

SEJAM BEM VINDAS {REACT BOOTCAMP}



WOMAKERS
CODE

Rede: Stone.Co

WIFI: Somos0Impossible

Recomendações para o curso

- [Documentação do React](#)
- [Javascript MDN](#)
- [Formik - Formulários em React](#)
- [Material UI - React UI Framework](#)
- [React Router](#)
- [Jest & Cypress](#)
- [Redux](#)
- [Dev.to](#)

Aula #1 - Javascript Avançado

- **Array Methods**
- **Object Methods**
- **Promises**
- **Fetch API**
- **Async await**

{ARRAY METHODS}



```
const pluckDeep = key => obj => key.split('.').reduce((accum, key) => accum[key], obj)

const compose = (...fns) => res => fns.reduce((accum, next) => next(accum), res)

const unfold = (f, seed) => {
  const go = (f, seed, acc) => {
    const res = f(seed)
    return res ? go(f, res[1], acc.concat([res[0]])) : acc
  }
  return go(f, seed, [])
}
```

Array Methods

Métodos que a linguagem disponibiliza para tratarmos arrays, listas, que são demarcadas com colchetes `[]`.

Exemplo

```
const compras = ['banana', 'leite', 'alface', 'limão', 'suco'];
```

Queremos adicionar novos itens na lista

```
const compras = ['banana', 'leite', 'alface', 'limão', 'suco'];
```

Possibilidades

```
const compras = ['banana', 'leite', 'alface', 'limão', 'suco'];
```

1 - [... compras, 'azeite', 'queijo'];

2 - compras.push('azeite', 'queijo');

3 - compras.concat('azeite', 'queijo');

Tips:

<https://www.freecodecamp.org/news/https-medium-com-gladchinda-hacks-for-creating-javascript-arrays-a1b80cb372b/>
<https://www.sitepoint.com/quick-tip-create-manipulate-arrays-in-javascript/>

Você precisa conhecer

`.concat()`

`.forEach()`

`.push()`

`.entries()`

`.includes()`

`.reduce()`

`.every()`

`.indexOf()`

`.slice()`

`.filter()`

`.join()`

`.splice()`

`.find()`

`.keys()`

`.sort()`

`.findIndex()`

`.map()`

`.values()`

`.flat()`

`.pop()`

Veja mais em https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

{OBJECT METHODS}



```
const obj = { 0: 'a', 1: 'b', 2: 'c' };  
Object.getOwnPropertyNames(obj).sort(); // ["0", "1", "2"]
```

Object Methods

Métodos que a linguagem disponibiliza para tratarmos objetos, suas propriedades e valores, identificado por chaves `{}`.

Exemplo

```
const aluna = {  
  nome: 'Jane Doe',  
  CPF: '00000000000',  
  endereço: 'rua das belezas',  
  matriculada: true  
};
```

Jane Doe se mudou de casa, como atualizamos?

```
const aluna = {  
  nome: 'Jane Doe',  
  CPF: '00000000000',  
  endereço: 'rua das belezas',  
  matriculada: true  
};
```

Possibilidades

- 1 - `aluna.endereco = 'rua das estrelas';`
- 2 - `aluna['endereco'] = 'rua das estrelas';`

Tips:

<https://www.digitalocean.com/community/tutorials/how-to-use-object-methods-in-javascript>
<https://towardsdatascience.com/everything-about-javascript-object-part-1-854025d71fea>
<https://www.digitalocean.com/community/tutorials/understanding-objects-in-javascript>

Resultado

```
const aluna = {  
  nome: 'Jane Doe',  
  CPF: '00000000000',  
  endereco: 'rua das estrelas',  
  matriculada: true  
};
```

Precisamos adicionar novas
informações sobre Jane.
Precisamos do email dela

```
const aluna = {  
  nome: 'Jane Doe',  
  CPF: '00000000000',  
  endereço: 'rua das estrelas',  
  matriculada: true  
};
```


Possibilidades

- 1 - `aluna.email = 'janedoe@mail.com';`
- 2 - `aluna['email'] = 'janedoe@mail.com';`
- 3 - `{ ...aluna, email: 'janedoe@mail.com' };`

Tips:

<https://www.digitalocean.com/community/tutorials/how-to-use-object-methods-in-javascript>

<https://towardsdatascience.com/everything-about-javascript-object-part-1-854025d71fea>

<https://www.digitalocean.com/community/tutorials/understanding-objects-in-javascript>

Assíncrono

Precisamos entender que o javascript é uma linguagem assíncrona, ou seja ela não vai esperar o retorno da sua primeira função. Ele vai continuar lendo o código enquanto a função x é processada.

Síncrono

Tudo tem uma ordem para ser processado, enquanto a função a é processada e depende de fatores externos para finalizar, a função b não será executada

Exemplos:

Assíncrono

Pintamos o cabelo, enquanto se passam 40 minutos, pintamos as unhas, assistimos uma série, hidratamos a pele.

A nossa vida não parou enquanto o cabelo era pintado.

Síncrono

Pintamos as unhas, não conseguimos fazer nada com as nossas mãos enquanto o esmalte não secar

Leia: <https://medium.com/@alcidesqueiroz/javascript-ass%C3%ADncrono-callbacks-promises-e-async-functions-9191b8272298>
<https://scotch.io/tutorials/javascript-promises-for-dummies>

Javascript Assíncrono



```
setTimeout(function() {  
    console.log('I am an asynchronous message');  
}, 1000);  
  
console.log('I am a synchronous message');
```

Código assíncrono sempre será executado depois que a thread principal estiver desocupada



```
setTimeout(function() {  
    console.log('I am an asynchronous message');  
}); // You can omit the 0  
  
console.log('Test 1');  
  
for (let i = 0; i < 10000; ++i) {  
    doSomeStuff();  
}  
  
console.log('Test 2');  
  
// The 'I am an asynchronous message' will be displayed when the main thread reach here  
  
function doSomeStuff() {  
    return 1 + 1;  
}
```

Casos assíncronos do javascript

- **HTTP requests**
- **Operações I/O com node**
- **WebSockets**

Callback

Callback é uma função passada como parâmetro para uma outra função.




```
setTimeout(function(){  
    alert("Hello");  
}, 3000);
```

Leia: <https://medium.com/@alcidesqueiroz/javascript-ass%C3%ADncrono-callbacks-promises-e-async-functions-9191b8272298>
<https://scotch.io/tutorials/javascript-promises-for-dummies>

Callbacks

Callback recebe um argumento e na execução cada callback deve saber o seu próximo callback, confuso não?



```
function job1(callback) {
  setTimeout(function() {
    callback('test 1');
  }, 2000);
}

function job2(callback) {
  setTimeout(function() {
    callback('test 2');
  }, 4000);
}

job1(function(data) {
  console.log(data);

  job2(function(data) {
    console.log(data);
  });
});
```


Callback Hell

```
1 function hell(win) {  
2   // for listener purpose  
3   return function() {  
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {  
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {  
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {  
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {  
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {  
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {  
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {  
11                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {  
12                  loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {  
13                   async.eachSeries(SRIPTS, function(src, callback) {  
14                     loadScript(win, BASE_URL+src, callback);  
15                   });  
16                 });  
17               });  
18             });  
19           });  
20         });  
21       });  
22     });  
23   });  
24 };  
25 }  
26 }
```



{PROMISES}



```
const myFirstPromise = new Promise((resolve, reject) => {  
  resolve(someValue);  
  reject("failure reason");  
});
```

Promise

é um objeto usado para
processamento assíncrono. Um
Promise (de "promessa") representa
um valor que pode estar disponível
agora, no futuro ou nunca. new

Promise() - Doc MDN

Leia: <https://medium.com/@alcidesqueiroz/javascript-ass%C3%ADncrono-callbacks-promises-e-async-functions-9191b8272298>
<https://scotch.io/tutorials/javascript-promises-for-dummies>

Criando uma promise



```
const promise = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve('data')  
  }, 2000)  
})  
  
promise  
  .then(  
    data => console.log(data)  
    err => console.error(err)  
  )
```

resolve, reject, .then() e .catch()

Resolve e Reject são funções que vão "cumprir" a sua promise com um resultado ou não.

Resolve é para sucesso

Reject é quando deu ruim :(

.then() é uma função da promise que no seu primeiro parâmetro de callback recebe o resultado do resolve, .catch() é quando cai no reject

Múltiplos callbacks



```
var promise = new Promise(function(resolve, reject) {  
  setTimeout(function() {  
    resolve('hello world');  
  }, 2000);  
});  
  
promise.then(function(data) {  
  console.log(data + ' 1');  
});  
  
promise.then(function(data) {  
  console.log(data + ' 2');  
});  
  
promise.then(function(data) {  
  console.log(data + ' 3');  
});
```

Tricks

Você pode invocar `resolve` e `reject` quantas vezes quiser dentro de uma `promise`, mas uma vez que a `promise` é finalizada não poderá ser processada de novo



```
var promise = new Promise(function(resolve, reject) {  
  setTimeout(function() {  
    resolve('hello world 1');  
    resolve('hello world 2');  
    resolve('hello world 3');  
    resolve('hello world 4');  
  }, 1000);  
});  
  
promise.then(function success(data) {  
  console.log(data);  
});  
  
//RESULT: hello world 1
```

Encadeamento de Promises (chaining)

Chaining é a razão pela qual as promises existem. É uma maneira de informar ao javascript a próxima coisa a fazer depois que a execução de uma função terminar.

O Resultado de then é sempre uma promise

```
var promise = job1();

promise

.then(function(data1) {
  console.log('data1', data1);
  return job2();
})

.then(function(data2) {
  console.log('data2', data2);
  return 'Hello world';
})

.then(function(data3) {
  console.log('data3', data3);
});

function job1() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      resolve('result of job 1');
    }, 1000);
  });
}


function job2() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      resolve('result of job 2');
    }, 1000);
  });
}
```


Mesmo assim cuidado...



```
function test() {  
  return job().then(function() {  
    return job2().then(function() {  
      return job3().then(function() {  
        return job4().then(function() {  
          doSomething();  
        });  
      });  
    });  
  });  
}
```

Chaining



```
function test() {  
  return job()  
  
  .then(function() {  
    return job2();  
  })  
  
  .then(function() {  
    return job3();  
  })  
  
  .then(function() {  
    return job4();  
  })  
  
  .then(function() {  
    doSomething();  
  });  
}
```

{ASYNC/AWAIT}

```
function func1(number){
  return new Promise(resolve =>{
    setTimeout(() => resolve(77 + number) , 1000)
  })
}

function func2(number){
  return new Promise(resolve =>{
    setTimeout(() => resolve(22 + number) , 1000)
  })
}

function func3(number){
  return new Promise(resolve =>{
    setTimeout(() => resolve(14 * number) , 1000)
  })
}

// Usando as funções
func1(4)
  .then(number => {
    console.log('Hello');
    return func2(number);
  })
  .then(number => {
    console.log('World');
    return func3(number);
  })
  .then(result => console.log(result))
```

VS

```
function func1(number){
  return new Promise(resolve =>{
    setTimeout(() => resolve(77 + number) , 1000)
  })
}

function func2(number){
  return new Promise(resolve =>{
    setTimeout(() => resolve(22 + number) , 1000)
  })
}

function func3(number){
  return new Promise(resolve =>{
    setTimeout(() => resolve(14 * number) , 1000)
  })
}

// Usando as funções
async function myAsyncFunction(){
  var number = await func1(4);

  console.log('Hello');
  number = await func2(number);

  console.log('World');
  var result = await func3(number);

  console.log(result);
}
```

Com `async await` nós podemos trabalhar com funções assíncronas de uma maneira mais simples.

Funções assíncronas sempre retornam promises



```
const fetch = require('node-fetch');

async function getPerson(id) {
  const response = await fetch(`http://swapi.co/api/people/${id}`);
  const person = await response.json();
  return person;
}

getPerson(1)
  .then(person => console.log(person.name));
```

await só pode ser usado
dentro do escopo de uma
função assíncrona



```
const fetch = require('node-fetch');

async function getPerson(id) {
  const response = await fetch(`http://swapi.co/api/people/${id}`);
  const person = await response.json();
  return person;
}

getPerson(1)
  .then(person => console.log(person.name));
```

Errors

Como sabemos uma promise pode ser rejeitada...

try catch vão te ajudar a resolver esse problema

```
function job() {  
  return new Promise(function(resolve, reject) {  
    setTimeout(reject, 500, 'Error happened');  
  });  
}  
  
async function test() {  
  try {  
    let message = await job();  
    console.log(message);  
  
    return 'Hello world';  
  } catch (error) {  
    console.error(error);  
  
    return 'Error happened during test';  
  }  
}  
  
test().then(function(message) {  
  console.log(message);  
});
```


throw

Para rejeitar uma
promise em uma
async function
basta dar um **throw**



```
async function job() {  
    throw new Error("Access denied");  
}  
  
job()  
  
    .then(function(message) {  
        console.log(message);  
    })  
  
    .catch(function(error) {  
        console.log(error);  
    });
```

{FETCH API}



```
fetch('https://example.com/todos')  
  .then(response => response.json())  
  .then(data => console.log(JSON.stringify(data)))
```

Fetch API

é uma API Javascript baseada em promises para fazer chamadas HTTP assíncronas no navegador similar ao XMLHttpRequest.

Leia: <https://dev.to/attacomsian/introduction-to-javascript-fetch-api-4f4c>



```
const user = {
  first_name: 'John',
  last_name: 'Lilly',
  job_title: 'Software Engineer'
};

const options = {
  method: 'POST',
  body: JSON.stringify(user),
  headers: {
    'Content-Type': 'application/json'
  }
}

fetch('https://reqres.in/api/users', options)
  .then(res => res.json())
  .then(res => console.log(res));
```

{TERMINAL}

Não tenha medo do terminal

Aprenda alguns
básicos que farão
parte do seu dia a
dia como
desenvolvedora



```
> sudo don't be afraid
```

Lista de comandos básicos

- `ls` - listar arquivos
- `cd` - navegar entre pastas
- `mkdir` - criar diretório
- `cp` - copiar e colar arquivos
- `rm` - remover arquivos, adicionando `-r` remove diretórios
- `grep` - encontrar um texto em um arquivo
- `cat` - ver conteúdo do arquivo
- `touch` - criar arquivo
- `pwd` - diretório atual

- **NPM - Node Package Manager**
- **Yarn - package manager**

{GIT & GITHUB}



O que é Git?

Git é um sistema de controle de versão para desenvolvimento de projetos onde vários programadores podem contribuir ao mesmo tempo no código, editando e criando novos arquivos sem o risco das alterações serem perdidas ou sobrescritas.

O que é Github?

O Github é uma plataforma web para hospedagem de projetos pessoais e profissionais, além de disponibilizar novos recursos ao Git, como gestão de issues e gerenciamento de usuários.

<https://tekzoom.com.br/descubra-como-trabalhar-com-git-e-github-tutorial-2019/>

{CHALLENGES}