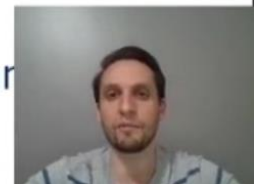


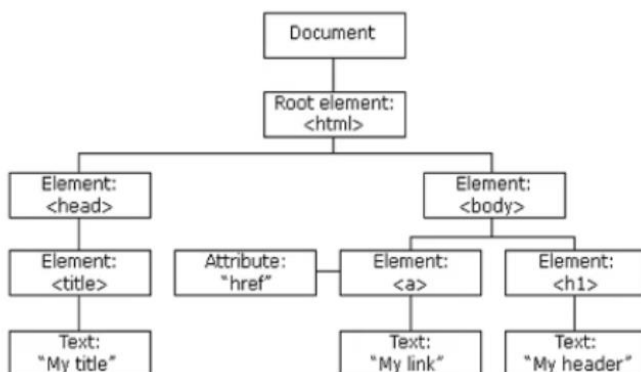
JavaScript

- Linguagem de script e multiplataforma
- Client Side – É executado do lado do cliente (usuário)
- Tem capacidade de interagir com elementos de uma página HTML
- Muito usado no desenvolvimento de páginas e também aplicativos mobile híbridos

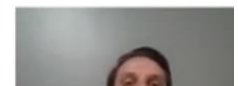


DOM

Document Object Model



- JavaScript pode alterar todos elementos do HTML
- JavaScript pode alterar todos os atributos e estilos de CSS de uma página



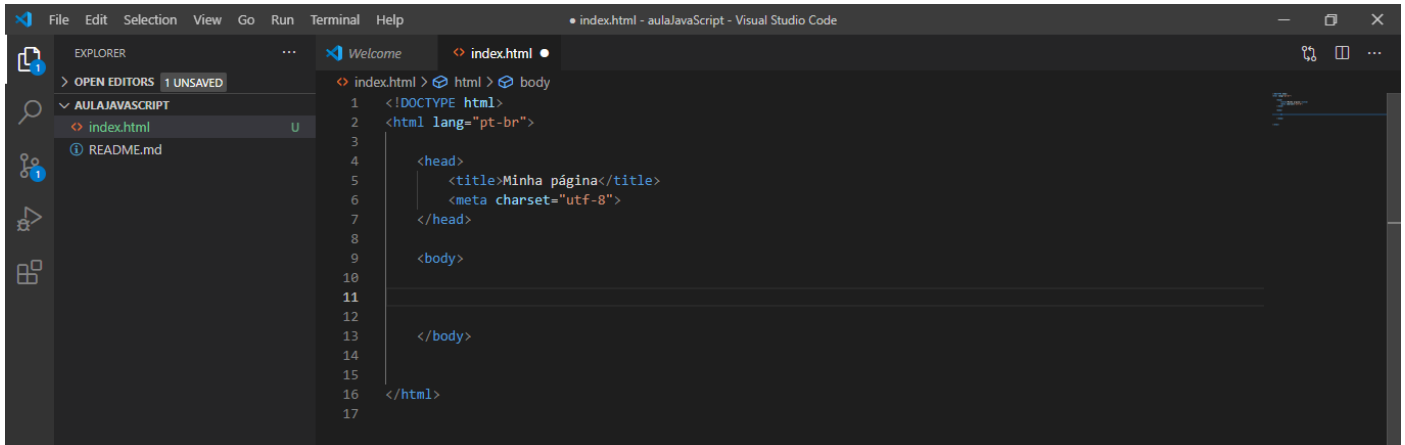
Para iniciar a aula foi criado um repositório no github.

<https://github.com/DiogoBarrosx/aulaJavaScript.git>

E criado um repositório local no meu workspace através de um git clone no git bash com o comando git clone + o endereço do repositório do github.

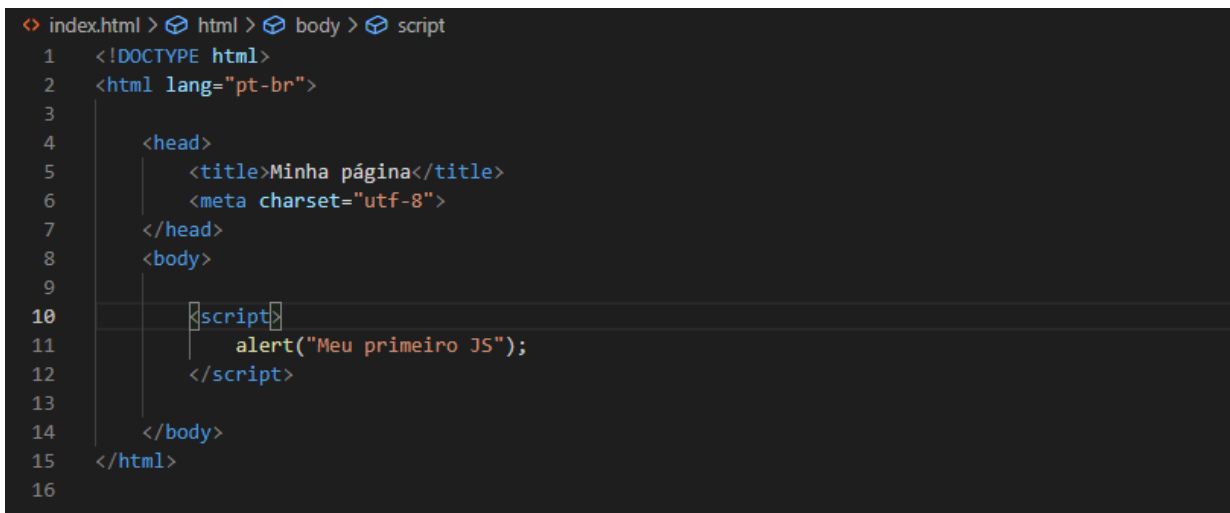
Com o repositório local criado é só abrir a pasta no visual code.

Para iniciar os primeiros códigos em JavaScript que tal criar um documento index.html básico?

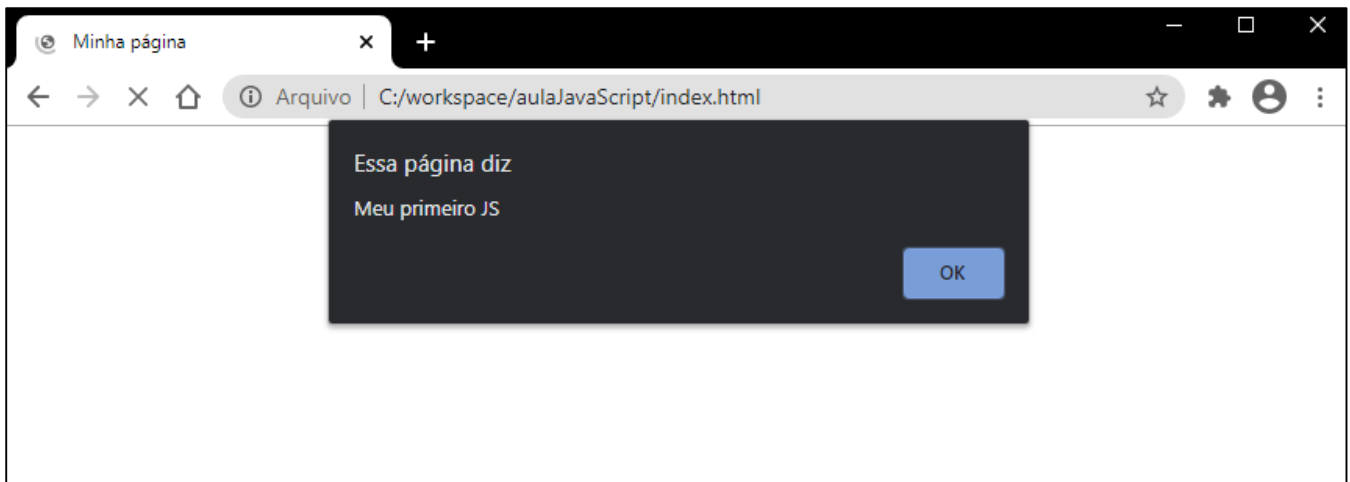


alert:

Esse comando cria uma caixa de diálogo no navegador como iremos ver a seguir.



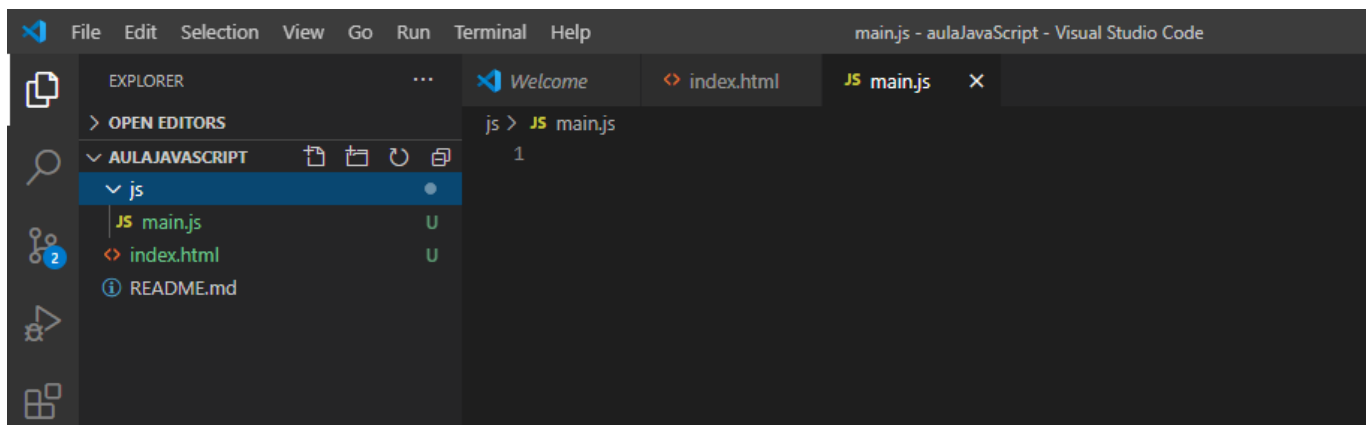
Com uma tag interna <script> adicionamos o código **alert("texto");** para criar uma caixa de diálogo (popup) no navegador.



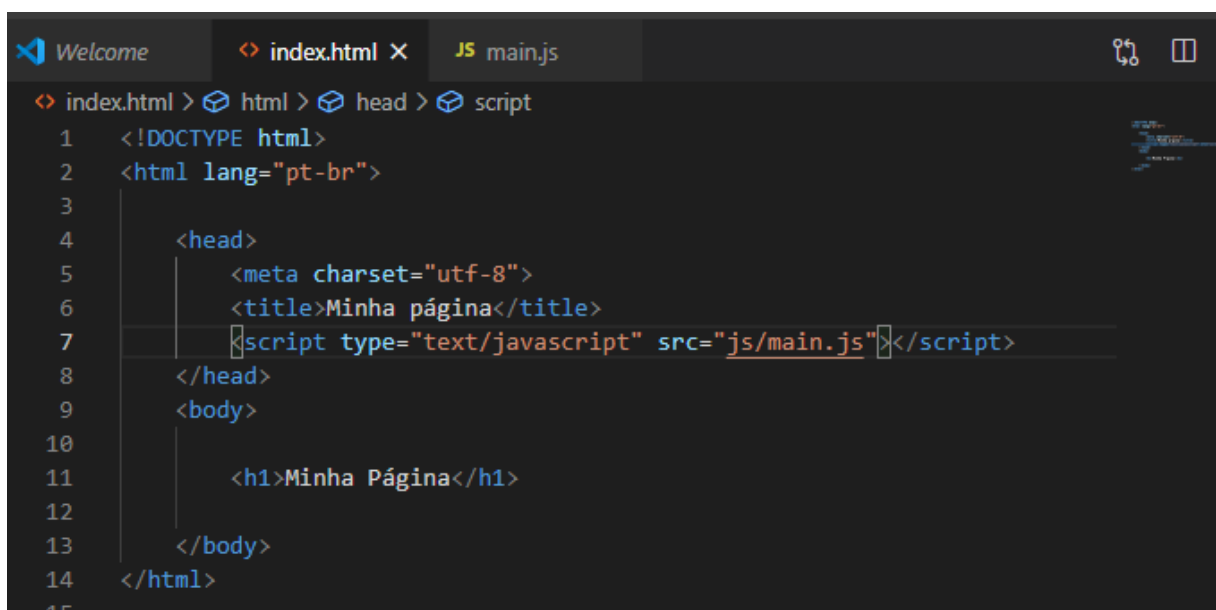
Ao abrir o index.html no navegador podemos visualizar o resultado.

JS externo:

Para trabalhar com js externo vamos criar uma pasta chamada **js** e um arquivo **main.js**.



No nosso index.html vamos adicionar ao <head> a tag <script> com o caminho do arquivo js que iremos utilizar na página.



Agora podemos trabalhar com o comando **alert();** e outros no nosso arquivo **main.js**.

Primeiras variáveis e concatenação.

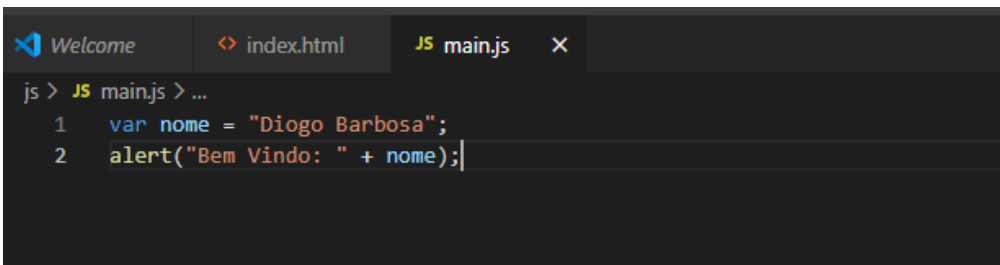
Para criar uma variável vamos utilizar a palavra **var** seguido do nome para essa variável e atribuir um valor com o símbolo de =.

var nome = "Diogo Barros";

Como o javascript tem tipagem dinâmica não foi preciso indicar que essa variável seria uma string pois ao atribuir o valor a variável o js já identificou isso, o mesmo serve para números. Strings são identificados por estarem em "" (aspas duplas).

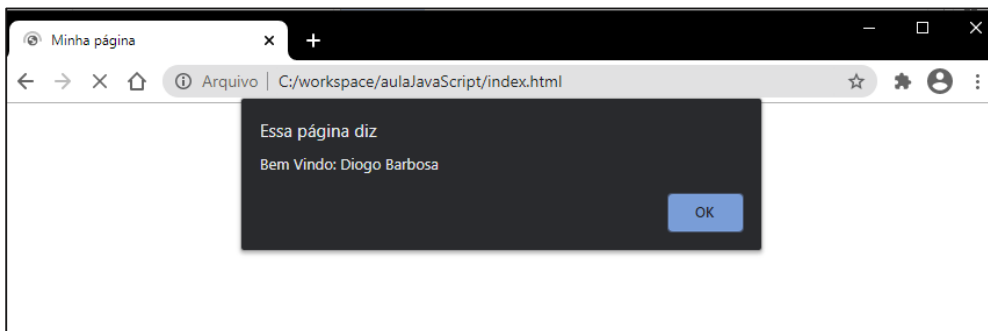
Para concatenar essa variável a algum texto vamos utilizar o símbolo + após o texto.

alert("texto" + variavel);

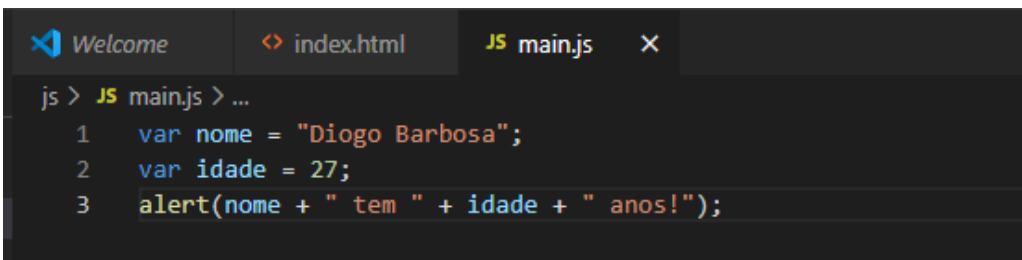
A screenshot of a code editor with a dark theme. The top bar shows three tabs: 'Welcome', 'index.html', and 'JS main.js'. The 'JS main.js' tab is active. The code in the editor is:

```
js > JS main.js > ...  
1  var nome = "Diogo Barbosa";  
2  alert("Bem Vindo: " + nome);
```

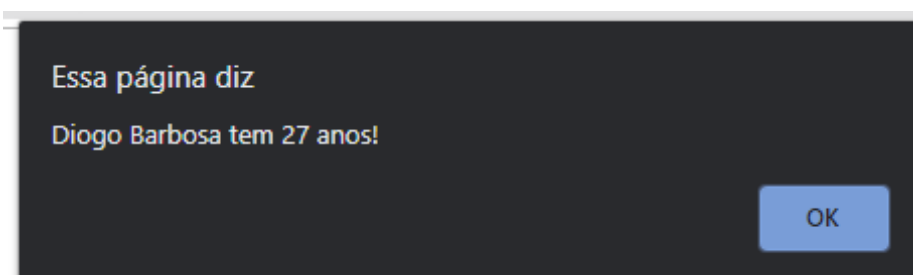
Resultado no navegador:



Mais um exemplo:

A screenshot of a code editor with a dark theme. The top bar shows three tabs: 'Welcome', 'index.html', and 'JS main.js'. The 'JS main.js' tab is active. The code in the editor is:

```
js > JS main.js > ...  
1  var nome = "Diogo Barbosa";  
2  var idade = 27;  
3  alert(nome + " tem " + idade + " anos!");
```

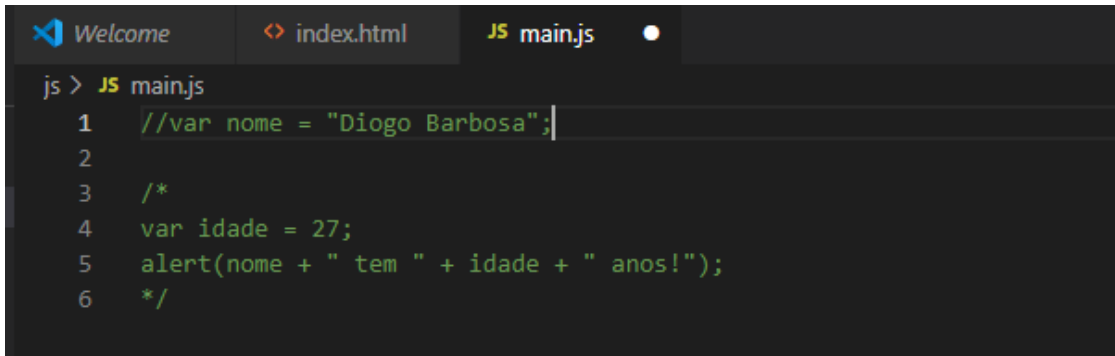


Comentários.

Podemos comentar códigos no javascript basicamente de duas formas.

// - comenta uma linha.

/* trecho de código */ - comenta todo código que esteja dentro desse bloco de comentário.

A screenshot of a code editor with a dark theme. The top bar shows tabs for 'Welcome', 'index.html', and 'JS main.js'. The editor content shows a JavaScript file named 'main.js' with the following code:

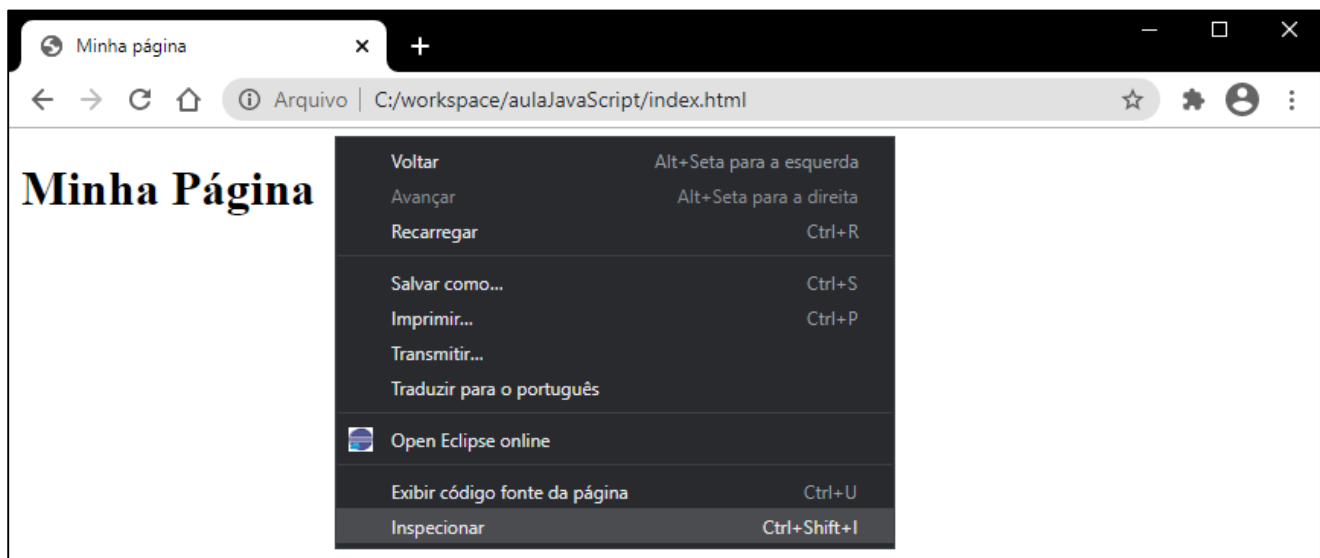
```
js > JS main.js
1 //var nome = "Diogo Barbosa";
2
3 /*
4 var idade = 27;
5 alert(nome + " tem " + idade + " anos!");
6 */
```

Todo código que estiver comentado será ignorado.

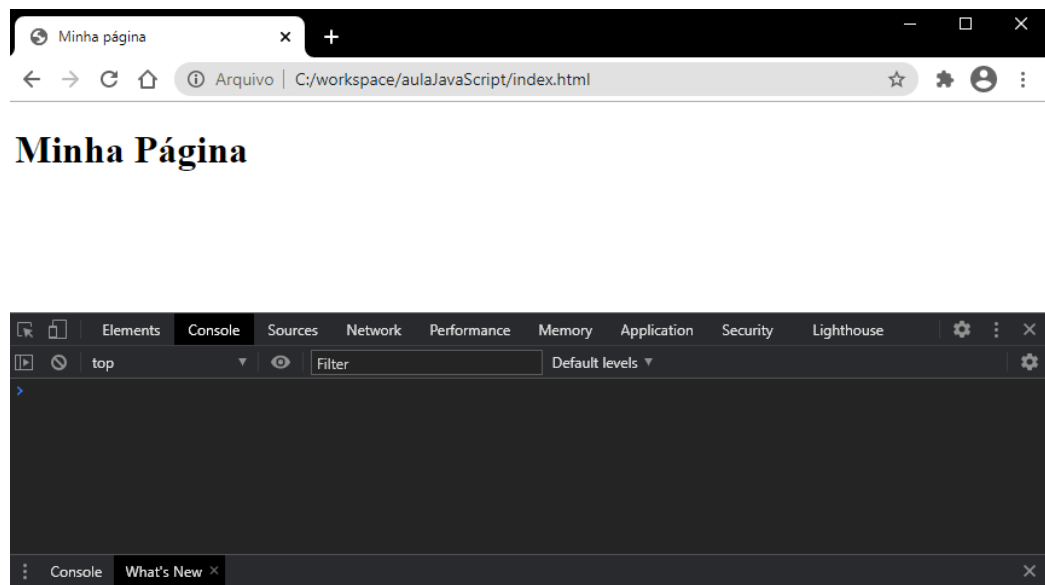
Comentários são uma boa prática, servem principalmente para documentar o projeto.

Console.

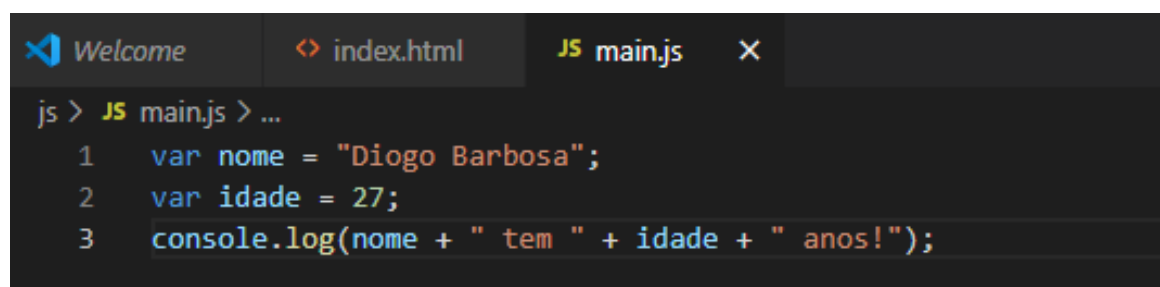
O console serve muito bem para acompanhar a saída do código sem precisar utilizar o **alert**. Para acessar o console no navegador é clicar com botão direito e ir em inspecionar ou **F12** no teclado.



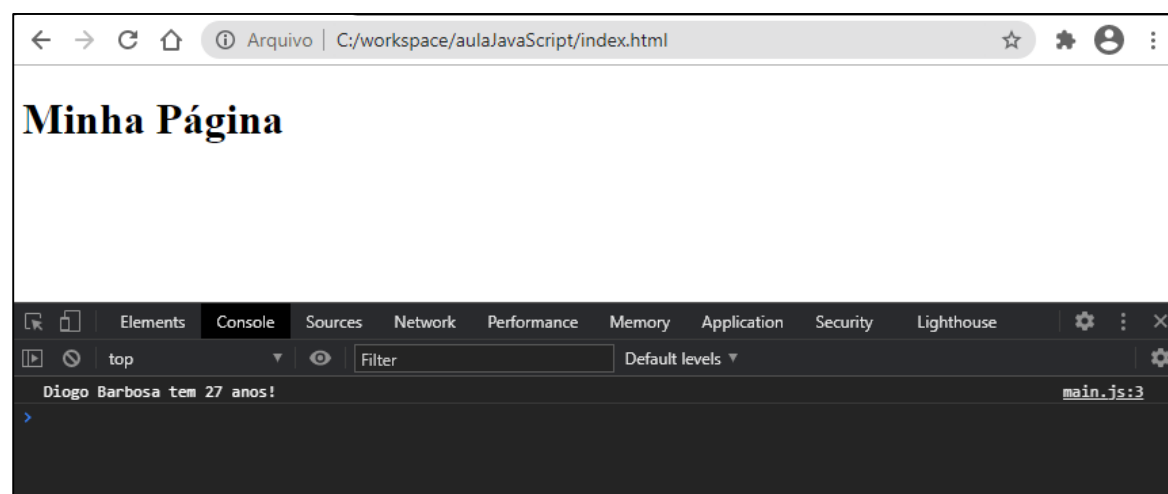
Quando abrir a caixa de ferramentas é só clicar na aba **console**.



Lá no nosso main.js vamos trocar o comando **alert** pelo **console.log** e já poderemos ver a saída diretamente no console ao invés da caixa de diálogo.



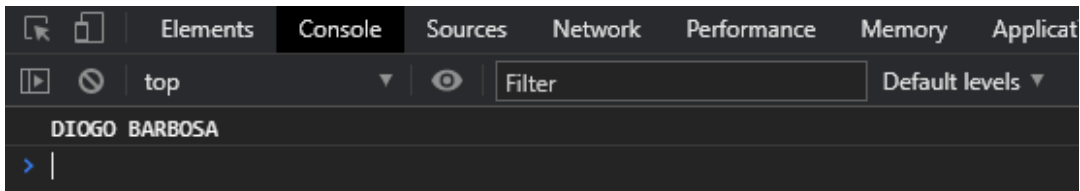
Saída:



Manipulando strings.

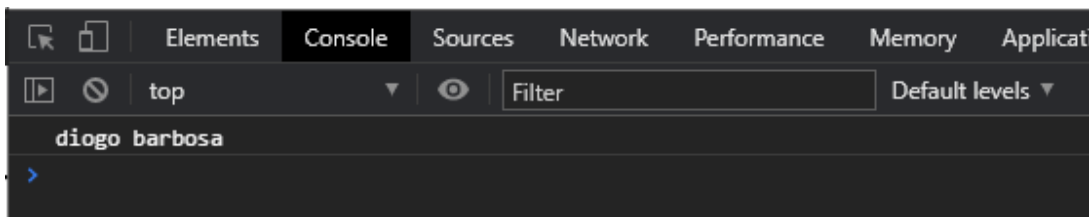
toUpperCase(): Transforma toda string em caixa alta (letras maiúsculas).

```
js > JS main.js > ...  
1  var nome = "Diogo Barbosa";  
2  var idade = 27;  
3  console.log(nome.toUpperCase());
```



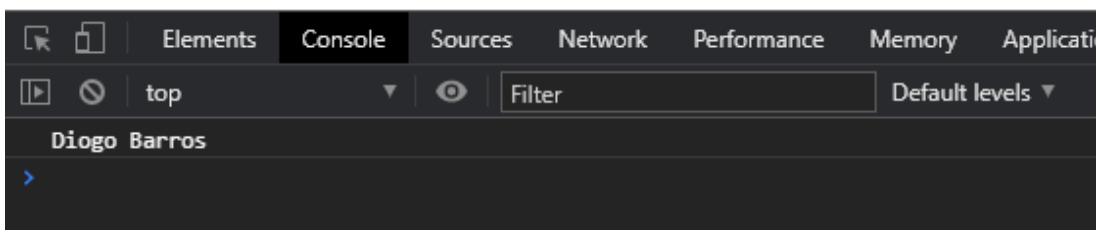
toLowerCase(): Transforma toda string em caixa baixa (letras minúsculas).

```
js > JS main.js > ...  
1  var nome = "Diogo Barbosa";  
2  var idade = 27;  
3  console.log(nome.toLowerCase());
```



Replace: Troca uma string especificada como parâmetro a esquerda por outra a direita separadas por vírgula.

```
js > JS main.js > ...  
1  var nome = "Diogo Barbosa";  
2  var idade = 27;  
3  console.log(nome.replace("Barbosa", "Barros"));
```



Operadores aritméticos.

Básicamente em js temos os operadores aritméticos, adição (+), subtração (-), multiplicação (*), e divisão (/).

```
js > JS main.js > ...
1  var num1 = 8;
2  var num2 = 2;
3
4  console.log("adição: " + (num1 + num2));
5  console.log("subtração: " + (num1 - num2));
6  console.log("multiplicação: " + (num1 * num2));
7  console.log("divisão: " + (num1 / num2));
```

```
Elements Console Sources Network Performance Memory Application
top Filter Default levels
adição: 10
subtração: 6
multiplicação: 16
divisão: 4
> |
```

Para concatenar uma operação aritmética com uma string é preciso colocar a operação entre parenteses.

Array – Listas.

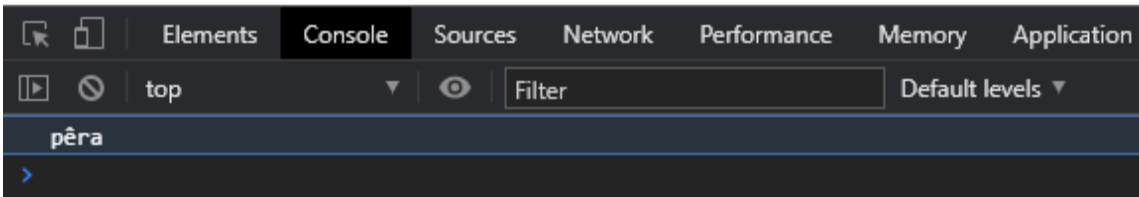
O tipo **array** é uma matriz de valores, seria como se tivéssemos vários elementos dentro de uma variável.

Para declarar uma lista em js vamos utilizar a sintaxe:

```
var lista = [0, 1, 2, 3];
```

Assim definimos uma lista de 4 posições a contar do 0 (zero);

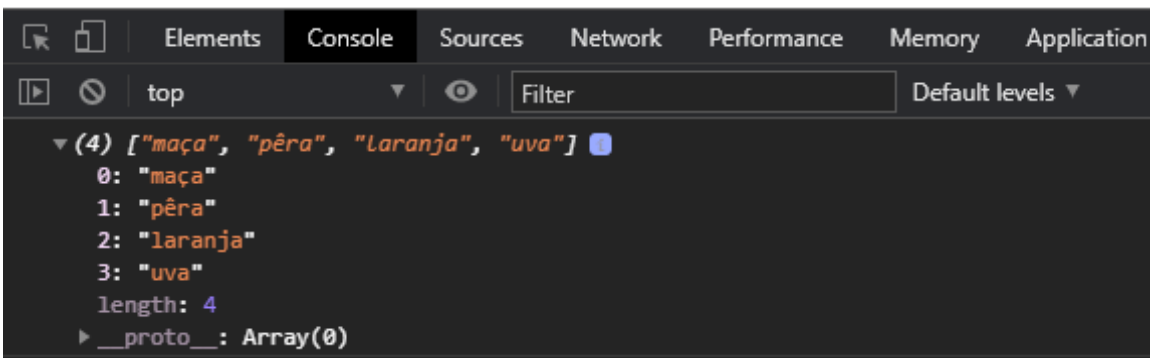

```
1 var lista = ["maça", "pêra", "laranja"];
2 console.log(lista[1]);
3
4
```



No código acima criamos um array de frutas chamado lista e acessamos a posição [1] “pêra” através do console.

push(); Adiciona mais um item a lista.

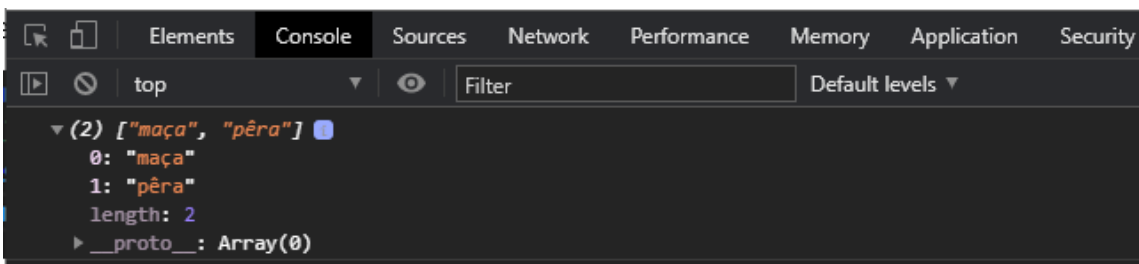
```
1 var lista = ["maça", "pêra", "laranja"];
2 lista.push("uva");
3 console.log(lista);
4
```



Adicionamos o item “uva” a lista e carregamos toda a lista no console.

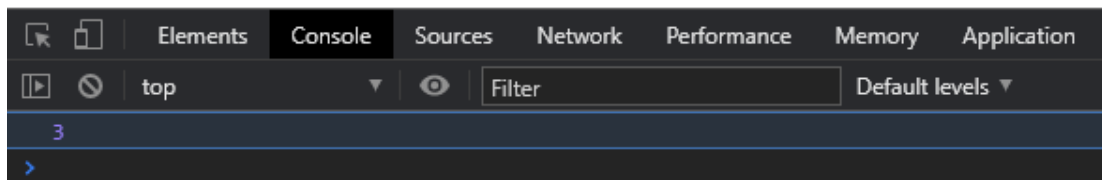
pop(); remove o ultimo item da lista.

```
1 var lista = ["maça", "pêra", "laranja"];
2 //lista.push("uva");
3 lista.pop();
4 console.log(lista);
5
```



lista.length; Mostra o tamanho do array. Ou seja, a quantidade de elementos da lista.

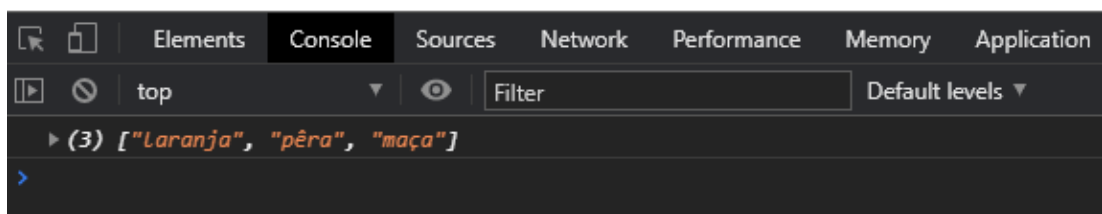
```
js > JS main.js > ...
1  var lista = ["maça", "pêra", "laranja"];
2  //lista.push("uva");
3  //lista.pop();
4  console.log(lista.length);|
5
```



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The output of the code is the number 3, indicating the length of the array.

reverse(); Retorna os itens da lista ao contrário.

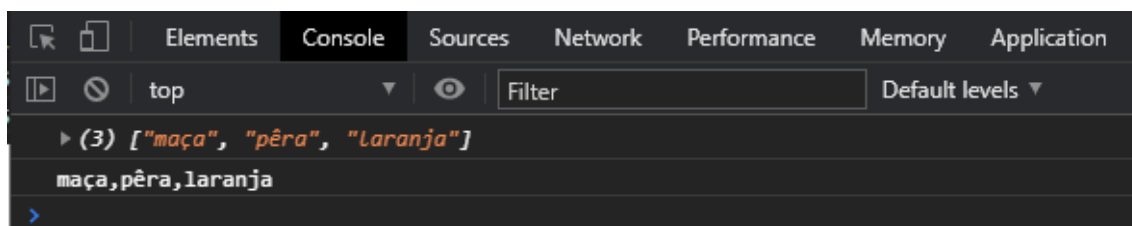
```
1  var lista = ["maça", "pêra", "laranja"];
2  //lista.push("uva");
3  //lista.pop();
4  console.log(lista.reverse());|
5
```



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The output of the code is an array ["Laranja", "pêra", "maça"], indicating that the original array has been reversed in place.

toString(); retorna a lista como string e não mais como um elemento array.

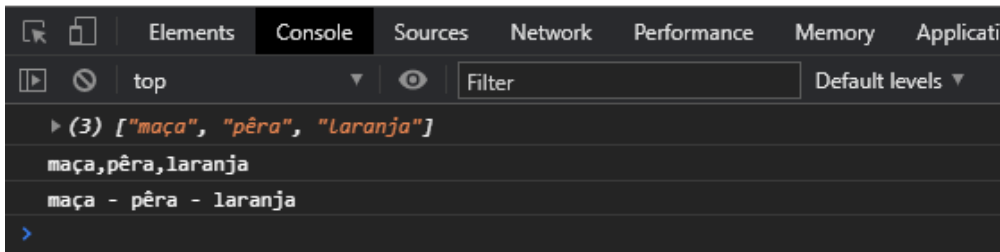
```
1  var lista = ["maça", "pêra", "laranja"];
2  //lista.push("uva");
3  //lista.pop();
4  console.log(lista);
5  console.log(lista.toString());|
```



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The output of the code is two lines: ["maça", "pêra", "laranja"] and maçã,pêra,laranja, indicating that the array was converted to a string.

join(); funciona como o **toString()** mas adiciona uma formação simples a lista o qual pode ser um espaço, traço, pipeline etc.

```
1 var lista = ["maça", "pêra", "laranja"];
2 //lista.push("uva");
3 //lista.pop();
4 console.log(lista);
5 console.log(lista.toString());
6 console.log(lista.join(" - "));
7
```

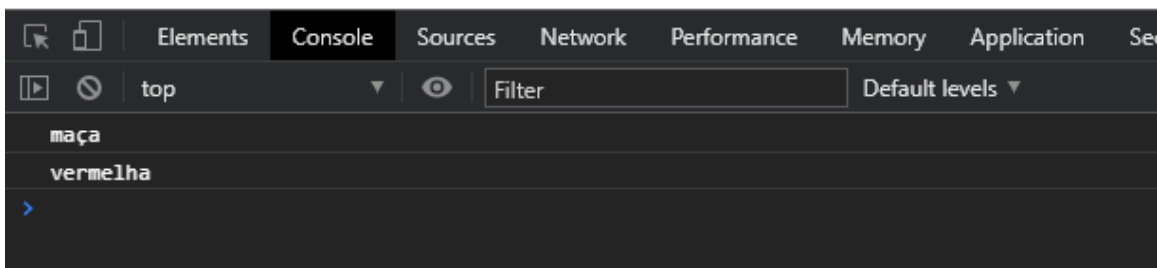


Dicionário.

Para criar um dicionário usamos uma sintaxe parecida como se cria um objeto atribuindo diferentes valores de um mesmo elemento.

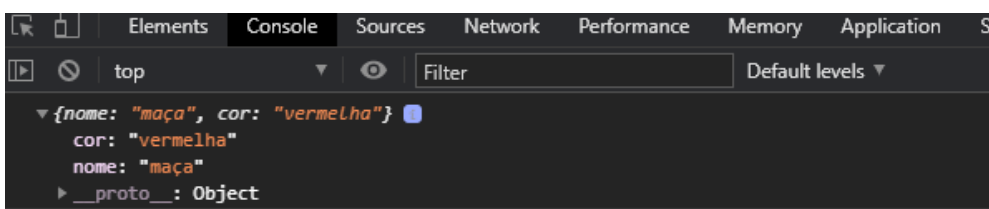
var fruta = {nome: " ", cor: " " }

```
2
3 var fruta = {nome:"maça" , cor:"vermelha"}
4 console.log(fruta.nome);
5 console.log(fruta.cor);
6
```



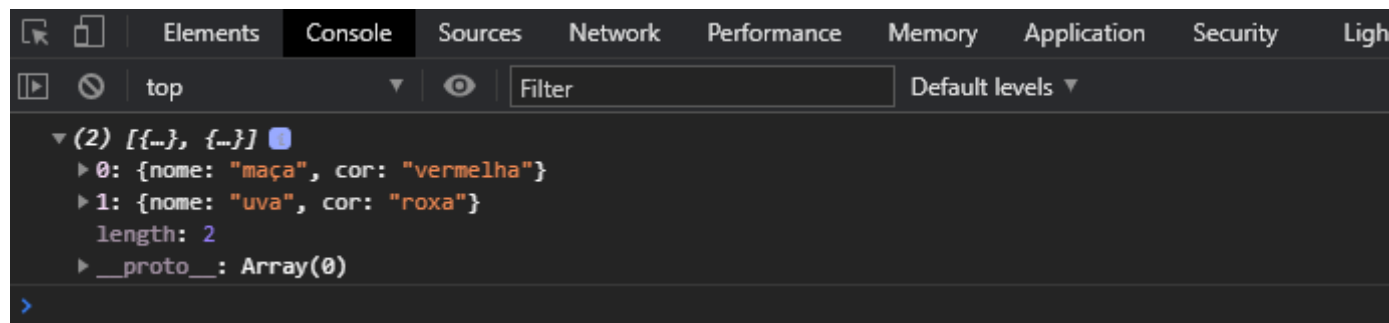
Criamos o objeto e acessamos suas propriedades através do console.

```
3 var fruta = {nome:"maça" , cor:"vermelha"}
4 //console.log(fruta.nome);
5 //console.log(fruta.cor);
6 console.log(fruta);
7
```



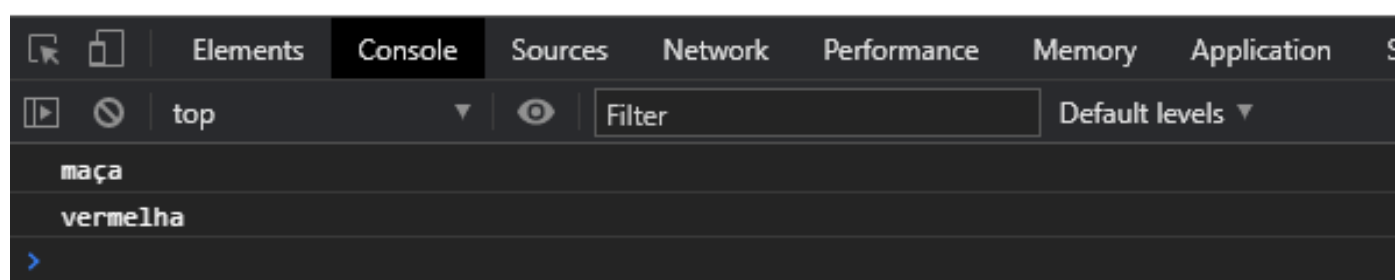
Apartir daí também podemos criar uma lista de dicionários.

```
3 var frutas = [{nome:"maça", cor:"vermelha"}, {nome:"uva", cor:"roxa"}]
4 console.log(frutas);
5
```



outro exemplo.

```
3 var frutas = [{nome:"maça", cor:"vermelha"}, {nome:"uva", cor:"roxa"}]
4 //console.log(frutas);
5 console.log(frutas[0].nome);
6 console.log(frutas[0].cor);
7
```



Estrutura Condicional.

If-Else

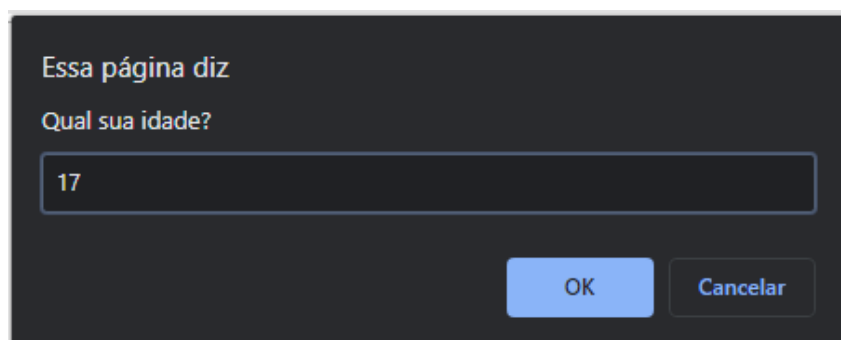
```
3  var idade = 18;
4
5  if (idade >= 18){
6      alert("Maior de idade!");
7  }else{
8      alert("Menor de idade!")
9  }
```



Na estrutura condicional o **if** irá avaliar uma condição se ela for verdadeira irá executar seu bloco de código caso seja falso irá passar para o proximo código no caso o **else**.

prompt(); Necessita de uma entrada de dados do usuário e armazena na variável.

```
3  var idade = prompt("Qual sua idade?");
4
5  if (idade >= 18){
6      alert("Maior de idade!");
7  }else{
8      alert("Menor de idade!")
9  }
```



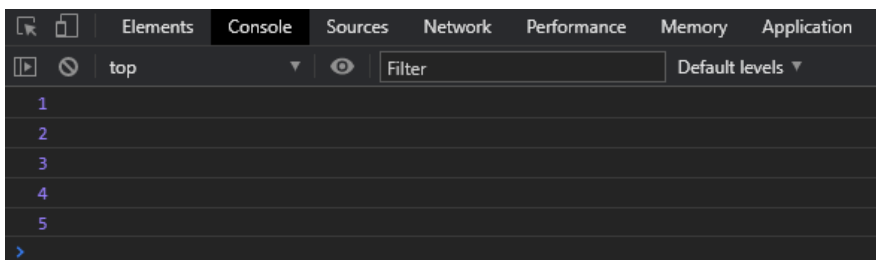
O **prompt()** recebe a informação armazena na variável e a estrutura condicional faz o seu trabalho.



Laços de repetição.

while(); Recebe uma condição a ser avaliada, se for verdadeira o bloco de código será executado várias vezes até satisfazer tal condição.

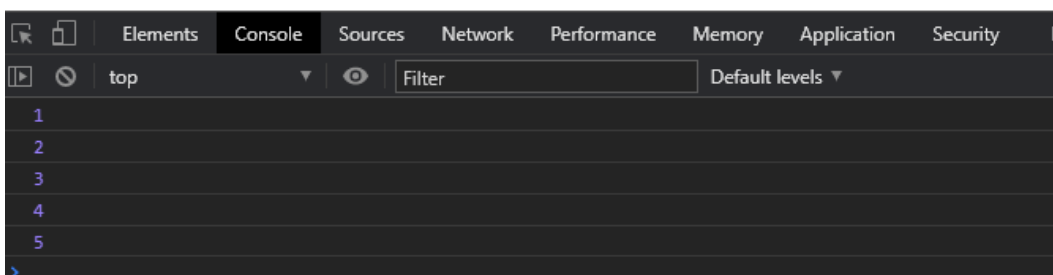
```
3  /** while */
4  var count = 1;
5  while (count <= 5){
6      console.log(count);
7      count++;
8  }
```



Avalia uma condição em que **count** é menor que 5, sendo verdadeira, o **console.log** é executado várias vezes e a cada interação **count** é incrementada em + 1, quando a condição se torna falsa (quando **count** se torna maior que 5) o **while** termina sua execução.

For(); O laço **for** segue a mesma lógica.

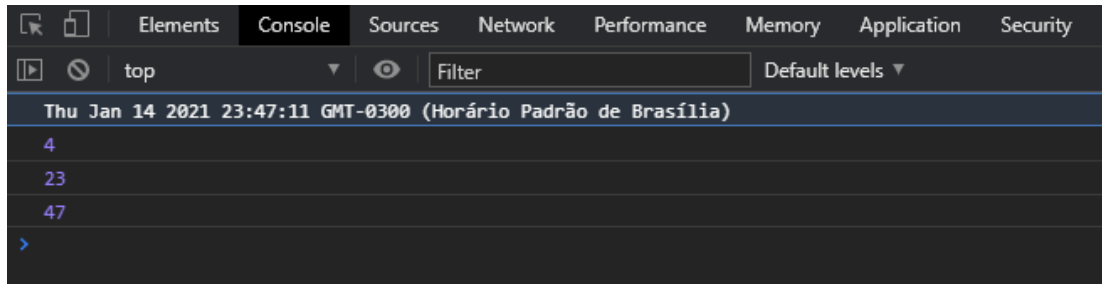
```
3  /** For */
4  var count;
5  for (count = 1; count <= 5; count++){
6      console.log(count);
7  }
```



Datas.

Para pegar datas, horas etc. vamos utilizar a função **Date()**;

```
3  var d = new Date();
4  console.log(d);
5  console.log(d.getDay());
6  console.log(d.getHours());
7  console.log(d.getMinutes());|
8
```



Aqui temos alguns exemplos de como recuperar a variável no **console** e também pode ser utilizado no **alert**.

Função.

Para chamar uma função utilizamos o comando **function** o nome dada a função seguido de parênteses **function nome()**.

Função para somar 2 números.

```
3  function soma(n1, n2){
4  |    return n1 + n2;
5  |  }
6
7  alert(soma(5 ,10));
8
```

Aqui criamos uma função chamada soma que contem duas variáveis, **n1** e **n2**, no bloco de código indica que essa função retorna a soma entre as duas variáveis. E por fim o **alert** chama a função e passa 2 números como parâmetro, o resultado será a soma desses 2 números.



Função replace().

```
3 function setReplace(nome, sobreNome, novoSobrenome){
4     return nome.replace(sobreNome, novoSobrenome)
5 }
6
7 alert(setReplace("Diogo Barbosa", "Barbosa", "Barros"));
8
```

Esse função recebe três variáveis e retorna o nome mas dando um **replace** (substituindo) o sobre nome por outro. Por fim o **alert** chama a função passando os três valores como parâmetro.

Essa página diz

Diogo Barros

OK

Função para validar idade.

```
3 var validar;
4 function validaIdade(idade){
5     if (idade >= 18){
6         validar = "Você é maior de idade!"
7     }else{
8         validar = "Você é menor de idade!"
9     }
10    return validar;
11 }
12
13 var idade = prompt("Qual a sua idade?");
14 validaIdade(idade)
15 alert(validar);
16
```

Essa página diz

Qual a sua idade?

18

OK

Essa página diz

Você é maior de idade!

OK

Tal função recebe tem uma variável **idade** que receberá uma valor como parâmetro através de uma pergunta ao usuário com o **prompt()**, se o valor (idade) for maior ou igual a 18 a variável **valor** recebe um texto se não recebe outro texto, que será exibido com o **alert**.

Manipulando elementos na página.

Vamos trabalhar com os elementos do nosso arquivo **html** para chamar e executar funções do nosso arquivo externo **js**.

button + onclick.

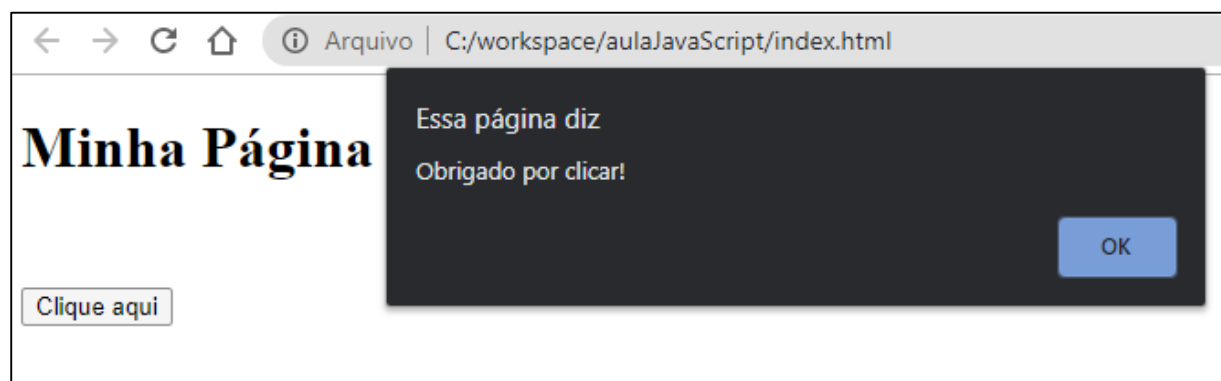
```
9      <body>
10
11      <h1>Minha Página</h1>
12      </br>
13      </br>
14      <button type="button" onclick="clizou()">Clique aqui</button>
15
16  </body>
```

No **body** do nosso html vamos criar um botão **<button>** que chama uma função com o evento **onclick="nome-da-função"**.

```
3  function clicou(){
4      alert("Obrigado por clicar!");
5  }
6
7
```

No nosso documento **js** precisamos agora criar a função que será executada ao clicar no botão. Essa função executa um **alert** na página.

Resultado.



document.getElementById() + innerHTML

Também é possível injetar algum valor dentro de um elemento html com o **innerHTML**.

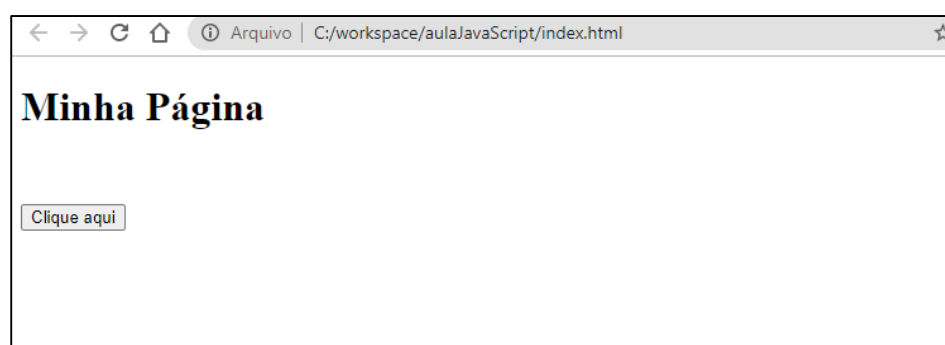
```
9      <body>
10
11      <h1>Minha Página</h1>
12      </br>
13      </br>
14      <button type="button" onclick="clique()">Clique aqui</button>
15      </br>
16      <h3 id="clique-text"></h3>
17
18  </body>
```

Criamos um título **h3** em nosso html com um **id** que será utilizado pelo **document.getElementById()** para identificar o elemento a ser manipulado na função.

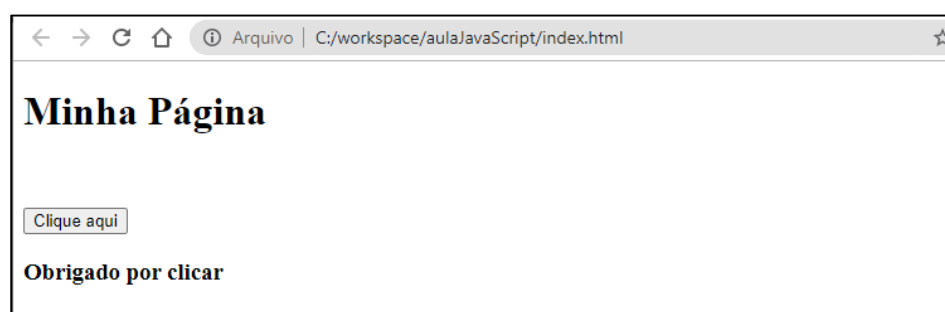
```
2
3  function clique(){
4      //alert("Obrigado por clicar!");
5      document.getElementById("clique-text").innerHTML = "Obrigado por clicar";
6  }
7
```

A função em vez de exibir um **alert** ao clicar no botão como antes, pegará um elemento do documento html através do **id** passado como parâmetro pro **document.getElementById()** e injetará naquele elemento um valor com o **innerHTML**, nesse caso um texto, mas também poderia ser uma **tag** HTML.

Resultado.



Ao clicar no botão será exibido o texto que foi injetado dentro do elemento **h3**.



onmouseover.

Executa uma ação ao passar o mouse sobre o elemento.

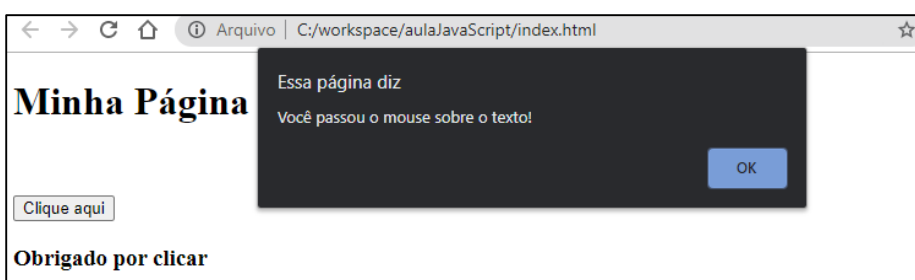
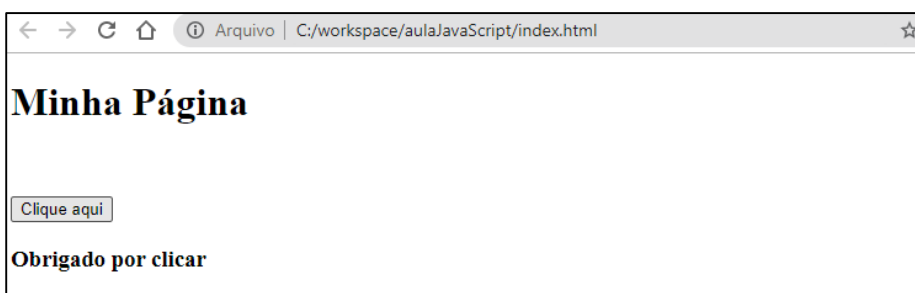
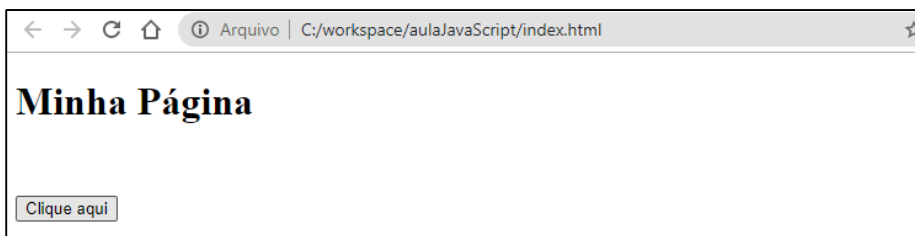
```
9      <body>
10
11      <h1>Minha Página</h1>
12      </br>
13      </br>
14      <button type="button" onclick="clique()">Clique aqui</button>
15      </br>
16      <h3 id="clique-text" onmouseover="textMouse()"></h3>
17
18  </body>
```

Com o mesmo código anterior adicionamos ao elemento **h3** o comando **onmouseover** chamando a função **“textMouse()”** no nosso documento html.

```
3  function clique(){
4      //alert("Obrigado por clicar!");
5      document.getElementById("clique-text").innerHTML = "Obrigado por clicar";
6  }
7
8  function textMouse(){
9      alert("Você passou o mouse sobre o texto!");
10 }
11
```

E no nosso documento js, criamos a função que irá disparar um **alert** quando o usuário passa o mouse sobre o texto.

Resultado.



onmouseover & onmouseout.

Podemos utilizar esses comandos em conjunto para trocar o conteúdo de um elemento.

```
9  <body>
10
11  <h1>Minha Página</h1>
12  </br>
13  </br>
14  <button type="button" onclick="clique()">Clique aqui</button>
15  </br>
16  <h3 id="clique-text" onmouseover="textMouse()"></h3>
17  </br>
18  <p id="mousemove" onmouseover="trocar()" onmouseout="voltar()">Passe o mouse aqui!</p>
19
20 </body>
```

No nosso documento html criamos uma tag **p** com **id** e chamada a duas funções com **onmouseover** e **onmouseout**.

```
11 function trocar(){
12     document.getElementById("mousemove").innerHTML = "Obrigado por passar o mouse!"
13 }
14 function voltar(){
15     document.getElementById("mousemove").innerHTML = "Passe o mouse aqui!"
16 }
```

E no documento js temos as duas funções. Ao colocar o mouse sobre o texto **onmouseover** será executada a primeira função e ao retirar o mouse **onmouseout** será executada a segunda função.

sem o mouse.

Passe o mouse aqui!

com o mouse.

Obrigado por passar o mouse!

sem o mouse.

Passe o mouse aqui!

onmouseover & onmouseout + this.

O exemplo acima utiliza o **id** junto com o **getElementById** para manipular tal elemento. Mas caso tivéssemos outros elementos para a mesma função com **id** diferente ou até mesmo sem **id** as funções não funcionariam nesses outros elementos. Por tanto vamos utilizar o **this** para pegar o elemento ao qual o mouse esteja sobre ele.

```
20 <p onmouseover="trocar2(this)" onmouseout="voltar2(this)">Sem o mouse!</p>
21 <p onmouseover="trocar2(this)" onmouseout="voltar2(this)">Sem o mouse!</p>
22 <p onmouseover="trocar2(this)" onmouseout="voltar2(this)">Sem o mouse!</p>
23
```

Criamos 3 tags **p** no documento html com **onmouseover** e **onmouseout** passando o argumento **this** como parâmetro nas funções.

```
18 function trocar2(elemento){
19     elemento.innerHTML = "Com o mouse!"
20 }
21 function voltar2(elemento){
22     elemento.innerHTML = "Sem o mouse!"
23 }
24
```

No documento js criamos as duas funções referente passando como parâmetro o argumento **elemento** que irá refenciar o **this** passado pela tag.

Resultado.

Com o mouse!
Sem o mouse!
Sem o mouse!

Sem o mouse!
Com o mouse!
Sem o mouse!

Sem o mouse!
Sem o mouse!
Com o mouse!

onload.

Chama uma função ao carregar a página.

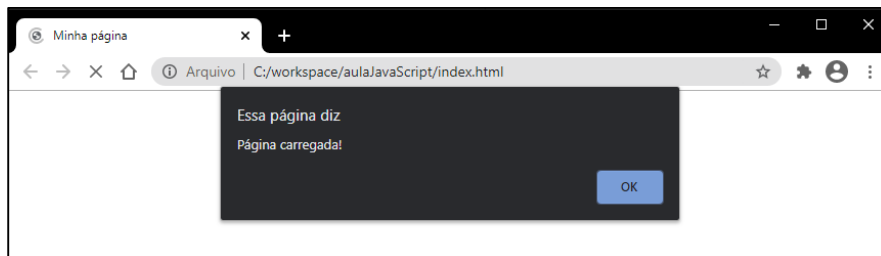
```
9 | <body onload="load()">
10 |
11 |     <h1>Minha Página</h1>
12 |     </br>
13 |     </br>
```

Adicionamos a tag **body** o **onload** chamando a função load que executará algo assim que o **body** for carregado.

```
25 | function load(){
26 |     alert("Página carregada!");
27 | }
```

Função load no documento js referenciado ao **body** irá executar um **alert** ao carregar a página.

Resultado.



onchange.

Quando se tem um combo de opções o **onchange** sempre o usuário trocar o valor a função irá pegar o valor selecionado.

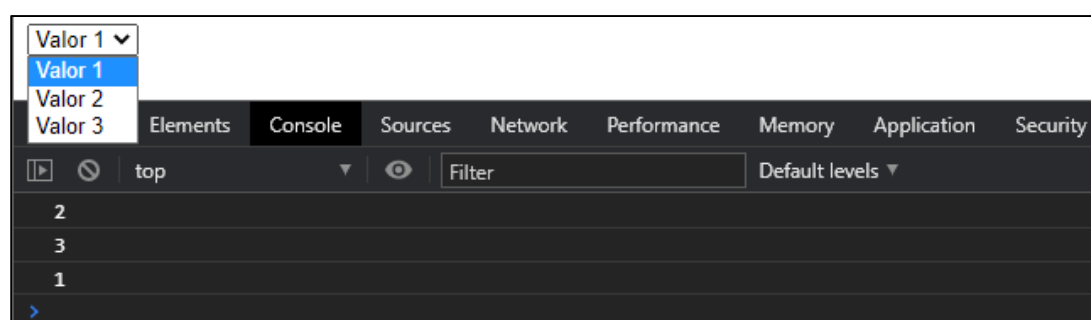
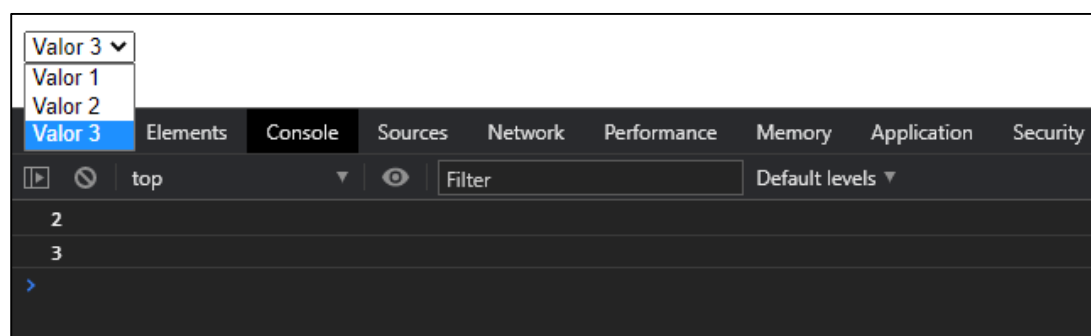
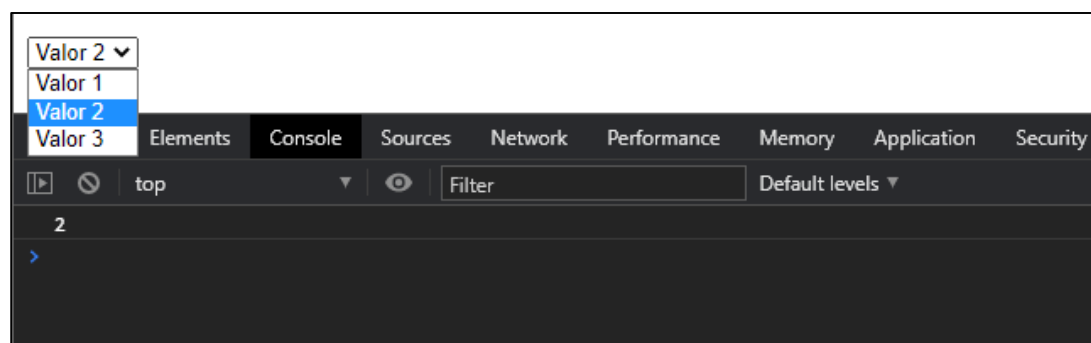
```
24 <select onchange="funcChange(this)">
25   <option value="1">Valor 1</option>
26   <option value="2">Valor 2</option>
27   <option value="3">Valor 3</option>
28 </select>
```

Criamos em nosso html um seletor com 3 opções, com o **onchange** chamando uma função.

```
29 function funcChange(elemento){
30   console.log(elemento.value);
31 }
32
```

A função em no documento js irá exibir o valor do elemento referenciado com o **value**.

Resultado.



window.open();

Abre determinado link em uma nova aba no navegador.

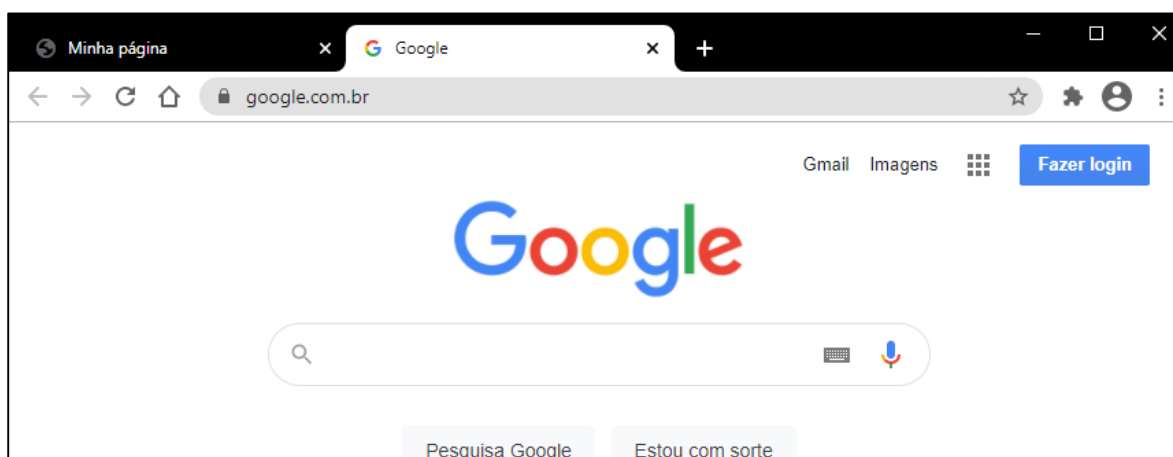
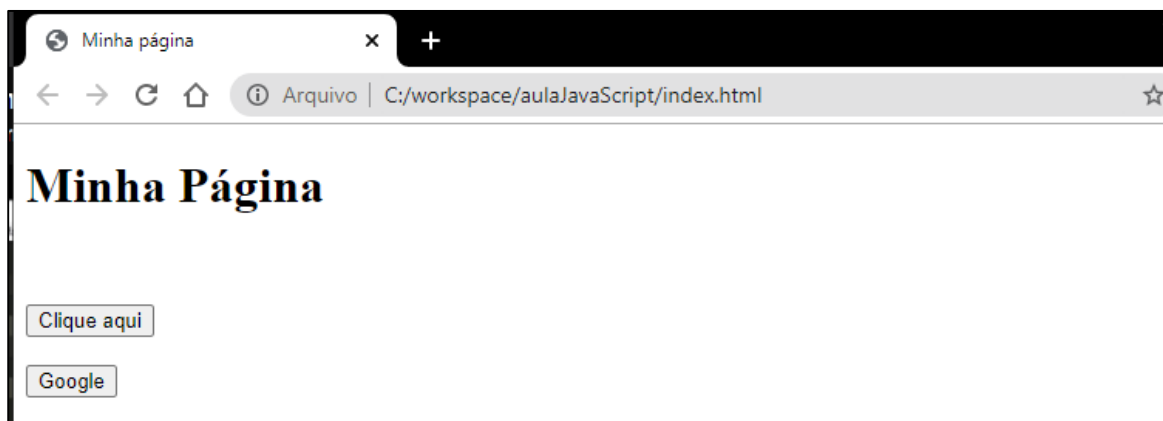
```
15     </br>
16     </br>
17     <button type="button" onclick="redirecionar1()">Google</button>
18 </br>
```

Logo abaixo do primeiro botão do nosso documento html criamos um botão **google** com um evento **onclick** fazendo uma chamada a função especificada, mas poderia ser um texto ou uma imagem etc.

```
33 function redirecionar1(){
34     window.open("https:\\google.com.br");
35 }
36
```

No documento js a função contém um **window.open()** passando um link ao qual devemos ser redirecionados a uma nova aba ao clicar no botão.

resultado.



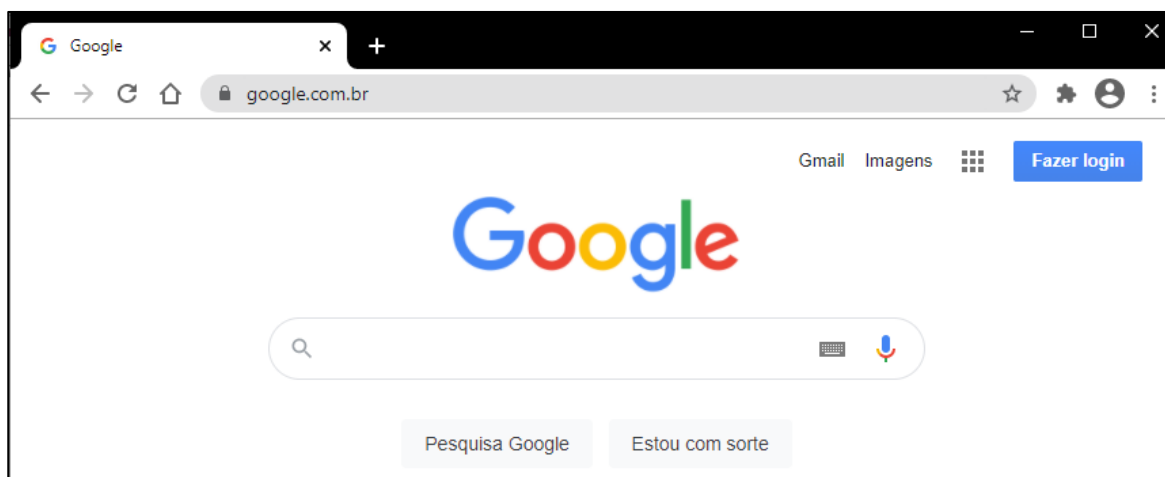
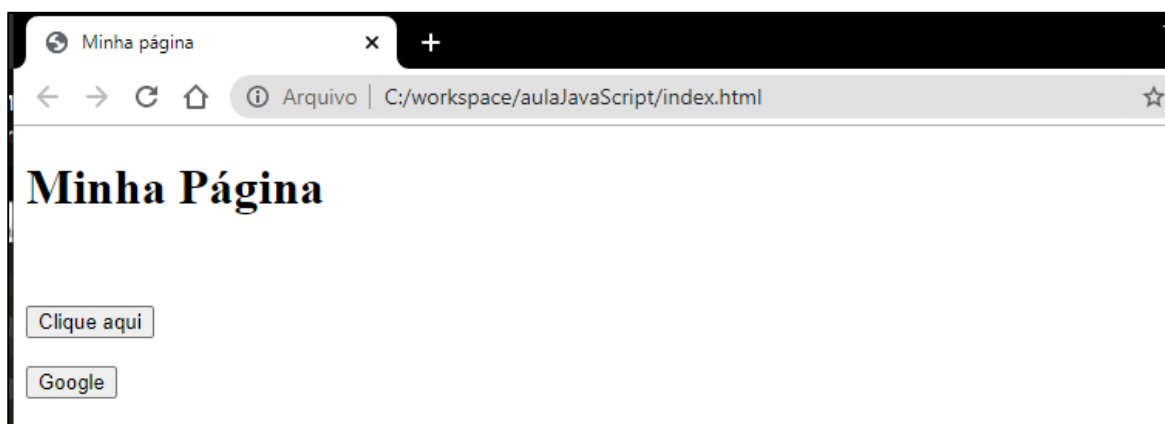
window.location.href = ""

Redireciona para um link especificado mas diferendo do **window.open** irá abrir o link na aba atual.

```
function redirecionar1(){  
    //window.open("https:\\google.com.br");  
    window.location.href="https:\\google.com.br";  
}
```

Vamos utilizar a mesma função, então precisamos comentar o código do **window.open** e adicionar no lugar o **window.location.href = ""**.

resultado.



Podemos observar que fomos redirecionados na mesma aba.