CAPÍTULO 5

JavaScript e interatividade na Web

"Design não é só como uma coisa aparenta, é como ela funciona."
— Steve Jobs

JavaScript é a linguagem de programação mais popular no desenvolvimento Web. Suportada por todos os navegadores, a linguagem é responsável por praticamente qualquer tipo de dinamismo que queiramos em nossas páginas.

Se usarmos todo o poder que ela tem para oferecer, podemos chegar a resultados impressionantes. Excelentes exemplos disso são aplicações Web complexas como Gmail, Google Maps e Google Docs.

5.1 - INTRODUÇÃO AO JAVASCRIPT

Para rodar JavaScript em uma página Web, precisamos ter em mente que a execução do código é instantânea. Para inserirmos um código JavaScript em uma página, é necessário utilizar a tag `<script>`:

```
<script>
  alert("Olá, Mundo!");
</script>
```

O exemplo acima é um "hello world" em JavaScript utilizando uma função do navegador, a função **alert**. É possível adicionar essa tag em qualquer local do documento que a sua renderização ficará suspensa até o término dessa execução.

Experimente criando essa tag em um HTML e reposicionando-a, verifique os elementos que já foram renderizados na página antes do aparecimento da caixa de diálogo e o que acontece após clicar em "OK".

Também é possível executar código que está em um arquivo externo, por exemplo:

No arquivo HTML

```
<script src="scripts/hello.js"></script>
```

Arquivo externo script/hello.js

```
alert("Olá, Mundo!");
```

Com a separação do script em arquivo externo é possível reaproveitar alguma funcionalidade em mais de uma página.

5.2 – Console do navegador

Alguns navegadores dão suporte a entrada de comandos pelo console. Por exemplo, no Google Chrome o console pode ser acessado por meio do atalho **Control + Shift + C**; no Firefox, pelo atalho **Control + Shift + K**.

Experimente executar um `alert` no console e veja o resultado:

```
alert("interagindo com o console!");
```


5.3 – Sintaxe básica

Operações matemáticas

Teste algumas contas digitando diretamente no console:

```
> 12 + 13
  25
```

```
> 14 * 3
  42
> 10 - 4
  6
> 25 / 5
  5
> 23 % 2
  1
```

Variáveis

Para armazenarmos um valor para uso posterior, podemos criar uma **variável**:

```
var curso = "WD-43";

alert(curso);
```

No exemplo acima, guardamos o valor `WD-43` na variável `curso`. A partir desse ponto, é possível utilizar a variável para obter o valor que guardamos nela. No JavaScript, também é possível alterar o valor da variável a qualquer momento, inclusive por valores de tipos diferentes (por exemplo, um número) sem problemas, coisa que não é possível de se fazer em algumas outras linguagens de programação.

Teste no console:

```
> var contador = 0;
  undefined
> contador++
  0
> contador
  1
> contador++
  1
> contador
  2
```

Tipos

O JavaScript dá suporte aos tipos **String** (letras e palavras), **Number** (números inteiros, decimais), **Boolean** (verdadeiro ou falso) entre outros.

```
var texto = "Uma String deve ser envolvida em aspas simples ou duplas.";
var numero = 2012;
var verdade = true;
```

Outro tipo de informação que é considerado um tipo no JavaScript é o **Array**, onde podemos armazenar uma série de informações de tipos diferentes:

```
var pessoas = ["João", "José", "Maria", "Sebastião", "Antônio"];
```

Quando utilizamos o JavaScript para interagir com os elementos da página é muito comum obtermos coleções. Para fazer alguma coisa com cada elemento de uma coleção é necessário efetuar uma iteração. A mais comum é utilizando o **for**:

```
var pessoas = ["João", "José", "Maria", "Sebastião", "Antônio"];

for (var i = 0; i < pessoas.length; i++) {
  alert(pessoas[i]);
}
```

Essa sintaxe utilizando os colchetes em `pessoas[i]` é uma maneira de selecionarmos um elemento de um Array, no caso o valor de `i` é um número para cada um dos elementos da coleção. Para explorar o comportamento do Array você pode realizar alguns testes com essa seleção:

```
var pessoas = ["João", "José", "Maria", "Sebastião", "Antônio"];

alert(pessoas[0]);
alert(pessoas[1]);
alert(pessoas[4]);
```

5.4 - Exercícios opcionais: fixação de sintaxe

Os próximos exercícios são opcionais e mostram mais aspectos de lógica de programação com algoritmos usando JavaScript.

1. Escreva um código que mostre os números ímpares entre 1 e 10.

2. Escreva um código que calcule a soma de 1 até 100. (obs: a resposta é 5050)

3. Crie um Array igual ao abaixo e mostre apenas os nomes das pessoas que tenham 4 letras.

```
var pessoas = ["João", "José", "Maria", "Sebastião", "Antônio"];
```

Dica: use o atributo `length` das Strings.

5.5 - Interatividade na Web

O uso do JavaScript como a principal linguagem de programação da Web só é possível por conta da integração que o navegador oferece para o programador, a maneira com que conseguimos encontrar um elemento da página e alterar alguma característica dele pelo código JavaScript:

```
var titulo = document.querySelector("#titulo");

titulo.textContent = "Agora o texto do elemento mudou!";
```

No exemplo anterior, nós selecionamos um elemento do documento e alteramos sua propriedade **textContent**. Existem diversas maneiras de selecionarmos elementos de um documento e de alterarmos suas propriedades. Inclusive é possível selecionar elementos de maneira similar ao CSS, através de seletores CSS:

```
var painelNovidades = document.querySelector("section#main
.painel#novidades");

painelNovidades.style.color = "red"
```

> **querySelector vs querySelectorAll**
>
> A função `querySelector` sempre retorna um elemento, mesmo que o seletor potencialmente traga mais de um elemento, neste caso, apenas o primeiro elemento da seleção será retornado.
>
> A função `querySelectorAll` retorna uma lista de elementos compatíveis com o seletor CSS passado como argumento. Sendo assim, para acessarmos cada elemento retornado, precisaremos passar o seu índice conforme o exemplo abaixo:
>
> ```
> var paragrafos = document.querySelectorAll("div p");
> paragrafos[0].textContent = "Primeiro parágrafo da seleção";
> paragrafos[1].textContent = "Segundo parágrafo da seleção";
> ```

Apesar de ser interessante a possibilidade de alterar o documento todo por meio do JavaScript, existe um problema com a característica de execução imediata do código. O mais comum é que as alterações sejam feitas quando o usuário executa alguma ação, como por exemplo, clicar em um elemento.

Para suprir essa necessidade, é necessário utilizar de dois recursos do JavaScript no navegador. O primeiro é a a criação de **funções**:

```
function mostraAlerta() {
  alert("Funciona!");
}
```

Ao criarmos uma função, a execução do código simplesmente guarda o que estiver dentro da função, e esse código guardado só será executado quando

chamarmos a função. Para exemplificar a necessidade citada acima, vamos utilizar o segundo recurso, os **eventos**:

```
// obtendo um elemento através de um seletor de ID
var titulo = document.querySelector("#titulo");

titulo.onclick = mostraAlerta;
```

Note que primeiramente é necessário selecionar um elemento do documento e depois definir o `onclick` desse elemento como sendo a função.

De acordo com os dois códigos acima, primeiramente guardamos um comportamento em uma função. Ao contrário do código fora de uma função, o comportamento não é executado imediatamente, e sim guardado para quando quisermos chamá-lo. Depois informamos que um elemento deve executar aquele comportamento em determinado momento, no caso em um evento "click".

Também é possível chamar uma função em um momento sem a necessidade de um evento, é só utilizar o nome da função abrindo e fechando parênteses, indicando que estamos executando a função que foi definida anteriormente:

```
function mostraAlerta() {
  alert("Funciona!");
}

// Chamando a função:
mostraAlerta();
```

Também é possível determinar, por meio de seus argumentos, que a função vai ter algum valor variável que vamos definir quando quisermos executá-la:

```
function mostraAlerta(texto) {
  // Dentro da função "texto" conterá o valor passado na execução.
  alert(texto);
}

// Ao chamar a função é necessário definir o valor do "texto"
mostraAlerta("Funciona com argumento!");
```

Existem diversos eventos que podem ser utilizados para que a interação do usuário com a página seja o ponto de disparo de funções que alteram os elementos da própria página:

- onclick: clica com o mouse
- ondblclick: clica duas vezes com o mouse
- onmousemove: mexe o mouse

- onmousedown: aperta o botão do mouse
- onmouseup: solta o botão do mouse (útil com os dois acima para gerenciar drag'n'drop)
- onkeypress: ao pressionar e soltar uma tecla
- onkeydown: ao pressionar uma tecla.
- onkeyup: ao soltar uma tecla. Mesmo acima.
- onblur: quando um elemento perde foco
- onfocus: quando um elemento ganha foco
- onchange: quando um input, select ou textarea tem seu valor alterado
- onload: quando a página é carregada
- onunload: quando a página é fechada
- onsubmit: disparado antes de submeter o formulário. Útil para realizar validações

Existem também uma série de outros eventos mais avançados que permitem a criação de interações para drag-and-drop, e até mesmo a criação de eventos customizados.

No próximo exercício, vamos usar funções JavaScript com eventos para lidar com a validação do campo de busca da nossa home page. A ideia é verificar se o campo foi preenchido ou se está vazio, quando o usuário clicar em buscar.

5.6 - Exercício: validação na busca com JS

1. Para testarmos o nosso formulário de busca, vamos usar o Google como mecanismo de busca apenas como ilustração. Para isso, basta fazer o formulário submeter a busca para a página do Google.

No `<form>`, acrescente um atributo `action=` apontando para a página do Google:

```
<form action="http://www.google.com.br/search" id="form-busca">
```

Repare que colocamos também um `id`. Ele será útil para depois manipularmos esse elemento via JavaScript.

Além disso, o `<input>` com o texto a ser buscado deve ter o nome de **q** para ser compatível com o Google. Basta fazer:

```
<input type="search" name="q" id="q">
```

Teste a página e submeta uma busca simples para o Google com o nosso formulário.

2. Queremos fazer uma validação: quando o usuário clicar em submeter, verificar se o valor foi preenchido. Se estiver em branco, queremos mostrar uma mensagem de erro em um `alert`.

A validação será escrita em JavaScript. Portanto, comece criando um arquivo **home.js** na pasta **js/** do projeto. É lá que vamos escrever nossos scripts.

Referencie esse arquivo no **index.html** usando a tag `<script>` no final da página, logo antes de fechar o `</body>`:

```
<script src="js/home.js"></script>
```

3. A validação em si será feita por uma **função JavaScript** a ser acionada no momento que o usuário tentar submeter o formulário.

A função deve ser definida dentro do arquivo **home.js** criado antes. Seu papel é verificar se o elemento com **id=q** (o campo de busca) está vazio. Se estiver, mostramos um `alert` e abortamos a submissão do formulário com `return false`:

```
function validaBusca() {
  if (document.querySelector('#q').value == '') {
      alert('Não podia ter deixado em branco a busca!');
      return false;
  }
}
```

Uma função é um bloco de código JavaScript que executa algum código quando

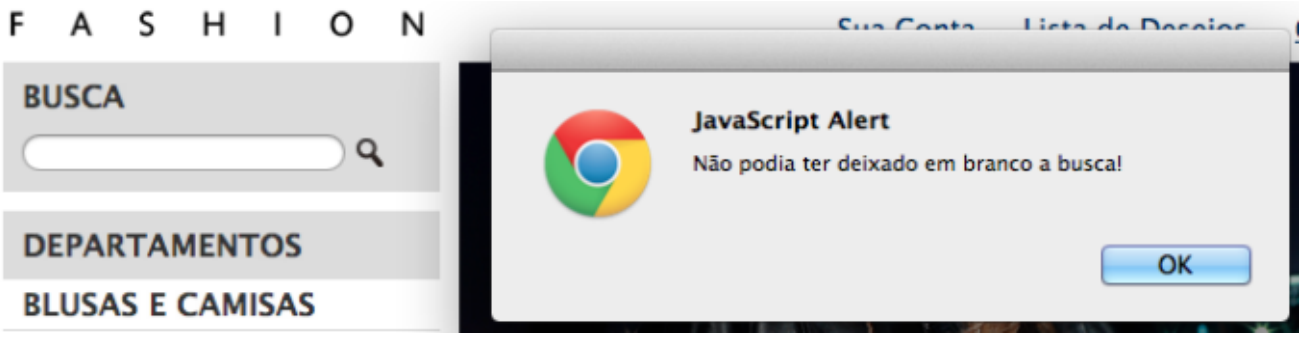
alguém chamar essa função. Nós podemos chamar a função explicitamente ou, melhor ainda, deixar que o navegador chame pra gente quando acontecer algum **evento**.

No nosso caso, queremos indicar que a função anterior deve ser executada quando o usuário disparar o evento de enviar o formulário (`onsubmit`). Coloque o final do arquivo JavaScript:

```
// fazendo a associação da função com o evento
document.querySelector('#form-busca').onsubmit = validaBusca;
```

É o código anterior que faz a associação da função `validaBusca` com o evento `onsubmit` do formulário.

4. Teste a página e observe seu comportamento. Tente submeter com o campo vazio e com o campo preenchido.



5. (opcional) Mostrar uma janela de erro é considerado por muitos uma ação muito agressiva contra o usuário. Talvez um feedback mais sutil seja pintar o fundo do formulário de vermelho, indicando um erro.

Na função de validação, remova a linha do alerta e em seu lugar pinte o fundo do formulário de vermelho em caso de erro. Código da função deverá ficar assim:

```
function validaBusca() {
  if (document.querySelector('#q').value == '') {
      document.querySelector('#form-busca').style.background = 'red';
      return false;
  }
}
```

6. (opcional avançado) No exercício criamos uma função JavaScript com nome `validaBusca` e a referenciamos no `onsubmit` do formulário. Mas repare que o único uso dessa função é nessa situação, nunca mais a chamamos.

Nesses casos, podemos usar um recurso do JavaScript chamado de *funções anônimas* que nos permite definir a função diretamente na definição do `onsubmit`. Troque nosso JavaScript para usar essa sintaxe.

```
document.querySelector('#form-busca').onsubmit = function() {
  if (document.querySelector('#q').value == '') {
      document.querySelector('#form-busca').style.background = 'red';
      return false;
  }
};
```

Validation API HTML5

Mais pra frente, veremos as novidades do HTML5 para validação de formulários de maneira mais simples, e até sem necessidade de JavaScript.

5.7 - Funções temporais

Em JavaScript, podemos criar um *timer* para executar um trecho de código após um certo tempo, ou ainda executar algo de tempos em tempos.

A função `setTimeout` permite que agendemos alguma função para execução no futuro e recebe o nome da função a ser executada e o número de milissegundos a esperar:

```
// executa a minhaFuncao daqui um segundo
setTimeout(minhaFuncao, 1000);
```

Se for um código recorrente, podemos usar o `setInterval` que recebe os mesmos argumentos mas executa a função indefinidamente de tempos em tempos:

```
// executa a minhaFuncao de um em um segundo
setInterval(minhaFuncao, 1000);
```

É uma função útil para, por exemplo, implementar um banner rotativo, como faremos no exercício a seguir.

clearInterval

As funções temporais devolvem um objeto que representa o agendamento que foi feito. É possível usá-lo para cancelar a execução no futuro. É especialmente interessante para o caso do *interval* que pode ser cancelado de sua execução infinita:

```
// agenda uma execução qualquer
var timer = setInterval(minhaFuncao, 1000);

// cancela execução
clearInterval(timer);
```

5.8 – Exercício: banner rotativo

1. Implemente um banner rotativo na home page da Mirror Fashion usando JavaScript.

Temos duas imagens, a **destaque-home.png** e a **destaque-home-2.png** que queremos trocar a cada 4 segundos; use o `setInterval` para isso.

Há várias formas de implementar essa troca de imagens. Uma sugestão é manter um array com os valores possíveis para a imagem e um inteiro que guarda qual é o banner atual.

```
var banners = ["img/destaque-home.png", "img/destaque-home-2.png"];
var bannerAtual = 0;

function trocaBanner() {
  bannerAtual = (bannerAtual + 1) % 2;
  document.querySelector('.destaque img').src = banners[bannerAtual];
}

setInterval(trocaBanner, 4000);
```

2. (opcional, avançado) Faça um botão de *pause* que pare a troca do banner.

Dica: use o `clearInterval` para interromper a execução.

3. (opcional, avançado) Faça um botão de *play* para reativar a troca dos banners.

> Consulte as vantagens do curso *Desenvolvimento Web com HTML, CSS e JavaScript*.

5.9 – Para saber mais: sugestão para o desafio de pause/play

Podemos criar no HTML um novo link para controlar a animação:

```
<a href="#" class="pause"></a>
```

O JavaScript deve chamar `clearInterval` para pausar ou novamente o `setInterval` para continuar a animação.

Precisamos **editar** o código anterior que chamava o `setInterval` para pegar o seu retorno. Será um objeto que controla aquele interval e nos permitirá desligá-lo depois:

```
var timer = setInterval(trocaBanner, 4000);
```

Com isso, nosso código que controla o pause e play ficaria assim:

```
var controle = document.querySelector('.pause');

controle.onclick = function() {
  if (controle.className == 'pause') {
    clearInterval(timer);
    controle.className = 'play';
  } else {
    timer = setInterval(trocaBanner, 4000);
    controle.className = 'pause';
  }

  return false;
};
```

Por fim, podemos estilizar o botão como pause ou play apenas trabalhando com bordas no CSS:

```
.destaque {
  position: relative;
}
.pause,
.play {
  display: block;
  position: absolute;
  right: 15px;
  top: 15px;
}
.pause {
  border-left: 10px solid #900;
```

```
  border-right: 10px solid #900;
  height: 30px;
  width: 5px;
}
.play {
  border-left: 25px solid #900;
  border-bottom: 15px solid transparent;
  border-top: 15px solid transparent;
}
```

5.10 – Para saber mais: vários callbacks no mesmo elemento

Nos exercícios que trabalhamos com eventos, usamos o `onclick` e o `onsubmit` diretamente no elemento que estávamos manipulando:

```
document.querySelector('#destaque').onclick = function() {
  // tratamento do evento
};
```

É uma forma fácil e portável de se tratar eventos, mas não muito comum na prática. O maior problema do código acima é que só podemos atrelar uma única função ao evento. Se tentarmos, em outra parte do código, colocar uma segunda função para executar no mesmo evento, ela sobrescreverá a anterior.

A maneira mais recomendada de se associar uma função a eventos é com o uso de `addEventListener`:

```
document.querySelector('#destaque').addEventListener('click', function() {
  // tratamento do evento
});
```

Dessa maneira, conseguimos adicionar vários listeners ao mesmo evento, deixando o código mais flexível. Só há um porém: embora seja o modo oficial de se trabalhar com eventos, o `addEventListener` não é suportado do IE8 pra baixo.

Para atender os IEs velhos, usamos a função `attachEvent`, semelhante:

```
document.querySelector('#destaque').attachEvent('onclick', function() {
  // tratamento do evento
});
```

O problema é ter que fazer sempre as duas coisas para garantir a portabilidade da nossa página. Essa questão é resolvida pelos famosos frameworks JavaScript, como o jQuery, que veremos mais adiante no curso.

CAPÍTULO ANTERIOR:

Mais HTML e CSS

PRÓXIMO CAPÍTULO:

CSS Avançado

Newsletter

Casa do Código

Twitter