

Graduação de Tecnologia em Análise e Desenvolvimento de Sistemas

Tecnologia Web II

Prof. Reinaldo Freitas



Agenda

- **Introdução**
- **IDE**
- **Conceitos básicos**
- **Estruturas condicionais**
- **Estruturas de repetição**
- **Funções**
- **Orientação a objetos**
- **Objetos de host**
- **Formulários**
- **AJAX**
- **Outros assuntos**



Introdução



Introdução



JavaScript

- Linguagem de programação criada com o objetivo de dar interatividade a uma página web.
- Desenvolvida pela Netscape em 1995 em parceria com a Sun Microsystems. Por conta da falta de padronização dos navegadores na época, visto que a linguagem funcionava em um navegador e em outros não, em 1996 a Netscape decidiu entregar o JavaScript para a ECMA (*European Computer Manufactures Association*) que ficou responsável pela padronização da linguagem.
- Por conta disso a linguagem JavaScript é conhecida como ECMAScript e as versões da linguagem estão associadas a este nome (ex: ECMAScript 6, ECMAScript 7, etc.). Maiores informações consulte o link abaixo:

<https://ecma-international.org/>

Introdução



JavaScript

- A linguagem JavaScript é *case sensitive*, ou seja, diferencia letras maiúsculas e minúsculas.
- Com ela é possível:
 - ☐ Manipular conteúdo e apresentação;
 - ☐ Manipular o navegador;
 - ☐ Interagir com formulários;
 - ☐ Interagir com outras linguagens.
 - ☐ Para maiores informações consulte:
<http://www.w3schools.com/js/default.asp>

Introdução



Arquitetura Web

- As aplicações Web normalmente trabalham em 3 camadas.
- A primeira camada é chamada de Apresentação e é onde são executadas no navegador (Chrome, Firefox, Edge, etc.) os códigos em html e scripts (JavaScript, VBScript, etc.).
- A segunda camada é camada de Negócio porque nela fica contida a lógica de negócio da aplicação, executando normalmente linguagens como PHP, Java e ASP, em servidores Web (Apache, IIS, Tomcat, WebLogic, etc.). Os códigos em JavaScript também podem ser executados nesta camada, apesar de ser mais comum a execução na primeira camada.
- A terceira camada é chamada de Persistência porque é onde são gravados (persistidos) em bancos de dados (Oracle, SQL Server, MySQL, etc.) as informações manipuladas pela aplicação.

Introdução

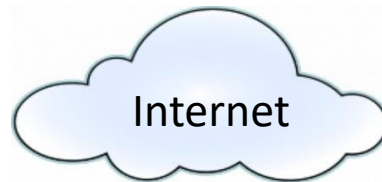


Arquitetura Web

Primeira camada
(Apresentação)

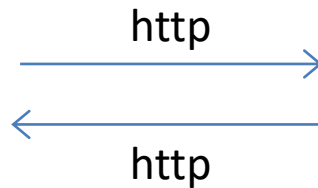
Segunda camada
(Negócio)

Terceira camada
(Persistencia)



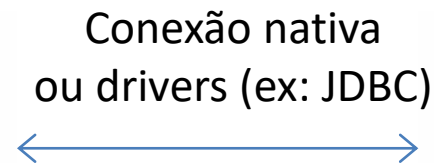
Navegador

HTML
Javascript
Applet
VBScript
Plug-in (Flash, adobe, etc.)



Servidor Web

PHP (Hypertext Preprocessor – extensão .php)
JSP (JavaServer Pages – extensão .jsp)
JSF (JavaServer Faces – extensão .xhtml)
ASP (Active Server Pages – extensão .asp)



Servidor de Banco de dados SQL

Formas de inclusão do código JavaScript

- O código JavaScript pode ser incluído em uma página Web das seguintes formas:
 - ☐ **Incorporado:** delimitado pela tag **<script>**. Será executado antes do carregamento da página se for incluído dentro da tag **<head>**.
 - ☐ **Externo:** dentro de um arquivo externo sendo chamado dentro da tag **<head>**;
 - ☐ **Inline:** dentro de alguma instrução contida na tag **<body>**.

Introdução



Formas de inclusão do código JavaScript

- Incorporado:

`<script type="text/javascript">` → Início do Javascript

Declarações ou Instruções; → Conteúdo do Javascript

`</script>` → Final do Javascript

Introdução



Formas de inclusão do código JavaScript

- Externo:

```
<script type="text/javascript" src="arquivo.js"></script>
```

- Inline:

```
<button type="button" onclick="alert('Atenção!')">Atenção</button>
```

Introdução



Exercícios:

- 1) O que é o Javascript?
- 2) Em que camada pode ser executado um JavaScript?
- 3) Qual a tag usada para delimitar um JavaScript?
- 4) Quais são as formas de inserção de um JavaScript em uma página web?

IDE



IDE (Integrated Development Environment)

- É um software que permite a desenvolvedores criarem programas usando uma plataforma integrada que facilita a codificação, aumentando assim a sua produtividade. Dentre os vários recursos de uma IDE podemos citar:
 - ❑ Preenchimento automático de códigos;
 - ❑ Correção automática de código;
 - ❑ Depuração de código;
 - ❑ Possibilidade de trabalhar com várias linguagens de programação através de plugins;
 - ❑ Integração com outras ferramentas.

IDE (Integrated Development Environment)

- Seguem algumas das IDEs mais conhecidas para a linguagem JavaScript:
 - ❑ Visual Studio Code (<https://code.visualstudio.com/>)
 - ❑ Netbeans (<https://netbeans.apache.org/>)
 - ❑ Sublime Text (<https://www.sublimetext.com/index2>)
- Utilizaremos nesta unidade curricular o software **Visual Studio Code**, também conhecido como **VS Code**.

Criação de programa

- No **Windows Explorer**, crie um diretório chamado **arquivos** para as nossas páginas HTML e dentro dele crie um subdiretório chamado **js** que conterá os nossos códigos em JavaScript. No **VS Code**, selecione o diretório **arquivos** através da opção **File -> Open Folder**. Clique na opção **File -> New Text File**, selecione **HTML (html)** na opção **Select a Language** e crie no diretório **arquivos** o arquivo **js1.html** com o conteúdo abaixo. Lembre-se de que a linguagem JavaScript é sensível ao tamanho da caixa (*Case Sensitive*), então as palavras reservadas da linguagem devem ser digitadas conforme são apresentadas.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Uso de caixa de alerta</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js1.js"></script>
8      <script type="text/javascript">
9          alert("Seja Bem-vindo(a)!");
10         confirm("Deseja continuar?");
11     </script>
12 </head>
13
14 <body>
15     <button type="button" onclick="exibeMensagem()">Exibir mensagem</button>
16 </body>
17
18 </html>
```

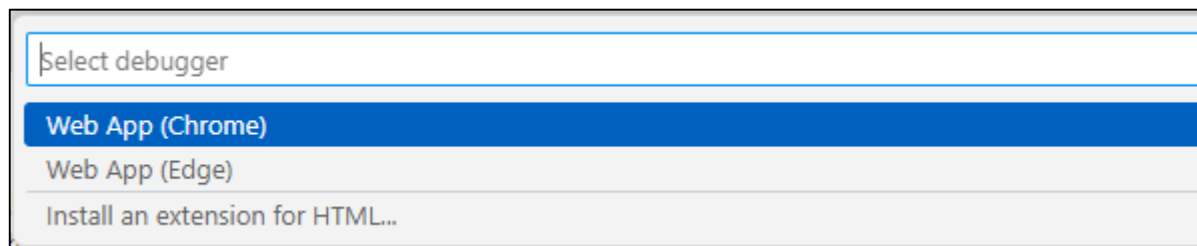
Criação de programa

- No **VS Code**, clique novamente na opção **File -> New Text File**, selecione **JavaScript (Javascript)** na opção **Select a Language** e crie também o arquivo **js1.js** dentro do subdiretório **js** com o conteúdo abaixo. Veja que o nome da função **exibeMensagem** não é uma palavra reservada da linguagem, então poderia ser escrita de diversas maneiras que não afetaria a execução, desde que a chamada da função também fosse modificada, mas crie da forma apresentada abaixo por questão de convenção que veremos mais adiante.

```
1  function exibeMensagem() {  
2      |      prompt("Entre com uma data:", "dd/mm/aaaa");  
3  }
```


Execução de programa

- Execute o arquivo **js1.html** no VS Code utilizando a opção **Run -> Run Without Debugging**. Pode também pressionar as teclas CTRL + F5.
- Ao ser apresentada a janela abaixo, selecione a opção **Web App (Chrome)** para abrir o arquivo no navegador **Chrome**.



Execução de programa

- Por conta do código **incorporado** contido na tag **<head>** da página html, serão exibidas duas caixas de mensagens antes da apresentação da página. Após aberta a página, clique no botão **Exibir mensagem** que contém a chamada **inline** da função **exibeMensagem** que está contida no arquivo **externo js1.js**. Após clicar no botão será apresentada uma nova caixa de mensagem.

Verificação de erros

- Se em algum momento o **VS Code** não abrir o arquivo desejado, verifique se existe um arquivo de nome **launch.json** dentro de um subdiretório de nome **.vscode**. Caso exista, delete o arquivo, pois ele serve para configurar o arquivo/url que deve ser aberta. Esta configuração pode ter sido criada acidentalmente através da opção **Run -> Add Configuration...**

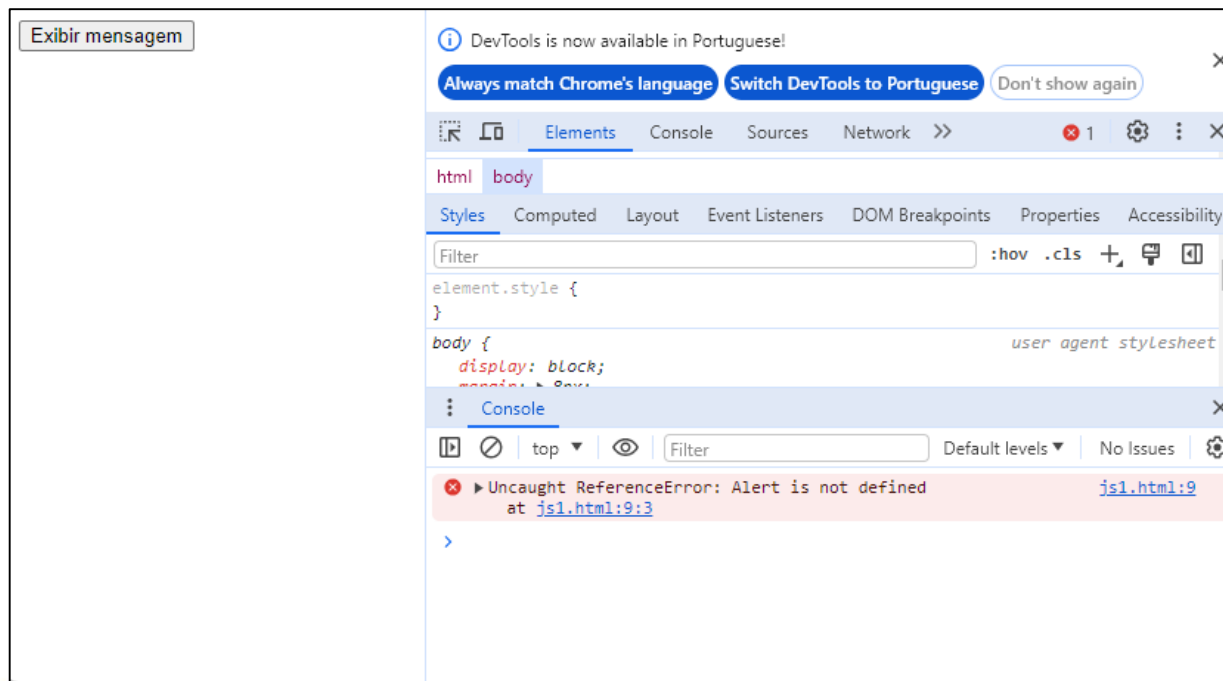
Verificação de erros

- Por conta do JavaScript ser *case sensitive*, ou seja, diferenciar letras maiúsculas de minúsculas, é muito comum cometer erros de digitação no código. Altere o arquivo **js1.html** de forma que a palavra **alert** fique com a primeira letra em maiúscula, conforme apresentado abaixo. Ao executar o arquivo, perceba que as caixas de diálogo iniciais não serão exibidas.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Uso de caixa de alerta</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js1.js"></script>
8      <script type="text/javascript">
9          Alert("Seja Bem-vindo(a)!");
10         confirm("Deseja continuar?");
11     </script>
12 </head>
13
14 <body>
15     <button type="button" onclick="exibeMensagem()">Exibir mensagem</button>
16 </body>
17
18 </html>
```

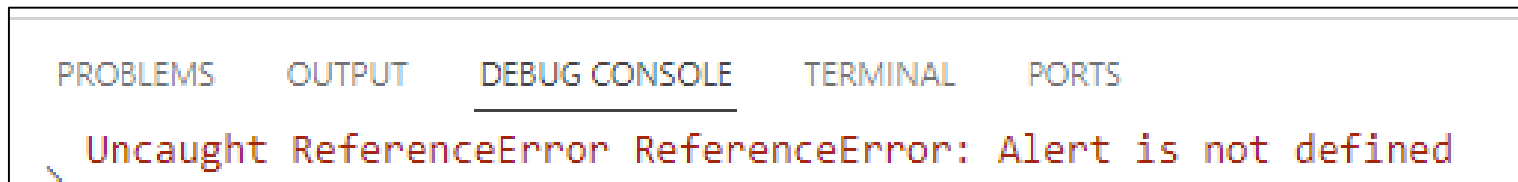
Verificação de erros

- Na janela do Chrome, clique nos 3 pontos na barra do navegador e clique na opção **Mais Ferramentas -> Ferramentas do desenvolvedor**. Podem ser também pressionadas as teclas **CTRL + SHIFT + I**. Na janela que será exibida, verifique o indicativo de erro contido na barra de menu (destacado em vermelho). Clique na parte inferior no nome do arquivo contendo o erro para ver o código. Ao final, feche a janela que foi aberta e o navegador.



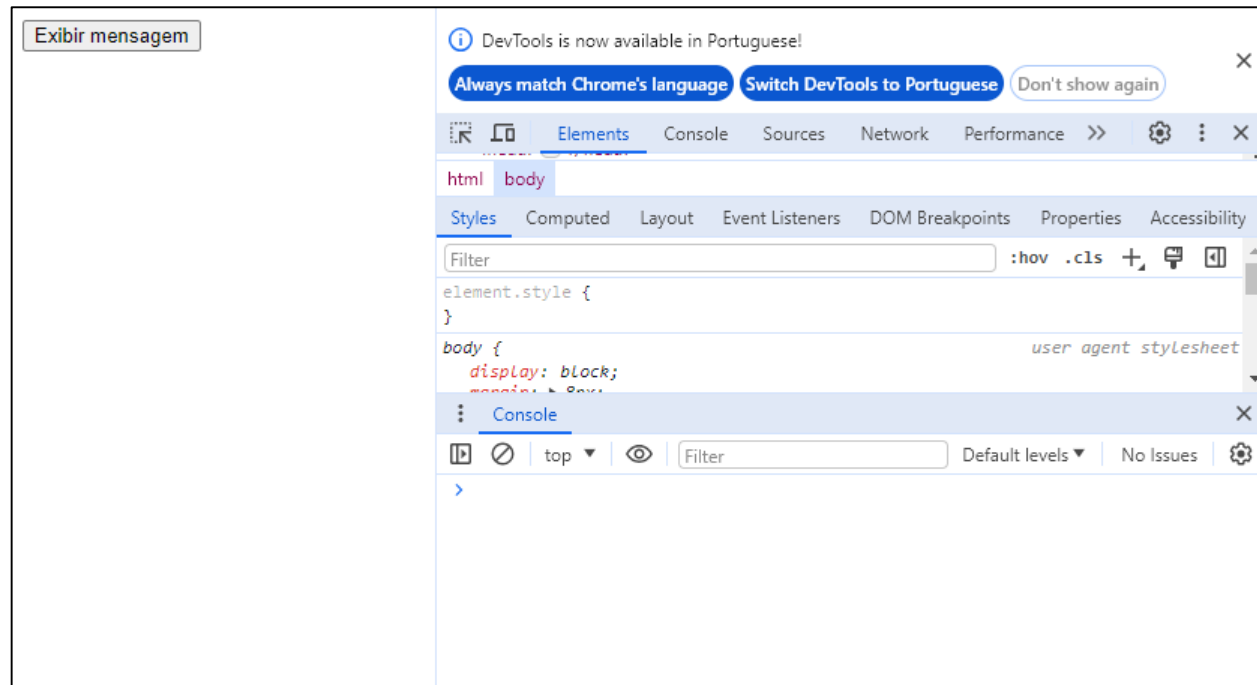
Verificação de erros

- O mesmo erro pode ser visualizado diretamente no VS Code na view **DEBUG CONSOLE** na parte inferior da janela.



Verificação de erros

- Corrija o erro e execute novamente o arquivo para confirmar que a página voltou a funcionar. Apresente novamente as ferramentas de desenvolvedor para confirmar que o indicativo de erro não será mais apresentado.



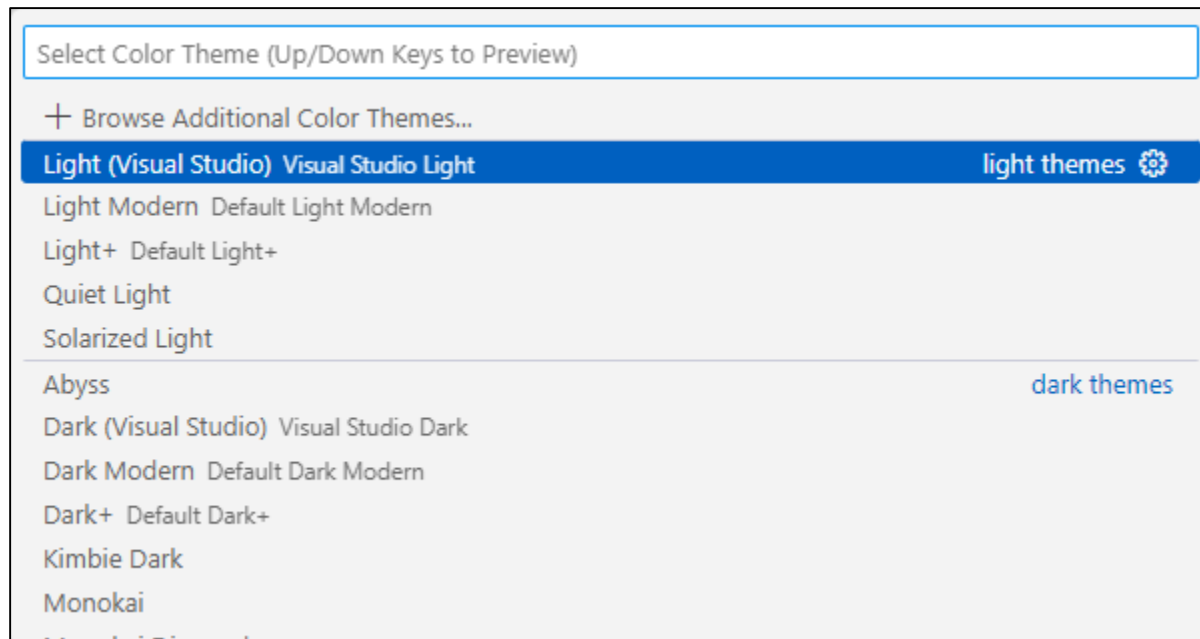
Formatação de código

- Manter um código bem formatado é uma boa prática de programação. Assim, para testar a formatação automática do **VS Code**, retire as tabulações das tags conforme apresentado abaixo. Depois clique com o botão direito do mouse e escolha a opção **Format Document** para formatar o documento. Podem ser utilizadas as teclas **SHIFT + ALT + F**.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5  <title>Uso de caixa de alerta</title>
6  <meta charset="utf-8">
7  <script type="text/javascript" src="js/js1.js"></script>
8  <script type="text/javascript">
9  alert("Seja Bem-vindo(a)!");
10 confirm("Deseja continuar?");
11 </script>
12 </head>
13
14 <body>
15 <button type="button" onclick="exibeMensagem()">Exibir mensagem</button>
16 </body>
17
18 </html>
```


Alteração de tema

- Uma das configurações normalmente feitas no VS Code é a alteração do tema da interface. Para alterá-la, basta clicar na opção **File -> Preferences -> Theme -> Color Theme**. Será apresentada a janela abaixo para escolha do tema. Altere o tema para verificar a alteração.



Exercícios:

- 1) O que é uma IDE?
- 2) Cite algumas IDEs que você conhece que são utilizadas para codificar programas em JavaScript?
- 3) Como pode ser executado um programa no VS Code?
- 4) Como pode ser verificado um erro em uma página através do VS Code?
- 5) Teste a formatação automática de código no VS Code.
- 6) Teste a alteração de tema no VS Code.

Conceitos básicos



Conceitos básicos



Utilização do ponto e vírgula

- Em JavaScript não é obrigatório o uso do ponto e vírgula ao final de cada instrução, mas como boa prática de programação deve ser sempre utilizado. Caso não seja informado o ponto e vírgula ao final de uma instrução, o JavaScript utilizará regras específicas para a inserção automática do ponto e vírgula, o que pode causar problemas no seu código. Para maiores informações consulte o link abaixo:
<https://262.ecma-international.org/13.0/#sec-automatic-semicolon-insertion>
- Altere o arquivo **js1.html** de forma a retirar o ponto e vírgula ao final das linhas 9 e 10. Execute novamente o arquivo para ver que o código continuará funcionando. Depois inclua novamente o ponto e vírgula no final das respectivas linhas.

Conceitos básicos



Utilização de aspas simples ou dupla

- Em JavaScript você pode utilizar aspas simples ou dupla para delimitar uma string. Se a string for aberta com aspas simples, deve ser fechada com aspas simples, e o mesmo vale para a aspas dupla. Aspas simples pode ser utilizada dentro de uma string delimitada com aspas dupla, assim como a aspas dupla pode ser utilizada dentro de uma string delimitada com aspas simples. Altere as linhas 9 e 10 do arquivo **js1.html** conforme apresentado abaixo. Execute o arquivo novamente para confirmar que está funcionando.

```
js1.html x
js1.html > html > head > script
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <title>Uso de caixa de alerta</title>
6   <meta charset="utf-8">
7   <script type="text/javascript" src="js/js1.js"></script>
8   <script type="text/javascript">
9     alert("Seja Bem-vindo(a)!");
10    confirm("Deseja continuar?");
11  </script>
12 </head>
13
14 <body>
15   <button type="button" onclick="exibeMensagem()">Exibir mensagem</button>
16 </body>
17
18 </html>
```

Conceitos básicos



Comentários

- Se o comentário estiver em uma única linha, insira os símbolos // no início da linha onde estiver o comentário.
- Se o comentário estiver em várias linhas, insira o símbolo /* no início da primeira linha onde iniciar o comentário e o símbolo */ no final do comentário.
- No VS Code, Pode ser utilizada a opção **Toggle Line Comment (CTRL + ;)** para comentários de uma linha e a opção **Toggle Block Comment (SHIFT + ALT + A)** para comentários de várias linhas. Não esqueça de selecionar as linhas quando o objetivo for comentar várias linhas.

Conceitos básicos



Comentários

- Insira os comentários no código conforme apresentado abaixo. Após executar o arquivo para confirmar que as caixas de mensagens não serão exibidas, retire os comentários das linhas 12 e 13 para que as caixas de diálogo voltem a ser exibidas.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Uso de caixa de alerta</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js1.js"></script>
8      <script type="text/javascript">
9          /*
10             Código em JavaScript
11          */
12          // alert("Seja 'Bem-vindo(a)!");
13          // confirm('Deseja continuar?');
14      </script>
15  </head>
16
17  <body>
18      <button type="button" onclick="exibeMensagem()">Exibir mensagem</button>
19  </body>
20
21  </html>
```

Conceitos básicos



Tipos primitivos de dados

- Em JavaScript são utilizados os seguintes tipos primitivos de dados:
 - ☐ **string**: Para armazenar String. Os valores são delimitados por aspas dupla ou simples.
 - ☐ **number**: Para armazenar números.
 - ☐ **bigint**: Para armazenar números inteiros que não podem ser armazenados como number. Deve ser utilizada a letra n no final do número.
 - ☐ **boolean**: Para armazenar booleanos, ou seja, true ou false.
 - ☐ **undefined**: Para armazenar valores indefinidos.
 - ☐ **null**: Para armazenar nulo.
 - ☐ **symbol**: Para armazenar valores únicos e assim ocupar um único lugar na memória. Criado no ECMAScript 6 (ES6).
- Utilizaremos cada um destes tipos ao longo da nossa unidade curricular.

Conceitos básicos



Variáveis

- Um variável serve para armazenar informações temporárias que serão utilizadas ao longo do código.
- O escopo de uma variável determina onde o valor dela pode ser utilizado. O escopo pode ser: global (criada fora de uma função ou de um bloco { }), função (criada dentro de uma função) ou de bloco (criada dentro de um bloco { }).
- Variáveis com escopo global poderão ser acessadas em qualquer parte do código a partir da sua declaração inicial, desde que não sejam redeclaradas em outro escopo.
- Variáveis com escopo de função só podem ser acessadas dentro da própria função onde forem declaradas.
- Variáveis de escopo de bloco só podem ser acessadas dentro do bloco onde forem declaradas.

Conceitos básicos



Variáveis

- Uma variável pode ser criada (declarada) utilizando os seguintes comandos:
 - ❑ **var**: a variável terá um escopo global, mesmo se declarada dentro de um bloco, e poderá ser utilizada antes de ter sido declarada (característica conhecida como **hoisting** - elevar), porque as declarações das variáveis com este comando contidas no código são implicitamente executadas antes de iniciar a execução do código, ou seja, são elevadas ao topo do escopo, sendo inicializadas neste caso com o valor **undefined**. A variável poderá ter o valor modificado dentro do seu escopo. É um comando antigo em JavaScript e foi mantido por questão de compatibilidade com versões anteriores. A variável pode ser redeclarada dentro do mesmo escopo. **Deve-se evitar usar este comando.**
 - ❑ **let**: a variável terá um escopo de bloco se criada dentro de um bloco e global se criada fora de uma função ou bloco. A variável poderá ter o valor modificado dentro do seu escopo. A variável não pode ser redeclarada dentro do mesmo escopo. A variável só pode ser acessada depois que for declarada.
 - ❑ **const**: a variável terá um escopo de bloco se criada dentro de um bloco e global se criada fora de uma função ou bloco. É obrigatória a atribuição de valor no momento da declaração da variável. Entretanto, seu valor será uma constante, ou seja, não poderá ser modificado dentro do seu escopo. A variável não pode ser redeclarada dentro do mesmo escopo. A variável só pode ser acessada depois que for declarada.

Conceitos básicos



Variáveis

- Variáveis declaradas com **let** e **const** também são elevadas ao topo do escopo (**hoisting**), mas como neste caso não são inicializadas, causam erro se tentar utilizá-las antes de terem sido inicializadas.
- Qualquer variável criada dentro de uma função com um destes comandos terá o escopo de função. Funções também são elevadas ao topo do escopo (**hoisting**), podendo ser inseridas mesmo depois de onde estão sendo utilizadas.
- O tipo da variável não é definido na sua declaração. O tipo é definido com base no valor que for atribuído a ela. Por esta razão dizemos que JavaScript não é uma linguagem fortemente tipada.
- Se não for atribuído um valor a uma variável declarada com **var** ou **let**, o valor dela será **undefined** e seu tipo será **undefined**.
- Os comandos **let** e **const** foram implementados a partir do ECMAScript 6 (ES6).

Conceitos básicos



Variáveis

- A declaração de variáveis segue a seguinte sintaxe:

[var | let | const] <nome da variável> [= <valor>];

- Os símbolos [] indicam que é opcional informar os comandos e o símbolo | significa que podem ser informados um destes comandos. Caso não seja informado nenhum comando, a variável só será reconhecida quando da atribuição de valores a ela e, a partir deste momento, o escopo dela será global. Entretanto, só é possível omitir um destes comandos caso não seja utilizado o modo estrito do JavaScript através do uso da diretiva "**use strict**". O modo estrito elimina algumas funcionalidades do JavaScript que podem ser usadas de maneira insegura. Consulte o link abaixo para maiores informações sobre o modo estrito:
https://www.w3schools.com/js/js_strict.asp
- Os símbolos < > indicam que o nome da variável deve ser informado e o valor dela apenas se for informado o sinal =.
- O nome da variável não deve:
 - ☐ Conter espaços;
 - ☐ Começar por número;
 - ☐ Conter caracteres especiais, tais como: +, ,, %, #, etc.
 - ☐ Não devem utilizar palavras reservadas da linguagem.
- ☐ Evite nomes de variáveis que não explicam o seu conteúdo. Então, em vez de criar uma variável **id** para armazenar a idade de uma pessoa, prefira uma variável chamada **idade**.
- ☐ Por convenção, os nomes das variáveis declaradas com **const** devem ser mantidas em letras maiúsculas, como por exemplo, **LIMITE_VALOR**. Os nomes das variáveis declaradas com **var** e **let** devem ser mantidas em letras minúsculas e, se for uma palavra composta, mantenha as iniciais de cada palavra em letras maiúsculas, com exceção da primeira palavra, como por exemplo, **salarioLiquido**.
- ☐ Não esqueça que uma variável chamada **nomealuno** é diferente de uma variável chamada **nomeAluno**.
- ☐ Não esqueça que uma variável declarada com o comando **const** terá que ser obrigatoriamente inicializada com algum valor.

Conceitos básicos








Variáveis

- Segue uma tabela com as diferenças entre var, let e const

Comando	Escopo de bloco	Redeclaração no mesmo escopo	Elevação de escopo completa	Alteração de valor
var		X	X	X
let	X			X
const	X			

- Segue a partir de que versão os navegadores passaram a suportar **let** e **const**. Elas não são suportadas no Internet Explorer 11 ou versões anteriores.

				
Chrome 49	Edge 12	Firefox 36	Safari 11	Opera 36
Mar, 2016	Jul, 2015	Jan, 2015	Sep, 2017	Mar, 2016

Conceitos básicos



Variáveis

- Crie o arquivo **js2.html** conforme apresentado abaixo para que possamos testar a declaração e escopo de variáveis. O operador **typeof** é utilizado para exibir o tipo de uma variável.

```
<> js2.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Uso de variáveis</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js2.js"></script>
8      <script type="text/javascript">
9          alert("O valor da variável a é " + a + " e seu tipo é " + typeof a);
10         var a = "Maria"; // Escopo global
11         var a = 10; // Redecaração de variável no mesmo escopo
12         let b; // Escopo global
13         alert(b);
14         // const b; // Não aceita redeclaração de variável no mesmo escopo
15         const c = true; // Escopo global
16         // let c = false; // Não aceita redeclaração de variável no mesmo escopo
17     </script>
18 </head>
19
20 <body>
21     <button type="button" onclick="exibirValorA()">Exibir valor global da variável a</button><br>
22     <button type="button" onclick="alterarValorA()">Alterar valor da variável a</button><br>
23     <br>
24     <button type="button" onclick="exibirValorB()">Exibir valor global da variável b</button><br>
25     <button type="button" onclick="alterarValorB()">Alterar valor da variável b</button><br>
26     <br>
27     <button type="button" onclick="exibirValorC()">Exibir valor global da variável c</button><br>
28     <button type="button" onclick="alterarValorC()">Alterar valor da variável c</button><br>
29 </body>
30
31 </html>
```

Conceitos básicos



Variáveis

- Crie também o arquivo **js2.js** que contém o código em JavaScript das funções chamadas no arquivo **js2.html**. Ao terminar, execute o arquivo **js2.html** para ver o resultado. Clique inicialmente nos botões de exibir o valor das variáveis e note que serão apresentados os valores globais das variáveis. Veja que a utilização dos botões para alterar o valor das variáveis **a** e **c** modificará o valor destas variáveis apenas dentro da função, por conta que as respectivas variáveis foram redeclaradas no escopo da função, permanecendo assim o valor global inalterado. Entretanto, ao clicar no botão para alteração do valor da variável **b**, a variável terá o seu valor modificado no escopo global, o que pode ser verificado ao clicar no botão de exibição do valor dela.

```
js > JS js2.js > alterarValorC
1  function exibirValorA() {
2      alert("O valor da variável a é " + a + " e seu tipo é " + typeof a);
3  }
4  function alterarValorA() {
5      var a = "João"; // Redecaração no escopo de função
6      alert("Valor da variável a foi alterado localmente para " + a);
7  }
8  function exibirValorB() {
9      alert("O valor da variável b é " + b + " e seu tipo é " + typeof b);
10 }
11 function alterarValorB() {
12     b = 30; // Alteração do valor no escopo global
13     alert("Valor da variável b foi alterado globalmente para " + b);
14 }
15 function exibirValorC() {
16     alert("O valor da variável c é " + c + " e seu tipo é " + typeof c);
17 }
18 function alterarValorC() {
19     const c = 10; // Redecaração no escopo de função
20     alert("Valor da variável c foi alterado localmente para " + c);
21 }
```

Conceitos básicos



Variáveis

- Altere o arquivo **js2.html** de forma a retirar os comentários das linhas 13 e 15. Veja que dará erro por conta da tentativa de redeclaração das variáveis b e c no mesmo escopo. Depois de verificar o erro, comente novamente as linhas.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Uso de variáveis</title>
6    <meta charset="utf-8">
7    <script type="text/javascript" src="js/js2.js"></script>
8    <script type="text/javascript">
9      alert 'const' declarations must be initialized. javascript eof a);
10     var a;
11     var a;
12     let b;
13     const b; // Não aceita redeclaração de variável no mesmo escopo
14     const c = true; // Escopo global
15     let c = false; // Não aceita redeclaração de variável no mesmo escopo
16   </script>
17 </head>
18
19 <body>
20   <button type="button" onclick="exibirValorA()">Exibir valor global da variável a</button><br/>
21   <button type="button" onclick="alterarValorA()">Alterar valor da variável a</button><br/>
22   <br/>
23   <button type="button" onclick="exibirValorB()">Exibir valor global da variável b</button><br/>
24   <button type="button" onclick="alterarValorB()">Alterar valor da variável b</button><br/>
25   <br/>
26   <button type="button" onclick="exibirValorC()">Exibir valor global da variável c</button><br/>
27   <button type="button" onclick="alterarValorC()">Alterar valor da variável c</button><br/>
28 </body>
29
30 </html>
```


Conceitos básicos



Template String

- É uma forma de apresentar strings concatenadas sem a necessidade de utilizar o operador +. Neste caso, as strings devem ser delimitadas através do sinal de crase (`) e as expressões variáveis incluídas no formato `${<expressão>}`.
- Altere no arquivo **js2.html** a instrução com **alert** conforme apresentado abaixo.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Uso de variáveis</title>
6    <meta charset="utf-8">
7    <script type="text/javascript" src="js/js2.js"></script>
8    <script type="text/javascript">
9      alert(`O valor da variável a é ${a} e seu tipo é ${typeof a}`);
10     var a = "Maria"; // Escopo global
11     var a = 10; // Redecaração de variável no mesmo escopo
12     let b; // Escopo global
13     // const b; // Não aceita redeclaração de variável no mesmo escopo
14     const c = true; // Escopo global
15     // let c = false; // Não aceita redeclaração de variável no mesmo escopo
16   </script>
17 </head>
18
19 <body>
20   <button type="button" onclick="exibirValorA()">Exibir valor global da variável a</button><br/>
21   <button type="button" onclick="alterarValorA()">Alterar valor da variável a</button><br/>
22   <br/>
23   <button type="button" onclick="exibirValorB()">Exibir valor global da variável b</button><br/>
24   <button type="button" onclick="alterarValorB()">Alterar valor da variável b</button><br/>
25   <br/>
26   <button type="button" onclick="exibirValorC()">Exibir valor global da variável c</button><br/>
27   <button type="button" onclick="alterarValorC()">Alterar valor da variável c</button><br/>
28 </body>
29
30 </html>
```

Conceitos básicos



Template String

- Agora altere no arquivo **js2.js** as instruções com **alert** das linhas 6, 13 e 20, conforme apresentado abaixo. Após a alteração, execute o arquivo **js2.html** para confirmar que o resultado será o mesmo.

```
js > JS js2.js > alterarValorC
1  function exibirValorA() {
2      alert("O valor da variável a é " + a + " e seu tipo é " + typeof a);
3  }
4  function alterarValorA() {
5      var a = "João"; // Redeclaração no escopo de função
6      alert(`Valor da variável a foi alterado localmente para ${a}`);
7  }
8  function exibirValorB() {
9      alert("O valor da variável b é " + b + " e seu tipo é " + typeof b);
10 }
11 function alterarValorB() {
12     b = 30; // Alteração do valor no escopo global
13     alert(`Valor da variável b foi alterado globalmente para ${b}`);
14 }
15 function exibirValorC() {
16     alert("O valor da variável c é " + c + " e seu tipo é " + typeof c);
17 }
18 function alterarValorC() {
19     const c = 10; // Redeclaração no escopo de função
20     alert(`Valor da variável c foi alterado localmente para ${c}`);
21 }
```

Conceitos básicos



Operadores matemáticos

- No JavaScript existem os seguintes operadores matemáticos:
 - ❑ **Soma:** Operador +
let a = 1;
a = a + 1; // 2
 - ❑ **Subtração:** Operador –
let a = 2;
a = a - 1; // 1
 - ❑ **Multiplicação:** Operador *
let a = 5;
a = a * 2; // 10
 - ❑ **Divisão:** Operador /
let a = 10;
a = a / 2; // 5
 - ❑ **Resto:** Operador %
let a = 10;
a = a % 2; // 0
 - ❑ **Exponenciação:** Operador **
let a = 10;
a = a ** 2; // 100

Conceitos básicos



Operadores matemáticos

- Crie o arquivo **js3.html** conforme apresentado abaixo e execute-o para ver o resultado.

```
<> js3.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Operadores matemáticos</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let a = 10;
12         a = 10 ** 2;
13         alert(a); // 100
14         a = a - 10;
15         alert(a); // 90
16         a = a / 3;
17         alert(a); // 30
18         a = a * 2;
19         alert(a); // 60
20         a = a + 20;
21         alert(a); // 80
22         a = a % 4;
23         alert(a); // 0
24     </script>
25 </body>
26
27 </html>
```

Conceitos básicos



Operadores de incremento e decremento

- Para incrementar 1 a uma variável pode ser utilizado o operador `++`. Então, ambas operações abaixo possuem o mesmo resultado.

`x = x + 1;`

`x += 1;`

`x++;`

- Para decrementar 1 de uma variável pode ser utilizado o operador `--`. Então, ambas operações abaixo possuem o mesmo resultado.

`x = x - 1;`

`x -= 1;`

`x--;`

Conceitos básicos



Operadores de incremento e decremento

- Os operadores ++ e -- podem ser utilizados de forma pós-fixada, quando utilizado após a variável (ex: x++, x--), ou pré-fixado, quando é utilizado antes da variável (ex.: ++x, --x).
- Quando são utilizados de forma pré-fixada, a variável é incrementada ou decrementada **ANTES** da execução da instrução.
- Quando são utilizados de forma pós-fixada, a variável é incrementada ou decrementada **APÓS** a execução da instrução.

Conceitos básicos



Operadores matemáticos

- Crie o arquivo **js4.html** conforme apresentado abaixo e execute-o para ver o resultado.

```
<> js4.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Operadores de incremento e decremento</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let x = 5;
12         // Pré-fixado
13         alert(++x); // 6
14         alert(--x); // 5
15         // Pós-fixado
16         alert(x++); // 5
17         alert(x); // 6;
18         alert(x--); // 6
19         alert(x); // 5
20     </script>
21 </body>
22
23 </html>
```

Conceitos básicos



Conversão de tipos

- Lembre-se que em JavaScript as variáveis podem ter os seus tipos alterados para outros tipos no decorrer de um programa, mas isso deve ser evitado. Existem dois tipos de conversão em JavaScript:

- ❑ **Conversões implícitas:** Conversão feita diretamente pela linguagem. Ocorre por exemplo na soma entre string e número, onde o número é convertido para string e o resultado final é a concatenação das strings. Nas demais operações matemáticas entre string e número, a string será convertida automaticamente para número. Outro exemplo são as operações matemáticas entre booleano e número, onde o booleano é convertido para número.

Ex:

```
let a = "10";  
a = a + 2; // a="102"  
let b = a * 2; // b=204  
let c = true;  
let d = c * 2; // d=2
```

- ❑ **Conversões explícitas:** É a conversão de tipos definida pelo programador. Para isso é necessário utilizar métodos globais para fazer a conversão. Métodos globais são como funções já disponíveis na linguagem. Seguem alguns exemplos de métodos globais:

- **String(<número>):** Converte um número para string.
- **String(<booleano>):** Converte um booleano para string.
- **Number(<string>):** Converte uma string para número. Retorna NaN (Not-A-Number) se não for possível a conversão.
- **Number(<booleano>):** Converte um booleano para número (0 ou 1). Retorna 1 se for passado true e 0 se passado false.
- **Boolean(<número>):** Converte um número para booleano (true ou false). O número 0 (zero) é convertido para false e qualquer outro número é convertido para true.
- **Boolean(<string>):** Converte uma string para booleano (true ou false). Uma string vazia é convertida para false e preenchida para true.

Ex:

```
Number("false"); // 0  
Boolean("0"); // true  
String(20) + 3; // 203
```


Conceitos básicos



Conversão de tipos

- Crie o arquivo **js5.html** conforme apresentado abaixo e execute-o para ver o resultado.

```
> js5.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Conversão de tipos</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     let a = "10";
12     let b = 1;
13     let c = true;
14     // Conversão implícita
15     alert(a + b); // 101
16     alert(a - b); // 9
17     alert(a * b); // 10
18     alert(a / b); // 10
19     alert(b + c); // 2
20     // Conversão explícita
21     alert(Number(a) + b); // 11
22     alert(Number(c) + 1); // 2
23     c = String(b);
24     alert("O valor da variável c é " + c + " e seu tipo é " + typeof c); // 1 e string;
25     let d = Boolean(b);
26     alert("O valor da variável d é " + d + " e seu tipo é " + typeof d); // true e boolean;
27     let e = Number("xxx");
28     alert(e); // NaN
29   </script>
30 </body>
31
32 </html>
```

Conceitos básicos



Entrada de dados

- Uma das formas de entrada de dados é através do método global **prompt**. O método apresentará uma caixa de diálogo para entrada de dados, onde pode ser informada uma resposta padrão. O resultado do método é uma string com a resposta preenchida. A sintaxe do comando é:
prompt(<mensagem da caixa de mensagem>[, <resposta padrão>])
- Crie o arquivo **js6.html** conforme apresentado abaixo e execute-o para ver o resultado. Veja que foi utilizado o método global **Number** para converter o resultado da entrada das notas de string para número. Utilize ponto como casas decimais. Depois de ver o resultado, execute novamente o arquivo e informe um texto nas notas. Veja que o método **Number** não conseguirá fazer a conversão das notas para número e o resultado da média será igual a **NaN (Not-a-Number)**.

```
> js6.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Entrada de dados</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let nome = prompt("Informe o nome do aluno:");
12         let nota1 = prompt("Informe a primeira nota do aluno:");
13         let nota2 = prompt("Informe a segunda nota do aluno:");
14         let media = (Number(nota1) + Number(nota2)) / 2;
15         alert(`O aluno ${nome} teve média de ${media}`);
16     </script>
17 </body>
18
19 </html>
```

Conceitos básicos



Entrada de dados

- Agora altere o arquivo **js6.html** de forma a retirar o método **Number** conforme apresentado abaixo e execute-o novamente para ver o resultado. Veja que as duas notas serão concatenadas e depois convertidas para número para a realização da média. Depois da execução do arquivo, retorne com o método **Number**.


```
> js6.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Entrada de dados</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let nome = prompt("Informe o nome do aluno:");
12         let nota1 = prompt("Informe a primeira nota do aluno:");
13         let nota2 = prompt("Informe a segunda nota do aluno:");
14         let media = (nota1 + nota2) / 2;
15         alert(`O aluno ${nome} teve média de ${media}`);
16     </script>
17 </body>
18
19 </html>
```

Conceitos básicos



Entrada de dados

- Outra forma de entrada de dados é através do método global **confirm**. O método apresentará uma caixa de diálogo com os botões **OK** e **Cancelar**. O resultado do comando será **true** se for clicado no botão **OK** e **false** se for clicado no botão **Cancelar**. A sintaxe do comando é: **confirm(<mensagem da caixa de mensagem>)**
- Crie o arquivo **js7.html** conforme apresentado abaixo e execute-o para ver o resultado.

```
> js7.html >  html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Entrada de dados</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let resposta = confirm("Deseja continuar (s/n)?");
12         alert(resposta);
13     </script>
14 </body>
15
16 </html>
```

Conceitos básicos



Exercícios:

- 1) Como são as formas de criação de comentários em JavaScript?
- 2) Quais são os tipos primitivos em JavaScript?
- 3) Crie um arquivo **js7-ex3.html** que utilize os comandos `var`, `let` e `const` em diferentes escopos. Execute o arquivo para ver o resultado.
- 4) Crie um arquivo **js7-ex4.html** que realize operações matemáticas. Execute o arquivo para ver o resultado.
- 5) Crie um arquivo **js7-ex5.html** que realize operações de incremento e decremento. Execute o arquivo para ver o resultado.
- 6) Crie um arquivo **js7-ex6.html** que realize conversões implícitas e explícitas de tipos. Execute o arquivo para ver o resultado.
- 7) Crie um arquivo **js7-ex7.html** que entre com o nome de um cliente, o total da conta, o valor pago e apresente o troco a ser recebido pelo cliente.

Estruturas condicionais



Estruturas condicionais



Operadores relacionais

Operador	Significado
==	Igual a (não considerando os tipos)
===	Estritamente igual a (considerando os tipos)
!=	Diferente de (não considerando os tipos)
!==	Estritamente diferente de (considerando os tipos)
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a

Estruturas condicionais



Operadores relacionais

- Crie o arquivo **js8.html** conforme apresentado abaixo e execute-o para ver o resultado.

```
<> js8.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Operadores relacionais</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let a = 1;
12         let b = 2;
13         let c = '1';
14         alert("a=" + a); // a=1
15         alert("b=" + b); // b=2
16         alert("c=" + c); // c='1'
17         alert("a==b -> " + (a==b)); // false
18         alert("a==c -> " + (a==c)); // true
19         alert("a===c -> " + (a===c)); // false
20         alert("a!=b -> " + (a!=b)); // true
21         alert("a!=c -> " + (a!=c)); // false
22         alert("a!==c -> " + (a!==c)); // true
23         alert("b>a -> " + (b>a)); // true
24         alert("a>b -> " + (a>b)); // false
25         alert("a<b -> " + (a<b)); // true
26         alert("b<=a -> " + (b<=a)); // false;
27     </script>
28 </body>
29
30 </html>
```


Estruturas condicionais



Operadores lógicos

Operador	Significado	Funcionamento
&&	E (AND)	A condição só será verdadeira se as expressões em comparação forem verdadeiras.
	Ou (OR)	A condição será verdadeira se pelo menos uma das expressões em comparação for verdadeira.
!	Não (NOT)	Se a expressão for verdadeira, o resultado da condição será falso. Se o a expressão for falsa, o resultado da condição será verdadeiro.

- Existem também os operadores de comparação bit a bit (bitwise). Maiores informações no link abaixo:
https://www.w3schools.com/js/js_bitwise.asp

Estruturas condicionais



Operadores lógicos

- Crie o arquivo **js9.html** conforme apresentado abaixo e execute-o para ver o resultado.

```
<> js9.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Operadores lógicos</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let a = 1;
12         let b = 2;
13         let c = '1';
14         alert("b>a E a==c -> " + (b>a&&a==c)); // true
15         alert("b>a E a==b -> " + (b>a&&a==b)); // false
16         alert("b>a OU a<=b -> " + (b>a||a<=b)); // true
17         alert(!(a==b)); // true
18     </script>
19 </body>
20
21 </html>
```

Estruturas condicionais



if / else

- Estrutura condicional que verifica uma condição lógica (booleana). Se a condição for verdadeira, executa o bloco contido na instrução **if** e, se a condição for falsa, executa o bloco contido na instrução **else**. A instrução **else** (e seu respectivo bloco) é opcional. Segue a sintaxe do comando:

```
if (<condição>) {  
    <instruções a serem executadas se a condição verdadeira>  
} else {  
    <instruções a serem executadas se a condição falsa>  
}
```

- As chaves são opcionais se houver apenas uma instrução dentro de um bloco do **if** ou do **else**.

Estruturas condicionais



if / else

- Crie o arquivo **js10.html** conforme apresentado abaixo e execute-o para ver o resultado. Veja que a variável **anosFaltando** foi declarada em escopo global e redeclarada em escopo de bloco. O valor dela dentro do bloco só existe no bloco, assim, o último comando **alert** apresentará 0 como resultado.

```
<> js10.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Uso do If / Else</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         const LIMITE_ADULTO = 18; // escopo global
12         const LIMITE_VOTAR = 16; // escopo global
13         let nome = prompt("Informe o seu nome:"); // escopo global
14         let idade = Number(prompt("Informe a sua idade:")); // escopo global
15         let anosFaltando = 0; // escopo global;
16         if (idade>=LIMITE_ADULTO)
17             alert(`${nome} você é um(a) adulto(a)`);
18         else
19             alert(`${nome} você ainda não é um(a) adulto(a)`);
20         if (idade>=LIMITE_VOTAR) {
21             alert("Você já pode votar.");
22             alert("Parabéns!");
23         } else {
24             alert("Você ainda não pode votar.");
25             let anosFaltando = LIMITE_VOTAR - idade; // escopo de bloco - redeclaração em outro escopo
26             alert(`Ainda faltam ${anosFaltando} anos.`);
27         }
28         alert(anosFaltando); // 0 - escopo global
29     </script>
30 </body>
31
32 </html>
```

Estruturas condicionais



if / else if / else

- Estrutura condicional que verifica várias condições lógicas (booleanas). Se a condição1 for verdadeira, executa o bloco contido na instrução **if**. Se a condição1 for falsa, verifica as condições contidas nos blocos **else if**, caso alguma delas for verdadeira executa o seu respectivo bloco de instruções. Se nenhuma condição for verdadeira, executa o bloco contido na instrução **else**. A instrução **else** (e seu respectivo bloco) é opcional. Segue a sintaxe do comando:

```
if (<condição1>) {  
    <instruções a serem executadas se a condição verdadeira>  
} else if (<condição2>) {  
    <instruções a serem executadas se a condição2 for verdadeira>  
} else if (<condiçãon>) {  
    <instruções a serem executadas se a condição for verdadeira>  
} else {  
    <instruções a serem executadas se nenhuma das condições forem verdadeiras>  
}
```

- As chaves são opcionais se houver apenas uma instrução dentro de um bloco do **if**, **else if** ou do **else**.

Estruturas condicionais



if / else if / else

- Abra o arquivo **js6.html** e salve com o nome **js11.html**. Inclua agora o código do comando **if** conforme apresentado abaixo. Execute o arquivo para ver o resultado.

```
<> js11.html > html > body > script
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Entrada de dados</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let nome = prompt("Informe o nome do aluno:");
12         let nota1 = prompt("Informe a primeira nota do aluno:");
13         let nota2 = prompt("Informe a segunda nota do aluno:");
14         let media = (Number(nota1) + Number(nota2)) / 2;
15         alert(`O aluno ${nome} teve média de ${media}`);
16         if (media >= 7) {
17             alert("Aprovado!");
18         } else if (media >= 5) {
19             alert("Recuperação");
20         } else {
21             alert("Reprovado");
22         }
23     </script>
24 </body>
25
26 </html>
```

Estruturas condicionais



switch

- Estrutura condicional que compara o valor de uma variável aos valores contidos nas cláusulas **case**. Caso ela seja igual a um dos valores contidos em uma das cláusulas **case**, executará o respectivo bloco de instruções do **case**. É necessário incluir a instrução **break** para que não sejam executadas as demais cláusulas **Case**. Se nenhuma condição for verdadeira, executa o bloco contido na instrução **default**. Segue a sintaxe básica do comando:

```
switch (expressão) {  
    case valor1:  
        <instruções a serem executadas se variável igual a valor1>;  
        break;  
    case valor2:  
        <instruções a serem executadas se variável igual a valor2>;  
        break;  
    case valorN:  
        <instruções a serem executadas se variável igual a valorN>;  
        break;  
    default:  
        <instruções a serem executadas se variável diferente de todos os valores>  
        break;  
}
```

Estruturas condicionais



switch

- Crie o arquivo **js12.html** conforme apresentado abaixo. Veja que no primeiro método **alert** foi utilizado o símbolo **\n** para que as opções do menu fiquem em linhas diferentes e nos demais foi utilizado a sequência de caracteres **** para inserir aspas dentro de uma string delimitada por aspas. Os caracteres **\n** e **** são o que chamamos de sequência de escape e é utilizada para apresentar caracteres que não seriam possíveis de serem inseridos de forma normal dentro de uma string. Além disso, veja que ao digitar o comando **switch** no **VS Code** será apresentada uma lista de opções, selecione a segunda opção que contém a declaração (*statement*) **switch**. Execute o arquivo para ver o resultado.

```
<> js12.html > html > body > script
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Menu</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     alert("Menu Principal\n1 - Depositar\n2 - Sacar\n3 - Transferir")
12     let op = prompt("Entre com a opção desejada:");
13     switch
14   </script>
15   switch
16 </html>
```

switch
switch Statement
SVGSwitchElement

```
<> js12.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Menu</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     alert("Menu Principal\n1 - Depositar\n2 - Sacar\n3 - Transferir")
12     let op = prompt("Entre com a opção desejada:");
13     switch (op) {
14       case "1":
15         alert("Você escolheu a opção \"1 - Depositar\"");
16         break;
17       case "2":
18         alert("Você escolheu a opção \"2 - Sacar\"");
19         break;
20       case "3":
21         alert("Você escolheu a opção \"3 - Transferir\"");
22         break;
23       default:
24         alert("Você não escolheu nenhuma das opções listadas!");
25         break;
26     }
27   </script>
28 </body>
29
30 </html>
```


Estruturas condicionais



Sequências de escape

- Seguem alguns exemplos:

Sequência de escape	Finalidade
<code>\t</code>	Inserir um tab.
<code>\n</code>	Inserir um novo caractere de linha (ENTER).
<code>\'</code>	Inserir uma aspa simples.
<code>\"</code>	Inserir uma aspa dupla.
<code>\\</code>	Inserir uma barra invertida.

- Para maiores informações consulte o link abaixo:
<https://262.ecma-international.org/5.1/#sec-7.8.4>

Estruturas condicionais



Operador ternário

- Recurso similar ao If / else que permite a criação de estruturas condicionais em apenas uma linha de código. Pode ser utilizado para inserir conteúdo dentro de outros comandos a partir do resultado de uma condição.

<condição> ? <conteúdo se a condição verdadeira> : <conteúdo se condição falsa>

- Crie o arquivo **js13.html** com o conteúdo abaixo e execute-o para ver o resultado.

```
<> js13.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Operador ternário</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         const LIMITE_ADULTO = 18;
12         const LIMITE_IDOSO = 60;
13         let idade = prompt("Entre com a sua idade:");
14         let adulto = (idade>=LIMITE_ADULTO) ? true : false;
15         adulto ? alert("Você é adulto!") : alert("Você não é adulto!");
16         (idade>=LIMITE_IDOSO) ? alert("Você é idoso!") : alert("Você não é idoso");
17     </script>
18
19 </html>
```

Estruturas condicionais



Exercícios:

- 1) Crie o arquivo **js14-ex1.html** que receba a velocidade de um carro (km/h) ao passar por um pardal e, com base na tabela abaixo, apresente o tipo de multa aplicável. Além disso, através do uso de operador ternário, informe se o motorista perderá ou não o direito de dirigir. O limite de velocidade no local é de 80 km/h.

Velocidade	Multa	Perde direito de dirigir
Até limite	Sem multa	Não
Até 20% acima do limite	Leve	Não
Mais de 20% acima do limite	Grave	Sim

Estruturas de repetição



Estruturas de repetição



for

- Comando que permite repetir instruções com base no valor de uma variável. Para isso é determinado o valor inicial, o valor final e o valor que será incrementado ou decrementado na variável a cada repetição (loop). Segue a sintaxe do comando:

```
for (<inicialização de variável>; <condição>; <incremento ou decremento>) {  
    <instruções a serem executadas>  
}
```

- As variáveis declaradas dentro do bloco não estarão disponíveis ao final dele, com exceção de quando declaradas com o comando var.

Estruturas de repetição



for

- Crie o arquivo **js15.html** com o conteúdo abaixo e execute-o para ver o resultado. Veja que a linha 14 foi comentada porque a variável *x* não existe no escopo global. A variável *y* até existe na linha 19, mas apenas no escopo global, visto que a declaração da variável na linha 16 faz com que a alteração dos valores dela só aconteça no escopo de bloco. Entretanto, a variável *z* existe na linha 24, mesmo tendo sido declarada no bloco, visto que, conforme verificamos anteriormente, variáveis declaradas com o comando *var* em bloco possuem escopo global.

```
<? js15.html > <? html>
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Estrutura de repetição - for</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     for (let x=1; x<=3; x++) {
12       alert("O valor da variável x é " + x); // escopo de bloco
13     }
14     // alert("Valor final de x é " + x); // a variável x não existe no escopo global
15     let y; // escopo global
16     for (let y=3; y>=1; y--) {
17       alert("O valor da variável y é " + y); // escopo de bloco
18     }
19     alert("Valor final de y é " + y); // undefined - escopo global
20     var z; // escopo global
21     for (var z=3; z>=1; z--) {
22       alert("O valor da variável z é " + z); // escopo global
23     }
24     alert("Valor final de z é " + z); // 0 - escopo global
25   </script>
26 </body>
27
28 </html>
```

Estruturas de repetição



while

- Comando que permite repetir instruções enquanto uma condição for verdadeira. Ele não executa as instruções se a condição for falsa logo na primeira verificação. Segue a sintaxe do comando:

```
while (<condição>) {  
    <instruções a serem executadas>  
}
```

Estruturas de repetição



while

- Crie o arquivo **js16.html** com o conteúdo abaixo e execute-o para ver o resultado.

```
<> js16.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Estrutura de repetição - while</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         // Exibe os números pares até 10 e a soma deles
12         let numero = 0;
13         let soma = 0;
14         while (numero<=10) {
15             alert(numero);
16             soma += numero;
17             numero += 2;
18         }
19         alert("A soma dos números é " + soma);
20     </script>
21 </body>
22
23 </html>
```


Estruturas de repetição



do while

- Comando que permite repetir instruções enquanto uma condição for verdadeira. Ele verifica a condição após a primeira execução das instruções, então elas são executadas pelo menos uma vez. Segue a sintaxe do comando:

```
do {  
    <instruções a serem executadas>  
} while (<condição>);
```

Estruturas de repetição



do while

- Crie o arquivo **js17.html** com o conteúdo abaixo e execute-o para ver o resultado.

```
<> js17.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Estrutura de repetição - do while</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         // Exibe os números pares até 10 e a soma deles
12         let numero = 0;
13         let soma = 0;
14         do {
15             alert(numero);
16             soma += numero;
17             numero += 2;
18         } while (numero<=10);
19         alert("A soma dos números é " + soma);
20     </script>
21 </body>
22
23 </html>
```

Estruturas de repetição



break / continue

- Pode ser utilizado o comando **break** para sair de um loop e o comando **continue** para ir diretamente para a próxima iteração do loop. Crie o arquivo **js18.html** com o conteúdo abaixo e execute-o para ver o resultado.

```
<> js18.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5  |   <title>Estrutura de repetição - break / continue</title>
6  |   <meta charset="utf-8">
7  </head>
8
9  <body>
10 |   <script type="text/javascript">
11 |       // Exibe os números ímpares até 11 e a soma deles
12 |       let numero = 0;
13 |       let soma = 0;
14 |       let resto;
15 |       do {
16 |           numero++;
17 |           resto = numero % 2;
18 |           if (resto == 0) { // verifica se é par
19 |               continue; // retorna ao do
20 |           }
21 |           alert(numero);
22 |           soma += numero;
23 |           if (numero>=11) {
24 |               break; // vai para depois do while
25 |           }
26 |       } while (true);
27 |       alert("A soma dos números é " + soma);
28 |   </script>
29 </body>
30
31 </html>
```

Estruturas de repetição



Exercícios:

- 1) Crie um arquivo **js19-ex1.html** que exiba a sequência de Fibonacci até uma quantidade de números informada pelo usuário. A sequência de Fibonacci é uma sequência iniciada por 1 e onde o próximo elemento é a soma do elemento atual com o anterior. Assim, caso seja informado o número 7, deve ser apresentado o seguinte resultado:
1 1 2 3 5 8 13

Funções



Funções



function

- Recurso que permite executar um conjunto de instruções a partir de valores informados (parâmetros) e, caso necessário, retornar um valor como resultado final da função através do comando **return**. Uma função pode ser declarada da seguinte forma:

```
function nome da função(parâmetro1, parâmetroN, ...) {  
    <instruções>;  
    return <valor retornado>;  
}
```
- A função é executada da seguinte forma:
nome da função (valor1, valorN, ...);

Funções



function

- Faça um cópia do arquivo **js11.html** com o nome **js20.html**. Altere o código da forma apresentada abaixo. Veja que é utilizada a função **carcularMedia** para calcular a média. Além disso, veja que o código não apresenta erro mesmo a função tendo sido declarada após a sua utilização. Isto por conta do recurso chamado **hoisting** (elevação) visto anteriormente que faz com que a função declarada com **function** seja elevada ao topo do seu escopo antes do início da execução do código. As variáveis **valor1**, **valor2** e resultado só existem dentro da função (escopo de função).

```
<? js20.html > <? html > <? body > <? script>
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Função</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let nome = prompt("Informe o nome do aluno:");
12         let nota1 = prompt("Informe a primeira nota do aluno:");
13         let nota2 = prompt("Informe a segunda nota do aluno:");
14         media = calcularMedia(Number(nota1), Number(nota2));
15         alert(`O aluno ${nome} teve média de ${media}`);
16         if (media >= 7) {
17             alert("Aprovado!");
18         } else if (media >= 5) {
19             alert("Recuperação");
20         } else {
21             alert("Reprovado");
22         }
23         function calcularMedia (valor1, valor2) {
24             let resultado = (valor1+valor2)/2;
25             return resultado;
26         }
27     </script>
28 </body>
29
30 </html>
```

Funções



function

- Funções podem executar elas próprias, o que chamamos de recursividade. Crie o arquivo **js21.html** conforme o exemplo abaixo e execute-o para ver o resultado. Veja que foi utilizada a função nativa **parseInt** para converter o número em formato de string para um número inteiro da base 10. Lembre-se de informar números decimais com o ponto no lugar da vírgula.

```
<? js21.html > <? html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Função</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     let resposta;
12     do {
13       let numero = Number(prompt("Informe um número inteiro entre 0 e 200:"));
14       if (numero < 0 || numero > 200 || numero!=parseInt(String(numero), 10)) {
15         alert("Entrada inválida!");
16       } else {
17         let fatorial = calcularFatorial(numero);
18         alert("O fatorial de " + numero + " é " + fatorial);
19       }
20       resposta = confirm("Deseja continuar ?");
21     } while (resposta == true);
22     function calcularFatorial(valor) {
23       let fatorial;
24       if (valor >= 0) {
25         switch (valor) {
26           case 0:
27             fatorial = 1;
28             break;
29           case 1:
30             fatorial = 1;
31             break;
32           default:
33             fatorial = valor * calcularFatorial(valor - 1);
34             break;
35         }
36       }
37       return fatorial;
38     }
39   </script>
40 </body>
41
42 </html>
```


Funções



function

- Agora execute o arquivo **js21.html** novamente, mas informe o número 200. Veja que o resultado será **Infinity**, pois o resultado será um número muito grande para ser armazenado como tipo **number**. Salve o arquivo com o nome **js22.html** e altere o código da forma abaixo. Veja a utilização da função **BigInt** para converter o número informado do tipo **number** para **bigint**. Além disso, como não é possível realizar operações de **number** com **bigint** diretamente, veja a utilização da letra **n** nos valores para indicar que são **bigint** e não **number**. Execute o arquivo novamente informando o valor **200** para ver o resultado.

```
<> js22.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Função</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         let resposta;
12         do {
13             let numero = Number(prompt("Informe um número inteiro entre 0 e 200:"));
14             if (numero < 0 || numero > 200 || !Number.isInteger(numero)) {
15                 alert("Entrada inválida!");
16             } else {
17                 let fatorial = calcularFatorial(BigInt(numero));
18                 alert("O fatorial de " + numero + " é " + fatorial);
19             }
20             resposta = confirm("Deseja continuar ?");
21         } while (resposta == true);
22         function calcularFatorial(valor) {
23             let fatorial;
24             if (valor >= 0) {
25                 switch (valor) {
26                     case 0n:
27                         fatorial = 1n;
28                         break;
29                     case 1n:
30                         fatorial = 1n;
31                         break;
32                     default:
33                         fatorial = valor * calcularFatorial(valor - 1n);
34                         break;
35                 }
36             }
37             return fatorial;
38         }
39     </script>
40 </body>
41
42 </html>
```

Arrow Function

- A partir do ECMAScript 6 (ES6) foi criada uma outra forma para declarar funções, sendo este novo formato chamado de **Arrow Function**. Neste novo formato é feito o tratamento para evitar o uso de funções com o mesmo nome. Além disso, a função precisa ser declarada antes de ser utilizada. Segue abaixo algumas formas como declarar uma função neste novo formato:

```
const <nome da função> = (parâmetro1, parâmetroN, ...) => {  
    <instruções>;  
    return <valor retornado>;  
}
```

```
const <nome da função> = (parâmetro1, parâmetroN, ...) => <instrução>;  
const <nome da função> = parâmetro1 => <instrução>;
```

- A função continua sendo executada da seguinte forma:
nome da função (valor1, valorN, ...);

Arrow Function

- Salve o arquivo **js22.html** com o nome **js23.html**. Altere o código da forma apresentada abaixo. Execute o arquivo para confirmar que o resultado será o mesmo.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Função</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     const calcularFatorial = (valor) => {
12       let fatorial;
13       if (valor >= 0) {
14         switch (valor) {
15           case 0:
16             fatorial = 1n;
17             break;
18           case 1:
19             fatorial = 1n;
20             break;
21           default:
22             fatorial = valor * calcularFatorial(valor - 1n);
23             break;
24         }
25       }
26       return fatorial;
27     }
28     let resposta;
29     do {
30       let numero = Number(prompt("Informe um número inteiro entre 0 e 200:"));
31       if (numero < 0 || numero > 200 || numero !== parseInt(String(numero), 10)) {
32         alert("Entrada inválida!");
33       } else {
34         let fatorial = calcularFatorial(BigInt(numero));
35         alert("O fatorial de " + numero + " é " + fatorial);
36       }
37       resposta = confirm("Deseja continuar ?");
38     } while (resposta == true);
39   </script>
40 </body>
41
42 </html>
```

Funções



Exercícios:

- 1) Crie um arquivo **js24-ex1.html** a partir do arquivo **js20.html**, alterando-o de forma a utilizar **Arrow Function** no lugar de **Function**. Além disso, crie uma função chamada **definirSituacao (usando Arrow Function)** que receba a média calculada e retorne se o aluno está aprovado, reprovado ou em recuperação, usando as mesmas condições usadas no arquivo original.

Orientação a objetos



Orientação a objetos



Objeto

- Pode ser considerado como qualquer coisa que tenha um significado dentro do contexto de um problema, seja ela concreta ou abstrata.
- Em termos de programação, podemos definir um objeto como sendo a abstração de uma entidade do mundo real, que apresenta sua própria existência, identificação, características de composição e que tem alguma utilidade, isto é, pode executar determinados serviços quando solicitado.

Orientação a objetos



Objeto literal

- Apesar da linguagem JavaScript não ser totalmente orientada a objetos, como podemos notar até o momento, objetos podem ser criados de forma a agrupar dados de diferentes características (propriedades ou atributos) e serviços (métodos) relacionados a um mesmo elemento, o qual chamamos de **objeto literal**. Assim, podemos criar objetos literais da seguinte forma:

```
[var | let | const] <nome do objeto> = {  
    <propriedade1> : <valor1>,  
    <propriedadeN> : <valorN>,  
    <nome de método1>: function([<parâmetro1>, ..., <parâmetro>]) {  
        <instruções>  
    },  
    <nome de método2>: function([<parâmetro1>, ..., <parâmetro>]) {  
        <instruções>  
    }  
};
```

- Veja que as funções utilizadas para criar os métodos não possuem nome. Por conta disso são chamadas funções anônimas.
- Utilizaremos como padrão de nomenclatura para nome de objetos, atributos e métodos o mesmo padrão utilizado para criar variáveis.

Orientação a objetos



Objeto literal

- Para ler o valor de uma propriedade de um objeto é utilizado o seguinte formato:
<nome do objeto>.<nome da propriedade>
- Para criar ou alterar o valor de uma propriedade de um objeto é utilizado o seguinte formato:
<nome do objeto>.<nome da propriedade> = <valor>;
- Para executar um método de um objeto é utilizado o seguinte formato:
<nome do objeto>.<nome do método>([<parâmetro1>, ..., <parâmetroN>]);

Orientação a objetos



Objeto literal

- Crie o arquivo **js25.html** e execute-o para ver o resultado. Veja que foi criado um objeto de nome **pessoa** com as propriedades **nome** e **idade**, além da criação dos métodos **imprimirDados** para exibir os dados da pessoa e **reajustarSalario** para reajustar o salário dela com base em um percentual informado. Repare que apesar de ter sido utilizado **const** para declarar o objeto, foi possível alterar o valor da propriedade **nome** na linha 24. Por conta disso normalmente é utilizado **const** para declarar objetos, pois assim não poderá ser atribuído um outro objeto à variável. Outro ponto a ser destacado é a utilização da palavra reservada **this** para acessar os valores das propriedades dentro do objeto.

```
<? js25.html > <? html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Objetos literais</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10 <script type="text/javascript">
11   const LIMITE_ADULTO = 18;
12   const pessoa = {
13     nome: "João",
14     idade: 25,
15     salario: 1000,
16     imprimirDados: function() {
17       alert('O nome da pessoa é ${this.nome}, salario ${this.salario} e tem ${this.idade} anos');
18     },
19     reajustarSalario: function(percentualReajuste) {
20       this.salario = this.salario + (this.salario * percentualReajuste / 100);
21       return this.salario;
22     }
23   };
24   pessoa.nome = "Zezinho";
25   pessoa.imprimirDados();
26   let percentual;
27   do {
28     percentual = Number(prompt("Entre com o percentual de reajuste: "));
29   } while(percentual < 0 || percentual > 100)
30   pessoa.reajustarSalario(percentual);
31   alert("O novo salário é " + pessoa.salario);
32   alert("a variável pessoa é do tipo " + typeof pessoa);
33 </script>
34 </body>
35
36 </html>
```

Orientação a objetos



Objetos nativos

- Também chamados de objetos globais. São objetos que fazem parte da linguagem JavaScript definida pela especificação ECMAScript. Assim, os objetos nativos funcionam em qualquer navegador. Cada objeto nativo possui as suas próprias propriedades e métodos. Seguem alguns exemplos de objetos nativos da linguagem:

Objeto nativo	Propriedades	Métodos
Object	https://www.w3schools.com/js/js_object_property.asp	https://www.w3schools.com/js/js_object_methods.asp
Number	https://www.w3schools.com/js/js_number_properties.asp	https://www.w3schools.com/js/js_number_methods.asp
Array	https://www.w3schools.com/js/js_arrays.asp	https://www.w3schools.com/js/js_array_methods.asp
String	https://www.w3schools.com/js/js_strings.asp	https://www.w3schools.com/js/js_string_methods.asp
Math	https://www.w3schools.com/js/js_math.asp	

Orientação a objetos



Array

- É um tipo de objeto nativo que pode armazenar diferentes valores (elementos) em uma mesma variável, sendo seus elementos acessíveis através de um índice. A criação de um Array pode ser feita das seguintes formas:

```
[var | let | const] <nome do array> = [ <valor1>, <valor2>, ..., <valor> ];  
[var | let | const] <nome do array> = new Array(<valor1>, <valor2>, ..., <valor> );
```

- Cada elemento do Array pode ser acessado da seguinte forma:

<nome do array> [<número do índice>]

- O primeiro elemento de um array possui índice igual a 0.
- Como qualquer objeto nativo, possui propriedades e métodos que podem ser utilizados.
- Pode ser utilizada a propriedade length para definir ou obter o número de elementos de um array.

Orientação a objetos



Array

- Seguem alguns métodos de objetos Array:

Método	Parâmetro de Entrada	Saída
indexOf()	<elemento do array>	Retorna a posição onde se encontra o elemento pesquisado e retorna -1 se não for encontrado.
at	<número de um índice de um elemento>	Retorna o elemento contido no índice informado.
find	<nome de uma função>	Executa uma função para cada elemento de um array e, caso o elemento esteja dentro da condição contida na função, retorna o elemento, e se nenhum estiver dentro da condição retorna undefined.
push	<valor>	Adiciona novos elementos para o final de um array e retorna o novo tamanho.
pop	Sem parâmetros	Remove o último elemento do array.
splice	<número do índice>, <número de elementos a serem removidos>	Remove um número de elementos a partir de um índice informado.

- Uma lista de métodos e propriedades de objetos Array podem ser obtidos no link abaixo:
https://www.w3schools.com/jsref/jsref_obj_array.asp

Orientação a objetos



Array

- Crie o arquivo **js26.html** e execute-o para ver o resultado. Veja que foi criado um array de nome **nomes** e os nomes são inseridos no array através do método **push**. Depois os nomes são listados usando-se o método **at** para obter as informações de cada elemento do array. Por fim, é utilizado o método **find** para executar a função **procurarNome**. Esta função compara cada elemento do array à variável **nomeProcurado** (linha 12), retornando o valor desta variável quando o nome é encontrado. Foi incluída uma linha comentada com o método **indexOf** como uma alternativa para pesquisar o nome. Teste também comentando a linha com o método **find** e descomentando a linha com o método **indexOf**.

```
js26.html > html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <title>Array</title>
6   <meta charset="utf-8">
7 </head>
8
9 <body>
10   <script type="text/javascript">
11     const procurarNome = (nome) => {
12       return (nome == nomeProcurado); // Condição a ser verificada
13     }
14     // Inserir nomes no array
15     let nomes = []; // Cria um array vazio
16     do {
17       let nome = prompt("Informe um nome:");
18       nomes.push(nome); // Adiciona nome ao array
19       resposta = confirm("Deseja adicionar mais nomes ?");
20     } while (resposta == true);
21     // Exibir os nomes cadastrados
22     for (let contador = 0; contador <= nomes.length - 1; contador++) {
23       alert(contador + " - " + nomes.at(contador));
24     }
25     // Procurar por nome
26     let nomeProcurado;
27     do {
28       nomeProcurado = prompt("Informe um nome a ser pesquisado: ");
29       if (nomes.find(procurarNome) == nomeProcurado) { // Dará o mesmo resultado com a linha comentada abaixo
30         //if (nomes.indexOf(nomeProcurado)!=-1) {
31           alert("Nome foi encontrado!");
32         } else {
33           alert("Nome não foi encontrado!");
34         }
35       resposta = confirm("Deseja procurar mais nomes ?");
36     } while (resposta == true);
37   </script>
38 </body>
39
40 </html>
```

Orientação a objetos



Classe

- Corresponde a objetos que possuem as mesmas características.
- Possui um conjunto de propriedades (atributos) e serviços (métodos) que podem ser executados. Um serviço também pode ser denominado operação.
- As instâncias de uma classe são os seus objetos.

Orientação a objetos



Classe

- Classes foram introduzidas em JavaScript no ECMAScript 6 (ES6).
- Em JavaScript uma classe é um *template* para objetos. Uma classe é definida da seguinte forma:

```
class <nome da classe> [extends <super classe>] {  
  [ constructor (<parâmetro1>, ..., <parâmetro>) {  
    this.<propriedade1> = <parâmetro1>;  
    this.<propriedadeN> = <parâmetroN>;  
  } ]  
  <propriedade1> [= <valor1>];  
  <propriedadeN> [= <valorN>];  
  <nome de método1> ( [<parâmetro1>, ..., <parâmetroN>] ) {  
    <instruções>  
  }  
  <nome de métodoN> ( [<parâmetro1>, ..., <parâmetroN>] ) {  
    <instruções>  
  }  
};
```

- A palavra-chave **extends** é utilizada quando uma classe herdará informações de uma outra classe (super classe). Neste caso, a classe será uma subclasse da super classe. Não é possível herdar de várias classes, ou seja, não existe herança múltipla em JavaScript. Caso não seja utilizada a palavra-chave **extends**, a classe herdará informações do protótipo (*prototype*) **Object**. Protótipo é o mecanismo pelo qual objetos JavaScript herdam recursos uns dos outros.
- O construtor é um método especial que serve para inicializar propriedades de um objeto da classe. Ele não é obrigatório, mas caso não seja criado, será criado automaticamente um construtor padrão vazio para a classe e, neste caso, as propriedades serão inicializadas com os valores definidos na declaração da classe ou como **undefined** se não forem definidos. Por convenção, o nome da classe deverá estar com a inicial maiúscula. Se o nome for composto, as iniciais das demais palavras também deverão estar em letras maiúsculas. O nome da classe deve ser um substantivo.
- **Por padrão as propriedades e os métodos são públicos, ou seja, visíveis em qualquer parte do código a partir da declaração da classe.**

Orientação a objetos



Classe

- Segue abaixo um exemplo de classe para o objeto pessoa criado em um exemplo anterior:

```
class pessoa {  
    constructor (nome, idade, salario) {  
        this.nome = nome;  
        this.idade = idade;  
        this.salario = salario;  
    }  
    nome;  
    idade;  
    salario;  
    imprimirDados() {  
        alert(`O nome da pessoa é ${this.nome}, salario ${this.salario} e tem  
        ${this.idade} anos`);  
    }  
    reajustarSalario(percentualReajuste) {  
        this.salario = this.salario + (this.salario * percentualReajuste);  
        return this.salario;  
    }  
};
```


Orientação a objetos



Instanciação de objetos

- Para que seja possível usar uma classe é necessário instanciar (criar) um objeto desta classe. A instanciação de um objeto é feita da seguinte forma:

`[var | let | const] <nome do objeto> = new <nome da classe> ([<parâmetro1>, ...]);`

- Não devem ser fornecidos parâmetros caso não tenha sido criado nenhum construtor para a classe.

Ex:

`const pessoa = new Pessoa();`

Orientação a objetos



Classe

- Crie uma cópia do arquivo **js25.html** com o nome **js27-1.html** e altere o arquivo conforme apresentado abaixo de forma a utilizar classes no lugar de objetos literais. Execute o arquivo para confirmar que o resultado será o mesmo.

```
<> js27-1.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Classe</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         const LIMITE_ADULTO = 18;
12         class Pessoa {
13             constructor (nome, idade, salario) {
14                 this.nome = nome;
15                 this.idade = idade;
16                 this.salario = salario;
17             }
18             nome;
19             idade;
20             salario;
21             imprimirDados() {
22                 alert(`O nome da pessoa é ${this.nome}, salario ${this.salario} e tem ${this.idade} anos`);
23             }
24             reajustarSalario(percentualReajuste) {
25                 this.salario = this.salario + (this.salario * percentualReajuste / 100);
26                 return this.salario;
27             }
28         };
29         const pessoa = new Pessoa("Paula", 32, 2000); // Instanciação do objeto pessoa
30         pessoa.nome = "Maria"; // Alteração da propriedade pública nome
31         pessoa.imprimirDados();
32         let percentual;
33         do {
34             percentual = Number(prompt("Entre com o percentual de reajuste: "));
35         } while(percentual < 0 || percentual > 100)
36         pessoa.reajustarSalario(percentual);
37         alert("O novo salário é " + pessoa.salario); // Exibição da propriedade pública salario
38         alert("a variável pessoa é do tipo " + typeof pessoa);
39     </script>
40 </body>
41
42 </html>
```

Orientação a objetos



Classe

- Conforme já comentado, por padrão, os membros da classe (propriedades e métodos) são públicos e, assim, ficam visíveis fora da classe, conforme foi apresentado no exemplo anterior. No **ECMAScript 2020 (ES11)** foi implementada a possibilidade de criar membros privados através do uso do símbolo **#** antes do nome do membro, o que permite que estes fiquem encapsulados na classe. Também foi disponibilizada nesta versão a possibilidade de criação de membros estáticos através da palavra-chave **static**. Membros estáticos ficam acessíveis sem a necessidade de criação de um objeto a partir da classe. Maiores informações no link abaixo:
<https://www.w3schools.io/javascript/es11-private-variables/>
- As propriedades privadas podem ser acessadas de fora da classe através de getters e setters. Maiores informações nos links abaixo:
https://www.w3schools.com/js/js_object_accessors.asp
<https://tr.javascript.info/private-protected-properties-methods>

Orientação a objetos



Classe

- Crie uma cópia do arquivo **js27-1.html** com o nome **js27-2.html** e altere o arquivo conforme apresentado abaixo de forma a utilizar as propriedades como privadas, sendo acessíveis através de getters e setters. Execute o arquivo para confirmar que o resultado será o mesmo. Depois altere a linha 42 de forma a tentar acessar a propriedade privada **#nome** (**pessoa.#nome**) e veja que será apresentado erro na linha. Retorne a linha como estava antes.

```
<> js27-2.html >  html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Classe</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10    <script type="text/javascript">
11      const LIMITE_ADULTO = 18;
12      class Pessoa {
13        constructor (nome, idade, salario) {
14          this.#nome = nome;
15          this.#idade = idade;
16          this.#salario = salario;
17        }
18        #nome;
19        #idade;
20        #salario;
21        get nome() {
22          return this.#nome;
23        }
24        get idade() {
25          return this.#idade;
26        }
27        get salario() {
28          return this.#salario;
29        }
30        set nome(nome) {
31          this.#nome = nome;
32        }
33        imprimirDados() {
34          alert(`O nome da pessoa é ${this.#nome}, salario ${this.#salario} e tem ${this.#idade} anos`);
35        }
36        reajustarSalario(percentualReajuste) {
37          this.#salario = this.#salario + (this.#salario * percentualReajuste / 100);
38          return this.#salario;
39        }
40      };
41      const pessoa = new Pessoa("Paula", 32, 2000); // Instanciação do objeto pessoa
42      pessoa.nome = "Maria"; // Alteração da propriedade privada #nome através do setter nome
43      pessoa.imprimirDados();
44      let percentual;
45      do {
46        percentual = Number(prompt("Entre com o percentual de reajuste: "));
47      } while(percentual < 0 || percentual > 100)
48      pessoa.reajustarSalario(percentual);
49      alert("O novo salário é " + pessoa.salario); // Exibição da propriedade privada #salario através do getter salario
50      alert("a variável pessoa é do tipo " + typeof pessoa);
51    </script>
52  </body>
53
54  </html>
```

Orientação a objetos



Exercícios:

- 1) Crie um arquivo **js28-ex1.js** que declare uma classe Conta com as seguintes informações:

Propriedades públicas:

- agencia
- numeroConta
- saldo

Métodos públicos:

- depositar
- sacar

Crie um arquivo **js28-ex1.html** que apresente um menu com as seguintes opções:

- 1 - Cadastrar conta
- 2 - Depositar
- 3 - Sacar
- 4 - Consultar saldo
- 5 - Encerrar

Caso seja escolhida a opção 1, deverão ser solicitadas/inseridas as propriedades da classe Conta.

Orientação a objetos



Exercícios:

Caso seja escolhida a opção 2, deverá ser solicitado o valor a depositar na conta e realizado o respectivo depósito através do método depositar.

Caso seja escolhida a opção 3, deverá ser solicitado o valor a sacar da conta e realizado o respectivo saque através do método sacar. A conta não deverá ficar negativa, então, caso não haja saldo suficiente, deverá ser apresentada a mensagem "Saldo insuficiente".

Caso seja escolhida a opção 4, deverá ser apresentado o saldo da conta.

Caso seja escolhida a opção 5, a aplicação deverá ser encerrada.

Com exceção da opção 5, ao término de cada uma das opções o menu deverá ser apresentado novamente.

OBS: A aplicação funcionará com apenas uma conta.

Objetos de Host



Objetos de Host



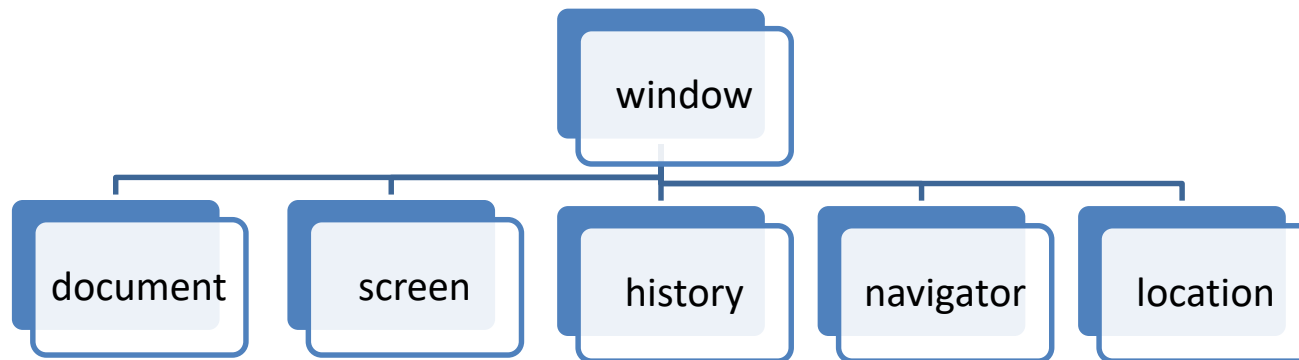
Conceito

- São objetos que são fornecidos pelo ambiente onde está sendo executado o código da linguagem JavaScript (Ex: Navegador). Os objetos do navegador são conhecidos como objetos BOM (Browser Object Model).

Objetos de Host

window

- É o principal objeto do navegador e corresponde a uma janela aberta do navegador. É assumido por padrão quando não é informado. Seguem os principais objetos que fazem parte da hierarquia dele:



- Variáveis com escopo global são propriedades do objeto window.
- Funções globais são métodos do objeto window.
- Para maiores informações consulte os links abaixo:

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

https://www.w3schools.com/js/js_window.asp#:~:text=The%20Window%20Object&text=It%20represents%20the%20browser's%20window,methods%20of%20the%20window%20object.

Objetos de Host



window

- Segue a definição de cada um dos objetos contidos na hierarquia do objeto **window**:
 - ❑ **document**: contém os objetos da página HTML que está aberta. É a raiz na hierarquia de objetos DOM (Document Object Model).
https://www.w3schools.com/js/js_htmlDOM_document.asp
 - ❑ **screen**: contém as informações sobre a tela do usuário que está acessando a página.
https://www.w3schools.com/js/js_window_screen.asp
 - ❑ **history**: contém as páginas visitadas pelo usuário.
https://www.w3schools.com/js/js_window_history.asp
 - ❑ **navigator**: contém informações sobre o browser que está aberto.
https://www.w3schools.com/js/js_window_navigator.asp
 - ❑ **location**: contém informações sobre o endereço da página que está aberta.
https://www.w3schools.com/js/js_window_location.asp

Objetos de Host



window

- Seguem as principais propriedades do objeto **window**:

Propriedade	Descrição
document	Retorna o objeto document da janela. https://www.w3schools.com/jsref/prop_win_document.asp
screen	Retorna o objeto screen da janela. https://www.w3schools.com/jsref/obj_screen.asp
history	Retorna o objeto history da janela. https://www.w3schools.com/JSREF/prop_win_history.asp
navigator	Retornar o objeto navigator da janela. https://www.w3schools.com/jsref/obj_navigator.asp
location	Retorna o objeto location da janela. https://www.w3schools.com/jsref/obj_location.asp
sessionStorage	Salva ou recupera informações da sessão do usuário. As informações são perdidas após o fechamento do navegador. https://www.w3schools.com/jsref/prop_win_sessionstorage.asp
localStorage	Salva ou recupera informações da sessão do usuário. As informações não são perdidas após o fechamento do navegador e estão disponíveis até que seja apagadas. Por conta das legislações de privacidade como a LGPD, GDPR, CCPA, os sites que utilizam este recurso devem obrigatoriamente comunicar a existência dele aos seus visitantes. https://www.w3schools.com/jsref/prop_win_localstorage.asp
console	Permite acesso ao console de debug do navegador. Ex: console.log("Passei por aqui");

Objetos de Host



window

- Seguem alguns métodos do objeto **window**:

Método	Descrição
alert	Exibe uma caixa de alerta na tela. Ex: <code>window.alert("Atenção");</code> https://www.w3schools.com/jsref/met_win_alert.asp
open	Abre uma nova janela. Ex: <code>window.open("http://www.google.com.br");</code> https://www.w3schools.com/jsref/met_win_open.asp
close	Fecha as janelas aberta pelo método open. Ex: <code>window.close();</code> https://www.w3schools.com/jsref/met_win_close.asp
confirm	Exibe uma caixa de diálogo contendo os botões OK e Cancelar. Ex: <code>window.confirm("Deseja continuar");</code> https://www.w3schools.com/jsref/met_win_confirm.asp
print	Imprime a janela atual. Ex: <code>window.print();</code> https://www.w3schools.com/jsref/met_win_print.asp
prompt	Abre uma caixa de diálogo de entrada de informações. Ex: <code>window.prompt("Digite uma data:", "dd/mm/aaaa");</code> https://www.w3schools.com/jsref/met_win_prompt.asp

Objetos de Host



window

- Crie o arquivo **js29.html** conforme apresentado abaixo. Execute-o para ver o funcionamento dele. Veja que foi criada uma variável global de nome **pagina** para armazenar a página aberta. Esta mesma variável é utilizada para fechar a página.

```
<> js29.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Objeto window</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <script type="text/javascript">
11     let pagina;
12     function abrirPagina () {
13       let urlPagina = window.prompt("Informe o endereço de uma página:", "http://");
14       if (urlPagina!=null)
15         pagina = window.open(urlPagina);
16     }
17     function fecharPagina () {
18       let resposta = window.confirm("Deseja fechar a página aberta?");
19       if (resposta)
20         pagina.close();
21     }
22     function imprimirPaginaAtual () {
23       window.print();
24     }
25   </script>
26   <p><button type="button" onclick="abrirPagina()">Abrir página</button></p>
27   <p><button type="button" onclick="fecharPagina()">Fechar página</button></p>
28   <p><button type="button" onclick="imprimirPaginaAtual()">Imprimir página atual</button></p>
29 </body>
30
31 </html>
```

Objetos de Host



document

- Seguem algumas propriedades do objeto **document**:

Propriedade	Descrição
body	Define ou retorna o elemento <body> do documento aberto. Ex: document.body.style.backgroundColor="blue"; https://www.w3schools.com/jsref/prop_doc_body.asp OBS: Para ver maiores informações sobre o objeto Style ver: http://www.w3schools.com/jsref/dom_obj_style.asp
forms	Retorna uma coleção contendo os elementos <form> de um documento. https://www.w3schools.com/js/js_validation.asp
title	Define ou retorna o título do documento aberto. https://www.w3schools.com/jsref/prop_doc_title.asp Ex: document.title = "Novo nome da página";
cookie	Salva ou recupera informações da sessão do usuário. Por padrão as informações não perdidas após o fechamento do navegador, mas pode ser configurada uma data de expiração para que continuem disponíveis após o fechamento do navegador. Por conta das legislações de privacidade como a LGPD, GDPR, CCPA, os sites que utilizam este recurso devem obrigatoriamente comunicar a existência dele aos seus visitantes. https://www.w3schools.com/js/js_cookies.asp Ex: document.cookie = "usuário=teste";

Objetos de Host



document

- Seguem alguns métodos do objeto **document**:

Propriedade	Descrição
getElementById	<p>Retorna o elemento que possui o id especificado.</p> <p>https://www.w3schools.com/jsref/met_document_getelementbyid.asp</p> <p>Ex:</p> <pre>document.getElementById("btn1").style.position="fixed"; document.getElementById("btn1").style.marginTop = "100px"; document.getElementById("btn1").style.marginLeft = "100px"; document.getElementById("cabecalho").innerHTML="Novo texto"; document.getElementById("txtNome").value="João"; document.getElementById("p1").innerText="Maria"; document.getElementById("p1").innerHTML="Maria"; document.getElementById("p1").focus();</pre>
getElementsByName	<p>Retorna uma coleção de elementos que possuem o name especificado</p> <p>https://www.w3schools.com/jsref/met_doc_getelementsbyname.asp</p> <p>Ex: document.getElementsByName("categoria")</p>
querySelector	<p>Retorna um elemento a partir de um seletor CSS (identificador, classe, etc.).</p> <p>https://www.w3schools.com/jsref/met_document_queryselector.asp</p>
write	<p>Escreve código HTML na página. OBS: Deve-se evitar.</p> <p>https://www.w3schools.com/jsref/met_doc_write.asp</p> <p>Ex: document.write("Texto em negrito");</p>

Objetos de Host

document



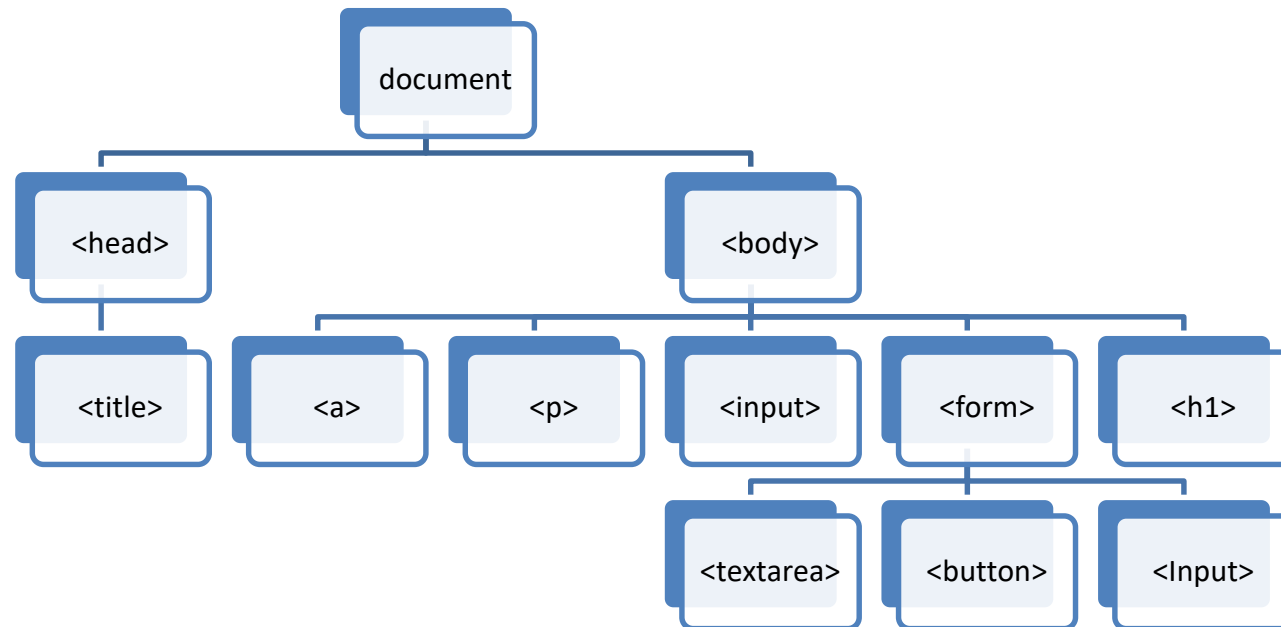
- Crie o arquivo **js30.html** conforme apresentado abaixo. Execute-o para ver resultado.

```
<> js30.html > html
 2  <html lang="pt-br">
 9  <body>
10    <script type="text/javascript">
12      document.write("<b>Clique nos botões</b>");
13      function trocarCorFundoPagina () {
14        do {
15          corPagina = window.prompt("Informe verde ou branco:");
16          if (corPagina!=null)
17            corPagina = corPagina.toUpperCase();
18        } while (corPagina!="VERDE" && corPagina!="BRANCO");
19        switch (corPagina) {
20          case "VERDE":
21            window.document.body.style.backgroundColor="green";
22            break;
23          case "BRANCO":
24            window.document.body.style.backgroundColor="white";
25            break;
26        }
27      }
28      function trocarTituloPagina () {
29        let tituloPagina = window.prompt("Informe vo novo título da página:");
30        if (tituloPagina!=null)
31          document.title = tituloPagina;
32      }
33      function exibirTexto() {
34        alert(document.getElementById('txtTexto').value);
35      }
36    </script>
37    <p><button id="btnTrocarTituloPagina" type="button" onclick="trocarTituloPagina()">Trocar título da página</button></p>
38    <p><button id="btnTrocarCorFundoPagina" type="button" onclick="trocarCorFundoPagina()">Trocar cor de fundo da página</button></p>
39    <p><input id="txtTexto" type="text" placeholder="Digite um texto" autofocus></p>
40    <p><button id="btnExibirTexto" type="button" onclick="exibirTexto()">Exibir texto</button></p>
41  </body>
42
43  </html>
```


Objetos de Host

document

- Segue uma parte da hierarquia de objetos de um objeto document:



Objetos de Host



screen

- Seguem as principais propriedades do objeto **screen**:


Propriedade	Descrição
height	Altura da tela em pixels. Ex: alert("Altura da tela: "+screen.height);
width	Largura da tela em pixels. Ex: alert("Largura da tela: "+screen.width);
colorDepth	Número de bits usados para exibir uma cor. Ex: alert("Número de bits para cor: "+screen.colorDepth);

Objetos de Host



screen

- Crie o arquivo **js31.html** conforme apresentado abaixo. Execute-o para ver o resultado.

```
<> js31.html >  html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Objeto screen</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <script type="text/javascript">
11         alert("Largura da tela: " + screen.width);
12         alert("Altura da tela: " + screen.height);
13         alert("Número de bits usado para cor: " + screen.colorDepth);
14     </script>
15 </body>
16
17 </html>
```

Objetos de Host



history

- Seguem as principais propriedades do objeto **history**:

Propriedade	Descrição
length	Quantidade de páginas contidas no histórico do usuário ativo. Ex: alert(history.length);

- Seguem os principais métodos do objeto **history**:

Método	Descrição
back	Carrega a página anterior do histórico do usuário ativo. Ex: history.back();
forward	Carrega a próxima página do histórico do usuário ativo. Ex: history.forward();

Objetos de Host



history

- Crie os arquivos **js32.html** e **js33.html** conforme apresentado abaixo. Execute o arquivo **js32.html** e veja que ao clicar no botão **Avançar** nada acontecerá porque existe apenas a própria página no histórico. Clique depois no botão **Abrir página** para abrir a página **js33.html**. Veja que o nome do arquivo foi passado como parâmetro para a função **abrirPagina** usando aspas simples por conta do evento **onClick** da tag **button** já usar aspas duplas. Além disso, veja que foi utilizado no método **open** do objeto **window** o parâmetro “**_self**” para abrir a página na mesma aba do navegador. Agora clique no botão **Voltar** da página **js33.html** para retornar a página anterior. Clique agora novamente no botão **Avançar** da página **js32.html** para ir para a página **js33.html** aberta anteriormente e que agora está no histórico de navegação.

```
js32.html > html > body
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <title>Objeto history</title>
6   <meta charset="utf-8">
7 </head>
8
9 <body>
10  <p id="txtHistoricoPaginas"></p>
11  <p><button id="btnAvançar" onClick="avancarPagina()">Avançar</button></p>
12  <p><button id="btnAbrirPagina" onClick="abrirPagina('js33.html')">Abrir página</button></p>
13  <script type="text/javascript">
14    function avancarPagina () {
15      history.forward();
16    }
17    function abrirPagina(pagina) {
18      window.open(pagina, "_self"); // _self para abrir na mesma aba do navegador
19    }
20    document.getElementById("txtHistoricoPaginas").innerText = "Páginas no histórico: "+history.length;
21  </script>
22 </body>
23
24 </html>
```

```
js33.html > html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <title>Objeto history</title>
6   <meta charset="utf-8">
7 </head>
8
9 <body>
10  <p id="txtHistoricoPaginas"></p>
11  <p><button id="btnVoltar" onClick="voltarPagina()">Voltar</button></p>
12  <script type="text/javascript">
13    function voltarPagina () {
14      history.back();
15    }
16    document.getElementById("txtHistoricoPaginas").innerText = "Páginas no histórico: "+history.length;
17  </script>
18 </body>
19
20 </html>
```

Objetos de Host



navigator

- Seguem as principais propriedades do objeto **navigator**:

Propriedade	Descrição
appName	Retorna o nome do navegador utilizado. Ex: alert("Nome do navegador: "+navigator.appName);
appVersion	Retorna a versão do navegador utilizado. Ex: alert("Versão do navegador: "+navigator.appVersion);
cookieEnabled	Retorna se os cookies estão ativos no navegador. Ex: alert("Cookies ativos: "+navigator.cookieEnabled);

- Seguem os principais métodos do objeto **navigator**:

Método	Descrição
javaEnabled	Retorna se o Java está habilitado no navegador. Ex: alert("Java ativo: "+navigator.javaEnabled());

Objetos de Host

navigator



- Crie o arquivo **js34.html** conforme apresentado abaixo. Execute-o para ver o resultado.

```
<> js34.html >  html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Objeto navigator</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <p id="txtNavegador"></p>
11     <p id="txtVersaoNavegador"></p>
12     <script type="text/javascript">
13         document.getElementById("txtNavegador").innerHTML = "<b>Navegador: "+navigator.appName+"</b>";
14         document.getElementById("txtVersaoNavegador").innerHTML = "<b>Versão do navegador: "+navigator.appVersion+"</b>";
15     </script>
16 </body>
17
18 </html>
```

Objetos de Host



location

- Seguem as principais propriedades do objeto **location**:

Propriedade	Descrição
hostname	Nome do domínio da página ativa. Ex: alert("Domínio da página: "+location.hostname);
href	Retorna o nome completo da página. Ex: alert("Endereço da página: "+location.href);
pathname	O caminho e nome do arquivo da página. Ex: alert("Caminho e nome do arquivo: "+location.pathname);
protocol	O protocolo da página. Ex: alert("Protocolo: "+location.protocol);

- Seguem os principais métodos do objeto **location**:

Método	Descrição
reload	Recarrega a página atual. Ex: location.reload();
assign	Carrega uma nova página, mas mantém a antiga no histórico. Ex: location.assign("http://www.google.com");
replace	Carrega uma nova página, mas não mantém a antiga no histórico. Ex: location.replace("http://www.google.com");

Objetos de Host



location

- Crie o arquivo **js35.html** conforme apresentado abaixo. Execute-o para ver o resultado. Após executar o arquivo, digite um conteúdo na caixa de texto e clique no botão **Recarregar a página**. Clique no botão **Substituir a página** e verifique que não será possível retornar com o botão **Voltar** porque o histórico terá sido apagado. Após executar novamente o arquivo, clique no botão **Carregar nova página** e verifique que será possível retornar com o botão **Voltar** porque o histórico não terá sido apagado.

```
<> js35.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Objeto location</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <h1>Informações da página</h1>
11     <p id="txtHistoricoPaginas"></p>
12     <p id="txtURL"></p>
13     <p id="txtProtocolo"></p>
14     <p id="txtDominio"></p>
15     <p id="txtCaminho"></p>
16     <script type="text/javascript">
17         document.getElementById("txtHistoricoPaginas").innerText = "Páginas no histórico: "+history.length;
18         document.getElementById("txtURL").innerHTML = "<b>URL: "+location.href+"</b>";
19         document.getElementById("txtProtocolo").innerHTML = "<b>Protocolo: "+location.protocol+"</b>";
20         document.getElementById("txtDominio").innerHTML = "<b>Domínio: "+location.hostname+"</b>";
21         document.getElementById("txtCaminho").innerHTML = "<b>Caminho e nome do arquivo: "+location.pathname+"</b>";
22     </script>
23     <p><label for="txtTexto">Informe um texto:</label><input id="txtTexto" type="text" placeholder="Digite um texto" autofocus></p>
24     <p><button id="btnRecarregarPagina" type="button" onclick="location.reload()">Recarregar a página</button></p>
25     <p><button id="btnSubstituirPagina" type="button" onclick="location.replace('js33.html')">Substituir a página</button></p>
26     <p><button id="btnCarregarPagina" type="button" onclick="location.assign('js33.html')">Carregar nova página</button></p>
27 </body>
28
29 </html>
```

Objetos de Host



Aplicação prática

- Agora criaremos uma aplicação com os conceitos apresentados neste capítulo. A aplicação será a conversão dos arquivos **js28-ex1.html** e **js28-ex1.js** de forma que não sejam mais utilizados métodos como **alert** e **prompt**. A ideia é utilizar os recursos de interação da linguagem JavaScript com as páginas HTML. Primeiramente criaremos o menu principal utilizando-se da tag `<a>` do HTML e chamando diferentes páginas HTML para cada opção do menu. Assim, crie o arquivo **js36.html** conforme apresentado abaixo. Execute-o para ver o resultado.

```
<> js36.html >  html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Menu Principal</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <h1>Menu Principal</h1>
11     <p><a href="javascript:location.assign('js36-cadastrar.html');">Cadastrar a conta</a></p>
12     <p><a href="javascript:location.assign('js36-depositar.html');">Depositar</a></p>
13     <p><a href="javascript:location.assign('js36-sacar.html');">Sacar</a></p>
14     <p><a href="javascript:location.assign('js36-consultar.html');">Consultar</a></p>
15     <p><a href="javascript:location.replace('js36-encerrar.html');">Encerrar</a></p>
16 </body>
17
18 </html>
```

Objetos de Host



Aplicação prática

- Para que possamos manter as informações da classe **Conta** em diferentes páginas, utilizaremos o objeto **sessionStorage** para armazenar os atributos da classe Conta de forma que possamos acessá-los em diferentes páginas da nossa aplicação. Desta forma, crie o arquivo **js36.js** dentro do subdiretório **js** conforme apresentado abaixo. Veja que foram criados os métodos **cadastrar** e **obterDados**. Repare também que o método **obterDados** é chamado nos métodos **depositar** e **sacar** de forma que os atributos da classe contenham o valor que estiver armazenado no objeto **sessionStorage**.

```
js > JS js36.js > Conta > saldo
1 class Conta {
2   agencia;
3   numeroConta;
4   saldo;
5   cadastrar(agencia, numeroConta) {
6     this.agencia = agencia;
7     this.numeroConta = numeroConta;
8     this.saldo = 0;
9     sessionStorage.setItem("agencia", this.agencia);
10    sessionStorage.setItem("numeroConta", this.numeroConta);
11    sessionStorage.setItem("saldo", this.saldo);
12  }
13  obterDados() {
14    this.agencia = Number(sessionStorage.getItem("agencia"));
15    this.numeroConta = Number( sessionStorage.getItem("numeroConta"));
16    this.saldo = Number(sessionStorage.getItem("saldo"));
17  }
18  depositar(valor) {
19    this.obterDados();
20    this.saldo = this.saldo + valor;
21    sessionStorage.setItem("saldo", this.saldo);
22  }
23  sacar(valor) {
24    this.obterDados();
25    let novoSaldo = this.saldo - valor;
26    if (novoSaldo < 0) {
27      return false;
28    }
29    this.saldo = novoSaldo;
30    sessionStorage.setItem("saldo", this.saldo);
31    return true;
32  }
33 }
```

Objetos de Host

Aplicação prática



- Agora vamos começar a criar as páginas correspondentes às opções do menu principal. Primeiramente criaremos a página **js36-cadastrar.html** para cadastrar os atributos da classe **Conta**. Crie o arquivo conforme apresentado abaixo e chame a opção correspondente a partir da página **js36.html**. Veja que a entrada de dados está sendo feita através da tag **<input>** do HTML. Foi utilizada a tag **<button>** do HTML para executar a funcionalidade de voltar a página anterior e de cadastrar as informações. O botão **Gravar** chama a função **cadastrar** através do evento **onClick** para validar os dados e, casos todos estejam preenchidos corretamente, chamar o método **cadastrar** da classe **Conta** para armazenar os atributos através do objeto **sessionStorage**.

```
> js36-cadastrar.html X
<> js36-cadastrar.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Cadastramento de conta</title>
6    <meta charset="utf-8">
7    <script type="text/javascript" src="js/js36.js"></script>
8  </head>
9
10 <body>
11   <h1>Cadastramento de Conta</h1>
12   <p><label for="txtAgencia">Agência:</label><input id="txtAgencia" type="text" placeholder="Digite uma agência" autofocus></p>
13   <p><label for="txtNumero">Número:</label><input id="txtNumero" type="text" placeholder="Digite uma conta"></p>
14   <p id="txtMensagem"></p>
15   <p><button id="btnRetornar" type="button" onclick="history.back()">Retornar</button>
16     <button id="btnGravar" type="button" onclick="cadastrar()">Gravar</button>
17 </p>
18 <script type="text/javascript">
19   function cadastrar() {
20     let mensagem;
21     if (document.getElementById('txtAgencia').value == "") {
22       mensagem = "Preencha a agência";
23       document.getElementById('txtAgencia').focus();
24     } else if (isNaN(document.getElementById('txtAgencia').value)) {
25       mensagem = "Preencha um valor numérico para a agência";
26       document.getElementById('txtAgencia').focus();
27     } else if (document.getElementById('txtNumero').value == "") {
28       mensagem = "Preencha o número da conta";
29       document.getElementById('txtNumero').focus();
30     } else if (isNaN(document.getElementById('txtNumero').value)) {
31       mensagem = "Preencha um valor numérico para a conta";
32       document.getElementById('txtNumero').focus();
33     } else {
34       const conta1 = new Conta();
35       conta1.cadastrar(Number(document.getElementById('txtAgencia').value),
36                       Number(document.getElementById('txtNumero').value));
37       mensagem = "Gravação realizada com sucesso!";
38     }
39     document.getElementById("txtMensagem").innerHTML = `<b>${mensagem}</b>`;
40   }
41 </script>
42 </body>
43
44 </html>
```

Objetos de Host



Aplicação prática

- Agora vamos criar a página **js36-depositar.html** para fazer o depósito na conta. Veja que foi utilizado o método global **isNaN** para verificar se o valor preenchido é numérico. Repare também na chamada da função **depositar** no evento **onClick** do botão **Gravar**. Esta função valida os dados antes da chamada do método **depositar** da classe **Conta**. Crie o arquivo conforme apresentado abaixo e chame a opção correspondente a partir da página **js36.html**.

```
> js36-depositar.html > html > head > title
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Depósito em conta</title>
6    <meta charset="utf-8">
7    <script type="text/javascript" src="js/js36.js"></script>
8  </head>
9
10 <body>
11   <h1>Depositar</h1>
12   <p><label for="txtValor">Valor a depositar:</label><input id="txtValor" type="text" autofocus></p>
13   <p id="txtMensagem"></p>
14   <p><button id="btnRetornar" type="button" onclick="history.back()">Retornar</button>
15     <button id="btnGravar" type="button" onclick="depositar()">Gravar</button></p>
16   <script type="text/javascript">
17     function depositar() {
18       let mensagem;
19       if (document.getElementById('txtValor').value=="") {
20         mensagem = "Preencha o valor a depositar";
21         document.getElementById('txtValor').focus();
22       } else if (isNaN(document.getElementById('txtValor').value)) {
23         mensagem = "Preencha um valor numérico";
24         document.getElementById('txtValor').focus();
25       } else {
26         const conta1 = new Conta();
27         conta1.depositar(Number(document.getElementById('txtValor').value))
28         mensagem = "Depósito realizado com sucesso!";
29       }
30       document.getElementById("txtMensagem").innerHTML = `<b>${mensagem}</b>`;
31     }
32   </script>
33 </body>
34
35 </html>
```

Objetos de Host



Aplicação prática

- Agora vamos criar a página **js36-sacar.html** para fazer o saque na conta. Veja que foi utilizado o método global **isNaN** para verificar se o valor preenchido é numérico. Repare também na chamada da função **sacar** no evento **onClick** do botão **Gravar**. Esta função valida os dados antes da chamada do método **sacar** da classe **Conta**. Crie o arquivo conforme apresentado abaixo e chame a opção correspondente a partir da página **js36.html**.

```
<? js36-sacar.html > <html>
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Saque em conta</title>
6    <meta charset="utf-8">
7    <script type="text/javascript" src="js/js36.js"></script>
8  </head>
9
10 <body>
11   <h1>Sacar</h1>
12   <p><label for="txtValor">Valor a sacar:</label><input id="txtValor" type="text" autofocus></p>
13   <p id="txtMensagem"></p>
14   <p><button id="btnRetornar" type="button" onclick="history.back()">Retornar</button>
15     <button id="btnGravar" type="button" onclick="sacar()">Gravar</button></p>
16   <script type="text/javascript">
17     function sacar() {
18       let mensagem;
19       if (document.getElementById('txtValor').value=="") {
20         mensagem = "Preencha o valor a depositar";
21         document.getElementById('txtValor').focus();
22       } else if (isNaN(document.getElementById('txtValor').value)) {
23         mensagem = "Preencha um valor numérico";
24         document.getElementById('txtValor').focus();
25       } else {
26         const conta1 = new Conta();
27         if (conta1.sacar(Number(document.getElementById('txtValor').value)))
28           mensagem = "Saque realizado com sucesso!";
29         else
30           mensagem = "Saldo indisponível!";
31       }
32       document.getElementById("txtMensagem").innerHTML = `<b>${mensagem}</b>`;
33     }
34   </script>
35 </body>
36
37 </html>
```

Objetos de Host



Aplicação prática

- Agora vamos criar a página **js36-consultar.html** para fazer a consulta do saldo da conta. Veja que foi utilizado o método **obterDados** da classe **Conta** para consultar os dados da conta. Repare também que os dados são recuperados a partir do objeto **sessionStorage** criado no método **cadastrar**. Crie o arquivo conforme apresentado abaixo e chame a opção correspondente a partir da página **js36.html**.


```
<> js36-consultar.html > html > head > title
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Consulta de conta</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js36.js"></script>
8  </head>
9
10 <body>
11     <h1>Consulta de Conta</h1>
12     <p><label for="txtAgencia">Agência:</label><input type="text" id="txtAgencia" readonly></p>
13     <p><label for="txtNumero">Número:</label><input type="text" id="txtNumero" readonly></p>
14     <p><label for="txtSaldo">Saldo:</label><input type="text" id="txtSaldo" readonly></p>
15     <p><button id="btnRetornar" type="button" onclick="history.back()" autofocus>Retornar</button></p>
16     <script type="text/javascript">
17         const conta1 = new Conta();
18         conta1.obterDados();
19         document.getElementById('txtAgencia').value = conta1.agencia;
20         document.getElementById('txtNumero').value = conta1.numeroConta;
21         document.getElementById('txtSaldo').value = conta1.saldo;
22     </script>
23 </body>
24
25 </html>
```


Objetos de Host



Aplicação prática

- Finalmente agora vamos criar a página **js36-encerrar.html** para fazer o encerramento da aplicação. Veja que esta página foi chamada através do método `replace` do objeto `location` de forma a não levar o histórico de páginas. Crie o arquivo conforme apresentado abaixo e chame a opção correspondente a partir da página **js36.html**.

```
<> js36-encerrar.html >  html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Encerramento</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <h1>Programa encerrado</h1>
11 </body>
12
13 </html>
```


Objetos de Host



Exercícios:

- 1) Crie o arquivo **js37-ex1.html** de forma que através de links possa carregar páginas conforme as opções escolhidas:
- Opção 1 - Cadastrar material: Abrir a página **js37-ex1-cadastrar.html**
 - Opção 2 - Entrada de material: Abrir a página **js37-ex1-entrar.html**
 - Opção 3 - Saída de material: Abrir a página **js37-ex1-sair.html**
 - Opção 4 - Consultar saldo em estoque: Abrir a página **js37-ex1-consultar.html**
 - Opção 5 - Encerrar: Abrir a página **js37-ex1-encerrar.html**

Crie um novo arquivo chamado **js37-ex1.js** dentro do subdiretório **js** e nele crie uma classe chamada **Material** que contenha os seguintes atributos:

- **codMaterial**
- **descMaterial**
- **qtdeEstoque**

Crie na classe **Material** os seguintes métodos:

- **entrarMaterial**: Deve receber uma quantidade de material e aumentar o estoque do material.
- **sairMaterial**: Deve receber uma quantidade de material e diminuir o estoque do material. O estoque não pode ficar negativo. Se o novo estoque calculado for negativo, o estoque deve permanecer inalterado e uma mensagem de "Não há estoque suficiente!" deve ser apresentada. Se o novo estoque calculado for igual ou maior que zero, o estoque deve ser reduzido e uma mensagem de "Saída do material realizada com sucesso!" deve ser apresentada.

Objetos de Host



Exercícios:

Se for escolhida a opção 1, o programa deve solicitar pelo teclado o código e a descrição do material e armazená-los na classe Material. O material deve ser criado com estoque igual a zero. Os dados da classe material deverão estar acessíveis em qualquer página da aplicação.

Se for escolhida a opção 2, o programa deve solicitar pelo teclado a quantidade de material a ser adicionada e, após o preenchimento, aumentar o estoque do material através do método **entrarMaterial**.

Se for escolhida a opção 3, o programa deve solicitar pelo teclado a quantidade de material a ser retirada e, após o preenchimento, reduzir o estoque do material através do método **sairMaterial**.

Se a opção escolhida for 4, o programa deve apresentar o código, a descrição e a quantidade em estoque do material.

A opção 5 deverá apresentar a mensagem "Programa encerrado" em formato HTML e não deve ser permitido retornar a página principal, nem mesmo com o botão Voltar do navegador.

Objetos de Host



Exercícios:

Com exceção da opção 5, todas as demais páginas deverão ter um botão **Voltar** que permita retornar a página principal.

A entrada de dados nas opções deverá ser feita através das tags label e input do HTML. Deve-se utilizar botões para confirmar a operação. Se algum campo não for preenchido ou estiver inválido deverá ser apresentada uma mensagem abaixo dos campos como "Preencha o campo XXX" ou "Campo XXX preenchido incorretamente".

Formulários



Formulários



Formulário

<form [atributos] [eventos]>
objetos
</form>

- Define um formulário em HTML. Os atributos podem ser:
 - ❑ **id**: identifica o formulário;
 - ❑ **name**: define um nome para o formulário;
 - ❑ **action**: define a página que irá receber os dados do formulário quando este for submetido;
 - ❑ **method**: define o método de envio dos dados. Pode ser:
 - **get**: envia os dados no endereço (URL) da página;
 - **post**: envia os dados encapsulados na mensagem http.

Formulários



Formulário

Os principais eventos associados a formulários são:

- **onsubmit:** executado antes do formulário a ser submetido. Normalmente utilizado com uma função de validação do formulário. Se o resultado desta função retornar **true**, o formulário será submetido. Se o resultado da função for **false**, o formulário não será submetido;
- **onreset:** executado antes do formulário ser resetado (inicializado). Pode ser utilizado com uma função para confirmação da ação. Se o resultado desta função retornar **true**, o formulário será resetado. Se o resultado da função for **false**, o formulário não será resetado.

Formulários



Formulário

Os objetos do formulário podem ser principalmente:

- Entrada de dados (**input**);
- Descrições (**label**);
- Campos de múltiplas linhas (**textarea**);
- Campos de seleção (**select**);
- Lista de dados (**Datalist**);
- Botões (**button**).

Formulários



Entrada de dados

<input [atributos] [eventos]>

- Define um campo de entrada de dados em um formulário. Os atributos podem ser:
 - ❑ **type**: define o tipo do campo. Podendo ser principalmente:
 - **text**: campo de texto;
 - **password**: campo de senha;
 - **number**: campo de número;
 - **email**: campo de e-mail;
 - **date**: campo de data. Não suportado pelo Internet Explorer;
 - **checkbox**: caixa de marcação;
 - **radio**: botão de seleção;
 - **button**: botão normal;
 - **reset**: botão para resetar (inicializar) o formulário;
 - **submit**: botão para submeter o formulário.

Formulários



Entrada de dados

<input [atributos] [eventos]>

- ☐ **id**: define um identificador para o campo;
- ☐ **name**: define um nome para o campo;
- ☐ **autofocus**: define que o campo receberá o foco ao ser aberta a página;
- ☐ **list**: define o identificador (id) de uma lista de dados (datalist);
- ☐ **value**: define o valor inicial do campo;
- ☐ **size**: número de caracteres do campo na tela;
- ☐ **maxlength**: número máximo de caracteres do campo;
- ☐ **checked**: define se um campo do tipo checkbox ou radio estará selecionado;
- ☐ **disabled**: define se o campo estará desativado;
- ☐ **readonly**: define se o campo estará somente para a leitura;
- ☐ **required**: define que o campo deve ser preenchido.

Formulários



Entrada de dados

<label [atributos] [eventos]> texto </label>

- Define um label para um campo de entrada (input). Os atributos podem ser:
 - ☐ **for**: define o nome do campo de entrada (input);
 - ☐ **id**: define um identificador para o campo.

Formulários



Entrada de dados

<textarea [atributos] [eventos]> Texto </textarea>

- Define um campo de entrada de dados de múltiplas linhas em um formulário. Os atributos podem ser:
 - ☐ **id**: define um identificador para o campo;
 - ☐ **name**: define um nome para o campo;
 - ☐ **autofocus**: define que o campo receberá o foco ao ser aberta a página;
 - ☐ **rows**: número de linhas visíveis no campo;
 - ☐ **cols**: número de colunas visíveis no campo;
 - ☐ **maxlength**: número máximo de caracteres do campo;
 - ☐ **disabled**: define se o campo estará desativado;
 - ☐ **readonly**: define se o campo estará somente para a leitura;
 - ☐ **required**: define que o campo deve ser preenchido.

Formulários



Entrada de dados

```
<select [atributos] [eventos]>  
  <option value=valor1> texto1 </option>  
  <option value=valorN> textoN </option>  
</select>
```

- Define uma lista de opções. Os atributos podem ser:
 - ☐ **id**: define um identificador para o campo;
 - ☐ **name**: define um nome para o campo;
 - ☐ **autofocus**: define o campo que deve receber o foco quando a página for carregada;
 - ☐ **disabled**: define que o campo ficará desativado;
 - ☐ **multiple**: define que mais de uma opção pode ser selecionada;
 - ☐ **required**: define que o campo deve ser preenchido antes do formulário ser submetido;
 - ☐ **size**: define o número de opções visíveis.

Formulários



Entrada de dados

```
<datalist [atributos] [eventos]>  
  <option value=valor1 >  
  <option value=valorN >  
</datalist>
```

- Define uma lista de dados para a tag input. Os atributos podem ser:
 - ❑ **id**: define um identificador para o campo.

Formulários



Execução de ação

<button [atributos] [eventos]> Texto </button>

- Define uma lista de dados para a tag input. Os atributos podem ser:
 - ☐ **id**: define um identificador para o campo.
 - ☐ **name**: define um nome para o botão.
 - ☐ **autofocus**: define o foco no botão após a carga da página.
 - ☐ **form**: define a qual formulário o botão pertence.
 - ☐ **disabled**: desativa o botão.
 - ☐ **formaction**: define a URL que será carregada quando for clicado em um botão com type="submit".
 - ☐ **value**: define um valor inicial para o botão.
 - ☐ **type**: define o tipo de ação que será executada pelo botão.
 - **button**: botão normal.
 - **submit**: submete o formulário para a URL contida no parâmetro action do formulário a que pertence.
 - **reset**: reinicia (reseta) os campos do formulário.

Formulários



Eventos

Os principais eventos associados a objetos de entrada de dados são:

- **onfocus**: executado quando um objeto ganha o foco;
- **onfocusout**: executado quando um objeto perde o foco;
- **oninput**: executado a cada entrada de dados em um campo do tipo **input** ou **textarea**;
- **onselect**: executado quando um texto é selecionado em um campo do tipo **input** ou **textarea**;
- **onchange**: executado quando é concluída a alteração de um campo;
- **onclick**: executado quando um campo é clicado;
- **ondblclick**: executado quando é aplicado um duplo clique em um campo.
- **onkeypress**: executado quando é clicada uma tecla em um campo do tipo **input**.
- **onload**: executado ao carregar uma página.

Validação

Para validar os campos de um formulário normalmente são incluídas condições no evento **onsubmit** do formulário com a inclusão de críticas usando o método **getElementById** do objeto **document**. Seguem alguns exemplos de propriedade e métodos muito usados:

- **document.getElementById("<id>").focus()**: define o foco no elemento identificado como <id>;
- **document.getElementById("<id>").style.color**: define a cor do texto de um elemento identificado como <id>;
- **document.getElementById("<id>").value**: define ou retorna o conteúdo de um elemento identificado como <id>;
- **document.getElementById("<id>").disabled**: define ou retorna como bloqueado (true) ou desbloqueado (false) o elemento identificado como <id>;
https://www.w3schools.com/tags/att_disabled.asp
- **document.getElementById("<id>").checked**: define ou retorna como marcado (true) ou desmarcado (false) o elemento (**radio** ou **checkbox**) identificado como <id>.

Formulários



Aplicação prática

- Agora criaremos uma aplicação com os conceitos apresentados neste capítulo. Será criada uma aplicação que permita manipular um conjunto de mensagens que deverão ser mantidas mesmo com o fechamento do navegador. Assim, primeiramente criaremos uma página como menu para acesso às páginas que permitirão a manutenção das mensagens, sendo uma página para a gravação das mensagens, uma página para listagem de todas as mensagens gravadas e uma página para apagar todas as mensagens gravadas. Crie o arquivo **js38.html** conforme apresentado abaixo e execute-o para ver resultado. Os links para as páginas ainda não funcionarão por conta da ausência das páginas.

```
<> js38.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Menu Principal</title>
6      <meta charset="utf-8">
7  </head>
8
9  <body>
10     <h1>Menu Principal</h1>
11     <p><a href="javascript:location.assign('js38-cadastrar-mensagem.html');">Cadastrar mensagem</a></p>
12     <p><a href="javascript:location.assign('js38-ler-mensagens.html');">Ler mensagens</a></p>
13     <p><a href="javascript:location.assign('js38-apagar-mensagens.html');">Apagar mensagens</a></p>
14 </body>
15
16 </html>
```

Formulários



Aplicação prática

- Para que possamos manter as mensagens criaremos uma classe chamada **Mensagem** para executar todas as ações em relação às mensagens. Veja que para manter as mensagens da classe **Mensagem** em diferentes páginas da nossa aplicação, e mantendo-as mesmo após o fechamento do navegador, utilizaremos o objeto **localStorage** para armazenar um *array* de objetos da classe **Mensagem**. Veja que foram criados os métodos **salvarMensagem** e **lerMensagens**, onde ambos utilizam o objeto **localStorage**. O método **salvarMensagem** permite gravar o array **mensagens** no **localStorage** através do método **stringify** de um objeto JSON (*JavaScript Object Notation*) que converte o array em string e salva no **localStorage** com o nome “**mensagens**”. O método **lerMensagens** realiza a leitura da string salva no **localStorage** com o nome “**mensagens**” e a converte de volta para um array através do método **parse** do objeto **JSON**, mas antes verifica a existência de “**mensagens**” no **localStorage** através do método **hasOwnProperty** do objeto **localStorage**. O método **salvarMensagem** faz a chamada ao método **lerMensagem** para obter as mensagens já gravadas e adicionar a nova mensagem ao final do array **mensagens**. Desta forma, crie o arquivo **js38.js** dentro do subdiretório **js** conforme apresentado abaixo.

```
js > JS js38.js > Mensagem
1 class Mensagem {
2   constructor(titulo, assunto, categoria, email) {
3     this.titulo = titulo;
4     this.assunto = assunto;
5     this.categoria = categoria;
6     this.email = Boolean(email);
7   }
8   salvarMensagem(mensagem) {
9     const mensagens = this.lerMensagens(); // ler mensagens do localStorage
10    mensagens.push(mensagem); // adiciona mensagem no array mensagens
11    localStorage.setItem("mensagens", JSON.stringify(mensagens)); // Converte array em string
12  }
13  lerMensagens() {
14    let mensagens = new Array();
15    if (localStorage.hasOwnProperty("mensagens")) { // Verifica se tem mensagens no localStorage
16      mensagens = JSON.parse(localStorage.getItem("mensagens")); // Lê as mensagens e converte em array
17    } else {
18      mensagens = []; // retorna array vazio
19    }
20    return mensagens;
21  }
22 }
```

Formulários



Aplicação prática

- Agora criaremos a página **js38-cadastrar-mensagem.html** para gravação das mensagens. Veja que todos os campos de entrada de dados do formulário possuem o atributo **name**, visto que este atributo que define os campos que serão enviados (submetidos) para a página identificada no atributo **action** do formulário quando este for enviado (submetido). O formulário será enviado ao pressionar a tecla ENTER em algum dos campos do formulário ou clicar no botão **Salvar** que possui o atributo **type="submit"**. Está sendo utilizado o método **get** para que a página **js38-salvar-mensagem.html** possa capturar na URL os campos enviados pelo formulário. Entretanto, antes de enviar o formulário, será chamada a função **validarCampos()** para que os campos possam ser validados antes do formulário ser enviado. O formulário só será enviado caso o retorno da função seja igual a **true**. Também é utilizada a função **limparCampos** para limpar os campos do formulário ao clicar no botão **Limpar**.

```
<? js38-cadastrar-mensagem.html > <? html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3
4 <head>
5   <title>Cadastrar mensagem</title>
6   <meta charset="utf-8">
7   <script type="text/javascript" src="js/js38.js"></script>
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9 </head>
10
11 <body onload="verificaGravacao()">
12   <form name="form1" action="js38-salvar-mensagem.html" method="get" onsubmit="return validarCampos()">
13     <p><label for="txtTitulo">Título:</label></p>
14     <p><input id="txtTitulo" name="txtTitulo" type="text" size="50" placeholder="Informe um título" autofocus></p>
15     <p><label for="txtAssunto">Assunto:</label></p>
16     <p><textarea id="txtAssunto" name="txtAssunto" cols="50"
17       placeholder="Descreva o assunto"></textarea></p>
18     <p><label for="selCategoria">Categoria:</label></p>
19     <p>
20       <select id="selCategoria" name="selCategoria">
21         <option value="">&nbsp;</option> <!-- espaço em branco -->
22         <option value="D">Dúvida</option>
23         <option value="S">Sugestão</option>
24         <option value="R">Reclamação</option>
25         <option value="O">Outros</option>
26       </select>
27     </p>
28     <p><label for="selCategoria">Enviar por e-mail:</label></p>
29     <p><input id="chkEmail" name="chkEmail" type="checkbox"></p>
30     <p id="txtMensagem"></p>
31     <p>
32       <button type="button" onclick="limparCampos()">Limpar</button>
33       <button type="button" onclick="history.back()">Voltar</button>
34       <button type="submit">Salvar</button>
35     </p>
36   </form>
37   <script type="text/javascript">
38     const verificaGravacao = () => {
39       if (sessionStorage.hasOwnProperty("resultado")) { // Verifica se houve tentativa de gravação
40         document.getElementById('txtMensagem').innerHTML = "<b>" + sessionStorage.getItem("resultado") + "<b>";
41         sessionStorage.removeItem("resultado"); // remove objeto do sessionStorage
42       }
43     }
44   </script>
45 </body>
46
47 </html>
```

Formulários



Aplicação prática

- Antes de executar a página **js38-cadastrar-mensagem.html** precisaremos criar no arquivo **js38.js** as funções **validarCampos** e **limparCampos** que ainda não existem no arquivo. Veja que a função **validarCampos** realiza todas as críticas dos campos do formulário e, se der erro, retorna **false** para que o formulário não seja submetido. Caso não haja erro no preenchimento, será retornado neste caso **true** para que o formulário seja submetido. A variável **mensagem** acumula as mensagens de erro para que o usuário receba todas as mensagens de erro de uma única vez no campo **txtMensagem** do formulário. Inclua as funções citadas no arquivo **js38.js**.

```
23 const validarCampos = () => {  
24     let mensagem, erro;  
25     mensagem = "";  
26     erro = false;  
27     if (document.form1.txtTitulo.value == "") {  
28         mensagem = "<b>Preencher o campo título<br></b>";  
29         erro = true;  
30     }  
31     if (document.form1.txtAssunto.value == "") {  
32         mensagem += "<b>Preencher o campo assunto<br></b>";  
33         erro = true;  
34     }  
35     if (document.form1.selCategoria.value == "") {  
36         mensagem += "<b>Selecionar o campo categoria<br></b>";  
37         erro = true;  
38     }  
39     document.form1.txtTitulo.focus(); // coloca o foco no campo título  
40     if (erro) { // se der erro  
41         document.getElementById('txtMensagem').innerHTML = mensagem;  
42         return false; // não submete o formulário  
43     } else {  
44         return true; // submete o formulário  
45     }  
46 }  
47 const limparCampos = () => {  
48     document.form1.txtTitulo.value = "";  
49     document.form1.txtAssunto.value = "";  
50     document.form1.selCategoria.value = "";  
51     document.form1.chkEmail.checked = false;  
52     document.getElementById("txtMensagem").innerText = "";  
53     document.form1.txtTitulo.focus();  
54 }  
55 }
```

Formulários



Aplicação prática

- Ainda antes de executar a página **js38-cadastrar-mensagem.html** precisaremos criar a página **js38-salvar-mensagem.html** que é a responsável por receber em sua URL os campos do formulário identificados com o atributo **name** e salvá-los no **localStorage** através do método **salvarMensagem** da classe **Mensagem**. Os campos passados pela URL são lidos através do método **searchParams** do objeto URL. O objeto URL recebe a url completa da página através de **location.href**. Veja que é salva na sessão do usuário a mensagem “**Gravação realizada com sucesso!**” ou a mensagem “**Limite de mensagens atingido!**” para que esta informação possa ser lida pela página **js38-cadastrar-mensagem.html** quando do retorno a ela. Conforme já vimos, estas mensagens salvas no **sessionStorage** são perdidas quando do fechamento do formulário. As mensagens são lidas na página **js38-cadastrar-mensagem.html** através da função **verificaGravacao** que é chamada no evento **onload** da página. Agora pode executar a página **js38.html** e escolher a opção “**Cadastrar mensagem**”. Teste a gravação com sucesso e sem preencher alguns campos para ver as mensagens de erro. O campo “**Enviar por e-mail**” não é obrigatório.

```
<> js38-salvar-mensagem.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Salvar Mensagem</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js38.js"></script>
8      <meta name="viewport" content="width=device-width, initial-scale=1">
9  </head>
10
11 <body>
12     <script type="text/javascript">
13         let mensagens = new Array(); {
14             const mensagem = new Mensagem(); // Cria o objeto mensagem
15             mensagens = mensagem.lerMensagens(); // Lê as mensagens do localStorage
16         }
17         if (mensagens.length == 5) { // Verifica se o limite de mensagens foi atingido
18             sessionStorage.setItem("resultado", "Limite de mensagens atingido!");
19         } else {
20             const parametros = new URL(location.href).searchParams; // Busca os parâmetros da URL
21             const titulo = parametros.get('txtTitulo'); // Busca o parâmetro txtTitulo
22             const assunto = parametros.get('txtAssunto'); // Busca o parâmetro txtAssunto
23             const categoria = parametros.get('selCategoria'); // Busca o parâmetro selCategoria
24             const email = parametros.get('chkEmail'); // Busca o parâmetro chkEmail
25             const mensagem = new Mensagem(titulo, assunto, categoria, email); // Cria objeto mensagem
26             mensagem.salvarMensagem(mensagem); // Salva mensagem no localStorage
27             sessionStorage.setItem("resultado", "Gravação realizada com sucesso!");
28         }
29         history.back();
30     </script>
31 </body>
32
33 </html>
```

Formulários



Aplicação prática

- Após o cadastramento com sucesso de uma mensagem, entre pelo Chrome na opção **Mais Ferramentas -> Ferramenta do desenvolvedor**. Clique no menu **Aplicativo** (ou *Application* em inglês) para ver no meio da janela as informações contidas na **localStorage** (**Armazenamento local** em português). Não haverá informação na **sessionStorage** (**Armazenamento da sessão** em português) porque a mensagem nela contida é eliminada através do método **removeItem** do **sessionStorage** logo após a exibição da mensagem na página.

Título:

teste

Assunto:

teste

Categoria:

Dúvida

Enviar por e-mail:

☒

Gravação realizada com sucesso!

Limpar

Voltar

Salvar

Aplicativo

Manifesto

Service workers

Armazenamento

Armazenamento

Armazenamento local

file://

Armazenamento da sessão

file://

IndexedDB

Cookies

Tokens de estado particular

Grupos de interesse

Armazenamento compartilhado

Armazenamento em cache

Buckets de armazenamento

file://

Origem file://

Chave	Valor
mensagens	[{"titulo":"teste","assunto":"teste","categoria":"D","email":true}]

▼ [{"titulo": "teste", assunto: "teste", categoria: "D", email: true}]

▶ 0: {titulo: "teste", assunto: "teste", categoria: "D", email: true}

Formulários



Aplicação prática

- Agora vamos construir a página **js38-ler-mensagens.html** para listar as mensagens armazenadas no **localStorage**. Veja que a tabela no html (tag `<table>`) não possui as linhas do corpo da tabela (tag `<tbody>`), visto que as linhas e as células da tabela são criadas dinamicamente através respectivamente dos métodos **insertRow** e **insertCell**. As mensagens são lidas do **localStorage** através do método **lerMensagens** da classe **Mensagem** e armazenados no **array mensagens**. O array **mensagens** é lido através do método **foreach** que recebe como parâmetro uma função anônima. Essa função é executada para cada item do **array**, onde cada item será armazenado na variável **item** a cada iteração do loop. Agora pode executar a página **js38.html** e escolher a opção **“Ler mensagens”**.

```
<> js38-ler-mensagens.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5    <title>Ler mensagens</title>
6    <meta charset="utf-8">
7    <script type="text/javascript" src="js/js38.js"></script>
8    <meta name="viewport" content="width=device-width, initial-scale=1">
9  </head>
10
11  <body>
12    <table border="1">
13      <thead>
14        <tr><th>Título</th><th>Assunto</th><th>Categoria</th><th>Enviar e-mail</th></tr>
15      </thead>
16      <tbody id="tblBody"></tbody>
17    </table>
18    <p><button type="button" onclick="history.back()">Voltar</button></p>
19    <script type="text/javascript">
20      let corpoTabela = document.getElementById("tblBody"); // para manipular o body
21      const mensagem = new Mensagem(); // Para acessar a classe Mensagem
22      let mensagens = new Array(); // Para armazenar no array as mensagens do localStorage
23      mensagens = mensagem.lerMensagens(); // Lê as mensagens do localStorage
24      mensagens.forEach(function (item) { // ler cada elemento do array e armazenar na variável item
25        let linhaTabela = corpoTabela.insertRow(); // cria linha na tabela
26        let celulaTitulo = linhaTabela.insertCell(); // cria célula na linha
27        celulaTitulo.innerText = item.titulo; // Insere titulo na célula
28        let celulaAssunto = linhaTabela.insertCell(); // cria célula na linha
29        celulaAssunto.innerText = item.assunto; // Insere assunto na célula
30        let celulaCategoria = linhaTabela.insertCell(); // cria célula na linha
31        switch (item.categoria) {
32          case "D":
33            celulaCategoria.innerText = "Dúvida"; // Insere categoria na célula
34            break;
35          case "S":
36            celulaCategoria.innerText = "Sugestão"; // Insere categoria na célula
37            break;
38          case "R":
39            celulaCategoria.innerText = "Reclamação"; // Insere categoria na célula
40            break;
41          case "O":
42            celulaCategoria.innerText = "Outros"; // Insere categoria na célula
43            break;
44        }
45        let celulaEmail = linhaTabela.insertCell(); // cria célula na linha
46        if (item.email)
47          celulaEmail.innerText = "Sim"; // Insere email na célula
48        else
49          celulaEmail.innerText = "Não"; // Insere email na célula
50      });
51    </script>
52  </body>
53
54  </html>
```

Formulários



Aplicação prática

- Agora para finalizar criaremos a página **js38-apagar-mensagens.html** que será responsável por apagar as mensagens do **localStorage** através do método **removeItem**. Após criar a página, execute a página **js38.html** e escolha a opção **“Apagar mensagens”**. Depois entre na opção **“Ler mensagens”** para confirmar que as mensagens foram apagadas.

```
<> js38-apagar-mensagens.html > html > head > script
1  <!DOCTYPE html>
2  <html lang="pt-br">
3
4  <head>
5      <title>Apagar Mensagens</title>
6      <meta charset="utf-8">
7      <script type="text/javascript" src="js/js38.js"></script>
8      <meta name="viewport" content="width=device-width, initial-scale=1">
9  </head>
10
11 <body>
12     <script type="text/javascript">
13         localStorage.removeItem("mensagens");
14         alert("Mensagens apagadas com sucesso!");
15         history.back();
16     </script>
17 </body>
18
19 </html>
```


Formulários



Exercícios:

- 1) Criar página de login com usuário (e-mail) e senha e permitir que os usuários se cadastrem antes através de um link de “Não tenho cadastro”. A tela de cadastro deverá ter no mínimo 4 Campos a mais do que o e-mail e a senha do usuário, sendo um deles o nome do usuário. O nome do usuário logado deverá ser apresentado em todas as páginas do sistema após o login.

Após o login deverá ser apresentada uma página com um menu onde a primeira opção seja um cadastro de um item de livre escolha. a segunda opção a listagem desse cadastro, a terceira opção a limpeza do cadastro e a quarta opção deve encerrar o sistema retornando para a página de login. Uma vez retornado para o login não deverá ser possível retornar para o menu sem efetuar login novamente. Caso o navegador seja fechado as informações do usuário logado deverão ser perdidas automaticamente de forma que um novo login seja necessário para entrar no sistema. O cadastro do usuário deve ser mantido mesmo com o fechamento do navegador.

Para a limpeza do cadastro deve ser apresentada uma caixa de diálogo para confirmação ou não da limpeza. A limpeza só deve ser realizada com a confirmação do usuário.

Formulários



Exercícios:

A página de cadastro deverá permitir o cadastro de vários itens, sendo que deverão ser limitados em 10 itens. Esta página deverá ter botões para: gravar os campos, limpar o formulário, retornar para o menu principal e ir diretamente para a página da listagem. A gravação dos dados deve ficar persistida (mantida) mesmo que o navegador seja fechado. Todos os campos devem ser obrigatórios e validados via Javascript, sendo apresentadas mensagens de erro quando do não preenchimento de algum campos. A página de cadastro deve ter no mínimo 5 campos e deverá utilizar obrigatoriamente as tags input (type text), input (type checkbox), textarea, button, label e select.

A página de listagem deve apresentar os itens em forma de tabela e deve existir um botão ao final da listagem para retornar ao menu principal.

Para os usuários e os itens a serem cadastrados deverão ser criadas duas classes em um arquivo Javascript (.js) separado das demais páginas HTML. Todas as ações sobre estas classes deverão ser efetuadas através de propriedades e métodos contidos nestas classes. Este arquivo Javascript deverá estar em um subdiretório js.

Deverá ser utilizado obrigatoriamente formulários, sessionStorage e localStorage no desenvolvimento da aplicação.

A utilização de bootstrap nas páginas receberá pontuação extra.

AJAX



Conceito

- Significa ***Asynchronous JavaScript and XML*** ou JavaScript e XML (*eXtensible Markup Language*) assíncrono. O XML é uma linguagem de marcação extensível que é utilizada para transmitir dados pela internet.
- Os formulários quando são submetidos pelo meio tradicional aprendido até agora fazem chamadas à nova página de forma síncrona, ou seja, a página chamadora fica aguardando o retorno da página chamada para continuar o processamento, recarregando (*refresh*) a página chamadora quando do retorno da chamada. A utilização de AJAX nos permitirá solucionar estes problemas, visto que as instruções que usam essa tecnologia, quando iniciam um processo de busca de um recurso na rede, retornam uma promessa (*promise*) que é cumprida assim que a resposta estiver disponível. Assim, enquanto a resposta não estiver disponível, o código JavaScript continuará a ser executado normalmente sem a necessidade de ficar aguardando o retorno do servidor, por isso dizemos que a resposta dele é assíncrona.
- Com o AJAX as aplicações enviam e recebem dados em segundo plano, podendo ser utilizado para atualizar pequenas partes das páginas sem recarregá-las e, assim, sem afetar a experiência do usuário,

AJAX



fetch

- Durante um bom tempo a forma nativa (sem *framework* ou *bibliotecas*) para fazer chamadas AJAX no JavaScript era através do objeto **XMLHttpRequest** (https://www.w3schools.com/xml/xml_http.asp). A partir do ECMAScript6 (ES6) foi implementado o método global **fetch** (https://www.w3schools.com/jsref/api_fetch.asp) que é uma API - *Application Programming Interface* (Interface de Programação de Aplicação) que facilitou bastante este tipo de chamada. Entretanto, algumas desvantagens do **fetch** em relação ao **XMLHttpRequest** é a falta de suporte em navegadores antigos e o fato do **fetch** não tratar as mensagens de erro de forma tão detalhada.
- A título de conhecimento, no ECMAScript6 (ES6) foi implementado o objeto **promise** (https://www.w3schools.com/js/js_promise.asp) e no ECMAScript 2017 foram implementadas as palavras-chave **async** e **await** (https://www.w3schools.com/js/js_async.asp). Estas novas funcionalidades também podem ser utilizadas para criar chamadas AJAX.
- Segue um site que apresenta APIs gratuitas que podem ser utilizadas nos testes com chamadas AJAX:
<https://free-apis.github.io/#/>

Fetch

- A sintaxe deste método é:

```
fetch (<url>, <opções>)  
  .then (<resposta>)  
  .then(<dados>)  
  .catch(<erro>)  
  .finally(<atividade>);
```

Onde:

- ❑ **url**: é a string com o endereço da página a ser consultada.
- ❑ **opções**: são as opções da chamada. Pode ser, por exemplo, enviar o método da chamada (GET, POST, PUT ou DELETE), *headers*, *body*, etc. É opcional. Maiores informações no link <https://javascript.info/fetch-api>.
- ❑ **resposta**: bloco de código que faz o tratamento da resposta vinda do servidor. Recebe um objeto **Response** que possui algumas propriedades para tratar a resposta. Por exemplo, a propriedade **ok** retorna **true** se funcionou e **false** se não funcionou. Já a propriedade **status** retorna o código de retorno da requisição. Os dados pode ser retornados em diversos formatos, um deles seria em formato JSON através do método **json** e outro seria em formato texto através do método **text**.
- ❑ **dados**: bloco de código que faz o tratamento dos dados recebidos. As propriedades dependem dos dados recebidos. É opcional.
- ❑ **erro**: bloco de código que faz o tratamento do erro. Retorna um objeto **Error** que possui várias propriedades dentre elas a propriedade **message** para retornar o texto do erro em formato string. É opcional.
- ❑ **atividade**: atividade que será executada ao final da consulta. É opcional.

Aplicação prática

- Agora criaremos uma aplicação com os conceitos apresentados neste capítulo. Será criada uma aplicação que permita acessar a API do **ViaCep** (<https://viacep.com.br/>) para buscar informações sobre um cep digitado em um formulário. Veja que está sendo executada a função **pesquisarCep** no evento **onchange** de forma que qualquer alteração no campo force a execução da função. A função **pesquisarCep** executa a url da api do **ViaCep** através do método global **fetch**. Este método recebe a resposta da API na variável **resposta** e executa o método **json** para converter a resposta para o formato **json**. Os dados são recebidos na variável **dados** que se estiver preenchida chamará a função **preencherCampos** para preencher os campos do formulário. Caso ocorra algum erro na chamada da API, o método **catch** apresentará uma mensagem de alerta com a mensagem do erro. Ao final da chamada, será gerado um log no console do navegador para futura conferência.
- Crie o arquivo **js39.html** conforme apresentado abaixo e execute-o para ver o resultado. Informe um cep com o traço.

```
<? js39.html > <html>
2  <html lang="pt-br">
3
4  <head>
5    <title>AJAX</title>
6    <meta charset="utf-8">
7  </head>
8
9  <body>
10   <form name="form1">
11     <p><label>CEP: </label><input type="text" id="cep" name="cep" onchange="pesquisarCep(document.form1.cep.value)"></p>
12     <p><label>Rua: </label><input type="text" id="rua" name="rua"></p>
13     <p><label>Complemento: </label><input type="text" id="complemento" name="complemento"></p>
14     <p><label>Bairro: </label><input type="text" id="bairro" name="bairro"></p>
15     <p><label>Cidade: </label><input type="text" id="cidade" name="cidade"></p>
16     <p><label>Estado: </label><input type="text" id="estado" name="estado"></p>
17     <p><button type="reset" id="btnLimpar" name="btnLimpar">Limpar</button></p>
18   <script type="text/javascript">
19     pesquisarCep = (cep) => {
20       const url = `https://viacep.com.br/ws/${cep}/json/`;
21       fetch(url)
22         .then(resposta => {
23           return resposta.json();
24         })
25         .then(dados => {
26           if (dados.erro == undefined)
27             preencherCampos(dados);
28           else {
29             alert("Cep não encontrado!");
30             document.form1.reset();
31           }
32         })
33         .catch(erro => {
34           alert("Erro na pesquisa do cep: " + erro.message);
35         })
36         .finally(console.log("Terminou a consulta"))
37     };
38
39     preencherCampos = (dados) => {
40       document.form1.rua.value = dados.logradouro;
41       document.form1.complemento.value = dados.complemento;
42       document.form1.bairro.value = dados.bairro;
43       document.form1.cidade.value = dados.localidade;
44       document.form1.estado.value = dados.uf;
45     }
46   </script>
47 </form>
48 </body>
49 </html>
```


Aplicação prática

- Agora vamos criar uma outra aplicação com os conceitos apresentados neste capítulo. Será criada uma aplicação que permita acessar a API da FIPE para consulta de marcas de carros (<https://parallelum.com.br/fipe/api/v1/carros/marcas>) para buscar informações sobre o código de um carro digitado em um formulário. Veja que está sendo executada a função **buscarCarro** no evento **onchange** de forma que qualquer alteração no campo **código** force a execução da função. A função **buscarCarro** executa a url da API da **FIPE** através do método global **fetch**. Este método recebe a resposta da API na variável **resposta** e executa o método **json** para converter a resposta para o formato **json**. Os dados são recebidos na variável **dados** que se estiver preenchida fará a pesquisa do código na lista de objetos retornado pela API para preencher os campos do formulário. Caso ocorra algum erro na chamada da API, o método **catch** apresentará uma mensagem de alerta com a mensagem do erro. Ao final da chamada, será gerado um log no console do navegador para futura conferência.
- Crie o arquivo **js40.html** conforme apresentado abaixo e execute-o para ver o resultado. Informe um código numérico.

```
<? js40.html > <? html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4    <title>AJAX</title>
5    <meta charset="utf-8">
6  </head>
7  <body>
8    <form name="form1">
9      <p><label>Código: </label><input type="number" id="codigo" name="codigo"
10        onchange="buscarCarro(document.form1.codigo.value)" autofocus></p>
11      <p><label>Descrição: </label><input type="text" id="descricao" name="descricao"></p>
12      <p><button type="reset" id="btnLimpar">Limpar</button></p>
13      <script type="text/javascript">
14        buscarCarro = (codigo) => {
15          if (document.form1.codigo.value != "") { // Só procurar se código preenchido
16            const url = "https://parallelum.com.br/fipe/api/v1/carros/marcas";
17            const opcoes = {
18              method: "GET",
19              headers: { "Content-Type": "application/json;charset=UTF-8" }
20            };
21          };
22          fetch(url, opcoes) // Funciona sem as opções
23            .then(resposta => {
24              return resposta.json();
25            })
26            .then(dados => {
27              let carroEncontrado = false;
28              if (dados != null) {
29                for (var x = 0; x < dados.length; x++) { // Percorre cada item da lista de objetos
30                  if (dados[x].codigo === document.form1.codigo.value) { // Verifica cada item da lista
31                    preencherCampos(dados[x]); // Envia o item inteiro da lista se encontrar o código
32                    carroEncontrado = true;
33                    break; // Sai do loop porque já encontrou o código
34                  }
35                }
36              }
37              if (!carroEncontrado) {
38                document.form1.descricao.value = ""; // Limpa descrição
39                alert("Código não encontrado!");
40              }
41            })
42            .catch(erro => {
43              alert("Erro na pesquisa do carro: " + erro.message);
44            })
45            .finally(console.log("Terminou a consulta")
46              );
47          }
48        }
49        preencherCampos = (carro) => {
50          document.form1.descricao.value = carro.nome;
51        }
52      </script>
53    </form>
54  </body>
55  </html>
```


AJAX



Exercícios:

- 1) Implemente alguma funcionalidade AJAX no site desenvolvido no exercício do capítulo anterior.

Outros assuntos



Outros assuntos



Bibliotecas

- Para facilitar o desenvolvimento de aplicações com JavaScript alguns desenvolvedores criaram as chamadas bibliotecas. Elas são códigos desenvolvidos para resolver problemas específicos e que permitem você manter total controle para decidir a lógica da sua aplicação. Seguem algumas bibliotecas JavaScript:
 - ☐ jQuery (<http://jquery.com/>)
 - ☐ React.js (<https://react.dev/>)
 - ☐ Anime.js (<https://animejs.com/>)
 - ☐ D3.JS (<https://d3js.org/>)

Outros assuntos



Frameworks

- Para agilizar o desenvolvimento de aplicações com JavaScript alguns desenvolvedores criaram os chamados frameworks. Eles são códigos que permitem o desenvolvimento de aplicações com pouco código, mas você passará a ter controle limitado da sua aplicação, visto que o framework fará boa parte do trabalho por você. Segue uma lista de frameworks JavaScript:
 - ☐ AngularJS (<https://angularjs.org/>)
 - ☐ Bootstrap (<https://getbootstrap.com/>)
 - ☐ Vue.js (<https://vuejs.org/>)
 - ☐ Node.js (<https://nodejs.org/en/>)
 - ☐ Next.js (<https://nextjs.org/>)

Outros assuntos



Validadores W3C

- O consórcio W3C (World Wide Web Consortium) criou validadores gratuitos para verificar se as páginas desenvolvidas estão dentro dos critérios definidos por eles, o que facilita a manutenção do seu projeto e ainda contribui para o melhor posicionamento do site nos buscadores. Seguem alguns validadores:
 - ☐ Validador HTML (<https://validator.w3.org/>)
 - ☐ Validador CSS (<http://jigsaw.w3.org/css-validator>)
 - ☐ Validador de links (<https://validator.w3.org/checklink>)
- A lista completa de validadores pode ser obtida no link abaixo:
<https://www.w3.org/developers/tools/>

Outros assuntos



Exercícios:

- 1) Qual a diferença entre uma biblioteca e um framework JavaScript?
- 2) Para que servem os validadores W3C?

Referências



Referências

- JavaScript Tutorial. Disponível em: <https://www.w3schools.com/js/>. Acesso em: 5 mar. 2024.
- SILVA, Maurício Samy. JavaScript Essencial – Guia Prático para Estudantes. ed. São Paulo: Novatec, 2024.
- IEPSEN, Edécio Fernando. Lógica de Programação e Algoritmos com JavaScript. ed. São Paulo: Novatec, 2022.

