



COM222

Desenvolvimento de

Sistemas na Web

Aula 09 – Callbacks e Promises



Códigos da Aula

- Precisa ter *nodejs* instalado
- Baixe: <https://github.com/Guilheeeeeerme/CallbacksJavascript>
- Entrar na pasta *CallbacksJavascript*
- Execute *npm install* (somente a primeira vez)
- Execute *npm start*
- Vai aparecer

```
C:\Users\Guilherme\Desktop\CallbacksJavascript>npm start

> Aulas@1.0.0 start C:\Users\Guilherme\Desktop\CallbacksJavascript
> http-server .

Starting up http-server, serving .
Available on:
  http://192.168.0.163:8080
  http://127.0.0.1:8080
  http://192.168.9.97:8080
Hit CTRL-C to stop the server
> []
```



Códigos da Aula

- Acesse no browser uma das urls
 - Exemplo: <http://127.0.0.1:8080>

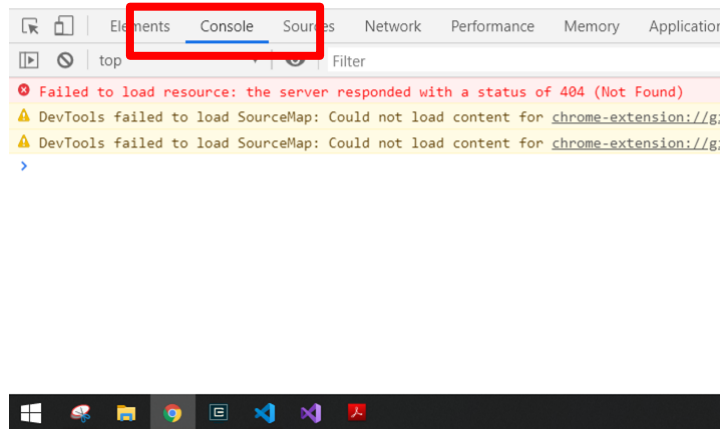
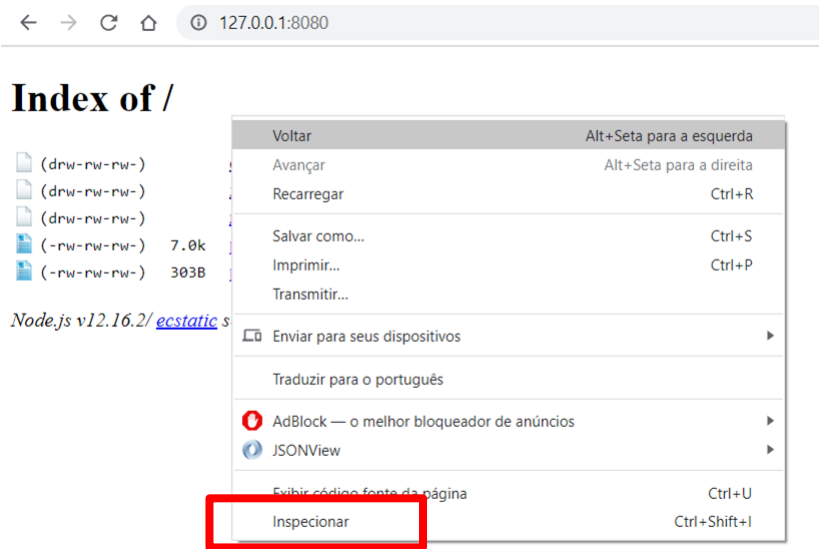
```
C:\Users\Guilherme\Desktop\CallbacksJavascript>npm start

> Aulas@1.0.0 start C:\Users\Guilherme\Desktop\CallbacksJavascript
> http-server .

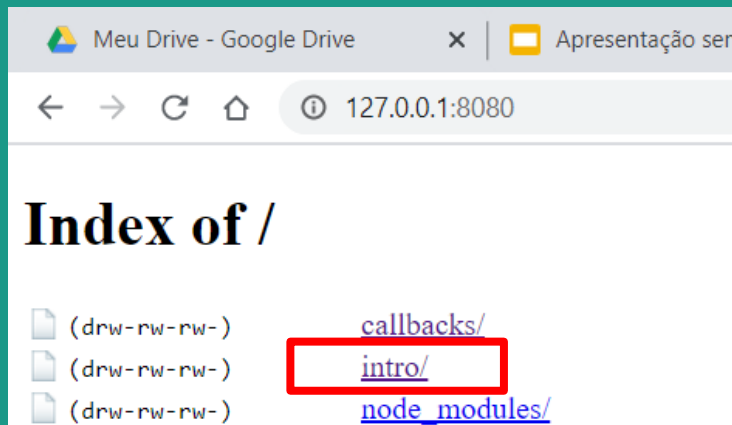
Starting up http-server, serving .
Available on:
  http://192.168.0.163:8080
  http://127.0.0.1:8080
  http://192.168.9.97:8080
Hit CTRL-C to stop the server
> 
```

Códigos da Aula

- Abra o console do browser



Introdução





Introdução

- O javascript tem tipagem dinâmica, isso significa que uma variável pode assumir qualquer valor
- Exemplo:

```
5  var umTexto = "Olá, sou um texto"
6  var umNumero = 10
7  var umaLista = [1, 2, 3]
8
9  console.log(umTexto);
10 // output: "Olá sou um texto"
11
12 console.log(umNumero);
13 // output: 10
14
15 console.log(umaLista);
16 // output: [1, 2, 3]
17 console.log(umaLista[0]);
18 // output: 1
```



Introdução

- Inclusive funções

```
var umaFuncaoQueDizOi = function(){  
  |   console.log("Sou a função que diz Oi !!!")  
}  
  
umaFuncaoQueDizOi();  
// output: Sou a função que diz Oi !!!|
```

```
var umTexto = "Olá, sou um texto"
var umNumero = 10
var umaLista = [1, 2, 3]

console.log(umTexto);
// output: "Olá sou um texto"

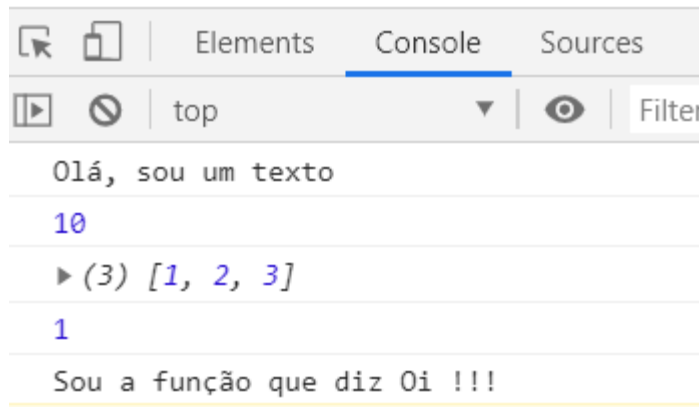
console.log(umNumero);
// output: 10

console.log(umaLista);
// output: [1, 2, 3]
console.log(umaLista[0]);
// output: 1

var umaFuncaoQueDizOi = function(){
  console.log("Sou a função que diz Oi !!!")
}

umaFuncaoQueDizOi();
// output: Sou a função que diz Oi !!!
```

Exemplo de resultado no console





Introdução

- Assim como qualquer função em qualquer linguagem de programação é possível passar parâmetros para as funções

```
var umaFuncaoQueFazSoma = function (a, b) {  
    return a + b;  
}  
  
console.log(umaFuncaoQueFazSoma(68, 55));  
// Output: 123 (68 + 55)|
```



Introdução

- Se uma variável pode assumir qualquer tipo
- Se uma variável pode ser inclusive uma função
- Se os parâmetros de uma função são variáveis
- Então...

*É possível passar **funções como parâmetro** para uma função !!!*




Oi?

```
var funcaoA = function(callback) {  
    console.log("Estou usando a função A");  
    callback();  
}  
  
var funcaoB = function() {  
    console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```



Oi?

Criei uma função
chamada *funcaoA*



```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```



Oi?

Criei uma função
chamada *funcaoA*

Essa função tem um
parâmetro chamado
callback

```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```



Oi?

Criei uma função
chamada *funcaoA*

Essa função tem um
parâmetro chamado
callback

```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```

Aqui eu executo
callback



Oi?

Criei uma função
chamada *funcaoA*

Criei uma função
chamada *funcaoB*

Essa função tem um
parâmetro chamado
callback

```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```

Aqui eu executo
callback



Oi?

Criei uma função chamada *funcaoA*

Criei uma função chamada *funcaoB*

Essa função não recebe parâmetros e só printa uma mensagem

Essa função tem um parâmetro chamado callback

```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```

Aqui eu executo *callback*



Oi?

Criei uma função chamada ***funcaoA***

Criei uma função chamada ***funcaoB***

Essa função não recebe parâmetros e só printa uma mensagem

Essa função tem um parâmetro chamado **callback**

Aqui eu executo ***callback***

Aqui eu chamo a **funcaoA** passando a **funcaoB** como parâmetro

```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```

Oi?

Criei uma função chamada **funcaoA**

Criei uma função chamada **funcaoB**

Essa função não recebe parâmetros e só printa uma mensagem

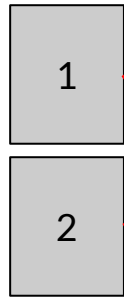
```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```

Essa função tem um parâmetro chamado **callback**

Aqui eu executo **callback**

callback vai receber **funcaoB**, então chamar **callback()** é equivalente a chamar **funcaoB()**

Sequência



```
var funcaoA = function(callback) {  
  console.log("Estou usando a função A");  
  callback();  
}  
  
var funcaoB = function() {  
  console.log("Estou usando a função B");  
}  
  
funcaoA(funcaoB);  
// Estou usando a função A  
// Estou usando a função B
```



Callbacks

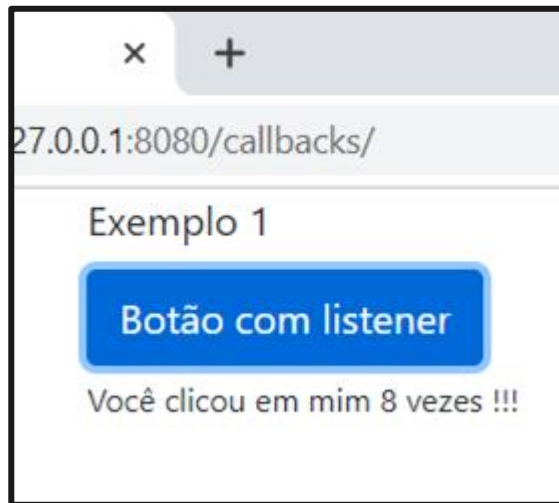
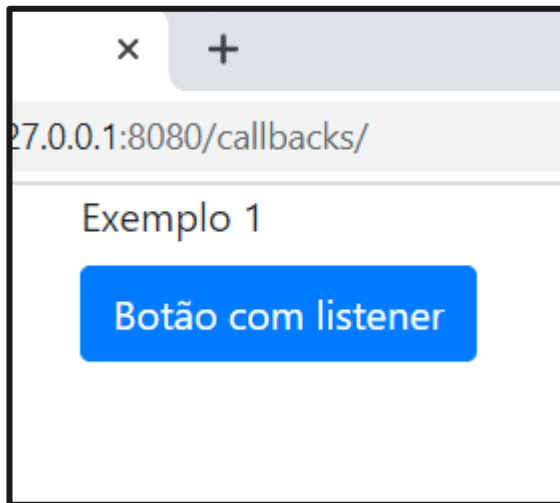


Callbacks

- Passar funções como parâmetro é o que chamamos de *callback*
- Esse recurso é amplamente utilizado quando fazemos procedimentos assíncronos
- Quando passamos uma função como parâmetro é o equivalente a dizer “*chame essa função para mim quando terminar de fazer algo*” ou “*chame essa função quando algo acontecer*”

Exemplos (/callback)

- O evento de click é um exemplo de callback





Código

index.html

```
<div class="col-md-3">
  <label>Exemplo 1</label><br/>
  <button id="click-callback" class="btn btn-primary">
    Botão com listener
  </button><br/>
  <small id="click-callback-result"></small>
</div>
```

callback.js

```
var QuandoClicarEmMinContador = 0;

function QuandoClicarEmMim(){
  console.log("Você clicou em mim !!!");
  QuandoClicarEmMinContador++;
  $("#click-callback-result").text("Você clicou em mim " + QuandoClicarEmMinContador + " vezes !!!");
}

$("#click-callback").click(QuandoClicarEmMim)
```

Código

index.html

```
<div class="col-md-3">
  <label>Exemplo 1</label><br/>
  <button id="click-callback" class="btn btn-primary">
    Botão com listener
  </button><br/>
  <small id="click-callback-result"></small>
</div>
```

callback.js

QUANDO OCORRER um click no
element *click-callback* execute a
função *QuandoClicarEmMin*

```
var QuandoClicarEmMinContador = 0;

function QuandoClicarEmMin(){
  console.log("Você clicou em mim !!!");
  QuandoClicarEmMinContador++;
  $("#click-callback-result").text("Você clicou em mim " + QuandoClicarEmMinContador + " vezes !!!");
}

$("#click-callback").click(QuandoClicarEmMin)
```




Eventos assíncronos

- Em uma página web é muito comum que se faça requisições para APIs
- Exemplo:
 - Vamos fazer uma página que busca o CEP e mostra o resultado na tela
 - viacep.com.br/ws/01001000/json/
- Requisições HTTP são eventos assíncronos
- O navegador cria uma thread para ir buscar o resultado da requisição
- Em algum momento esse resultado chega e nós precisamos de um callback para obtê-lo
- Vamos usar função `$.get('link_do_recurso', callback)`

index.html

```
<div class="col-md-3">
  <label>Exemplo 2</label><br/>
  <input type="text" id="click-callback-cep-input" class="form-control">
  <button id="click-callback-cep" class="btn btn-primary">
    |   Buscar CEP
  </button><br/>
  <small id="click-callback-cep-result"></small>
</div>
```

```
$("#click-callback-cep").click(BuscarCep)
```

Colocando a função
BuscarCep como
callback

Exemplo 2

37500182

Buscar CEP

Rua Prefeito Tigre Maia

```
function BuscarCep(){

    // estetica
    console.log("-----");

    // tirando o resultado anterior
    Logradouro = null;
    $("#click-callback-cep-result").text(Logradouro);

    // Lendo o que o usuario preencheu no input
    var cep = $("#click-callback-cep-input").val();

    // marcando a hora que chamamos a API
    console.log("antes ", new Date());

    // indo buscar os dados do CEP e chamando a função
    $.get("https://viacep.com.br/ws/" + cep + "/json/", QuandoVoltarComOCep);

    // imediatamente após a chamada
    console.log("depois ", new Date());

    // ainda deve ser null
    console.log("Logradouro ", Logradouro);
}
```

```
var Logradouro;

function QuandoVoltarComOCep (info) {

    // preenchendo nossa variavel
    Logradouro = info.logradouro;

    // marcando a hora da
    console.log("chegou !!! ", new Date());

    $("#click-callback-cep-result").text(Logradouro);
}
```

\$.get vai buscar as
informações do CEP digitado
e chamar a função
QuandoVoltarComOCep

```
function BuscarCep(){  
  
    // estetica  
    console.log("-----");  
  
    // tirando o resultado anterior  
    Logradouro = null;  
    $("#click-callback-cep-result").text(Logradouro);  
  
    // Lendo o que o usuario preencheu no input  
    var cep = $("#click-callback-cep-input").val();  
  
    // marcando a hora que chamamos a API  
    console.log("antes ", new Date());  
  
    // indo buscar os dados do CEP e chamando a função  
    $.get("https://viacep.com.br/ws/" + cep + "/json/", QuandoVoltarComOCep);  
  
    // imediatamente após a chamada  
    console.log("depois ", new Date());  
  
    // ainda deve ser null  
    console.log("Logradouro ", Logradouro);  
}
```

```
var Logradouro;  
  
function QuandoVoltarComOCep (info) {  
  
    // preenchendo nossa variavel  
    Logradouro = info.logradouro;  
  
    // marcando a hora da  
    console.log("chegou !!! ", new Date());  
  
    $("#click-callback-cep-result").text(Logradouro);  
}
```

Logradouro possivelmente será null, pois a execução segue normalmente após a linha do **\$.get**

```
function BuscarCep(){  
  
    // estetica  
    console.log("-----");  
  
    // tirando o resultado anterior  
    Logradouro = null;  
    $("#click-callback-cep-result").text(Logradouro);  
  
    // Lendo o que o usuario preencheu no input  
    var cep = $("#click-callback-cep-input").val();  
  
    // marcando a hora que chamamos a API  
    console.log("antes ", new Date());  
  
    // indo buscar os dados do CEP e chamando a função  
    $.get("https://viacep.com.br/ws/" + cep + "/json/", QuandoVoltarComOCep);  
  
    // imediatamente após a chamada  
    console.log("depois ", new Date());  
  
    // ainda deve ser null  
    console.log("Logradouro ", Logradouro);  
}
```

```
var Logradouro;  
  
function QuandoVoltarComOCep (info) {  
  
    // preenchendo nossa variavel  
    Logradouro = info.logradouro;  
  
    // marcando a hora da  
    console.log("chegou !!! ", new Date());  
  
    $("#click-callback-cep-result").text(Logradouro);  
}
```

Logradouro será preenchido quando o **\$.get** terminar e chamar **QuandoVoltarComOCep**

E execução segue normalmente

Você pode notar nos console.logs

```
function BuscarCep(){  
    // estetica  
    console.log("-----")  
  
    // tirando o resultado anterior  
    Logradouro = null;  
    $("#click-callback-cep-result").text(Logradouro);  
  
    // Lendo o que o usuario preencheu no input  
    var cep = $("#click-callback-cep-input").val();  
  
    // marcando a hora que chamamos a API  
    console.log("antes ", new Date());  
  
    // indo buscar os dados do CEP e chamando a função  
    $.get("https://viacep.com.br/ws/" + cep + "/json/", QuandoVoltarComOCep);  
  
    // imediatamente após a chamada  
    console.log("depois ", new Date());  
  
    // ainda deve ser null  
    console.log("Logradouro ", Logradouro);  
}  
  
var Logradouro;  
  
function QuandoVoltarComOCep (info) {  
    // preenchendo nossa variavel  
    Logradouro = info.logradouro;  
  
    // marcando a hora da  
    console.log("chegou !!! ", new Date());  
  
    $("#click-callback-cep-result").text(Logradouro);  
}
```

```
function BuscarCep() {
```

```
  // estetica
```

```
  console.log("-----");
```

```
  // tirando o resultado anterior
```

```
  Logradouro = null;
```

```
  $("#click-callback-cep-result").text(Logradouro);
```

```
  // lendo o que o usuario preencheu no input
```

```
  var cep = $("#click-callback-cep-input").val();
```

```
  // marcando a hora que chamamos a API
```

```
  console.log("antes ", new Date());
```

```
  // indo buscar os dados do CEP e chamando a função
```

```
  $.get("https://viacep.com.br/ws/" + cep + ".json/", QuandoVoltarComOCep);
```

```
  // imediatamente após a chamada
```

```
  console.log("depois ", new Date());
```

```
  // ainda deve ser null
```

```
  console.log("Logradouro ", Logradouro);
```

```
var Logradouro;
```

```
function QuandoVoltarComOCep (info) {
```

```
  // preenchendo nossa variavel
```

```
  Logradouro = info.logradouro;
```

```
  // marcando a hora da
```

```
  console.log("chegou !!! ", new Date());
```

```
  $("#click-callback-cep-result").text(Logradouro);
```

```
}
```

antes Tue Apr 21 2020 21:49:50 GMT-0300 (Horário Padrão de Brasília)

depois Tue Apr 21 2020 21:49:50 GMT-0300 (Horário Padrão de Brasília)

Logradouro null

chegou !!! Tue Apr 21 2020 21:49:50 GMT-0300 (Horário Padrão de Brasília)



Callback

- No exemplo do click com busca de CEP nós temos dois callbacks
 - Click no botão tem um callback *BuscarCep*
 - Chamada de API com \$.get tem um callback *QuandoVoltarComOCep*

```
$("#click-callback-cep").click(BuscarCep)
```

```
// indo buscar os dados do CEP e chamando a função  
$.get("https://viacep.com.br/ws/" + cep + "/json/", QuandoVoltarComOCep);
```




Atenção

- É muito comum encontrar esse tipo de código onde a função é declarada diretamente como callback
- Isso se chama *Função Anônima*. Facilita a escrita, mas dificulta a manutenção de código.
- Essa prática é proibida em muitas empresas

```
$("#click-callback-cep").click(function () {  
  console.log("função anonima");  
})
```



Conclusão

- Por fim, o callback é um meio de “não ficar esperando” pois a execução segue normalmente
- Foi um mecanismo muito utilizado para sincronizar eventos
- É importante para não travar a tela enquanto aguarda uma resposta

Promises



Promises

- Com o passar do tempo o Javascript foi avançando e naturalmente muitas bibliotecas surgiram
- Principalmente por causa do Angular e do Node a comunidade sentiu a necessidade de padronizar a maneira como se passava os callbacks
- Era muito comum que as funções das bibliotecas deixassem os dois últimos parâmetros da função como *successCallback* e *errorCallback*, ou seja, um callback se a função der certo e uma se deu errado
- Ex:
 - `$.get(url, successCallback, errorCallback);`



Problema

*O javascript é muito flexível
para passagem de parâmetros,
assim, poderíamos chamar...*

*FazDivisao(1,2)
FazDivisao(1,2,null)
FazDivisao(1,2,3,4)*

*Essa flexibilidade abre um
leque imenso de problemas. Por
exemplo, se passarmos a string
“Farofa” como successCallback
nós teremos problemas*

```
function FazDivisao(a, b, successCallback, errorCallback) {  
  try {  
    if(b !== 0) {  
      if(successCallback !== null) {  
        successCallback(a / b);  
      }  
    } else {  
      if(errorCallback !== null) {  
        errorCallback("Não pode dividir por zero");  
      }  
    }  
  } catch (e) {  
    if(errorCallback !== null) {  
      errorCallback(e);  
    }  
  }  
}
```



Problema

Perceba que para todas as chamadas de `successCallback` ou `errorCallback` nós temos que verificar se o parâmetro foi enviado e mesmo assim temos o risco do parâmetro não ser uma função

Em alguns casos também o callback é passado adiante e o pode-se perder o controle - Múltiplos callbacks encadeados (Callback hell)

```
function FazDivisao(a, b, successCallback, errorCallback) {  
  try {  
    if(b !== 0) {  
      if(successCallback !== null) {  
        successCallback(a / b);  
      }  
    } else {  
      if(errorCallback !== null) {  
        errorCallback("Não pode dividir por zero");  
      }  
    }  
  } catch (e) {  
    if(errorCallback !== null) {  
      errorCallback(e);  
    }  
  }  
}
```



Promises

*A mesma função escrita com
promises fica assim*

```
function FazDivisao(a, b) {  
  return new Promise(function(resolve, reject) {  
    try {  
      if(b !== 0) {  
        resolve(a / b);  
      } else {  
        reject("Não pode dividir por zero");  
      }  
    } catch (e) {  
      reject(e);  
    }  
  });  
}
```



Promises

E o resultado se obtém assim

```
function ErroDivisao(erro) {  
  console.log("Deu errado !!!", erro);  
}  
  
function SucessoDivisao(resultado) {  
  console.log("Deu certo !!!", resultado);  
}  
  
FazDivisao(1,2)  
  .then(SucessoDivisao)  
  .catch(ErroDivisao)
```


Promise

- Uma promise se cria com `new Promise(function(resolve, reject) { })`
- A promise tem um método `resolve` que chama o método `.then` em caso de sucesso
- A promise tem um método `reject` que chama o método `.catch` em caso de erro

```
function FazDivisao(a, b) {  
  return new Promise(function(resolve, reject) {  
    try {  
      if(b !== 0) {  
        resolve(a / b);  
      } else {  
        reject("Não pode dividir por zero");  
      }  
    } catch (e) {  
      reject(e);  
    }  
  });  
}
```

```
function ErroDivisao(erro) {  
  console.log("Deu errado !!!", erro);  
}  
  
function SucessoDivisao(resultado) {  
  console.log("Deu certo !!!", resultado);  
}  
  
FazDivisao(1,2)  
  .then(SucessoDivisao)  
  .catch(ErroDivisao)
```



CEP com Promise

CEP com Promise

```
<div class="col-md-4">
  <label>Exemplo 3 Promise</label><br/>
  <input type="text" id="click-callback-promise-cep-input" class="form-control">
  <button id="click-callback-promise-cep" class="btn btn-primary">
    |   Buscar CEP
  </button><br/>
  <small id="click-callback-promise-cep-result"></small>
</div>
```

Exemplo 3 Promise

Buscar CEP

Rua Prefeito Tigre Maia

```
$("#click-callback-promise-cep").click([BuscarCepPromise])
```

Colocando a função
BuscarCepPromise
como callback

```
function BuscarCepPromise(){

    // estetica
    console.log("-----");

    // tirando o resultado anterior
    LogradouroPromise = null;
    $("#click-callback-promise-cep-result").text(LogradouroPromise);

    // Lendo o que o usuario preencheu no input
    var cep = $("#click-callback-promise-cep-input").val();

    // marcando a hora que chamamos a API
    console.log("antes ", new Date());

    // indo buscar os dados do CEP e chamando a função
    GetCep(cep)
        .then(QuandoVoltarComOCepPromise);

    // imediatamente após a chamada
    console.log("depois ", new Date());

    // ainda deve ser null
    console.log("LogradouroPromise ", LogradouroPromise);
}
```

```
function GetCep(cep) {
    return new Promise(function (resolve, reject) {
        $.getJSON("https://viacep.com.br/ws/" + cep + "/json/", resolve, reject);
    });
}
```

```
var LogradouroPromise;

function QuandoVoltarComOCepPromise (info) {

    // preenchendo nossa variavel
    LogradouroPromise = info.logradouro;

    // marcando a hora da
    console.log("chegou !!! ", new Date());

    $("#click-callback-promise-cep-result").text(LogradouroPromise);
}
```

GetCep vai buscar as informações do CEP digitado e chamar a função **QuandoVoltarComOCepPromise**

```
function BuscarCepPromise(){

    // estetica
    console.log("-----");

    // tirando o resultado anterior
    LogradouroPromise = null;
    $("#click-callback-promise-cep-result").text(LogradouroPromise);

    // Lendo o que o usuario preencher no input
    var cep = $("#click-callback-promise-cep-input").val();

    // marcando a hora que chamamos a API
    console.log("antes ", new Date());

    // indo buscar os dados do CEP e chamando a função
    GetCep(cep)
        .then(QuandoVoltarComOCepPromise);

    // imediatamente após a chamada
    console.log("depois ", new Date());

    // ainda deve ser null
    console.log("LogradouroPromise ", LogradouroPromise);
}
```

```
function GetCep(cep) {
    return new Promise(function (resolve, reject) {
        $.getJSON("https://viacep.com.br/ws/" + cep + "/json/", resolve, reject);
    });
}
```

```
var LogradouroPromise;

function QuandoVoltarComOCepPromise (info) {

    // preenchendo nossa variavel
    LogradouroPromise = info.logradouro;

    // marcando a hora da
    console.log("chegou !!! ", new Date());

    $("#click-callback-promise-cep-result").text(LogradouroPromise);
}
```

**Quando GetCep chamar 'resolve'
o .then vai ser disparado**

```
function BuscarCepPromise(){  
  
    // estetica  
    console.log("-----");  
  
    // tirando o resultado anterior  
    LogradouroPromise = null;  
    $("#click-callback-promise-cep-result").text(LogradouroPromise);  
  
    // Lendo o que o usuario preencheu no input  
    var cep = $("#click-callback-promise-cep-input").val();  
  
    // marcando a hora que chamamos a API  
    console.log("antes ", new Date());  
  
    // indo buscar os dados do CEP e chamando a função  
    GetCep(cep)  
        .then(QuandoVoltarComOCepPromise);  
  
    // imediatamente após a chamada  
    console.log("depois ", new Date());  
  
    // ainda deve ser null  
    console.log("LogradouroPromise ", LogradouroPromise);  
}
```

```
function GetCep(cep) {  
    return new Promise(function (resolve, reject) {  
        $.getJSON("https://viacep.com.br/ws/" + cep + "/json/", resolve, reject);  
    });  
}
```

```
var LogradouroPromise;  
  
function QuandoVoltarComOCepPromise (info) {  
  
    // preenchendo nossa variavel  
    LogradouroPromise = info.logradouro;  
  
    // marcando a hora da  
    console.log("chegou !!! ", new Date());  
  
    $("#click-callback-promise-cep-result").text(LogradouroPromise);  
}
```

Quando o `.then` for acionado, vai
chamar
`QuandoVolarComOCepPromise`



Exercício Proposto

- Dentro de *QuandoVoltarComOCepPromise* e *QuandoVoltarComOCep* temos o parâmetro *info*
- Escreva *console.log(info)* dentro dessas funções e veja os atributos no console do navegador
- Existe um atributo chamado *ibge* dentro de *info (info.ibge)*
 - *igual info.logradouro*
- Existe uma API do IBGE que informa os detalhes do municipio por código ibge
 - https://servicodados.ibge.gov.br/api/v1/localidades/municipios/{codigo_ibge}



Exercício Proposto

- Munido do atributo *ibge*, busque a região e complemente o exemplo 2 (usando callbacks igual o CEP)
- Exemplo:

Exemplo 2

Buscar CEP

Rua Prefeito Tigre Maia **Fica no Sudeste**



Exercício Proposto

- Itajubá tem código ibge 3132404, logo você irá chamar a API
<https://servicodados.ibge.gov.br/api/v1/localidades/municipios/3132404>
- Clique e veja o exemplo de como o dado irá chegar
- Complemente o Exemplo 3 usando Promise para buscar a região



Créditos

- Slides produzidos por Guilherme Ferreira
 - ferreiraga@outlook.com