

# JavaScript

**Manipulando o DOM**

# Manipulando elementos

# Dando continuidade ao projeto TO-DO App

Já colocamos um texto na notificação de sucesso, mas na prática, o correto seria **só exibir esse componente** após o clique no botão “logar”.

**Podemos fazer isso de três formas:**



**Opção 1:** Utilizar o atributo *innerHTML* para criar um novo componente.



**Opção 2:** Criar um novo elemento com *createElement()* e neste momento adicionar o texto.



**Opção 3:** Manter a tag de notificação escondida com `display: none`. Quando o botão “logar” for clicado, alterá-la para `display: block`;

## Opção 1 (exe1.html)



**Opção 1:** Utilizar o atributo *innerHTML* para criar um novo componente.

A propriedade `innerHTML` pode ser utilizada para obter ou definir o conteúdo HTML interno de uma tag. É uma má prática (do ponto de vista de manutenção) colocar um trecho HTML inteiro dentro de aspas duplas ou simples para definir o HTML de um elemento.

```
var tagDiv = document.querySelector('div');  
tagDiv.innerHTML = "<span>Meu conteúdo legal...</span>";
```

# Opção 1 com Template Strings (exe2.html)



**Opção 1:** o uso de Template Strings (EcmaScript 6) torna a opção 1 uma boa alternativa.



Usar EcmaScript 6 sem um transpiler pode quebrar a aplicação no browser do usuário.

A diferença é que o Template Strings trabalha com crase (ao invés de aspas), aceita quebra de linhas e concatenação com variáveis de forma elegante.

```
tagDiv.innerHTML = `  
  <span>  
    Meu conteúdo legal...  
  </span>  
`;  
;
```

## Opção 2 (exe3.html)



**Opção 2:** Criar um novo elemento com *createElement()* e neste momento adicionar o texto.

Podemos criar elementos novos no DOM através do método *createElement()*. Depois de criado, podemos adicionar os elementos através do *appendChild()*.

```
var tagSpan = document.createElement('span');  
tagSpan.textContent = 'Meu conteúdo legal...';  
var tagDiv = document.querySelector('div');  
tagDiv.appendChild(tagSpan);
```



Cuidado que neste caso, a cada novo clique um novo `<span>` será criado e adicionado à `<div>`. Use *innerHTML = ""* para limpar a tag.

## Opção 3 (exe4.html)



**Opção 3:** Manter a tag de notificação escondida com `display: none`. Quando o botão “logar” for clicado, alterá-la para `display: block`;

```
var tagDiv = document.querySelector('div');  
var tagSpan = tagDiv.querySelector('span');  
tagSpan.textContent = 'Meu conteúdo legal...';  
tagDiv.style.display = 'block';
```

Note que podemos utilizar o método *querySelector()* a partir de outra tag (não só do *document*). Para alterar um atributo CSS do elemento, deve-se utilizar a propriedade *style* seguida do nome da propriedade.

## Opção 3 (exe4.html)

Neste caso, devemos trabalhar com o padrão **lowerCamelCase** para representar os atributos com duas ou mais palavras. Exemplos:

Sintaxe JavaScript	Sintaxe CSS
backgroundColor	background-color
boxSizing	box-sizing
fontSize	font-size
textShadow	text-shadow
fontStyle	font-style



## Opção 3 (exe5.html)



**Atenção:** é uma má prática alterar estilo da página diretamente pelo JavaScript.



A melhor opção é criar classe CSS e atribui-la às tags através da propriedade *classList*, presente em todas as tags.

```
var tagDiv = document.querySelector('div');  
var tagSpan = tagDiv.querySelector('span');  
tagSpan.textContent = 'Meu conteúdo legal...';  
tagSpan.classList.add('estilo-legal');  
tagDiv.classList.add('mostrar-div');
```

## Recursos interessantes (exe6.html e exe7.html)

<code>document.getElementById()</code>	Obtém um elemento pelo ID. Neste caso, não precisa colocar o símbolo #.
<code>tag.classList.contains('estilo')</code>	Verifica se a tag possui uma determinada classe CSS.
<code>tag.classList.remove('estilo')</code>	Remove uma classe da lista de classes da tag.
<code>tag.classList.toggle('estilo')</code>	Um método que facilita a inclusão ou retirada de uma classe da tag. Se a classe existir, ele tira. Se ela não existir, ele adiciona.

# Exercício (exe8/index.html)

Dando continuidade ao projeto, aplique uma propriedade *display: none* à mensagem de bem-vindo. Em seguida, quando o usuário logar na tela, exiba-a incluindo uma nova classe com a propriedade *display: block*

## Recursos interessantes (exe9.html e exe10.html)

<code>setTimeout(function(){},1000);</code>	Executa uma função após N milissegundos.
<code>setInterval(function(){},1000);</code>	Executa uma função indefinidamente a cada N milissegundos. Ela retorna o ID do intervalo, que pode ser guardado em uma variável para posterior interrupção via método <i>clearInterval()</i>
<code>clearInterval(id);</code>	Interrompe o <i>setInterval()</i> a partir de um ID passado como parâmetro.

# Exercício (exe8/index.html)

Crie um efeito de “desvanecer” para que a notificação suma da tela da aplicação após 4 segundos. **Dicas:**

- Utilize *@keyframes* para que a classe *desvanecer* vá de 1 a zero na propriedade *opacity*.
- Não é possível suavizar a transição da propriedade *display*. Para resolver isso, retire a propriedade *display: block* da notificação após 4 segundos com a função *setTimeout()*

# Validando dados

# Próximo passo do projeto TO-DO App

A partir de agora, vamos ajustar o HTML e o CSS do projeto para posteriormente manipular os elementos via JavaScript.

- Aplique um estilo com *display: none* nas notificações e na <div> que contém o formulário e as tarefas.
- No caso das tarefas, vamos criá-las inteiramente a partir do JavaScript. Por isso, apague-as do HTML.
- **Sugestão:** guarde uma delas em outro arquivo apenas para facilitar a criação do JavaScript depois.

## Validação simples do usuário (exe8/index.html)

Vamos verificar se o usuário clicou no botão “Logar” sem ter digitado. No evento de *click*, verifique se a propriedade *length* do conteúdo da tag `<input>` possui comprimento zero.

***length*** - uma propriedade que retorna o tamanho de uma string (ou array). Se o usuário não digitar nada, a string retornada será vazia.

```
if (nome.length == 0) {  
    // Nome inválido  
} else {  
    // Nome válido  
}
```



## Validação simples do usuário (exe8/index.html)

Se o nome estiver vazio, devemos exibir a notificação de falha informando isso ao usuário. Para manter o efeito de **desvanecer** aplicado à mensagem de sucesso, poderíamos simplesmente replicar parte do código do `setInterval()`.



Replicar código é sempre ruim!



Crie funções JavaScript para reutilizar trechos de código mais de uma vez em seu sistema.

Seguindo as boas práticas, ao invés de copiar o código, vamos refatorá-lo para dentro de uma função.

## Validação simples do usuário (exe8/index.html)

Crie a função notificar para encapsular o comportamento de exibir e desvanecer a mensagem de notificação.

```
function notificar(tag) {  
    tag.classList.add('visivel');  
    tag.classList.add('desvanecer');  
  
    setTimeout(function(){  
        tag.classList.remove('visivel');  
        tag.classList.remove('desvanecer');  
    },4000);  
}
```

## Alterando propriedades do HTML (exe11.html)

Além de manipular as propriedades de estilo e as classes dos elementos, às vezes é necessário manipular os atributos das tags HTML através do JavaScript.

Suponha uma situação onde após o clique, o usuário deve aguardar por uma ação do sistema. Isso pode ser feito, por exemplo, incluindo o atributo *disabled* às tags `<input>` e `<button>`.

## Utilizando o *this* (exe11.html)

Quando trabalhamos com eventos, o elemento no qual o método *addEventListener()* foi definido é automaticamente repassado à função de tratamento através da sintaxe *this*

```
var botao = document.querySelector('button');  
botao.addEventListener('click',function(){  
    console.log(this);  
});
```

Neste exemplo, o *this* faz referência ao elemento *botao*.

# Exercício (exe8/index.html)

Caso o usuário entre com um nome válido:

- Desabilite o `<input>` e o botão associados ao nome do usuário.
- Exiba a `<div>` principal com o formulário de cadastro de novas tarefas.

**Obtendo dados dos campos**

## Obtendo dados do <input> de texto (exe12.html)

Já vimos que para pegar o valor de tags <input> é necessário acessar a propriedade *value*.

### HTML:

```
<input id="nome" type="text" />
```

### JavaScript:

```
var nome = document.querySelector('#nome').value;
```

## Obtendo dados do <select> (exe12.html)

A tag <select> possui um conjunto de tags <option>:

```
<select id="empresa">  
  <option value="">Selecione...</option>  
  <option value="Google">Google</option>  
  <option value="Twitter">Twitter</option>  
</select>
```

A obtenção da tag <select> via *querySelector()* é idêntica, porém existem outras propriedades para obtenção da opção selecionada.

```
var tagSelect = document.querySelector('#empresa');
```



## Obtendo dados do <select> (exe12.html)

```
var tagSelect = document.querySelector('#empresa');
```

A partir do código acima, teremos acesso a:

<pre>var tagsOption = tagSelect.options;</pre>	Obtém todos os <option> internos da <select>
<pre>var index = tagSelect.selectedIndex;</pre>	Obtém o índice do <option> selecionado na <select>
<pre>var selecionada = tagsOption[index];</pre>	Com auxílio de ambos, é possível obter a <option> selecionada

## Obtendo dados do <select> (exe12.html)

Resumindo, com o código a seguir podemos obter o valor do <option> selecionado no <select>:

```
var tagSelect = document.querySelector('#empresa');  
var tagsOption = tagSelect.options;  
var index = tagSelect.selectedIndex;  
var empresaSelecionada = tagsOption[index];  
console.log('Empresa: ' + empresaSelecionada.value);
```

## Obtendo dados do <input> de radio (exe12.html)

O elemento <input> do tipo *radio button* pode ser obtido um a um através dos métodos *querySelector()*.

### HTML:

```
<input id="r1" name="meu-radio" type="radio" value="valor1" />
```

```
<input id="r2" name="meu-radio" type="radio" value="valor2" />
```

```
<input id="r3" name="meu-radio" type="radio" value="valor3" />
```

### JavaScript:

```
var r1 = document.querySelector('#r1');
```

```
var r2 = document.querySelector('#r2');
```

```
var r3 = document.querySelector('#r3');
```



Essa não é uma boa forma de obter os radio buttons!

## Obtendo dados do <input> de radio (exe12.html)



Como pode ser visto a seguir, a manutenibilidade ficaria ruim porque precisaríamos verificar a propriedade *checked* de cada elemento, um a um.

```
var valor = "";  
var r1 = document.querySelector('#r1');  
if (r1.checked) { valor = r1.value; }  
var r2 = document.querySelector('#r2');  
if (r2.checked) { valor = r2.value; }  
var r3 = document.querySelector('#r3');  
if (r3.checked) { valor = r3.value; }  
// Podem existir vários...
```

## Obtendo dados do <input> de radio (exe12.html)



Para resolver isso, podemos utilizar um *for* do JavaScript para percorrer uma lista de radio buttons.

### Mas como podemos obter vários elementos de uma vez?

```
var tagsFilhos = document.querySelectorAll('input[type=radio]');
```

Através do método *querySelectorAll()* nós podemos buscar todos os elementos que atendam ao seletor informado no parâmetro. No exemplo acima, a variável *tagsFilhos* conterá uma lista de `<input type="radio">`

## Obtendo dados do <input> de radio (exe12.html)

Variável utilizada para guardar o índice de cada iteração.

O **for** será executado enquanto **i** for menor que o tamanho (**length**) do array de radio buttons.

```
for (var i = 0; i < tagsFilhos.length; i++) {  
    if (tagsFilhos[i].checked) {  
        console.log('Valor selecionado: ' + tagsFilhos[i].value);  
    }  
}
```

Para cada iteração, **i** será acrescido em 1.

## Obtendo dados do <input> de radio (exe12.html)



Uma forma mais elegante é obter os elementos de um array através da sintaxe do *forEach*.

```
tagsFilhos.forEach(function(elemento, indice){  
    if (elemento.checked) {  
        console.log('Valor selecionado: ' + elemento.value);  
        console.log('Índice selecionado: ' + indice);  
    }  
});
```

O *forEach* deve receber uma *function* como parâmetro que, por sua vez, recebe o elemento e o índice da iteração como parâmetros.

# Obtendo dados de um formulário



## Obtendo dados de um <form> (exe13.html)

Até o momento nós obtemos o valor dos campos de entrada um a um, sem nenhum vínculo com a tag <form>.



Uma boa prática é encapsular os campos em uma tag <form> e escutar o evento *submit*.

```
var formulario = document.querySelector('form');  
formulario.addEventListener('submit',function(){  
    // Tratamento do evento submit  
});
```



Ao trocarmos o <main> pelo <form> e escutarmos o evento *submit* (ao invés de escutar o evento *click*), a função de tratamento parou de funcionar. **Por que?**

## Obtendo dados de um <form> (exe14.html)

Um botão exposto dentro de um formulário dispara **por padrão** o evento *submit*, que na prática envia dos dados coletados no formulário para o **back-end** especificado no atributo *action* do <form>.

No nosso caso, **não** queremos que o formulário seja enviado. Localmente vamos obter os dados para inclusão das tarefas. Para **prevenir** o comportamento padrão do *submit* precisamos chamar o método a seguir:

```
evento.preventDefault();
```

## Obtendo dados de um <form> (exe14.html)

### Mas como podemos obter o evento disparado?

O evento é informado automaticamente como parâmetro da função de tratamento do *addEventListener()*. Com ele nós podemos convocar o método *preventDefault()*, conforme o exemplo abaixo.

```
var formulario = document.querySelector('form');  
formulario.addEventListener('submit',function(evento){  
    evento.preventDefault();  
    console.log(evento); // Imprimindo evento no console...  
});
```

# Obtendo dados de um <form> (exe14.html)

Elements Console Sources Network Performance Memory Application Security Audit

top Filter Default levels ▾ ☒ Group similar

▼ Event {isTrusted: true, type: "submit", target: form, currentTarget: form, eventPhase: 2, ...} ⓘ

- bubbles: true
- cancelBubble: false
- cancelable: true
- composed: false
- currentTarget: null
- defaultPrevented: true
- eventPhase: 0
- isTrusted: true
- ▶ path: (5) [form, body, html, document, Window]
- returnValue: false
- ▶ srcElement: form
- ▶ target: form
- timestamp: 807.5999999998658
- type: "submit"
- ▶ \_\_proto\_\_: Event

▼ target: form

- ▶ 0: input#nome.itens-formulario
- ▶ 1: select#empresa.itens-formulario
- ▶ 2: input#filhos-sim
- ▶ 3: input#filhos-nao
- ▶ 4: input#termos
- ▶ 5: button

## Obtendo dados de um <form> (exe15.html)

Note que através do evento **submit** nós podemos obter o formulário enviado (através do atributo **target**) e, conseqüentemente, podemos acessar seus campos de entrada diretamente.

```
var formulario = document.querySelector('form');  
formulario.addEventListener('submit',function(evento){  
    evento.preventDefault();  
    console.log('Nome: ' + evento.target.nome.value);  
});
```



É uma boa prática obter os campos de entrada do formulário diretamente a partir do evento *submit*.

## Obtendo dados de um <form> (exe16.html)

Agora que obtemos todos os dados do formulário, podemos manipular o DOM para exibi-los ao usuário.

Podemos criar linhas <tr> através do método *createElement()* e adicioná-las a uma <table> através do método *appendChild()*

Nome:

Empresa:

Filhos? ☐ Sim ☒ Não

☒ Aceite dos termos

Nome	Empresa	Filhos?	Termos?
Guilherme	Google	Não	Sim

# Exercício (exe8/index.html)

Altere o projeto TO-DO App para:

- Capturar o evento *submit* do formulário de cadastro de novas tarefas e através dele obter os dados do formulário.
- Criar uma nova tarefa na seção apropriada do HTML com os métodos *createElement()* e *appendChild()*
- **Dica:** Envolver o texto da tarefa em um `<span>`.
- **Atenção:** veremos a sintaxe para obter a data e hora corrente em breve. Por enquanto, crie a tarefa **sem** essa informação.

# Organizando o código



# Trabalhando com datas (exe17.html)

O JavaScript oferece o objeto **Date()** para trabalhar com datas. Como trata-se de um objeto, precisamos instanciá-lo com a palavra **new**.

```
var d = new Date();
```

Depois de instanciado, ele nos oferece diversos métodos para se trabalhar com as datas. Sua interface não é muito agradável, mas é possível fazer muita coisa programaticamente.

Mais detalhes: [https://www.w3schools.com/jsref/jsref\\_obj\\_date.asp](https://www.w3schools.com/jsref/jsref_obj_date.asp)

# Trabalhando com MomentJS (exe18.html)

Uma biblioteca muito utilizada para facilitar a utilização de datas é o MomentJS, cujo endereço é: <https://momentjs.com/>

Através dela podemos facilmente obter uma data formatada.

```
moment().format('DD/MM/YYYY HH:mm:ss');
```

Para mais detalhes, veja: <https://momentjs.com/docs/#/get-set/>


# Trabalhando com objetos (exe19.html)

Até o momento nós temos guardado o valor obtido dos campos do formulário em variáveis isoladas. O ideal, neste caso, é agrupar todas as informações em um objeto JavaScript.

```
var cliente = {  
  nome: 'Guilherme',  
  empresa: 'fsma',  
  filhos: 'n',  
  termos: true  
};  
console.log(cliente);
```



Trabalhar com **Orientação a Objetos** é uma excelente prática em quase todas as situações.



```
▼ {nome: "Guilherme", emp  
  empresa: "fsma"  
  filhos: "n"  
  nome: "Guilherme"  
  termos: true  
  ▶ proto : Object
```

# Trabalhando com objetos e métodos (exe20.html)

```
var cliente = {  
    // Atributos do objeto  
  
    possuiFilhos: function() {  
        return (this.filhos) == 's' ? 'Sim' : 'Não';  
    },  
    aceitouTermos: function() {  
        return this.termos ? 'Sim' : 'Não';  
    }  
};
```



Outra excelente prática é **encapsular** lógica pertinente ao objeto em **métodos** que podem ser declarados no momento de sua criação.

Neste caso **this** faz referência ao próprio objeto.

## Organize o código com funções (exe21.html)



Por fim, organizar o código em funções é outra excelente prática que ajuda no reaproveitamento de código e na manutenibilidade do sistema. Quando for necessário, utilize também arquivos JavaScript diferentes.

Veja a seguir, por exemplo, um código que facilita a criação de colunas em uma tabela.

```
function renderizarColuna(tr,valor) {  
    var td = document.createElement('td');  
    td.textContent = valor;  
    tr.appendChild(td);  
}
```

# Exercício (exe8/index.html)

Altere o projeto TO-DO App para:

- Importar a biblioteca MomentJS para exibir a data em que a tarefa foi cadastrada.
- Guardar os valores da tarefa em um objeto.
- Criar funções para evitar duplicidade de código na renderização da tarefa no DOM.
- **Extra:** Após incluir a tarefa, apague os dados preenchidos no formulário com o comando *formulario.reset()*;

Obrigado!