

JavaScript

Aula 4

jQuery

Vanilla JavaScript vs jQuery

Código mais simples

Código mais limpo

Compatibilidade entre browsers

Elementos de UI sofisticados (jQuery UI)

Otimizado para UI mobile (jQuery Mobile)

Seletor CSS otimizado (Sizzle)

Testes em JavaScript (QUnit)

Código mais simples e Seletor CSS otimizado (Sizzle) (exe1.html)

```
/* Vanilla JavaScript */
```

```
var sectionVanilla =
```

```
document.querySelector('section.vanilla');
```

```
console.log(sectionVanilla.textContent);
```

```
/* jQuery */
```

```
console.log($('section.jquery').text());
```

Código mais limpo (exe2.html)

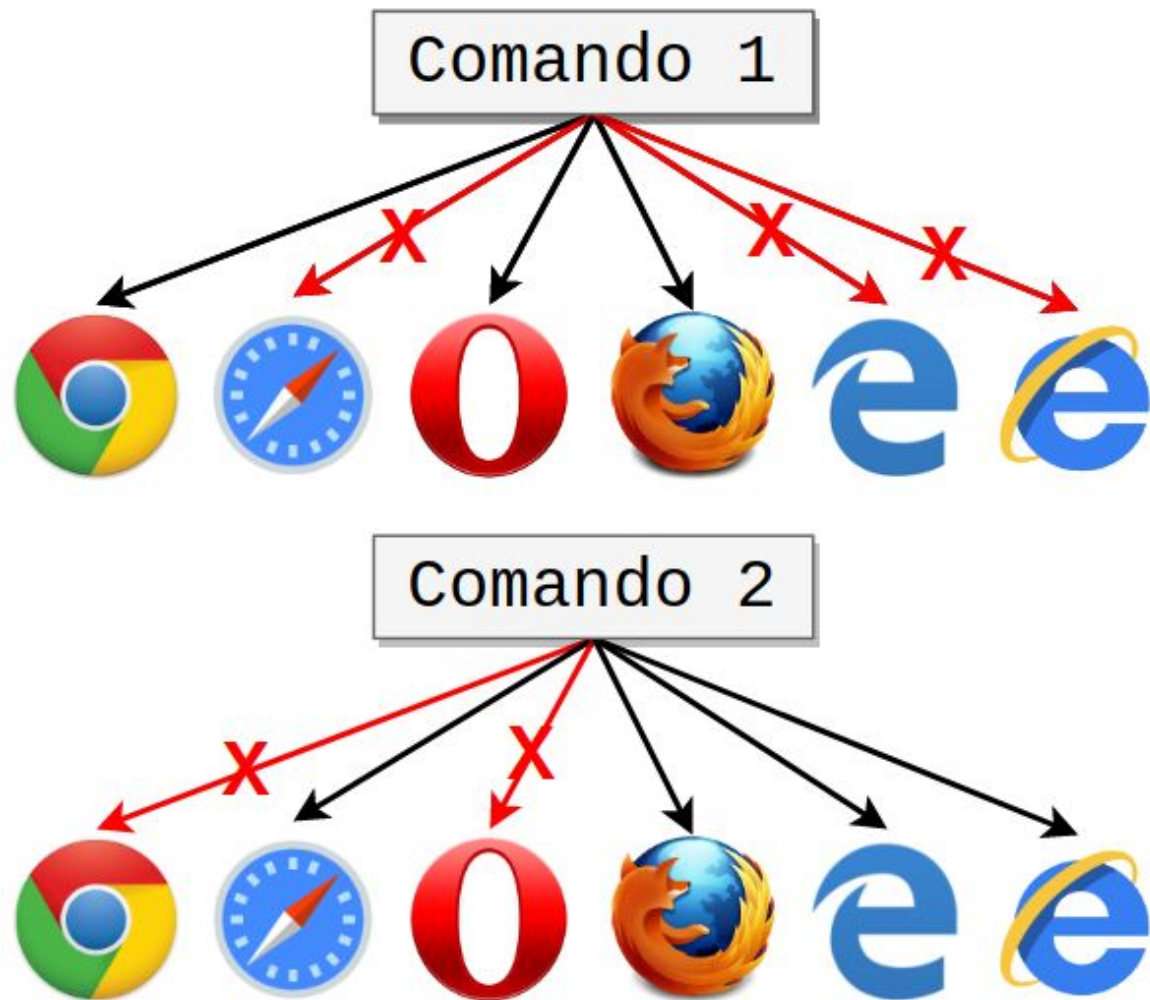
```
/* Vanilla JavaScript */
```

```
var divs = document.querySelectorAll('section.vanilla > div');  
divs.forEach(function(div){  
    div.textContent = 'JS';  
});
```

```
/* jQuery */
```

```
$('section.jquery > div').text('jQuery');
```

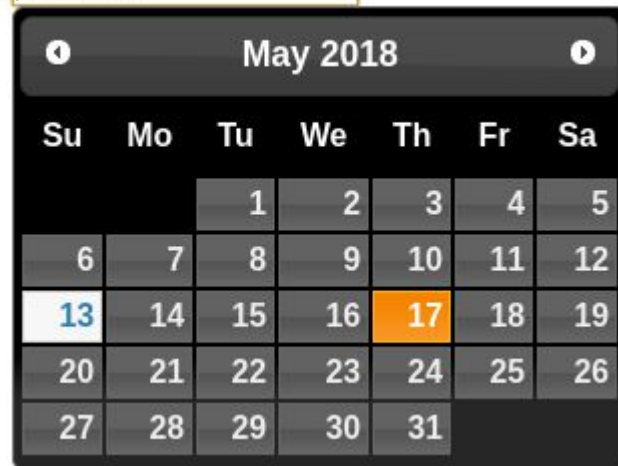
Compatibilidade entre Browsers



Elementos de UI sofisticados (jQuery UI) (exe3.html)

```
<input type="text" id="datepicker">  
<script type="text/javascript">  
    $( "#datepicker" ).datepicker();  
</script>
```

Data: 05/17/2018



Componentes: <https://jqueryui.com/>

Otimizado para UI mobile (jQuery Mobile) (exe4.html)

O próprio jQuery e o jQuery UI possuem opções para uso em dispositivos mobile. Além do que eles oferecem, há também a biblioteca ***jQuery Mobile*** que oferece recursos próprios para celulares e tablets.

Atenção: caso opte em usá-lo, atente-se que ele não é compatível com as versões mais novas do jQuery.

Mais detalhes: <http://jquerymobile.com/>

Atenção: seu uso para sites mobile NÃO é obrigatório. Ele possui facilidades que poderiam ser implementadas com jQuery puro.

Testes em JavaScript (QUnit) (exe5.html)

// Código que precisa ser testado

```
function somarNumeros(n1,n2) {  
    return n1 + n2;  
}
```

// Uso do QUnit

```
QUnit.test( "Teste de funções", function( assert ) {  
    var atual = somarNumeros(2,2);  
    var esperado = 4;  
    assert.equal( atual, esperado, "Função somarNumeros() está ok!" );  
});
```

Qual versão jQuery vamos utilizar?

Download: <http://jquery.com/download/>

Veja: <https://jquery.com/browser-support/>

Current Active Support

Desktop

- Chrome: (Current - 1) and Current
- Edge: (Current - 1) and Current
- Firefox: (Current - 1) and Current
- Internet Explorer: 9+
- Safari: (Current - 1) and Current
- Opera: Current



E os IE 8 e inferiores?

Para navegadores antigos, como Internet Explorer 6-8, Opera 12.1x ou Safari 5.1+, usar [jQuery 1.12](#)



Caso você não utilize animações e AJAX, opte pela versão **jQuery Slim**, que é menor por não possuir suporte a esses recursos.

Sintaxe e Exemplos

Obtendo o texto dos elementos (exe6.html)

Podemos obter o texto através do método *text()*. Este mesmo método pode ser utilizado para definir um valor se passarmos um conteúdo entre os parênteses.

```
/* Vanilla JavaScript */
```

```
var div1 = document.querySelector('.div1');  
console.log(div1.textContent);
```

```
/* jQuery */
```

```
console.log($('.div2').text());  
$('.div3').text('Texto inserido na div!');
```

jQuery com JavaScript puro, pode? (exe7.html)

Claro que pode! O jQuery é uma biblioteca auxiliar ao código JavaScript. Veja o exemplo a seguir, onde obtemos o texto de uma <div>, fazemos um **split** (separa as palavras) para exibir as palavras no console.

```
/* jQuery e JavaScript */  
var texto = $('div').text();  
var palavras = texto.split(' ');  
console.log('Total de palavras: ' + palavras.length);  
palavras.forEach(function(palavra){  
    console.log(palavra);  
});
```

Divide o texto em palavras com o separador especificado entre parênteses (neste caso, um espaço em branco).

Obtém o número de elementos do array.

Obter e definir o valor dos elementos (exe8.html)

Para obter e definir o valor dos elementos, devemos utilizar o método *val()*, conforme a seguir.

```
/* JavaScript */
```

```
var tagInput1 = document.querySelector('#input1');  
tagInput1.value = "Novo valor";
```

```
/* jQuery */
```

```
$('#input2').val("Novo valor");
```

Tratando eventos (exe8.html)

O jQuery também oferece uma sintaxe simplificada para captura dos eventos. Podemos “escutar” eventos com o método *on()*

```
/* JavaScript */
```

```
var tagSelect1 = document.querySelector('#select1');  
tagSelect1.addEventListener('change',function(evento){  
    console.log(evento.target.value);  
});
```

```
/* jQuery */
```

```
$('#select2').on('change',function(evento){  
    console.log(evento.target.value);  
});
```

Quais eventos existem no jQuery? (exe9.html)

Os mesmos que existem no JavaScript padrão. Além da sintaxe pelo método *on()*, o jQuery fornece métodos específicos para os principais eventos utilizados no desenvolvimento. Exemplos:

```
$('#a').click(function(){ console.log('Link clicado!'); });  
$('#input[type=text]').focus(function(){ console.log('Eu fui  
selecionado!'); });  
$('#input[type=text]').blur(function(){ console.log('Que pena...!'); });  
$('#form').submit(function(evento){  
    evento.preventDefault();  
    console.log('Formulário enviado!');  
});
```

Mais detalhes: <https://api.jquery.com/category/events/>

Método `one()` (exe10.html)

Outro recurso legal é o uso do método `one()`. Ele também dispara uma função para um determinado evento, porém **SOMENTE** permite **UMA** execução.

```
$('form').one('submit',function(evento){  
    evento.preventDefault();  
    console.log('Formulário enviado!');  
});
```



É uma boa prática utilizar o método `one()` para eventos que só devem ser disparados uma única vez.

Exercício (exe11/index.html)

Vamos agora reescrever o TO-DO App com jQuery. Iniciaremos com a obtenção do nome do usuário do `<input>` e sua exibição dentro da **Notificação de Sucesso** com o texto “Seja bem-vindo XYZ”, onde XYZ é o nome digitado.

Manipulando atributos (exe11.html)

Através do método `attr()` é possível manipular atributos dos elementos. O primeiro parâmetro recebe uma string com o nome do atributo, enquanto o segundo recebe seu valor.

```
$('#button.esquerda').on('click',function(){  
    $('#input').attr('disabled',false);  
    $('#input').attr('value','Um valor...');  
});
```



Veja nesse caso que há duas consultas seguidas ao DOM. Apesar de funcionar perfeitamente, isso prejudica a performance. **Mas como fazer?**

Manipulando atributos (exe12.html)

```
$('#input').attr('disabled',false);  
$('#input').attr('value','Um valor...');
```



```
var input = $('#input');  
input.attr('disabled',false);  
input.attr('value','Um valor...');
```

Uma alternativa é guardar o elemento em uma variável.



```
$('#input')  
  .attr('disabled',false)  
  .attr('value','Um valor...');
```

Outra é sequenciar os métodos através da interface fluente do jQuery!

Exercício (exe11/index.html)

Altere o projeto TO-DO App para, após o clique do botão de logar, desabilitar o input do nome e o próprio botão.

Dica: não realize buscas seguidas de um mesmo elemento.

Resolução do exercício

```
var botao = $('#botao-logar');  
botao.one('click',function(evento){  
    var nome = $('#input-nome').attr('disabled',true).val();  
    var texto = 'Seja bem-vindo ' + nome + '!';  
    botao.attr('disabled',true);  
    $('#sucesso').text(texto);  
});
```



Podemos obter o elemento por *event.target*?

Podemos usar a palavra reservada *this*?

Resolução do exercício

```
$('#botao-logar').one('click',function(evento){  
    console.log(evento.target);  
});
```



Obter o *target* do evento é possível, mas ele **NÃO** retorna um objeto do jQuery. Ou seja, não é possível acessar o método *attr()* de *evento.target*. Ele retorna um elemento JavaScript padrão.

```
$('#botao-logar').one('click',function(){  
    $(this).attr('disabled',true);  
});
```



Para obter o objeto jQuery cujo evento foi capturado, deve-se utilizar a sintaxe exemplificada acima: **\$(this)**

Efeitos de *show()* e *hide()* (exe13.html)

Através dos métodos *show()* e *hide()* é possível, respectivamente, fazer com que o elemento seja mostrado ou escondido. Por padrão, eles provocam um efeito abrupto.

```
$('#a.left').on('click',function(){  
    $('#h1').show(2000);  
});  
$('#a.right').on('click',function(){  
    $('#h1').hide(2000);  
});
```



É possível utilizar apenas um botão?

Método *is()* (exe14.html)

Através do método *is()* nós podemos verificar se o elemento é uma determinada tag, se possui classes ou um seletor próprio do jQuery chamado *:visible*

```
$('#a').on('click',function(){  
    var h1 = $('h1');  
    if ( h1.is(":visible") ) {  
        h1.hide();  
        $(this).text('Mostrar');  
    } else {  
        h1.show();  
        $(this).text('Esconder');  
    }  
});
```



**Podemos melhorar
esse código?**

Efeito *toggle()* (exe15.html)

Outra melhoria seria utilizar o *toggle()*. Este método aplicará o *hide()* se o elemento estiver visível e o *show()* caso contrário.

```
$('#a').on('click',function(){  
    $('#h1').toggle(0,alterarLabel);  
});
```

Como primeiro parâmetro, ele recebe o tempo (em milissegundos) de duração do efeito.

Como segundo parâmetro, ele recebe uma função que será executada quando o *toggle()* terminar seu efeito.

Efeitos de *fadeIn()* e *fadeOut()* (exe16.html)

Através dos métodos *fadeIn()* e *fadeOut()* é possível, respectivamente, fazer com que o elemento seja mostrado ou escondido. Neste caso o efeito de desvanecer e exibir são suaves, de acordo com os milissegundos passados por parâmetro. O valor default é de 400 milissegundos.

```
$('#a.left').on('click',function(){  
    $('#h1').fadeIn(2000);  
});  
$('#a.right').on('click',function(){  
    $('#h1').fadeOut(2000);  
});
```



**Podemos melhorar
esse código?**

Efeito *fadeToggle()* (exe17.html)

Assim como o *toggle()*, também existe o *fadeToggle()*. Este método aplicará o *fadeOut()* se o elemento estiver visível e o *fadeIn()* caso contrário.

```
$('#a').on('click',function(){  
    $('#h1').fadeToggle(2000, function(){  
        console.log('Efeito concluído!');  
    });  
});
```



E se clicarmos várias vezes no link?

Ele também aceita uma função que será executada **após** a conclusão do efeito.

Interromper efeitos com *stop()* (exe18.html)

Através do método *stop()* é possível interromper os efeitos disparados anteriormente. É uma boa prática utilizá-lo antes de efeitos que podem ser invocados várias vezes.

```
$('#a').on('click',function(){  
    $('#h1').stop().fadeToggle(2000, alterarLabel);  
});
```



É uma boa prática utilizar o método *stop()* antes de invocar efeitos não instantâneos que podem ser disparados várias vezes.

Exercício (exe11/index.html)

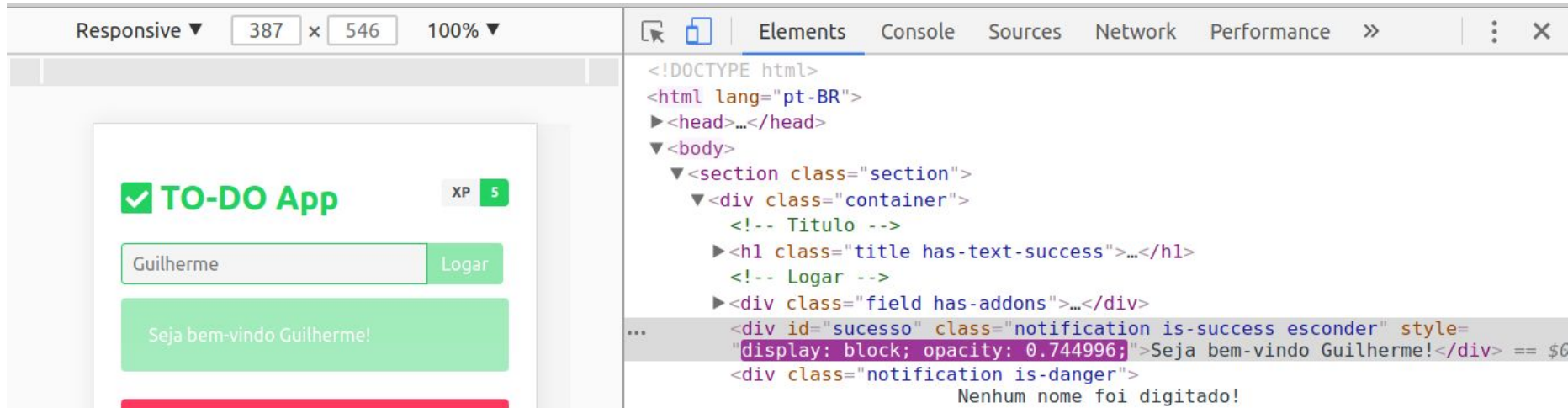
Altere o projeto TO-DO App para:

- Esconder as notificações de sucesso e falha (via CSS)
- Quando o botão de logar for clicado, dispare sequencialmente os métodos de *show()* e *fadeOut()* com 3 segundos de delay para as notificações:
 - Falha: se nenhum valor for digitado
 - Sucesso: se um valor for digitado, use o texto “Seja bem-vindo XYZ!”

Resolução do exercício (veja na aba *Elements*)

```
$('#sucesso').text(texto).show().fadeOut(3000);
```

Mas afinal, qual é a mágica do *fadeOut()*?



The screenshot displays a web browser window with a responsive design view (387x546, 100% zoom). The browser shows a "TO-DO App" interface. On the left, there is a green checkmark icon and the text "TO-DO App". To the right of this is a small box labeled "XP 5". Below this, there is a text input field containing "Guilherme" and a "Logar" button. Underneath the input field is a green notification box with the text "Seja bem-vindo Guilherme!". At the bottom of the interface, there is a red bar.

The right side of the screenshot shows the browser's developer tools, specifically the "Elements" tab. The DOM tree is expanded to show the following structure:

```
<!DOCTYPE html>
<html lang="pt-BR">
  <head>...</head>
  <body>
    <section class="section">
      <div class="container">
        <!-- Titulo -->
        <h1 class="title has-text-success">...</h1>
        <!-- Logar -->
        <div class="field has-addons">...</div>
        ...
        <div id="sucesso" class="notification is-success esconder" style=
          "display: block; opacity: 0.744996;">Seja bem-vindo Guilherme!</div> == $6
        <div class="notification is-danger">
          Nenhum nome foi digitado!
```



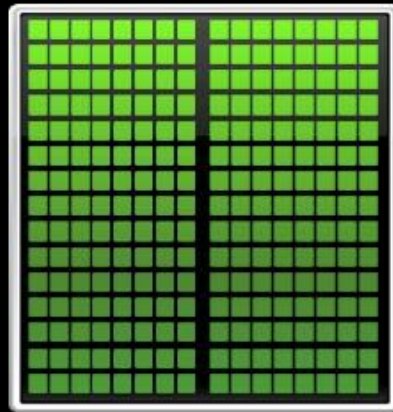
Tudo não passa de um `setInterval()`

Na prática o jQuery faz um `setInterval()` ajustando a opacidade iniciando de 1 até 0 em intervalos regulares de tempo.

Beleza, mas e daí?



CPU



GPU

Então, resumindo:

Use os efeitos de jQuery com parcimônia, preferencialmente **apenas** nos casos em que os atributos CSS ainda não possuem compatibilidade com os browsers mais atuais.

Como a sintaxe de *animation* e *@keyframes* ainda possuem pouca compatibilidade, escolha bem o uso de ***fadeIn()* / *fadeOut()*** versus o **CSS puro**.

*Eu nunca mais vou usar jQuery para aplicar efeitos.
Eu nunca mais vou usar jQuery para aplicar efeitos.
Eu nunca mais vou usar jQuery para aplicar efeitos.*



Sintaxe simplificada dos eventos (exe19.html)

O jQuery fornece um método direto no elemento para os principais eventos disparados. Veja alguns exemplos:

```
$('#nome').focus(function(){});
```

```
$('#livros').change(function(){});
```

```
$('#termos').click(function(){});
```

```
$('#form').submit(function(evento){});
```

Mais detalhes: <https://api.jquery.com/category/events/>

Alterando estilos CSS dos elementos (exe20.html)

Para alterarmos um estilo diretamente nos elementos, devemos utilizar o método `css()`, onde o primeiro parâmetro é a propriedade e o segundo é o valor.

```
/* JavaScript */  
var botaoEdit = document.querySelector('#edit');  
botaoEdit.addEventListener('click',function(){  
    var tagH1 = document.querySelector('h1');  
    tagH1.style.backgroundColor = 'darkgrey';  
});  
/* jQuery */  
$('#edit').click(function(){  
    $('h1').css('background-color','darkgrey');  
});
```



Lembre-se que estilizar diretamente pelo JavaScript pode prejudicar a manutenibilidade do sistema.

Use com parcimônia!

Adicionando/removendo classes CSS (exe21.html)

Como uma alternativa da estilização via JavaScript direta, há também os métodos *addClass()*, *removeClass()* e *toggleClass()*. Note que é possível informar várias classes por vez, separando-as por um espaço.

```
var tag = $('h1');  
tag.removeClass('classe-legal');  
tag.removeClass('darkgrey tomato green forestgreen gold blue');  
tag.addClass('gold');  
tag.toggleClass('efeito aumentar');
```



Estilos exclusivos por classes é uma boa prática, **porém...**

Veja um contraexemplo no *exe21.html* que tornou a manutenção do código PIOR do que estilizar diretamente pelo JavaScript!

Exercício (exe22/index.html)

Altere o projeto TO-DO App:

- Esconda as divs do formulário e de exibição das tarefas.
- Apague as tarefas estáticas (se quiser, guarde o HTML para facilitar o código de criação futuro).
- Após o clique no botão logar, mostre o formulário de cadastro de novas tarefas.

Criando elementos (exe23.html)

Para criar elementos, utilize a sintaxe `$()` passando como parâmetro a tag (entre sinais de menor e maior) a ser criada. Veja que logo na criação já é possível invocar métodos sequencialmente.

```
var div = $('<div>').addClass('estilo-legal').text('Texto legal');
```

Para adicionar as tags criadas nos elementos do DOM, podemos utilizar os métodos *append()* ou *prepend()* para, respectivamente, acrescentar um elemento no final ou no início.

```
$('#main').append(div);
```

```
$('#main').prepend(div);
```

Criando elementos (exe24.html)

A criação de elementos diretamente a partir de strings (comuns ou template string) também é possível pelo jQuery.

```
var novaDiv = `  
  <div class="${classe}">  
    ${quantasDivs + 1}  
  </div>  
`;  
;
```



Note que utilizando template strings é possível informar variáveis diretamente dentro das crases através da notação `${variavel}`. Lembre-se que este recurso é do EcmaScript 6.



É mais recomendado criar elementos pela sintaxe do `$()`, porém, caso seja necessário (ou mais fácil) utilizar a criação por strings, opte pela utilização de template strings.

Outros recursos para manuseio das tags (exe25.html)

```
var divs = $('main').find('div');
```

Através do *find()* é possível obter tags, atributos ou elementos do jQuery que existam internamente na tag cujo método foi invocado. No exemplo acima, caso `<main>` possua muitas `<div>`'s, o método retornará um array.

```
 $('<div>').click(function(){  
    $(this).remove();  
});
```

Veja como adicionar um listeners para os eventos. O método *remove()* pode ser invocado para excluir o elemento.

Podemos utilizar os métodos *parent()* e *children()* para respectivamente acessar o elemento pai e os elementos filhos.

```
 $('main').parent();  
 $('main').children();
```

Exercício (exe22/index.html)

Altere o projeto TO-DO App para criar novas tarefas a partir do formulário de criação. Cada nova tarefa já deve ter seus respectivos botões de “Apagar” e “Feita!” funcionando. Por enquanto eles devem apenas removê-las.

Dica: para remover as tarefas, você pode navegar no DOM utilizando *parent()* ou definir um listener na <div> mais alta da tarefa e trabalhar com a fase de bubbling.

Dica: não se esqueça do *preventDefault()* do formulário.

Observação: por enquanto utilize a sintaxe a seguir para criar a data:

```
var dataCriacao = (new Date()).toLocaleString();
```

AJAX com jQuery : GET (exe26.html e exe27.html)

```
$.ajax({  
  method: 'GET',  
  url: 'https://viacep.com.br/ws/27933-140/json/'  
}).done(function(dados){  
  console.log(dados);  
});
```

ou

```
var url = 'https://viacep.com.br/ws/27933-140/json/';  
$.get(url, function(dados){  
  console.log(dados);  
});
```

Exemplo de uso da API: <http://lgapontes.com/aulas/cadastro/>

AJAX com jQuery : POST (exe28.html e exe29.html)

```
$.ajax({  
    method: 'POST',  
    url: url,  
    data: JSON.stringify(usuario)  
}).done(function(json){  
    console.log(json);  
});
```

```
var usuarioString = JSON.stringify(usuario);  
$.post(url,usuarioString,function(json){  
    console.log(json);  
});
```

OU

Exemplo de uso da API: <http://lgapontes.com/aulas/cadastro/>

AJAX com jQuery : PUT (exe29.html)

```
$.ajax({  
    method: 'PUT',  
    url: url,  
    data: JSON.stringify(usuario)  
}).done(function(json){  
    console.log(json);  
});
```

Exemplo de uso da API: <http://lgapontes.com/aulas/cadastro/>

AJAX com jQuery : DELETE (exe29.html)

```
$.ajax({  
    method: 'DELETE',  
    url: url,  
    data: JSON.stringify(usuario)  
}).done(function(json){  
    console.log(json);  
});
```

Exemplo de uso da API: <http://lgapontes.com/aulas/cadastro/>

Exercício (exe22/index.html)

Quando o usuário clicar no botão de logar:

- Aplique a classe *is-loading* à <div> que envolve o <input> de nome para indicar que a página está consultando dados na API.
- Obtenha os tipos de tarefas a partir da API
<http://lgapontes.com/aulas/todoapp/api/tipos>
- Crie <input>'s do tipo radio para representá-los no formulário.

Tratando erros do AJAX (exe30.html e exe31.html)

```
$.ajax({  
  method: 'GET',  
  url: url,  
  data: dados  
}).done(function(json){  
  console.log(json);  
}).fail(function(json){  
  console.log(json);  
}).always(function(){  
  console.log(json);  
});
```

OU

```
$.get(url,dados,function(json){  
  console.log(json);  
}).fail(function(json){  
  console.log(json);  
}).always(function(){  
  console.log(json);  
});
```

Para definir todo o conteúdo html de um elemento (igual à sintaxe `innerHTML`), utilize o método *html()*

Exemplo de uso da API: <http://lgapontes.com/aulas/cadastro/consulta.html>

Exercício (exe32/index.html)

Agora podemos reescrever toda TO-DO App com recursos jQuery. Ajuste o código para:

- Realizar o login na API, ajustando o XP
- Incluir etapas, ajustando o XP
- Apagar ou realizar etapas, ajustando o XP

Dica: consulte o material da Aula 3 para resgatar o endereço dos métodos da API do TO-DO App publicada na Internet.

Dica: Lembre-se de criar campos `<input>` para guardar o código do usuário e das tarefas.

Execução após carregamento da página (exe33.html)

Veja a seguir mecanismos para executar ações após o carregamento do DOM e da página.

```
/* JavaScript */
```

```
document.addEventListener('DOMContentLoaded',  
    function(){ console.log('Ok'); });
```

```
window.addEventListener('load',  
    function(){ console.log('Ok'); });
```

```
/* jQuery */
```

```
$(document).ready(function(){ console.log('Ok'); });
```

```
$(function(){ console.log('Ok'); });
```

```
$(window).on('load',function(){ console.log('Ok'); });
```

**DOM
carregado
(sem
imagens, etc)**

**Página
carregada
(imagens,
scripts, etc)**

Obrigado!