Desenvolvimento Web

Cap. 7 - Javascript e Interatividade

Universidade Estácio de Sá
Análise de Sistemas
Desenvolvimento Web em HTML5, CSS3, Javascript e PHP
Professor: Marlan Külberg
marlan.kulberg@estacio.br

JavaScript

- Na página de produto foi criado um input range para selecionar o tamanho da roupa

- Não há feedback visual de qual valor está selecionado.

- É possível criar um outro elemento visual apenas para mostrar o valor atualmente selecionado
- A tag <output> representa a saída de algum cálculo ou valor simples obtido a partir de um ou mais campos de um formulário.
- Possui um atributo for que aponta de qual elemento saiu o seu valor

```
<output for="tamanho" name=“valortamanho">42</output>
```

- O valor em si está como 42 porque foi colocado a mão, na tag.
- É preciso atualizar o valor toda vez que o valor do input range mudar

- Mudar o conteúdo de uma tag baseado numa ação do usuário (dentro do navegador) não é função do HTML.
- Para isso, é necessário uma ferramenta como o JavaScript

História do JavaScript

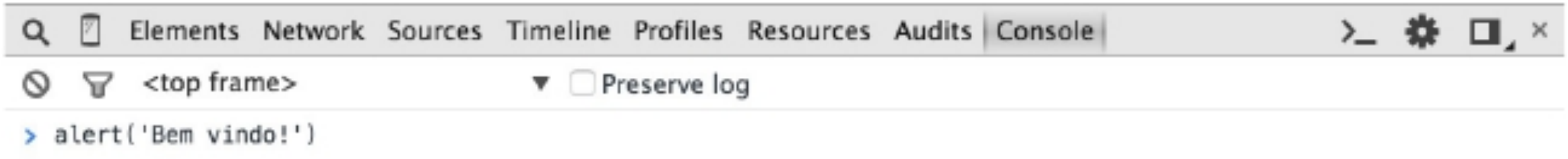
- No início da Internet, páginas eram pouco interativas, apresentavam exatamente o conteúdo a ser exibidos no navegador
- Navegar através de links e enviar informações através de formulários era tudo o que se podia fazer
- Visando o potencial da Internet para o público geral e a necessidade de haver uma interação maior do usuário com as páginas, a Netscape, criou o Livescript, que permitia a execução de scripts contidos nas páginas dentro do próprio navegador.
- Aproveitando sucesso do Java, a Netscape logo rebatizou como JavaScript para alavancar o uso
- A Microsoft, adicionou ao Internet Explorer o suporte a scripts escritos em VBScript (JScript)
- JavaScript é a linguagem de programação mais popular no desenvolvimento Web e suportada por todos os navegadores

Características do JavaScript

- O JavaScript é uma linguagem de scripting
- Permite ao programador controlar aplicações de terceiros
- Linguagem interpretada, não dependem de compilação para serem executadas
- No JavaScript, o código é interpretado e executado conforme é lido pelo navegador, linha a linha, assim como o HTML
- Grande tolerância a erros, pois conversões automáticas são realizadas durante operações
- Para diferenciar script de um código html é necessário envolver o script dentro da tag <script>

Console do Navegador

- Existem várias formas de executar códigos JavaScript em um página
- Uma delas é executar códigos no Console do Navegador
- A maioria dos navegadores já vem com essa ferramenta instalada



Sintaxe Básica

Operadores

- É possível efetuar operações matemáticas como em qualquer linguagem

- Pode-se obter resultados digitando diretamente no console

```
> 12 + 13
25
> 14 * 3
42
> 10 - 4
6
> 25 / 5
5
> 23 % 2
1
```

Sintaxe Básica

Variáveis

- Para armazenarmos um valor para uso posterior, pode-se criar uma variável

```
> var resultado = 102 / 17; //guarda o resultado de 102 / 17 na variável resultado
undefined
```

- O resultado de criar uma variável é sempre undefined

- Para obter o valor armazenado ou alterar o valor:

```
> resultado
6
> resultado = resultado + 10
16
> resultado
16
```

- Também é possível alterar o valor da variável usando operações básicas com sintaxe compacta:

```
> var idade = 10; // undefined
> idade += 10; // idade vale 20
> idade -= 5; // idade vale 15
> idade /= 3; // idade vale 5
> idade *= 10; // idade vale 50
```

Sintaxe Básica

Tipos de dados

- O JavaScript tem vários tipos de dados

Number

- Com esse tipo de dados é possível executar todas as operações vistas anteriormente:

```
var pi = 3.14159;
var raio = 20;
var perimetro = 2 * pi * raio
```

- É possível omitir o ponto e vírgula no final de cada declaração
- A omissão de ponto e vírgula funciona no JavaScript devido ao mecanismo chamado automatic semicolon insertion (ASI)

String

- Uma string em JavaScript é utilizada para armazenar trechos de texto:

```
var empresa = “Estácio";
```

- Para exibir o valor da variável fora do console, executa-se o comando:

```
alert(empresa);
```

- O comando alert serve para criação de popups com conteúdo de texto colocado entre parênteses

```
var numero = 30;
alert(numero)
```

- Qualquer variável pode ser usada no alert . O JavaScript não diferencia o tipo de dados que está armazenado numa variável
- Se necessário, tenta converter o dado para o tipo desejado

A tag Script

- O console permite testar códigos diretamente no navegador
- Não é razoável sempre abrir o console, copiar e colar o código para ser executado
- Para inserir código JavaScript em uma página, utiiza-se a tag <script>

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

- A tag <script> pode ser declarada dentro da tag <head> ou <body>, mas o código é lido imediatamente dentro do navegador

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Aula de JS</title>

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

    </head>
    <body>
        <h1>JavaScript</h1>
        <h2>Linguagem de programação</h2>
    </body>
</html>
```

- Ao ser executado, o script trava o processamento da página.
- É interessante carregar a página primeiro antes da execução por questão de performance e experiência
- Colocar o script no final do <body>

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Aula de JS</title>
    </head>
    <body>
        <h1>JavaScript</h1>
        <h2>Linguagem de Programação</h2>

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

    </body>
</html>
```

JavaScript em Arquivo Externo

- É possível importar scripts dentro da página utilizando a tag <script>:
- No arquivo HTML:

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

- No Arquivo externo js/hello.js:

```
alert("Olá, Mundo!");
```

- Separando o script em arquivo externo é possível reaproveitar funcionalidades em mais de uma página.
- Mensagens ocultas vistas pelo console
- Para visualizar varíaveis ou resultados de operação durante a execução do código.
- Conteúdo exibido apenas para o desenvolvedor, console do navegador é utilizado no lugar do alert para impressão

```
var mensagem = "Olá mundo";
console.log(mensagem);
```

Document Object Model (DOM)

- Quando se carrega o HTML da página, navegadores carregam em memória uma estrutura de dados que representa cada uma das tags no JavaScript.
- Utilizado para permitir alterações na página
- Estrutura conhecida como DOM (Document Object Model)
- Pode ser acessada através da variável global document
- Termo "documento" é utilizado em referências à página
- No contexto de frontend, documento e página são sinônimos

QuerySelector

- Antes de alterar uma página, é preciso acessar no JavaScript o elemento que será alterado
- Para alterar o conteúdo de um título da página, faz-se o seguinte acesso

  document.querySelector(“h1")

- Comando querySelector usa seletores CSS para encontrar os elementos na página.
- Pode-se utilizar outros seletores

  document.querySelector(“.class")

  document.querySelector(“#id")

Elemento da Página como Variável

- Se um mesmo elemento for utilizado várias vezes, é possível armazenar o resultado de uma querySelector em uma variável

```
var titulo = document.querySelector(“h1")
```

- Percebe-se que elemento correspondente é selecionado, no console

- Pode-se ver e manipular o conteúdo

```
titulo.textContent

titulo.textContent = "Novo título”
```

querySelectorAll

- Utilizado se for preciso selecionar vários elementos na página.
- Por exemplo, várias tags de uma mesma classe
- querySelectorAll devolve uma lista de elementos (array)

```
document.querySelectorAll(“.cartao")
```

- Acesso dos elementos da lista é feito através da posição dele (começando em zero) e usando o colchetes:

```
// primeiro cartão
document.querySelectorAll(“.cartao")[0]
```

Funções e Eventos do DOM

- Por padrão, qualquer código colocado no <script>, é executado assim que o navegador o lê
- É interessante que as alterações por JavaScript sejam feitas quando se executa alguma ação, não apenas quando a página carrega.
- Guardar o código para ser executado em outro momento,
- Deve-se criar uma função:

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

- Código só será executado quando a função for chamada

```
// fazendo uma chamada para a função mostraAlerta, que será executada nesse momento
mostraAlerta()
```

- Para chamar a função utiliza-se o nome da função e abre e fecha parênteses
- Para que a função seja chamada quando o usuário clicar no botão:

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

// obtendo um elemento através de um seletor de ID
var botao = document.querySelector(“#botaoEnviar");

botao.onclick = mostraAlerta;
```

- É necessário selecionar o botão e depois definir no onclick que será executada a função mostraAlerta

Eventos

- Diversos eventos podem ser utilizados
    - oninput: quando elemento input tem valor modificado
    - onclick: quando ocorre clique do mouse
    - ondblclick: quando ocorre duplo clique
    - onmousemove: quando movimenta o mouse
    - onmousedown: quando aperta o botão do mouse
    - onmouseup: quando solta o botão do mouse (útil com os eventos acima para gerenciar drag'n'drop)
    - onkeypress: quando pressiona e solta uma tecla
    - onkeydown: quando pressiona uma tecla
    - onkeyup: quando solta uma tecla
    - onblur: quando um elemento sai de foco
    - onfocus: quando um elemento entra em foco
    - onchange: quando input, select ou textarea tem valor alterado
    - onload: quando página é carregada
    - onunload: quando a página é fechada
    - onsubmit: disparado antes de submeter o formulário (útil para validações)
- Existem outros eventos que permitem criação de interações drag-and-drop, e criação de eventos customizados

Atividade - Mostrando o tamanho do Produto utilizando JavaScript

1. Na página produto.html, adicione o elemento output do HTML5 logo após o input range, ainda dentro do fieldset de escolha de tamanho.

```
<output for="tamanho" name=“valortamanho">42</output>
```

Repare que esse elemento não tem visual específico e também não atualiza seu valor sozinho. Vamos implementar isso via JavaScript.

2. O preenchimento inicial e atualização do valor no output deve ser feita via JavaScript. Quando o input range mudar de valor (evento oninput), pegamos seu valor e jogamos no output.

Para escrever o JavaScript, pode-se criar um novo arquivo produto.js e importá-lo na página.

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

O código é:

```
var inputTamanho = document.querySelector(‘[name=tamanho]')
var outputTamanho = document.querySelector(‘[name=valortamanho]')

function mostraTamanho(){
    outputTamanho.value = inputTamanho.value
}

inputTamanho.oninput = mostraTamanho
```

Teste o funcionamento no navegador, veja se o output atualiza de valor corretamente.

Para suportar o IE10, é preciso usar o evento onchange .

O correto no HTML5 seria usar o evento oninput , que até funciona melhor nos browsers modernos.

```
inputTamanho.oninput = mostraTamanho
inputTamanho.onchange = mostraTamanho
```

Se elemento output não for corretamente reconhecido pelo navegador, alterar a propriedade value dele não vai ter o resultado esperado. Para o código
funcionar, é preciso alterar diretamente no texto do elemento:

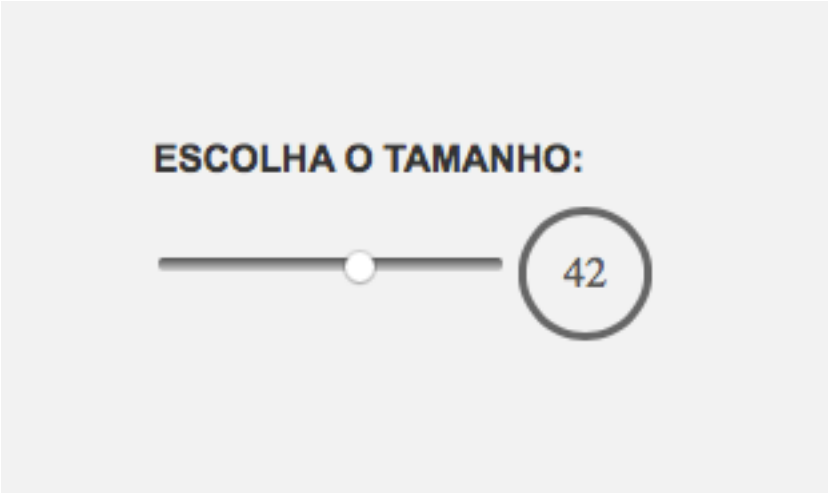
```
function mostraTamanho() {
    outputTamanho.value = inputTamanho.value
    outputTamanho.textContent = event.target.value
}
```

Atividade - Mostrando o tamanho do Produto utilizando JavaScript

3. Estilize o output para ter um design mais ajustado na página de produto.

Adicione no arquivo produto.css

```
.tamanhos output {
    display: inline-block;
    height: 44px;
    width: 44px;
    line-height: 44px;
    text-align: center;
    border: 3px solid #666;
    border-radius: 50%;
    color: #555;
}
```



Funções Anônimas

- No exercício anterior foi indicado que a função mostraTamanho seria executada quando o usuário inserisse o tamanho do produto em <input type="range">

- Apenas indica o nome da função que deve ser executada

```
inputTamanho.oninput = mostraTamanho

function mostraTamanho(){
    outputTamanho.value = inputTamanho.value
}
```

- Algumas funções podem ter uma única referência no código
- Nesses casos, o JavaScript permite a criação da função no lugar onde antes era indicado seu nome

```
inputTamanho.oninput = function() {
    outputTamanho.value = inputTamanho.value
}
```

- Função mostraTamanho transformada uma função anônima
- Continuará sendo executada quando o usuário alterar o valor para o tamanho

Manipulação de Strings

- Variáveis que armazenam strings permitem outras funções
  - Consultar tamanho
  - Realizar transformações em seu valor

```
var empresa = “Estácio”;

empresa.length; // tamanho da string
empresa.replace("cio","tua"); // retorna Estátua
```

- A partir da variável empresa , utiliza-se operador ponto seguido da ação replace

- String é imutável.
- Para obter uma string modificada, é necessário receber o retorno de cada função que manipula a string

```
var empresa = “Estácio";

// substitui a parte "cio" por "tua"
empresa.replace("cio","tua");
console.log(empresa); // imprime Estácio, não muda a variável

empresa = empresa.replace(“cio","tua");
console.log(empresa); // imprime Estátua, pois houve o retorno da função para a variável
```

Conversões e Manipulação de Números

- JavaScript possui funções de conversão de string para number

```
var textoInteiro = “10";
var inteiro = parseInt(textoInteiro);

var textoFloat = “10.22";
var float = parseFloat(textoFloat);
```

- Number também é imutável
- Exemplo altera número de casas decimais com a função toFixed
- Função retorna uma string
- Para funcionar corretamente, seu retorno precisa ser capturado

```
var milNumber = 1000;
var milString = milNumber.toFixed(2);
// recebe o retorno da função
console.log(milString);
// imprime a string “1000.00"
```

Concatenações

- É possível unir tipos diferentes
- JavaScript se encarrega de realizar a conversão entre os tipos
- Pode resultar em algo não esperado
- String com String:

```
var s1 = “Universidade ";
var s2 = “Estácio";
console.log(s1 + s2); // imprime Universidade Estácio
```

String com Outros Tipos de Dados

- JavaScript tentará realizar conversões existirem tipos diferentes em operações

```
var num1 = 2;
var num2 = 3;
var nome = “Estácio"

// Exemplo 1:
console.log(num1 + nome + num2); // imprime 2Estácio3

// Exemplo 2:
console.log(num1 + num2 + nome); // imprime 5Estácio

// Exemplo 3:
console.log(nome + num1 + num2); // imprime Estácio23

// Exemplo 4:
console.log(nome + (num1 + num2)); // imprime Estácio5

// Exemplo 5:
console.log(nome + num1 * num2); // imprime Estácio6
// A multiplicação tem precedência
```

- O resultado de console.log(10-"curso") é NaN (not a number)
- Significa que operações matemáticas com subtração, só podem ser feitas com números.
- O valor NaN possui uma peculiaridade, definida em sua especificação:

```
var resultado = 10-"curso"; // retorna NaN
resultado == NaN; // false
NaN == NaN; // false
```

- Não é possível comparar uma variável com NaN, nem mesmo NaN com NaN
- Para saber se uma variável é NaN , usa-se a função isNaN:

```
var resultado = 10-"curso";
isNaN(resultado); // true
```

Atividade - Calculando o Total da Compra

1. Para que o usuário possa escolher a quantidade do produto que ele quer comprar, criar um campo para a quantidade e outro para exibição do valor total.

Os campos serão inseridos depois do fechamento da div.card.mb-3 dentro da div.col-md-4 na página checkout.html . Antes de inserir, busque a div.card.mb-3 no código. Ela deve estar assim:

```
<div class=“col-md-4">
    <div class="card mb-3">
    <div class="card-header">
        Sua compra
    </div><!-- fim .card-header -->
    <div class="card-body">
        <img src="img/produtos/foto1-verde.png" alt="Fuzzy Cardigan"
        class="img-thumbnail mb-3">
        <dl>
            <dt>Produto</dt>
            <dd>Fuzzy Cardigan</dd>
            <dt>Cor</dt>
            <dd>Verde</dd>
            <dt>Tamanho</dt>
            <dd>40</dd>
            <dt>Preço</dt>
            <dd>R$ 129,90</dd>
        </dl>
    </div><!-- fim .card-body -->
    </div><!-- fim .card mb-3 -->
    <!-- Aqui virá o código novo -->
</div><!-- fim .col-md-4 -->
```

Adicionar o código abaixo dentro da div.col-md-4:

```
<div class="card mb-3">
    <div class="card-body">
        <div class="form-group">
            <label for="qtd">Quantidade:</label>
            <input type="number" id="qtd" min="1" max="99" value="1"
            class="form-control">
        </div>
        <div class="form-group">
            <label for="total">Total:</label>
            <output id="total" class="form-control">R$ 129,90</output>
        </div>
    </div>
</div>
```

Atividade - Calculando o Total da Compra

2. Ainda dentro da div.col-md-4 , adicione um id à <dd> do preço para que o preço unitário seja acessível de forma mais simples quando implementarmos o JavaScript.

```
<dd id="valor">R$ 129,90</dd>
```

Agora que tanto o <dd> do preço e o <input> da quantidade têm ids, é possível dizer que a tag <output> é resultado de alguma operação dos dois:

```
<output for="qtd valor” id="total" class="form-control">
    R$ 129,90
</output>
```

3. O usuário já consegue inserir a quantidade que ele deseja, porém, nada acontece. Para que o valor total da compra seja alterado quando o usuário alterar a quantidade, utiliza-se o JavaScript.

Crie o arquivo total.js dentro da pasta js e importe ele na página.

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

Atividade - Calculando o Total da Compra

4. Agora, dentro de total.js é preciso acessar os elementos da página. Pegaremos o conteúdo da <dd> de preço do produto e multiplicaremos pela quantidade que o usuário digitar no <input> de quantidade.

```
var $input_quantidade = document.querySelector("#qtd");
var $output_total = document.querySelector(“#total");

$input_quantidade.oninput = function() {
    var preco = document.querySelector(“#valor").textContent;
    preco = preco.replace("R$ ", "");
    preco = preco.replace(",", ".");
    preco = parseFloat(preco);

    var quantidade = $input_quantidade.value;
    var total = quantidade * preco;
    total = "R$ " + total.toFixed(2)
    total = total.replace(".", “,");

    $output_total.value = total;
}
```

É possível definir que a função vai ter algum valor variável que será definido ao 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!");
```

Agora, teste a página, o cálculo do total já deve estar funcionando.

Array

- Array é utilizado para trabalhar com diversos valores armazenados

```
var palavras = ["Estácio", "Ensino"];
palavras.push("Inovação"); // adiciona a string "Inovação" na última posição no array
```

- É possível guardar valores de tipos diferentes:

```
var variosTipos = ["Estácio", 10, [1,2]];
```

- Tamanho de um array vai de 0 até o seu tamanho - 1.

```
console.log(variosTipos[1]) // imprime o número 10
```

- Pode-se adicionar quantos elementos se desejar
- Tamanho do array aumentará quando necessário

Adicionar elementos pelo índice

- Em vez de usar função push, que adiciona o elemento como último do array é possível fazer

```
var palavras = ["Estácio", "Ensino"];
palavras[9] = “Inovação";
```

- Altera tamanho do array para dez e adiciona na última posição a string “Inovação"
- Posições intermediárias recebem o valor undefined

Blocos de Repetição

- Executa trecho de código repetidamente até que uma condição seja contemplada, ou enquanto uma condição for verdadeira

for

- Bloco for precisa de algumas informações de controle para evitar que execute infinitamente

```
for (/* variável de controle */; /* condição */; /* pós execução */) {
    // código a ser repetido
}
```

- Somente o campo de condição é obrigatório, mas normalmente utiliza-se todas as informações

```
var palavras = ["Estácio", “Ensino"];

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

while

- Bloco while executa determinado código repetitivamente enquanto uma condição for verdadeira

- Diferente do bloco for, variável de controle e sua manipulação, não são responsabilidades do bloco

```
var contador = 1;

while (contador <= 10) {
    alert(contador + " Mississípi...");
    contador++;
}

alert("Valor do contador: " + contador);
```

Funções Temporais

- É possível criar um timer para executar um trecho de código após um certo tempo, ou executar algo de tempos em tempos
- Função setTimeout permite agendar alguma função para execução no futuro
- 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, pode-se utilizar setInterval
- Recebe os mesmos argumentos mas executa a função indefinidamente

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

clearInterval

- Funções temporais devolvem objeto que representa o agendamento que foi feito.
- Pode-se utilizar este objeto para cancelar a execução no futuro.
- 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);
```

Atividade - Banner Rotativo

1. Implemente um banner rotativo na index.html 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.

Crie o arquivo banner.js e o adicione no HTML da index.html

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

Na banner.js:

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

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

setInterval(trocaBanner, 4000);
```

Atividade - Banner Rotativo

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

Dica: use o clearInterval para interromper a execução.

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

Pause/Play

Pode-se criar um novo link para controlar a animação

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

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

Edita o código anterior que chamava o setInterval para pegar o retorno.

Objeto que controla o interval permite desligá-lo

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

Com isso, o código que controla o pause e play fica

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

Para estilizar o botão como pause ou play trabalha-se 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;
}
```