JavaScript

Desenvolvimento web para bioinformática

Aula 4

let, const, var

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

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

var x = 10;

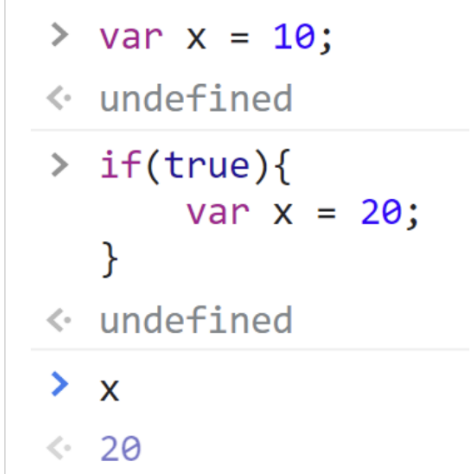
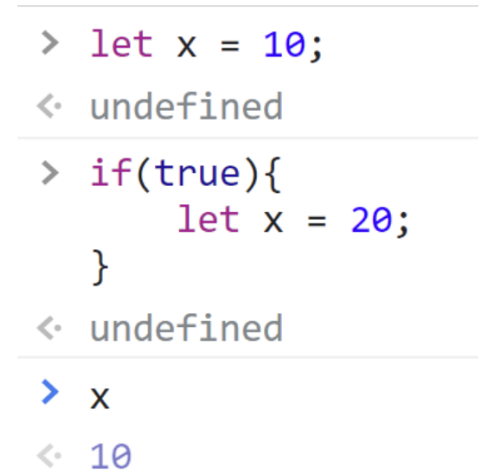
- **let**: declara uma variável de escopo;

let x = 10;

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

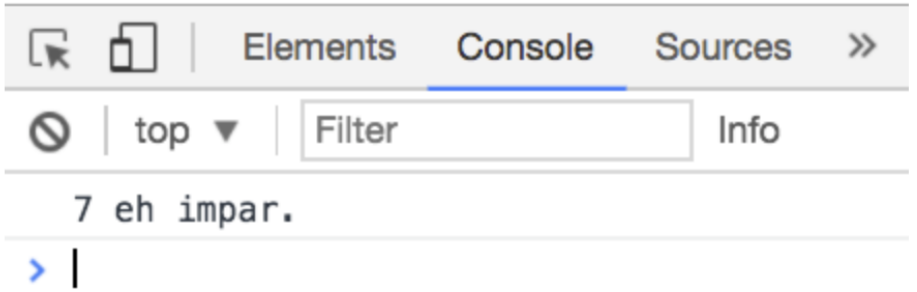
const x = 10;

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

var	let
	

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.

```
/* Script */
var i = 0;
while(i < 10){
    console.log(i);
    i++;
}
```

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

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

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'
```

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:

```
/* Script soma */
function soma(a,b){
    return a + b;
}
var x = 1;
var y = 2;
var resultado = soma(x,y);
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 }
```

```
y = n => n+1
```

Note como as funções x e y obtém resultados idênticos:

```
> x(100)
<· 101
```

```
> y(100)
<· 101
```

Eventos

Eventos permitem que ações que ocorrem no HTML acionem 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”>
```

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

DataTables

1 – Inclua esses dois arquivos

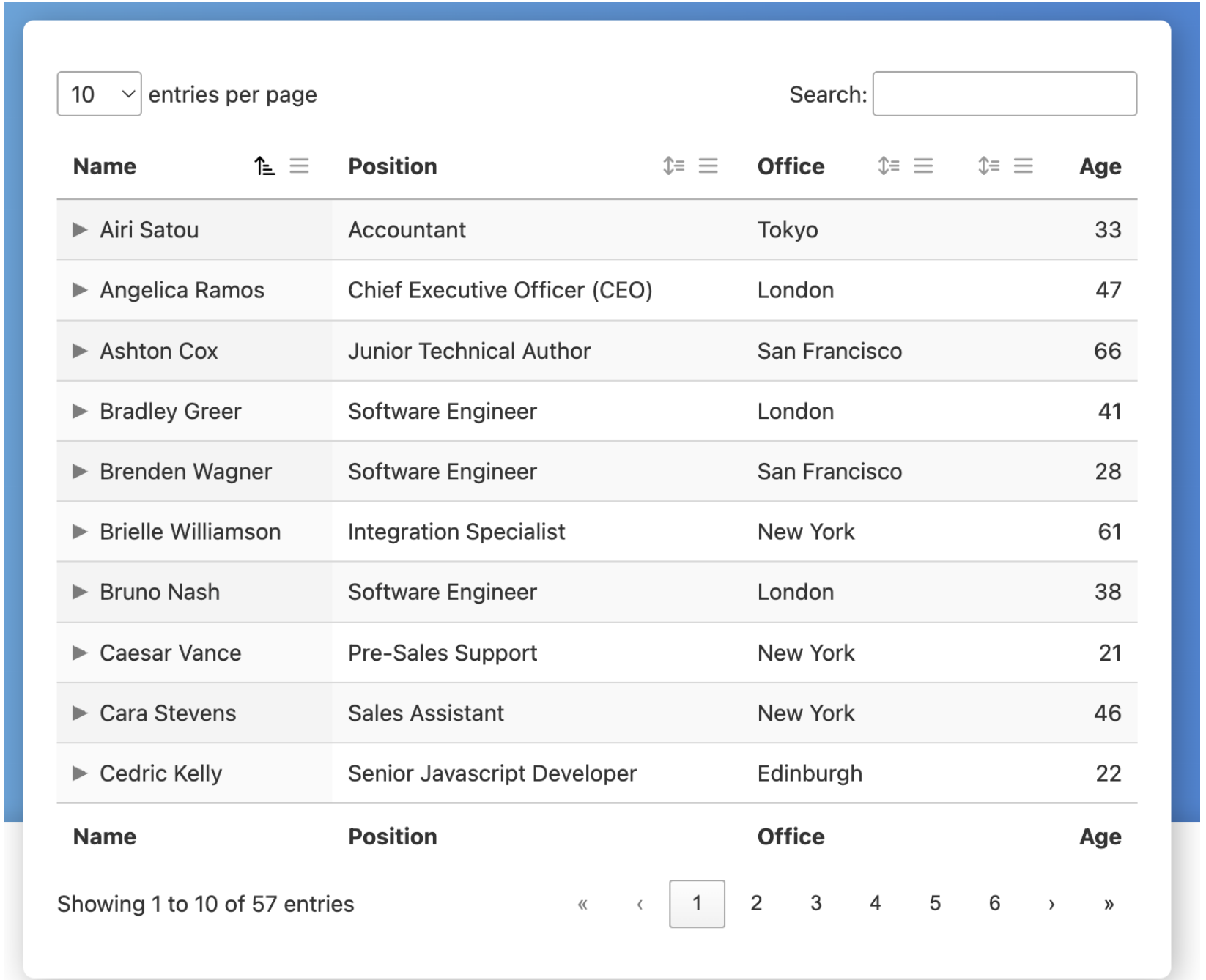
```
CSS  //cdn.datatables.net/2.3.2/css/dataTables
```

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

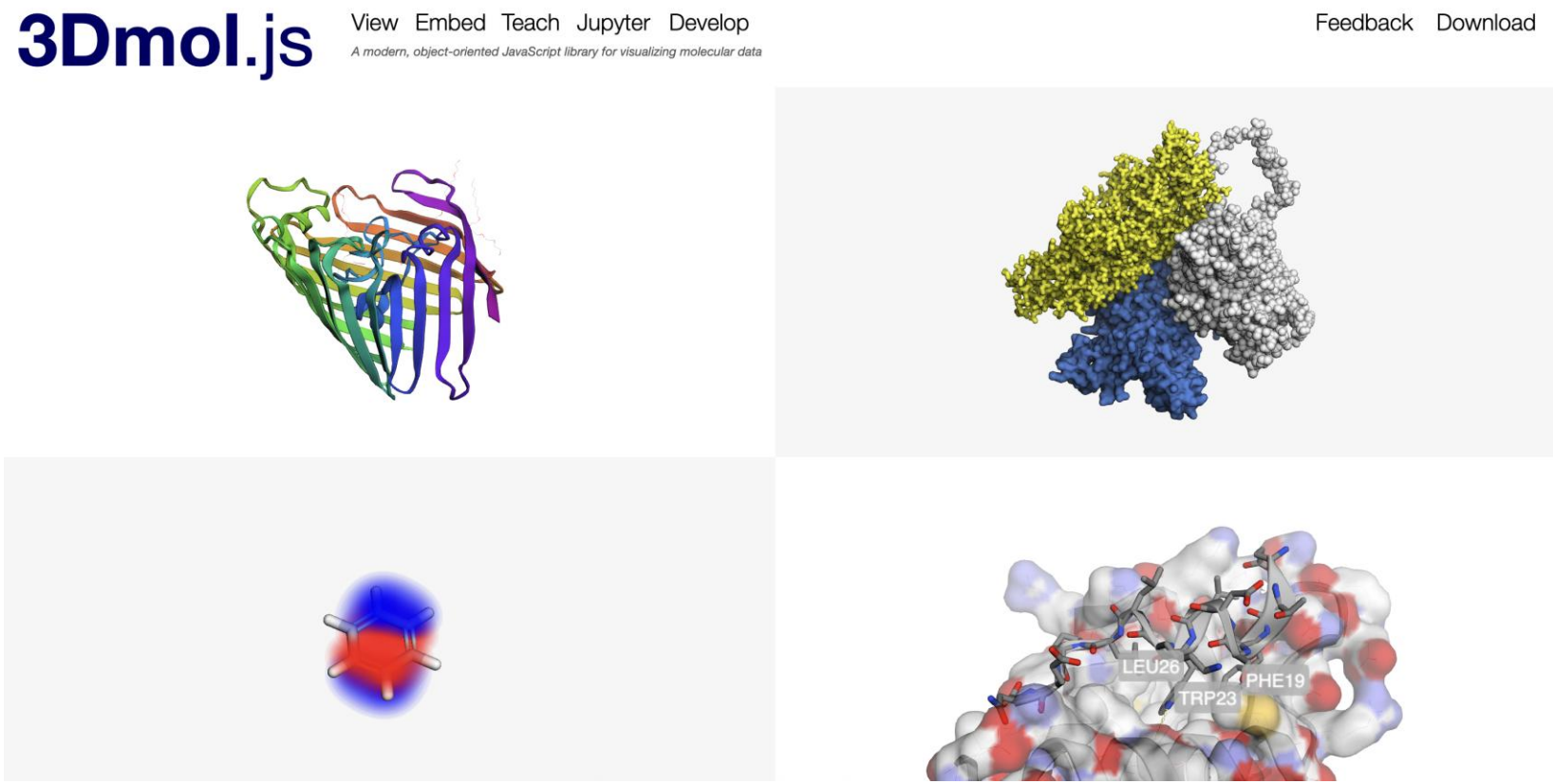
2 – Inicialize sua DataTable:

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

https://datatables.net/



3Dmol.js

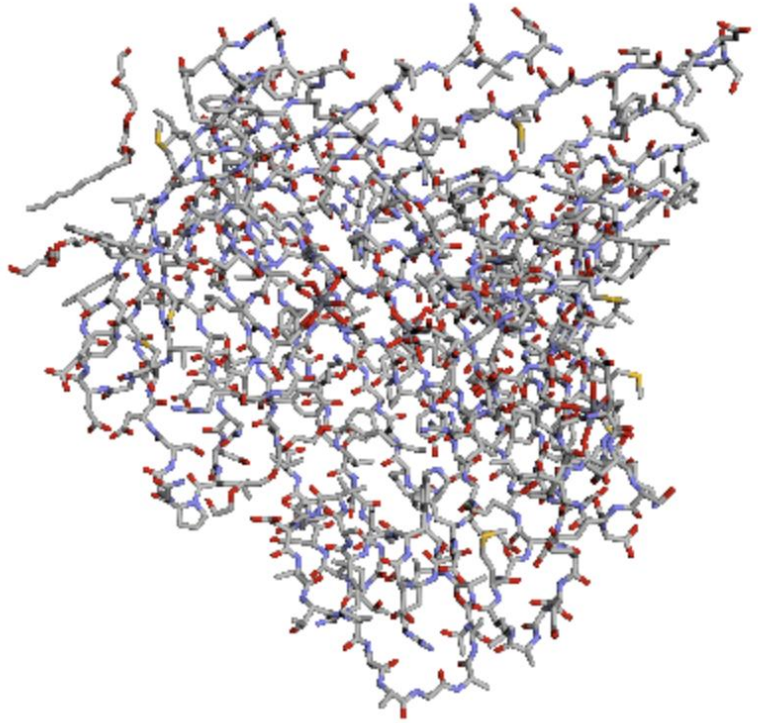


https://3dmol.csb.pitt.edu/

Exemplo (carregamento via HTML)

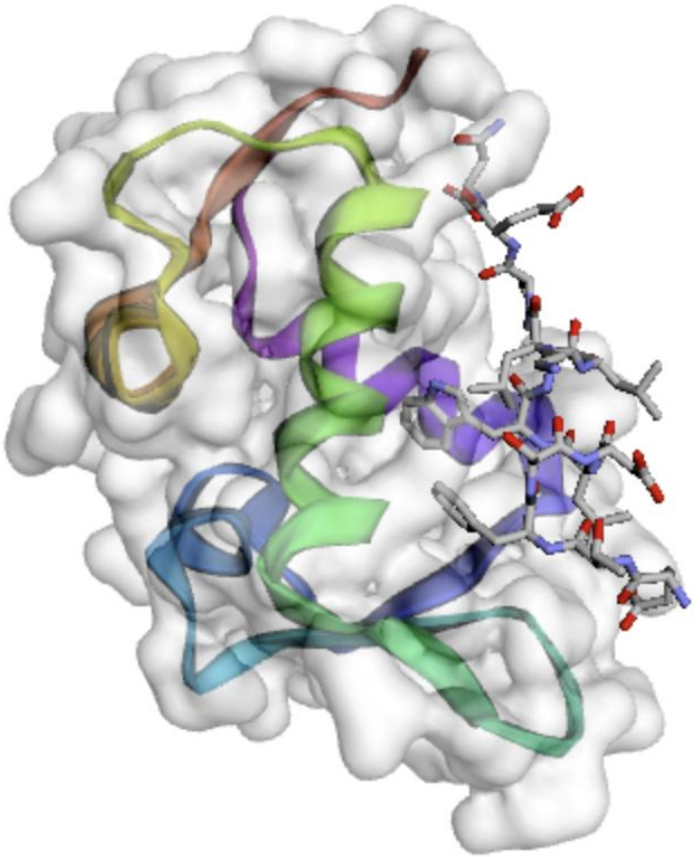
```
<script src="http://3Dmol.csb.pitt.edu/build/3Dmol-min.js"></script>

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



Múltiplas cadeias

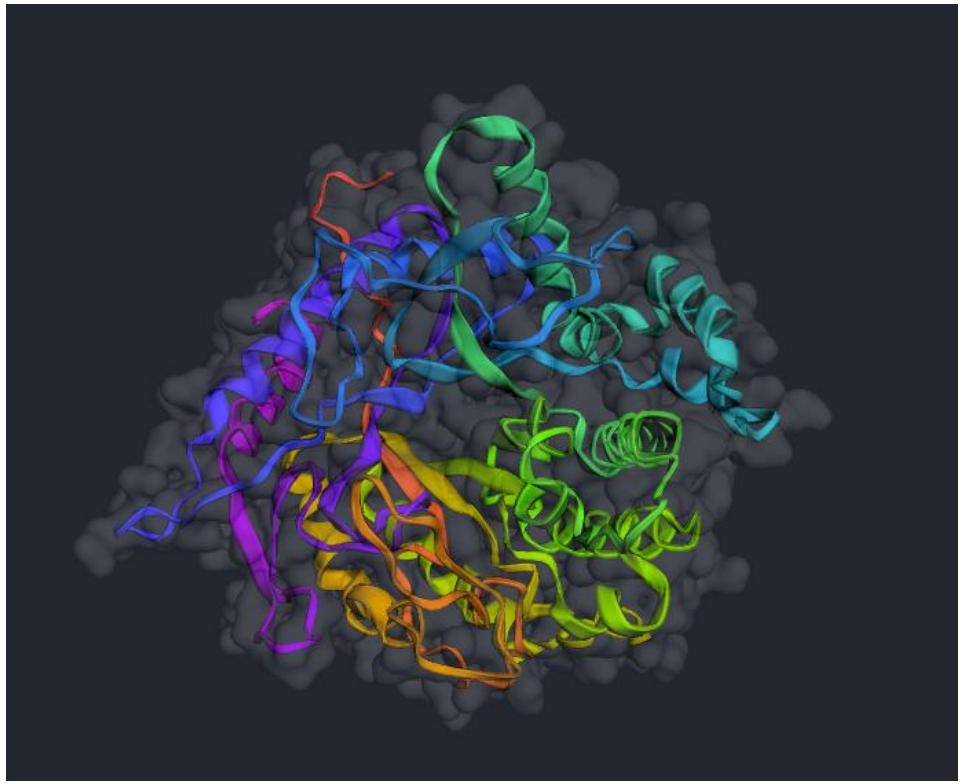
```
<div
style="height: 400px; width: 400px; position: relative;"
class='viewer_3Dmoljs'
data-pdb='1YCR'
data-backgroundcolor='0xffffff'
data-select1='chain:A'
data-style1='cartoon:color=spectrum'
data-surface1='opacity:.7;color:white'
data-select2='chain:B'
data-style2='stick'
></div>
```



Exemplo (JavaScript)

No HTML

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



```
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 do 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();
    });
}
```