**JavaScript – Curso em vídeo:**

**Cliente x Servidor:** Quando uma pessoa (no celular, computador, tablet) usa um serviço de internet, na verdade ela é um “**client**” e está fazendo uma **requisição de dados** para um servidor.

* **Client:** é a pessoa usando a internet e fazendo requisições de dados
* **Browser (navegador)**: é uma interface gráfica “facilitada” que permite o cliente acessar a internet e ver graficamente um site
* **Nuvem / estrutura internet:** É uma estrutura de redes que recebe a requisição por meio de uma **URL** e desvia a rota do navegador para o site pedido

**ECMAscript:** Com a briga pllas linguagens com a Microsoft, os criadores do JS decidiram padronizar a linguagem do JS contratando a ECMA. Portanto, **ECMAScript** é a padronização da linguagem JavaScript. Versão é a **ES5** lançada em 2009. **ES6** em 2015. A última versão é ES2018

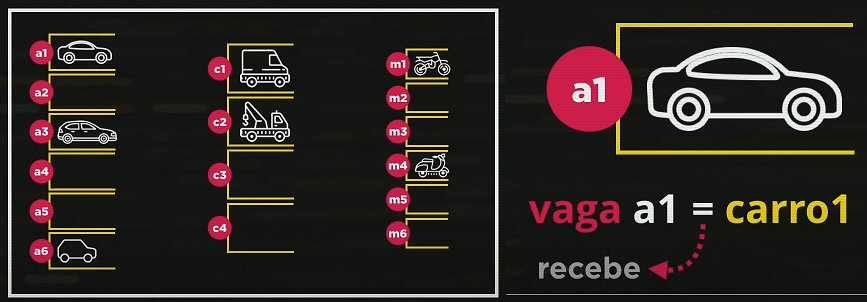
**Node.JS:** Criado em 2010 ele é uma engine que permite rodas o **JS** fora do navegador. Foi criado com o códido do V8 motor de JS do Google Chrome.

**Tecnologias (Bibliotecas e Frameworks):** Que surgiram a partir do JS.

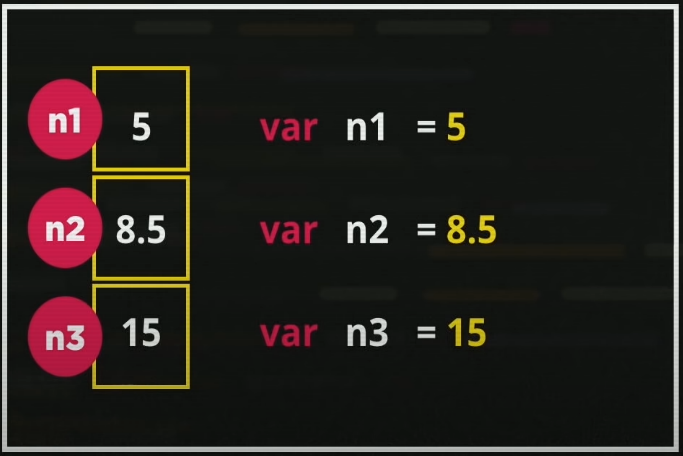
* JQuery
* Angular (Google)
* React (facebook)
* VUE (Engenheiro Google)
* Electron (GitHub)
* Ionic (dispositivos móveis)
* Cordova

**Leituras recomendadas:** JS o guia definitivo (David – O raily). Guias de referência Mozzila e da ECMA

**Primeiros passos <script>:** Em um site, o <script> pode aparecer dentro do código HTML ou pode vir numa folha separada. Se vier dentro do documento html ele aparece na parte final do <body> isso por que queremos carregar o <script> depois de tudo. Todos os comandos que vierem dentro da <scrip> serão códigos de programação. **JavaScript** é case sensitive, ou seja, tem diferença escrever com letras minúsculas e maiúsculas. Obs: para deixar comentários no JS existem duas maneiras: 1) // Escrever depois ou 2) /\* Escrever algo \*/

**Variáveis e tipos primitivos:** Em JS um **único sinal de =** significa **recebe. Variáveis** são slots (espaços) que recebem (=) valores que podem ser alterados ou até mesmo nulos. Esses valores são guardados na memória do computador e podem ser requisitados. **Variáveis** guardam **dados.**

**No JS** as variáveis são utilizadas usando **var** ou **let.**

**Atribuição de variáveis:** é quando passamos um determinado valor para um determinado slot que recebe essa variável e armazena ela temporariamente.

**Identificadores:** São os nomes dos slots de variáveis (n1, n2, n3). Para dar **nome** aos identificadores temos que seguir regras: 1) **Não pode** ser palavra **reservada** (comandos do próprio JS). 2) Pode: **letras** (A..Z) **/ números** (1.. ∞) **/ acentos** (â..é) **/ símbolos** (π). 3) **Pode** começar com **letra / $ / \_** 4) **Não pode** começar com **números** (1n)ter **espaços** (variável um)

**TIPOS PRIMITIVOS de variáveis no JS:** Para saber, basta digitar typeof no node. **\*Importante:** uma característica do JS é que quando declaramos uma variável ela assume o tipo de valor que é passado, podendo, assumir diferentes tipos e tamanhos de acordo com o valor passado. Var n = 10 🡪 typeof: number se depois de um tempo declarar n = ‘Matheus’ 🡪 typeof: string. Podemos escrever variáveis de 3 maneiras com funcionalidades diferentes: **var; let; const.**

1. **NUMBER:** Variáveis do tipo numérica – recebem números de todos os tipos (inteiros, positivos e negativos, reais, fracionados).
2. Infinity
3. NaN (Not a Number)
4. **STRING (CADEIA CARACTERES):** Podem ser compostos por letras, números, traços e pontos, parênteses.Especificados dentro de aspas “, ‘, `, Nomes, sentenças, CPF, telefone.
5. **BOOLEAN (BOOLEANOS):** Valores booleanos (True; False;).
6. **Null**
7. **Undefined**
8. **Object**
9. Array (vetores)
10. **Function**

**POR QUE USAR LET NO LUGAR DE VAR:**

Usar let como variável é mais **seguro** pois ele sempre irá obedecer o let dentro daquele bloco em que você está trabalhando (em questão), se eu tivesse o costume de usar var eu poderia acabar **sobrescrevendo** um valor que eu gostaria que fosse imutável com let.

Principalmente quando estamos trabalhando com várias funções com várias variáveis, o aconselhado é que dentro de cada bloco dessas funções usemos a variável let.

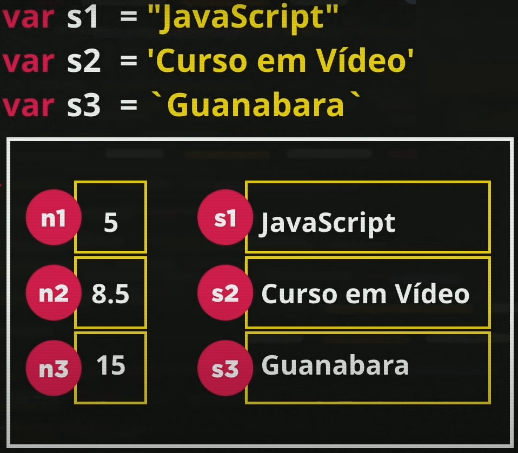
Usar **let** é assumir uma **limitação** proposital em prol de segurança.(de escopo local – só funciona dentro do bloco onde estamos trabalhando)

**Dicas**

* **para criar identificadores: Letras maiúsculas e minúsculas** fazem diferença. Então se o identificar de uma variável for n1 não conseguiremos retorná-la usando o identificador N1.
* **Strings e não strings**, Se eu usar var x = ‘10’ e var x = 10, embora os valores sejam os memos a primeira recebe a string escrita 10 e a segunda recebe o número 10; caso 1 x + x vai retornar: 1010, ou seja, ele vai concatenar string e formar um texto numérico no caso 2 vai retornar: 20, ou seja, a soma de 10 + 10.
* **Use nomes coerentes para identificadores:** caso 1 usar var nome para receber nome, var tel para receber telefone e var idade para receber a idade é **mais coerente** do que usar n1 para nome n2 para telefone e n3 para idade. No segundo caso, se for preciso retornar alguma variável nem saberemos mais qual variável corresponde a qual valor.

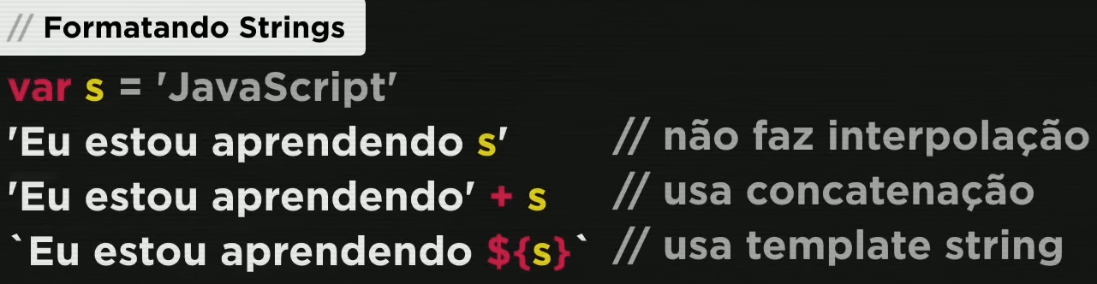
**Mas e o tamanho:** Existem slots de tamanhos diferentes capazes de receber dados de tamanhos diferentes.

**Node.JS dentro do VScode:** Terminal > Novo terminar ou Ctrl + shift + ‘ > digitar ‘node’ no prompt. Para limpar o prompt: Ctrl + l.

**Exemplo:** JS recebendo uma string (‘cadeia de caracteres’). No JS pode usar ‘aspas simples’, “aspas duplas” ou `aspas`. Cada uma tem uma função específica.

**Sinal de + no JS:** O sinal de + no JS pode servir como **soma** ou como **concatenação.** Isso vai depender do tipo de variável, por exemplo: se for (number + number) automaticamente o JS irá somar, por outro lado, se for (string + string) o instinto do JS vai ser concatenar.

**Template strings: ${n}** é possível digitar mais fácil e rápido se usar crase e tamplate string

**FORMATANDO STRINGS:**

**Concatenação de strings:** Possível, mas é mais trabalhoso de escrever

****

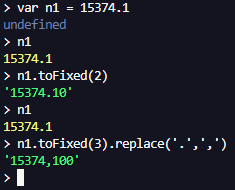
**Template string (com crase ` `):** Mais rápido de escrever. Recente e criado por ECMAScript. Fica mais elegante, mais fácil de ler de escrever e dar manutenção.



**s.length:** Quantos caracteres a string tem.

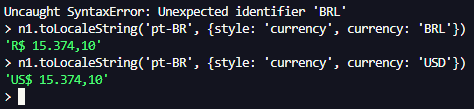
**s.toUpperCase( ):** Passatudo para ‘MAIÚSCULA’

**s.toLowerCase( ):** Passatudo para ‘minúscula’

**FORMATANDO NUMBERS:**

variável.**toFixed(n):** Fixa quantidade de casas decimais, n = quantidade de casas decimais.

variável.**toFixed(n).replace(‘.’,’,’):** Fixa quantidade de casas decimais, n = quantidade de casas decimais e o replace é: troque ‘isso’ por ‘isso’



**Manipulação de dados JS:**

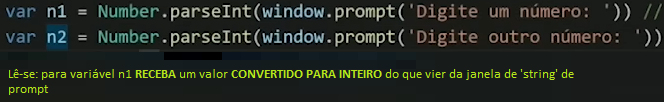
****

**Convertendo variáveis sendo (n) o que quero converter (parse = converter, parsear):**

**Number.parseInt(n):** Converte um número para um número inteiro. (usado para forçar inteiro)

**Number.parseFloat(n):** Converte um número para um número real. (usado para forçar real)

**Number(n):** Conver um número de acordo com o valor recebido, de acordo com o contexto que ele julgar como correto.

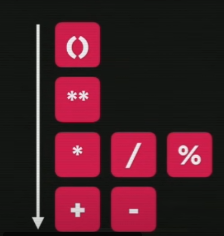
****

**Convertendo number 🡪 string**

**String(n):** Converta (n) para String.

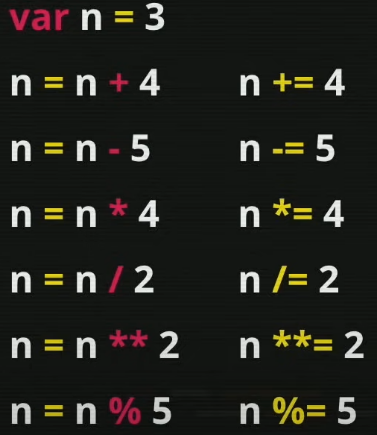
**n.toString( ):** (n) converta para String.

**Operadores no JS:**

**** **Aritméticos:** (+) (-) (\*) (/) (%) (\*\*). Eles precisam de 2 operandos. (/) é o operador de divisão real. (%) é o operador de resto da divisão inteira, divisão sem vírgula, ele retorna o valor do resto. (\*\*) é operador de **potência 2\*\*3 = 8.**

Precedência de operadores: é a prioridade de operação a ser feita, por exemplo: 5+3/2 = 6,5. A precedência de operadores é:

**Parênteses > potenciação > multiplicação, divisão e divisão de resto inteiro > soma, subtração.**

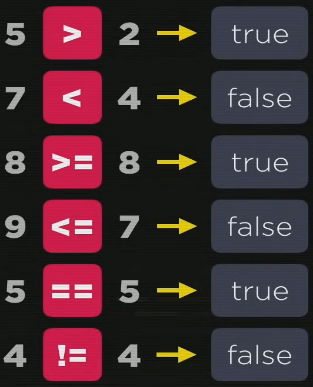
É possível **guardar** os valores das operações aritméticas em variáveis. Além disso é possível atribuir um valor à uma variável e depois que ela passar a existir seu slot pode receber outros valores futuramente. Ao longo de um processo uma variável pode começar em um valor e atribuir vários outros valores ao longo do caminho.

**Shorthand:** Quando **var n = n + 4** (quando uma variável recebe ela mesmo mais outro valor) é possível usar o shorthand **n += 4.** Portanto para a mesma variável **n = n + 4 🡪(é igual) 🡪 n += 4.** A grande maioria das linguagens atuais aceita essa sintaxe.

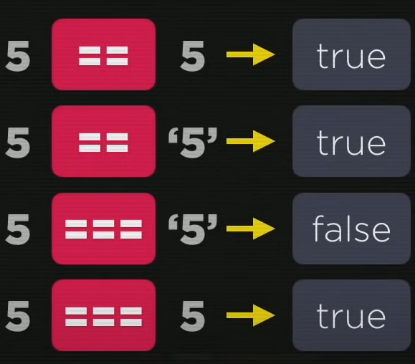
**Operadores de incremento (pré-incremento e pós-incremento // pré-decremento e pós-decremento):** Servem para quando queremos somar ou subtrair **1 unidade.** Para uma variável n, ficaria: n++ (soma sem mostrar) ou ++n (soma já mostrando) ou n- - (subtrai sem mostrar) ou - - n(subtrai já mostrando).

****

****

 **Atribuição:**

**Relacionais:** Sãos operadores relacionais da maioria das linguagens. Para todas expressão que tenha um operador relacional o resultado será sempre um valor booleano (true / false). São eles: **Maior // Menor // Maior OU igual // Menor OU igual // igual // diferente.**

****

**Operadores de identidade:** == (é igual) === (é idêntico restrito). **Igual** no JS o igual se refere **apenas** ao valor da variável e **não ao tipo**, então 5 == 5: true e 5 == ‘5’: true. No entando podemos usar um identificador mais específico que é o === (é idêntico), nesse caso se refere ao **valor** e também ao **tipo** da variável, então 5 === 5: true e 5 === ‘5’: é false pois o primeiro é do tipo primitivo number e o segundo é do tipo primitivo string. Var x = 5 🡪 typeof ‘number’ e var y = ‘5’ 🡪 typeof ‘string’. Ele também existe para o != (diferente) e **!== (desigual restrito)**

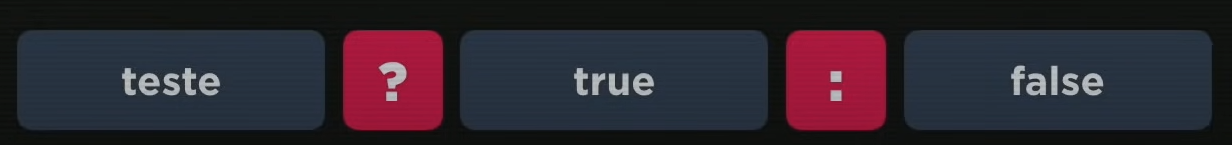


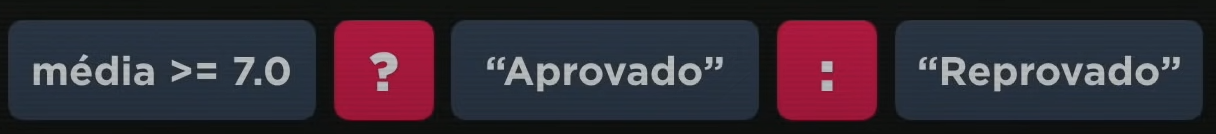
**Lógicos:** São operadores lógicos tradicionais **! (negação ~ lógico) && (conjunção ‘e’ lógico) || (disjunção ‘ou’ lógico)** . **! (nagação)** é um operador unário, ou seja, só tem um operando. **&&** (‘e’ lógico) é um operador binário, ou seja, com dois operandos

**Hierarquia de operadores lógicos: ! > && > ||**

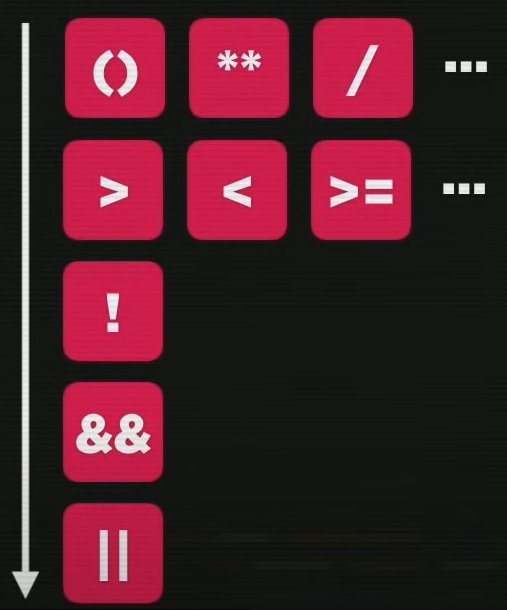
Negação > ‘e’ lógico > ‘ou’ lógico.

**Ternário: ? e :** Ele se chama ternário pois tem 3 partes (operandos). Teste lógico > o que vai acontecer se o teste lógico for verdadeiro e o que vai acontecer se o teste lógico for falso.





**Ordem de precedência total:**

****

**Introdução e eventos DOM: Document Object Model** está presente quando se está rodando JS dentro do navegador.

Como fazer a **ligação** entre JS e HTML e CSS.

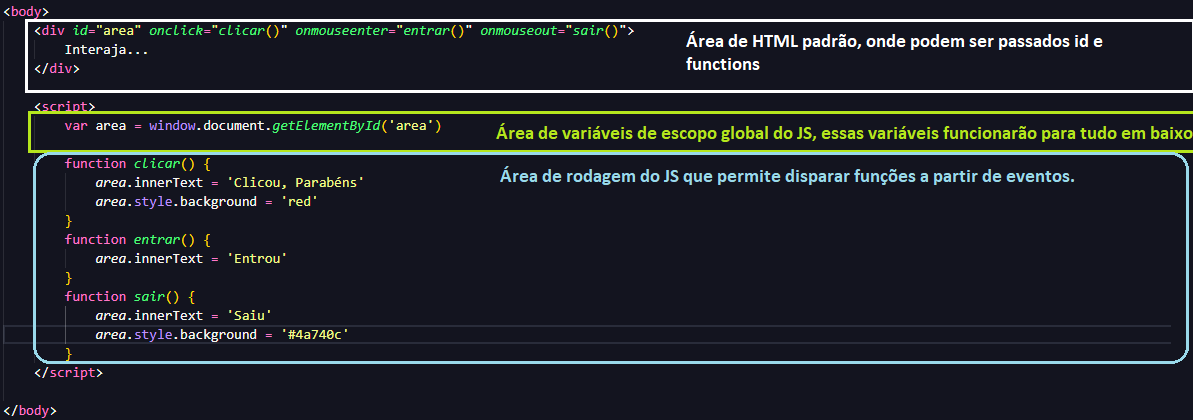
É importante perceber que quem pertence “está a baixo” é uma child e quem está “a cima” é um pai.

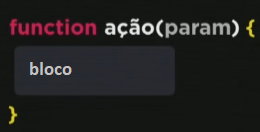
E cada um desses elementos pode ser acessado de 5 maneiras diferentes. **Marca (TAG) / ID / Nome / Classe / Seletor.**

Essas 5 técnicas são as maneiras de chamar elementos com JS.

1. **MARCA (TAG):** getElementsByTagName( ) 🡪 window.document.getElementsByTagName(‘p)[0]. Na janela navegador > no documento > pegue o elemento pela tag > que for <p> > o primeiro
2. **ID:** getElementByid( )
3. **NOME:** getElementsByName( )
4. **Classe:** getElementsByClassName( )
5. **Seletor:** querySelector( ) ou plural querySelectorAll( )

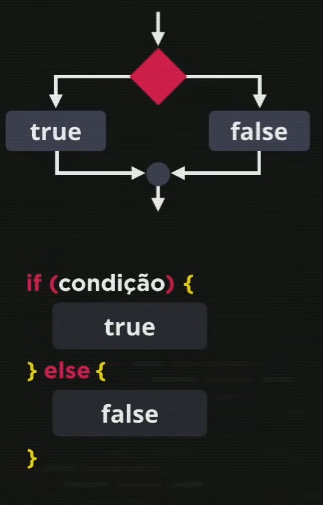
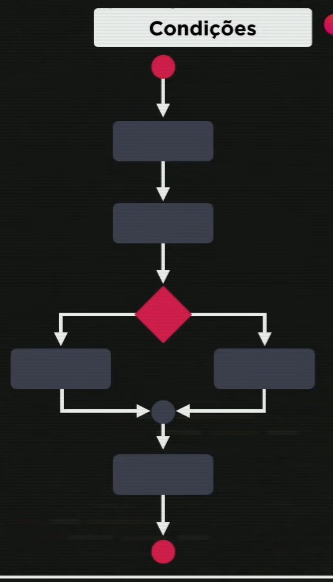
**Eventos DOM:** [**https://developer.mozilla.org/en-US/docs/Web/Events**](https://developer.mozilla.org/en-US/docs/Web/Events)Eventos são acontecimentos que podem acontecer com os elementos de um site. Mouseenter (mouse entrando no objeto), mousemove (microeventos mouse movendo dentro do objeto), mousedown (clique), mouseup (mouse soltantdo), click, mouseout (mouse saionde de cima do elemento).

**Funções:** são códigos que disparam quando um evento ocorre.b



****

**CONDICIONAIS: (if / else 🡪 se / senão)** Desvio condicional 🡪 Programa é executado de cima para baixo até um momento onde ocorre uma bifurcação.

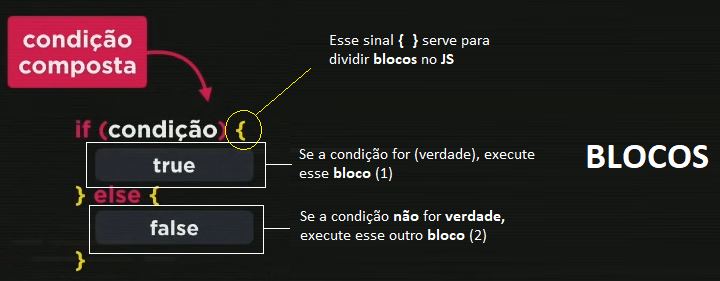
******ESTRUTURA DA CONDICIONAL:** O primeiro bloco está entre as **primeira** alternativa **if { }** e o **segundo** bloco está entre a segunda alternativa **else { }.**

**É LIDO:** Se (if) condição 1 for verdadeira, então primeiro { }, senão (else) executar segundo { }.

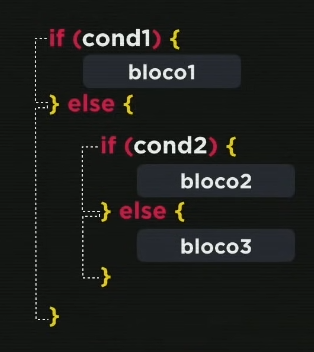
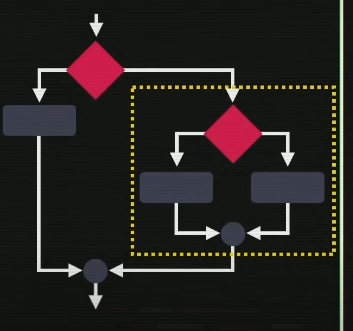
**Condição simples:** Tem apenas o if { }, é somente o “se” caso seja verdade ele executa esse comando

**Condição composta:** if { } else { }.

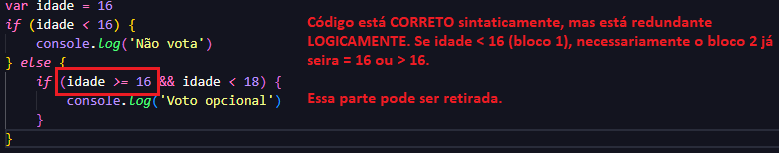
Para condições compostas é muito importante entender como estão localizados os **blocos** (partes) de cada sentença.

****

**Condições compostas (condições aninhadas):**

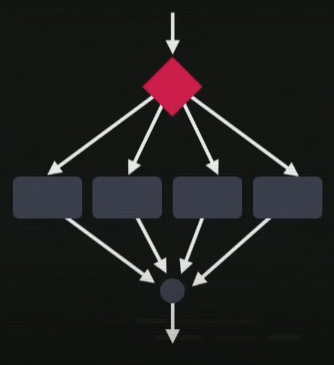
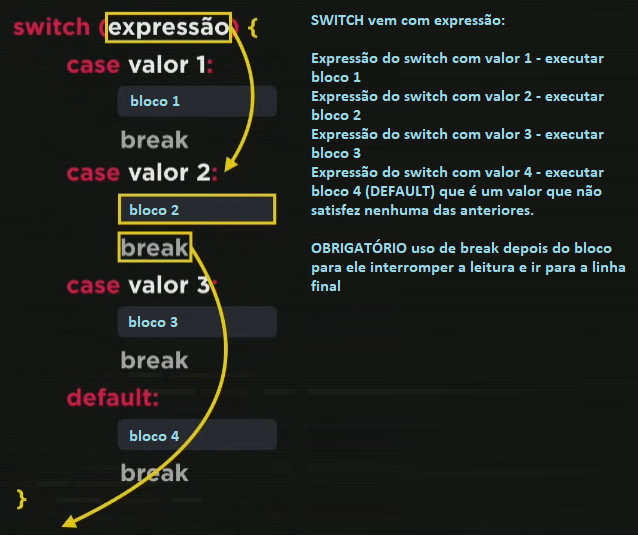
****Colocar uma estrutura condicional dentro de outra estrutura condicional.

No caso do código escrito acima, para chegar a executar o bloco 2 ou o bloco 3 é necessário que a condição do bloco 1 seja falsa

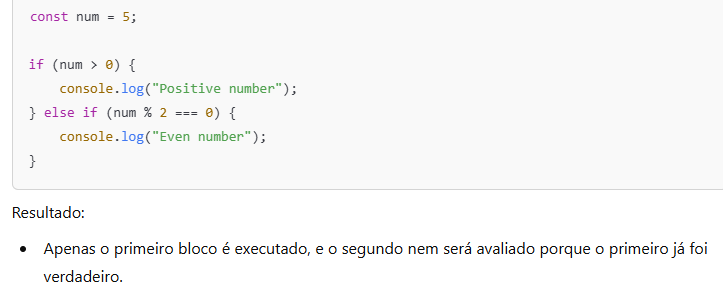
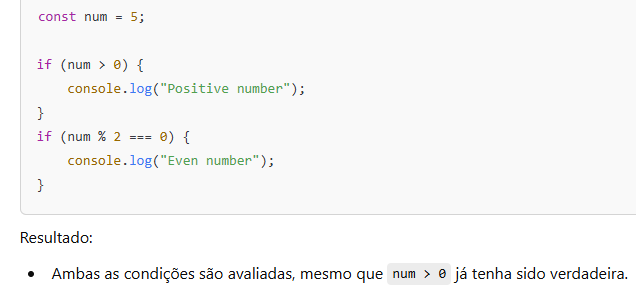
****

**O QUE APRENDI COM IF ELSE HORA:** /\**Para avaliar a madrugada TEMOS que começar as condições pela madrugada, pois se coloco uma condição (hora <12) e no final do código uma condição (hora > 0 && hora <6), esta não consegue ser avaliada pois aquela intercepta e avalia valores que também são compreendidos na madrugada. Ela faz a análise primeiro*\*/

Além disso terminando com a última expressão (apenas else) garantimos que tudo que não está compreendido retornará o resultado “hora inválida” caso o usuário digite algo errado ou uma hora inexistente.

**CONDIÇÕES MÚLTIPLAS:** Ideal para trabalhar com **valores fixos** e não com intervalos de valores (como if else)

**\*MUITO IMPORTANTE:** É possível usar a condição **if else** de duas formas diferentes quando temos mais de uma condição. No primeiro caso pode ser escrito o 1° bloco com “if”, e no 2° bloco com “if” e para terminar else. Nesse caso o programa irá **checar** se o primeiro if é verdade (ture), depois ele faz uma 2° **checagem** para saber se o segudo if é verdade (true) e por final, se nenhuma dessas duas condições forem verdade ele parte para o último bloco **else** e **checa** se esta condição se satisfaz. **(IMPACTO = CADA IF É VERIFICADO DE FORMA INDEPENDENTE, MESMO QUE A CONDIÇÃO ANTERIR SEJA VERDADEIRA.** MAIS TEMPO DE VERIFICAÇÃO DE CONDIÇÕES e PODE LEVAR A VERIFICAÇÕES DESNECESSÁRIAS) (**USADA = PRECISAMOS QUE MAIS DE UMA CONDIÇÃO SEJA VERDADEIRA.** QUANDO QUEREMOS FORÇAR A VERIFICAÇÃO DE IF’s PARA CHECAR SE MAIS INFORMAÇÕES SÃO VERDADEIRAS).

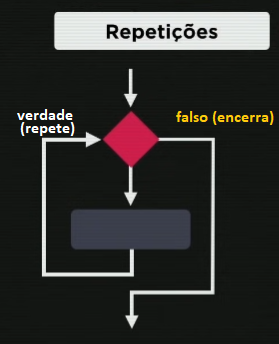
****Usando **else if** este bloco só será avaliado casa um if ou else if anterior seja **falso. (IMPACTO = GARANTE QUE APENAS UMA CONDIÇÃO SEJA VERDADEIRA EM UM SEQUÊNCIA DE ALTERNATIVAS) (USADA = EM CASOS EM QUE APENAS UMA CONDIÇÃO DEVE SER VERDADEIRA).**

****

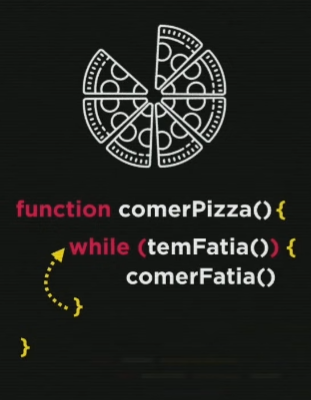
**SWITCH(EXPRESSÃO):** é perfeito para tratar expressões pontuais 0, 1, 2, 3.. ou até mesmo strings pontuais. Para intervalos ou grandes intervalos prefira a estrutura if else. É uma estrutura muito mais limitada que o if else, mas pode ser muito útil em situações específicas.

**Obs:** Funciona para números inteiros (0, 1, 2, 3...) ou caracteres (‘dia’)

**REPETIÇÕES (LAÇOS / ITERAÇÕES):** Igual a um laço que da volta, a estrutura de repetição é pensada de modo a repetir **enquanto** a condição for verdadeira. Quando a condição passar a ser falsa é terminada a repetição.

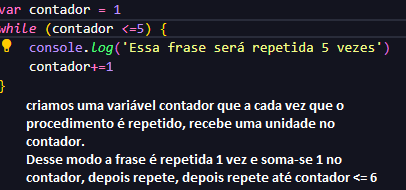
\*while (enquanto) 🡪 **ESTRUTURA DE REPETIÇÃO COM TESTE LÓGICO NO INÍCIO**

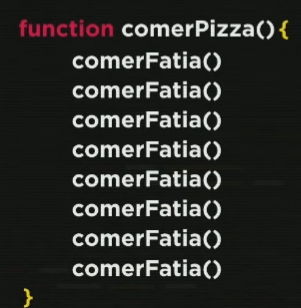


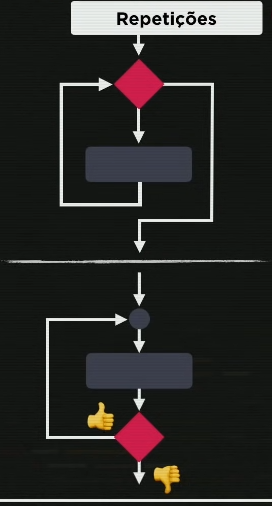
****

**VANTAGEM:** A grande vantagem das estruturas de repetição é que precisamos escrever nossa (condição) do enquanto **apenas uma vez.** Não precisamos escrever comer pizza o número de vezes de pedaços existentes, pois quem **tratará disso** é a condição.

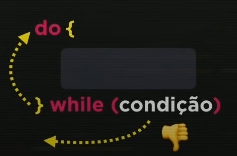
Além disso com while (condição) { } serve para qualquer caso repetível, a function de baixo fica limitada a uma pizza de 8 pedaços.

****



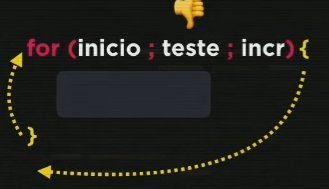
**Outro tipo de teste:** Primeiro executa o bloco e depois faz o teste lógico.

\*do { } wuile (condição) 🡪 **ESTRUTURA DE REPETIÇÃO COM TESTE LÓGICO NO FINAL**

****

**WHILE (CONDIÇÃO { } e DO { } WHILE (CONDIÇÃO) === Essencialmente realizam a mesma tarefa, a diferença é como elas realizam. Sendo while com teste lógico no início e do while com teste lógico no final.**

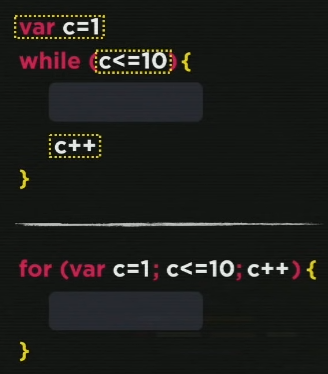
**ESTRUTURA DE REPETIÇÃO COM VARIÁVEL DE CONTROLE (FOR):** É composta por INICIALIZAÇÃO > TESTE LÓGICO > INCREMENTO. A inicialização só é feita a primeira vez, na volta ele faz o teste lógico + incremento, repete até o teste lógico dar falso (false).

A estrutura **for** é melhor que a estrutura **while** quando eu sei os **limites das minhas execuções.**

**\*INICIALIZAÇÃO ===** Variável de controle

\***TESTE ===** Teste lógico

\***INCREMENTO ===** Incremento da variável

****

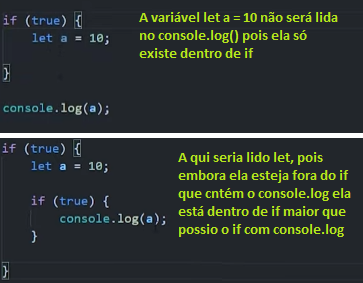
**VANTAGEM:** A grande vantagem de usar for é que o código fica limpo, ou seja, é mais fácil de enxergar seu esquema e de manipular variáveis, testes lógicos e incrementos, pois eles estão **pertinho um dos outros. VIZUALIZAÇÃO FACILITADA.**

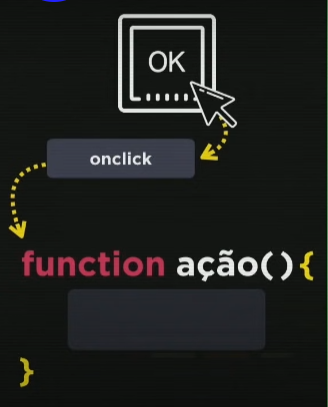
**Obs:** Modo de depuração

**VARIÁVEIS SIMPLES X COMPOSTAS:**

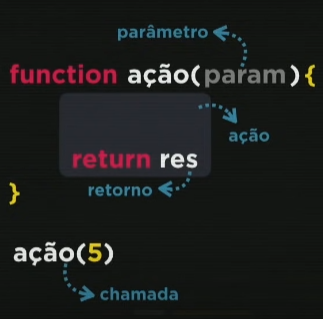
* **SIMPLES:** Só conseguem armazenar **um valor** por vez. Ex: let a = 5, se let a = 8 ela deixa de guardar o 5 e passa a valer 8. A desvantagem de variáveis simples é consumir muito código, ou seja, cada variável precisa ser declarada gerando muitas linhas de código.
* **COMPOSTA:** Consegue armazenar **vários valores** dentro de uma mesma estrutura.Ex: guarda 5, 8, 12...

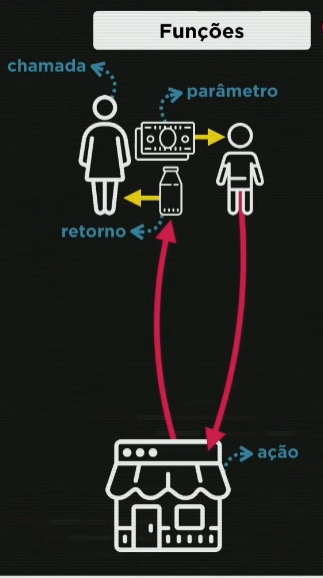
**VARIÁVEIS:** VAR // LET // CONST. **Var e let** é um tipo de armazenamento e manipulação de dado, ambos conseguem ser modificados. Tanto no var quanto no let conseguimos colocar um valor lá dentro e modificá-lo, no entanto a diferença entre **var** e **let** é **escopo. Var = escopo GLOBAL** (funciona em qualquer lugar do programa) **e let = escopo LOCAL** (funciona apenas dentro do bloco onde foi declarada).

**Const:** são constantes de escopo LOCAL e permanecem com o mesmo valor, ou seja, o valor não pode ser reatribuído a uma constante que já foi inicializada. Se ela for do tipo arrey ou objeto ai as propriedades contidas nesses arreys e objetos podem ser modificadas.

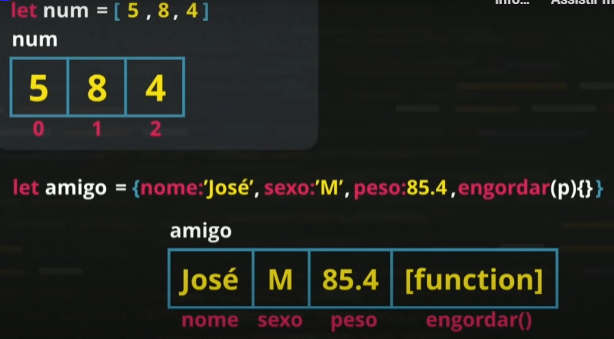
**FUNÇÕES:** São **ações** executadas a partir de **chamadas** ou em decorrência de **eventos**. Uma função pode receber **parâmetros** e retornar um **resultado.**

**DISPARO COM EVENTO:** Podemos, por exemplo, disparar uma função colocando dentro da tag input:button do HTML um evento onclick=”calcular()”, ou seja, um evento, que será lido via DOM pelo JS. Desse modo, toda vez que tivermos esse evento ocorrendo no HTML, será disparada a função calcular() { }.

**DISPARO COM CHAMADA:** Para uma função ser disparada com uma ação, ela vai no final da função com o valor do parâmetro.

Caso a função seja criada e esteja sintaticamente correta, ela não funcionará, pois precisa da chamada lá em baixo.

**OBJETOS (OBJECTS):** SÃO MAIS EVOLUIDOS QUE OS ARREYS. Enquanto arreys servem para resolver coisas simples, pois ficam com índices sempre fixos e toda vez que precisamos chamar determinado valor (precisamos manualmente escrever o índice). Os **objetos** permitem uma criação e manipulação mais completa, complexa e automatizada. Podemos guardar valores dentro de um objeto declarado acrescentando a esses “valores” subcategorias da seguinte forma.

Os objetos são declarados com **{ }**

**Let objeto = { }**

Inclusive é possível colocar funções dentro de uma cápsula de objeto. Portanto objetos podem guardar **funcionalidade (functions) que são chamados de métodos.**

**this:** Para chamar a propriedade de um objeto dentro dele mesmo usamos this.propriedade

constOBJETOCARRO={

*marca*:'ford',

*modelo*:'ka',

*ano*:2015,

*placa*:'ABC1234',

*preco*:38900,

*buzina*:function(){alert('Biiiii')},

*completo*:function(){

*return* `A marca é ${this*.*marca} e o modelo é ${this*.*modelo}`;//*Para pegar uma propriedade dentro do próprio objeto se usa this*

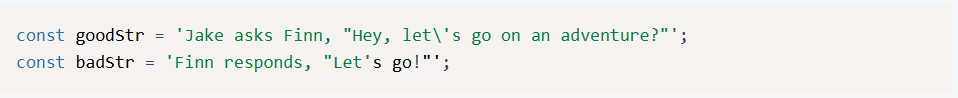
}

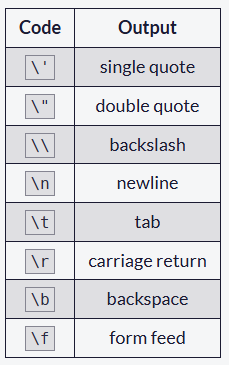
}

**Boas práticas de JS:**

* **Nomes de variáveis:** camalCase (primeira palavra minúsculas, depois todas iniciais em maiúsculas).

**Var** thisVariableNameIsSoLong;

* **Nomes de variáveis: const (**MAIÚSCULAS ou camelCase)
* **Aspas dentro de aspas:** Para funcionar basta colocar barra invertida para avisar ao JS
* const sampleStr = "Alan said, \"Peter is learning JavaScript\"."
* 

****

**Comandos JavaScript:**

**window.alert(‘ ‘):** retorna pop up com texto ‘entre aspas’ + botão OK.

**window.confirm(‘ ‘):** Retorna pop up com texto ‘entre aspas’ + botão OK / CANCELAR.

**window.prompt(‘ ‘):** Retorna pop up com característica **string** ‘entre aspas’ + campo (prompt) para digitar + botão OK / CANCELAR.

**document.write(` `):** Escreve no documento a string na mesma linha.

**document.writeln(` `):** Escreve no documento a string com quebra de linha.

**Number.parseInt(n):** Converte uma variável tipo Number para Number Inteiro. (usado para forçar inteiro).

**Number.parseFloat(n):** Converte variável tipo Number para Number Real. (Usado para forçar Real).

**Number(n):** Converte variável Number de acordo com valor recebido, de acordo com o contexto que JS achar correto.

**String(n):** Converte variável (n) para String.

**n.toString( ):** variável (n) convertida para String.

**+:** concatena e **${n}:** Template Float.

variável**.length:** Quantos caracteres a string tem.

variável**.toUpperCase( ):** Passatudo para ‘MAIÚSCULA’

variável**.toLowerCase( ):** Passatudo para ‘minúscula’

variável.**toFixed(n):** Fixa quantidade de casas decimais, n = quantidade de casas decimais.

Variável.**replace(‘isso’,’aquilo’):** A partir de uma variável, troca o primeiro valor pelo segundo.

variável.**toFixed(n).replace(‘.’,’,’):** Fixa quantidade de casas decimais, n = quantidade de casas decimais e o replace é: troque ‘isso’ por ‘isso’

variável.**toLocaleString(‘pt-br’, {style: ‘currency’, currency: ‘BRL’}):** Chama a string em valor monetário de acordo com a região (para isso podemos trocar ‘BRL’ por ‘USD’)

**id.addEventListener(‘click’, ação):** Adiciona lista de eventos que anteriormente precisavam ser colocados dentro da tag. (esse campo pode ser colocado em baixo das variáveis globais)

**function ação ( ) {**

**id.innerText = ‘Voltar’**

**}**

**document.write( ):** Escreve a sentença no body. Usado para web

**console.log (‘ ‘):** Retorna o texto escrito no terminal ou no developer tool da web.

**new Date( ):** Permite buscar **QUALQUER** data do seu sistema (data, dia, ano, horas, milissegundos, minutos, mês, segundos, tempo...) adicionando a seguinte estrutura **get.Date // get.Day // get.FullYear // get.Hours // get.Milliseconds / get.Minutes // get.Month // get.Seconds // get.Time // get.Time...).** Os valores que esses scripts retornam é onde eles estão rodando (servidor, no cliente, no seu sistema, entre outros).

Let num = [2, 3, 4]

**num[3] = 6**: Muda o elemento de índice 3 do array de “4” para “6”

**num.push(7):** Cria um novo índice (posição) elemento dentro do arrey num e acrescenta o valor 7

**num.length:** Encontra o número de elementos do vetor num

**num.sort():** Ordena os valores dentro dos elementos de um arery para ordem crescente

**num.indexOf(7):** Retorna o índice de onde se encontra o valor “7”.

**Array.isArray(nome do array) =** Se for array retorna true, se não for retorna false

**nomedoarray.shift( ) ==** Remove o primeiro item do array e muda os índices

**nomedoarray.unshift( ) ==** Adiciona o primeiro item no array e muda os índices.

**nomedoarray.pop() ==** remove o último item de um array (matriz)

**nomedoarray.push(‘novo item’) ==** Adiciona o item na última posição do array

**delete nomedamatriz[2] ==** Deleta o valor do item de índice 2 (3° item) da matriz e deixa o valor undefined dentro dele sem excluir a posição.

**nomearray.splice(0, 1, ‘novo item 1’, ‘novo item 2’ ) ==** escolhe o índice, diz quantos itens serão eliminados a partir dele e escreve novos itens

**ARRAY1.concat(ARRAY2);** método nativo do js para concatenar e juntar listas, primeiro é exibida a lista 1 completa, depois em seguida lista2.

**nomedoarray.slice(2, 6) ==** Recorta a partir do índice 2 para frente até chegar no 6 e escvreve a fatia de índice 2 até o índice 6, se não escrever o ‘, 6’ ele mostra tudo de índice 2 para frente.

**nomedoarray.sort( ) ==** Coloca o array (matriz em ordem alfabética se for string ou ordem crescente **do TEXTO DO primeiro número** se for number. 1, 10, 105, 25, 29, 31, 40.

**nomearray.filter(fuçãoqualquer) ==** Filtra a matriz com determinada função.