Co-funded by the
Erasmus+ Programme
of the European Union

code4sp
coding for social promotion

Materiais de Formação em JavaScript
Subcapítulo 4 – JavaScript Avançado

WP3: Materiais de Formação do Code4SP

Elaborado por: CD DR

spel · CITIZENS IN POWER · Center for Social Innovation · CD DR · ZAUG gGmbH · Action · social hackers academy · GSPE

Co-funded by the
Erasmus+ Programme
of the European Union

Subcapítulo 4 – JavaScript Avançado

Co-funded by the
Erasmus+ Programme
of the European Union

Dia e Hora em JavaScript

Pode criar um novo objeto de data ao passar uma linha de representação de uma data ao construtor Date(), ou ao passar os componentes individuais de uma data como parâmetros para o construtor Date() (ano, mês, dia, hora, minuto, segundo, milissegundo).

```
// Create a date object for today's date:
const now = new Date();
// Outputs today's date and time in standard format
console.log(now);
```

Testemos as várias possibilidades desta funcionalidade:
https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Date

Co-funded by the
Erasmus+ Programme
of the European Union

Operações matemáticas em JavaScript

Pode realizar operações numéricas usando o JavaScript. Isto pode ser feito com a ajuda de operadores de + (adição), - (subtração), * (multiplicação), / (divisão) and % (módulos).

No exemplo em baixo usamos o operador de adição para somar dois números:

```
var x = 10; var y = 5; var z = x + y;
```

Isto irá dar-nos o valor de z como 15.

Testemos as várias possibilidades desta funcionalidade:
https://www.w3schools.com/js/js_arithmetic.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Tipo de Conversões em JavaScript

É possível converter um valor para um tipo de dados especifico usando métodos built-in, como parseInt() para converter um valor para um número interno, ou parseFloat() para converter um valor num float.

É também possível usar o método Number() para converter um valor num número, ou o método Boolean() para converter um valor num booleano.

No seguinte exemplo iremos converter uma linha num número usando o método Number():

```
var x = "100"; var y = Number(x); console.log(y); // 100.
```

Testemos as várias possibilidades desta funcionalidade:
https://www.w3schools.com/js/js_type_conversion.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Listagem de Eventos em JavaScript

As listagens de eventos JavaScript permite definir funções que irão correr quando ocorrer um evento em específico num elemento do DOM.
Existe um número de eventos diferentes que podem ocorrer num elemento DOM, quando, por exemplo, o elemento é carregado, passado por cima ou até mesmo quando o conteúdo dos elementos for alterado.

Para adicionar uma listagem de eventos num elemento, é necessário, primeiro, selecionar o elemento usando um dos métodos de seleção DOM, como document.querySelector() ou document.getElementById() .

Uma vez que o elemento é selecionado, pode usar o método addEventListener() para anexar uma listagem de eventos no elemento.

O método addEventListener() inclui dois argumentos: o nome dos eventos a ouvir e uma função que corra quando ocorre o evento.

Por exemplo, para adicionar um clique a uma listagem de eventos, é necessário usar o seguinte código:

```
button.addEventListener ( "click" , function ( ) { console.log ( "The button was clicked!" ) ; } );
```

Testemos as várias possibilidades desta funcionalidade:
https://www.w3schools.com/js/js_htmldom_eventlistener.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Propagação de Eventos em JavaScript

Quando um evento ocorre num elemento, o evento pode ser reconhecido pelo JavaScript e agirem de acordo com tal.
Por exemplo, se carregar num botão numa página web, pode programar o JavaScript para reconhecer o evento e agir.

O evento que ocorreu foi um evento *click*.

Quando carrega num elemento, o evento *click* é feito no elemento que foi carregado.

O evento propaga, então, na árvore DOM.

Isto significa que, se houver um manipulador num elemento parente, esse manipulador de evento será executado.

Se houver um manipulador de eventos num elemento avô, esse manipulador de eventos será executado.

A propagação nas árvores DOM continua até o evento atingir o elemento de raiz da árvore DOM ou até o evento parar.

Alguns exemplos de propagação de eventos:
https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation

Co-funded by the
Erasmus+ Programme
of the European Union

Empréstimo de Métodos em JavaScript

Usaremos o exemplo de um objeto que contem uma propriedade que precisa de emprestar.

```
var myObject = {

someProperty: "foo",

};
```

Podemos emprestar a propriedade someProperty de myObject da seguinte forma:

```
var myProperty = myObject.someProperty;
```

myProperty vai agora conter o elemento "*foo*".

Mais informação acerca de empréstimo de métodos:
https://www.tutorialrepublic.com/javascript-tutorial/javascript-borrowing-methods.php

Co-funded by the
Erasmus+ Programme
of the European Union

Hoisting em JavaScript

Hoisting é um comportamento único do JavaScript. É um conceito que é, muitas vezes, incompreendido. Muitos programadores acreditam que o JavaScript faz hoist de expressos de variáveis de funções para o topo do escopo.

Isto não é de todo correto. O que o JavaScript faz, na verdade, é mover as expressões para o topo do escopo e não a inicialização. Isto pode levar a alguns comportamentos inesperados.

Considere o seguinte código:

```
var foo = 1; function bar() { if (!foo) { var foo = 10; } console.log(foo); } bar();
```

Qual é que acha que vai ser o resultado deste código?

Se adivinhou 10, está errado. O resultado deste código é 1.
Isto deve-se à declaração de *foo* estar em *hoist* no topo do escopo, mas a inicialização não. Quando a expressão for executada, foo é indefinido e é, assim, um conjunto de 10.

Isto pode ser um pouco confuso, mas é importante entender como funciona o *hoisting* em JavaScript. Pode ajudar a evitar alguns erros comuns.

Mais informações acerca de *hoisting*:
https://www.w3schools.com/js/js_hoisting.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Encerramentos em JavaScript

Os encerramentos em JavaScript são uma função interior que tem acessi às variáveis exteriores (encerramento) da cadeia de escopo de funções. O encerramento tem três cadeias de escopos: tem acesso ao seu próprio escopo (as variáveis definidas entre as chavetas), tem acesso às funções exteriores das variáveis e tem acesso às variáveis globais.
Os encerramentos de JavaScript são criados quando a função interior é feita dentro da função exterior. Os encerramentos são usados com bastante frequência nas bibliotecas de JavaScript como jQuery.

Aqui está um simples exemplo de um encerramento em JavaScript:

```
function outerFunction(x) {
  var innerFunction = function(y) {
    return x + y;
  }
  return innerFunction;
}

var add5 = outerFunction(5);
var add10 = outerFunction(10);

console.log(add5(2)); // 7
console.log(add10(2)); // 12
```

Mais informação acerca de encerramentos:
https://www.w3schools.com/js/js_function_closures.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Lidar com Erros em JavaScript

JavaScript é uma linguagem flexível, o que significa que existem várias maneiras de escrever código. Esta flexibilidade pode levar a erros no programa se não houver cuidado sobre a sua estruturação.

Uma forma usual de lidar com os erros é usar os blocos *try/catch*. Estes blocos permitir-lhe-ão "tentar" algum bocado de código e "apanhar" alguns erros que possam ocorrer. A sintaxe para um bloco *try/catch* é assim:

```
try { // Code to try goes here } catch (error) { // Code to handle errors goes here }
```

O código dentro do bloco "*try*" irá ser realizado primeiro. Se não ocorrerem erros, o código no bloco "*catch*" nunca será corrido. No entanto, se houver um erro, a execução do código irá diretamente para o bloco "*catch*" e qualquer outro código no bloco "*try*" será ultrapassado.

Descubra as várias possibilidades de lidar com erros em JavaScript:
https://www.w3schools.com/js/js_errors.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Expressões Regulares em JavaScript

As expressões regulares são ferramentas poderosas usadas para criar padrões de combinação e funções “search-and-replace” no texto.

A expressão regular é, basicamente, uma sequência de caracteres que definem um padrão de busca particular. Por exemplo, o seguinte *regex* iria emparelhar qualquer linha que contenha um A maiúsculo: /A/

Uma expressão regular literal é, simplesmente, uma linha que contem um padrão que o programador deseja encontrar, fechado por duas barras (/). Por exemplo:

```
const regex = /abc/;
console.log(regex); // => /abc/
```

Descubra as várias possibilidades das expressões regulares:
https://www.w3schools.com/jsref/jsref_obj_regexp.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Formulário de Validação em JavaScript

O JavaScript é uma linguagem de script da visão do cliente, o que significa que o script corre no computador do visitante. Isto permite que se façam coisas como verificar se o endereço de email foi inserido corretamente antes de ser enviado para o servidor. Reduz também a tensão no servidor porque os dados dos formulários não têm que ser submetidos até depois de terem sido verificados os seus erros pelo JavaScript.

Descubra as várias possibilidades de formulários de validação do JavaScript: https://www.w3schools.com/js/js_validation.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Cookies em JavaScript

Criar cookies em JavaScript é bastante simples. É apenas necessário usar o objeto do documento e o seu método createCookie(). A sintaxe desta função assemelha-se ao seguinte:

```
document.cookie = "name=value; expires=date";
```

O parâmetro do nome representa o nome da cookie, enquanto que o valor significa o seu conteúdo (linha). Se quiser estabelecer uma data de validade, adicione outro atributo de nome *expires* com uma linha válida de Date() como valor, ou passe, simplesmente, para nulo, caso não precise dele.

Mais informação acerca de Cookies:
https://www.w3schools.com/js/js_cookies.asp

Co-funded by the
Erasmus+ Programme
of the European Union

Requisitos de Ajax em JavaScript

Ajax é uma técnica de desenvolvimento para criar aplicações web interativas. O objetivo do Ajax é fazer páginas web sentirem-se com maior resposta ao trocar pequenas quantidades de dados com o servidor nos bastidores, para que a página completa não tenha que ser recarregada cada vez que o utilizador faz alguma alteração.

Uma vez que tenha incluído a biblioteca Ajax na sua página web, pode começar a fazer requisitos ao servidor. Por exemplo, se quisesse carregar alguns dados de um ficheiro num servidor, deve usar o seguinte código:

```
$.ajax({ url: 'data.json', success: function(data) { // do something with the data } });
```

Este código fará um pedido a Ajax para o URL 'data.json'. Se este pedido for bem-sucedido, a função *callback* em "*success*" será corrido com os dados do servidor como argumento.

Experimente as várias possibilidades dos requisitos de Ajax em JavaScript:
https://www.w3schools.com/xml/ajax_intro.asp

Co-funded by the
Erasmus+ Programme
of the European Union

OBRIGADO!

Co-funded by the
Erasmus+ Programme
of the European Union