Linguagem JavaScript

Você vai estudar os conceitos básicos e as principais funcionalidades do JavaScript: uma das linguagens de programação mais importantes e utilizadas na internet.

Prof. Alexandre de Oliveira Paixão

Propósito

Para aplicação dos exemplos, será necessário um editor de texto com suporte à marcação HTML. No sistema operacional Windows, é indicado o Notepad++ ou o VS Code. No Linux, o Nano Editor. Lembrando que todas essas ferramentas são gratuitas. Uma alternativa aos editores instalados no computador são os interpretadores on-line, como CodePen e JSFiddle.

Para começar a nossa jornada, acesse os códigos-fontes originais propostos para o aprendizado de JavaScript. Baixe e você poderá utilizar os códigos como material de apoio ao longo do conteúdo.

Objetivos

- Identificar os conceitos básicos, a sintaxe e as formas de utilização do JavaScript.
- Aplicar as estruturas de decisão e de repetição.
- Identificar o conceito de vetor e sua utilização em JavaScript.
- Reconhecer os recursos assíncronos Ajax e JSON.

Introdução

Segundo Flanagan (2011), JavaScript é a linguagem de programação da Web mais utilizada pelos sites. Além disso, todos os navegadores – estejam eles em desktops, jogos, consoles, tablets ou smartphones – incluem interpretadores JavaScript, fazendo dela a linguagem de programação mais onipresente da história.

Ao longo deste conteúdo, veremos os conceitos básicos e as principais funcionalidades do JavaScript. Além disso, aprenderemos a integrá-lo às páginas HTML e a utilizá-lo – desde tarefas básicas, como manipular a interface DOM, às mais complexas, como transmitir dados entre o cliente e o servidor de forma assíncrona.

Quer saber mais sobre essa linguagem? Para isso, assista ao vídeo a seguir.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Árvore DOM

O JavaScript é uma linguagem *client side*, ou seja, uma linguagem que roda no lado do cliente, considerando o ambiente web. Nesse ambiente, além do JavaScript, temos ainda duas outras principais tecnologias: a linguagem de marcação HTML e o CSS. No vídeo a seguir, você aprenderá a utilizar o JavaScript para interagir com páginas HTML por meio da árvore DOM.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

JavaScript

Esta linguagem faz parte da tríade de tecnologias que compõem o **ambiente web**. Veja a seguir:

HTML

Cuida da estrutura e do conteúdo das páginas.

CSS

É responsável pela apresentação das páginas.

JavaScript

Cuida do comportamento e da interatividade das páginas web.

Trata-se de uma linguagem de programação **interpretada e multiparadigma**, uma vez que possui suporte à programação estruturada, orientada a objetos e funcional.

O JavaScript, conhecido como JS, foi criado inicialmente para o ambiente web, no lado cliente. Entretanto, evoluiu a ponto de atualmente ser utilizado também no lado servidor, além de ser uma das principais linguagens para o desenvolvimento de aplicativos mobile.

> Embora possuam nomes parecidos, as linguagens de programação JavaScript e Java não têm nenhuma relação.

Interface DOM

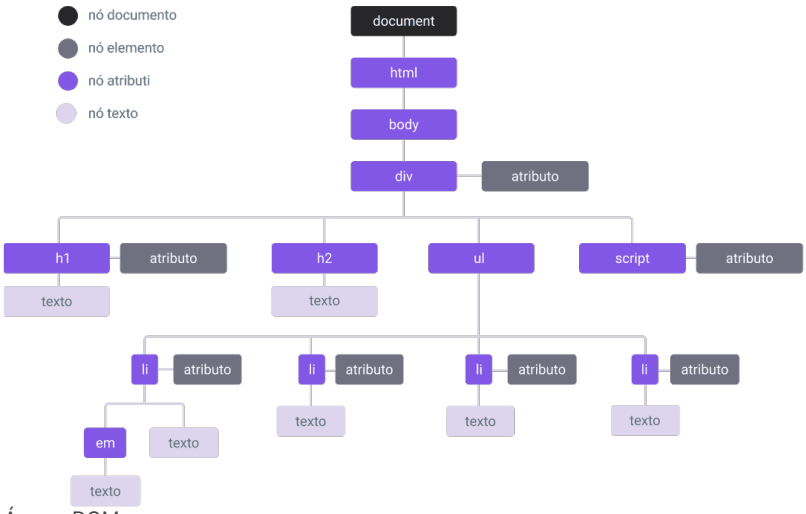
Iniciaremos nosso estudo de JavaScript entendendo o que é e como manipular a interface, ou árvore, DOM. Isso auxiliará o entendimento de como essa linguagem se integra e interage com a HTML.

A sigla DOM (*document object model*) significa modelo de objeto de documento. Trata-se de uma interface que permite a manipulação, via programação, de documentos HTML (e outros, como XML, por exemplo). Além de interface, é também chamado de árvore, por fornecer uma representação estruturada do documento nesse formato.

A árvore DOM é composta por nós e objetos. Ambos possuem propriedades e métodos, além de eventos. Por meio dessa estrutura, é possível acessar, por exemplo, o conteúdo textual de um elemento – como a tag

> ⚠
>
> Atenção
>
> Embora sejam frequentemente associados, o DOM não faz parte do JavaScript, podendo também ser manipulado por outras linguagens. Neste conteúdo, ficaremos restritos à manipulação do DOM por meio do JavaScript.

Veja a seguir uma ilustração da árvore DOM e de seus elementos:



Árvore DOM

Manipulando o DOM com JavaScript

Por meio do JavaScript, é possível:

- Inserir dinamicamente, em tempo de execução, novos elementos à árvore DOM, mesmo que eles não façam parte da estrutura inicial da página.

- Excluir e alterar elementos preexistentes ou dinamicamente criados.

- Manipular os estilos de um documento, de modo similar ao que é feito via CSS.

Os elementos preexistentes ou dinamicamente criados, porém, existirão somente enquanto durar a sessão atual do usuário, ou seja, trata-se de uma manipulação dinâmica e dependente de estado, de ações e interações por parte do usuário, mas que se perderão quando ele sair da página.

Atividade 1

JavaScript é uma linguagem tipicamente *client side*, embora também utilizada, mais recentemente, no lado servidor. Sobre sua utilização no lado cliente, e mais precisamente sobre sua relação com o DOM, assinale a afirmativa correta.

A

JavaScript permite que a estrutura inicial de uma página HTML seja modificada. Além disso, como também é uma linguagem com suporte do lado servidor, ela permite que esses códigos HTML modificados sejam salvos na página HTML original.

B

Um script JS pode ser incluído tanto no corpo do documento HTML como por meio de um arquivo externo. A diferença principal entre essas duas formas está no fato de que o código inserido diretamente na HTML faz parte da árvore DOM – sendo, portanto, a única forma de manipular os elementos dessa interface.

C

Com a utilização da linguagem JavaScript, é possível ter acesso à árvore DOM. Com isso, tarefas como a modificação de elementos existentes e a inclusão de novos elementos, assim como conteúdos, tornam-se possíveis.

D

Os códigos JavaScript incorporados ao final da página não permitem a manipulação da árvore DOM, já que são interpretados apenas após o carregamento de todos os elementos.

E

JavaScript permite que novos elementos sejam inseridos dinamicamente na árvore DOM, desde que esses elementos façam parte da estrutura inicial da página.

A alternativa C está correta.

Por meio de JavaScript, é possível manipular a árvore DOM, independentemente do modo de incorporação ao documento HTML, e os estilos CSS aplicados no documento. A única ressalva diz respeito a eventos de manipulação que tentem acessar os nós e elementos DOM antes que toda a página seja renderizada, como visto em um dos exemplos demonstrados.

Sintaxe e conceitos

A linguagem JavaScript oferece uma série de recursos, além de possuir uma sintaxe própria de utilização. Veremos, no vídeo a seguir, alguns desses recursos e como fazer para incorporar código JS em páginas HTML.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Sintaxe JavaScript

Observe o código a seguir:

```
java




 Resultado da Multiplicação:
```

O código visto mostra tanto as formas de inserção do JavaScript em uma página HTML (linhas 7 e 13) quanto alguns aspectos de sua sintaxe. Esses aspectos serão vistos em detalhes a seguir, mas, antes de começarmos, recomendamos que você copie o código e o execute – pode ser em seu computador ou utilizando uma ferramenta de codificação on-line, como o CodePen ou JSFiddle.

Existem duas formas de inserir comentários no código JavaScript. Observe!

Uma linha

Utilizamos as duas barras “//”.

Múltiplas linhas

Utilizamos “/*” e “*/”.

A seguir, vamos conhecer a sintaxe de JavaScript, na qual serão utilizados os números das linhas do editor do código anterior para facilitar a localização de fragmentos e elementos abordados.

Variáveis

Vamos declarar as variáveis utilizando a palavra reservada “var”, sucedida por seu nome. Não devem ser utilizados caracteres especiais como nomes de variáveis.

Dica

Embora existam diferentes convenções, procure utilizar um padrão e segui-lo ao longo de todo o seu código para a nomeação das variáveis, sobretudo as compostas.

Na linha 25, a palavra composta “resultado multiplicação” foi transformada em uma variável por meio do padrão camelcase, ou seja, a segunda palavra (e demais palavras, quando for o caso) é iniciada com a primeira letra maiúscula.

Outra característica importante de uma variável em JS é que, por se tratar de uma **linguagem fracamente tipada**, não é necessário declarar o tipo de dado. Com isso, uma variável que armazenará um número (inteiro, decimal etc.) e outra que armazenará uma palavra (string, char etc.) são declaradas da mesma forma.

Linguagem fracamente tipada

Uma linguagem é considerada fracamente tipada quando não é necessário informar o tipo de dado no momento de criação de uma variável.

Após declaradas, as variáveis podem ser utilizadas sem a palavra reservada “var”, como visto nas linhas 19 e 33 do código anterior.

Atribuição

As variáveis precisam ser declaradas antes de sua utilização. Entretanto, há uma forma simplificada, vista na linha 25, na qual é feita uma atribuição de valores no momento de declaração da variável “resultadoMultiplicacao”.

> A respeito da atribuição de valores, embora declarados da mesma maneira, os tipos de dados são atribuídos de formas distintas.

Um número pode ser atribuído diretamente a uma variável, enquanto uma string precisará estar envolta em aspas – simples ou duplas. A linha 13 mostra dois números sendo atribuídos à “variável”.

Ao atribuirmos uma string a uma variável, obtemos:

```
javascript

var nomeDisciplina = “Javascript”
```

Ponto e vírgula

Repare o final de cada linha de código – todas as linhas foram terminadas com a utilização de um ponto e vírgula.

Recomendação

Diferentemente de outras linguagens, em JavaScript, não é obrigatória a utilização de caracteres para indicar o final de uma linha de código. Porém, seguindo uma linha de boas práticas, adote uma convenção e a utilize em todo o seu código.

Outros elementos

Ao longo do código apresentado anteriormente, foram utilizados outros elementos. Na tabela a seguir, cada um deles será descrito:

Elemento	Para que serve	Linha do código
alert	Exibir uma caixa de diálogo no navegador. Existem outras funções nativas de diálogo, inclusive para receber input de valores.	23
document.getElementById	Referenciar um elemento da árvore DOM por meio do valor de seu atributo “id”. Pesquise também sobre document.getElementByClassName.	28
innerHTML	Propriedade DOM relativa ao conteúdo de um elemento. Permite tanto a inclusão quanto a exclusão e a modificação do conteúdo do elemento.	33
+=	Operador de atribuição composta. A linha em questão poderia ser escrita de forma simplificada com esse operador, ou em sua forma completa, como: “divLocal.innerHTML = divLocal.innerHTML + resultadoMultiplicacao”.	33
function	Palavra reservada, utilizada para indicar que será declarada uma função. É procedida pelo nome da função e por parênteses. Caso receba parâmetros, eles devem ser declarados dentro dos parênteses._ipsum	36
return	Palavra reservada, utilizada para indicar o conteúdo a ser retornado pela função. Nem todas as funções retornam valores. Logo, essa instrução só deve ser utilizada por funções que retornem algum resultado.	48

Lista de elementos utilizados no código de exemplo
Alexandre de Oliveira Paixão

Incorporando o JavaScript à HTML

A incorporação do JavaScript a páginas HTML pode ser feita de duas maneiras:

Códigos no corpo da página

Incluindo os códigos diretamente no corpo da página – utilizando a tag .</body>

Arquivos externos

Mediante arquivos externos, com extensão .js, linkados ao documento também dentro da seção .

Para a otimização do desempenho do carregamento da página, devemos mover todo o código JavaScript para seu final, após o fechamento da tag .

Atenção

Cuidado para não mover códigos ou scripts incluídos que sejam necessários à correta visualização ou aos comportamentos da página, já que os códigos movidos para o final só serão lidos e interpretados após todo o restante da página.

No código demonstrado na seção Sintaxe Javascript, você poderá ver os exemplos de incorporação do JS em uma página HTML. Todo o código está comentado, a fim de descrever o que acontece em cada linha.

Atividade 2

A respeito dos tipos e da utilização de variáveis em JavaScript, analise as afirmativas a seguir:

I. Os valores podem ser atribuídos quando a variável é declarada.
II. Valores de qualquer tipo podem ser atribuídos da mesma forma.
III. JavaScript é uma linguagem fracamente tipada. Logo, não é necessário informar o tipo de dado no momento de criação da variável.

Está correto o que se afirma em:

A

Somente II.

B

I e III.

C

II e III.

D

Somente I.

E

I e II.

A alternativa B está correta.

JavaScript é uma linguagem bastante flexível em relação à declaração e utilização de variáveis. Entretanto, alguns cuidados são necessários, entre eles a atribuição de valores do tipo string, que precisam ser englobados por aspas – duplas ou simples.

JavaScript na prática

Na aprendizagem de qualquer linguagem de programação, os exercícios práticos são fundamentais para a fixação do conteúdo e o entendimento de suas particularidades, de sua sintaxe e de seus demais recursos. Então, vamos praticar? Para isso, assista ao vídeo a seguir.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Nesta prática, utilizaremos como ponto de partida o código visto na seção Sintaxe JavaScript. A partir dele:

- Peça ao usuário para inserir dois números inteiros positivos.
- Armazene os números inseridos pelo usuário em duas variáveis.
- Crie uma função para dividir números inteiros.
- Exiba na tela uma caixa de diálogo com o resultado da divisão precedido pela frase “O resultado da divisão é igual a:”.

Por fim, caso queira ir além, crie uma calculadora para realizar as quatro operações matemáticas. Nesse caso, você precisará pedir ao usuário que escolha a operação a ser realizada, além dos números de entrada.

Como solução, teremos o seguinte código:

```
javascript
```

Faça você mesmo!

Vamos retomar um código já apresentado no conteúdo:

```
javascript









 Resultado da Multiplicação:
```

Faça as seguintes alterações no código:

- Mova o código que está entre as linhas 12 e 51, inclusive, para dentro da seção <head>.
- Salve a alteração.
- Carregue novamente sua página.

Assinale a alternativa correta que demonstra o resultado da execução do código, após modificado, no navegador web.

A

O resultado é o mesmo exibido antes da modificação do script, ou seja, será exibido o “alert” com o resultado da soma e, na div “exibe_resultado”, o resultado da multiplicação.

B

O “alert” é exibido, mas o resultado da multiplicação não é exibido na div “exibe_resultado”.

C

O “alert” não é exibido, mas o resultado da multiplicação é exibido na div “exibe_resultado”.

D

Nem o “alert” nem o resultado da multiplicação, na div, são exibidos.

E

É exibida, no corpo da página, uma mensagem dizendo que ocorreu um erro durante a execução do script.

A alternativa B está correta.

Ao final da execução do script, quando a página web é carregada, o alerta é exibido, mas a div “exibe_resultado” não recebe o valor de resultado da multiplicação. Isso acontece porque, quando está no início da página, o código é lido pelo navegador antes que o restante seja renderizado. Portanto, a tag , por exemplo, ainda não foi carregada e não está presente na árvore DOM.

Estruturas condicionais

Neste vídeo, você vai compreender as estruturas condicionais do JavaScript, entre elas as instruções if, else, else if e switch. Confira!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Segundo Flanagan (2011), as estruturas de decisão, também conhecidas como condicionais, são instruções que executam ou pulam outras instruções, dependendo do valor de uma expressão especificada. São os pontos de decisão do código, também conhecidos como ramos, uma vez que podem alterar o fluxo do código, criando um ou mais caminhos.

Aprofundando o conceito

Para melhor assimilação do conceito de estruturas condicionais, vamos usar um exemplo a partir do código construído anteriormente, como veremos a seguir:

```
javascript




 Resultado da Multiplicação:
```

As orientações do programa afirmam que deve ser realizada a divisão de dois números inteiros positivos.

O que acontece se o usuário inserir um número inteiro que não seja positivo? Ou como forçá-lo a inserir um número positivo?

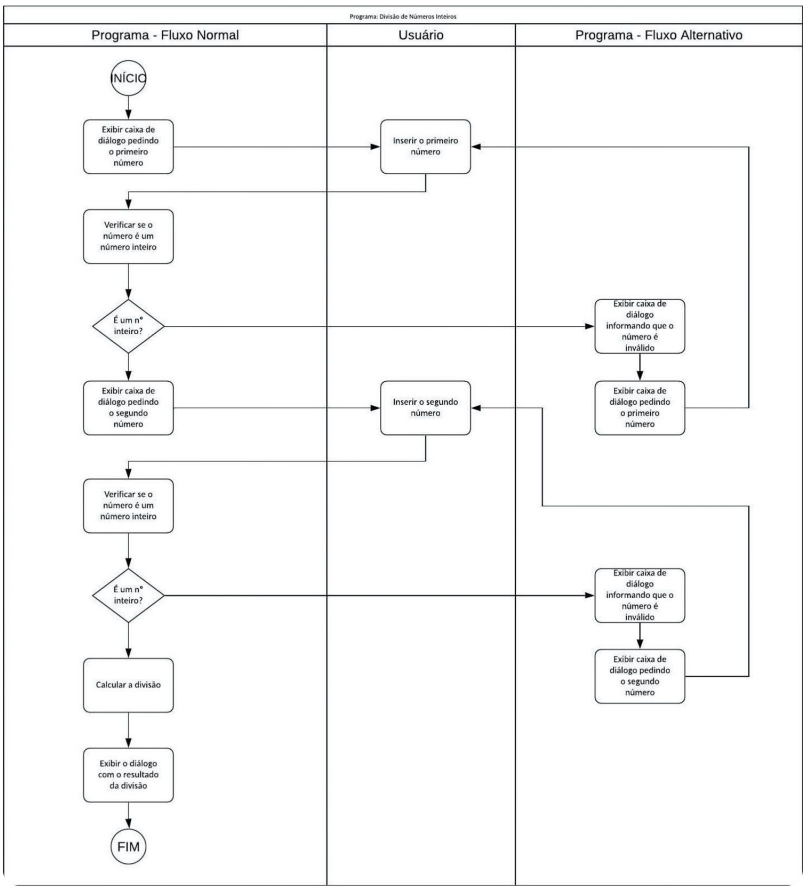
Para essa função, podemos utilizar uma condição, ou seja, se o usuário inserir um número inteiro não positivo, deve-se avisar que o número não é válido, solicitando que seja inserido um número válido.

Nesse caso, o fluxo normal do programa é receber dois números positivos, calcular a divisão e exibir o resultado. Perceba que a condição cria um novo fluxo, um novo ramo, em que outro diálogo é exibido, e o usuário é levado a inserir novamente o número.

O fluxo normal e o fluxo resultado da condicional podem ser vistos na imagem a seguir, em que são apresentados os passos correspondentes ao nosso exercício, separando as ações do programa e as do usuário.

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Fluxo normal e fluxo alternativo

Repare que a verificação “é um nº inteiro positivo” permite apenas duas respostas: “sim” e “não”. Essa condição, mediante a resposta fornecida, é responsável por seguir o fluxo normal do código ou o alternativo.

O fluxograma de exemplo foi simplificado para fornecer mais detalhes. Logo, a respectiva notação padrão não foi utilizada em sua confecção.

Nas linguagens de programação, utilizamos as instruções condicionais para implementar o tipo de decisão apresentado no exemplo. Em JavaScript, estão disponíveis as instruções "if/else" e "switch", como veremos a seguir.

Instrução “If”

A sintaxe da instrução "if/else" em JavaScript possui algumas formas. A primeira e mais simples é apresentada do seguinte modo:

if (condição) instrução

Nessa forma, é verificada uma única condição. Caso seja verdadeira, a instrução será executada. Do contrário, não. Antes de continuarmos, cabe destacar os elementos da instrução:

- É iniciada com a palavra reservada “if”.
- É inserida, dentro de parênteses, a condição (ou condições).
- É inserida a instrução a ser executada, caso a condição seja verdadeira.

Outro detalhe importante: caso exista mais de uma instrução para ser executada, é necessário envolvê-las em chaves. Veja o exemplo:

```
javascript

if (condição1 && condição2){
instrução1;
instrução2;
}
```

Nesse segundo caso, além de mais de uma instrução, também temos mais de uma condição. Quando é necessário verificar mais de uma condição, em que cada uma delas precisa ser verdadeira, utilizamos os caracteres “&;&;”.

Na prática, as instruções 1 e 2 só serão executadas caso as condições 1 e 2 sejam verdadeiras. Vamos a outro exemplo:

```
javascript

if (condição1 || condição2){
instrução1;
instrução2;
}
```

Repare que, nesse código, os caracteres “&;&;” foram substituídos por “||”. Esses últimos são utilizados quando uma ou outra condição precisa ser verdadeira para que as instruções condicionais sejam executadas.

> E o que acontece se quisermos verificar mais condições?

Nesse caso, podemos fazer isso tanto para a forma em que todas as condições precisam ser verdadeiras, separadas por “&;&;”, quanto para a forma em que apenas uma deve ser verdadeira, separadas por “||”. Além disso, é possível combinar os dois casos na mesma verificação. Veja o exemplo:

```
javascript

if ( (condição1 && condição2) || condição3){
instrução1;
instrução2;
}
```

Nesse fragmento, as duas primeiras condições são agrupadas por parênteses. A lógica aqui é a seguinte:

Execute as instruções 1 e 2 SE as condições 1 e 2 forem verdadeiras OU se a condição 3 for verdadeira.

Por fim, há outra forma: a de negação.

> Como verificar se uma condição é falsa (ou não verdadeira)?

Veremos a seguir:

```
javascript

if (!condição1){
instrução1;
instrução2;
}
```

O sinal “!” é utilizado para negar a condição. As instruções 1 e 2 serão executadas caso a condição 1 não seja verdadeira.

Vamos praticar?

Nos três emuladores de código a seguir, apresentamos as estruturas de decisão vistas até o momento. No primeiro emulador, temos o uso da estrutura de decisão “if” de maneira simples, contendo apenas uma única condição:

Conteúdo interativo

esse a versão digital para executar o código.

Já no emulador seguinte, a estrutura de decisão “if” é implementada com duas condições, além dos operadores lógicos AND (&;&;) e OR (||):

Conteúdo interativo

esse a versão digital para executar o código.

Por fim, no emulador a seguir, temos a estrutura “if” sendo usada de uma maneira mais elaborada, com mais de duas condições, combinação dos operadores &;&; e II, assim como o uso do operador lógico de negação NOT (!):

Conteúdo interativo

esse a versão digital para executar o código.

Instrução “else”

A instrução “else” acompanha a instrução “if”. Embora não seja obrigatória, como vimos nos exemplos, sempre que “else” for utilizado, deve vir acompanhado de “if”. O “else” indica se alguma instrução deve ser executada caso a verificação feita com o “if” não seja atendida. Vejamos:

```
javascript

if(número fornecido é inteiro e positivo){
     Guarde o número em uma variável;
}else{
     Avise ao usuário que o número não é válido;
     Solicite ao usuário que insira novamente um número;
}
```

Perceba que o “else” (senão) acompanha o “if” (se). Logo, SE as condições forem verdadeiras, faça isto; SENÃO, faça aquilo.

No último fragmento, foi utilizado, de modo proposital, **português-estruturado** nas condições e instruções. Isso porque, mais adiante, você mesmo codificará esse "if/else" em JavaScript.

> Português-estruturado
>
> Linguagem de programação ou pseudocódigo que utiliza comandos expressos em português.

Instrução “else if”

Veja o exemplo a seguir:

```
javascript

if (numero1 < 0){
     instrução1;
     }else if(numero == 0){
          instrução2;
     }else{
          instrução3;
     }
```

Repare que uma nova instrução foi usada no fragmento. Trata-se de “else if”, instrução utilizada quando queremos fazer verificações adicionais sem agrupá-las todas dentro de um único “if”. Além disso, ao utilizarmos essa forma, caso nenhuma das condições constantes no “if” e no(s) “if else” seja atendida, a instrução “else” será executada obrigatoriamente ao final.

> Recomendação
>
> Otimize os códigos presentes nos emuladores anteriores usando o “else if”. Como exemplo, apresentamos o código do primeiro emulador modificado, no qual as quatro estruturas de decisão com “if” foram transformadas em uma única estrutura de decisão.

Note que, antes, eram geradas duas saídas redundantes (“a é maior que b” e “b é menor que a”), pois se tratava de quatro estruturas independentes. Por isso, todas elas eram avaliadas. Isso não ocorrerá mais com o uso de uma estrutura de decisão composta de “if” e “else if”, pois, quando a primeira condição verdadeira for encontrada (“a é maior que b”), nenhuma das outras condições será avaliada. Logo, teremos:

```javascript
var a = 10;
var b = 3;

console.log ("if com uma única condição:");
if (a > b){
     console.log("a é maior que b");
} else if (a == b){
     console.log("a é igual a b");
} else if (a < b){
     console.log("a é menor que b");
} else if (b < a){
     console.log("b é menor que a");
}
```

Instrução “switch”

A instrução “switch” é bastante útil quando uma série de condições precisa ser verificada. É bastante similar à instrução “else if”. Vejamos:

```javascript
switch(numero1){
     case 0:
          instrução1;
          break;
     case 1:
          instrução2;
          break;
     default:
          instrução3;
          break;
}
```

De maneira geral, o switch é usado quando há uma série de condições, nas quais diversos valores para a mesma variável são avaliados. Vamos detalhar o código anterior:

- Após o “swicth” dentro de parênteses, temos a condição a ser verificada.
- A seguir, temos os “case”, em quantidade equivalente às condições que queremos verificar.
- Depois, dentro de cada “case”, temos a(s) instrução(ões) e o comando “break”.
- Por fim, temos a instrução “default”, que será executada caso nenhuma das condições representadas pelos “case” seja atendida.

Atividade 1

Quanto às estruturas de decisão, mais precisamente a instrução “switch”, analise as afirmativas a seguir:

I. Com essa instrução, conseguimos realizar verificações que não são possíveis apenas utilizando “if” e “else”.
II. Essa instrução é uma forma de reduzir a complexidade proveniente da utilização de vários “if” e “else”.
III. Essa instrução é utilizada para testar várias opções de condicionais.

Está correto o que se afirma em:

A

Somente II.

B

I e III.

C

II e III.

D

Somente I.

E

I e II.

A alternativa C está correta.

A instrução "switch", assim como as instruções "if/else", permite que o fluxo de um programa seja alterado a partir de verificações de condicionais. Logo, essas instruções não se diferem, sendo a "switch" mais utilizada em situações em que há muitas condições a serem verificadas, diminuindo, assim, a complexidade do código caso fosse utilizado "if/else".

Estruturas condicionais na prática

Após estudar a teoria sobre as estruturas condicionais na linguagem JavaScript, chegou a hora de praticar um pouco. No vídeo a seguir, vamos codificar um script, no qual algumas condições serão verificadas e, mediante o resultado delas, um ou mais resultados serão exibidos. Vamos lá?

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Para realização da prática proposta, siga estes passos:

1

Crie um script “.js” utilizando o editor ou ide de sua preferência.

2

Declare duas variáveis e atribua a ambas os seguintes valores: 5 e -1, respectivamente.

3

Crie uma instrução “if” que verifique se a primeira variável é maior que a segunda E se a segunda variável é maior que 0 (zero).

4

Caso a instrução acima seja verdadeira, exiba no console.log a mensagem: “A primeira variável é maior que a segunda e a segunda variável é um número positivo”.

5

Crie uma instrução “if” que verifique se a primeira variável é maior que a segunda E a segunda é menor que 0 (zero).

6

Caso a instrução acima seja verdadeira, exiba no console.log a mensagem: “A primeira variável é maior que a segunda e a segunda variável não é um número positivo”.

7

Crie uma instrução “if” que verifique se a primeira variável é menor que a segunda variável OU se a segunda variável é maior que 0 (zero).

8

Caso a instrução acima seja verdadeira, exiba no console.log a mensagem: “A primeira variável é menor que a segunda ou a segunda variável é um número positivo”.

Como solução, teremos o seguinte código:

```
javascript

var num1 = 5;
var num2 = -1;

if (num1 > num2 && num2 > 0){
     console.log("A primeira variável é maior que a segunda e a segunda variável é um
número positivo");
}
if (num1 > num2 && num2 <= 0){
     console.log("A primeira variável é maior que a segunda e a segunda variável não é um
número positivo");
}
if (num1 < num2 || num2 > 0){
     console.log("A primeira variável é menor que a segunda ou a segunda variável é um
número positivo");
}
```

Faça você mesmo!

Com base na atividade prática proposta, qual das alternativas produz o resultado a seguir?

“O nome dele é João Francisco e ele tem 4 anos de idade.”

A

if (nome = “João Francisco” \ idade = 4);

B

if (nome = João Francisco \ idade = 4);

C

if (nome === João Francisco \ idade === 4);

D

if (nome === “João Francisco” || idade === 4);

E

if (nome === “João Francisco” \ idade === 4);

A alternativa E está correta.

A comparação de uma string deve ser feita encapsulando/englobando seu valor com aspas (duplas ou simples). Além disso, para verificar duas condições em uma instrução condicional, devemos utilizar o operador &&. Por fim, utilizamos três sinais de igual (===) quando desejamos comparar tanto o valor quanto o tipo de dado da variável.

Estruturas de repetição

Neste vídeo, você vai conhecer as estruturas de repetição do JavaScript: for, while, do/while e for/in.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

As estruturas de repetição – também chamadas de laços – permitem que uma ação seja executada de maneira repetitiva. Logo, sempre que uma ação recorrente se fizer necessária, no lugar de criar N linhas de código para repetir tal ação, podemos executá-la dentro de um laço, diminuindo, assim, nosso código, além de facilitar o trabalho.

for

No emulador de código a seguir, podemos ver a sintaxe do laço “for”: uma das estruturas de repetição presentes no JavaScript.

> Em sua forma mais comum, no laço ”for”, há uma variável, chamada normalmente de contador, que recebe um valor inicial (no exemplo, 0) e é incrementada (pelo “++”) até atingir uma condição (ser menor que 10).

Observe:

Conteúdo interativo

esse a versão digital para executar o código.

Algumas variações possíveis nesse código seriam iniciar o contador em 1, por exemplo (o mais usual, em programação, é iniciarmos nossos índices em zero), ou ir até o número 10. Desse modo, no lugar do sinal de menor, utilizaríamos o “menor igual”, dessa forma: “ <= 10”. Teste essas variações no emulador anterior e veja as diferenças.

A seguir, veremos as outras estruturas de repetição, além do “for”, presentes na linguagem JavaScript.

while

Veja o fragmento a seguir para entender o comportamento do laço “while” (enquanto):

Conteúdo interativo

esse a versão digital para executar o código.

Esse código tem o mesmo resultado daquele visto no exemplo utilizando o laço “for”. Sua sintaxe é simples: “Enquanto uma condição fornecida for verdadeira, faça isso.” Clique em Executar no emulador anterior e verifique!

do/while

Embora semelhante ao laço “while”, há uma diferença fundamental entre ambos. Observe:

do/while	while
A condição é testada no final.	A condição é testada no início.

Com isso, pelo menos uma vez, a instrução (ou as instruções) do laço “do/while” será, obrigatoriamente, executada. Veja o exemplo:

Conteúdo interativo

esse a versão digital para executar o código.

Execute o código no emulador anterior e veja que o resultado é exatamente igual ao do código usando “while”.

Agora, modifique a última linha do código do laço “do/while” para:

} while (contador < 0);

Execute novamente o código e perceba que, apesar de a condição nunca ter sido verdadeira, o bloco de instruções do laço foi executado uma vez. Essa é a diferença entre o laço “do/while” e o laço “while”.

for/in

Assim como os demais laços em uma linguagem de programação, o “for/in” é bastante utilizado com arrays (vetor ou matriz contendo dados não ordenados).

> Normalmente, precisamos percorrer o conteúdo de um array e manipular ou apenas exibir seu valor. Para isso, podemos usar laços.

Na emulador a seguir, serão apresentados dois fragmentos de código para uma mesma função: um utilizando “for” e outro, “for/in”.

Conteúdo interativo

esse a versão digital para executar o código.

Analisando o código, é possível notar as diferenças de sintaxe entre os laços “for” e “for/in”. Observe:

“for”	“for/in”
Definimos uma variável contador (“i”) e devemos percorrer o array “frutas” começando em seu índice 0 até seu tamanho (length).	Declaramos uma variável contador (“fruta”) e dizemos ao código que percorra o array imprimindo seu conteúdo a partir do índice fornecido – nesse caso, a variável “fruta”.

Atividade 2

Observe o fragmento de código a seguir:

```
var cont = 1;
do{
cont += 1;
```

```
}while (cont < 10);
alert(cont);
```

Após sua execução, qual é o valor da variável cont – exibida na instrução "alert(cont)"?

A

10

B

1

C

9

D

11

E

2

A alternativa A está correta.

O laço "do/while" executa a primeira instrução antes de testar a condição fornecida. Nesse caso, a instrução consiste em incrementar, de 1 em 1, o valor da variável "cont". Como se inicia em 1 e vai até 9, ao final, seu valor será 10.

Estruturas de repetição na prática

Para fixar melhor os conceitos sobre estruturas de repetição, vamos praticar um pouco. Neste vídeo, vamos criar um script em que utilizaremos o laço de repetição “for” para realizar a soma de números naturais. Vamos lá?

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Para realizar a prática proposta, siga estes passos:

1

Crie um script “.js” utilizando o editor ou ide de sua preferência.

2

Declare duas variáveis – uma para conter o resultado da soma (somaTotal) e outra para conter a quantidade máxima de interações (qtdeInteracoes) – e atribua a elas os seguintes valores: 0 e 50, respectivamente.

3

Crie um laço “for” que seja decrementado de 1 em 1, começando na quantidade máxima de interações e chegando a 2 (ou a maior do que 1).

4

Dentro do laço, incremente em uma unidade a variável que armazena o resultado da soma.

5

Ao final do script, fora do laço “for”, imprima o resultado da soma.

Como solução, teremos o seguinte código:

```
javascript

let somaTotal = 0;
const qtdeInteracoes = 50;
for(let i = qtdeInteracoes; i >= 1; i-- ) {
     somaTotal += i; // equivalente a utilizar somaTotal = somaTotal + i
}
console.log('soma total:',somaTotal);
```

Faça você mesmo!

Considere o fragmento de código a seguir, em que um laço “while” é utilizado:

```
var contador = 0;
while (contador <= 3) {
console.log("O valor do contador é: " + contador);
contador++
}
```

Assinale a alternativa que corresponde à saída do código:

A

“O valor do contador é: 3”

B

"O valor do contador é: 0"
"O valor do contador é: 1"
"O valor do contador é: 2"

C

“O valor do contador é: 4”

D

"O valor do contador é: 0"
"O valor do contador é: 1"
"O valor do contador é: 2"
"O valor do contador é: 3"

E

“O valor do contador é: 2”

A alternativa D está correta.

A variável "contador" é inicializada com o valor "0". Como condição do laço "while", foi definido que o código, dentro do laço, deverá ser executado enquanto a variável "contador" for menor ou igual a "3". Consequentemente, o resultado da execução do script será:

"O valor do contador é: 0"
"O valor do contador é: 1"
"O valor do contador é: 2"
"O valor do contador é: 3"

Composição e criação de vetores

Neste vídeo, você vai compreender a composição e a criação de vetores de repetição no JavaScript.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Um vetor é uma estrutura de dados simples utilizada para armazenamento de objetos do mesmo tipo. Vamos aprender mais sobre o assunto!

Função e composição dos vetores

Na literatura relacionada, a linguagens de programação, um vetor é chamado de array.

> Normalmente, um array é tratado em conjunto com outra estrutura: a matriz.

Em linhas gerais:

Vetor

É um array unidimensional.

Matriz

É um array multidimensional (um vetor de vetores).

A função prática de um vetor é simplificar a utilização de variáveis. No exemplo apresentado anteriormente, vimos que eram necessárias duas variáveis para guardar os números solicitados ao usuário. Com a utilização de um vetor, porém, precisaríamos de apenas uma variável de duas posições.

Exemplo

Imagine que seja necessário armazenar as notas de 50 alunos para, ao final, calcular as respectivas médias. Embora possa parecer sem importância o uso de arrays nesse caso, seriam necessárias 50 variáveis (ou apenas 1 array).

Os vetores também são vistos na matemática quando usamos uma tabela organizada em linhas e colunas no formato m X n, sendo m o número de linhas e n, o de colunas.

> Um vetor é composto por uma coleção de valores, e cada um deles é chamado de elemento. Além disso, cada elemento possui uma posição numérica dentro do vetor, conhecida como índice.

No exemplo a seguir, usando notação da linguagem JavaScript, veja um array contendo nomes de frutas:

```javascript
var frutas = ['Laranja', 'Uva', 'Limão'];
```

Nesse exemplo, “Laranja”, “Uva” e “Limão” são os elementos do vetor “frutas”. Considerando que o índice de um array inicia em 0, temos: o conteúdo do vetor “frutas” na posição/índice 0 é “Laranja”; na posição/índice 1 é “Uva”; e na posição/índice 2 é “Limão”.

A seguir, veremos como declarar e utilizar vetores na linguagem JavaScript.

Criação e utilização de vetores

Em JavaScript, **os vetores não possuem tipo**, a exemplo do que vimos quando tratamos das variáveis. Logo, é possível criar um array composto por números, strings, objetos e até mesmo outros arrays.

Comentário

Em JS, um vetor pode ter, no máximo, 4.294.967.295 elementos. Outra característica importante é que, em JavaScript, os arrays possuem tamanho dinâmico, ou seja, não é necessário informar o tamanho do vetor ao declará-lo.

Vejamos mais alguns exemplos de criação de vetores em JS:

```javascript
var alunos = []; //array vazio
var alunos = ['Alex', 'Anna', 'João']; // array de strings
var notas = [10.0, 9.5, 9.5]; // array de números decimais
var mistura = ['Um', 2, 3, 'Quatro']; //array de diversos tipos de dados
```

Outra forma de criação de vetores em JavaScript é usando o construtor (conceito relacionado à programação orientada a objetos) array. Veja o exemplo:

```javascript
var alunos = new Array();
var alunos = new Array('Alex', 'Anna', 'João');
```

Atividade 1

Em relação aos conceitos e ao uso de vetores em JavaScript, analise as afirmativas a seguir:

I. Um vetor ou array é um grupo de variáveis que contém valores do mesmo tipo ou de tipos diferentes.
II. É possível acessar o último elemento de um array da seguinte forma: vetor[vetor.length-1].
III. Um array só permite dados do mesmo tipo.

Está correto o que se afirma em:

A

Somente II.

B

I e III.

C

II e III.

D

Somente I.

E

I e II.

A alternativa E está correta.

A linguagem JavaScript permite que um array seja composto por dados de diferentes tipos.

Manipulação de vetores

Neste vídeo, você vai aprender a acessar, exibir e remover elementos do vetor no JavaScript, além de conhecer a propriedade lenght.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A manipulação de vetores em JavaScript pode ser feita de forma direta ou por meio da utilização de funções predefinidas. Vamos conhecer algumas das opções disponíveis para essa tarefa?

Acesso e exibição de elementos do vetor

Em termos de acesso aos elementos de um array, a forma mais simples é utilizar seu índice. Vamos ao exemplo:

```
javascript

var alunos = ['Alex', 'Anna', 'João']; // array de strings
alert(alunos[0]); // exibirá “Alex” na caixa de diálogo
```

Observe e entenda o seguinte:

- A função “alert” imprimirá o conteúdo da posição zero do array “alunos”, ou seja, “Alex”.

- Seguindo a mesma lógica, se quiséssemos imprimir “João”, utilizaríamos o índice 2.

Outra forma de acessar e exibir os elementos de um vetor é usando um laço de repetição. Veja novamente o exemplo contido no emulador de código da seção “for/in”.

A linguagem JavaScript possui métodos nativos para tratamento de arrays. Em termos de acesso e manipulação, veremos, agora, como utilizar o push.

push

Para compreender em que situações o método push pode ser útil, vamos voltar a nosso vetor “alunos”. Imagine que, após ter sido declarado inicialmente com 3 valores, seja necessário incluir novos valores a esse array, em tempo de execução. O método push nos auxilia nessa tarefa. Sua sintaxe é:

nome_do_array.push(valor)

Usando nosso array de exemplo, poderíamos adicionar um novo elemento. Desse modo:

```
javascript

alunos.push('Helena');
```

É possível, ainda, inserir múltiplos valores utilizando push:

```
javascript

alunos.push('Helena', 'Maria');
```

Tamanho do array

Há outras maneiras de adicionar elementos a um array de forma dinâmica. A primeira delas pode ser vista a seguir:

```
javascript

alunos[alunos.length] = 'Maria';
```

Nesse caso, devemos utilizar o tamanho do array para informar que desejamos adicionar um novo elemento. Isso pode ser feito informando o número, caso o saibamos, ou de forma dinâmica, usando a propriedade length – que retorna justamente o tamanho do array. Vamos entender melhor essa importante propriedade!

Propriedade length

Uma das necessidades mais comuns quando se trabalha com arrays é saber o seu tamanho. Como vimos em alguns de nossos exemplos, em JavaScript, está disponível a propriedade length, que retorna o tamanho ou número de elementos de um array. Sua sintaxe é:

nome_do_array.length

splice

É um método multiuso em JavaScript que serve tanto para excluir elementos de um array, como veremos a seguir, quanto para substituir e inserir. Sua sintaxe é:

Array.splice(posição,0,novo_elemento,novo_elemento,...)

Em que:

1. ‘posição’
   É o índice no qual o novo elemento será incluído.

2. ‘0’
   Indica ao método que nenhum elemento do array será excluído.

3. ‘novo_elemento’
   É o novo elemento que se deseja adicionar ao array.

Vejamos um exemplo:

```
javascript

var alunos = ['Alex', 'Anna', 'João'];
alunos.splice(3,0,'Helena');
console.log(alunos); //imprimirá 'Alex', 'Anna', 'João', 'Helena'
```

Além disso, com esse método, também é possível substituir um dos elementos do array. Veja o exemplo:

```
javascript

var alunos = ['Alex', 'Anna', 'João'];
alunos.splice(1,1,'Helena');
Console.log(alunos); //imprimirá 'Alex, 'Helena', 'João'
```

Aqui, ao passarmos o número 1 como segundo parâmetro, informamos ao método que um elemento, o de índice 1, deveria ser excluído. Entretanto, como inserimos ao final o nome ‘Helena’, o método realizou a substituição do elemento excluído pelo novo elemento inserido.

Remoção de elementos do vetor

Delete

Em JavaScript, a remoção de elementos de um array pode ser feita com a utilização do método nativo delete. Veja como esse método funciona utilizando nosso array de exemplo:

```
javascript

delete frutas[0];
```

Observe que sua sintaxe é composta por:

- Nome do método (delete);
- Nome do array;
- Índice do elemento que queremos remover.

Esse método possui uma particularidade: embora o valor seja excluído do array, este não é “reorganizado”, permanecendo com o mesmo tamanho.

pop

Este método, que não recebe parâmetros, remove um elemento do final do array, atualizando seu tamanho. Sua sintaxe é:

frutas.pop();

shift

Embora seja similar ao pop, este método remove um elemento do início do array. Após a remoção, este é **reindexado**, ou seja, o elemento de índice 1 passa a ser o de índice 0, e assim sucessivamente. Além disso, o tamanho do array também é atualizado. Sua sintaxe é:

frutas.shift();

Outras formas de remover elementos do vetor

Existem outras maneiras para excluir elementos de um array. Uma forma simples é determinar o tamanho do array utilizando a **propriedade length**. Isso fará com que ele seja reduzido ao novo tamanho informado.

```
javascript

var primos = [2,3,5,7,11,13,17];
alert(primos.length); //imprimirá 7
primos.length = 4;
console.log(primos.length); //imprimirá 4
```

Nesse exemplo, ao definirmos o tamanho do array como 4, ele será reduzido, sendo mantidos os elementos do índice 0 ao 3 e excluídos os demais. Utilize o emulador a seguir para praticar a remoção de elementos um pouco mais.

Conteúdo interativo

esse a versão digital para executar o código.

Existe, ainda, outro método para a remoção de elementos de um array: filter. Entretanto, ele não modifica o vetor original, mas **cria um novo a partir dele**.

Comentário

O método filter utiliza uma sintaxe mais complexa, assim como conceitos e funções de call-back que fogem ao escopo deste conteúdo.

Atividade 2

Deseja-se excluir o último elemento do array a seguir:

var pares = [2,4,6,8,10,12];

Analise os métodos a seguir para realizar essa ação:

I. pares.splice(5,1);
II. pares.splice(6,0,0);
III. delete(pares[5]);

Está correta a aplicação dos seguintes métodos:

A

Somente II.

B

I e III.

C

II e III.

D

Somente I.

E

I e II.

A alternativa B está correta.

O método splice pode ser utilizado tanto para remover quanto para adicionar ou substituir elementos de um array. Quando usado para remover, sua sintaxe corresponde ao código pares.splice(5,1), em que indicamos o índice e a quantidade de elementos, a partir dele, a ser removida. Já o código pares.splice(6,0,0) faz com que seja adicionado um novo elemento, com valor 0, após o índice 6.

Vetores na prática

Neste vídeo, vamos realizar algumas atividades práticas para fixar todos os conceitos vistos sobre vetores em JavaScript.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Para criar e manipular os dados de um vetor em JavaScript, siga estes passos:

- Solicite ao usuário que insira dois números inteiros positivos.
- Utilize um vetor para armazenar esses dois números.
- Verifique se os números inseridos são inteiros positivos. Caso contrário, solicite ao usuário para inseri-los novamente.
- Divida os dois números inteiros positivos.

Há mais de uma forma de desenvolver o programa. Logo, não há código certo ou errado.

Recomendação

Neste exercício, você pode utilizar funções JavaScript para organizar melhor seu código. Inclusive, pode usar um pouco de recursividade.

Como solução, teremos o seguinte código:

javascript

```
//Declaração do array 'numeros' sem tamanho definido e sem elementos atribuídos
var numeros = [ ];

//O primeiro elemento (o de índice 0) recebe o retorno da função que solicita o primeiro
número
numeros[0] = soliticaPrimeiroNumero();

//O segundo elemento (o de índice 1) recebe o retorno da função que solicita o segundo
número
numeros[1] = soliticaSegundoNumero();

//Declaração de atribuição de valor à variável que armazenará o resultado da divisão
//O resultado da divisão virá da função 'divida' (essa função recebe como parâmetro o
array 'numeros')
var resultadoDivisao = divida(numeros);

//Exibindo o resultado da divisão na tela
alert('O resultado da divisão é igual a: ' + resultadoDivisao);

/*
Função Javascript
Esta função não recebe parâmetros
*/
function soliticaPrimeiroNumero(){
     //Declaração e atribuição de variável. Ela receberá o número inserido pelo usuário
var numero1 = prompt("Insira o primeiro número: ");

     //Condição para verificar se o número é positivo.
     //Caso não, o retorno da função será chamar a própria função novamente.
     // Esta operação será repetida até que um número válido seja inserido.
     //Caso sim, retorna o valor inserido pelo usuário
     if(numero1 < 0){
          alert("O número precisa ser inteiro e positivo");

          //Este tipo de retorno, onde a própria função é chamada novamente, é conhecido
como recursividade
          return soliticaPrimeiroNumero();
}else{
          return numero1;
     }
}

function soliticaSegundoNumero(){
     var numero2 = prompt("Insira o segundo número: ");

     if(numero2 < 0){
          alert("O número precisa ser inteiro e positivo");
          return soliticaSegundoNumero();
     }else{
          return numero2;
     }
}

/*
Esta função recebe como parâmetro um array - que contém os 2 números que desejamos dividir
*/
function divida(numeros){
     var resultado = 0;

     //Os números a serem divididos são acessados através dos índices do array
     resultado = numeros[0] / numeros[1];
     return resultado;
}
```

Faça você mesmo!

Vamos retomar um código já apresentado no conteúdo:

```
javascript









 Resultado da Multiplicação:
```

Faça as seguintes alterações no código:

- Mova o código que está entre as linhas 12 e 51, inclusive, para dentro da seção.
- Salve a alteração.
- Carregue novamente sua página.

Assinale o método que nos permite acessar o segundo elemento do array:

arr = [5,9,15]

A

arr.indexOf(2)

B

arr.indexOf(1)

C

arr[2]

D

arr[1]

E

arr[arr.length-3]

A alternativa D está correta.

O acesso a um elemento do array pode ser feito por meio de seu índice. Considerando que esse índice se inicia em 0, para acessarmos o segundo elemento, deveremos utilizar o índice 1.

Requisições assíncronas em JavaScript

Neste vídeo, você vai compreender as requisições assíncronas na linguagem de programação JavaScript.

> **Conteúdo interativo**
>
> Acesse a versão digital para assistir ao vídeo.

O conceito de requisição no ambiente web diz respeito às informações solicitadas ou submetidas no lado cliente – por meio do navegador, por exemplo – e tratadas pelo lado servidor, que, após processar a requisição, devolverá uma resposta ao solicitante. Ainda no contexto do ambiente web, há dois tipos de requisições: síncronas e assíncronas.

Requisições síncronas e assíncronas

No ambiente web, há dois tipos possíveis de **requisições**. Vamos conhece-lâs a seguir:

Síncronas

Quando realizadas, bloqueiam o remetente, ou seja, o cliente faz a requisição e fica impedido de realizar qualquer nova solicitação até que a anterior seja respondida pelo servidor. Com isso, só é possível realizar uma requisição de cada vez.

Assíncronas

Quando realizadas, não são dotadas de sincronismo. Logo, várias requisições podem ser realizadas em simultâneo, independentemente de ter havido resposta do servidor às solicitações anteriores.

E a qual tipo de requisição devemos dar preferência?

> Em comparação às requisições síncronas, devemos dar preferência à utilização das assíncronas, porque elas não apresentam os problemas de desempenho e congelamento do fluxo da aplicação – naturais das síncronas.

Como as requisições assíncronas funcionam na prática?

Para entendermos melhor o funcionamento das requisições assíncronas, vamos usar como exemplo o feed das redes sociais. Nessas páginas, novos conteúdos são carregados sem que seja necessário recarregar o navegador e, consequentemente, todo o conteúdo visto.

Eventos como a rolagem de tela, por exemplo, fazem com que os conteúdos sejam carregados do servidor e exibidos na tela do dispositivo.

AJAX

Em JavaScript, quando tratamos de requisições assíncronas, naturalmente, falamos de AJAX (Asynchronous JavaScript and XML). Esse termo foi empregado pela primeira vez em 2005 e engloba o uso não de uma, mas

de várias tecnologias: HTML (ou XHTML), CSS, JavaScript, DOM, XML (e XSLT), além do elemento mais importante, o objeto XMLHttpRequest.

Resumindo

A utilização de AJAX permite que as páginas e aplicações Web façam requisições a scripts do lado servidor e carreguem, de forma rápida e muitas vezes incremental, novos conteúdos sem que seja necessário recarregar a página inteira.

Embora o “X” no acrônimo se refira a XML, esse não é o único formato disponível. Além dele, temos:

- HTML;
- Arquivos de texto;
- JSON.

Este último o mais utilizado atualmente. Veremos sobre ele mais adiante.

Em relação aos recursos para realização de requisições, há dois disponíveis em JS:

- Objeto XMLHttpRequest;
- Interface Fetch API.

XMLHttpRequest

Inicialmente, foi implementado no navegador Internet Explorer por meio de um objeto do tipo ActiveX. Posteriormente, outros fabricantes fizeram suas implementações, dando origem ao XMLHttpRequest, que se tornou o padrão atual.

> Versões antigas do Internet Explorer só possuem suporte ao ActiveX.

O XMLHttpRequest possui alguns métodos e algumas propriedades. Veja um exemplo simples de sua utilização:

```
javascript
```

Imagens Aleatórios de Cachorros

```
 A partir do click no botão abaixo uma nova imagem aleatória de cachorros será carregada
utilizando requisições assíncronas com XMLHttpRequest


Aguardandando a imagem ser carregada


        Carregar Imagens
```

O código anterior contém tanto funcionalidades JavaScript vistas neste conteúdo quanto algumas novas, além do XMLHttpRequest, que veremos mais adiante. Ao utilizar esse código, você terá um exemplo real de dados sendo requisitados a um servidor – nesse caso, uma API que retorna imagens aleatórias de cachorros – e exibidos na página, sem que ela seja recarregada a cada nova requisição/click no botão.

Por ora, vamos nos concentrar no XMLHttpRequest. Utilizando as linhas contidas no código anterior, observe:

- Na linha 19, uma instância do objeto é criada. Esse é o primeiro passo para sua utilização. A partir desse ponto, toda referência deverá ser feita pelo nome da variável utilizada (em nosso exemplo, xmlHttpRequest).

- A linha 22 mostra a utilização do método open, que recebe três parâmetros: o método de requisição dos dados, a url remota/do servidor que queremos acessar e o tipo de requisição – onde “true” define que será feita uma requisição assíncrona e “false”, uma síncrona. Esse argumento é opcional. Logo, pode não ser definido, assumindo o valor padrão “true”.

- Continuando o código, na linha 24 temos a propriedade “onreadystatechange”, que monitora o status da requisição XMLHttpRequest – propriedade “readyState” – e especifica uma função a ser executada a cada mudança.

- Repare, agora, na linha 25: o status 3 significa que a requisição ainda está sendo processada. Logo, poderíamos, por exemplo, exibir em nossa tela uma mensagem (ou imagem, como é muito comum) avisando que a informação requisitada está sendo carregada. Perceba que, dependendo do tempo de resposta do servidor remoto, nem sempre será possível ver essa informação.

- Já na linha 28, temos o tratamento do status quando ele for igual a 4, ou seja, quando a requisição estiver concluída. Além da propriedade “readyState”, também poderíamos monitorar a propriedade “status”, que armazena o código de resposta do servidor Http utilizado pela XMLHttpRequest.

- Ainda na linha 30, repare que também foi utilizado outro método: o parse. Esse método não pertence ao objeto XMLHttpRequest, mas é necessário quando o recurso requisitado devolve o conteúdo em formato JSON.

- Por fim, na linha 41, é utilizado o método send, que envia a requisição.

Outros métodos e outras propriedades

O código anterior mostra um exemplo simples do que é possível fazer utilizando AJAX. Além disso, apenas algumas propriedades e alguns métodos foram vistos.

Saiba mais

Na seção Explore +, estão disponíveis sugestões de conteúdo que permitirão um aprofundamento do uso do AJAX. É recomendável a leitura desse material, assim como a implementação e até mesmo modificação do código anterior para melhor assimilação do conteúdo.

API Fetch

Esta API é, em termos conceituais, similar a XMLHttpRequest – ou seja, permite a realização de requisições assíncronas a scripts do lado servidor. Entretanto, por ser uma implementação mais recente, essa interface JavaScript apresenta algumas vantagens, como:

1

O uso de promise ("Promise" significa promessa. É um objeto utilizado para processamento assíncrono, representando um valor que pode estar disponível agora, no futuro ou nunca).

2

O uso em outras tecnologias, como service workers, por exemplo ("Service workers" são scripts executados em segundo plano no navegador, separados da página web).

O código a seguir apresenta o mesmo exemplo utilizado no tópico sobre XMLHttpRequest, mas substituindo o XMLHttpRequest pela API Fetch. Vejamos alguns métodos e algumas propriedades:

```
javascript
```

Imagens Aleatórios de Cachorros

```
A partir do click no botão abaixo uma nova imagem aleatória de cachorros será carregada
utilizando requisições assíncronas com XMLHttpRequest


Aguardando a imagem ser carregada


        Carregar Imagens
```

Em relação à sua sintaxe, podemos notar algumas semelhanças com a XMLHttpRequest:

- URL do servidor remoto – definida na linha 19 e utilizada na linha 20.

- Fetch options – em nosso exemplo, utilizamos apenas um parâmetro, o método. Esse parâmetro, inclusive, é opcional. Veja a linha 21, na qual declaramos o GET, que é o método padrão. Além desse parâmetro, há outros disponíveis.

- Tipo de dado retornado pela requisição – veja a linha 24, onde foi utilizado o objeto correspondente ao tipo de dado retornado pela requisição – nesse caso, JSON. Há outros tipos de objetos, como texto e até mesmo bytes, sendo possível, por exemplo, carregar imagens, arquivos pdf, entre outros.

Ainda em relação à sintaxe, merece destaque a forma, própria, como a API Fetch trata a requisição (request) e seu retorno (response).

Resumindo

Quando o método fetch é executado, retorna uma promessa (promise) que fornece a resposta (response) à requisição, sendo esta bem-sucedida ou não.

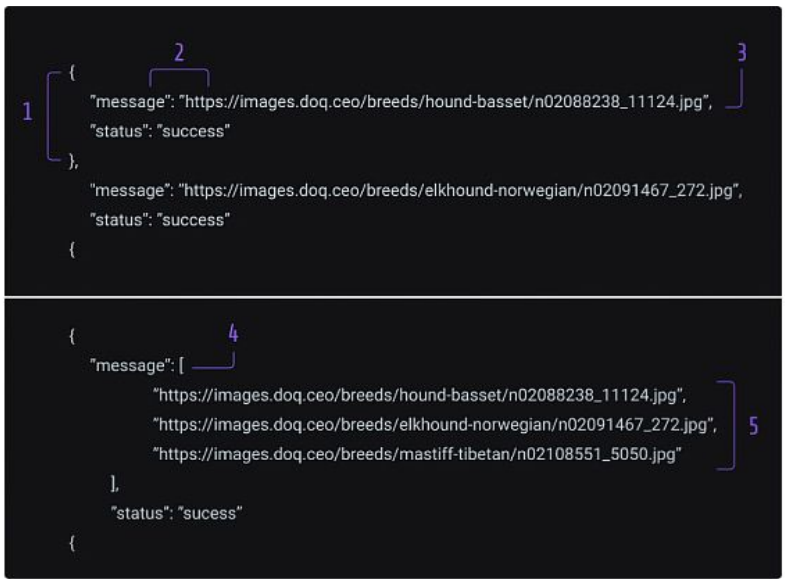
JSON

Pode ser traduzido para notação de objetos JavaScript. Trata-se de um tipo ou formatação leve para troca de dados. Essa, inclusive, é sua principal vantagem em relação aos outros tipos. Além disso, destaca-se também sua simplicidade para ser interpretado tanto por pessoas quanto por “máquinas”.

Embora normalmente associado ao JavaScript – tendo sido definido, inclusive, na especificação ECMA-262, de dezembro de 1999 –, o JSON é um formato de texto independente de linguagem de programação.

> A facilidade do uso em qualquer linguagem contribuiu para que o JSON se tornasse um dos formatos mais utilizados para a troca de dados.

A imagem a seguir apresenta alguns exemplos da estrutura de um objeto JSON:



Estrutura de objetos JSON

Observe a seguir a legenda da imagem:

1. O objeto é englobado por chaves;
2. Objeto JSON (par nome/valor);
3. Os valores são separados por vírgulas;
4. Os valores do array são englobados por colchetes;
5. Array.

Como vemos, um objeto JSON tem as seguintes características:

- É composto por um conjunto de pares nome/valor – “status” é o nome de um objeto e “success”, seu valor. Esses pares são separados por dois pontos “:”.

- Utiliza a vírgula para separar pares, valores ou objetos.

- O objeto e seus pares são englobados por chaves “{ }”.

- É possível definirmos arrays, que são englobados por colchetes “[ ]”.

O JSON fornece suporte a uma gama de tipos de dados. Além disso, possui alguns métodos, como o JSON.parse(), visto em um de nossos exemplos.

Saiba mais

A seção Explore + sugere materiais adicionais sobre o JSON. Recomendamos a leitura desse material.

Atividade 1

Sobre as requisições assíncronas em JavaScript − AJAX, analise as afirmativas a seguir:

I. Essas requisições tornam a interação na página mais lenta, já que dependem do retorno de dados que são requisitados ao servidor.
II. As requisições assíncronas não bloqueiam o cliente – por exemplo, o navegador Web –, permitindo que outras operações sejam realizadas enquanto se aguarda o retorno da requisição.
III. Requisições assíncronas não apresentam problemas de congelamento do fluxo da aplicação e de desempenho.

Está correto o que se afirma em:

A

Somente II.

B

I e III.

C

II e III.

D

Somente I.

E

I e II.

A alternativa C está correta.

As requisições assíncronas tornam a interação mais rápida no cliente, uma vez que a página não fica bloqueada, aguardando o retorno do servidor. Isso permite que outras ações, incluindo novas requisições, sejam realizadas.

AJAX na prática

Chegou a hora de praticarmos tudo o que vimos até aqui. Neste vídeo, você vai aprender a implementar uma requisição assíncrona utilizando AJAX.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Para validar os conceitos sobre requisições assíncronas em JavaScript (AJAX), crie uma página HTML e os códigos JavaScript, conforme instruções a seguir:

1

A página HTML deverá conter um título introdutório e um parágrafo, informando ao usuário que ele deverá clicar no botão disponível a seguir, cujo título deverá ser “Clique para carregar imagens aleatórias”.

2

Crie um botão, conforme mencionado, que, ao ser clicado, chamará uma função JavaScript.

3

Na função JavaScript, utilize a API Fetch para acessar a URL https://dog.ceo/api/breeds/image/random e ler o JSON retornado a partir dela.

4

O JSON mencionado retorna, como uma de suas chaves, uma URL para uma imagem, aleatória, de cachorros. Essa URL deverá ser atribuída ao atributo “src” de uma tag contida no HTML – tudo isso utilizando JavaScript.

5

Os possíveis erros provenientes da chamada da URL, pela API Fetch, deverão ser tratados no código.

Durante a realização do exercício, lembre-se:

Dica

Há mais de uma forma de desenvolver o programa. Logo, não há código certo ou errado.

Como solução, teremos o seguinte código:

```
javascript
```

Imagens Aleatórios de Cachorros

A partir do click no botão abaixo uma nova imagem aleatória de cachorros será carregada utilizando requisições assíncronas com XMLHttpRequest

Aguardando a imagem ser carregada

Carregar Imagens

Atividade 2

Observe o código a seguir e repare que há um fragmento incompleto: a linha “if (????)”.

```
javascript
function runAjaxRequest() {
    var xhttp = new XMLHttpRequest();
     xhttp.onreadystatechange = function() {
          //Verificar o status do response
         if ( ???? ) {
              document.getElementById("demo").innerHTML =
               this.responseText
         }
     };
     xhttp.open("GET", "ajax_info.txt", true);
     xhttp.send();
}
```

É uma boa prática verificar o status da resposta da requisição, a fim de conferir se ela foi executada com sucesso e, a partir daí, coletar seu resultado e utilizá-lo. Há, em linhas gerais, duas propriedades do objeto XMLHttpRequest que podemos verificar para esse fim. Assinale a alternativa que apresenta esses dois objetos.

A

this.responseText \ this.responseXML;

B

xhttp.success \ xhttp.OK;

C

this.readyState == 4 \ this.status == 200;

D

this.httpGetStatus == 200 \ this.state == 4;

E

this.ajax.result == success \ this.status == 200;

A alternativa C está correta.

Os objetos do XMLHttpRequest que podem ser usados para verificar se a requisição foi executada com sucesso são: this.readyState == 4 &;&; this.status == 200. Esses objetos podem ser usados em conjunto ou isoladamente, mas devemos dar preferência à primeira opção.

JSON na prática

Após termos visto alguns conceitos e a estrutura de um documento no formato JSON, vamos manipular uma string em tal formato, utilizando JavaScript e o método fetch. Neste vídeo, você aprenderá a implementar uma requisição assíncrona utilizando JSON.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Uma das formas mais comuns de manipularmos strings no formato JSON é coletarmos esse dado por meio de requisições a APIs REST. Na prática proposta, utilizamos uma API free, que possui dados fake, conforme estes passos:

1

Nosso recurso, a string JSON, será consumido a partir da URL https://dummyjson.com/products/1. Se você abrir esse recurso diretamente no navegador, terá como resposta uma string JSON, com dados falsos de um produto.

2

Para consumir o recurso anterior, crie uma página HTML e, dentro dela, o código JavaScript necessário para realizar a requisição. Use a API Fetch para essa tarefa.

3

Há diferentes formas de realizar o passo anterior. Por exemplo, você pode inserir um botão no HTML por meio do qual um método JS será acionado. Dentro desse método, você incluirá a chamada ao recurso.

4

Analise a string retornada a partir da URL anterior para entender quais chaves estão disponíveis.

5

Por fim, imprima – pode ser a partir do “console.log” – os valores de todas as chaves contidas na string JSON.

Durante a realização do exercício, lembre-se que há mais de uma forma de desenvolver o programa. Logo, não há código certo ou errado.

Como possível resolução, teremos o seguinte código:

```
javascript



        Carregar Json
```

Faça você mesmo!

Crie, em seu computador, em qualquer pasta, dois arquivos. O primeiro deve apresentar a extensão “.json”, contendo o seguinte conteúdo:

```
javascript


{
     "id": 1,
     "title": "Hello World",
     "description": "Example data"
}
```

O segundo arquivo, uma página html, deve ser criado na mesma pasta em que salvou o primeiro, contendo o seguinte código:

```
javascript









     Carregar Json Local
```

Em seguida, clique no arquivo html para que ele seja aberto no navegador. Com a página aberta, clique no botão “Carregar Json Local”. Por fim, abra o inspecionador de elemento e analise o conteúdo da aba “console”.

Após a execução desses passos, nenhum resultado será exibido no navegador. Entretanto, um erro será apresentado no inspecionador de elementos. Esse erro – relacionado a CORS policy – ocorre porque

A

os arquivos .json e .html não estão no mesmo diretório.

B

não é possível ler um arquivo .json por meio de JavaScript.

C

há incompatibilidade entre o navegador web e o arquivo .json.

D

não foi especificada, no código, a versão do JavaScript a ser utilizada.

E

existe uma política de segurança que impede a leitura de arquivos locais.

A alternativa E está correta.

Quando tentamos ler arquivos .json a partir de nosso próprio computador, em um diretório comum, obtemos um erro de CORS que está associado a políticas de segurança dos navegadores, as quais impedem a leitura de arquivos locais. Para que seja possível ler um arquivo .json a partir de código JavaScript, em uma página HTML, é necessário que ambos estejam hospedados em um servidor web.

Considerações finais

O que você aprendeu neste conteúdo?

- Conceitos básicos da linguagem JavaScript.
- O que é a interface/árvore DOM.
- Como manipular a árvore DOM com JavaScript.
- A sintaxe e alguns recursos de JavaScript.
- Como incorporar código JavaScript em páginas HTML.
- As estruturas de decisão e repetição disponíveis em JavaScript.
- Como utilizar as estruturas de decisão.
- Como utilizar as estruturas de repetição.
- O que são vetores.
- Como manipular vetores em JavaScript.
- Os conceitos de requisições síncronas e assíncrona.
- Como realizar requisições assíncronas (AJAX) em JavaScript.
- O que é um arquivo no formato JSON.
- Como consumir dados no formato JSON com Javascript.

Explore +

- Para saber mais sobre DOM, leia o **Modelo de Objeto de Documento** (DOM) e **Examples of Web and XML development using the DOM**, da comunidade Mozilla.
- Acesse a jQuery, uma biblioteca de funções JavaScript.
- Leia **AJAX –The XMLHttpRequest Object**, do site W3 schools.
- Para aprender mais sobre JSON, leia **JSON – Introduction**, do site W3schools.
- Acesse os sites das comunidades CodePen e JSFiddle para testar códigos HTML, CSS e JavaScript.

Referências

BARBOSA, A. **DOM**. Medium, 4 set. 2017.

FLANAGAN, D. **Javascript**: The Definitive Guide. [s.l.]: O'Reilly Media, 2011.

RAUSCHMAYER, A. **Speaking Javascript**. [s.l.]: O'Reilly Media, 2014.