$ (1º é feita a multiplicação e depois a soma)
 - $(2 + 2) * 3 ** 2 \rightarrow 36$ (1º é feita a exponenciação, depois a soma e só depois a multiplicação)

JS – Tipos de Dados



- Como noutras linguagens de programação, o valores no JavaScript também possuem um tipo associado.
- Tipos Primitivos:
 - number e bigint (números)
 - boolean (tipo lógico, pode assumir os valores `true` e `false`)
 - string (texto)
 - null (assume o valor `null` – algo como nulo, mas diferente de zero)
 - undefined (assume o valor `undefined`, quando ainda não foi feita qualquer atribuição de valor a uma variável)
 - symbol (uma chave que identifica univocamente um objeto)
- Tipo Objeto
 - O tipo Objeto é um tipo complexo e em JavaScript, um objeto é visto como um conjunto de propriedades.

JS – Execução Condicional



- Palavras reservadas para construção de instruções condicionais:
 - **if** – para especificar um bloco de código a executar se determinada condição é verdadeira
 - **else** – para especificar um bloco de código a executar se a condição anterior é falsa
 - **else if** – para especificar uma nova condição a testar, caso a primeira seja falsa
 - **switch** – para especificar diversos blocos alternativos de código a executar, mediante determinadas condições

JS – Sintaxe da Instrução if



- Sintaxe da instrução if - else if - else

```
if ( expressão lógica ) // SE o resultado da expressão lógica for true
{
    // Executa este bloco de instruções
}
else if ( outra expressão lógica ) // SENÃO SE
{
    // Executa este bloco alternativo de instruções
}
else // SENÃO
{
    // Executa este bloco alternativo de instruções
}
```

JS – Sintaxe da Instrução switch



```
switch ( valor ou expressão )
{
  case valor1:
    instruções ...
    break;
  case valor1:
    instruções ...
    break;
  ...
  default:
    instruções ...
}
```

- O **switch** calcula a expressão e/ou toma o valor dado para verificar se alguns dos casos (**case**) possui esse valor.
- Caso encontre o valor num **case**, executa o bloco de instruções desse caso.
- De contrário, executa o bloco de instruções por defeito (**default**).

JS – Execução Cíclica



- Instruções (palavras reservadas) que permitem a execução cíclica de blocos de instruções:
 - **for** – executa um número determinado de vezes
 - **for in** – executa através de um conjunto de propriedades de um objeto
 - **for of** – executa através de um conjunto de elementos numa estrutura de dados
 - **while** – executa enquanto uma condição for verdadeira
 - **do while** – executa até que uma condição seja falsa

JS – Sintaxe da Instrução for



```
for (pré-execução; condição; pós-execução )  
{  
    // Executa este bloco de instruções  
}
```

- A expressão “pré-execução” é executada 1 vez antes de se iniciar o ciclo
- Enquanto a expressão “condição” for verdadeira, o ciclo continua a executar
- A expressão “pós-execução” é sempre executada no fim de cada iteração do ciclo

JS – Sintaxe da Instrução while



```
while ( condição )  
{  
    // Executa este bloco de instruções  
}
```

```
do  
{  
    // Executa este bloco de instruções  
}  
while ( condição )
```

- Enquanto a expressão “**condição**” for verdadeira, o ciclo continua a executar

JS – Funções



- Sintaxe de uma função

```
function nome_funcao(param1, param2, param3, ...)  
{  
    // instrução 1  
    // instrução 2  
    ...  
    return exp;  
}
```

- Utiliza a palavra reservada **function** para indicar que o bloco de instruções seguinte é uma função.
- De seguida é dada um nome à função.
- Entre parênteses é indicado o conjunto de parâmetros, por onde podem ser passados os dados de entrada, chamados de argumentos. (**Opcional**)
- No interior são escritas instruções, como num programa.
- No final, a palavra reservada **return** permite devolver o resultado da função. (**Opcional**)

JS – Exemplo de Funções



- Programa, com 2 funções, que pede valores inteiros positivos, faz a sua soma e escreve o resultado.

```
js aula04.js > ...
1
2  function leNumero()
3  {
4      let x;
5      do
6      {
7          x = parseInt(prompt("Insira um numero inteiro positivo: "));
8      }
9      while (x <= 0)
10     return x;
11 }
12
13 function soma(x, y)
14 {
15     let z = x + y;
16     return z;
17 }
18
19 // Variáveis de entrada de dados
20 let a, total;
21
22 // Processamento dos dados
23 total = 0;
24 do
25 {
26     a = leNumero();
27     total = soma(total, a);
28 }
29 while(a != 100)
30
31 // Escrita de resultados
32 document.write("Total dos valores inseridos: " + total);
```



JS - JavaScript

DOM – Document Object Model


JS – DOM



- Quando o JavaScript é executado num browser consegue aceder a qualquer elemento da página HTML, assim como a propriedades e funcionalidades da janela do browser.
- Isto permite alterar o conteúdo da página, como os seus elementos e estilos, sem que seja necessário voltar a carregar a página a partir do servidor.
- O que possibilita isso é um modelo chamado DOM (*Document Object Model*), ou seja, modelo de objetos do documento.
- Este modelo pode ser aplicado a qualquer documento que possa seja estruturado na forma de nós.

JS – HTML - Nós



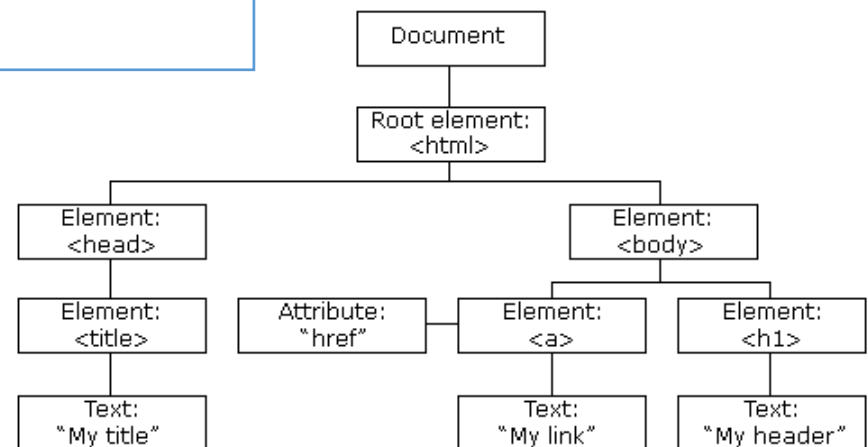
- Num documento HTML, tudo são nós: 
 - O documento em si é um nó do tipo `document`;
 - Todos os elementos HTML são nós do tipo `element`;
 - Todos os atributos HTML são nós de tipo `attribute`;
 - O texto dentro dos elementos HTML é um nó do tipo `text`;
 - Comentários são nós do tipo `comment`;

JS – HTML - Nós



- Exemplo:

```
<!doctype html>
<html lang="en">
  <head>
    <title>My title</title>
  </head>
  <body>
    <h1>My header</h1>
    <a href="http://www.ua.pt">My link</a>
  </body>
</html>
```



JS – HTML DOM



- O HTML DOM é um modelo standard de **objetos** do documento HTML e preconiza uma interface programática para interagir com esses objetos.
- Este define:
 - que todos os **nós** do HTML são considerados **objetos**;
 - as **propriedades** de todos os objetos HTML
 - valores dos objetos que podem ser lidos ou modificados
 - os **métodos** para aceder a todos os objetos HTML
 - ações que podem ser realizadas sobre os objetos
 - os **eventos** para todos os objetos HTML
 - acontecimento que ocorreu com o objeto

JS – HTML DOM



- Acesso ao conteúdo de um elemento HTML

```
<html>
<body>
  <p id="mypara"></p>
  <script>
    document.getElementById("mypara").innerHTML = "Olá. Este é o meu parágrafo!!!";
  </script>
</body>
</html>
```

- getElementById
 - **método** que procura por um elemento (objeto) que possui como id o valor “mypara”
- innerHTML
 - **propriedade** que permite aceder e alterar o conteúdo HTML de um elemento (objeto)

JS – HTML DOM



- Métodos para Pesquisa de Elementos

Method	Description
<code>document.getElementById(<i>id</i>)</code>	Find an element by element id
<code>document.getElementsByTagName(<i>name</i>)</code>	Find elements by tag name
<code>document.getElementsByClassName(<i>name</i>)</code>	Find elements by class name

- Propriedades para Acesso e Alteração de Elementos

Property	Description
<code>element.innerHTML = <i>new html content</i></code>	Change the inner HTML of an element
<code>element.attribute = <i>new value</i></code>	Change the attribute value of an HTML element
<code>element.style.property = <i>new style</i></code>	Change the style of an HTML element

JS – HTML DOM - Exemplo



- Aplicação Calculadora

Calculadora

<input type="text" value="5"/>	<input type="text" value="+"/>	<input type="text" value="3"/>	<input type="text" value="→"/>	<input type="text" value="8"/>
--------------------------------	--------------------------------	--------------------------------	--------------------------------	--------------------------------

Calcular

JS – HTML DOM - Exemplo



- Aplicação Calculadora (HTML)

Prática

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8" />
5   <title>Calculadora</title>
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" />
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" rel="stylesheet">
9   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
10 </head>
11 <body>
12   <div class="container pt-4">
13     <h3 class="border-bottom p-2"><i class="bi bi-calculator"></i> Calculadora</h3>
14
15     <div class="row border-bottom p-4">
16       <div class="col-md-2"><input id="op1" type="text" class="form-control" style="width:150px; margin:auto;"></div>
17       <div class="col-md-2"><input id="oper" type="text" value="+" class="form-control" style="width:150px; margin:auto; text-align:center;"></div>
18       <div class="col-md-2"><input id="op2" type="text" class="form-control" style="width:150px; margin:auto;"></div>
19       <div class="col-md-1" style="text-align:center"><i class="bi bi-box-arrow-right" style="font-size: 25px;"></i></div>
20       <div class="col-md-2"><input id="res" type="text" class="form-control" style="width:150px; margin:auto;"></div>
21     </div>
22
23     <div class="row p-4">
24       <div class="col-md-2" style="display: flex; justify-content: center;">
25         <button class="btn btn-primary" onclick="calcular()">Calcular</button>
26       </div>
27     </div>
28
29   </div>
30
31   <!-- My JavaScript -->
32   <script src="aula03.js"></script>
33 </body>
34 </html>
```

JS – HTML DOM - Exemplo



- Aplicação Calculadora (JS)

Prática

```
1  function calcular()  
2  {  
3      let res;  
4      let op1 = parseInt(document.getElementById("op1").value);  
5      let op2 = parseInt(document.getElementById("op2").value);  
6      let oper = document.getElementById("oper").value;  
7      if (oper == '+')  
8      |   res = op1 + op2;  
9      else  
10     |   res = undefined;  
11     document.getElementById("res").value = res;  
12 }
```