Java Script do Curso em Video

No VsCode > escreva > html:5 > enter > Faz o html normal

No final do body adiciona <script></scricpt>

Aula 4: ex001:

<script>

        window.alert('Minha primeira mensagem.')

</script>



O conteúdo some

<script>

        window.confirm('Esta gostando de Java Script?')

</script>



<script>

        window.prompt('Qual é o seu nome?')

</script>

****

<script>

        window.alert('Minha primeira mensagem.')

        window.confirm('Esta gostando de Java Script?')

        window.prompt('Qual é o seu nome?')

</script>

Variáveis: serve para guardar dados

Comentários: // - serve para uma única linha e /\*\*/ - mais de uma linha

##### Identificadores das variáveis:

* Podem começar com letra, $ ou \_
* Não podem começar com números
* É possível usar letras ou números
* É possível usar acentos e símbolos
* Não podem conter espaços
* Não podem ser palavras reservadas (comandos do JS)

No Node.js ou no VsCode > Terminal > New terminal



Para limpar a tela: ctrl + L

Para sair do programa: .exit

Para mostrar qual o tipo da variável no terminal ou Node.js se coloca typeof

Diferença entre VAR  X  LET  X CONST Aula html > udemy > Var.Let.Const.js

//Usando Var - pouco usado

var value; //pode apenas declarar sem descrição de primeira

value = '1232';

//Usando Let

let color = 'Red'; //pode apenas declarar sem descrição de primeira

color = 'Blue'

//Usando Const - NÂO pode alterar, redefinir, da erro - mais usado

const fullName = 'Larytgs' //sempre tem q ter uma descrição



console.log(value);

console.log(color);

console.log(fullName);

#### Tipos Primitivos Primordiais

|  |  |
| --- | --- |
| Number | 5 18 -12 0.5 -15.9 3.14 8.0  Infinity NaN(not a number) |
| String | “Google” ‘JavaScript’ `Maria` “098-97.09” |
| Boolean | True e False |
| Null | Valor vazio |
| Undefined | Quando não tá recebendo nenhum valor |
| Object | [ ] { } Array(vetor) |
| Function | Funções |

## Uso de caracteres especiais em string

|  |  |
| --- | --- |
| \0 | Byte nulo |
| \b | Backspace |
| \f | Alimentador de formulário |
| \n | Nova linha |
| \” | Aspas duplas |
| \’ | Apóstrofo ou aspas simples |

Array – serve para fazer uma lista Aula html > udemy > array.js

*const* times = ['Flamengo', 'Coxa', 'Botafogo', 'Atlético', 'Cruzeiro'];

console.log('Acessando os times: ', times) //mostrar em lista

console.log('O segundo time é: ', times [1]) //mostrar o segundo time

console.log('Sao',times.length, 'arrays.') // tem 5

console.table(times) //mostrar em tabela

//Operaçoes com o array

times.unshift('Corinthians') //add no começo do array

times.push('USA') //add no final do array

console.log('\nAdicionando mais dois times no inicio e no final:')

console.table(times)

console.log('Agora o tamanho da array tem:' ,times.length , 'times.') // tem 7

times.shift() //remove o primeiro elemento

times.pop() //remove o ultimo elemento

console.log('\nRemovendo o primeiro e o ultimo:')

console.table(times)

console.log('\nAdicionando um novo time na posição 2:')

times.splice(2, 1, 'Palmeiras') //add um elemento em uma posição específica, e removendo o antigo. 1º coloca o indice q quer remover, e quantos elementos quer remover(ate 2), e add ate 2

console.table(times)

console.log('\nOutro exemplo, adicionando 2 times na posição 1')

times.splice(1, 1, 'USA', 'Botafogo') //outro exemplo, removendo 1, add 2

console.log(times)

[Video explicando sobre o splice( )](https://www.google.com/search?q=usando+o+splice%28%29+no+javascript&sca_esv=2ee0018a8b8908ea&sca_upv=1&sxsrf=ACQVn08MNfXSHjLs3_sCgFh6fJxHR7xg4Q%3A1712942062144&ei=7msZZvi1CKyChuMP7MykKA&udm=&oq=usando+o+splice%28%29&gs_lp=-LAQ&sclient=gws-wiz-serp#fpstate=ive&vld=cid:83e966bc,vid:SyuCjQCn05U,st:0)

## Conversão de string para numero: (parse-conversao) Aula06-Ex003

* Number.parseInt(n)
* Number.parseFloat(n)
* Number(n)

<script>

        var n1 = Number.parseFloat(window.prompt("Digite um numero:"))

        var n2 = Number(window.prompt("Digite outro numero: "))

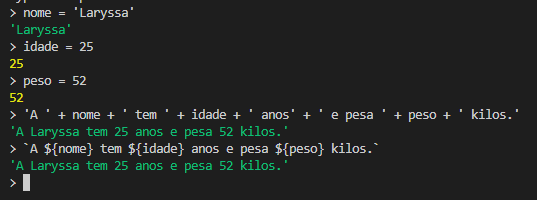
        var s = n1 + n2

        window.alert('A soma dos dois numero é ' + s)

</script>

#### Conversão de numero para string:

* Sting(n)
* n.toSting( )

Template de String – Formatação de strings – utilizando `crases` e ${n} – no node

## Aula06-Ex004:

* s.length : quantos caracteres a string tem
* s.toUpperCase() : tudo em letra ‘MAIUSCULAS’
* s.toLowerCas() : tudo em letra ‘minusculas’

<script>

        var nome = window.prompt("Qual é oseu nome? ")

        document.write(`Ola ${nome}. Seu nome ${nome.length} tem letras. <br>`)

        document.write(`Seu nome em maiusculas fica ${nome.toUpperCase()} <br>`)

        document.write(`Seu nome em minusculas fica ${nome. toLowerCase()}`)

</script>

#### Formatação de números – no node

* n1.toFixed(2) : para colocar duas casas depois do ponto
* n1.toFixed(2).replace(‘.’ , ‘,’) : para substituir oponto por virgula
* n1.toLocalesString(‘pt=BR’, {style: ‘currency’,

currency: ‘BRL’}) : colocando R$

* n1.toLocalesString(‘pt=BR’, {style: ‘currency’,

currency: ‘USD’}) : colocando US$

* n1.toLocalesString(‘pt=BR’, {style: ‘currency’,

currency: ‘EUR’}) : colocando €

## **Operadores de Java Script**

|  |  |  |
| --- | --- | --- |
| * Aritméticos     Ordem de precedência:  ( )  \*\*  \* / %  + -  .  5 % 2 = é o resto da divisão  .  5 \*\* 2 = é a potencia | * Atribuição     Atribuiçao simples <- | |
| * Relacionais     Identidade:  .  5 é igual a 5  5 é iguala ‘5’  5 não é idêntico a ‘5’  5 é idêntico a 5 | * Lógicos     && significa “e”  || significa “ou”   * Ordem de precedência:   1º Aritméticos  2º Relacionais  3º Lógicos | |
| * Ternário     O teste logico que pode dar verdadeiro ou falso    Vai aparecer aprovado ou Reprovado dependendo do teste  Ex1:  A média é 5.5  A média é maior que 7?  Se sim, Aprovado, se não Reprovado  Ex2:  X é 8  8 % 2 == 0  Resto de 8 dividido por 2 é 0 então 0 é igual a 0  Se a resposta for True = 5 se for False = 9  Res = 5  Ex3: | | |

## **DOM document Object Model**

Aula 9 ex 05:

Arvore do DOM

3 dos objetos:

Location : URL

Document: o doc atual

History: de onde vim p onde vou

**Ver mais sobre na Aula 9 – Introdução ao DOM**

* Por marca

|  |
| --- |
| <script>  *var* p1 = window.document.getElementsByTagName('p')[1] //seleciona o 1º paragrafo          window.document.write("<hr>Esta escrito assim: &nbsp" + p1.innerText)  //innerHTML: mostra com as formataçoes  </script> |

* Por ID : colocando: id=”msg” la na <div>

|  |
| --- |
| <script>  *var* d = window.document.getElementById('msg')          window.document.write('<hr>' + d.innerHTML)          d.style.backgroundColor = 'green'          d.style.color = 'white'          d.style.fontSize = '2em'          d.innerText = 'Mudando o texto da div com ID'  </script> |

* Por nome colocando: name=”msg” la na <div>

|  |
| --- |
| <script>  *var* d = window.document.getElementsByName('msg')[0]       d.style.textAlign = 'left'       d.style.color = 'green'       d.innerText = 'Agora com o Name, deixando na esquerda'  </script> |

* Por classe colocando: class=”msg” la na <div>

|  |
| --- |
| <script>  *var* d = window.document.getElementsByClassName('msg')[0]          d.style.background = 'orange'          d.style.color = 'blue'          d.style.fontSize = '2em'          d.innerText = 'Mudando agora com o Class'  </script> |

* Por seletor colocando por id=”msg” la na <div>

|  |
| --- |
| <script>  *var* d = window.document.querySelector('div#msg')          d.style.color = 'blue'          d.innerText = 'Mudando agora por QuerySelector'  </script>  Por class=”msg”  Apenas mudar no var no <div>:  ('div.msg')  querySelector : singular  querySelectorAll : plural |

## **Eventos DOM – o que pode acontecer com as tags Aula 10 ex 06 e udemy > exerciciosSimples**

* Mouseenter – entrar com o mouse em cima da tag
* Mouseout – tirar o mouse de cima
* Mousemove – mexer o mouse dentro da tag
* Mousedown – pressionar o mouse
* Mouseup – soltar o mouse
* Click – clicar em cima

|  |
| --- |
| <div id="area" onmouseenter="entrar()" onmouseout="sair()">          Interaja...  </div>  <script>  *var* a = window.document.getElementById('area') //colocando p fora, serve para todos          a.addEventListener('click', clicar)          a.addEventListener('mouseenter', entrar)          a.addEventListener('mouseout', sair)    *function* clicar() {              a.innerText = 'Voce clicou!!'              a.style.background = 'green'              a.style.color = 'yellow'          }  *function* entrar() {              a.innerText = 'entrou'          }  *function* sair() {              a.innerText = 'saiu'              a.style.background = 'purple'          }  </script> |

## Método **addEventListener Aula 10 ex 06**

Na div pode tirar tudo aquilo pra não ficar poluído, e adicionar no script

a.addEventListener('click', clicar)       //addEventListener recebe 2 parametros:

a.addEventListener('mouseenter', entrar)  //o evento e a function q vai executar

a.addEventListener('mouseout', sair)

## **udemy > exerciciosSimples:**

<form id="form"> //primeiro exemplo

            <label>Informe sua idade: </label>

            <input type="number" id="age">

            <br><br>

            <button type="submit">Enviar</button> //colocando botao tipo submit, para funcionar

            <button id="btn">Diga ola</button> //segundo exemplo

            <script>

*const* form = document.getElementById("form")

              form.addEventListener('submit', enviar) //o evento e a function q vai executar

*function* enviar(*e*){

*e*.preventDefault() //para nao carregar a pagina sozinho

*const* idade = document.getElementById('age')

                    if (idade.value < 18){

                        alert('Voce é menor de idade')

                    }else if(idade.value >= 18){

                        alert('Voce é maior de idade')

                    }else if(idade.value > 60){

                        alert('Voce é um velhote')

                    }}

            </script>

</form>

Mais tipos acessa 🡪 <https://developer.mozilla.org/en-US/docs/Web/Events>

## Nova função

preventDefault()

Serve para nao carregar a página automaticamente

Cancela o comportamento que o elemento geralmente tem na página, então se o comportamento padrão de um link é abrir um site, nós vamos cancelar isso.

## **Condições**

* Condições Simples aula11 ex08

|  |
| --- |
| var velocidade = 80.5  console.log (`A velocidade do seu carro é ${velocidade}`)  if (velocidade > 70){      console.log (`Voce ultrapassou o limite de velocidade! Multado!`)  }  console.log(`Use sempre o cinto de segurança!`) |

* Condições Compostas aula11 ex09

|  |
| --- |
| aula11 ex09  var pais = "EUA"  console.log(`Vivendo em ${pais}`)  if (pais == "França"){      console.log('Voce é Brasileiro');  }else{      console.log('Voce é Estrangeiro')  }  aula11 ex10 em html  <body>      <h1>Paises</h1>      Em que país voce vive? <input type="text" name="nome" id="txtnome" >      <input type="button" value="Verificar" onclick="verificar()">      <div id="vive"></div>      <script>          function verificar(){              var pais = window.document.getElementById('txtnome')              var vive = window.document.querySelector('div#vive')              var nac = String(pais.value)              vive.innerHTML = `Vivendo em ${nac}`              if (nac == 'Brasil' || nac == 'brasil'){                  vive.innerHTML += "<p>Voce é Brasileiro.</p>"              }else{                  vive.innerHTML += "<p>Voce é Estrangeiro.</p>"              }          }      </script>  </body> |

## Elemento: propriedade: ClassList aula11 ex10.html

Usar classList: uma alternativa conveniente para acessar a lista de classes de um elemento como uma string delimitada por espaço via [element.className](https://developer.mozilla.org/en-US/docs/Web/API/Element/className).

Embora a classListpropriedade em si seja somente leitura, você pode modificá-la DOMTokenListusando os métodos [add()](https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList/add), [remove()](https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList/remove), [replace()](https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList/replace)e [toggle()](https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList/toggle).

<style>

        .blue{

*color*: blue; }

        .red{

*color*: red; }

</style>

<body>

<p id="text" class="texto">

        Lorem ipsum dolor sit amet consectetur adipisicing elit. Natus quis nobis.

</p>

<script>

        const texto = document.getElementById('text')

        texto.classList.add('blue') 

texto.classList.remove('texto') //vai remover a claas texto

      texto.classList.remove('blue') //vai tirar a cor azul

        texto.classList.add('red') // adicionando a cor vermelha



</script>

</body>

* Condições. Aninhadas

|  |
| --- |
| Aula12 ex12  var idade = 89  console.log(`Sua idade é ${idade}`)  if (idade < 16){      console.log('Voce NÂO VOTA.')  }else{      if (idade < 18 || idade >= 60){          console.log("Votação OPCIONAL")}      else{          console.log("Voto OBRIGATÓRIO.")} }  Aula12 ex13  var agora = new Date //para mostrar as hora  var hora = agora.getHours() //exata q estiver executando  console.log(`Agora sao ${hora} horas`)  if (hora < 12){      console.log("Bom diaaa")  }else if (hora <= 17){      console.log("Boa tardee")  }else{      console.log("Boa noiteee") } |

* Condições Múltiplas

|  |
| --- |
| Obrigatório o break  var agora = new Date() // dia atual  var diaSem = agora.getDay()  switch(diaSem) {      case 0:          console.log("Domingo")          break      case 1:          console.log("Segunda")          break        case 2:          console.log("Terça")          break      case 3:          console.log("Quarta")          break      case 4:          console.log("Quinta")          break      case 5:          console.log("Sexta")          break      case 6:          console.log("Sabado")          break  } |

Exercício de idades - aula12ex1

Para carregar o ano atual:

var data = new.data()

var ano = data.getFullYear() // ano atual

Exercício do horário - aula12ex2 horario.html

Dicas:

Para adicionar imagens no HTML pelo JS:

<body onload="carregar()">

    <section>

        <div id="msg">

            Aqui vai aparecera a hora

        </div>

    <script src="script.js"></script> <!--script separado tb-->

</body>

<script> img.src= ("fotos/manha.png")</script>

ou

<script> foto.innerHTML= '<img src="fotos/manha.png">'</script>

Para carregar a hora atual:

var agora = new Date //para mostrar as hora

var hora = agora.getHours() //exata q estiver executando

Para mostrar o dia atual:

var agora = new Date() // dia atual

var diaSem = agora.getDay()

Para que o código JS execute assim que a página é carregada, usa-se Onload no HTML:

onload=" ()"

aula12ex2 fotonoJS.html

Dicas:

Para adicionar imagens e titulo pelo JS :

<body>

    <div id="conteiner"></div>

    <script>

*let* conteiner = document.getElementById('conteiner')

*let* manha = document.createElement('img') //criando um elemento img

*let* titulo = document.createElement('h1') //criando um elemento h1

        titulo.innerHTML = 'Imagem de manhã' //add o caminho para o titulo

        manha.src = './fotos/manha.png' //add o caminho para a foto

        conteiner.appendChild(titulo) //add no html, para aparecer o titulo no site

        conteiner.appendChild(manha)  //add no html, para aparecer a imagem no site

        manha.style.display = 'flex'

        manha.style.width = '400px'

        manha.style.height = '400px'

        //conteiner.removeChild(titulo) //removendo do html

    </script>

</body>

Atributos aula12ex2 atributos.html

<body>

    <img src="fotos/noite.png" alt="noite" width="400px" id="imagem">

    <script>

*const* img = document.getElementById('imagem')

*const* atributo = img.getAttribute('alt') //pegar o valor do atributo



        console.log(atributo) //mostrando no console ==

img.setAttribute('width', '500px') //modifica o valor de um atributo existente num elemento específico

        img.removeAttribute('width') //vai remover o atributo escolhido

    </script>

</body>

## Evento **onSubmit** udemy > exercicios simples

É um evento que acontece quando tenta "submter" o formulário, ou seja, por exemplo quando clica num botão submit . Então você pode executar uma ou mais functions, pode fazer todas as validações que deseja.

<body>

    <main>

        <form onsubmit="enviar()">

            <label>Informe sua idade: </label>

            <input type="number" id="age">

            <br><br>

            <button type="submit">Enviar</button> //tem que ter um botao tipo submit, para funcionar

            <script>

*function* enviar(){

*const* idade = document.getElementById('age')

                    if (idade.value < 18){

                        alert('Voce é menor de idade')

                    }else if(idade.value >= 18){

                        alert('Voce é maior de idade')

                    }else if(idade.value > 60){

                        alert('Voce é um velhote')

                    }

                }

            </script>

        </form>

    </main>

</body>

## Evento **onLoad** aula12ex2 horario.html

Para que o código JS execute assim que a página é carregada, usa-se Onload no HTML:

onload=" ()"

## Colocando o nome do function do JS

function carregar()

Foi colocado no body: HTML:

<body onload="carregar()">

    <header>

        <h1>Hora do Dia</h1>

    </header>

    <section>

        <div id="msg">

            Aqui vai aparecera a hora

        </div>

        <div id="imagem">

            <img src="fotos/manha.png" alt="foto do dia">

        </div>

    </section>

    <footer>

        <p>&copy;Curso em Video</p>

    </footer>

    <script src="script.js"></script> <!--script separado tb-->

</body>

SCRIPT.JS :

function carregar() { //para carregar, coloquei no html(body)

    var msg = window.document.getElementById('msg')

    var foto = document.getElementById('imagem')

    var data = new Date()  //data atual

    var hora = data.getHours() //hora atual

    //var hora = 8

    if (hora >= 7 && hora < 12){

        msg.innerText = `Bom dia! Agora sao ${hora} horas.`

        img.src= ("fotos/manha.png")

        document.body.style.background = '#FEDC5A'

    } else if (hora >= 12 && hora <= 18){

        msg.innerHTML = `Boa tarde! Agora sao ${hora} horas.`

        foto.innerHTML= '<img src="fotos/tarde.png">'

        document.body.style.background = '#f59629'

    } else{

        msg.innerHTML = `Boa noite! Agora sao ${hora} horas.`

        foto.innerHTML= '<img src="fotos/noite.png">'

        document.body.style.background = '#1e4c5c'

    }

}

### **Repetições**

* Estrutura de while Aula13

|  |
| --- |
| .  //esse faz o teste antes e executa o comando  var c = 1  while (c <= 6) { //enquanto o contador der 6..      console.log(`Passo ${c}`)      c++ //c = c + 1  } |

* Estrutura Do while Aula13

|  |
| --- |
| .  //esse executa o comando e  depois faz o teste  var s = 1  do {      console.log(`Passo ${s}`)      c++  } while (s <= 6) |

* ForEach Aula13 🡪 tem mais 2 exercicios la no VsCode

|  |
| --- |
| //repetição com array  *const* cidades = ['São Paulo', 'Rio de Janeiro', 'Curitiba', 'Joinville']    cidades.forEach( (*elemento*) *=>* { //add uma variavele      console.log(*elemento*)        //e uma função anônima 🡪  })    //mostrando o indice junto  cidades.forEach( (*elemento*, *index*) *=>* { 🡪      console.log('Executando o:', *index*)      console.log(*elemento*)  })  console.log('') |

* Estrutura de for Aula14

|  |
| --- |
| /\*var c = 1  while(c <= 5) {      console.log(c)      c++  }\*/  for (var c = 1 ; c<=10; c++){      console.log(c) } |

Aula 14 ex 2

function tabuada() { //para carregar, coloquei no html

    var num = document.getElementById('txtn')

    var tab = document.getElementById('seltab')

    if (num.value.length == 0) {

        window.alert('Por favor, digite um numero!')

    } else {

        var n = Number(num.value)

        var c = 1

        tab.innerHTML = '' //antes de mostrar a tabuada, limpe a tabela antes

        while (c <= 10){

            var item = document.createElement('option') //criar um elemento de opçoes

            item.text = `${n} x ${c} = ${n\*c}`

            tab.appendChild(item) //adicionar os itens

            c++

        }    }}

## **Aula 15 – Variáveis Compostas usando colchetes []**

Uma variável composta tem vários elementos, cada elemento é composto por um valor e por uma chave de identificação.



let num = [8, 4, 3, 1, 9]

num [3] = 6 //adicionar na posição 3 o nº 6

num.push(7) //adicionar no final

console.log(num)

console.log(`Nosso vetor é ${num}`)

console.log(`O vetor tem ${num.length} elementos`) //para ver quantos elementos tem

console.log(`Deixando em ordem crescente ${num.sort()}`)

console.log(`O primeiro valor agora é ${num[0]}`)

/\*for (let pos = 0; pos < num.length; pos++){ //enquanto ele n chegar no final do vetor..

    console.log(`A posição ${pos} = ${num[pos]}`)

}\*/

// Mais simplificado:

for(let pos in num) { //para cada posição dentro de num

    console.log(`A posição ${pos} tem valor ${num[pos]}`)

}

var pos = num.indexOf(7) //vai mostrar em que posição esta

console.log(`O valor 7 esta na posição ${pos}`)

## **Aula 16 – Funções**

* São ações executadas assim que são chamadas ou em decorrência de algum evento.
* Uma função pode receber parâmetros e pode retornar um resultado.

function parimp(n) {

    if (n % 2 == 0) {

        return `O numero ${n} é PAR` EX1

    } else {

        return `O numero ${n} é IMPAR`

    }

}

let res = parimp(1127)

console.log(res)

*function* soma(*n1*=0, *n2*=0) { //se o n1 ou n2 n for chamado é = 0

    return *n1*+*n2* EX2

}

console.log(soma(4))

let v = function(x) { //uma variavel recebe uma função EX3

    return x\*2 //o dobro

}

console.log('O dobro de 5 é', v(5))

console.log('O dobro de 7 é', v(7))

function fatorial(n){

    let fat = 1

    for (let c = n; c > 1; c--) {

        fat \*= c EX4

    }

    return fat

}

console.log(fatorial(5))

let msg = ('O fatorial de 5 é '+ fatorial(5))

console.log(msg)

//RECURSIVA

*function* fatorial(*n*) {

    if (*n*==1) {

        return 1

    } else {

        return *n* \* fatorial(*n*-1)

    }

}

console.log(fatorial(5))

## Colocando tabela pelo JS e aparecer no HTML **Ex: Documents\Projetos HTML e CSS\Musculacao**

var container = document.getElementById("container");

container.innerHTML = [

'<table>',

'<thead>',

'<tr>',

'<th>id</th>',

'<th>col1</th>',

'<th>col2</th>',

'<th>col3</th>',

'</tr>',

'</thead>',

'<tbody>',

'<tr>',

'<td>1</td>',

'<td>data</td>',

'<td>data</td>',

'<td>data</td>',

'</tr>',

'<tr>',

'<td>2</td>',

'<td>data</td>',

'<td>data</td>',

'<td>data</td>',

'</tr>',

'<tr>',

'<td>3</td>',

'<td>data</td>',

'<td>data</td>',

'<td>data</td>',

'</tr>',

'</tbody>',

'</table>'

].join("\n");

### [Utilizando metodos tostring e join em listas](https://www.dio.me/articles/utilizando-metodos-tostring-e-join-em-listas-com-javascript)

## Manipulando Tabelas com Javascript

<https://www.youtube.com/watch?v=ImkWbhXnISM>

Onde o usuário insere os dados solicitados no

formulário da pagina em HTML. Mostrando na tabela

quando clicar em adicionar.

## Busca em tabela com Javascript

<https://www.youtube.com/watch?v=HWZfqk6gSfI>

Onde o usuário quer fazer uma buscar de nome

na tabela

## [Clicar no botão que leva para uma outra parte da mesma página](https://pt.stackoverflow.com/questions/175871/click-no-link-que-leva-para-uma-outra-parte-da-mesma-pagina) 🡪 Projeto Meu\_site\_de\_sites

Usa href="#idDoElement" com o elemento da ID. Assim quando clicares na ancora a página muda o scroll para mostrar esse elemento.

<a href="#texto5"><h1>Serviços</h1></a>

<a href="#contatos"><h1>Contatos</h1></a>

Com scroll suave:

Colocando:  scroll-behavior: smooth no html

html{

*scroll-behavior*: smooth; /\*rolagem suave\*/

}

Só fica uma pequena questão: a tela vai rolar para o exato ponto em que o título (ou o conteúdo que seja) começa sendo bastante comum o “corte” do texto desse título — o que não é bom para a experiência de quem acessa o site.

Felizmente, agora é possível contar com a propriedade scroll-margin-top.

html{

*scroll-behavior*: smooth;

}

:target {

*scroll-margin-top*: 5em;

}

main #responsivo:target{

*scroll-margin-top*: 10em;

}

Perceba que :target pode ser usado em elementos específicos, mas, assim como muitos outros pseudo em CSS, se usado “solto”, representa a estilização em tudo o que é afetado pelo estilo em questão.

## [Como adicionar em um botão para voltar ao topo da página](https://pt.stackoverflow.com/questions/245814/como-adicionar-em-um-bot%c3%a3o-a-anima%c3%a7%c3%a3o-para-voltar-ao-topo-da-p%c3%a1gina-usando-a-bib) 🡪 Projeto Vet São Lourenço

HTML:

<div class="scroll-top on">

        <a id="botao-top" onclick="scrollToTop()"><img src="imagens/setapcima.png"

alt=""></a>

</div>

CSS:

.scroll-top {

*position*: fixed;

*bottom*: 35px;

*right*: 30px;

}

#botao-top {

*cursor*: pointer;

}

O nome que eu coloquei na função foi ScrollToTop. No nosso JavaScript iremos fazer uma const para armazenar a nossa arrow function, dentro dela colocamos o método window.scrollTo() que vai fazer a rolagem para determinado ponto da página.

JS:

*const* scrollToTop = () *=>* {

    window.scrollTo({

   top: 0,

   behavior: 'smooth'

    })

}

Dentro do window.scrollTop() vamos passar, entre chaves, o ponto que no caso seria o topo da nossa página, o topo inicial de uma página é zero.

Por fim, adicionamos o comportamento dessa rolagem para que ela fique suave aos olhos do usuário.

## Colocando a opçao que fica “invisível” na tela e aparece apenas quando o usuário estiver rolando a página **ainda não achei o codigo certo**

## [Quando clicar no botão ir para fazer o cadastro](https://pt.stackoverflow.com/questions/464160/quando-clicar-no-botao-ir-para-fazer-o-cadastro) por JS

Tem 2formas:

1. Redirecionando com onclick= window.location

<div class="inscrever">

        <input class="bnt st-button" type="button" value ="Inscrever-se"/>

        <input class="bnt es-button" type="button" value ="Entrar">

</div>

obs: location é sinonimo de location.href

<input class="bnt st-button" type="button" value ="Inscrever-se" onclick="window.location='https://pt.stackoverflow.com/questions';" />

<input class="bnt st-button" type="button" value ="Entrar" onclick="window.location='https://pt.stackoverflow.com/';" />

1. função javascript

<input class="bnt st-button" type="button" value ="Inscrever-se" onclick="novaPagina(0)">

<input class="bnt es-button" type="button" value ="Entrar" onclick="novaPagina(1)">

*function* novaPagina(*num*) {

*var* url=new *Array*();

    url[0]="https://pt.stackoverflow.com/questions";

    url[1]="https://pt.stackoverflow.com/";

    window.location=url[*num*];

}

Usando Bootstrap

## Ex Documents\Faculdade TI\Pós Front-end\Meus\_exercicios\_da\_Facul\LinguagensEPadroesWeb\Tema4\Bloco3

[getbootstrap.com](https://getbootstrap.com/docs/4.0/getting-started/introduction/)

*let* input = document.querySelector('input[name=tarefa') /\*busque para mim um input que tem o nome tarefa \*/

*let* btn = document.querySelector('#botao');

*let* lista = document.querySelector('#lista')

*let* card = document.querySelector('.card')

*let* tarefas = JSON.parse(localStorage.getItem('tarefas')) || [] /\*buscar o local de armazenamento, ele vai buscar tds os elementos da tarefa, ou se ele nao encontrar o localStorage, vai me mostrar vazio\*/

/\*getItm: buscar a informação

  setItem: utilizada para armazenar um valor no local storage\*/

O código anterior cria as variáveis que serão utilizadas no decorrer do código. Temos os querySelector, que buscam elementos pelo texto inserido entre parênteses e como parâmetros. A variável que se difere das demais é a “tarefas”, que cria um vetor com as tarefas registradas. Note que o conceito de seletores em JavaScript são o que criam a integração entre os elementos HTML e as funções programadas em tal linguagem.

Na sequência, temos a função renderizarTarefas(), que cria uma lista de itens que serão exibidos. Essa lista receberá todas as tarefas inseridas. Temos, no corpo desta função, um for que percorre todas as tarefas, povoando a lista e criando-a.

*function* rendenizarTarefas(){

    lista.innerHTML = ''

    for(tarefa of tarefas){ /\*vai percorrer toda lista que tenho, criando uma lista\*/

*let* itemLista = document.createElement('li')

        itemLista.setAttribute('class', 'list-group-item list-group-item-action') /\*lista do bootStrap\*/

        itemLista.onclick = *function*(){

            deletarTarefa(this)

        }

        /\*a partir que eu excluir um item da lista, ele vaideslocar todo os valores q eu tinha armazenado p um nivel anterior, para nao ficar nenhum "buraco" no registro que eu removi\*/

*let* itemTexto = document.createElement(tarefa)

        itemLista.appendChild(itemTexto) /\*insirir um texto na minha lista\*/

        lista.appendChild(itemLista)

    }

}

rendenizarTarefas();

A seguir, temos a variável btn, que referencia o botão e que, no momento em que este for “clicado” pelo usuário, ativará a função que, por meio de uma estrutura condicional if, verifica se a novaTarefa não está vazia e adiciona no LocalStorage, por meio da função salvarDadosNoStorage().

A chamada da função removerSpans() cria um alerta com possíveis problemas, caso o texto da tarefa esteja vazio, exibindo ainda uma mensagem para que o usuário entre com tal texto.

btn.onclick = *function*(){ /\*coloquei a função direta\*/

*let* novaTarefa = input.value; /\*pega o valor do input, joga para uma variavellocal, para verificar \*/

    if(novaTarefa !== ""){ /\*caso nao esteja vazio\*/

        tarefas.push(novaTarefa)/\*vai inserir tds as tarefas\*/

        rendenizarTarefas() /\*função para mostrar as tarefas\*/

        input.value = '';

        removerSpans() /\*função p remover os espaços\*/

        salvarDadosNoStorage() /\*função p salvar os dados\*/

    }else{ /\*se tiver tarefas\*/

        removerSpans()

*let* span = document.createElement('span')

        span.setAttribute('class', 'alert alert-warning')/\*alerta do bootStrap\*/

*let* msg = document.createTextNode('Voce precisa digitar a terafa que deseja registrar!')

        /\*caso o usuario nao digitar nenhuma tarefa, vai mostrar a mensagem de alerta\*/

        span.appendChild(msg)/\*vai inserir a msg dentro do span\*/

        card.appendChild(span)/\*dps vai inserir o span dentro do card\*/

    }

}

*function* removerSpans(){

*let* spans = document.querySelectorAll('span')

    for(*let* i = 0; i < spans.length; i++){

        card.removeChild(spans[i])

    }

}

A função deletarTarefa() é adicionada à tarefa que foi acrescentada, por meio da função renderizarTarefas(). Faz com que a tarefa da lista, ao ser “clicada” pelo usuário, seja excluída.

*function* deletarTarefa(*tar*){

    tarefas.splice(tarefas.indexOf(*tar*.textContent), 1) /\*splice: busca um indice de um determinado conteudo, e remove aquele conteudo\*/

    /\*indexOf(): utilizado para retornar o índice da primeira ocorrência

    de uma determinada palavra ou caractere em uma string,

    lembrando que a contagem das posições começa sempre do zero.\*/

    rendenizarTarefas()

    salvarDadosNoStorage()

}

Finalmente, a última função coloca, no localStorage, a tarefa digitada pelo usuário. O localStorage, ou armazenamento na web, também conhecido como armazenamento DOM, fornece ao desenvolvedor a possibilidade de armazenar dados no lado do cliente. É como se tivéssemos um banco de dados local, inserido no próprio navegador, que mantém as informações armazenadas até serem totalmente removidas pelo usuário. Assim, já temos um projeto inicial que trabalha, inclusive, com o localStorage.

*function* salvarDadosNoStorage(){

    localStorage.setItem('tarefas', JSON.stringify(tarefas))

    /\*setItem: utilizada para armazenar um valor no local storage\*/

}

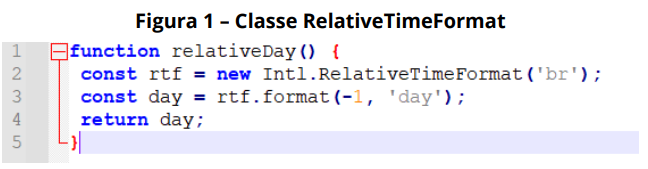
O framework do Bootstrap adiciona uma folha de estilos CSS específica, com suas próprias classes. Com sua documentação e a comunidade que a mantém atualizada, temos uma ótima alternativa de estilização sem muitos esforços,. bastando apenas, sabermos as classes a serem utilizadas, o que pode ser consultado diretamente na documentação do Bootstrap (BOOTSTRAP, 2021).

Agora, para finalizar o projeto, utilize seus conhecimentos para alterar as páginas e também ajustar a integração da folha de estilos criada para a página inicial com o CSS do Bootstrap.

# Novas Classes (partir da facul)

Podemos utilizar a API Intl para realizar a tradução de datas, horas e linguagem de um determinado site. Para realizar está tradução, está API utiliza as classes:

* **DateTimeFormat** (responsável por traduzir datas): Você pode formatar uma data para ser visualizada no padrão português Brasil, em que teríamos o formato DD/ MM/AAA, em que DD é o dia, MM é mês e AAA é o ano, por exemplo (05/03/2022). Outra combinação possível é obter o padrão DD/MM HH/ MIN (dia, mês, hora e minuto), em que o formato numérico será de dois dígitos, por exemplo, 05/08 09:36.
* **NumberFormat** (responsável por traduzir números): é muito semelhante à classe DateTimeFormat. Ela é utilizada para formatar números conforme sua localização geográfica, e ainda consegue distinguir e identificar sinais, números decimais e valores monetários. A ideia da classe NumberFormat é criar uma constante que armazenará a localidade de um determinado idioma. Ao criar a constante e guardar a localidade o valor será formatado.
* **RelativeTimeFormat** (responsável por traduzir intervalos de tempo): pode ser utilizada para formatar qualquer período relativo, como um dia atrás, duas semanas atrás etc. A Figura 1 exemplifica o comportamento da classe



 <script>

        //DateTimeFormat

*const* data = new *Date*('2024-20-01');

        console.log(new Intl.*DateTimeFormat*('pt-BR', {

            year: 'numeric',

            month: 'long',

            day: 'numeric'}).format(data))

        console.log(new Intl.*DateTimeFormat*('pt-BR', {timeZone: 'UTC'}).format(data))

*const* d2 = new Intl.*DateTimeFormat*('en-US',{

            year: 'numeric',

            month: 'long',

            day: 'numeric'})

        console.log(d2.format(data))

        //NumberFormat

*const* n = 100000

        //para converter em Reais: R$ 100.000,00

        console.log(new Intl.*NumberFormat*('pt-BR', {style: 'currency', currency: 'BRL'}).format(n));

        //para converter em Euros: 100.000,00 €

        console.log(new Intl.*NumberFormat*('de-DE', { style:'currency', currency: 'EUR' }).format(n));

        //para converter em Yenes: ￥100,000

        console.log(new Intl.*NumberFormat*('ja-JP', {style: 'currency', currency: 'JPY'}).format(n));

        //RelativeTimeFormat

*const* df = new Intl.*RelativeTimeFormat*('pt')

        console.log(df.format(-1, 'day'))

        console.log(df.format(-1, 'month'))

        console.log(df.format(-1, 'year'))

        console.log(df.format(5, 'day'))

        console.log(df.format(5, 'month'))

        console.log(df.format(5, 'year'))

</script>

* **Promises**: representa a eventual conclusão (ou falha) de uma operação assíncrona e seu valor resultante. Além dos métodos estáticos, uma promise pode ter métodos dinâmicos. São o método **then**, que verificam qual bloco que será executado de acordo com o cumprimento da promise e o método catch que verifica qual bloco será executado caso a promise seja rejeitada.

Uma Promise está em um destes estados:

* + *pending*: estado inicial, nem cumprido nem rejeitado.
  + *fulfilled*: significa que a operação foi concluída com sucesso.
  + *rejected*: significa que a operação falhou.

<script>

*const* timeout = (*duration*) *=>* {

            return new *Promise*((resolve, reject) *=>* {

                setTimeout(resolve, *duration*)

            })

        }

        timeout(5000)

            .then(*function*(){

                console.log('Passou 5 segundos')

            })

</script>

* **Iterators:** é um objeto que oferece o método next() , o qual retorna o próximo item da sequência. O objetivo deste método é definir como o iterator pode ser percorrido. Para descobrir de que forma o iterator é percorrido, o protocolo retorna um objeto que possui um método chamado next. Por sua vez, o método next retorna outro objeto que armazena uma chave conhecida como done. Está chave possui um valor que pode ser verdadeiro ou falso. Dessa forma, será indicado se a iteração chegou ao fim ou não.

<script>

        *const* makeInterator = (*array*) *=>* {

*let* nextIndex = 0;

            return {

                next: () *=>* {

                    return nextIndex < *array*.length ?

                    {value: *array*[nextIndex ++], done: false} : {done: true};

                }

            }

        };

*let* it = makeInterator(['yo', 'ya']);

        console.log(it.next().value); //'yo'

        console.log(it.next().value); //'ya'

        console.log(it.next().done); //'true'

</script>

* **Generators:** Um generators está diretamente relacionado a um iterators, isso porque um generators sempre retorna um iterators. Um generators é uma função que permite a definição de um algoritmo, mantendo seu estado próprio
* **Async/Await:** Async é uma função assíncrona, implementadas da 8ª edição da linguagem JavaScript. Por meio desta função, podemos utilizar uma palavra-chave chamada de await. A função async, aliada com a chave await, são implementados de acordo com uma classe promises. Segundo Hanashiro (2018, [s.p.]): “Com async e await podemos trabalhar com código assíncrono em um estilo mais parecido com o bom e velho código síncrono.” Sendo assim, após ser executada, a função async será pausada até que uma promise seja executada. A função async e await facilitam a estrutura de código, no que diz respeito a seu fluxo e leitura.

**async** faz uma função retornar uma Promise

**await** faz uma função esperar por uma Promise

<script>

*function* myDisplayer(*some*) {

        document.getElementById("demo").innerHTML = *some*;

        }

        async *function* myFunction() { return "Hello"; }

        myFunction().then(

*function*(*value*) {myDisplayer(*value*);},

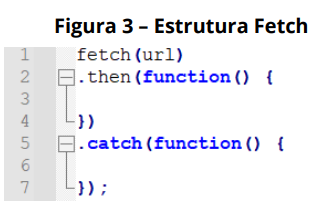
*function*(*error*) {myDisplayer(*error*);}

        );

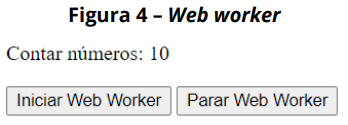
</script>

## Fetch, módulos, web Works, proxy e reflect

* **Fetch:** é um método utilizado em uma fetch API, outra novidade que surgiu na linguagem JavaScript após atualizações. Esta API é utilizada para realizar buscas de dados. A Fetch API invoca o método Fetch e este método aceita como parâmetro a URL da API para seu funcionamento. A Figura 3 apresenta a estrutura Fetch.

****

* **Módulos**: Módulos geralmente são blocos de códigos, que geralmente ficam em arquivos separados. Um módulo pode ser um bloco de código de funções, classes e até mesmos constantes. Sendo assim, toda vez que desejamos utilizar um módulo devemos importá-los. Duas das principais vantagens da criação de módulos são a facilidade da manutenção e reusabilidade de um projeto JavaScript.
* **Web worker:** O web worker é um script da linguagem JavaScript, executado em segundo plano, que faz com que a página responda somente quando o script seja finalizado. Em outras palavras, enquanto o script web worker é executado, você pode continuar realizando ações na página até escolher finalizar o script. A Figura 4 apresenta um exemplo de web worker.

****

* **Proxy**: oO proxy é um objeto marcador que é utilizado na linguagem JavaScript para definir diversos comportamentos. Esses comportamentos podem ser, por exemplo, de leitura e gravação de dados. Sua sintaxe é let proxy = new Proxy(target, handler). Esta sintaxe demonstra que o proxy utiliza os objetos target, que é um objeto utilizado para armazenar o proxy e encapsular algum módulo, por exemplo, funções. Já o objeto handler utiliza funções para configurar o proxy e define quando uma operação pode ser realizada.
* **Reflect**: o reflect é um objeto utilizado para facilitar a criação de um prozy. É um objeto interno que fornece métodos para operações JavaScript interceptáveis. Os métodos são os mesmos dos manipuladores de proxy. Reflect não é um objeto de função, portanto, não é construtível.

# Armazenamento de dados, frameworks e JavaScript fora de navegadores

Vários recursos podem ser utilizados para o armazenamento de dados e desenvolvimento de programas e aplicativos. É possível utilizar um conjunto de códigos protos para desenvolver programas com a linguagem JavaScript e utilizar o HTML5 para o armazenamento de dados. Neste tópico, você conhecerá sobre o armazenamento de dados em navegadores web desenvolvidos em HTML5, além de estudar sobre os frameworks utilizados em JavaScript e, ainda, de que forma o JavaScript é utilizado fora de navegadores.

* LocalStorage e SessionStorage.

O LocalStorage e SessionStorage são novidades do HTML5 e ambos permitem o armazenamento de texto em um navegador web, obedecendo um limite entre 5 MB e 10 MB, dependendo do navegador. O SessionStorage armazena dados, de forma temporária, nas abas do navegador e esses dados serão excluídos quando as abas forem fechadas. Já o LocalStorage também armazena dados nas abas dos navegadores, mas não possuem dada de expiração, ou seja, os dados são mantidos nos navegadores, mesmo que este seja fechado. O localStorage não é muito utilizado atualmente, porque pode ser substituído pelo IndexedDB e possui suas limitações que são:

• Execução é síncrona.

• Realiza armazenamento de dados de até 5 MB.

• Não possui acesso por web workers.

• IndexDB

O IndexedDB é classificado como um sistema de banco de dados transacional, baseado em SQL. Dessa forma, este banco utiliza transações nos fluxos de mensagens e executa tarefas e resultados. Apesar de ser baseado em SQL, o IndexedDB é orientado a objetos, que utiliza a linguagem JavaScript.

* Frameworks:

São conjuntos de códigos prontos e que normalmente são usados no desenvolvimento de aplicativos e sites. Um framework implementa funcionalidades, comandos e até mesmo estruturas já prontas. Existem alguns frameworks que são desenvolvidos em linguagem JavaScript; o mais antigo e mais conhecido é o react. O react é uma ferramenta para o front, uma biblioteca que cria interfaces. Oferece facilidade na hora de ter um código limpo e reutilizável.

Um framework descreve a arquitetura de um sistema orientado a objetos, os tipos de objetos e as interações entre os mesmos. Ele pode ser vislumbrado como o esqueleto – template – de uma aplicação que pode ser customizado pelo programador e aplicado a um conjunto de aplicações de um mesmo domínio.

### JavaScript além de navegadores

O JavaScript é uma linguagem de programação muito utilizada por desenvolvedores para implementações web. É possível realizar diversas implementações, interações de usuários com páginas webs, utilizando a linguagem de programação JavaScript, mas é importante saber que, além da web, o JavaScript também é utilizado para o desenvolvimento de jogos, robótica, automação, e é utilizado em banco de dados e desenvolvimento de aplicativos.

O MongoDB é um banco de dados e possui uma ferramenta que usamos para acessar dados, executando JavaScript. Com JavaScript também podemos criar aplicativos mobile. Sendo assim, com a utilização do JavaScript, podemos desenvolver aplicativos para diversas plataformas. Neste tema você teve a oportunidade de conhecer um pouco sobre a internacionalização do JavaScript, estudou sobre as novidades do JavaScript, no que diz respeito às promisses, aos interators, generators, async e await, além de aprender sobre Requisições com Fetch, Módulos, web workers, Proxy e Reflect

# JAVASCRIPT E SUAS PROPRIEDADES

* JavaScript e node

O node, mais conhecido como node.js, é uma tecnologia muito utilizada para executar códigos JavaScript que foram desenvolvidos fora de um navegador web. Segundo Pereira (2016): O Node.js é uma plataforma altamente escalável e de baixo nível. Nele, você vai programar diretamente com diversos protocolos de rede e internet, ou utilizar bibliotecas que acessam diversos recursos do sistema operacional. Para programar em node.js basta dominar a linguagem javascript, isso mesmo, javascript! E o runtime javascript utilizado nesta plataforma é o famoso javascript V8, que é usado também no Google Chrome. (PEREIRA, 2016, p. 13) Dessa forma, com o node.js, é possível desenvolver diversas aplicações voltadas para web, microserviços, desenvolver implementações de APIs e, além disso, pode ser utilizado como um ambiente de armazenamento de projetos. O node.js possui um gerenciador de pacotes conhecido como node package manger (npm). Este gerenciador possui duas funções:

1. Atua como repositório, armazenando diversos projetos node.js. Este repositório funciona como uma plataforma on-line, e dessa forma, muitos desenvolvedores utilizam para publicar e compartilhar aplicativos desenvolvidos em JavaScript.

2. Atua como um ambiente de desenvolvimento de linhas de comando, auxiliando na instalação de vários pacotes, gerenciando e controlando versões e projetos. O npm interage com navegadores e servidores para que possa realizar esse controle e gerenciamento. Outro gerenciador de pacotes muito conhecido, assim como o npm, é o gerenciador yarn. Esse gerenciador de pacotes é instalado pelo ambiente de linhas de comandos do npm e realiza as mesmas ações que o npm, porém de forma mais rápida.

* ES6 O ES6 ou EcmaScript 6

É a sexta versão de padronização do JavaScript, ou seja, é uma versão do JavaScript que surgiu em 2015. Esta padronização tornou-se muito importante devido a diversas funcionalidades que ofereçam um maior dinamismo nos projetos desenvolvidos em JavaScript. Isso porque, além de funções simples, o EcmaScript 6 trouxe novas possibilidades de iteração com objetos e possibilidade de declarações de variáveis como os comandos let e const, além da modularização de classes. A importância do ES6 se dá pela possibilidade, pela primeira vez, de utilizar o JavaScript além das páginas web. Isso porque a partir desta padronização e com o auxílio de frameworks, o JavaScript passou a ser utilizado não mais só como front-end, mas como back-end. Dentre alguns recursos que o ES6 trouxe, podemos citar:

C:\Users\User\Documents\Faculdade TI\Pós Front-end\Meus\_exercicios\_da\_Facul\TecnicasAvançadasCssEJS\Tema4\propriedadesNodes.html

**Funções Arrow.** Função nova na linguagem JavaScript que dispensa a utilização da declaração return, porque esta função realiza um retorno deduzido de um determinado valor.

**Função Find.** Find é um método utilizado na linguagem JavaScript para realizar uma busca dentro de um array. Imagine, por exemplo, que você tem uma lista com informações de funcionários de uma determinada empresa e pretende buscar somente pelas informações de um usuário específico. <https://warcontent.com/array-find-javascript/>

**Função map.** É um método que executa função dentro de array, porém, não sobrescreve o array original. Dessa forma, este método retorna um novo array. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/map>

**Função Reduce.** É um método utilizado para reduzir um determinado array em um único resultado. Este método possui dois parâmetros: o primeiro é o próprio resultado da redução; já o segundo, mostra o índice de cada array.

**Função filter.** É um método utilizado para filtrar informações de um array. Para isso, este método utiliza uma condição que retorna um valor verdadeiro ou falso. Essa condição é aplicada em cada item de um array e para aquele em que a condição é satisfeita, um novo array é adicionado. Dessa forma este método retorna um objeto, um valor.

**Função Every.** O método every é muito parecido com o método filter. Ele utiliza uma condição, testa todos os elementos de um array, e retorna um valor verdadeiro ou falso. Porém, diferentemente da função filter, não retorna um valor ou um objeto, retorna somente um valor boolean.

**Função some.** O método some testa valores de um array e retorna um valor verdadeiro assim que um valor do array satisfaz a condição.

**Comandos Let e Const.** Os comandos Let e Const são bem simples de serem entendidos. Com o comando Let, é possível iniciar ou declarar uma variável no meio do código, mais conhecido como escopo do código. O problema disso é que mesmo declarada por meio do comando let, é possível que ocorra um retorno undefined (indefinido(a)). Para resolver este problema, utilizamos o comando const, que garante que a variável inicializada com o let assuma o valor determinado.

## Programação modular

A programação modular em JavaScript é muito importante e utilizada atualmente em projetos complexos, quando existe a necessidade do desenvolvimento de muitos códigos. Em projetos deste tipo, é muito comum a implementação de várias funcionalidades e, com isso, código f ica cada vez maior. É verdade que por muito tempo, a linguagem JavaScript existiu sem algum tipo de sintaxe modular, isso porque os projetos desenvolvidos em JavaScript eram bem simples, o que significa que não existia a necessidade de criar módulos para a resolução de problemas. Com o surgimento da necessidade de desenvolvimento de projetos complexos, houve a necessidade de criar alternativas e funcionalidades que dessem um suporte maior, resoluções para os problemas começaram a ser mais bem trabalhadas, e com isso, surgiu o JavaScript modular.

Um módulo, então, pode ser entendido como um pedaço de código, parte de um código ou até mesmo um script, que realizará uma ação. Um módulo pode ser visto como uma função, um código implementado para realizar um cálculo ou um código desenvolvido para interface de botões e formulários. Existem muitos exemplos de módulos JavaScript. Aqui, iremos conhecer três deles:

* **AMD –** Um dos sistemas de módulos mais antigos, implementado inicialmente pela biblioteca require.js. O módulo AMD é utilizado quando dois ou mais módulos são chamados ou requisitados de forma assíncrona para realizar uma ação. Por exemplo, em um determinado código é solicitado que o usuário insira 2 valores em um campo de um formulário. Em seguida os valores são armazenados em um módulo que realizará uma operação entre esses valores, por exemplo, uma soma.
* **CommonJS –** É o sistema de módulos criado para o servidor Node.js. Ele utiliza a função requere para importar módulos e para exportá-los, utiliza o module.export ou exports.
* **UMD –** Mais um sistema de módulos, sugerido como universal, compatível com AMD e CommonJS. Este módulo busca por um método de importar e exportar que funcione tanto em ambientes AMD e CommonJS. Sendo assim, este módulo realiza uma adaptação de módulos para cada caso.

# Programação funcional, assíncrona e prototype com JavaScript

Como você já sabe, a EcmaScript 6 trouxe muitas novidades e funcionalidades para a linguagem JavaScript. Neste tópico, você estudará sobre as características da programação funcional e porque a linguagem JavaScript é classifica como uma. Você também conhecerá sobre o JavaScript assíncrono, além de entender sobre a importância do prototype.

## Programação funcional

Antes de falarmos de JavaScript como programação funcional, precisamos entender o que é a programação funcional. Este tipo de programação pode ser entendido como o desenvolvimento de software que implementa e utiliza funções. Dessa forma, o estado de elementos não é compartilhado, os dados podem sofrer alterações e dizemos que é um tipo de programação declarativa.

A linguagem JavaScript é uma linguagem de programação funcional, isso porque esta linguagem é fundamentada em funções. O JavaScript utiliza classes, objetos, herança e outros conceitos. Todos eles são implementados com funções. Outro ponto positivo desta linguagem e que é uma característica da programação funcional, é que seus códigos não são executados em tempo de compilação, e sim em tempo de execução (runtime).

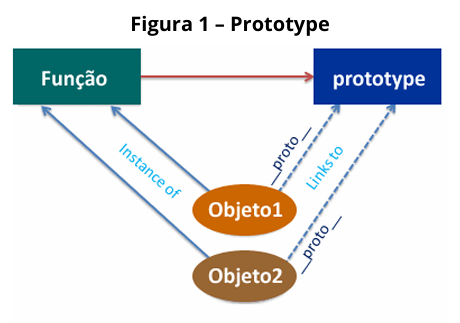
## Programação assíncrona

A linguagem de programação JavaScript é classificada como uma linguagem assíncrona, e isso significa que as instruções não possuem uma ordem definida para serem executadas. Dessa forma, a linguagem de programação JavaScript possui instruções complicadas de serem entendidas, mas fáceis de serem lidas. Isso garante que programas desenvolvidos com JavaScript tenham um melhor desempenho.

A linguagem JavaScript passou por diversas transformações com o passar do tempo. Dessa forma, operações assíncronas não eram tão comuns na web como são hoje. É por isso que atualmente, com a programação assíncrona, boa parte da lógica da aplicação desenvolvida com JavaScript é encontrada no front-and e não mais no seu backend.

## Prototype

Se pudéssemos definir um prototype, poderíamos dizer que basicamente ele é um objeto disponível para funções da linguagem JavaScript. A linguagem JavaScript é fundamentada em protótipos, e isso significa que esta linguagem utiliza uma cadeia de protótipos. Todas as propriedades, assim como os métodos de uma linguagem JavaScript, são definidas na propriedade prototy:



Exemplos: C:\Users\User\Documents\Faculdade TI\Pós Front-end\Meus\_exercicios\_da\_Facul\TecnicasAvançadasCssEJS

# APIs Rest e GraphQI

A linguagem JavaScript pode ser utilizada em diversos tipos de projetos de desenvolvimentos, e um desses projetos pode estar ligado ao desenvolvimento e consumo de APIs. Esta é mais uma forma de comprovar a utilização desta poderosa linguagem de programação. Neste tópico, você estudará sobe o APIs Rest e GraphQI, verificando conceitos importantes que envolvem JavaScript, API Rest e GraphQl.

## Consumo de APIs Rest

Precisamos relembrar alguns conceitos de APIs, APIs Rest e consumo de APIs antes de estudar o consumo de APIs Rest com JavaScript. Segundo o site RedHat:

Uma API permite que sua solução ou serviço se comunique com outros produtos e serviços sem precisar saber como eles foram implementados. Isso simplifica o desenvolvimento de aplicações, gerando economia de tempo e dinheiro. Ao desenvolver novas ferramentas e soluções (ou ao gerenciar aquelas já existentes), as APIs oferecem a flexibilidade necessária para simplificar o design, a administração e o uso, além de fornecer oportunidades de inovação.

Em outras palavras, a application programming interface (interface de programação de aplicativos), mas conhecido como API, é classificada como um conjunto de rotinas e padrões de programação que possuem o objetivo de acessar aplicativos de software ou plataformas baseadas na web. Já REST é uma arquitetura de API, que realiza as transações de inserção, atualização, exclusão e recuperação de informação na internet. Os sistemas baseados na arquitetura REST são conhecidos como RESTFull.

Por fim, temos o conceito de consumo de APIs. Consumir uma API está relacionado a utilizar pelo menos uma funcionalidade da aplicação em questão. Logo, o desenvolvedor que está diante de um projeto de construção de uma API ou webservice, deve entender que as funções de suas aplicações serão utilizadas, consumidas ou gastadas por alguém.

Então, agora sabemos que o consumo de APIs Rest é utilizar pelo menos uma funcionalidade de uma determinada aplicação. Podemos, ainda, consumir APIs com JavaScript, utilizando linhas de códigos em plataformas específicas e, também, utilizando uma das formas mais fáceis de consumo, que é por meio de um framework, sendo um dos mais conhecidos, o framework angular.

## GraphQl GraphQL

É uma linguagem de consulta utilizada para solicitar e extrair dados de clientes. O GraphQL é muito usado em servidores para interfaces de programação de aplicações (APIs). Segundo Ribeiro (2017):

GraphQL é uma linguagem de consulta de dados desenvolvida e usada pelo Facebook para realizar requisições e entregar informações para aplicações web e mobile desde 2012. GraphQL facilita o processo de entregar ao client apenas o que foi requisitado pelo mesmo e na ordem em que foi requisitado. (RIBEIRO, 2017, [s.p.]) Neste tema, você teve a oportunidade de conhecer um pouco sobre o aprimoramento do JavaScript com node, ES6 e programação modular. Você estudou sobre as características específicas da programação funcional, programação assíncrona e prototype com JavaScript, e, por fim, pôde conhecer as aplicabilidades do consumo de APIs Rest e GraphQL com JavaScript.

# Programação com Node.JS

Node.js é um ambiente de execução de JavaScript que permite executar código JavaScript fora do navegador, principalmente no lado do servidor. Ele é construído sobre o motor V8 do Google Chrome, o que garante alto desempenho na execução do código

## Scripts lado do servidor (server side)

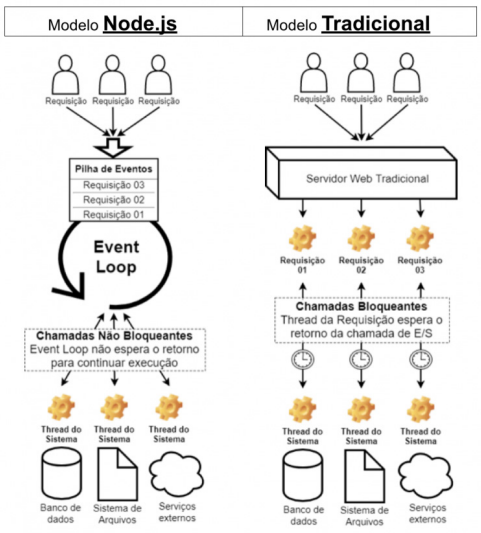
O script executado no servidor web, ou server side, é uma técnica de desenvolvimento que visa executar o software no servidor, proporcionando que algumas operações, como a personalização de uma página web, a alteração dinâmica no conteúdo do site, a geração de respostas às consultas e o acesso ao banco de dados, rodem no servidor. De forma resumida, é o que conhecemos como back-end

## Estrutura do Node.js

Em um servidor web utilizando, por exemplo, o PHP, para cada requisição recebida, é criada uma nova thread para tratar essa requisição. Logo, para cada requisição recebida, serão demandados recursos computacionais, como memória e CPU. Porém, se esses recursos forem limitados, as threads não serão muitas, já que elas também serão limitadas. Dessa forma, as novas requisições terão que esperar na fila de processos. Como no Node uma thread é responsável por tratar essas requisições, então ela recebe o nome de Event Loop, já que trata as requisições como eventos. Assim, ela fica sempre em execução, esperando novos eventos para tratar, e, para cada requisição, um novo evento é criado.

O fato de o Node ser single-thread não o torna menos eficiente se comparado a outras tecnologias, já que sua arquitetura faz com que um número maior de requisições concorrentes seja atendido. Em comparação às multi-threads, o Node se mostrou mais eficiente.

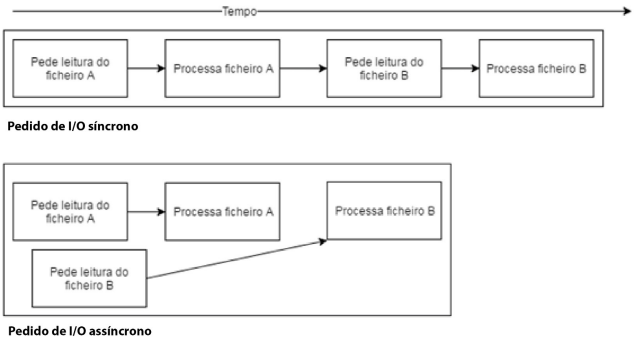
A Figura mostra a diferença entre o modelo tradicional e o Node.js:



# Blocking versus Non-blocking

Segundo o documento oficial do Node.js, a diferença entre chamadas com e sem bloqueio no Node.js está relacionada ao loop de eventos e libuv.

Para que você comece o estudo do padrão de chamada, é necessário abordar execução síncrona e assíncrona. A execução síncrona é quando o código roda em sequência. Já na programação assíncrona, o programa não espera que a tarefa seja concluída e pode passar para a próxima tarefa. No geral, uma operação assíncrona está relacionada com E/S, porque as operações de E/S são muito lentas e, portanto, se fossem síncronas, iriam bloquear a execução adicional do código.



O bloqueio irá acontecer sempre que a execução de JavaScript, ocorrendo pelo Node.js, tiver que esperar até que uma outra operação seja concluída. Isso acontece devido ao loop de eventos não poder continuar rodando o JavaScript durante a execução de uma AS, que é uma operação de bloqueio.

Um ponto relevante a ser observado é que todos os métodos de E/S na biblioteca do Node.js fornecem versões assíncronas, que por sua vez não são bloqueantes e aceitam funções de retorno de chamada. Outro ponto sobre blocking e non-blocking é que os métodos de bloqueio são executados de forma síncrona e os métodos sem bloqueio são executados de forma assíncrona.

## Single-thread

As aplicações do Node.js serão single-thread, mas o que isso significa? Entenda que uma thread é uma porção do software que trabalha como um subsistema, como se fosse uma parte do todo, ou seja, do processo que se divide em uma ou mais tarefas.

Assim, o Node.js utiliza o conceito de thread única para fazer a gestão de pilhas de eventos ou pilha de chamada (Call Stack), que por sua vez adota um comportamento do tipo LIFO (última entrada, primeira saída). Porém, existe uma discussão entre o Node ser single ou multi-thread. Isso se deve ao fato de as operações em background do Node serem gerenciadas por works, que por sua vez podem conter operações multi-threads. Os works são processos que rodam em background, de E/S assíncronos, ou seja, não são bloqueantes e são gerenciados pela biblioteca libuv.

Observação importante: esse modelo de thread única para a manipulação de Call Stack é o que garante a performance dessa plataforma. Vamos falar mais sobre isso a seguir.

## Multi-thread

Nesse ponto, você já está ciente de que o Node.js é single-thread. Essa característica, a princípio, parece não ser eficiente, o que leva a pensarmos que o desempenho será ruim. Porém, o que ocorre na verdade é o contrário, pois o fato de o Node.js ser single-thread o torna mais eficiente e escalável em comparação a outras alternativas multi-thread, como a linguagem JAVA.

Falando em multi-thread, essa execução envolve o aproveitamento de vários núcleos de um sistema. Diante disso, se uma thread estiver na fila para execução, as outras threads ainda estão em execução. Na teoria, o modelo de multi-thread parece ser o ideal, porém o que não é levado em consideração é que nele uma thread poderá ser bloqueada mesmo tendo outras disponíveis.

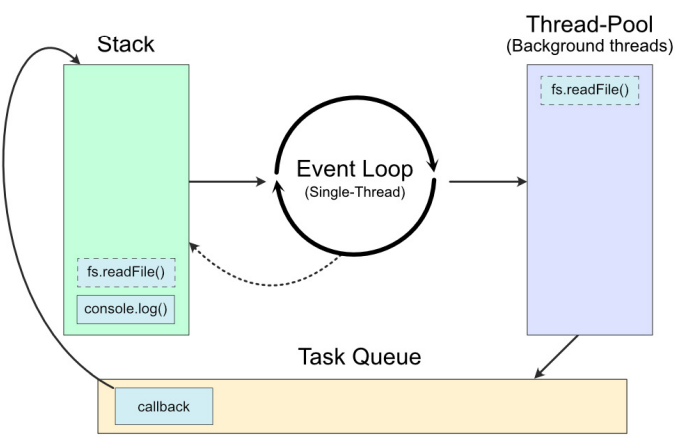
## Call Stack

Call Stack, ou pilha de chamadas, é um mecanismo para rastrear as chamadas de função. Essa característica vem da própria linguagem JavaScript, que já usa a pilha de chamadas para gerir as execuções de funções. Para entender melhor, quando um script é executado, o motor JavaScript cria um contexto de execução global e coloca o processo no topo da pilha de chamadas, princípio conhecido como LIFO. Assim que a execução termina, como seu endereço está armazenado em memória, ela é retirada da pilha e retoma a execução de onde havia parado.

## Event Loop

De uma forma mais simples, um Event Loop é responsável por monitorar a Stack, a pilha, com o intuito de buscar eventos a serem processados em espera. Quando um evento é encontrado, logo ele é executado e o Event Loop é liberado para buscar outros eventos prontos para serem executados.

Cabe ressaltar que isso não representa uma operação longa. Se fosse o caso, como acontece na leitura de arquivos em disco, o Node.js não teria a performance que tem.



A imagem ilustra dois eventos na Stack. Levando em consideração que a Stack é do tipo LIFO, o “fs.readFile()” será executado primeiro.

Resumindo, o Event Loop é um mecanismo responsável por executar e emitir eventos. Na prática, ele é basicamente um loop infinito que, a cada ciclo, verifica em sua fila de eventos se tem algum para ser disparado. Quando isso acontece, o Event Loop executa e envia para a fila de executados.

# Gerenciamento de Controle de Fluxo de uma Aplicação Node.js

Cabe ressaltar que controle de fluxo é a capacidade que uma linguagem tem de desviar o fluxo do programa de acordo com um determinado teste. A exemplo, temos as estruturas condicionais, as quais verificam um determinado bloco de comandos e depois escolhem um caminho a seguir.

O Node.js segue o modelo de programação assíncrono. E o que isso significa?

Esses dois modelos, programação síncrona e assíncrona, são modelos de desenvolvimento intimamente relacionados ao fluxo de execução. No modelo síncrono, uma operação precisa ser finalizada para que outra seja executada. Pode-se dizer que é um modelo linear, previsível, cujas execuções acontecem de maneira cíclica, uma após a outra.

Esse mesmo modelo é seguido por outras linguagens de programação, como o PHP. Em comparação com o modelo síncrono, tem-se o modelo assíncrono, em que uma operação não precisa esperar outra terminar. Em vez disso, elas alternam o controle da execução entre si. Dessa forma, se torna um modelo imprevisível, que não garante a ordem da execução, favorecendo a concorrência.

Em Node.js, é utilizada a função de callback como mecanismo para a escrita de código. Essa técnica foi por muito tempo a forma mais comum para escrita de códigos assíncronos. Cabe ressaltar que hoje em dia estes são considerados obsoletos, já que tornam os códigos confusos no momento da leitura.

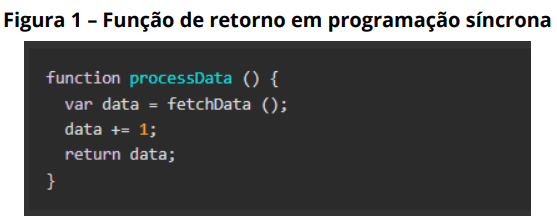
Como já visto, o JavaScript, e por conseguinte o Node.js, tenta simplificar os problemas de códigos simultâneos fornecendo um loop de eventos de threads. A seguir trazemos as responsabilidades do loop de eventos:

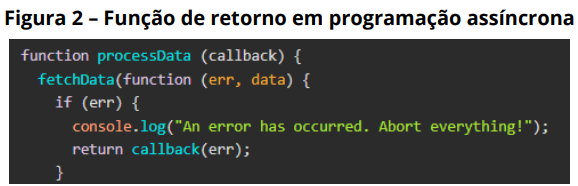
* Executar o código.
* Coletar o processamento de eventos.
* Executar as subtarefas em espera.

Entender o fluxo de controle do Node.js é muito importante para criar aplicações com grandes escalas que precisam ser confiáveis, fornecendo valor comercial previsível.

## Funções de retorno

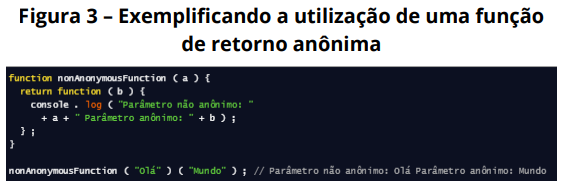
As funções de retorno, ou callbacks, são métodos passados como argumentos dentro de outras funções com a finalidade de serem chamadas em outro momento, predefinido no código. Essa técnica é utilizada, na maioria das vezes, para rodar uma rotina logo após uma execução assíncrona. No entanto, não existem impedimentos para utilização na programação síncrona.





## Funções de retorno anônimas

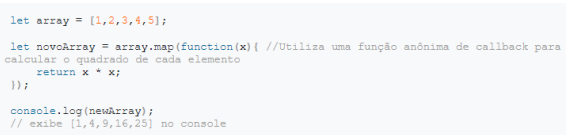
Foi dito no tópico geral sobre funções de retorno que elas são passadas como argumentos dentro de outras funções. Porém, temos as callbacks anônimas, que são funções que não precisam estar ligadas a outras funções. A seguir, trazemos um exemplo dessas funções. Na Figura 3, são ilustradas a sintaxe básica e a aplicação de uma função de retorno anônima.



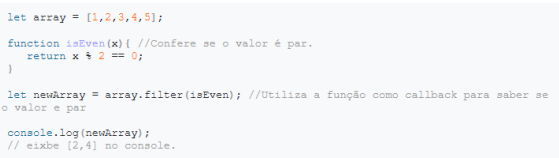
## Métodos que utilizam as funções de retorno (callback)

Retomando o que foi abordado sobre as funções de retorno estarem como argumentos em outros métodos, os principais métodos que utilizam as call-backs são:

* **map():** esse método chama a função de retorno em cada elemento do vetor e, então, traz um novo vetor com os resultados. A baixo mostra como a função map() utiliza as funções de callback.



* **filter():** esse método remove elementos do vetor que não passam por um critério específico, o qual ainda será definido na função de retorno. Para elucidar, veja o exemplo abaixo.



## Programação assíncrona

Neste ponto do material, é perceptível a importância dos conceitos vistos até aqui. Isso porque esse paradigma foca no uso das chamadas assíncronas, ou seja, como visto anteriormente, não deixa a CPU ociosa quando realiza uma operação I/O.

O código do tipo síncrono realiza carregamento sequencial de arquivos, fazendo leitura, uma após a outra, na ordem de execução. Diferentemente do síncrono, o desenvolvimento assíncrono realiza um carregamento em paralelo, e, assim, o uso da CPU é otimizado enquanto estiver ocorrendo uma execução de E/S.

É de suma importância que o código Node.js desenvolvido invoque o mínimo possível de funções bloqueantes, ou seja, funções síncronas. Essa boa prática tem o objetivo de obter a melhor performance na execução do código, otimizando o uso da CPU.

## Async e Await

Antes de falarmos do operador Await, será necessário entender o que é Promise. Como o próprio nome sugere, Promise quer dizer promessa e representa um valor que pode estar ou não disponível. Dessa forma, permite que métodos assíncronos retornem valores como métodos síncronos, ou seja, em vez do valor final, o método retorna uma promessa ao valor em algum momento no futuro.

Para fazer um resumo sobre Promises, iniciamos falando sobre Await. Esse operador foi desenvolvido justamente para esperar uma Promise, ou seja, ele faz a execução de uma função Async pausar, esperando o retorno de uma Promise, e, depois disso, resume a execução da função Async quando o valor da Promise for resolvido.

No resumo sobre Promise, foi dito que ela pode ou não retornar algo. Nesses casos, o Await terá diferente abordagens. Caso a Promise não seja retornada como valor final, ele será convertido para uma Promise resolvida. Ao mesmo tempo, se a Promise for rejeitada, o Await invocará uma Exception com valor rejeitado.

Abaixo ilustra o processo de execução de uma Promise em programação assíncrona.



Para finalizar, cabe ressaltar que somente é possível utilizar uma Await se definirmos a função como Async. Dessa forma, a Await poderá receber a Promise.

## Programação orientada a eventos

Resumindo tudo que foi abordado até aqui, pode-se dizer que a programação com o Node.js é orientada a eventos. Isso significa que, em geral, o cliente não espera pela ação executada no servidor, ou seja, é uma interação **non-blocking (assíncrona).**

Logo, o paradigma orientado a eventos é um fluxo de programação dirigido por eventos específicos, com execução de rotinas que são responsáveis por disparar esses eventos.

# APIs Nativas do Node.js

Para fazer um novo Projeto, abrir uma pasta desejada, e no terminal colocar: npm init

Ele vai fazer umas perguntas

*"name"*: "meus\_exercicios\_da\_facul", NOME DO PROJETO

*"version"*: "1.0.0", ENTER

*"description"*: "Projeto academico", DESCRIÇAO

*"main"*: "index.js", ENTER

*"scripts"*: { ENTER

*"test"*: "echo \"Error: no test specified\" && exit 1"

  },

*"author"*: "Laryssa", NOME DE QUEM VAIFAZER

*"license"*: "ISC" ENTER



Assim ele cria um pacote.json com as informações passadas

