Callbacks

Quando não utilizamos callback em nossas funções com setTimeout, por exemplo o código não espera que queremos esperar ele contar alguns milésimos para entregar nosso resultado, por exemplo:

```
function getUser(){
    const user = {
            id: 323,
            name: 'Kaleb Crayel',
            birthOfDate: new Date()
        }
    setTimeout(() => {
        return user;
    }, 2000)
}

const user = getUser()
console.log(user)
```

Isso vai gerar um erro dizendo que não é possivel ler user porque ele não está definido, pois o setTimeout ainda não tinha executado por completo, para isso vamos utilizar o conceito de callbacks.

A callback é uma maneira de se passar uma função como parâmetro para executar assim que o tempo de um setTimeout acabar (por exemplo).

Contextualização: Temos um banco de dados com informações de usuário e precisamos resgatar essas informações para exibi-lás em algum lugar.

- Obter usuário.
- Obter endereço do usuário.

```
function getUser(callback){
    setTimeout(() => {
      return callback(null, {
        id: 323,
      name: 'Gabriel',
      birthOfDate: new Date()
    })
  }, 2000)
}

function getAddressUserById(id, callback){
    setTimeout(() => {
      return callback(null, {
        street: 'Street 32',
      number: 123
    })
  })
}

function resolveUser(error, user){
```

```
    console.log('user', user)
}

getUser(function resolveUser(error, user){
    if(error){
      console.log('error in user', error)
    return;
  }
  getAddressUserById(user.id, function resolveAddress(errorAddress, address){
      if(errorAddress){
        console.log('error in address', errorAddress)
      return;
    }
    console.log(`${user.name}`)
    console.log(`${address.street}`)
  })
})
```

Grande problemas de callback múltiplas conhecida como Callback hell, é ficar encadeando callbacks dentro de callbacks

```
funcOne(){
    funcTwo(){
        funcThree(){

        }
    }
}
```

Para isso foi implementado uma maneira mais eficiente e legível chamada de Promises.

Promises

Promises vieram para salvar a utilização de diversos callbacks em suas aplicações. No mundo real, quando fazemos uma requisição à dados no DB, não sabemos quanto tempo levará para recebermos os dados de volta para "cuspir" json para o front end em uma API REST, por exemplo. Por mais que sua aplicação seja muito performática e com ótima conexão, saber exatamente o tempo que irá terminar de executar e receber o resultado é um tanto quanto hard.

Com isso utilizamos o sistema de Promises. Exemplo: Precisamos obter um usuário do nosso banco de dados e o seu endereço.

```

    // RESOLVE: Quando ocorrer tudo corretamente.
    // REJECT: Quando algo falhar.

    function getUser(){
    return new Promise(function resolvePromise(resolve, reject){
        setTimeout(() => {
              return resolve({
                    id: 323,
                name: 'Gabriel',
```

```
                birthOfDate: new Date()
            })
          }, 2000)
    })
}

function getAddressUserById(id){
    return new Promise(function resolveAddress(resolve, reject){
        setTimeout(() => {
              return resolve({
                street: 'Street 32',
                number: 123
            })
        }, 2000)
    })
}

const user = getUser()
user
    .then(user => {
        let { id } = user;
        return getAddressUserById(id)
            .then(address => {
              console.log(user, address)
          })
    })
    .catch(error => {console.log(error)})
```