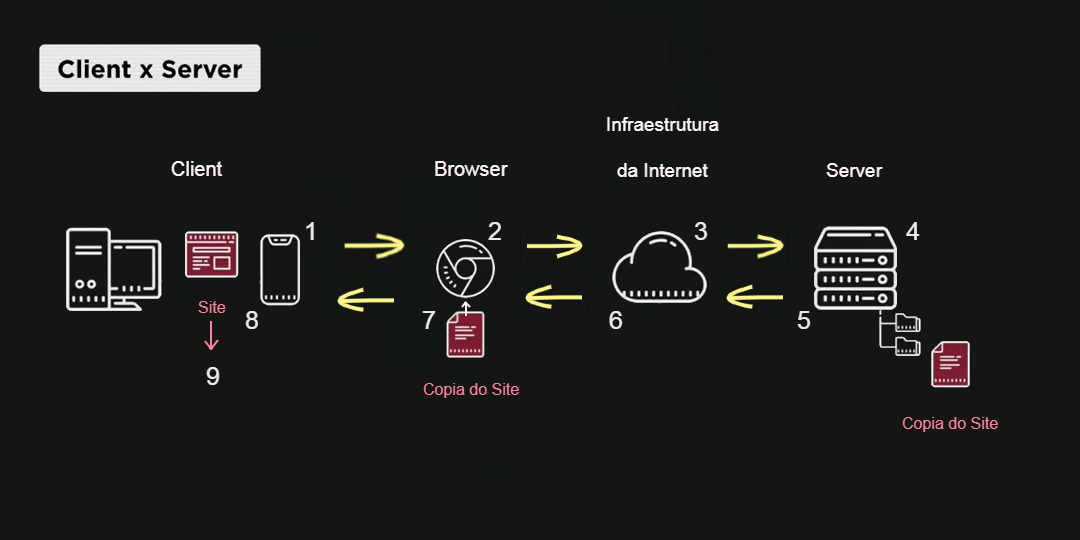
01/09/2022

**JavaScript**

**Client x Server.**

Computadores e celulares ao acessar um conteúdo/site na internet estão utilizando um serviço que está sendo fornecido por um servidor. O JavaScript foi criado basicamente para clientes, mas também funciona para servidores.



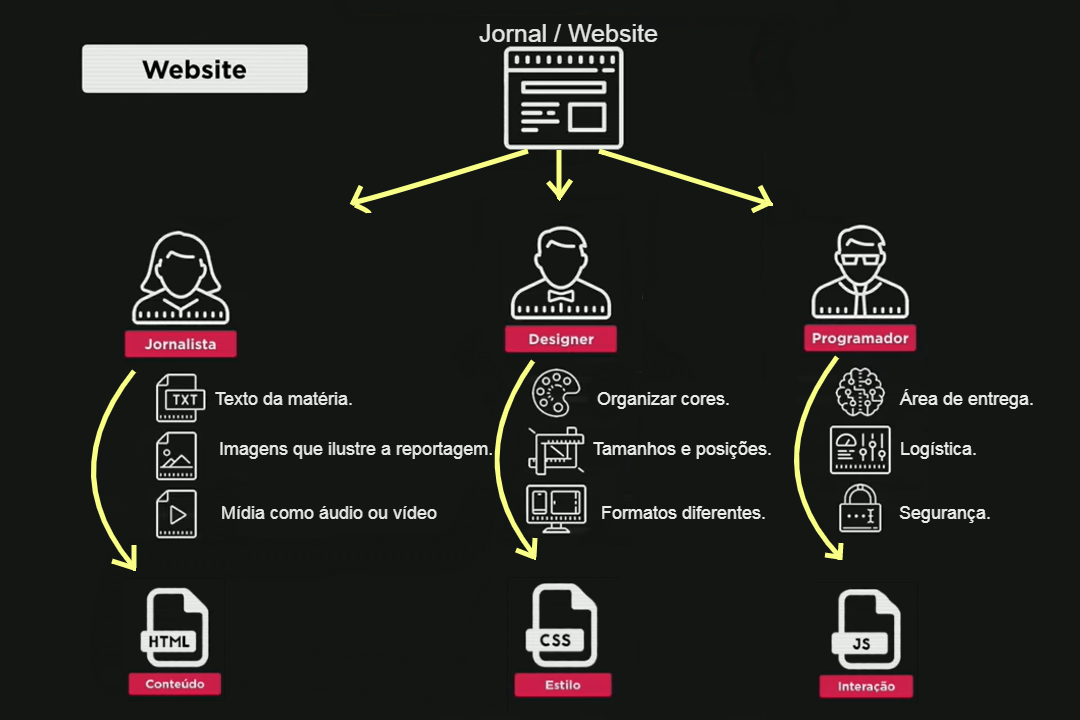
**Website:** Na maioria das vezes é composto por três tecnologias muito importantes.

**HTML** - É responsável pelo conteúdo: textos, imagens, áudios, vídeos, etc…

**CSS** - É responsável pelo designer: cor, tamanhos, posições, bordas, etc…

**JavaScript** - É responsável pela interação: botões, passar o mouse em determinado ponto, direcionamentos de páginas, etc…

Essas três tecnologias funcionam em conjunto e são elas que compõem a parte de clientes da grande maioria dos sites de hoje em dia.



**Bibliografia de Referência para Estudos:**

* JavaScript: O guia definitivo - David Flanagan
* JavaScript: Guia do programador - Maurício Samy Silva
* [Mozilla: Guia de referência](https://developer.mozilla.org/pt-BR/) (Gratuito)
* [Ecma: Guia de referência](https://www.ecma-international.org/) (Gratuito)

**Programas Para Iniciar as Aulas Prática:**

* **Browser** (Navegador).
* **Visual Studio Code** - Para edição de código.

**Plugins:**

**Live Server - Ritwick Dey** - Para atualizar a página de edição automaticamente.

**htmltagwrap - Brad Gashler** - O que é isso

Envolve sua seleção em tags HTML. Pode envolver seleções em linha e seleções que abrangem várias linhas (funciona com seleções únicas e várias seleções ao mesmo tempo).

Para usar, selecione um ou vários pedaços de código e pressione "Alt + W" ("Option + W" para Mac).

**Node.js Exec - Miramac** - Possibilita pela tecla F8 a rodar o código JavaScript pelo Node.js.

* **Node.js** - Para rodar o código fora do Browser.

**Aula Prática:**

* Um bom programador deve organizar os arquivos em pastas. O html, css, JavaScript todas em pastas separadas.
* Sempre coloque os nomes dos arquivos com letras minúsculas.
* Sempre coloque a extensão do arquivo: **ex001.html …,** **.html**, **.css**, **.js**
* Todo site tem **head**(cabeça): onde vai as configurações e **body**(corpo): onde vai o conteúdo.
* O **style**(CSS) é escrito no **head** da construção **html**.
* Em **CSS** todo final de marcação tem ponto e vírgula **;** .
* O código **script** geralmente é escrito antes do fechamento do **body**, para que o script seja carregado depois do código base.
* Em **JavaScript** ponto e vírgula **;** é opcional em final de fechamento de comando. ex: **window.alert(‘texto’);** .

**Executar e Depurar - Debug**

Como tratar erros no VS Code? Clique na área **Executar e Depurar**, depois clique nas opções dentro do **Depurador** e escolha a opção **Run Current File**, em seguida dê atenção a área de **Inspeção** selecione as variáveis que deseja inspecionar e vá ao código, ao passar o mouse ao lado esquerdo do código vai aparecer bolas vermelhas como se fosse de gravação, marque uma bola vermelha a partir de onde você gostaria de começar, para **Inspeção** seguir vá pressionando **F10**.

**Variáveis e tipos primitivos**

É muito importante fazer comentários dentro do código, pois ficará mais fácil de identificar para que serve aquele comando específico. Existem duas maneiras de comentar em JavaScript.

* **//texto** (duas barras), é possível comentar em apenas uma linha.
* **/\*texto\*/** , é possível anexar várias linhas de comentários.
* **=** (atribuição / recebe) , Significa que vai atribuir um valor a variável.
* Existem três formas de delimitar uma **string** (corda / caractere) em uma variável: **“”** (aspas duplas), **‘’** (apóstrofo / aspas simples), **``** (crase), cada uma tem sua diferença.

**var s1 = “JavaScript”**

**var s2 = ‘Dinho’**

**var s3 = `Curso em Vídeo`**

* **+ (concatenação)**, serve para juntar uma frase a uma variável.

window**.**alert(‘Prazer em te conhecer ’ **+** variável **+** ‘!’)

No JavaScript moderno, além de utilizar a palavra **var**, também podemos utilizar a palavra **let**.

**Tipos de identificadores:**

* Podem começar com letra, $, \_

**$1 \_número**

**contador\_A n**

* Não podem começar com números.

**1s**

**2contador**

* É possível usar letras ou números.

**s1**

**s2**

* É possível usar acentos e símbolos.

**média π** (símbolo de PI)

**número**

* Não podem conter espaços.

**contador A**

**média aluno**

* Não podem ser palavras reservadas.

**alert var**

**prompt confirm**

**Dicas para identificadores:**

* Maiúsculas e minúsculas fazem diferença.

**a** e **A** são variáveis diferentes.

**a** (armazenar um dado)

**A** (armazenar outro dado)

* Tente escolher nomes coerentes para as variáveis.

**nome** (para armazenar nome)

**salário** (para o salário)

* Evite se tornar um ‘programador alfabeto’ ou um ‘programador contador’.

programador alfabeto: a, b, c, d

programador contador: n1, n2, n3, n4

**Tipos primitivos primordiais:**

**Number (número):**

**int -** 5 18 -12

**float -** 0.5 -15.9 3.14 8.0

**String (corda / caractere):**

**“Google”**

**‘JavaScript’**

**`Maria`**

**Boolean (valor booleano / valor lógico):**

**true (verdadeiro)**

**false (falso)**

**Data Types / Tipo de Dados:**

Para trabalhar com todos os tipos de variáveis existe um comando que é **typeof**. Para identificar o tipo da variável damos o comando para arquivo HTML e JavaScript:

**HTML** - document.write(typeof maisvariável)

**JavaScript** - console.log(typeof maisvariável)

* **number**

Infinity (infinito)

NaN (not a number / não é um número)

* **string**
* **boolean**
* **null**
* **undefined**
* **object**
* **array (interno)**
* **function**

**Tratamentos de dados**

Como armazenar string a uma variável? A variável nome vai receber uma string, veja no exemplo abaixo.

var nome = window.prompt(‘Digite seu nome: ’) -

window**.**alert(‘Prazer em te conhecer ’ **+ nome** **+** ‘!’); - Vai mostrar o valor armazenado na variável nome.

A comando prompt armazena valores **string**, mesmo adicionando números o prompt vai guardá-los como **string**, então você precisa converter os números que são **string** para **number**.

(number + number) + para adição

4 + 5 = 9

(string + string) + concatenar / juntar

4 + 5 = 45

**Convertendo string para number:**

**Number.parseInt(n)** - Para números inteiros.

parse (analizar / converter)

var nome = Number.parseInt(window.prompt(‘Digite um valor: ’));

**Number.parseFloat(n)** - Para números ponto flutuante (fracionários)

parse (analizar / converter)

var nome = Number.parseFloat(window.prompt(‘Digite um valor: ’));

**Number()** - É para o JavaScript decidir com base nos dados que ele recebeu, se é um valor **Inteiro** ou **Ponto flutuante**.

var nome = Number(window.prompt(‘Digite um valor: ’));

Pode fazer a conversão direta, ao pegar um valor de tipo objeto armazenado em getElement… você pode converter usando **Number()** e **.value**. Exemplo:

var nome = Number(document.getElementById(‘nome ’).value);

Ou se você quiser o valor em string para pegar comprimento e ativar alguma condição.

var nome = document.getElementById(‘nome ’).value;

**Formatando números: var n1 = 1545.5**

**n1.toFixed(2)** -toFixed método interno de todos os números.

document.write(n1**.toFixed(2)**);

Na tela vai ficar com duas casas decimais: **1545.50**

**n1.toFixed(2).replace(“.” , “,”)** - As pontuações que estão em aspas serão trocadas.

document.write(n1**.toFixed(2).replace(“.”** , **”,”**));

Na tela o ponto será trocado pela vírgula: **1545,50**

**n1.toLocaleString(‘pt-br’, {style: ‘currency’, currency: ’BRL’ })**;- O comando *toLocaleString* **(para string de localidade)** é uma String localizada por parte do mundo, neste caso está localizando a moeda brasileira. O que está em chaves { } é um objeto. O **style: ‘currency’** vai mostrar em valor monetário.

document.write(n1**.toLocaleString(‘pt-BR’ , {style: ‘currency’ , currency: ‘BRL’})**);

Será mostrado na tela: **R$1.545,50**.

**document.write** -Escreve no documento **body**. (write / escreva).

**document.writeln** -Escreve e pula uma linha no documento **body**.

**Convertendo number para string:**

**String(n)** - Converter para **string**.

window**.**alert(‘A soma entre os valores é ’ **+** String(**s**));

Pode fazer a conversão direta, ao pegar um valor de tipo objeto armazenado em getElement… você pode converter usando **String()** e **.value**. Exemplo:

Vai converter o valor em string.

var nome = document.getElementById(‘nome ’).value;

Se o primeiro não funcionar tente esse outro.

var nome = String(document.getElementById(‘nome ’));

Ou os dois juntos.

var nome = String(document.getElementById(‘nome ’).value);

**n.toString()** - Valor que quero jogar para uma **string**. **(toString / para string)**

**n.length** - Quantos caracteres a string tem. **(length / comprimento)**

**n.toUpperCase()** - **Para ‘MAIÚSCULAS’**

document.write(`Seu nome é ${n.toUpperCase()}`);

n**.toLowerCase()** - **Para ‘minúsculas’**

**Formatando Strings: var s = ‘JavaScript’**

**‘Eu estou aprendendo** **s’**  // Não faz interpolação.

**‘Eu estou aprendendo ’ +** **s** //Usa concatenação.

window**.**alert(‘A soma entre os valores é ’ **+ s**);

**`Eu estou aprendendo** **${s}`** // Usa template string (formatar string).

window**.**alert(`A soma entre os valores é **${s}**`);

**document.write** -Escreve no documento **body**. (write / escreva).

**document.writeln** -Escreve e pula uma linha no documento **body**.

**Operadores do JavaScript**

**Operadores** -Operadores que serão mostrados no curso JavaScript.

* **Aritméticos**

| **+** | **Adição** | **5 + 2 = 7** |
| --- | --- | --- |
| **-** | **Subtração** | **5 - 2 = 3** |
| **\*** | **Multiplicação** | **5 \* 2 = 10** |
| **/** | **Divisão** | **5 / 2 = 2.5** |
| **%** | **Resto da Divisão** | **5 % 2 = 1** |
| **\*\*** | **Exponencial (Potência)** | **5 \*\* 2 = 25** |

**Ordem de precedência** -Em uma expressão que tenha operadores Aritméticos, Lógicos e Relacionais ao mesmo tempo, são executados primeiro os de preferência mais alta.

| **1º Operadores Aritméticos** | |
| --- | --- |
| **( )** | **1º** |
| **\*\*** | **2º** |
| **\* / %** | **3º** |
| **+ -** | **4º** |

| **2º Operadores Relacionais** | |
| --- | --- |
| **>** | **Todos** |
| **<** |
| **>=** |
| **<=** |
| **==** |
| **===** |
| **!=** |
| **!==** |

| **3º Operadores Logísticos** | |
| --- | --- |
| **!** | **1º** |
| **&&** | **2º** |
| **||** | **3º** |

**OBS:** Precedência de operadores:

**5 + 3 / 2 = 6.5**

Mudando a ordem de precedência:

**(5 + 3) / 2 = 4**

* **Atribuição**

**Atribuição Simples: =**

| **=** | **Atribuição / recebe** |
| --- | --- |
| **var a = 5 + 3** | **8** |
| **var b = a % 5** | **3** |
| **var c = 5 \* b \*\* 2** | **45** |
| **var d = 10 - a / 2** | **6** |
| **var e = 6 \* 2 / d** | **2** |
| **var f = b % e + 4 / e** | **3** |

**Auto-Atribuição: +=**

| **var n = 3** | **Simplificando** |  |
| --- | --- | --- |
| **n = n + 4** | **n += 4** | **7** |
| **n = n - 5** | **n -= 5** | **2** |
| **n = n \* 4** | **n \*= 4** | **8** |
| **n = n / 2** | **n /= 2** | **4** |
| **n = n \*\* 2** | **n \*\*= 2** | **16** |
| **n = n % 5** | **n %= 5** | **1** |

**Incremento -** Pré-Incremento **++x** ouPós-Incremento **x++**

O pré-Incremento soma antes.

O pós-Incremento soma depois.

| **var x = 5** | **Simplificando** |  |
| --- | --- | --- |
| **x = x + 1** | **x ++** | **6** |
| **x = x - 1** | **x - -** | **5** |

* **Relacionais**

| **>** | **Maior que** | **5 > 2** | **true** |
| --- | --- | --- | --- |
| **<** | **Maior que** | **7 < 4** | **false** |
| **>=** | **Maior ou Igual** | **8 >= 8** | **true** |
| **<=** | **Menor ou Igual** | **9 <= 7** | **false** |
| **==** | **Igual** | **5 == 5**  **5 == ‘5’** | **true**  **true** |
| **===** | **Identidade**(Igualdade restrita) | **5 === 5**  **5 === ‘5’** | **true**  **false** |
| **!=** | **Diferente**(Não igual) | **4 != 4**  **4 != ‘4’** | **false**  **false** |
| **!==** | **Desigual restrito** | **4 !== 4**  **4 !== ‘4’** | **false**  **true** |

**Operadores de Identidade -** O operador **==** não testa o tipo e sim a

grandeza. O **operador de Identidade** (operador de igualdade restrita) **===**

verifica se tem o mesmo valor e tipo. Serve também para o operador **!=**

e **!==**

| **5 == 5** | **5 != 5** | **true** |
| --- | --- | --- |
| **5 == ‘5’** | **5 != ‘5’** | **true** |
| **5 === ‘5’** | **5 !== ‘5’** | **false** |
| **5 === 5** | **5 !== 5** | **true** |

* **Lógicos**

**Exemplo:**

**if (**país == “Brasil” **||** país == “brasil”**) {**

**Bloco**

**}**

| **!** | **Negação**(Não) |
| --- | --- |
| **&&** | **Conjunção**(e) |
| **||** | **Disjunção**(ou) |

**Exemplos:**

| **!** | true | **false** |
| --- | --- | --- |
| false | **true** |

Não quero uma caneta **azul**, então pode ser qualquer que não seja **azul**.

| true | **&&** | true | **true** |
| --- | --- | --- | --- |
| true | false | **false** |
| false | true | **false** |
| false | false | **false** |

Eu quero uma caneta **azul** e uma **preta**. Só ficarei satisfeito se receber

uma caneta **azul** e uma caneta **preta**. Para qualquer outra combinação

não ficarei satisfeito.

| true | **||** | true | **true** |
| --- | --- | --- | --- |
| true | false | **true** |
| false | true | **true** |
| false | false | **false** |

Eu quero uma caneta **azul** ou **preta**, ficarei satisfeito se receber as duas, ou uma das duas. Não ficarei satisfeito se não receber nenhum.

* **Ternário**

O operador ternário é **?** e **:** dentro de uma mesma expressão, e é composto

por três partes.

teste **?**  true  **:**  false .

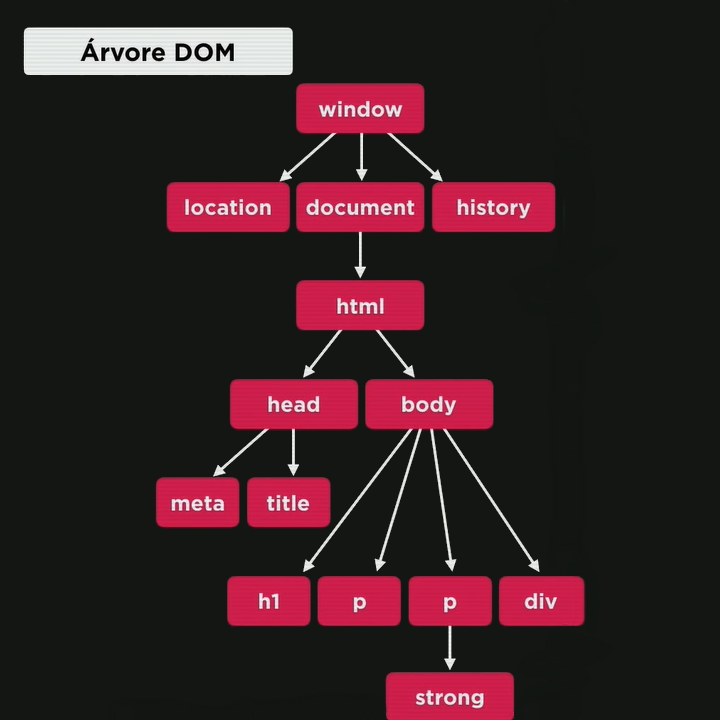
média >= 7.0 **?** “Aprovado” **:** “Reprovado”

**Introdução DOM**

**DOM** é um acrônimo para **Document Object Model** (Modelo de Objeto para Documento), são conjuntos de objetos dentro do seu navegador que vai dar acesso aos componentes internos do seu Website.

A **Árvore DOM** começa da raiz dentro do navegador, e é chamado de **window**. Tudo dentro do JavaScript está dentro de um objeto chamado **window**, que na sua tradução significa janela. Dentro do **window** tem vários outro objetos como:

* **location -** Diz qual a localização do site a URL, qual foi a página anterior, qual é a página atual.
* **document -** Que é o documento atual **ex009.html**.



* **history -** Que guarda o histórico , de onde você veio, a onde você vai, facilitando a navegação dentro do site.

**Selecionado -** Posso selecionar esses elementos para poder navegar dentro da minha árvore DOM, existem vários métodos para isso.

* **por Marca**

Quando o JavaScript pega os dados do DOM (HTML) esse dado vem com tipo objeto, mas se colocar **.value** no final ele converte o dado para string,

Exemplo:

**var p1 = document.getElementById(‘txt’).value**

getElementsByTagName() - O significado do comando **getElementsByTag Name** é “obter elementos pelo nome da tag”. Quando tem mais de um objeto com o mesma tag(marcação) é utilizado o colchete“**[ ]**” para selecionar a tag : exemplo : p ,p (parágrafo).

//Vai selecionar os elementos da tag ‘**p’** , já o número dentro do colchetes é que corresponde ao primeiro parágrafo.

var p1 = window.document.getElementsByTagName(‘p’)[0]

//Vai escrever na tela, o innerText é o texto que está dentro do primeiro parágrafo.

window.document.write(p1.innerText)

* **por ID**

getElementById() - O significado do comando *getElementById* é “obter elemento por ID”. É possível identificar um elemento, objeto por ID. Para selecionar um objeto ou a família dele vou utilizar esse comando.

Exemplo:

var nome = document.getElementById(‘nome’)

* **por Nome**

getElementsByName() - O significado do comando *getElementsByName* é “obter elementos pelo nome”. Quando tem mais de um objeto com o mesmo nome use o colchetes “**[ ]**” para selecionar o objeto.

* **por Classe**

getElementsByClassName() - O significado do comando **getElementsBy ClassName** é “obter elementos pelo nome da classe”. Quando tem mais de um objeto com o mesmo nome de classe é utilizado colchetes “**[ ]**” para selecionar o objeto.

* **por Seletor**

querySelector() e querySelectorAll() - “Seletor de consulta” e “Seletor de consulta todos” é a forma nova de selecionar os objetos, e é recomendada pela maioria dos manuais. Toda “**id**” é representada por **#** (reichstag), e toda “**class**” é representada por **.** (ponto). Esse comando às vezes não consegue levar dados para dentro de uma função.

Exemplo:

var mensagem = window.document.querySelector(‘div**#**msg’)

var mensagem = window.document.querySelector(‘p**.**msg’)

**Eventos DOM**

É tudo que pode acontecer com qualquer elemento, exemplo especificamente com o elemento **<div>**. Os eventos podem ser configurados na parte **html** ou diretamente do **JavaScript**.

Também podemos utilizar eventos com parâmetros. No exemplo abaixo temos os eventos de mouse **move**, **out** e **click** com parâmetros 1, 2 e 3. Logo acima da função temos a variável **quebrada** com valor **false**, a função **mudaLampada** com chamada de parâmetro nomeado como **tipo**, vai receber os valores do evento. Para verificar os valores usaremos a condição composta **if/else**, ou seja, **se** **tipo** for igual a **1**, aparecerá a imagem da lâmpada acesa, **se** for **2** aparecerá a imagem da lâmpada apagada, **se** for **3** aparecerá a imagem da lâmpada quebrada. Então para que a lâmpada depois de quebrada não acenda, usaremos mais uma condição simples, se a lâmpada não estiver quebrada a imagem de acesa ou apagada irá aparecer dependendo do evento, e dentro da condição simples colocaremos outra condição simples para validar a lâmpada quebrada, fazendo com a variável **quebrada** receba **true** “verdadeiro”, então a lâmpada depois de quebrada não acenderá mais.

<h1>Acenda a Lâmpada</h1>

<img

src="\_imagens/lampada-apagada.jpg" id="luz"

onmousemove="mudaLampada(1)"

onmouseout="mudaLampada(2)"

onclick="mudaLampada(3)"

>

<script>

let quebrada = false

function mudaLampada(tipo) {

if (tipo == 1) {

arquivo = '\_imagens/lampada-acesa.jpg'

} else if (tipo == 2) {

arquivo = '\_imagens/lampada-apagada.jpg'

} else if (tipo == 3) {

arquivo = '\_imagens/lampada-quebrada.jpg'

}

if (!quebrada) {

document.getElementById('luz').src = arquivo

if (tipo == 3) {

quebrada = true

}

}

}

</script>

Podemos simplificar o código:

<h1>Acenda a Lâmpada</h1>

<img

src="\_imagens/lampada-apagada.jpg" id="luz"

onmousemove="mudaLampada('lampada-acesa')"

onmouseout="mudaLampada('lampada-apagada)"

onclick="mudaLampada('lampada-quebrada)"

>

<script>

let quebrada = false

function mudaLampada(tipo) {

if (!quebrada) {

document.getElementById('luz').src = ‘\_imagens/’ + tipo + ‘.jpg’

if (tipo == ‘lampada-quebrada’) {

quebrada = true

}

}

}

</script>

**Eventos de mouse:**

* Ao colocar o mouse dentro da **tag**, dispara um evento chamado de **mouseenter**.
* Ao mover o mouse dentro da **tag**, vai disparar várias vezes o evento **mousemove**.
* Ao clicar e manter pressionado o botão esquerdo do mouse dentro da **tag**, vai disparar o evento **mousedown**.
* Ao soltar o botão esquerdo do mouse que está pressionado dentro da **tag**, vai disparar o evento **mouseup**.
* Ao clicar normal dentro da **tag**, vai disparar o evento **click**.
* Ao mover o mouse para fora da área **tag**, vai disparar o evento **mouseout**.

Podemos alterar imagens ao passar o mouse pelo ícone.

Exemplo:

**HTML:** Criamos dois eventos, o primeiro é **onmousemove** com o parâmetro **mudaIcone** mais o endereço da foto, que mudará a foto quando passar o mouse por cima do objeto, o segundo vai alterar a imagem quando o mouse sair de cima do objeto. Mais abaixo temos o script que vai nos levar para um arquivo script separado do html.

<li onmousemove="mudaIcone('\_imagens/home.png')" onmouseout="mudaIcone('\_imagens/especificacoes.png')"><a href="index.html">Home</a></li>

<script src="\_javascript/funcoes.js"></script>

**JavaScript:** Criamos um função com a chamada de parâmetro **foto**, ou seja, quando o evento de mover o mouse por cima do menu home, o **mudaIcone** irá receber a foto para que possa alterar, então selecionamos o objeto por **id** para que possa receber a outra foto.

function mudaIcone(foto) {

document.getElementById('icone').src = foto

}

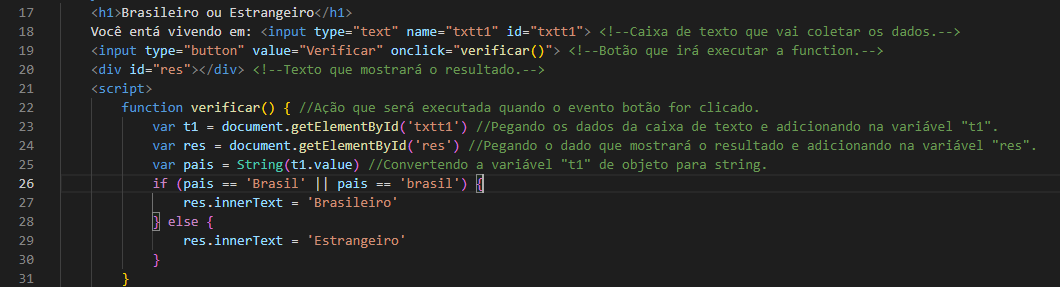
O **setAttribute** é utilizado para mudar o atributo do HTML, foi muito útil para o relógio digital. No meu exercício coloquei a função **relogioDigital** dentro do **setAttribute**.

Exemplo:

setInterval(relogioDigital, 10)

**Funções:**

* **bloco** - São as linhas de código, esse bloco não é executado automaticamente, só será executado quando o evento ocorrer.



* **chaves** - Para executar só quando o evento ocorrer o **bloco** deve ser delimitado dentro de **chaves**. ex:

**{**

**bloco**

**}**

* **function** - Esse bloco tem que ser nomeado como um **function**. ex:

**function{**

**bloco**

**}**

* **nome da função** - Para que o método possa funcionar, tem que nomear a função, geralmente funções de evento é o nome da **ação** que posso fazer. ex:

**function ação( ){**

**bloco**

**}**

* **parâmetros** - E opcionalmente dentro dos parênteses da **ação** colocar **parâmetros**. ex:

**function ação(param){**

**bloco**

**}**

**Condições**

É uma estrutura de controle e ações que podem ser executadas caso a condição seja “**true**” verdadeira ou “**false**” falso. Caso a primeira condição seja falsa, será executada a segunda condição. Existem dois tipos de condições, são elas:

**Condição Simples:**

**if (**condição**)** **{**

**true**

**}**

**Condição Compostas:**

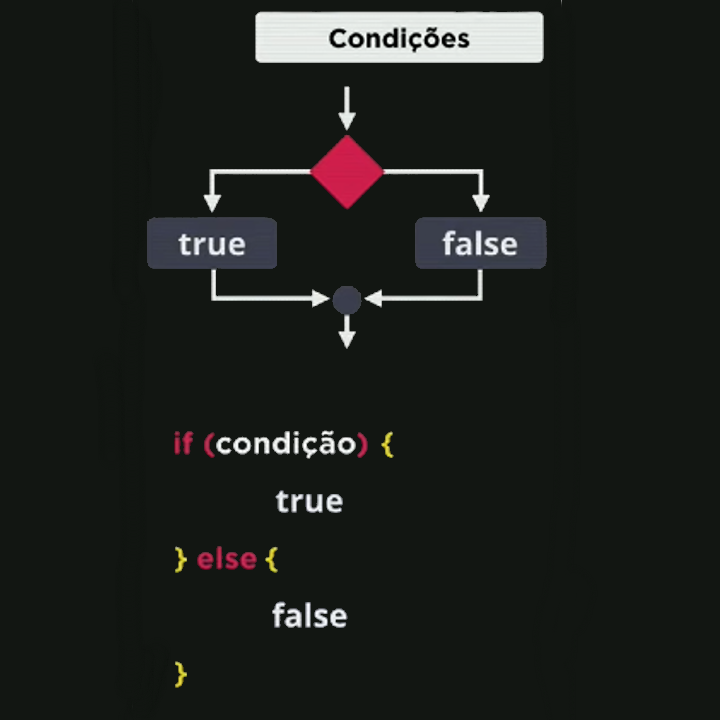
**if** **(**condição**)** **{**

**true**

**}** **else** **{**

**false**

**}**



Para escrever na tela em arquivo JavaScript é utilizado o comando **console.log**(**‘’**)

* **Condição Aninhada:**

**obs:** O **new Date()** também pode ser escrito dessa forma:

hora = new Date().getHours() dessa forma a variável armazenará a hora atual.

//Pegando a hora atual “new Date()”. Serve também para pegar a hora do sistema que está executando seu script, se estiver executando para o cliente é pego a hora do cliente se estiver em um servidor é a hora do servidor.

var agora = new Date()

var hora = agora.getHours()

console.log (` Agora são exatamente ${hora} horas.`)

**if (hora < 6) {**

console.log(‘Boa madrugada!’)

**} else if (hora < 12) {**

console.log(‘Bom dia!’)

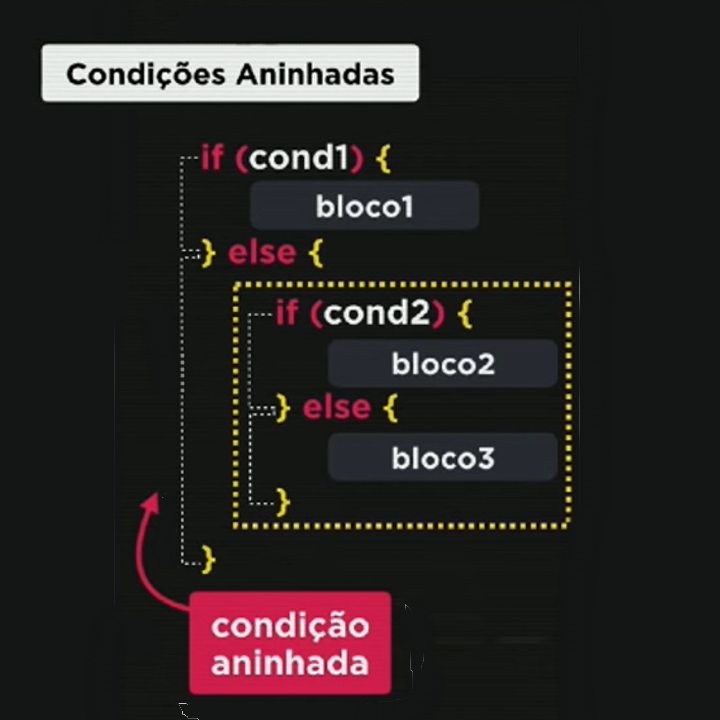
**} else if (hora <= 18) {**

console.log(‘Boa tarde!’)

**} else if (hora <= 24) {**

console.log(‘Boa noite!’)

**}**

****

* **Condição Múltipla:** **switch** **(expressão)** **{ }** É muito útil para trabalhar princi- palmente com valores fixos. O **default** (predefinição) é como **outro caso** e é opcional, já **break** (parar) é obrigatório nesse comando, exceto em **default** pois é opcional.

**switch** **(expressão)** **{**

**case valor1:**

**bloco**

**break**

**case valor2:**

**bloco**

**break**

**case valor3:**

**bloco**

**break**

**default:**

**bloco**

**break**

**}**

**Anotações de marcação, style e comando.**

* **HTML:**

**<head></head>** - Cabeça onde adiciona as configurações como títulos, style, etc…

**link:css** - **<link rel= ‘stylesheet’ href= ‘nomeDoArquivo.css’>** Adicione um link para CSS ser construído separadamente do arquivo HTML. Ao ser adicionado passe o mouse por cima do nome do arquivo ‘nomeDoArquivo.css’ segure Ctrl e clique o VS Code vai perguntar se quer criar aquele arquivo.

**<body></body>** - Corpo onde adiciona o conteúdo texto, foto, vídeos, marcações e interação JavaScript.

**<header></header>** - Cabeçalho.

**<section></section>** - Seção.

**<footer></footer>** Rodapé.

**input:text** - **<input type= ‘text’ name= ‘texto’ id= ‘texto’>** Cria uma caixa de texto, podendo colocar o mínimo ou máximo de caractere dentro da marcação sendo opcional: **min= ‘0’**

O ID é criado para poder identificar a marcação e buscar seu valor através do ID.

**input:number** - Cria uma caixa de número, sendo opcional colocar o mínimo ou máximo de caractere dentro do input e ao final: **max= ‘4’**

**input:button** - **<input type= ‘button’ name= ‘nomeBotão’ onclick= ‘acãoBotão()’>**

**input:radio** - **<input type= ‘radio’ name= ‘texto’ id= ‘texto’>** caixa radio. Se for utilizar mais de um **radio** deve-se colocar na área **name** o mesmo nome da primeira. Quando se quer obrigar o usuário a escolher uma opção se utilizar **checked** dentro do input e ao final.

**img** - **<img id= ‘imagem’ src= ‘imagem.png’ alt= ‘textoAlternativo’>** - Adicionar imagem, o **src** deve ter o nome da imagem com seu formato **.png** que é recomendável. Para que esse dessa forma a imagem deve estar na mesma pasta que o arquivo HTML.

imagem.src = **‘nomedaimagem.png’**

**<div></div>** - Divisões.

**<h1></h1>** Título que é adicionado no body, existem seis níveis.

**<p></p>** Parágrafo. Se for escrito **<p>&copoy;** CursoEmVídeo**</p>** mostrará o símbolo de copyright.

**<label></label>** - **<label for= ‘id’>**Text**</label>** Se for adicionar uma etiqueta na área radio, deve-se colocar o ID do input na área **for** do radio, e em Text coloque o texto que vai ficar ao lado da caixa radio. Para que uma das opções seja selecionada, o input deve ter o mesmo nome como no exemplo abaixo: **tSexo**, usamos **checked** para deixar um das opções previamente marcada.

Exemplo:

<input *type*="radio" *name*="tSexo" *id*="sexMasc" *checked*>

<label *for*="sexMasc">Masculino</label>

<input *type*="radio" *name*="tSexo" *id*="sexFemi">

<label *for*="sexFemi">Feminino</label>

**<select></select>** - **<select name= ‘’ id= ‘’></select>** Selecionar serve para colocar itens dentro e selecioná-los. Para mostrar o valor sem que clique na seleção, você pode determinar o tamanho com o comando **size= ‘’**.

**<option></option>** - **<option value= ‘’>**text**</option>** Opção pode ser escrito dentro do **select**, pois vai selecionar opções. O value não é muito útil para JavaScript, mas para outras linguagens é muito importante. Quando você seleciona a opção dentro do selecionar, basicamente ele irá retornar o valor do que foi selecionado.

**script:src** - **<script src= ‘nomeDoArquivo.js’></script>** Cria um arquivo script separadamente do HTML. Ao ser adicionado passe o mouse por cima do nome do arquivo ‘nomeDoArquivo.js’ segure Ctrl e clique o VS Code vai perguntar se quer criar aquele arquivo.

* **CSS:**

**background / background-color** - muda a cor do fundo.

**color** - muda a cor da letra.

**font** - **font: normal 15pt Arial;** - muda a fonte da letra, **pt** é o tamanho da letra **arial** é o estilo da fonte.

**font-style** - **font-style: italic;** - Estilo de fonte.

**border-radius** - Raio da borda **px**.

**width** - Largura **px**.

**height** - Altura **px**.

**padding** - Preenchimento **px**.

**margin** - Margem **px**.

**margin-top** - Margem do topo **px**.

**margin-left** - Margem da esquerda **px**.

**margin-bottom** - Margem de baixo **px**.

**margin-right** - Margem da direita **px**.

**text-align** - Alinhamento de texto.

**box-shadow** - **box-shadow: 2px 3px 5px black;** - Sombra da caixa. O 2px é deslocamento lateral, 3px deslocamento vertical, 5px espalhamento e black é a cor da sombra.

**filter: drop-shadow(3px 3px 5px cor);** - Filtra a sombra para o contorno das letras.

* **JavaScript:**

**new** **Date**() - Pega dados do servidor ou cliente, como data, hora, ano e etc.

**createElement()** - **var img = document.createElement(‘img’)** - Pode criar elementos, nesse caso criou um elemento imagem.

**setAttribute()** - **img.setAttribute(‘id’, ‘foto’)** - Definir atributo, nesse caso está definindo o **id** e o nome da foto em **.png**, o id seria o **src**.

**appendChild()** - **res.appendChild(img)** - Anexar filho, nesse caso vai anexar a imagem e mostrar na tela.

**\u{ }** - **`\u{1F449}`** - Serve para colocar Emojis.

**item.value = `tabuada${contador}`** - Serve para mostrar o valor da opção selecionada. É útil para outras linguagens.

**item.value = ‘’** - Limpa o espaço.

**item.text = `${número} X ${contador} = ${número \* contador}`** - Serve para mostrar o texto de um Elemento Criado.

**item.focus()** - Deixa o cursor fixo para digitação, com isso não precisará clicar toda vez que digitar outro dado.

**Repetições em JavaScript**

A estrutura de repetição e controle **while** tem seu teste lógico no início. A tradução para **while** é “**enquanto**”. Na imagem abaixo o comando vai se repetir enquanto tiver fatias de pizza.

**while (condição) {**

**bloco**

**}**



No exemplo abaixo a estrutura de repetição **while** tem uma condição lógica no início, **while “enquanto”** a condição for verdadeira o comando executa o bloco, ou seja, **while “enquanto”** avariável **c** for menor ou igual a **5** o comando executa o bloco, em seguida a variável contadora **c** vai receber +**1** e se repete todo ciclo.

var c = 1 // Variável contador.

**while (**c <= 5**) {**

console.log(`Passo ${c}.`)

c++ // c = c + 1

**}**

Vai mostrar na tela:

Passo 1.

Passo 2.

…

Enquanto a variável “**c**” for menor ou igual a cinco, o comando vai se repetir.

A estrutura de repetição e controle **do..while** tem seu teste lógico no final. A tradução para **do..while** é “**fazer enquanto**”.

**do {**

**bloco**

**} while (condição)**

No exemplo abaixo a estrutura de repetição **do..while** tem uma condição lógica no final, **do** **“fazer”** o bloco primeiro, em seguida a variável contadora **c** vai receber +**1**, **while “enquanto”** a condição for verdadeira o comando se repete, ou seja, **while “enquanto”** a variável **c** for menor ou igual a **5** , o comando se repete.

var c = 1 // Variável contador.

**do** **{**

console.log(`Passo ${c}.`)

c++ // c = c + 1

**}** **while (**c <= 5**)**

Vai mostrar na tela:

Passo 1.

Passo 2.

…

Enquanto a variável “**c**” for menor ou igual a cinco, o comando vai se repetir.

A estrutura de repetição **for**, é chamada de repetição com variável de controle.

**for (inicio; teste; incremento) {**

**bloco**

**}**

No exemplo abaixo a estrutura de repetição **for “para”** tem o **inicialização**, **teste lógico** se o teste for verdadeiro **executa o bloco**, depois **soma +1 ao incremento**, variável contadora **c** e todo ciclo se repete.

**for** **(**var c = 1**;** c <= 5**;** c++**)** **{**

console.log(`Passo ${c}.`)

**}**

**Variáveis Compostas**

Diferença entre variáveis simples e compostas.

* Variáveis simples só conseguem armazenar um valor por vez.
* Variáveis compostas devem ser capazes de armazenar vários valores em uma mesma estrutura.

A variável composta sempre é declarada acompanhada dos colchetes “**[ ]**” e você pode armazenar quantos valores quiser dentro dela, exemplo: **var frutas = [‘maçã’, ‘banana’, ‘jaca’]**. Você pode identificar esses valores pelo índice ou chave que cada uma recebe ao ser armazenado dentro de uma variável composta e esse índice começa a partir do número zero.

A variável **frutas** é um **array** (variedades / vetor), e é composto por elementos que agrupa o espaço da memória, o valor colocado dentro dele e o índice, basicamente um vetor é uma variável composta por vários elementos e cada elemento é composto por seu valor e uma chave de identificação.

**var num = [8, 5, 9]**

**0 1 2 índice/posição**

Como adicionar mais um valor ao vetor? Quero adicionar a string **uva** na posição 3 ou qualquer outra posição. Não se esqueça dos colchetes “**[ ]**”.

**var num = [8, 5, 9, 1]**

**0 1 2 3 índice/posição**

**num[3] = 1**

**console.log(num) //**vai mostrar na tela a forma padrão **[8, 5, 9, 1]**.

Agora quero adicionar um valor na última posição, o comando **.push** “empurrar” é um método interno que vai empurrar o novo valor para última posição.

**var num = [8, 5, 9, 1, 3]**

**0 1 2 3 4 índice/posição**

**num[3] = 1**

**num.push(3)**

O comando interno **length** mede o comprimento do vetor, ou seja, a quantidade de elementos dentro do vetor. No caso acima temos cinco elementos.

**console.log(`O vetor tem ${num.length} posições.`)**

O comando interno **sort()** ordena todos os elementos em forma crescente. Se adicionar valores depois do comando **sort()** o valor não ficará ordenado.

**var num = [1, 3, 5, 8, 9]**

**0 1 2 3 4 índice/posição**

**num.sort()**

Para trabalhar com o vetor de forma personalizada, é muito utilizado o comando **for** de duas formas, sendo a segunda forma mais adequada.

**var num = [8, 5, 9, 1, 3]**

**for (var c = 0; c < num.length; c++) {**

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

**}**

E vai se repetir enquanto a variável **c** for menor que o comprimento do vetor.

//Na tela: A posição 0 tem o valor 8.

A posição 1 tem o valor 5.

…

A segunda forma de fazer o exemplo acima é utilizando o **for** de forma simplificada, esse comando é otimizado para variáveis compostas e objetos.

**var num = [8, 5, 9, 1, 3]**

//Ler dessa forma: Para cada posição **c** dentro de **num** execute um bloco. O **for** se ler PARA e o **in** se ler “dentro / em”.

**for (let c in num) {**

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

**}**

//Na tela: A posição 0 tem o valor 8.

A posição 1 tem o valor 5.

…

Para buscar um valor dentro de vetor se utiliza o comando **.indexOf** “índice de” com a letra “O” maiúscula.

**num.indexOf(9)**

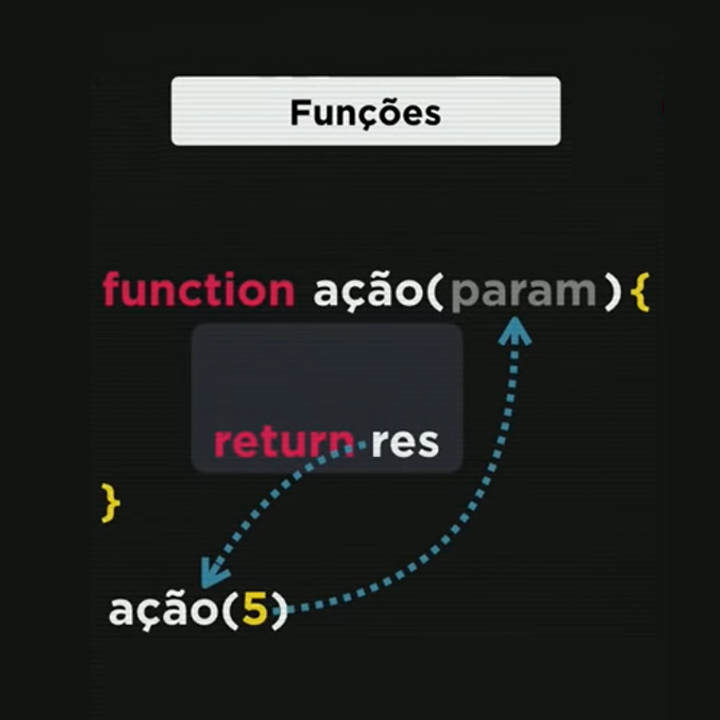
**var num = [8, 5, 9, 1]**

**0 1 2 3 índice/posição**

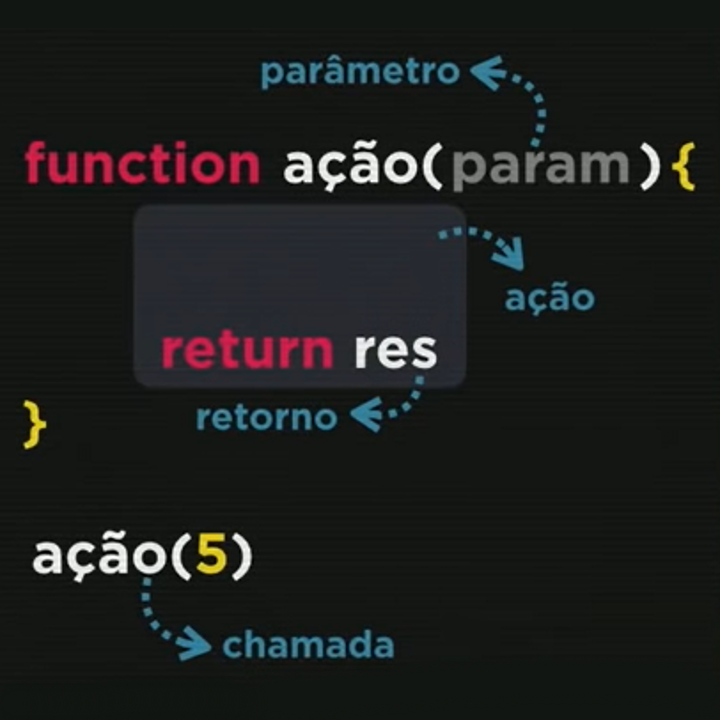
O comando vai me retornar a posição onde está armazenado o valor **9**, por tanto vai me retornar o índice **2**, se você buscar um valor que não tem como o valor **6** o comando vai retornar **-1**.

**Funções**

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



Toda função tem um nome e ações, a função na imagem abaixo tem o nome “ação”, logo abaixo temos a chamada para função, essa chamada com o nome “ação” e tem o valor cinco, que passará para função através de parâmetro, a função será executada e ao seu final, irá retornar um valor para chama “ação”.



Nessa outra imagem, temos um função com chamada, para saber se um número é par ou ímpar e com esse exemplo fica mais claro.



**Próximos Passos**

* **functions**
* **objetos**

Para problemas mais complexos utiliza se o object. Na array o índice é fixo, já o object você pode dar um nome correspondente ao valor. Veja na imagem abaixo.



* **modularização**
* **RegEx - Expressões regulares**
* **JSON**
* **AJAX**
* **NojeJS**