

Programação Web

- ✓ JavaScript
- ✓ Objetos e Função construtora
- ✓ Módulos
- ✓ Classes
- ✓ jQuery
- ✓ Cookies
- ✓ Web Storage
- ✓ Exemplos
- ✓ Exercícios

Até este ponto, trabalhamos com diversos tipos de variáveis, como inteiras, reais, lógicas e strings.

Imagine que você precisa representar uma bolinha para um jogo em JavaScript, esta bolinha deve ter os seguintes dados: cor, posição x, posição y, raio. Quais variáveis seriam criadas?

Ex:

```
let cor = "red";    let x = 10;    let y = 20;    let raio = 10;
```

Perceba que os dados ficam “soltos”, uma alternativa para representar esta bolinha no código é pensar nela como um objeto. Os objetos em JS básico são criados com **{...}** e dentro das chaves colocamos nossas propriedades que representam o objeto que queremos criar.

Ex:

```
let bolinha = {  
  cor: "red",  
  x: 10,  
  y: 20,  
  raio: 10  
}
```

Cada propriedade é composta por **chave: valor**

Para acessar os dados do objeto, usamos a sintaxe:

objeto.propriedade

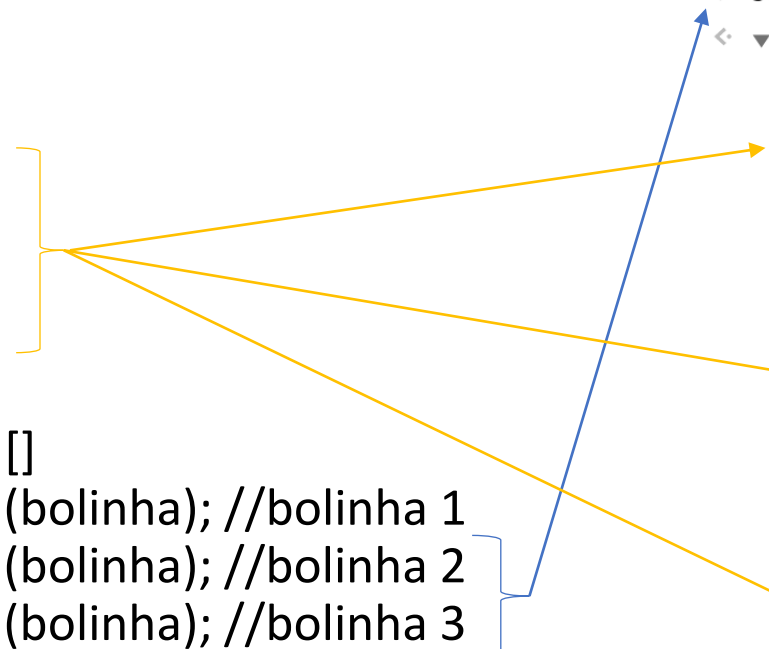
Ex: bolinha.cor ou ainda bolinha.cor="blue"

Podemos usar objetos de forma simples ou então armazenados em um vetor, ou seja, podemos ter um vetor de objetos, imagine o exemplo anterior, podemos ter um vetor de bolinhas.

Ex:

```
let bolinha={  
  cor: "red",  
  x: 10,  
  y: 10,  
  raio: 20  
}
```

```
let bolinhas = []  
bolinhas.push(bolinha); //bolinha 1  
bolinhas.push(bolinha); //bolinha 2  
bolinhas.push(bolinha); //bolinha 3
```



```
> bolinhas  
< ▼ (3) [{...}, {...}, {...}] ⓘ  
  ▼ 0:  
    cor: "red"  
    raio: 20  
    x: 10  
    y: 10  
    ► __proto__: Object  
  ▼ 1:  
    cor: "red"  
    raio: 20  
    x: 10  
    y: 10  
    ► __proto__: Object  
  ▼ 2:  
    cor: "red"  
    raio: 20  
    x: 10  
    y: 10  
    ► __proto__: Object
```

Para acessar os dados do objeto dentro do vetor:

vetor[índice].propriedade

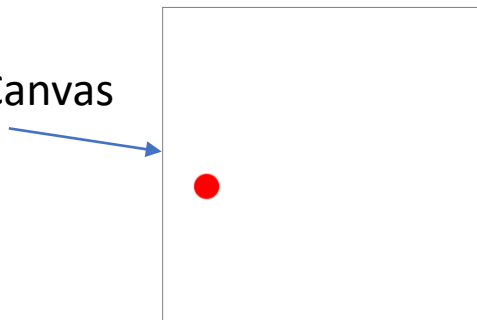
Ex:

`bolinhas[0].cor`

ou

`bolinhas[0].cor = "blue"`

Tag Canvas



```
let bolinha={  
  cor: "red",  
  x: Math.round(Math.random()*500),  
  y: Math.round(Math.random()*500),  
  raio: 20,  
}
```

exemplo1.html

Nome: Fulano da Silva, Idade: 32

Nome: Maria da Silva, Idade: 25

```
let pessoa1={  
  nome: "Fulano da Silva",  
  altura: 1.8,  
  idade: 32  
};  
...  
let pessoas = [pessoa1, pessoa2];  
...
```

```
for (let p of pessoas){  
  campo.innerHTML += `

# 

}
```

ou

```
for (let i=0; i < pessoas.length; i++){  
  campo.innerHTML += `

# 

}
```

exemplo2.html

No exemplo anterior, criamos um único objeto com seus valores iguais e fixos, no entanto, é comum precisarmos criar diversos objetos com valores de propriedades distintos, para isso, criamos uma função construtora de objetos (isso foi feito até o ES5 e ainda é correto).

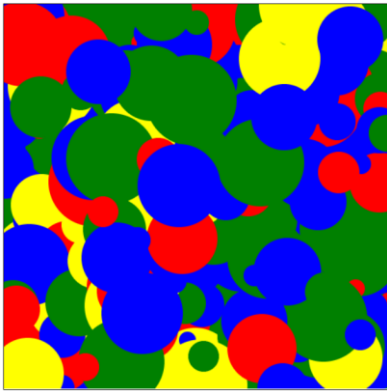
```
function Bolinha(cor,x,y,raio){  
  this.cor = cor;  
  this.x = x;  
  this.y = y;  
  this.raio = raio;  
  this.exibir = function(){  
    console.log("Cor da bolinha: ", this.cor);  
  }  
}
```

A palavra `this` define as propriedades do objeto que está sendo criado.

Podemos criar métodos (ações) para nossos objetos declarando funções no objeto.

```
let bolinha1 = new Bolinha("red",10,10,20);  
let bolinha2 = new Bolinha("blue",20,20,50);  
bolinha1.exibir();
```

Cada objeto poderá chamar seus métodos.



```
function Bolinha(cor,x,y,raio){  
    this.cor = cor;  
    this.x = x;  
    this.y = y;  
    this.raio = raio;  
    this.exibir = function(){  
        console.log("Cor da bolinha: ", this.cor);  
    }  
}
```

```
for (let i=1; i <= qtd_bolinhas; i++){  
    let bola = new Bolinha(cores[Math.round(Math.random()*(cores.length-1))],  
        Math.round(Math.random()*500),  
        Math.round(Math.random()*500),  
        Math.round(Math.random()*50)+10);  
    vet_bolinhas.push(bola);  
}
```

exemplo3.html

Nossos códigos e aplicações vão crescer naturalmente e sentiremos a necessidade de trabalhar com vários arquivos .js, isso é comum e necessário para separarmos as funcionalidades.

No JS tradicional, podemos carregar diversos arquivos externos diretamente no HTML, contudo, **a partir do ES6 temos disponível a opção de exportação e importação dos nossos códigos.**

Isso facilita muito, mas tome cuidado com a compatibilidade nos navegadores antigos.

Um módulo é um arquivo, logo, um script.js é um módulo em nossa aplicação.

Usamos duas palavras chaves para trabalhar com módulos:

- **export** : palavra-chave para usar antes de classes, funções ou variáveis que queremos exportar.
- **import** : palavra-chave usada para importar as funcionalidades de outros módulos.

Precisamos informar no bloco de script que o arquivo é um módulo colocando type="module".

Ex: `<script type="module" src="script.js"></script>`



OBS: Esta funcionalidade só funciona por meio do HTTP, logo nossa aplicação deve estar em um servidor ou podemos usar o Live Server do Visual Studio Code.

O caminho do arquivo deve sempre existir e respeitar o caminho relativo em nosso projeto.

As importações se comportam como constantes, para alterar variáveis por exemplo, somente métodos do módulo original podem alterar as suas variáveis.



```
export let titulo = 'Módulo ES6';
```

```
export let user={  
  name: "admin",  
  level: 3,  
  date: "12/12/2002"  
}
```

dados.js

```
export function setTitulo(t) {  
  titulo = t;  
}
```

```
export function exibirTitulo(){  
  alert(titulo);  
}
```

```
export function exibirUser(campo){  
  campo.innerHTML = `User: ${user.name}...`;  
}
```

```
...  
<body>  
  <div id="saida"></div>  
  <script type="module" src="js/script4.js"></script>  
</body>  
...
```

exemplo4.html



```
import { titulo, user, setTitulo, exibirTitulo, exibirUser }  
  from './dados.js';
```

```
let campo = document.querySelector("#saida");
```

```
alert(titulo);  
setTitulo("Título alterado");  
exibirTitulo();  
console.log(user);  
exibirUser(campo);
```

script4.js

OBS: Podemos importar todo o conteúdo do módulo (que possui o export) usando: **import * as nome from 'arquivo_modulo.js';**

O ES6 introduziu a possibilidade de criarmos classes nos script, esta nova sintaxe utiliza a palavra chave `class` e traz os conceitos básicos de orientação a objetos como herança (`extends`) e encapsulamento (`_` para dados protegidos e `#` para dados privados).

Vamos ver uma introdução neste material.

Para JS, uma classe não passa de uma função.

A classe é um tipo de molde que nos permite criar objetos ao qual passamos diferentes valores para seus atributos.

O nome configurado no `set` e `get` não pode ser igual ao da propriedade propriamente dito.

```
class NomeDaClasse {  
    constructor(...) { // construtor  
        //propriedades  
    }  
    metodos(...) {...} // métodos da classe  
    get identificador_prop() {...} // método get  
    set identificador_prop(...) {...} // método set  
}
```



```
export class Bolinha {  
  constructor(cor,x,y,raio) {  
    this.cor = cor;  
    this.x = x;  
    this.y = y;  
    this.raio = raio;  
  }  
  get bcor() { return this.cor; }  
  set bcor(valor) { this.cor = valor; }  
}
```

```
<body>  
  <script src="js/Pessoa.js"></script>  
  <script type="module" src="js/script5.js"></script>  
</body>
```

exemplo5.html

```
class Pessoa {  
  constructor(nome, idade, altura) {  
    this.nome = nome;  
    this.idade = idade;  
    this.altura = altura;  
  }  
  toString(){  
    return `Nome: ${this.nome}, Idade: ${this.idade}, ...`;  
  }  
}
```

Pessoa.js

```
import { Bolinha } from './Bolinha.js';  
  
let bola = new Bolinha("blue",10,20,30);  
alert (bola.bcor); // get bcor, retorna a cor da bola  
bola.bcor = "red"; //set bcor, configura a cor da bola  
  
console.log(bola);  
  
//a classe Pessoa foi carregada no próprio HTML  
let p = new Pessoa("Alcides", 40, 1.69);  
console.log(p.toString());
```

script28.js

jQuery é uma biblioteca JavaScript gratuita, que simplifica a programação JavaScript, desenvolvida e mantida pela jQuery Foundation.

Antes de utilizar o jQuery (ou qualquer pacote de Código) devemos ter um conhecimento básico das tecnologias abaixo:

- HTML
- CSS
- JavaScript

Para utilizar o jQuery devemos basicamente adicionar o arquivo js da biblioteca nos nossos códigos HTML com a tag a seguir:

```
<script src="jquery.js"></script>  
ou  
<script src="jquery.min.js"></script>
```

Para baixar o jQuery acesse <http://jquery.com/>

Outra opção é carregarmos o arquivo de um servidor que hospede o jQuery (CDN - Content Delivery Network), como por exemplo

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

A documentação da API da biblioteca poderá ser consultada em:

<http://api.jquery.com/> <http://docs.jquery.com/>

Quando baixamos o jquery, o nome do arquivo vem estipulando a versão (ex: **jquery-3.6.1.min.js**), porém com o intuito de facilitar a manutenção posterior do código para novas versões, iremos renomear o arquivo para:

jquery.min.js

Assim não teremos que alterar nossas páginas quando baixarmos novas atualizações.

Um conceito inicial do jQuery diz respeito aos seletores que podemos trabalhar. Esses seletores são semelhantes aos seletores do CSS, os principais são:

Seletor único

- CSS:
#tudo
- jQuery:
`$("#umid")`

Seletor classe

- CSS:
.estrutura
- jQuery
`$(".nomeclasse")`

Seletor tipo

- CSS:
p
- jQuery
`$("umatag")`

\$: função, é possível
selecionar elementos

```
graph TD; A["Seletor único  
• CSS: #tudo  
• jQuery: $("#umid")"] --> D["$: função, é possível  
selecionar elementos"]; B["Seletor classe  
• CSS: .estrutura  
• jQuery: $(".nomeclasse")"] --> D; C["Seletor tipo  
• CSS: p  
• jQuery: $("umatag")"] --> D;
```

Se precisarmos verificar o carregamento completo da página para executar no código, podemos usar as instruções a seguir. Essa verificação geralmente é necessária quando colocamos o script no Head.

Em JS puro

```
window.addEventListener("load",function(){  
    ...  
})
```

exemplo6.html

Em jQuery

Usaremos esse formato



```
$( function(){  
    ....  
} );
```

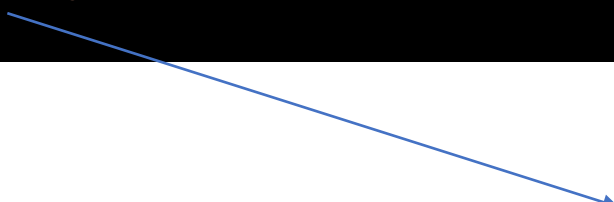
exemplo7.html



http://www.w3schools.com/jquery/jquery_events.asp


```
<body>
  <h2>Cabeçalho</h2>
  <p>Lorem ipsum ...</p>
  <p>...!</p>
  <button id="btn1">Clique aqui para ocultar as tags P</button>
  <button id="btn2">Clique aqui para mostrar as tags P</button>

  <script src="js/jquery.min.js"></script>
  <script src="js/script8.js"></script>
</body>
```



```
$("#btn1").click(function() {
    $("p").hide();
});

$("#btn2").click(function() {
    $("p").show();
});
```

Altera ou retorna o conteúdo HTML:

- **Sintaxe:**

- `$(seletor).html("novo conteúdo");`
- `var conteudo = $(seletor).html();`

Altera ou retorna o texto do elemento HTML:

exemplo9.html

- **Sintaxe:**

- `$(seletor).text("Texto");`
- `var conteúdo = $(seletor).text();`

Adiciona conteúdo HTML:

- **Sintaxe:**

- `$(seletor).append("novo conteúdo");`
- `$(seletor).prepend("novo conteúdo");`

- Limpar conteúdo / eliminar:

- **Sintaxe:**

- `$(seletor).empty(); //limpar`
 - `$(seletor).remove(); //eliminar`

Adicionar conteúdo no início de um elemento HTML:

exemplo9.html

- **Sintaxe:**

- `$(selector).prepend("novo conteúdo");`

Limpa conteúdo:

- **Sintaxe:**

- `$(selector).empty();`

Remove o elemento:

- **Sintaxe:**

- `$(selector).remove();`

Comparando...

O método `.text()` retorna os textos combinados de cada elemento no conjunto de elementos especificados pelo seletor, inclusive de seus descendentes. Não altera/retorna HTML, apenas texto.

Diferente do método `.html()`, o método `.text()` poderá ser utilizado em documentos XML e HTML.

O resultado do método `.text()` é uma string que contém os textos combinados de todos os elementos especificados pelo seletor.

O método `.text()` não pode ser usado em elementos input de formulários, nem com elementos scripts.

Para alterar ou pegar valores de elementos input de formulários ou textarea, utilize o método `.val()`.

Para alterar ou pegar valores de elementos script, utilize o método `.html()`.

Podemos facilmente manipular propriedades do CSS através do jQuery

- **`$('elemento').css(nomeDaPropriedade)`**
este comando retorna o valor da propriedade
- **`$('elemento').css(nomeDaPropriedade, valor)`**
este comando altera o valor da propriedade do elemento
- **`$('elemento').css({propriedade: valor, propriedade: valor,...})`**
este comando define múltiplos valores em múltiplas propriedades

Exemplos:

```
document.getElementById("btn").style.color="red";
```

Em JavaScript
puro



```
$('#btn').css("color","red");
```

Em JQuery

Na documentação podemos encontrar diversos métodos interessantes, como por exemplo o método `slideToggle()` que exhibe ou oculta um conteúdo com um *sliding motion*.

```
<script src="js/jquery.min.js"></script>
<script>
    $(function() {
        $("#bt").click(mostraDados);
    });
    function mostraDados() {
        $("#res").append( // append adiciona; usar html() para alterar
            "<span>Nome: </span>" + $("#nome").val() +
            "<br><span>E-mail: </span>" + $("#email").val() + "<br><br>");
    }
</script>
</head>

<body>
    <form action="" method="post">
        Nome:
        <input type="text" name="nome" id="nome">
        <br> E-mail: <input type="text" name="email" id="email">
        <input type="button" name="bt" id="bt" value="Mostrar">
    </form>
    <article id="res"></article>
</body>
```

```
<script>
$(document).ready(function(){
    $("p").on({
        mouseenter: function(){
            $(this).css("background-color", "lightgray");
        },
        mouseleave: function(){
            $(this).css("background-color", "lightblue");
        },
        click: function(){
            $(this).css("background-color", "yellow");
        }
    });
});
```

Click or move the mouse pointer over this paragraph.

Click or move the mouse pointer over this paragraph.

Click or move the mouse pointer over this paragraph.

Para manipular dados de formulários usamos a função:

`.val()` // equivalente à propriedade `value` do JS básico

Para capturar o valor do campo

Ex: `var x = $("#campo").val();`

Para atribuir um valor ao campo

Ex: `$("#campo").val("Valor que queremos atribuir");`

exemplo11.html

exemplo12.html

Basicamente temos três formas de armazenar dados no cliente com JS puro e que são suportadas pelos navegadores mais conhecidos e atuais:

- Cookies (modelo bem antigo de armazenamento de dados no cliente);
- API Web Storage (API disponibilizada com o surgimento do HTML5).
- IndexedDB (destinado a aplicativos offline mais robustos)
 - <https://www.w3.org/TR/IndexedDB/>

Embora a API Web Storage tenha diversas vantagens sobre o uso de cookies, iremos abordar os dois conteúdos, visto que também podemos manipular cookies em linguagens server-side e muitos projetos utilizam esse recurso.

São informações gravadas por uma aplicação web na máquina cliente. Para cada domínio da aplicação é gerado um arquivo txt que contém os cookies.

Os cookies são formados por um par composto de:

- Nome do cookie
- Valor do cookie

Os cookies são mantidos na máquina cliente enquanto seu prazo de "vida" não expirar ou enquanto o usuário não limpa-os através do navegador.

Geralmente um cookie não pode ultrapassar 4Kb cada. O número total de cookies, por domínio, que podemos gravar no navegador vai depender de qual estamos utilizando. Alguns chegam a permitir na faixa de 80+-.

Para gravar algumas informações que posteriormente poderemos usar em nossa aplicação Web.

Vários sites com publicidade gravam informações para posterior consulta, isso pode mudar devido a lei de privacidade.

Em um exemplo clássico, podemos trabalhar com os cookies em um e-commerce, enquanto o usuário está fazendo a escolha dos pedidos e colocando-os no carrinho de compra, podemos armazenar os dados dos produtos em cookies. Após a finalização da compra, gravamos os dados em um banco de dados e limpamos o cookie.

Qualquer site pode gravar cookies em sua máquina, desde que essa permissão esteja habilitada em seu navegador. Ou que você permita atualmente realizar esta gravação.

Um cookie não pode apagar arquivos em sua máquina e muito menos "roubar" suas senhas.

Um site só pode acessar os cookies gravados pelo seu domínio, ou seja, uma aplicação só pode acessar os cookies gravados por ela.

Um cookie, resumidamente, é composto por:

- Nome
- Valor
- Tempo (tempo de validade do cookie)

Podemos manipular os cookies em JavaScript através do comando:

- **document.cookie**

Um cookie é composto, detalhadamente, por:

NAME=Valor; EXPIRES=Data; PATH=Caminho; DOMAIN=Nome_Dominio;

NAME: é uma string de caracteres usada para identificar o cookie.

EXPIRES: é a data em que o cookie deve ser removido do arquivo de cookies se ele ainda estiver lá. Não sendo especificada nenhuma data, o cookie terá sua validade vencida assim que o usuário sair do browser e encerrar a sessão.

PATH: é o nome do caminho ou URL do documento que criou o cookie. (motivo: privacidade).

DOMAIN: é o nome completo do domínio do servidor (ou computador central) que criou o cookie.

Podemos manipular os cookies facilmente utilizando um plugin para JS ou mesmo para o jQuery.

Plugin cookie para o jQuery (arquivo *jquery.cookie.js*, usado nos exemplos)

<https://github.com/carhartl/jquery-cookie>

Plugin cookie para JS (arquivo *js.cookie.min.js*, usado nos exemplos)

<https://github.com/js-cookie/js-cookie>

Como utilizar o plugin para JS puro

- Para gravar um cookie

`Cookies.set('nome', 'valor')`

ou para definir configurações como expiração, domínio e segurança (sameSite)


`Cookies.set('nome' , 'valor' , conf)`

- Para recuperar os dados do cookie

`Cookies.get('nome')`

- Para apagar um cookie

`Cookies.remove('nome')`



```
let conf={  
  expires: 10,  
  sameSite: 'strict',  
  domain: ''  
};
```

exemplo13.html

RGM:

Nome:

Profissão:

CPF:

```
<script src="js/jquery.min.js"></script>
<script src="js/js.cookie.min.js"></script>
<script src="js/script13.js"></script>
```

```
function carregarCookie(){
    if (Cookies.get("rgm") != null){
        $("#rgm").val(Cookies.get("rgm"));
        $("#profissao").val(Cookies.get("profissao"));
        ...
    }else{
        alert("Nenhum Cookie encontrado");
    }
}
```

```
let conf={
    expires: 10,
    sameSite: 'strict',
    domain: ''
};
```

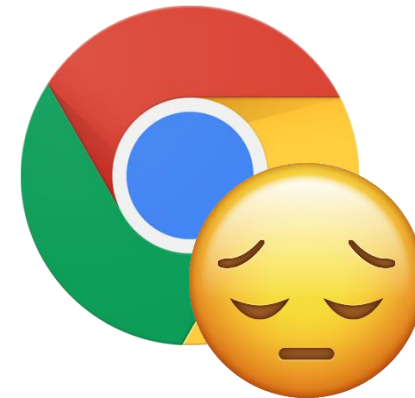
```
function gravarCookie(){
    Cookies.set("rgm",$("#rgm").val(),conf);
    Cookies.set("nome",$("#nome").val(),conf);
    ...
    alert("Dados gravados");
    limparCampos();
}
```

```
function apagarCookie(){
    if (Cookies.get("rgm") != null){
        Cookies.remove("rgm");
        Cookies.remove("profissao");
        ...
    }else{
        alert("Nenhum Cookie encontrado");
    }
    limparCampos();
}
```

Por questões de segurança o Chrome não permite gravar cookies no teste do arquivo local ou testando pelo servidor local do VS Code, para funcionar, deve ser hospedado em um servidor e se faz necessário configurar o domínio no objeto de configuração.

```
let conf={  
  expires: 10,  
  sameSite: 'strict',  
  domain: 'insira aqui o domínio'  
};
```

Localmente não roda



Para testar localmente, use o Firefox

Neste roda



Com essa API o desenvolvimento fica mais fácil e temos algumas vantagens sobre os cookies. A mais significativa seria a capacidade de armazenamento que varia de acordo com o navegador, de 10MB (Chrome).

A Web Storage oferece dois tipos de armazenamento: `sessionStorage` e `localStorage`.

- **`sessionStorage`**
disponível apenas para a janela (aba) que criou o dado até que esta seja fechada
- **`localStorage`**
compartilhada por todas as janelas (abas) abertas pela aplicação e os dados só são apagados caso sejam deletados pela aplicação ou pelo usuário através das opções do navegador.

Como usar (os métodos são os mesmos para os objetos sessionStorage e localStorage).

- Para gravar dados

- `setItem(nome, valor)`

Ex: `localStorage.setItem("nome", "Maria");`

- Para recuperar dados gravados

- `getItem(nome)`

Ex: `localStorage.getItem("nome")`

- Para apagar dados gravados

Ex: `localStorage.removeItem("nome")`

- Para apagar todos os dados do storage

- `clear()`

Ex: `localStorage.clear()`

Exemplo (localStorage)

RGM: Nome: Profissão: CPF:

```
function apagarDados() {  
    if (localStorage.length > 0) {  
        localStorage.clear();  
    } else {  
        alert("Nenhum registro encontrado");  
    }  
    limparCampos();  
}
```

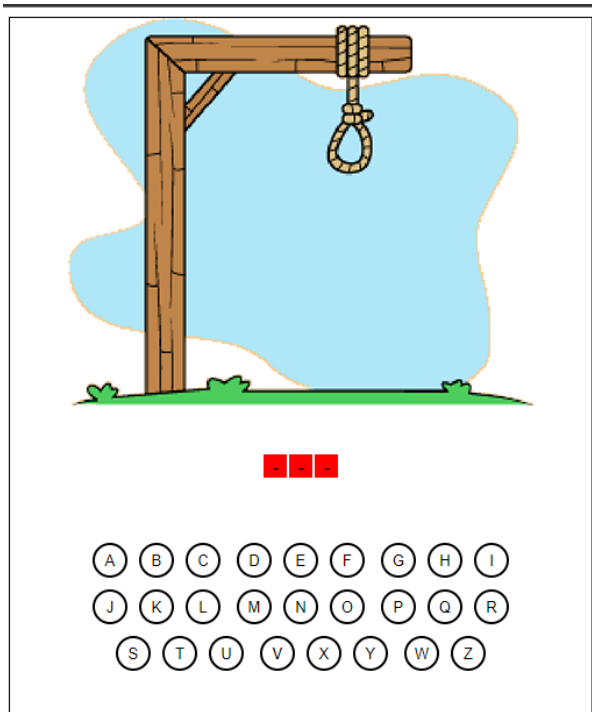
```
function carregarDados() {  
    if (localStorage.length > 0) {  
        $("#rgm").val(localStorage.getItem("rgm"));  
        $("#nome").val(localStorage.getItem("nome"));  
        ...  
    } else {  
        alert("Nenhum registro encontrado");  
    }  
}
```

```
function gravarDados() {  
    localStorage.setItem("rgm", $("#rgm").val());  
    localStorage.setItem("nome", $("#nome").val());  
    ...  
    alert("Dados gravados");  
    limparCampos();  
}
```

Uma forma simples de verificar se o navegador suporta localStorage seria com o bloco de if abaixo:

```
if (typeof(localStorage) == 'undefined' )
{
    alert("Seu navegador não suporta HTML5 localStorage.");
}
else
{
    //fazer o código para manipular o storage
}
```

Obs: Outra opção seria utilizar a biblioteca Modernizr; essa biblioteca testa todos os recursos do HTML5 e CSS3 (<http://modernizr.com/>).



Ping Pong

Pontos do Jogador A : 0
Pontos do Jogador B : 0



CSS3 Matching Game

