PADO

Labs

INTRODUÇÃO JAVASCRIPT

Desenvolvimento Web - Aula VI

Contato

Professor


David Krepsky


david@hausenn.com.br


DKrepsky

Monitor


Kevin Araujo


kevin.araujo@hausenn.tech


NivekDesign

Java Script

- Objetos
- Array
- Date Objects
- Date Formats
- Get Date Methods
- Set Date Methods
- JSON
- Callbacks
- Chamadas Assíncronas
- Primises
- Async/Await
- Fetch

Objetos

Um objeto é uma coleção de dados e/ou funcionalidades relacionadas (que geralmente consistem em diversas variáveis e funções - que são chamadas de propriedades e métodos quando estão dentro de objetos).

Objetos

A criação de um objeto geralmente começa com a definição e a inicialização de uma variável.

```
var pessoa = {};
```

Objetos


```
var pessoa = {
  nome: ['Bob', 'Smith'],
  idade: 32,
  sexo: 'masculino',
  interesses: ['música', 'esquiar'],
  bio: function() {
    alert(this.nome[0] + ' ' + this.nome[1] + ' tem ' + this.idade + ' anos de
idade. Ele gosta de ' + this.interesses[0] + ' e ' + this.interesses[1] + '.');
  },
  saudacao: function() {
    alert('Oi! Eu sou ' + this.nome[0] + '.');
  }
};
```

Objetos

Agora você tem alguns dados e funcionalidades dentro de seu objeto e é capaz de acessá-los com uma sintaxe simples

```
pessoa.nome
pessoa.nome[0]
pessoa.idade
pessoa.interesses[1]
pessoa.bio()
pessoa.saudacao()
```

Objetos

Um objeto é composto de vários membros, cada um com um nome e um valor (ex. : ['Bob', 'Smith'] e 32). Cada par nome/valor deve ser separado por uma vírgula e o nome e valor, em cada caso, separados por dois pontos.

```
var nomeDoObjeto = {
  nomeMembro1: valorMembro1,
  nomeMembro2: valorMembro2,
  nomeMembro3: valorMembro3
};
```

Arrays

O objeto Array do JavaScript é um objeto global usado na construção de 'arrays': objetos de alto nível semelhantes a listas.

Criando um Array:

```
var frutas = ['Maçã', 'Banana'];

console.log(frutas.length);
// 2
```

Acessar um item (index) do Array


```
var primeiro = frutas[0];
// Maçã

var ultimo = frutas[frutas.length - 1];
// Banana
```

JavaScript Array Methods

Método toString()

O método JavaScript toString()converte um array em uma string de valores de array (separados por vírgula).

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Resultado:

Banana * Orange * Apple * Mango

JavaScript Array Methods

Método join()

O join()método também une todos os elementos do array em uma string. Ele se comporta como toString(), mas além disso você pode especificar o separador.

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Resultado:

Banana * Orange * Apple * Mango

JavaScript Array Methods

Método pop()

O pop()método remove o último elemento de um array:

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

O método que retorna o valor "extraído":

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

JavaScript Array Methods

Método push()

O push() método adiciona um novo elemento no final de uma array

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

O método que retorna o novo tamanho do array

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

JavaScript Array Methods

Método shift()

O shift() método remove o primeiro elemento da matriz e “desloca” todos os outros elementos para um índice inferior.

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

O método que retorna o valor que foi “deslocado”.

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```

JavaScript Array Methods

Método unshift()

O unshift() método adiciona um novo elemento ao início de um array e “desloca” os elementos mais antigos:

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

O método que retorna o novo comprimento da matriz.

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

JavaScript Array Methods

Alterando Elementos

Os elementos do array são acessados usando seu número de índice :

Os índices de matriz começam em 0:

[0] primeiro elemento da matriz

[1] segundo elemento

[3] terceiro elemento ….

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

JavaScript Array Methods

Array Length

O Length propriedade fornece uma maneira fácil de anexar um novo elemento a uma matriz:

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";
```

JavaScript Array Methods

Método delete()

Os elementos da matriz podem ser excluídos usando o operador JavaScript delete.

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];
```

JavaScript Array Methods

Método concat()

O método cria um novo array mesclando (concatenando) arrays existentes:

Exemplo (Mesclando Dois Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);
```

JavaScript Array Methods

O splice() método que adiciona novos itens a uma matriz

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

O primeiro parâmetro (2) define a posição onde novos elementos devem ser adicionados (spliced in).

O segundo parâmetro (0) define quantos elementos devem ser removidos.

O resto do parâmetro (“Lemn”, “Kiwi”) define os novos elementos a serem adicionados.

Sorting Arrays

O sort() método classifica um array em ordem alfabética

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

Reversing an Array

O reverse() método que inverte os elementos em uma array

pode usá-lo para classificar uma matriz em ordem decrescente:

Exemplo

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
fruits.reverse();
```

Sorting Arrays

Função de Comparação

A finalidade da função de comparação é definir uma ordem de classificação alternativa. A função de comparação deve retornar um valor negativo, zero ou positivo, dependendo dos argumentos:

```
function(a, b){return a - b}
```

Quando a sort() função que compara dois valores, ela envia os valores para função de comparação e classifica os valores de acordo com o valor retornado (negativo, zero, positivo)

Sorting Arrays

Numeric Sort()

Por padrão o sort() classifica os valores como strings, isso funcionaria para strings tipo “apple” vem antes de “banana”.

No entanto, se os números forem classificados como strings, “25” é maior que “100”, porque “2” é maior que “1”.

Por esse motivo o sort() produzirá um resultado incorreto ao classificar os números, pode ser corrigido fornecendo uma função de comparação

Sorting Arrays

Numeric Sort()

Exemplo

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
```

Use o mesmo método para classificar um array decrescente

Exemplo

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});
```

Array iteration

Array forEach()

Método que chama uma função (uma função de retorno de chamada) uma vez para cada elemento do array.

Exemplo

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value + "<br>";
}
```

Os métodos de iteração do array operam em cada item do array

Array iteration

Observe que a função recebe 3 argumentos:

- O value do item
- O index do item
- O array

Exemplo

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value) {
  txt += value + "<br>";
}
```

Array iteration

Array map()

Método que cria uma nova array executando uma função em cada elemento do array, não executa a função para elementos array sem valores. O método não altera a matriz original

Exemplo

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Este exemplo multiplica cada valor de matriz por 2:

Array iteration

Array filter()

Método que cria uma nova array com elementos de um array que passa por um teste

Exemplo

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Este exemplo cria uma nova matriz de elementos com um valor maior que 18:

Array iteration

Array reduce()

Método que executa uma função em cada elemento de uma matriz para produzir (reduzi-lo) a um único valor. No array o método funciona da esquerda para a direita, o método não reduz a matriz original

Exemplo

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Este exemplo encontra a soma de todos os números em uma matriz:

Date Objects

O Date object nos permite trabalhar com datar.

Por Padrão o JavaScript usará o suo horário do navegador e exibirá uma data como uma string de texto completo:

Qua 06 de julho de 2022 11:56:19 GMT-0300 (Horário Padrão de Brasília)

Date Objects

Criando Date Objects

Objetos de data são criados com o construtor new Date().

Existem 4 maneiras de criar um novo date objects:

1. new Date()
2. new Date(year, month, day, minutes, seconds, milliseconds)
3. new Date(milliseconds)
4. new Date(date string)

Date Objects

new Date()

cria um novo objeto de data com a data e hora atuais

Exemplo

```
const d = new Date();
```

Date Objects

new Date(year, month, …)

cria um novo objeto de data com data e hora especificada

7 números especificam ano, mês, dia, hora, minuto, segundo e milissegundo (nessa ordem)

Exemplo

```
const d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

O JavaScript conta meses de 0 a 11. Janeiro =0, Dezembro = 11

Date Objects

new Date(dateString)

cria um novo objeto de data a partir de uma string de data

Exemplo

```
const d = new Date("October 13, 2014 11:13:00");
```

Date Formats

Date Input

geralmente, existem 3 tipos de formatos de entrada de data JavaScript

Modelo	Exemplo
Data ISO	"2015-03-25" (O Padrão Internacional)
Encontro curto	"25/03/2015"
Data Longa	"25 de março de 2015" ou "25 de março de 2015"

Date Formats

Date Output

Independente do formato de entrada, o JavaScript irá (por padrão) produzir datas no formato de string de texto completo.

```
Wed Jul 06 2022 12:09:26 GMT-0300 (Horário Padrão de Brasília)
```

Date Formats

ISO Dates

A ISO 8601 é a forma internacional para a representação de datas e horas. A sintaxe ISO 8601 (AAAA-MM-DD) também é o formato de data JavaScrit mais usado.

Exemplo (data completa)

```
const d = new Date("2015-03-25");
```

Data ISO (Ano e Mês)

As datas ISO podem ser escritas sem especificar o dia (AAAA-MM)

Exemplo

```
const d = new Date("2015-03");
```

Get Date Methods

Esses métodos podem ser usados para obter informações de um objeto de Date:

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Set Date Methods

Os métodos Set Date são usados para definir uma parte de uma data:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

JSON

JSON é um formato para armazenar e transportar dados, é frequentemente usado quando os dados são enviados de um servidor para uma página da web

O que é JASON?

- Significa Java Script Object Notation
- Formato de intercâmbio de dados leve
- Independente de idioma
- “autodescritivo” e fácil de entender

A sintaxe JSON é derivada da sintaxe de notação de objeto JavaScript, mas o formato JSON é somente texto. O código para leitura e geração de dados JSON pode ser escrito em qualquer linguagem de programação.

JSON

Exemplo de JSON

```
{
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
}
```

Esta sintaxe JSON define um objeto de funcionários: uma matriz de 3 registros de funcionários (objetos):

JSON

O formato JSON avalia objetos JavaScript, é sintaticamente idêntico ao código para cria objetos JavaScript.

Devido a essa semelhança, um programa JavaScript pode facilmente converter dados JSON em objetos JS nativos.

Regras de sintaxe JSON

- Os dados estão em pares nome/valor
- Os dados são separados por vírgulas
- Chaves encaracoladas seguram objetos
- Os colchetes mantêm matrizes

JSON Data

Os dados JSON são escritos como pares nome/valor, assim como as propriedades do objeto JavaScript.

Um par nome/valor consiste em um nome de campo (entre aspas duplas), seguido por dois pontos, seguido por um valor:

```
"firstName":"John"
```

Os nomes JSON exigem aspas duplas. Os nomes JavaScript não.

JSON Objects

São escritos dentro de chaves. Assim como no JS, os objetos podem conter vários pares nome/valor:

```
{"firstName":"John", "lastName":"Doe"}
```

JSON Array

São escritas entre colchetes. Assim como em JS, um array pode conter objetos:

```
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
```

No exemplo acima, o objeto "employees" é um array. Ele contém três objetos.

Cada objeto é um registro de uma pessoa (com um nome e um sobrenome).

Convertendo um texto JSON em um objeto JavaScript

Um uso comum do JSON é ler dados de um servidor da web e exibir os dados em uma página da web. Para simplificar, isso pode ser demonstrado usando string como entrada. Primeiro, crie uma string JS contendo a sintaxe JSON:

```
let text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

Em seguida, use a função incorporada do JS JSON.parse() para converter a string em um objeto JS:

```
const obj = JSON.parse(text);
```

Convertendo um texto JSON em um objeto JavaScript

Use o novo objeto JavaScript em sua página:

Exemplo

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

Callbacks

É uma função passada como argumento para outra função.

Esta técnica permite que uma função chame outra função

Uma função de retorno de callback pode ser executada após a conclusão de outra função

Callbacks Function Sequence

Function Sequence

As funções JS são executadas na sequência em que são chamadas. Não na sequência em que são definidos

Sequence Control

Às vezes você gostaria de ter um melhor controle sobre quando executar uma função. SUponha que você queira fazer um cálculo e, em seguida, exibir o resultado

Você pode chamar uma função de calculadora (myCalculato), salvar o resultado e depois chamar outra função (myDisplayer) para exibir o resultado:

Chamadas Assíncronas

Funções executadas em paralelo com outras funções são chamadas de assíncronas, um bom exemplo é o JavaScript setTimeout()

```
function myDisplayer(something) {
  document.getElementById("demo").innerHTML = something;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

No exemplo acima, `myDisplayer`é o nome de uma função.

Ele é passado para `myCalculator()` como um argumento.

Chamadas Assíncronas

Esperando timeout

Ao usar a função JS setTimeout(). você pode especificar uma função de retorno de chamada a ser executada no tempo limite

Exemplo

```
setTimeout(myFunction, 3000);

function myFunction() {
  document.getElementById("demo").innerHTML = "I love You !!";
}
```

No exemplo, `myFunction` é usado como callback.

`myFunction` é passado `setTimeout()` como argumento.

3000 é o número de milissegundos antes do tempo limite, então `myFunction()` será chamado após 3 segundos.

Chamadas Assíncronas

Aguardado intervalos

Ao usar função JavaScript setInterval(), você pode especificar uma função de retorno de chamada a ser executada para cada intervalo:

No exemplo `myFunction` é usado como callback.

`myFunction` é passado `setInterval()` como argumento.

1000 é o número de milissegundos entre os intervalos, então `myFunction()` será chamado a cada segundo.

Exemplo

```
setInterval(myFunction, 1000);

function myFunction() {
  let d = new Date();
  document.getElementById("demo").innerHTML=
  d.getHours() + ":" +
  d.getMinutes() + ":" +
  d.getSeconds();
}
```

Chamadas Assíncronas

Aguardado arquivos

Se criar uma função para carregar um recurso externo (como um script ou um arquivo), não poderá usar o conteúdo antes que ele seja totalmente carregado, este é o momento perfeito para usar um retorno de chamada.

Chamadas Assíncronas

Este exemplo carrega um arquivo HTML (`mycar.html`) e exibe o arquivo HTML em uma página da Web, após o arquivo ser totalmente carregado:

Aguardando um arquivo:

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function getFile(myCallback) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.html");
  req.onload = function() {
    if (req.status == 200) {
      myCallback(this.responseText);
    } else {
      myCallback("Error: " + req.status);
    }
  }
  req.send();
}

getFile(myDisplayer);
```

No exemplo, `myDisplayer` é usado como callback.

`myDisplayer`é passado `getFile()` como argumento.

Promises

Um Promises é um objeto JavaScript que vincula a produção de código e o código de consumo.

Um objeto JS promise contém o código de produção e as chamadas para o código de consumo

Sintaxe da promessa

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Promises

Propriedades do objeto de promises

Um objeto JS promise pode ser:

- Peding
- Fulfilled
- Rejected

O objeto promise oferece suporte a duas propriedades: state e result

Enquanto um objeto está “peding” (funcionando), o resultado é indefinido

Quando um objeto Promise é ”Fulfilled”, o resultado é um valor.

Quando um objeto Promise é “Rejected”, o resultado é um objeto de erro

Promises

exemplo de como usar uma promises

```
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Promise.then() recebe dois argumentos, um callback para sucesso e outro para falha.

Ambos são opcionais, portanto, você pode adicionar um retorno de chamada apenas para sucesso ou falha.

Async/Await

A palavra-chave async antes de uma função faz com que a função retorna uma promises

Exemplo

```
async function myFunction() {
  return "Hello";
}
```

É o mesmo que:

```
function myFunction() {
  return Promise.resolve("Hello");
}
```

Aqui está como usar a Promessa:

```
myFunction().then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Async/Await

aguarde Sintaxe

A palavra chave await antes de uma função faz com que a função espere por uma promises

```
let value = await promise;
```

await só pode ser usada dentro de uma função async

Fetch

Use o fetch() para postar dados codificados em JSON

```
const data = { username: 'example' };

fetch('https://example.com/profile', {
  method: 'POST', // or 'PUT'
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

Fetch

Carregando um arquivo

Os arquivos podem ser carregados usando um <input type=”file” /> elemento de entrada HTML FormData() e arquivos fetch() .

```
const formData = new FormData();
const photos = document.querySelector('input[type="file"][multiple]');

formData.append('title', 'My Vegas Vacation');
for (let i = 0; i < photos.files.length; i++) {
  formData.append(`photos_${i}`, photos.files[i]);
}

fetch('https://example.com/posts', {
  method: 'POST',
  body: formData,
})
.then(response => response.json())
.then(result => {
  console.log('Success:', result);
})
.catch(error => {
  console.error('Error:', error);
});
```

Exercício 1 - Listando repositórios com a API do GitHub

https://docs.github.com/en/rest

Exercício 2 - Mostrando dados de clima

https://openweathermap.org/api

Exercício 3 - Aplicação TODO List

Api em construção

Referências

1. https://www.caelum.com.br/apostila/apostila-html-css-javascript.pdf
2. https://www.scriptbrasil.com.br/download/apostila/837/
3. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#uploading_json_data
4. https://www.w3schools.com/js
5.

PADO

Labs