

# JavaScript

Desenvolvimento web para bioinformática

**Aula 4**

# Sugestões de projetos

- Reproduza a interface de uma ferramenta do LBS usando um framework de sua preferência
  - Exemplo: Next.js, Angular, Vue.js, Laravel, Django, Gatsby, etc.
- Crie pelo menos duas páginas navegáveis
- Lista de ferramentas
  - <http://bioinfo.dcc.ufmg.br/>



**RNAPEDIA**  
Luana Bastos, Mariano e de Mel...

RNApedia is a public database of RNA-protein interactions structures. RNApedia calculates contacts b...

[Ver mais](#)



**COCaDA**  
Rafael Lemos, D. Mariano, Sabr...

COCaDA (Contact Optimization by alpha-Carbon Distance Analysis) is a efficient algorithm for identif...

[Ver mais](#)



**PROPEDIA2**  
Martins, Mariano, et al.

PROPEDIA é um banco de dados de complexos peptídeo-proteína agrupados em três metodologias: base...

[Ver mais](#)



**GLUTANTBASE**  
Diego Mariano, Naiara Pautza

Glutantbase é um banco de dados, ferramenta web e método para avaliar mutações para proteínas ?...

[Ver mais](#)



**SSV**  
Diego Mariano

SSV (Structural Signature Variation ) is a method to propose mutations for enzymes used in industria...

[Ver mais](#)



**E-VOLVE**  
Vitor Pimentel / André Rodri...

Evolve is a Webtool designed to model mutations in the input protein complex using Modeller. Then, u...

[Ver mais](#)



**VTR**  
Vitor Pimentel / Diego Mariano

VTR is a Webtool for calculate and match contacts in two proteins. Using TM-Align for the Protein Al...

[Ver mais](#)



**PROPEDIA**  
Pedro Martins; Lucianna H. San...

PROPEDIA is a database of peptide-protein complexes clustered in three methodologies: (i) peptide ...

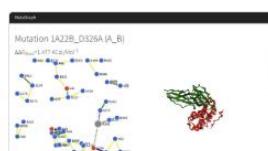
[Ver mais](#)



**PROTEUS**  
José Renato / Diego Mariano

Protein engineering is a widely adopted technique for the creation of synthetic proteins not found i...

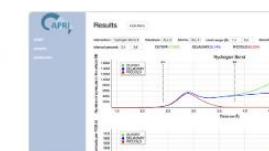
[Ver mais](#)



**MUTAGRAPH**  
Laerte Mateus Rodrigues

The change of one amino acid by another within a tertiary structure may or may not affect the functi...

[Ver mais](#)



**CAPRI**  
Pedro Martins

Studies involving protein structures most often deal with a large amount of information. To understa...

[Ver mais](#)



**PROTEINGO**  
Marcos Felipe, Pedro Martins, ...

Proteingo é um jogo para análise de contatos em proteínas.

[Ver mais](#)

# JAVASCRIPT

# O que é?

- Linguagem de programação *client-side*
- Tipagem dinâmica e fraca
- Principais tecnologias da Internet
  - HTML
  - CSS
  - JavaScript



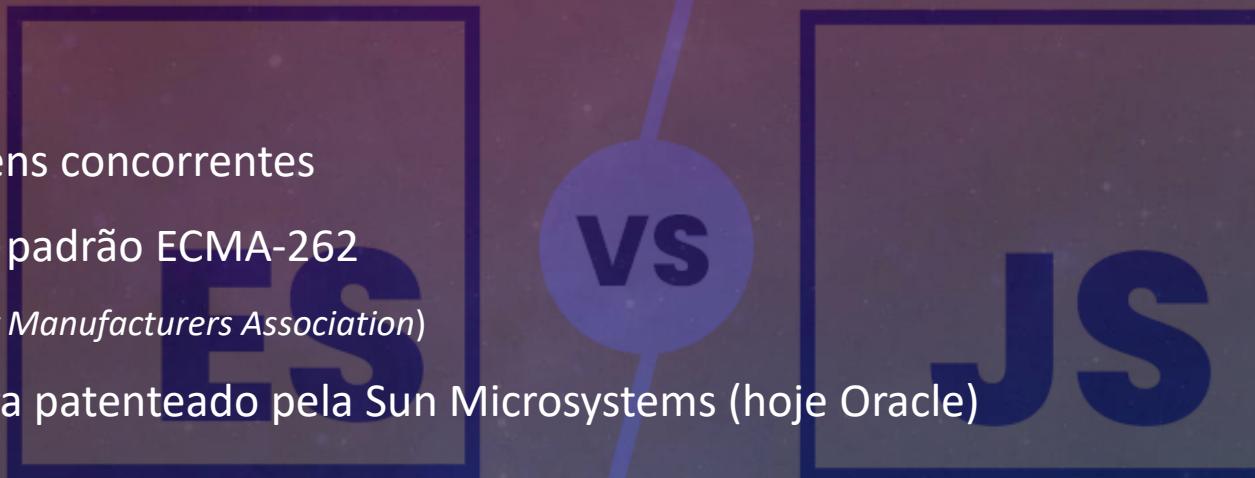


CRIADA EM 1995  
| BRENDAN EICH  
| NETSCAPE

# JavaScript ≠ Java

# ECMAScript

- Anos 1990 - Linguagens concorrentes
- ECMA decidiu criar o padrão ECMA-262
  - (*European Computer Manufacturers Association*)
- O nome JavaScript era patenteado pela Sun Microsystems (hoje Oracle)
- ECMAScript
  - Popular: JavaScript



# Exemplo de uso

---

**40% off**WE'VE EXTENDED OUR  
CYBER MONDAY SALE

SAVE NOW

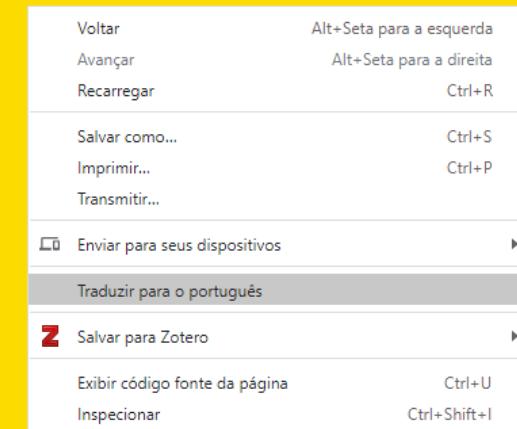
CELEBRATING

# 25 years of JavaScript

1,444,231 *libraries* and counting...

## 25 free courses for #JavaScript25

To celebrate one of the most popular languages in the world, we're making five of our expert-authored JavaScript courses free every week in December.



**40% off**

WE'VE EXTENDED OUR  
CYBER MONDAY SALE

[SAVE NOW](#)

Exceto texto em imagens

A C O M E M O R A R

# 25 anos de JavaScript

1,444,231 **bibliotecas** *e contando ...*

**25 cursos gratuitos para # JavaScript25**

Para celebrar uma das linguagens mais populares do mundo, estamos disponibilizando gratuitamente cinco de nossos cursos de autoria especializada em JavaScript todas as semanas em dezembro.

Com um clique toda a página foi traduzida para o português

Pesquise no Google Maps



Casa  
Definir local

Trabalho  
Definir local

Trânsito leve nesta área  
Atrasos de até 30 min



Pesquisar nesta área



Supermercados



Restaurantes

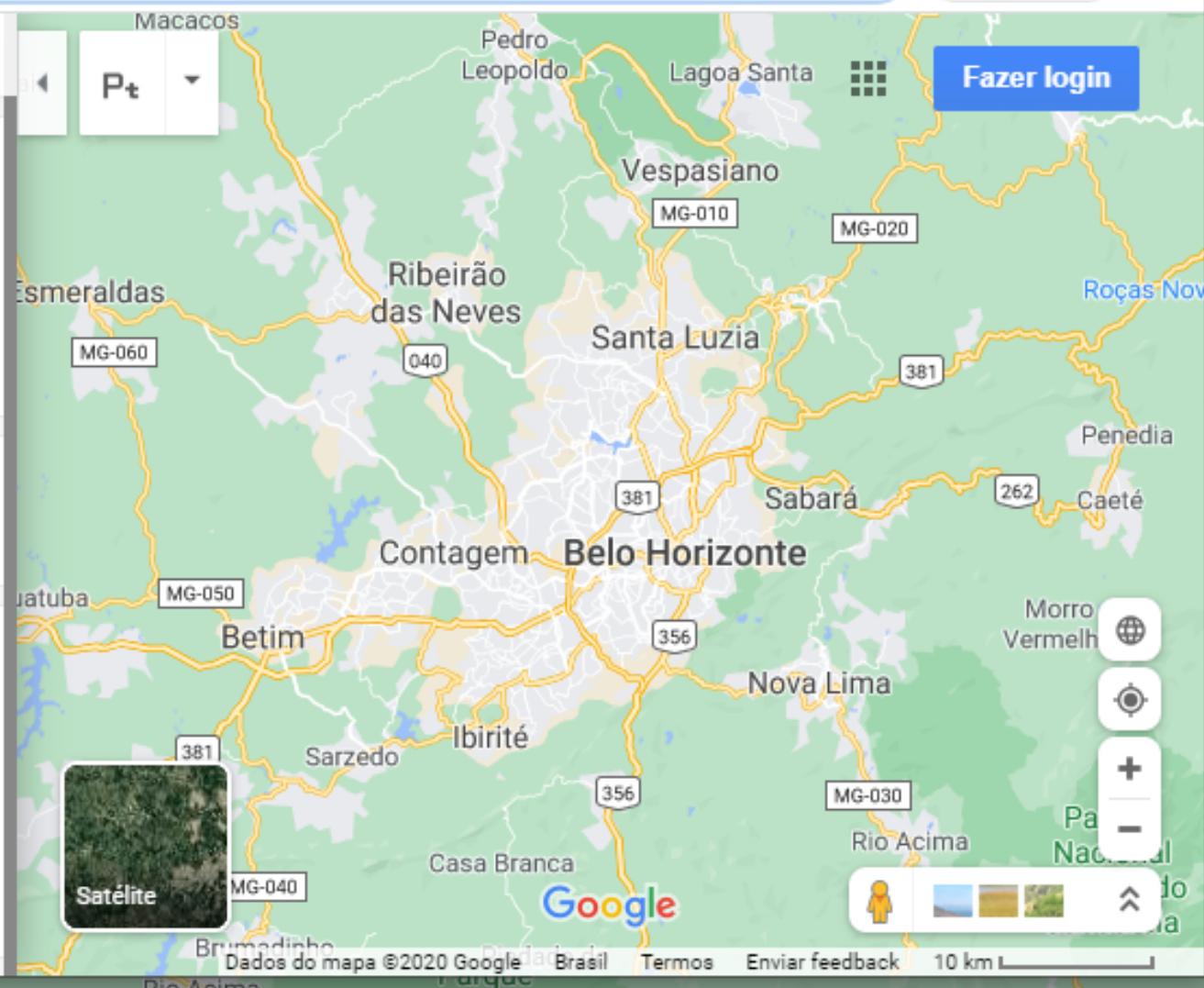


Hotéis



Mais

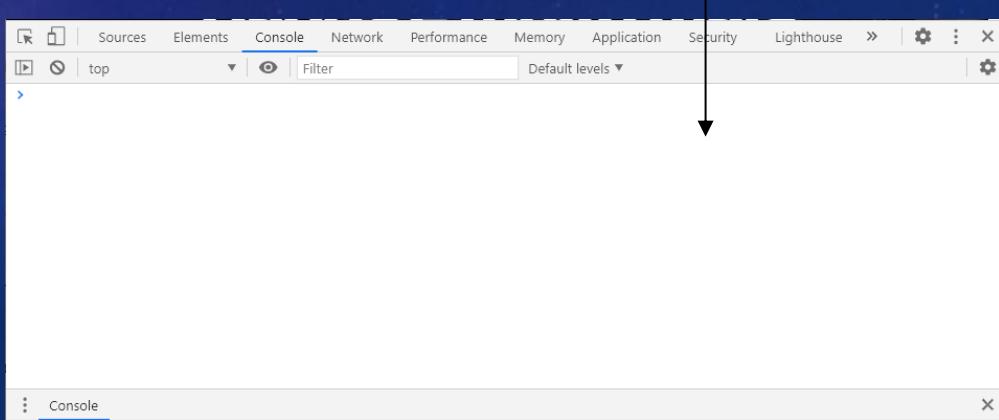
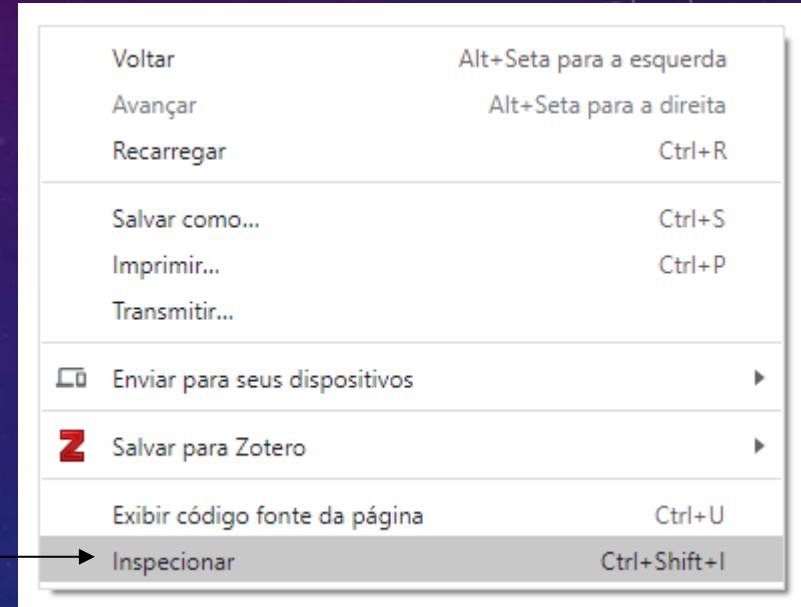
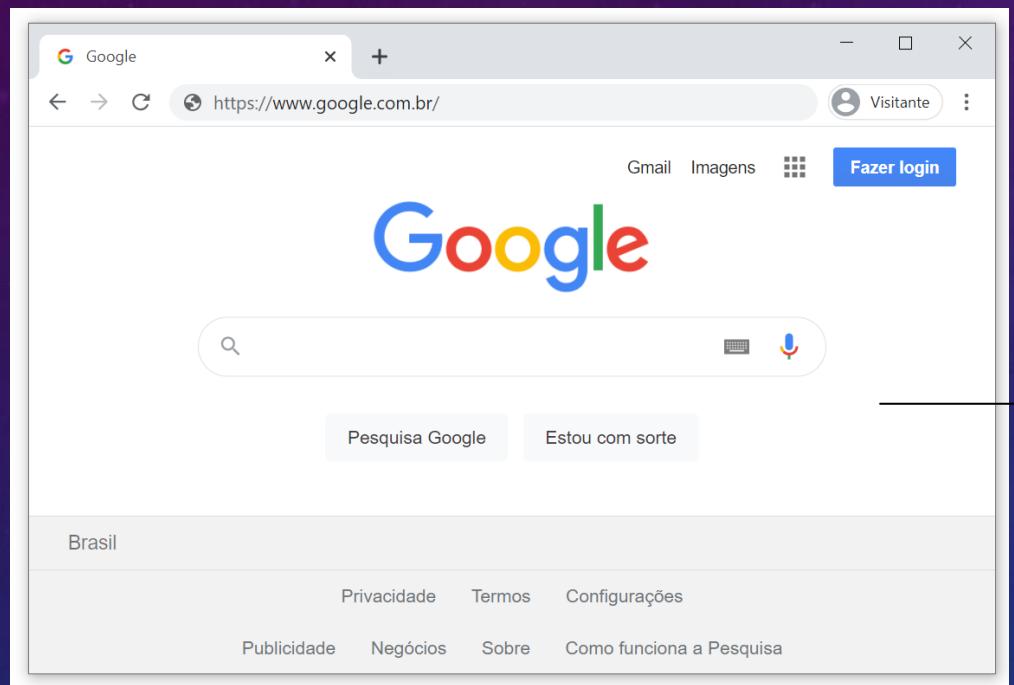
Ocultar tudo

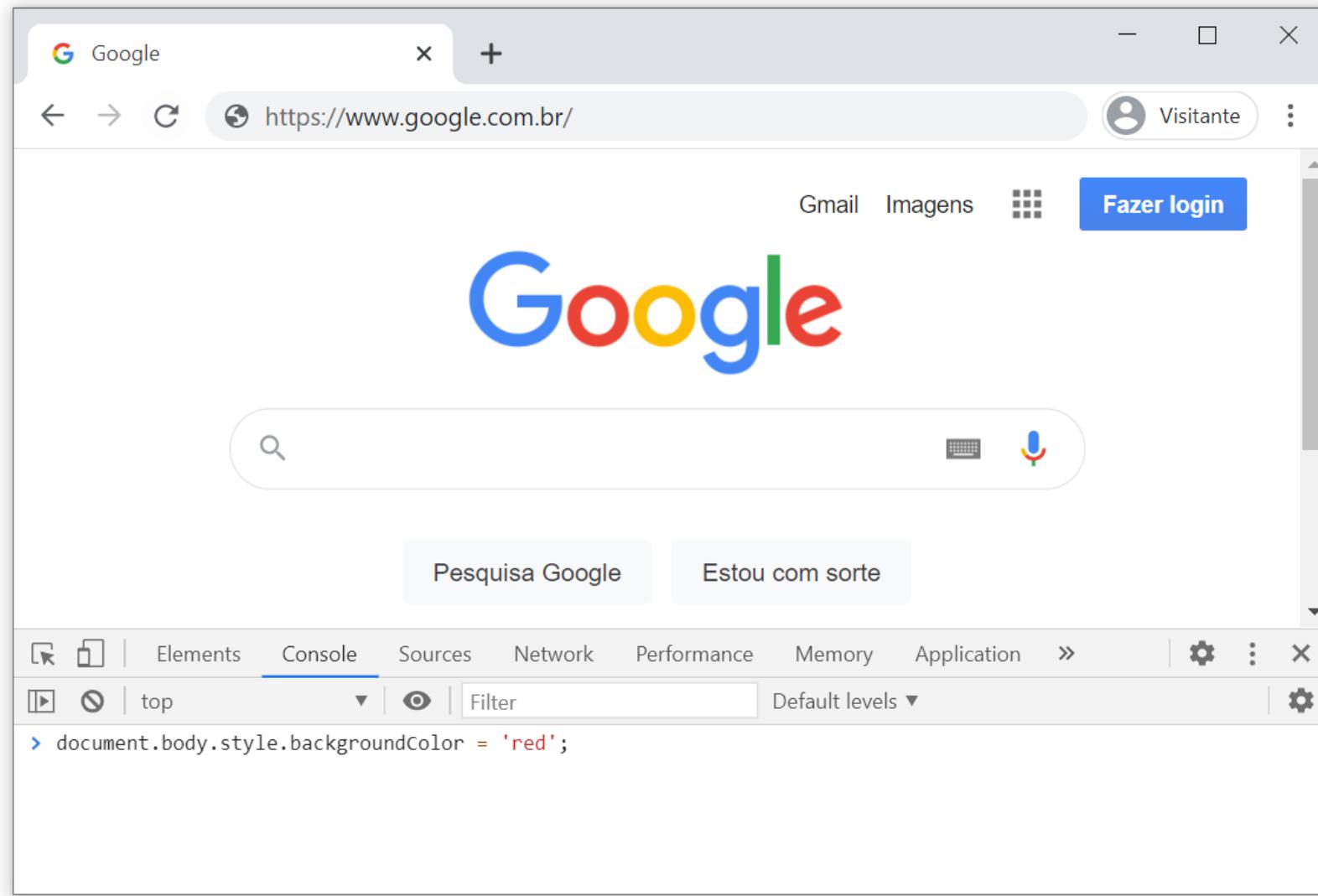


# EXEMPLO PRÁTICO

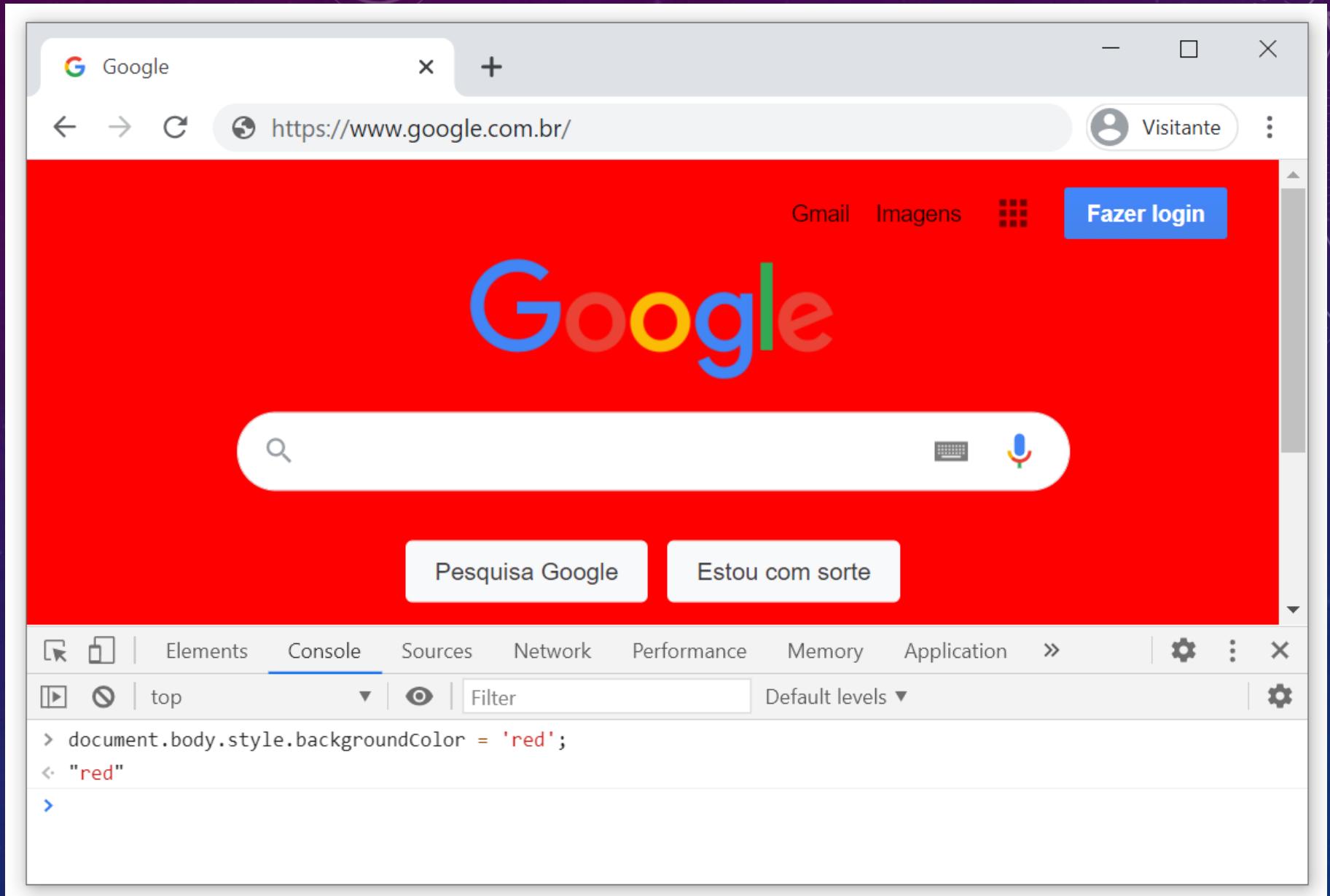
USANDO JAVASCRIPT PARA  
ALTERAR A COR DE FUNDO

# INSPECIONAR ELEMENTO





```
document.body.style.backgroundColor = 'red';
```



```
document.body.style.backgroundColor = 'red';
```

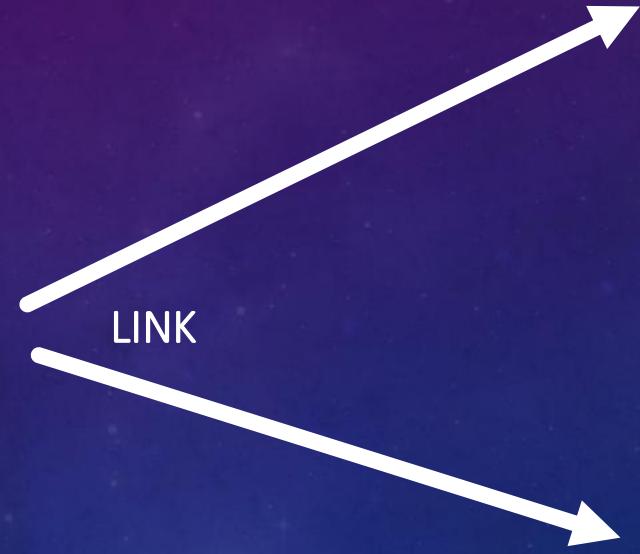
documento

tag <body>

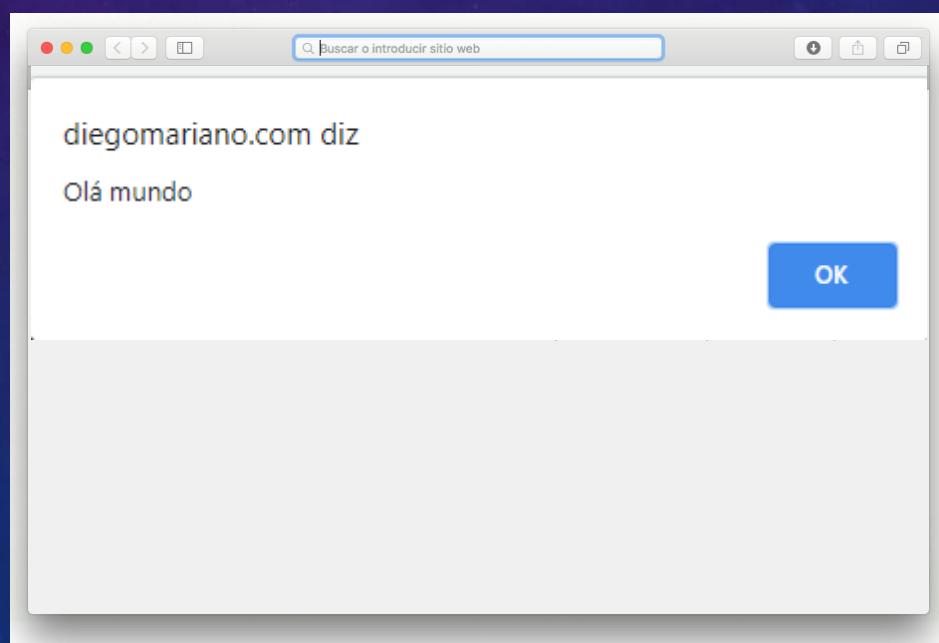
Cor de fundo  
(propriedade *background-color*)

Alterar o estilo

vermelho



```
index.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>js</title>
5   <meta charset="utf-8">
6
7   <!-- scripts -->
8   <script src="script.js"></script>
9
10 </head>
11
12 <body>
13
```



```
script.js
1 alert("Olá mundo!")
```

# Recursos da linguagem

---

# let, const, var

Há três formas de declaração no JavaScript: **var**, **let** e **const**.

- **var**: declara uma variável global;

```
var x = 10;
```

- **let**: declara uma variável de escopo;

```
let x = 10;
```

- **const**: declara uma constante (ou seja, somente leitura).

```
const x = 10;
```

Observe a diferença entre **var** e **let**:

var

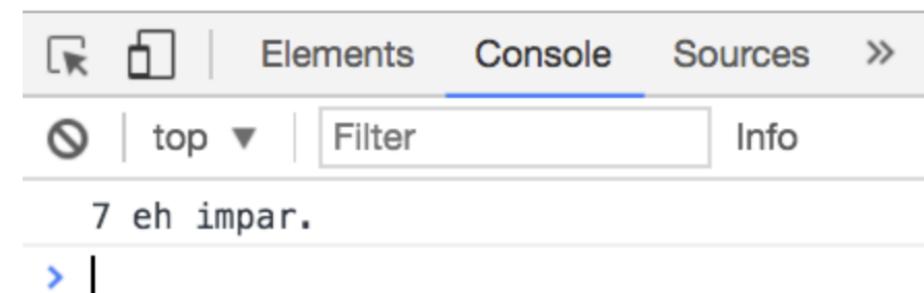
```
> var x = 10;  
< undefined  
> if(true){  
    var x = 20;  
}  
< undefined  
> x  
< 20
```

let

```
> let x = 10;  
< undefined  
> if(true){  
    let x = 20;  
}  
< undefined  
> x  
< 10
```

# Comandos condicionais if/else

```
1  /* Script */
2  let num = 7;
3  if (num % 2 == 0) {
4      console.log(num+" eh par.");
5  } else if (num % 2 == 1) {
6      console.log(num+" eh impar.");
7  } else {
8      console.log("Numero invalido.");
9 }
```



## Loop while

As sintaxes dos laços *for* e *while* são bastante parecidas com as sintaxes desses comandos em PHP.  
O *loop while* percorre dados verificando se uma condição pré-determinada foi estabelecida.

```
1  /* Script */
2  var i = 0;
3  while(i < 10){
4      console.log(i);
5      i++;
6  }
```

## Loop for

O *loop for* permite a iteração controlada sobre dados. Observe o exemplo a seguir:

```
1  for(var i = 0; i < 10; i++){
2      console.log(i);
3  }
```

# Objetos em JavaScript

JSON

Em JavaScript, retirando os tipos de dados primitivos, como *null* e *undefined*, tudo é um objeto<sup>[1]</sup>. Mesmo dados primitivos como *strings* e números podem ser manipulados como se fosse objetos: JavaScript cria um wrapper que os transforma em objetos<sup>[2]</sup>.

Observe como podemos acessar um objeto:

- `objeto.propriedade`
- `objeto.método()`

```
> let objeto = {  
    "a": "ameixa",  
    "b": "biscoito",  
    "c": "coelho"  
}  
< undefined  
-----  
> objeto['a']  
< 'ameixa'
```

```
> objeto.a  
< 'ameixa'  
-----  
> objeto.b  
< 'biscoito'  
-----  
> objeto.c  
< 'coelho'
```

# Exercício

- Use um laço de repetição para exibir todo o conteúdo do objeto a seguir.
  - Dica: use o operador `in`

```
> let objeto = {  
    "a": "ameixa",  
    "b": "biscoito",  
    "c": "coelho"  
}
```

# Arrays em JavaScript

Em JavaScript, *arrays* (ou listas) são considerados um tipo de objeto. Podemos criar objetos do tipo *array* usando valores separados por vírgulas dentro de colchetes [ ]. Cada elemento de um *array* pode ser acessado por sua posição de inserção (note que a contagem começa em zero). Por exemplo:

```
> let x = [1, 2, 3]
```

```
< undefined
```

---

```
> x
```

```
< ▶ (3) [1, 2, 3]
```

---

```
> x[0]
```

```
< 1
```

---

# Funções

Funções em JavaScript funcionam de maneira similar a PHP. Funções devem ser declaradas antes de sua chamada. Podem receber parâmetros e retornar dados. Observe o exemplo da implementação de uma função que recebe dois numerais e retorna o resultado da soma:

```
1 /* Script soma */
2 function soma(a,b){
3     return a + b;
4 }
5 var x = 1;
6 var y = 2;
7 var resultado = soma(x,y);
8 console.log(resultado); //3
```

# Declarando uma função

Podemos declarar uma função utilizando a palavra-chave *function* seguido de um nome e parênteses. Observe uma função simples que apenas exibe uma mensagem no console.

```
> // declaração
  function saudacao(){
    console.log("Olá!");
  }

  // chamada
  saudacao()
  Olá!
```

# Passando parâmetros de parâmetros

Funções podem receber valores como entrada.

```
> // declaração
  function saudacao(nome){
    console.log("Olá, "+nome+"!");
  }

  // chamada
  saudacao("José");

olá, José!
```

```
> saudacao("José");
  Olá, José!
< undefined
> saudacao("Maria");
  Olá, Maria!
< undefined
> saudacao("Pedro");
  Olá, Pedro!
< undefined
> saudacao("Carla");
  Olá, Carla!
```

# Function expression

!

Uma expressão de função (*function expression*) se difere de uma declaração de função tradicional na forma a qual declaramos um nome.

*function declaration*

```
> function soma(a, b){  
    return a+b;  
}
```

*function expression*

```
> const soma = function(a, b){ return a+b; }
```

# Arrow functions

*Arrow functions*, ou na tradução para o português “funções de seta”, são um tipo de sintaxe utilizada para escrever funções de forma mais condensada. Observe a sintaxe do uso de uma *arrow function*:

**variável = (parâmetros) => { /\* ... \*/ };**

*function expression*

```
const incrementa = function(n){ return n+1 }
```

*arrow function*

```
const incrementa_AF = (n) => { return n+1 }
```

## Simplificando

`x = n => { return n+1 }`

Dá pra simplificar mais?

`y = n => n+1`

Resultado:

Note como as funções x e y obtém resultados idênticos:

```
> x(100)
```

```
< 101
```

---

```
> y(100)
```

```
< 101
```

# Exercício

- Escreva uma função que retorne o quadrado de um número
- Reescreva a função usando *arrow function*

## Eventos

Eventos permitem que ações que ocorrem no HTML açãoem interações com o código JavaScript.

Eventos podem ser:

- Quando uma página é carregada;
- Quando um botão é clicado;
- Quando passamos o mouse por cima de um elemento HTML;
- Quando digitamos em um campo de texto;
- Dentro outras inúmeras ações.

Eventos podem ser descritos em um elemento HTML usando um atributo específico:

```
<elemento evento= "alguma_acao_javascript">
```

<b>Evento</b>	<b>Descrição</b>
<i>onchange</i>	Quando um elemento HTML foi mudado.
<i>onclick</i>	Quando um elemento é clicado.
<i>onmouseover</i>	Quando o mouse passa por cima de um elemento.
<i>onmouseout</i>	Quando o mouse sai de cima de um elemento.
<i>onkeydown</i>	Quando uma tecla do teclado é digitada.
<i>onload</i>	Quando uma página é carregada.

## EXEMPLO

### Antes do click

Cabeçalho

Colorir fundo

Rodapé

```
<script>
    function colorir(){
        document.querySelector('body').style.backgroundColor = 'green';
    }
</script>

<button onclick="colorir()">Colorir fundo</button>
```

### Após o click

Cabeçalho

Colorir fundo

Rodapé

# Manipulando o DOM

- DOM = *document object model*
- Selecione o elemento com o **querySelector**

```
let selecionado = document.querySelector([SELETOR])
```

- Altere o estilo com **style.[propriedade]**

```
selecionado.style.backgroundColor = 'green'; // muda o fundo para verde
```

# Inserindo textos

```
selecionado.textContent = 'Valor gravado na tela';
```

Ou inclua um **HTML**:

```
selecionado.innerHTML = '<div class="bg-danger">Olá</div>';
```

# Bibliotecas JavaScript

+

•

○

+

•

○

# DataTables

1 - Inclua esses dois arquivos ↓

css //cdn.datatables.net/2.3.2/css/dataTables

js //cdn.datatables.net/2.3.2/js/dataTables.

2 - Inicialize sua DataTable: ↓

```
1 let table = new DataTable('#myTable');
```

<https://datatables.net/>

10 entries per page					Search:
Name	Position	Office	Age		
► Ari Satou	Accountant	Tokyo	33		
► Angelica Ramos	Chief Executive Officer (CEO)	London	47		
► Ashton Cox	Junior Technical Author	San Francisco	66		
► Bradley Greer	Software Engineer	London	41		
► Brenden Wagner	Software Engineer	San Francisco	28		
► Brielle Williamson	Integration Specialist	New York	61		
► Bruno Nash	Software Engineer	London	38		
► Caesar Vance	Pre-Sales Support	New York	21		
► Cara Stevens	Sales Assistant	New York	46		
► Cedric Kelly	Senior Javascript Developer	Edinburgh	22		

Showing 1 to 10 of 57 entries

« < 1 2 3 4 5 6 > »

Veja também <https://diegomariano.com/datatables/>

# Exercício

Crie a seguinte tabela com DataTables:

Código necessário

## CSS

<https://cdn.datatables.net/2.3.2/css/dataTables.dataTables.min.css>

## Script

<https://cdn.datatables.net/2.3.2/js/dataTables.min.js>

## Iniciando a tabela

```
let table = new DataTable('#nome-tabela');
```

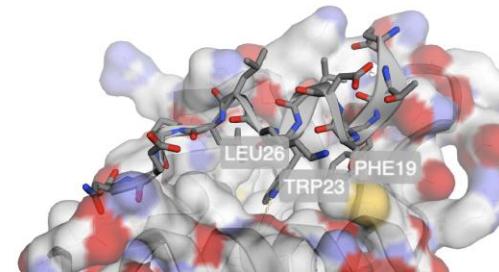
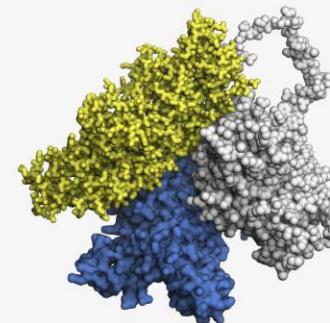
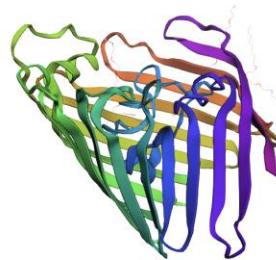
Código	Nome
A	Alanina
R	Arginina
N	Asparagina
D	Aspartato
C	Cisteína
E	Glutamato
Q	Glutamina
G	Glicina
H	Histidina
I	Isoleucina
L	Leucina
K	Lisina
M	Metionina
F	Fenilalanina
P	Prolina
S	Serina
T	Treonina
W	Triptofano
Y	Tirosina
V	Valina

# 3Dmol.js

**3Dmol.js**

View Embed Teach Jupyter Develop  
*A modern, object-oriented JavaScript library for visualizing molecular data*

Feedback Download

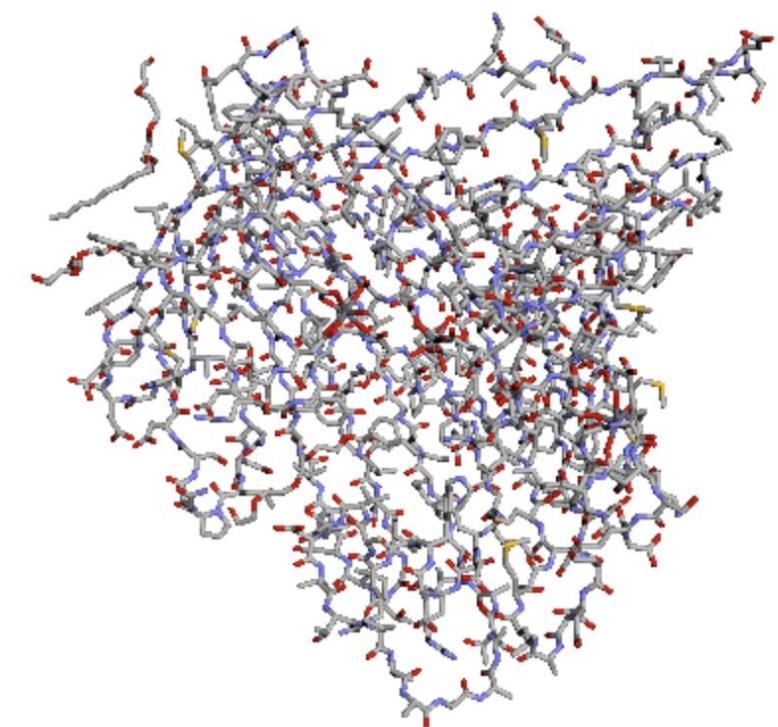


<https://3dmol.csb.pitt.edu/>

# Exemplo (carregamento via HTML)

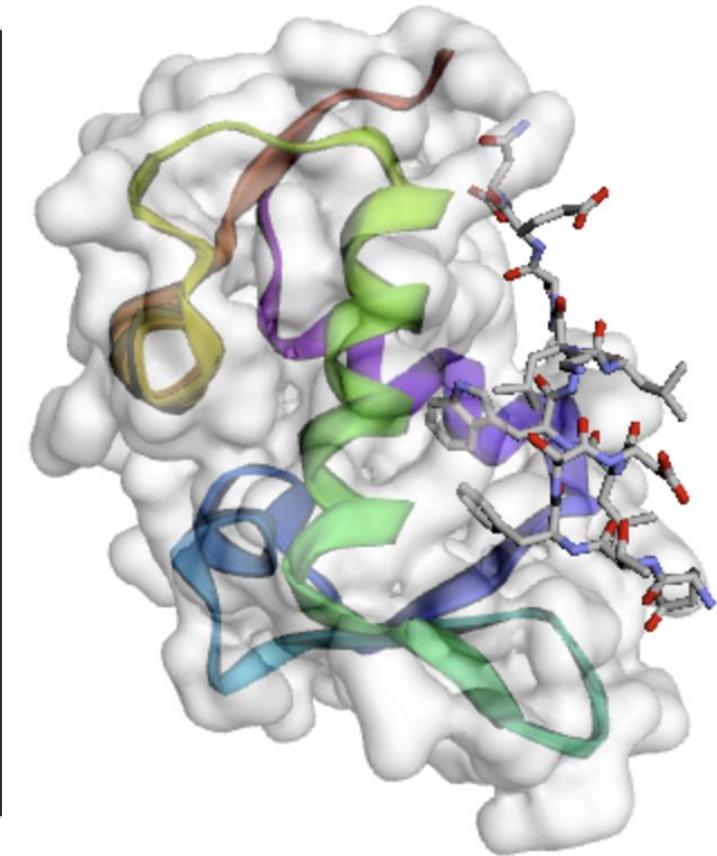
```
<script src="http://3Dmol.csb.pitt.edu/build/3Dmol-min.js"></script>

<div
  style="height: 400px; width: 400px; position: relative;"
  class='viewer_3Dmoljs'
  data-pdb='2POR'
  data-backgroundcolor='0xffffffff'
  data-style='stick'
></div>
```



# Múltiplas cadeias

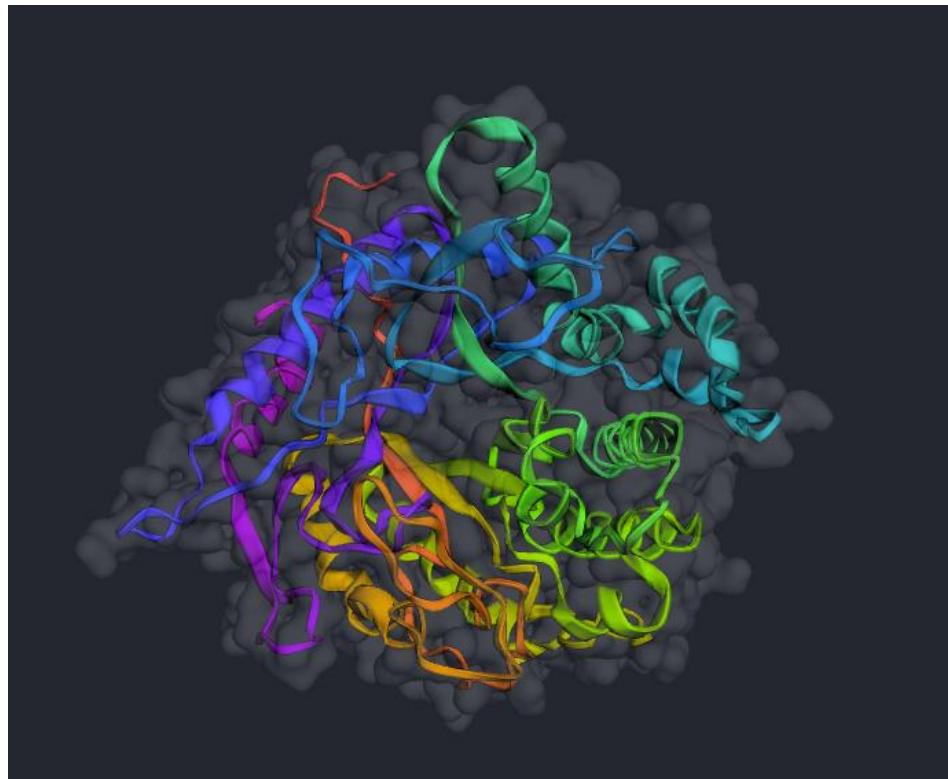
```
1 <div  
2 style="height: 400px; width: 400px; position: relative;"  
3 class='viewer_3Dmoljs'  
4 data-pdb='1YCR'  
5 data-backgroundcolor='0xffffffff'  
6 data-select1='chain:A'  
7 data-style1='cartoon;color=spectrum'  
8 data-surface1='opacity:.7;color:white'  
9 data-select2='chain:B'  
10 data-style2='stick'  
11></div>
```



# Exemplo (JavaScript)

## No HTML

```
1 <div id="pdb"></div>
```



```
1 var glviewer = null;
2
3 /* Load default PDB */
4 document.onload = readPDB("1bga.pdb");
5
6 /* Reading PDB */
7 function readPDB(id){
8     var txt = id;
9
10    $.post(txt, function(d) {
11        moldata = data = d;
12
13        /* Cria visualizacao */
14        glviewer = $3Dmol.createViewer("pdb", {
15            defaultcolors : $3Dmol.rasmolElementColors
16        });
17
18        /* Define cor do fundo*/
19        glviewer.setBackgroundColor(0x242830);
20
21        receptorModel = m = glviewer.addModel(data, "pqr");
22
23        /* Define o tipo de visualizacao*/
24        glviewer.setStyle({}, {cartoon:{color:'spectrum'}}); /* Cartoon multi-color */
25        glviewer.addSurface($3Dmol.SurfaceType, {opacity:0.3}); /* Surface */
26
27        /* Define os nomes dos atomos*/
28        atoms = m.selectedAtoms({});
29        for ( var i in atoms) {
30            var atom = atoms[i];
31            atom.clickable = true;
32            atom.callback = atomcallback;
33        }
34
35        glviewer.mapAtomProperties($3Dmol.applyPartialCharges);
36        glviewer.zoomTo();
37        glviewer.render();
38    });
39 }
```

```
var glviewer = null; /* Load default PDB */ document.onload = readPDB("1bga.pdb"); /* Reading PDB */ function readPDB(id){ var txt = id; $.post(txt, function(d) { moldata = data = d; /* Cria visualizacao */ glviewer = $3Dmol.createViewer("pdb", { defaultcolors : $3Dmol.rasmolElementColors }); /* Define cor do fundo*/ glviewer.setBackgroundColor(0x242830); receptorModel = m = glviewer.addModel(data, "pqr"); /* Define o tipo de visualizacao*/ glviewer.setStyle({}, {cartoon:{color:'spectrum'}}); /* Cartoon multi-color */ glviewer.addSurface($3Dmol.SurfaceType, {opacity:0.3}); /* Surface */ /* Define os nomes dos atomos*/ atoms = m.selectedAtoms({}); for ( var i in atoms) { var atom = atoms[i]; atom.clickable = true; atom.callback = atomcallback; } glviewer.mapAtomProperties($3Dmol.applyPartialCharges); glviewer.zoomTo(); glviewer.render();}); }
```

# Exercício

- Crie uma página que exiba o PDB: **1k0p**
- Exiba a estrutura como:
  - Cartoon
  - Sticks
  - Lines
  - Surface

# Fetch API: Lendo dados de uma API com JavaScript

Por Diego Mariano

3 de outubro de 2022

```
1 const json = fetch('https://jsonplaceholder.typicode.com/posts')
2   .then(resposta => resposta.json())
3
4 json.then(dados=>{
5   console.log(dados)
6 })
```

## then() é um método assíncrono

```
const texto = fetch('https://jsonplaceholder.typicode.com/posts') .then(resposta => resposta.text())
texto.then(dados=>{ console.log(dados) })
```

# Exemplo

<https://rest.uniprot.org/uniprotkb/P10635.fasta>

<pre>>sp|P10635|CP2D6\_HUMAN Cytochrome P450 2D6 OS=Homo sapiens OX=9606 GN=CYP2D6 PE=1 SV=2  
MGLEALVPLAVIVAIFLLLVDLMHRRQRWAARYPPGPLPLPGLGNLLHVDFQNTPYCFDQ  
LRRRGFDVFSLQLAWTPVVNLNGLAVERALVTHGEDTADRPPVPTQILFGPRSQGVF  
LARYGPAWREQRRFSVSTRLRNGLGKKSLEQWVTEEAACLAAFAHNSGRPFRPNGLLDK  
AVSNVIASLTGRRFEYDDPRFLRLLDAQEGLKEESGLREVNAVPLVLLHIPALAGKV  
LRFQKAFLTQDDELLTEHRMTWDPAQPPRDLTEAFLAEMEKAKGNESSFNDENLRIVVA  
DLFSAGMVTTS TTLAWGLLLMILHPDVQRRVQQEIDDVIGQVRPEMDQAHMPYTTAVI  
HEVQRFGDIVPLGVTHMTSRDIEVQGFRIPKGTTLITNLSSVLKDEAVWEKPFRHPEHF  
LDAQGHFVKPEAFLPFSAGRRACLGEPARMELFLFTSLLQHFSFSVPTGQPRPSHHGV  
FAFLVSPSPYELCAVPR</pre>

```
<script>  
  
// url de entrada  
const url = "https://rest.uniprot.org/uniprotkb/P10635.fasta";  
  
// requisição dos dados  
const requisicao = fetch(url).then(resposta => resposta.text());  
  
// define o local que o dado será gravado  
const local = document.querySelector("#dados");  
  
// grava o dado no documento  
requisicao.then(dados => local.textContent = dados);  
  
</script>
```

```
<script>  
  
// url de entrada  
const url = "https://rest.uniprot.org/uniprotkb/P10635.fasta";  
  
// requisição dos dados  
const requisicao = fetch(url).then(resposta => resposta.text());  
  
// define o local que o dado será gravado  
const local = document.querySelector("#dados");  
  
// grava o dado no documento  
requisicao.then(dados => local.textContent = dados);  
  
</script>
```

# Exercício

- Construa um script que faça uma requisição a API do UniProt e exiba os dados formatados na tela
  - Para cada sequência exiba o *fullname* e o campo *sequence*

<https://rest.uniprot.org/uniprotkb/search?query=myoglobin&format=json&size=10>

# Formatando sequências FASTA

```
const le_fasta = fasta => {
  return fasta.trim().split(/\^>/m).filter(Boolean).reduce((acc, entry) => {
    const [header, ...seqLines] = entry.trim().split(/\r?\n/);
    acc[header] = seqLines.join('<br>');
    return acc;
  }, {});
}
```

```
<div id="dados"></div>
```

```
// url de entrada
const url = "https://rest.uniprot.org/uniprotkb/P10635.fasta";

// requisição dos dados
const requisicao = fetch(url).then(resposta => resposta.text());

// define o local que o dado será gravado
const local = document.querySelector("#dados");
```

```
<script>
// código completo para copiar
const le_fasta = fasta =>{
  return fasta.trim().split(/\^>/m).filter(Boolean).reduce((acc, entry) =>{
    const [header, ...seqLines] = entry.trim().split(/\r?\n/);
    acc[header] = seqLines.join('<br>');
    return acc;
  }, {});
}

// url de entrada
const url = "https://rest.uniprot.org/uniprotkb/P10635.fasta";

// requisição dos dados
const requisicao = fetch(url).then(resposta => resposta.text());

// define o local que o dado será gravado
const local = document.querySelector("#dados");

// grava o dado no documento
requisicao.then(dados =>{
  //local.textContent = dados
  let json = le_fasta(dados)
  console.log(json)
  for (const chave in json){
    local.innerHTML = `

<div class="accordion" id="accordionExample">
<div class="accordion-item">
<h2 class="accordion-header">
<button class="accordion-button" type="button" data-bs-toggle="collapse" data-bs-target="#collapseOne" aria-expanded="true" aria-controls="collapseOne">
${chave}
</button>
</h2>
<div id="collapseOne" class="accordion-collapse collapse show" data-bs-parent="#accordionExample">
<div class="accordion-body" style="white-space: pre-wrap; max-width: 500px">
${json[chave]}
</div>
</div>
</div>
</div>`;

  }
});

</script>
```

```
// grava o dado no documento
requisicao.then(dados => {
    // local.textContent = dados
    let json = le_fasta(dados)
    console.log(json)

    for (const chave in json) {

        local.innerHTML = `
<div class="accordion" id="accordionExample">
    <div class="accordion-item">
        <h2 class="accordion-header">
            <button class="accordion-button" type="button" data-bs-toggle="collapse"
            data-bs-target="#collapseOne" aria-expanded="true" aria-controls="collapseOne">
                ${chave}
            </button>
        </h2>
        <div id="collapseOne" class="accordion-collapse collapse show"
            data-bs-parent="#accordionExample">
            <div class="accordion-body" style="white-space: pre-wrap; max-width: 500px">
                ${json[chave]}
            </div>
        </div>
    </div>
</div>`;
    }
});
```

# Exibindo com Bootstrap

sp|P10635|CP2D6\_HUMAN Cytochrome P450 2D6 OS=Homo sapiens OX=CYP2D6 PE=1 SV=2

MGLEALVPLAVIVAIFLLLVDLMHRRQRWAARYPPGPLPLPGLGNLLHVDFQNTPYCFDQLRRRGDVFSLQLAWTPVVVLNGLAARREALVTHGEDTADRPPVITQILGFGRPSQGVFLARYGPAWREQRRFSVSTLRNLGLGKKSLEQWVTEEAACLCAAFANHSGRPFRPNGLLDKAVSNVIASLTCGRRFEYDDPRFLRLLDAQEGLKEESGFLREVNAVPLLHIPALAGKVLRFQKAFLTQLDELLTEHRMTWDPAQPPRDLTEAFLAEIMEKAKGNPESSFNDENLRIVADLFSGAMVTSTTLAWGLLLMILHPDVQRRVQQEIDDVIGQVRRPEMGDQAHMPYTTAVIHEVQRFGDIVPLGVTHMTSRDIEVQGFRIPKGTLITNLSVVKDEAVWEKPRFHPEHFIDAQGHFVKPEAFLPFSAGRRACLGEPLARMELFLFTSLLQHFSFSVPTGQPRPSHHGVFAFLVSPSPYELCAVPR

# Node.js



- JavaScript é executado diretamente no navegador
- Node.js permite a execução no servidor
- <https://nodejs.org>

# Frameworks: Next.js

<https://diegomariano.com/next-js/>

Next.js é um framework baseado na biblioteca React usado na construção de aplicações web.

Por ser uma biblioteca, React não fornece opinião sobre como você irá construir sua aplicação. Isso tem vantagens e desvantagens, uma vez que desenvolvedores podem se sentir perdidos sobre como começar a desenvolver uma aplicação com React.

Next.js surge para resolver esse problema especificando exatamente como uma aplicação React deve ser construída. Next.js diz como os arquivos devem ser organizados, como as rotas devem ser criadas, como o ambiente deve ser configurado e muito mais. Com Next.js você consegue criar aplicações React de maneira muito mais organizada!



**Exemplo de aplicação Next.js**

**Welcome to Next.js!**

Get started by editing `pages/index.js`

# Próxima aula

Instalar o XAMPP

**Opcional:** PHP + Composer