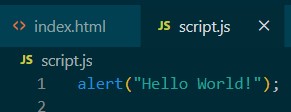
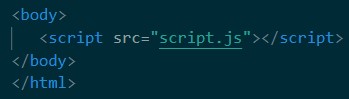
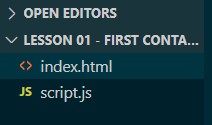
**JAVASCRIPT BÁSICO (IMPORTANTE)**

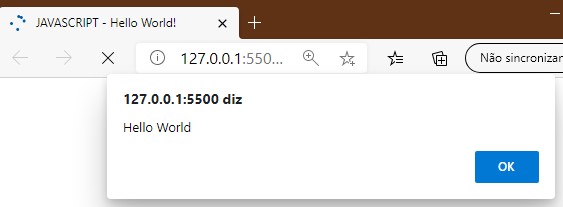
➤ **PRIMEIRO CONTATO (HELLO WORLD):** No javascript, em vez de criar um arquivo style.css, que é o caso do css, você vai criar um arquivo: **script.js**! (FORMA EXTERNA!)

➜ **FORMA EXTERNA:** Aqui, você precisa criar o arquivo javascript separado, e chamar ele no HTML, assim como faz no CSS.



**OU**

➜ **FORMA INTERNA:** Você pode utilizar o javascript INTERNO, como se fosse o CSS interno, dentro do próprio HTML:

**OBS:** Esse “ALERT” abrir uma janela assim que você acessa a página, com o que você quiser dizer.

➤ **Algorítmos:** É uma sequência de passos para você realizar uma tarefa. E o computador funciona à base de algoritmos. Então a ORDEM deles influencia bastante, precisa seguir a sequência. O computador é burro, então precisa ter toda uma ordem de variaveis, para diferentes casos, EX:

Se o queijo estiver fechado, abrir o queijo. (FIM)

Pegar a faca.

Abrir o pão com a faca.

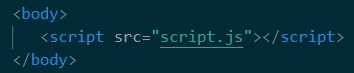
Cortar um pedaço do queijo com a faca.

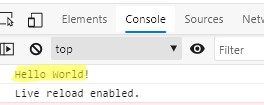
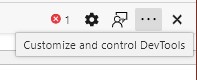
Passar o queijo na parte interna do pão com a faca.

Largar a faca e comer o pão.

Então isso é o algoritmo, são passos finitos em uma ordem pré-definida que vai resolver algum tipo de problema, o algoritmo não pode ter BUGS, então sempre que tiver, você vai ter que ajustar até ele fizer justamente aquilo que você queria que ele fizesse. Às vezes o BUG é apenas uma ordem ERRADA!

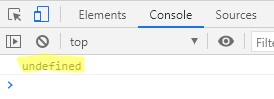
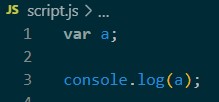
➤ **CONSOLE:** Aqui você vai poder verificar as mensagens com o INSPECIONAR → CONSOLE do google chrome.



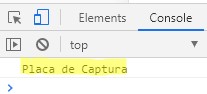
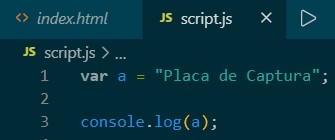


➤ **VARIÁVEL:** A variável serve para a gente conseguir lidar com dados sem saber exatamente o que esse dado é, ou seja, o dado varia, mas mesmo assim conseguiremos lidar com ele.

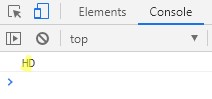
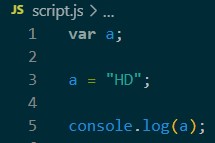
Enquanto você não especificar o que é a variável A, por exemplo, ela será indefinida:



Você pode especificar o que tem dentro da variável:

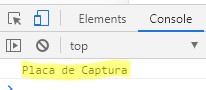
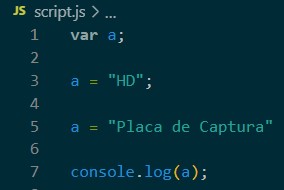


**OU**

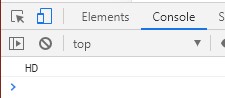
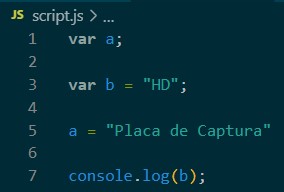
****

**OBS:** No JAVASCRIPT, o ponto e vírgula é opcional, mas é bom colocar para sempre seguir um padrão.

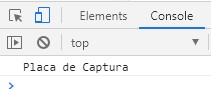
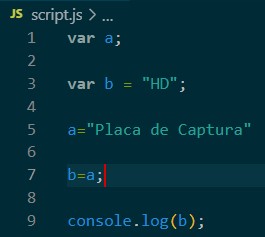
⧫ E se a variável A receber 2 comandos? O que acontece?

 Ela adota o ÚLTIMO!

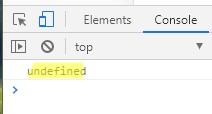
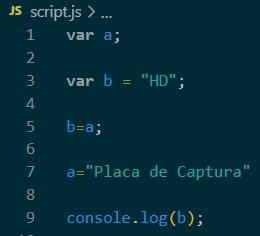
**OBS 2:** Você pode ter várias variáveis, mas ela só vai imprimir a que você mandar imprimir.



⧫ Mas se novamente, você colocar 2 dados, dizendo que B recebe A, ele lerá o último:

 A ORDEM É CRUCIAL!

⧫ Outro exemplo:

 O A era indefinido, quando o B foi lido.

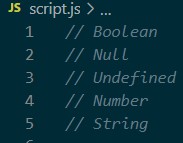
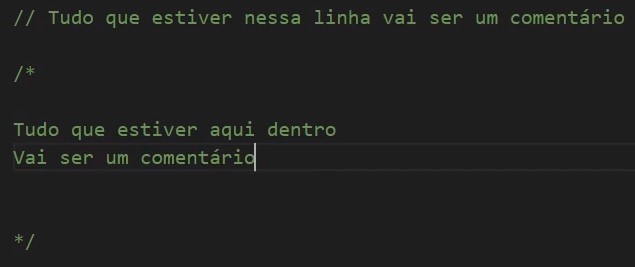
**OBS 3: NOMES:** A variável pode receber qualquer nome, mas é bom relacioná-lo ao que você quer dizer, ao invés de “a”, a variável poderia se chamar, “caixa”, ou “caixa1”, mas não pode COMEÇAR com número, pode também usar o nome: “caixa\_$” mas o $ não é indicado a usar, pois ele será muito usado no JQUERY(?). Mas no mais é isso, usa-se: LETRAS/NÚMEROS/UNDERLINE/CIFRÃO.

Para nomes compostos é indicado usar: “caixaGrande” ou “caixa\_grande”. NÃO USAR: “caixagrande”.

É PADRÃO, os nomes das variáveis se iniciam com MINÚSCULO, mas não existe impedimento.

AQUI RESPEITA-SE OS MAIÚSCULOS E MINÚSCULOS, se usou tudo minúsculo não pode mudar.

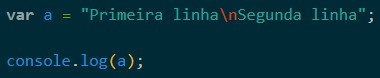
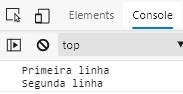
➤ **TIPOS DE DADOS:** São 5 os tipos de dados, e nós vamos entender cada um deles, mas antes só precisa saber o seguinte:

Essas duas barras significam que você fez um comentário, e quando o computador estiver processando o nosso código, ele vai ignorar todo o comando que estiver na frente dessas duas barras, a outra forma de fazer comentário é a mesma que aprendemos no HTML e CSS: /\* → no início e \*/ → no fim.

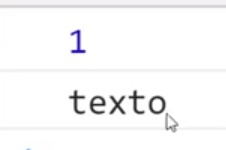
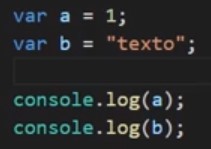
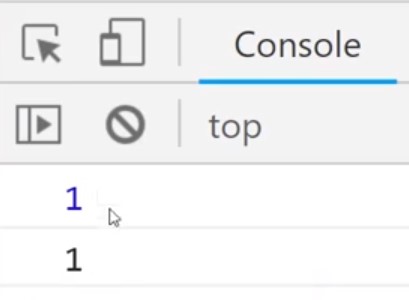
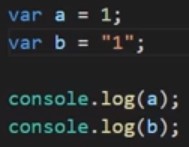
➜ **STRING:** É uma cadeia de caracteres contida entre ASPAS, ou seja, “abc” é uma string pois ela é um caractere do lado do outro que forma uma palavra, então palavras, frases, textos inteiros, são STRINGS, são cadeias de caracteres, e um desses caracteres é o ESPAÇO, ou o caractere NENHUM, então mesmo fazendo isso, estaremos criando uma STRING, mesmo não tendo nada, é uma cadeia de caracteres com zero elementos:   , etc.

**OBS: \n:** É o <br> do HTML/CSS, ele vai fazer pular linha entre os dois elementos:

**\ = (Alt+92)**

➜ **NÚMEROS:** Também podem ser usados na variável, mas ficam de cor AZUL, pois diferentemente da STRING que tem a cor preta, cada um tem a sua função.

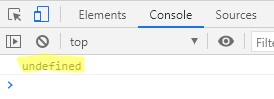
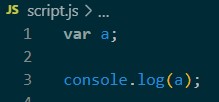
 

⧫ E QUAL A DIFERENÇA DE NÚMEROS PARA STRING?

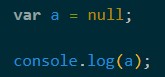
**NA STRING:** 

**NOS NÚMEROS:** 

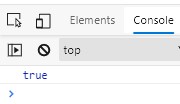
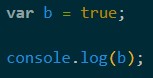
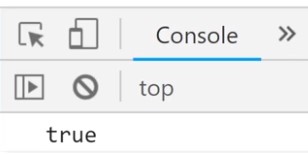
➜ **UNDEFINED:** É quando eu não inicializo a variável, e o nosso programa não sabe o que tem ali dentro, então ele fica INDEFINIDO:



➜ **NULL:** É a certeza de que não TEM NADA DENTRO daquela variável! Ela está vazia.

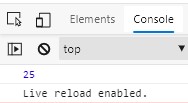
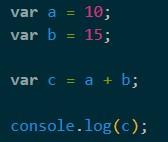
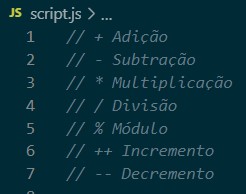


➜ **BOOLEAN:** É o TRUE ou FALSE, também aparece na cor azul para diferenciar das STRINGS que são pretas.

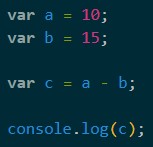
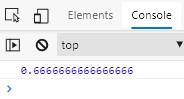
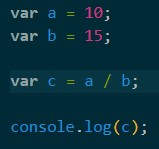
 STRING: 

O boolean te permite fazer operações lógicas, você consegue verificar se uma coisa é verdadeira ou não e com isso tomar uma opção.

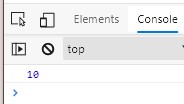
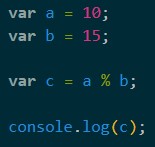
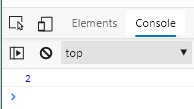
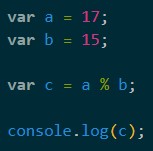
➤ **OPERAÇÕES ARITMÉTICAS:** As operações que fazemos com JAVASCRIPT, são:

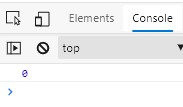
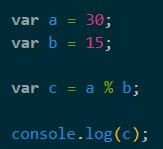


As variáveis podem ser positivas, negativas, podem ter números quebrados, não importa!

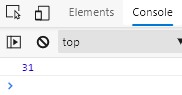
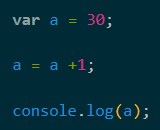
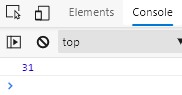
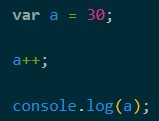
 **OU**

➜ **MÓDULO:** Ele tenta fazer uma divisão, e retorna o que SOBROU de uma forma que não tenha números quebrados. Ele retorna o valor que SOBRA da divisão, então como 10 dividido por 15 não sobra nada, pois 15 é maior que 10, ele devolve o número sem ser dividido, caso a “var a” fosse maior que a “var b”, ele tiraria o valor de B e retornaria o que sobrou:

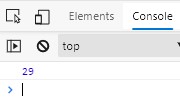
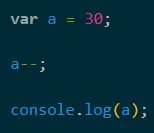
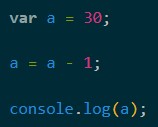
**OU**

****Aqui dá 0, pois a divisão dá 2 e não sobra NADA.

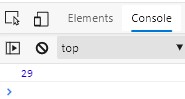
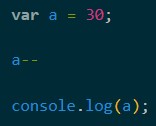
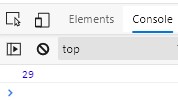
➜ **INCREMENTO:** Quando eu faço incremento de A, quer dizer que eu estou pegando o A e eu estou colocando MAIS um nele, mas escrevendo na forma de incremento fica mais fácil:

**OU**

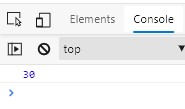
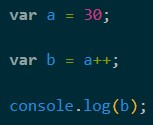
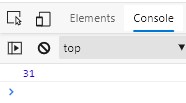
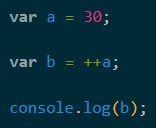
➜ **DECREMENTO:** É a mesma coisa que o “incremento”, só que NEGATIVO:



**OBS 3: ORDEM DOS SINAIS:** Os sinais podem vir tanto na frente quanto atrás, quando estivermos tratando de apenas UM elemento, mas a ordem influencia quando estamos tratando de mais de um:



**DIFERENTE DE:**

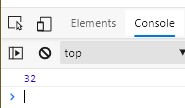
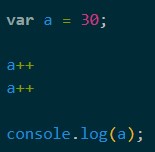
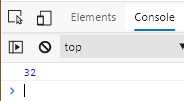
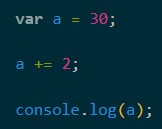
****

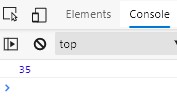
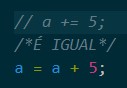
Na programação a ORDEM IMPORTA MUITO!

➤ **ATRIBUIÇÕES:** As atribuições são conhecidas por:

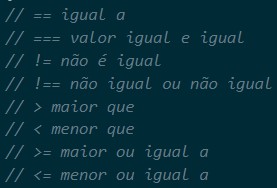


E se você quisesse incrementar o “A” + de uma vez? E não precisar digitar mais de uma vez para isso?

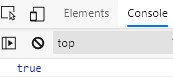
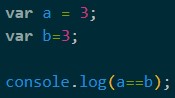
**OU**

**OU SEJA: **

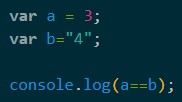
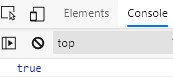
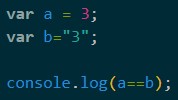
➤ **OPERADORES COMPARATIVOS:** Servem para comparar um dado com o outro, e ele retorna se a comparação é verdadeira ou falsa. É uma operação do tipo: **BOOLEAN.**

****

➜ **==:** Se você colocasse apenas um igual, você estaria falando que um valor recebe o outro, e não perguntando se um numeral é igual ao outro, então no javascript usa-se 2 iguais:

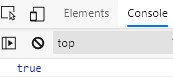
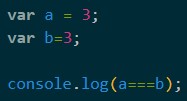
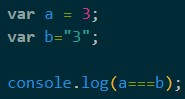


**OBS: STRING:** E se você perguntar se o número 3 é igual à PALAVRA “3”? Afinal, como vimos, os números entre aspas são strings, e significam cadeias de significado.

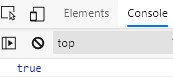
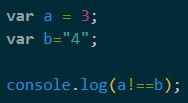
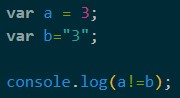


E como mudar isso? Afinal, números não são iguais a palavras, então deveria ser errado nas duas opções.

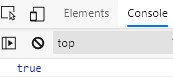
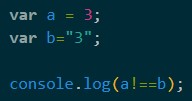
➜ **===:** Nesse caso, comparamos não apenas o valor para ver se é igual, mas comparamos também o TIPO, então nesse caso, 3 = 3, mas o 3 número e o “3” STRING são DIFERENTES.



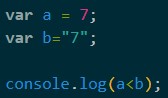
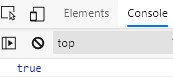
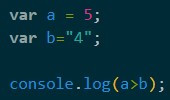
➜ **!=:** Nesse caso, perguntamos se uma variável é diferente da outra:



➜ **!==:** Assim como no igual, precisamos de um IGUAL A MAIS para comparar VALOR E TIPO.

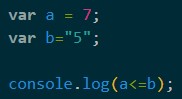
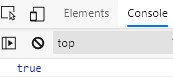
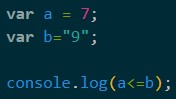
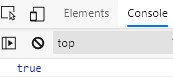
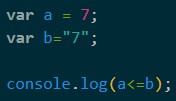


➜ **> ou <:** O maior que (>) e o menor que (<) apenas executam seu papel.

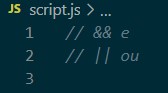


**OBS: NÚMEROS IGUAIS:** Mas 7 não é menor nem maior que 7, como diferenciar?

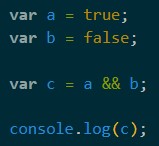
➜ **>= ou <=:** Para números iguais, é melhor usar o MAIOR ou IGUAL a:

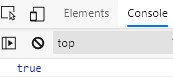
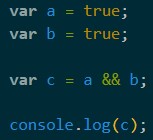


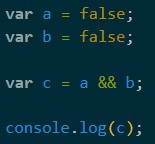
➤ **OPERADORES LÓGICOS:** Outros Operadores

 → **| = (Alt+124)**

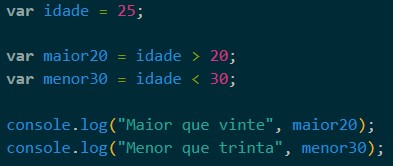
➜ **&& (“e”):** Ele compara dois valores booleanos, se ambos forem verdadeiros, ele retorna verdadeiro, e qualquer coisa diferente disso, ele retorna FALSO, basta uma delas ser falsa, já dá FALSO:

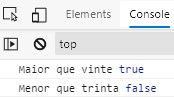
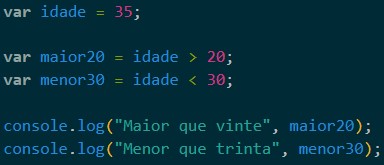
 → “A” é true e “B” também é true? FALSO!

 → “A” é true e “B” também é true? VERDADEIRO!

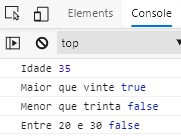
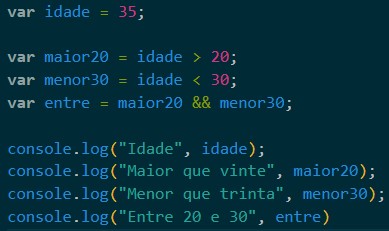
 → “A” é true e “B” também é true? FALSO!

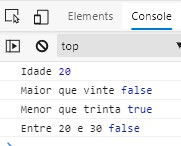
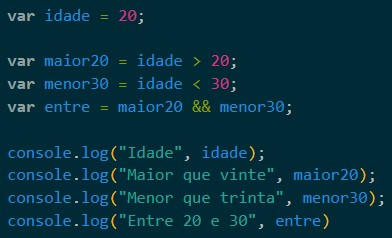
**OBS: Exemplo:** um comparativo de idades:



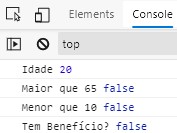
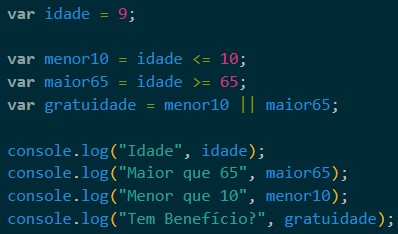


⧫ Usando o comparativo && (“e”):

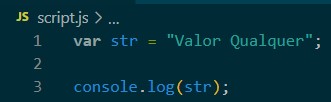


 Precisaria do “>=” p/ true.

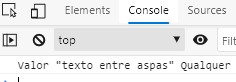
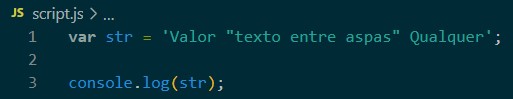
➜ **|| (“OU”):**  **| = (Alt+124):** Se um dos dois for verdadeiro, o “OU” será verdadeiro. Ele só será falso se ambas as variáveis forem FALSAS.

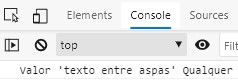
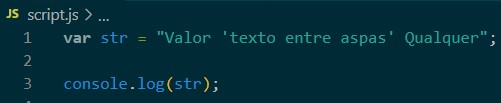


➤ **STRINGS:** No Javascript, você pode criar STRINGS, tanto com aspas duplas, ou aspas simples:

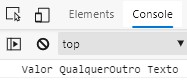
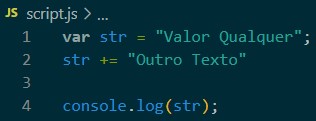


**OBS: Texto entre “Aspas”:** Mas se você quiser usar alguma coisa entre aspas, DENTRO da STRING, você vai precisar usar aspas simples para ativar a STRING e depois usar aspas duplas dentro dela, ou vice-versa:

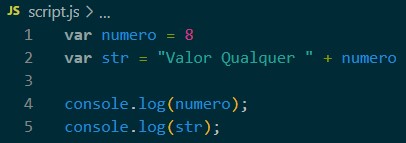




➜ **Adição:** Na STRING, a adição vira uma concatenação, ou seja, ela concilia as palavras, juntando elas, por exemplo:

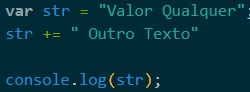
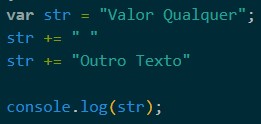
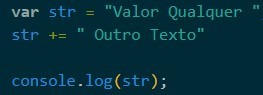


⧫ E se você tivesse adicionado um número? Ele seria azul, como ocorre com os números?

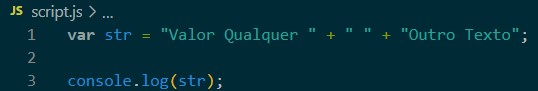


O número somado com a string ficou na cor preta, pois ele se JUNTOU a ela, e a String é sempre preta.

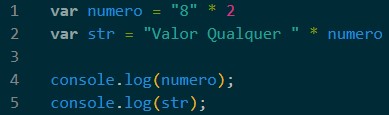
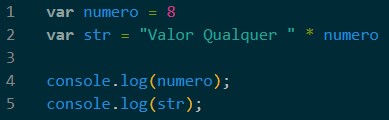
**OBS: ESPAÇO:** Para dar espaço entre as palavras, você pode fazer:



**OBS 2: ÚNICA VARIÁVEL:** Poderia fazer tudo em uma variável só, também:

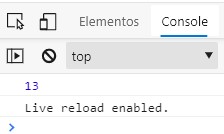


**OBS 3: Outras Operações:** A STRING só funciona com a adição, se você tentasse alguma outra coisa, daria o erro NAN, ou seja: Not a Number. A não ser se a sua String fosse representante de um número:

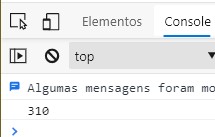
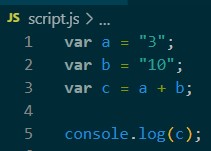


Observe que depois da multiplicação da string “8” pelo 2, ele virou realmente um número e ficou em azul.

➤ **TRANSFORMANDO STRINGS EM NÚMEROS:** Veremos que de vez em quando podemos ter um comportamento que não é exatamente do jeito que a gente queria, e como podemos resolver isso com “parseInt” e “parseFloat”.

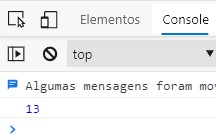


Quando temos números e somamos, fica tudo certo, o console vai entender e vai somar, mas lembra que quando falamos de STRING e na STRING nós temos a concatenação? E o que aconteceria caso meus números fossem STRINGS?

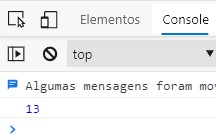
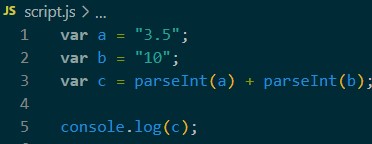
 Aconteceria a “CONCATENAÇÃO”

E como podemos tentar resolver isso?

➜ **“parseInt”:** Faz com que a string vire NUMERAL deixando adicionar em vez de CONCATENAR!

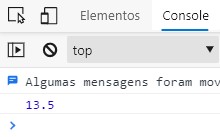
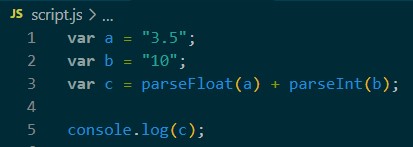


Mas e se fosse um número quebrado? Será que o parseInt iria me ajudar?



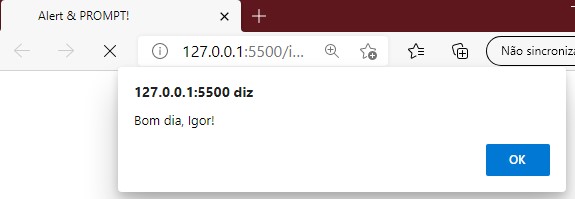
Não ajudaria, pois o “parseInt” não lê números quebrados pois parseInt significa números INTEIROS.

➜ **“parseFloat”:** Calcula também os números QUEBRADOS, sem ser inteiros.

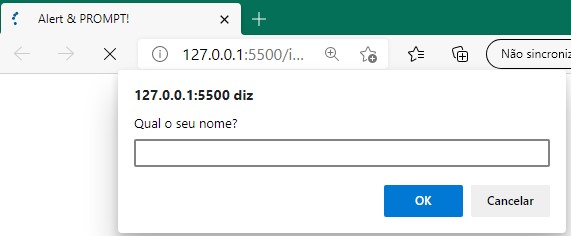


➤ **ALERT & PROMPT:** Novas ferramentas para serem usadas ao invés do console.log.

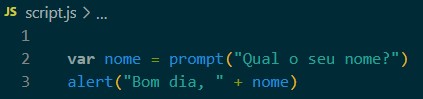
➜ **“ALERT”:** Ele emite uma mensagem para o usuário que acessa a sua página.

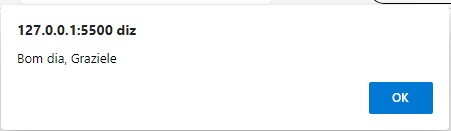


➜ **“PROMPT”:** Já o promt abre uma caixa de diálogo, onde você pode responder o que for perguntado.

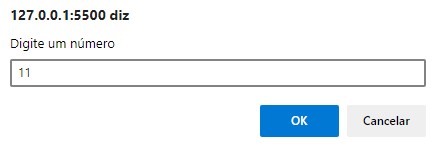
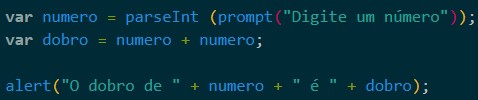


Com isso, você pode criar códigos, onde utiliza essa resposta em alguma outra coisa:



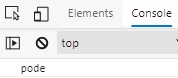
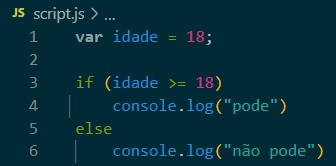


E existem milhares de coisas possíveis de se fazer com esses dois comandos, como por exemplo:



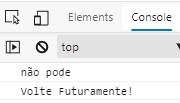
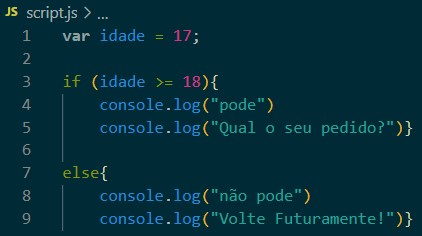
➤ **ESTRUTURAS CONDICIONAIS:** São estruturas que checam certa condição e tomam um determinado caminho dependendo dessa condição, ou seja, se uma coisa for verdade, você faz uma coisa, mas se essa coisa não for verdade, você faz outra coisa → SISTEMA **BOOLEAN**.

Vamos supor um site que vende bebida alcoólica:

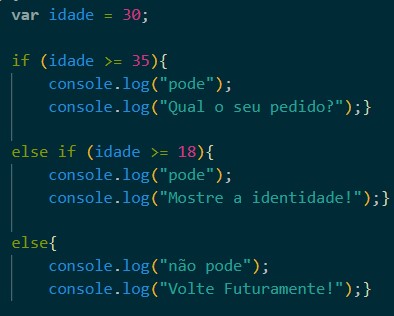


Aqui, colocamos uma condicional SE e SENÃO. SE for maior de 18 anos, pode comprar, SENÃO, não pode.

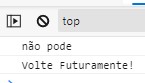
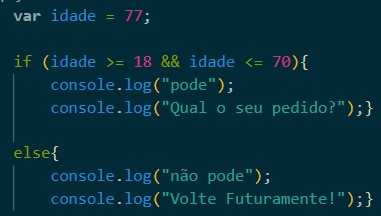
**OBS: MAIS DE UM COMANDO:** Quando você quer dar mais de um comando, você precisa colocar os comandos entre chaves, pois é uma CADEIA DE COMANDOS a ser seguida:



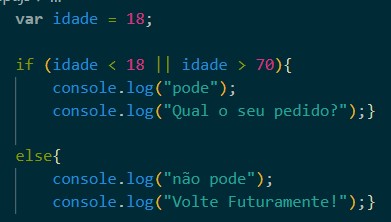
**OBS: MAIS cadeias de comando:** Agora eu quero 3 opções diferentes, se for menor de 18, não pode comprar bebidas, se for de 18 até 34, precisa mostrar a identidade, e acima de 35, é livre para comprar.



⧫ E se em outro caso hipotético, você só pudesse vender bebidas para maiores de 18 e menores que 70?



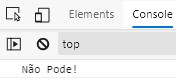
⧫ E com o “OU” → || ???? Obs: Essa loja só vende para menores de 18 e maiores de 77:



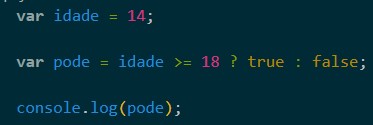
**OBS: RECAPITULANDO:** ➜ **&& (“e”):** No “e”, ambas as sentenças precisam ser verdade, para ser true.

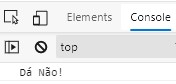
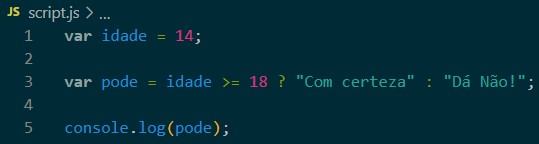
➜ **|| (“ou”):** No “ou”, apenas uma precisa ser verdade, para ser true. Para false, ambas precisam ser falsas.

➤ **OPERADOR TERNÁRIO:** Faz a mesma coisa que o “IF” e o “ELSE” mas de uma forma mais simples, em uma única linha, com símbolos específicos:



**OU**

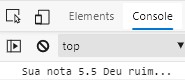
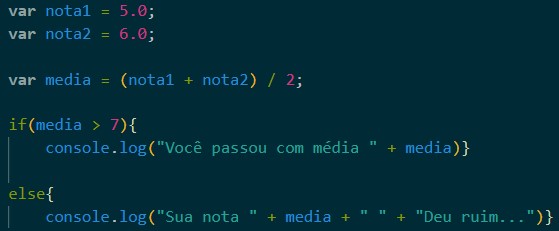
****

****

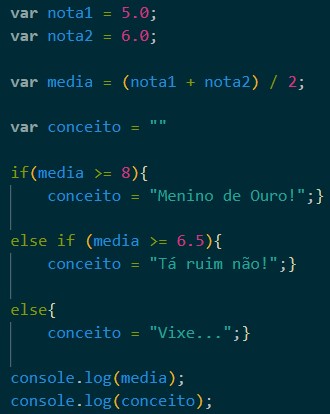
⧫ Você pode até mesmo escrever a resposta que você quiser dar, sem ser a variável original.

➤ **CONDICIONAL SWITCH:**

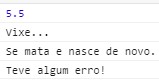
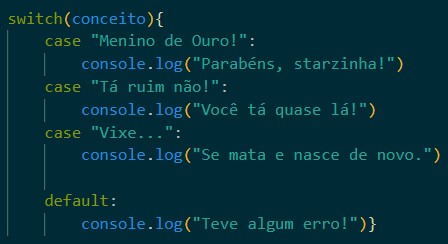
Vamos supor que queremos saber se um aluno passou ou não de ano:



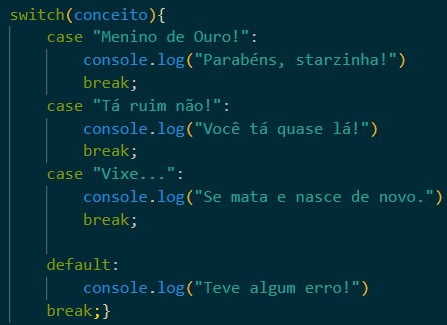
⧫ E caso quiséssemos dividir os alunos entre: REGULAR / BOM / ÓTIMO?



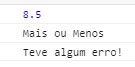
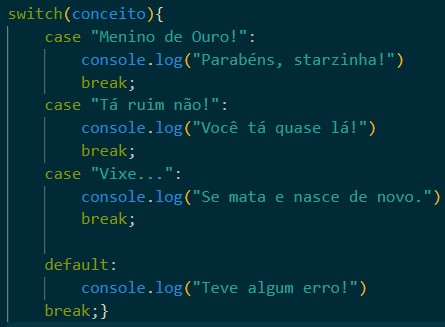
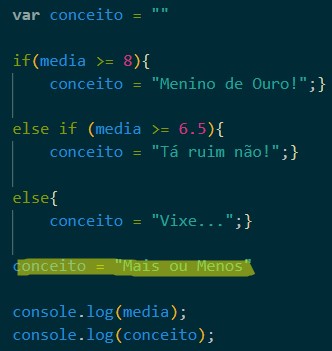
⧫ **ANOTHER WAY:** USANDO O **SWITCH**:



Repare que depois que ele imprimiu o “case Vixe”, ele continuou imprimindo os debaixo, para não deixar isso acontecer, você pode utilizar o **BREAK**:



**OBS: Forçando uma outra resposta:** Aqui, eu posso forçar o meu conceito a ter outra resposta do que as estipuladas, fazendo dar um erro, vejamos:

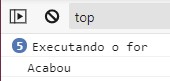
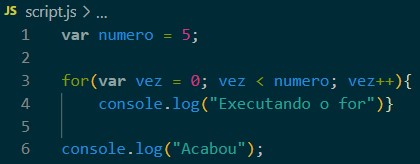


Então como a resposta não foi nem “Menino de Ouro”, “Tá ruim não” e nem “Vixe…”, foi o conceito forçado “Mais ou Menos” que não estava previsto no SWITCH, deu a resposta do “DEFAULT”.

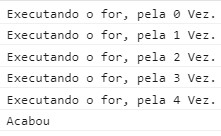
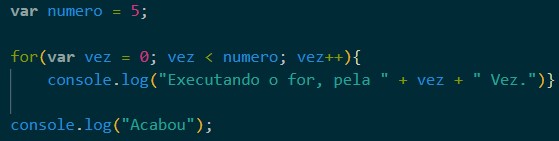
➤ **REPETIÇÃO “FOR” e “WHILE”:**

➜ **“FOR” (PARA):** Aqui, vemos um laço com repetição determinada, ou seja, eu sei exatamente quantas vezes a instrução vai ser repetida.

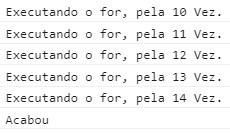
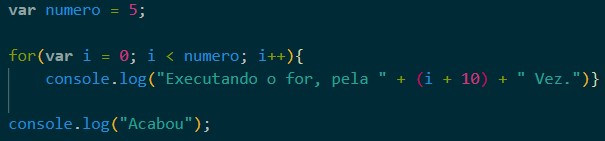
Podemos fazer um comando, para o servidor imprimir quantas vezes mandarmos ele imprimir, e ele avisar depois que acabar de imprimir. Vamos supor que queremos que ele imprima 5 vezes:



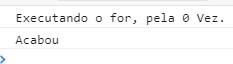
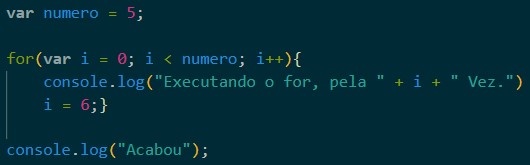
⧫ No google chrome, ele coloca quantas vezes você mandou imprimir, avisando que foram 5 vezes, mas aqui ele agrupa as 5 vezes em uma só, você poderia querer ver ele falando todas as vezes separadamente:



⧫ E o céu é o limite, você pode brincar com o código como quiser:

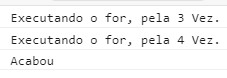


⧫ Causando um erro: Como você tinha pré-determinado que o i é menor do que 5:

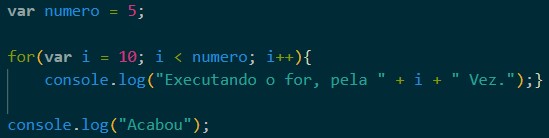


Quando ele for maior do que 5, só vai imprimir a primeira vez e depois parar o looping.

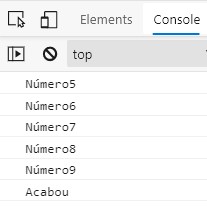
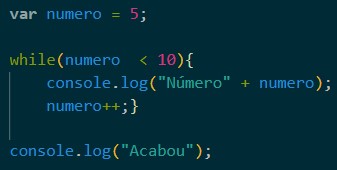
⧫ Posso também já começar da impressão número 3, por exemplo:



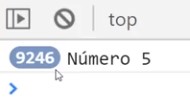
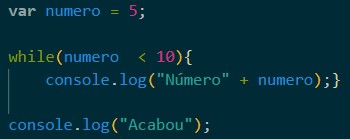
⧫ E se a impressão começar no 10, que é um número maior do que 5?

 Não vai imprimir nenhuma vez!

➜ **“WHILE” (ENQUANTO):** Pode ser operado de duas formas, como repetição determinada ou como repetição indeterminada. O “WHILE” fará uma verificação como se fosse o “IF”, e se a função for verdadeira, ele vai fazer o que tiver ali dentro. Então ele será bem parecido com o “IF”.



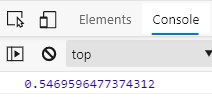
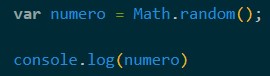
Nesse caso você queria os números MENORES que 10, e começou do número 5, então ele imprimiu do 5 ao 9, se tivesse colocado <= imprimiria também o número 10.



Caso não colocássemos o “número++” ele ficaria imprimindo o número 5 em um looping infinito, para sempre, até o computador ficar sem memória. Pois o 5 é menor do que 10.

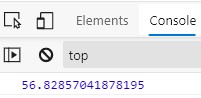
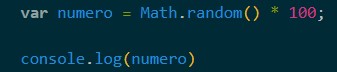
Então tome cuidado para não entrar em Looping Infinito, pois quando estamos trabalhando, esse tipo de erro faz criar um bug que em algum momento faz com que entremos em um looping infinito, causando problemas.

**OBS: “Math.random()”:** Faz criar um número random entre 0 e 1, toda vez que você atualiza a página, esse número muda!

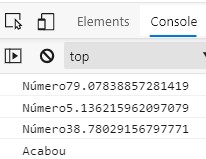
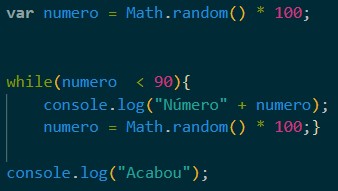


Precisa ter os parênteses depois do random, mesmo que vazios, ou não vai funcionar.

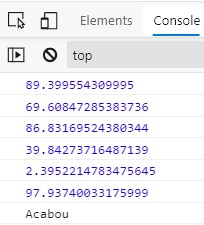
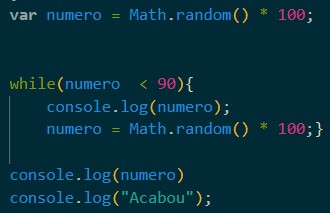
E se eu quiser que o número seja entre 0 e 100?



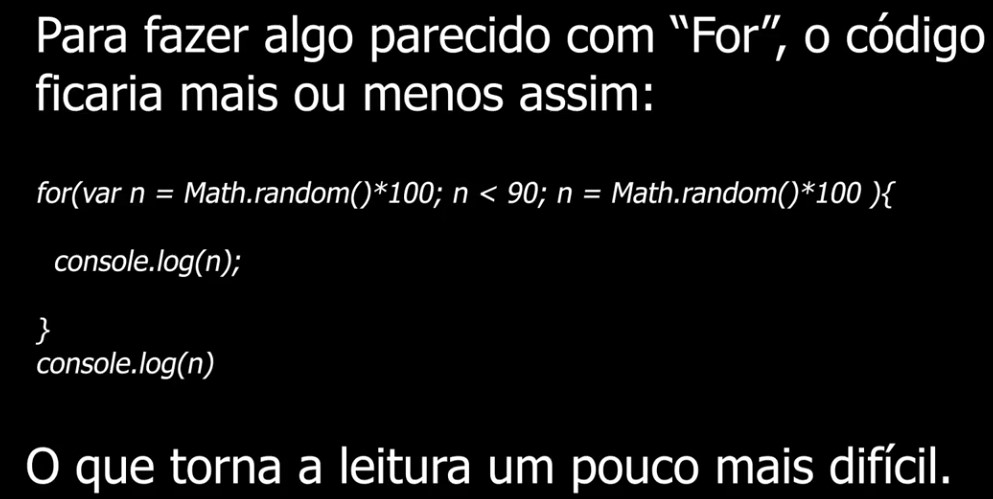
Também poderíamos juntar essa multiplicação com o “WHILE”:



Mas nesse caso que colocamos, ele imprimiria só os que saíssem menores do que 90 que foi o que estabelecemos, e quando vem um maior ele PARA! Se quiséssemos que ele continuasse imprimindo:

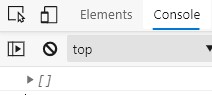
Lembrando que vão de 0 até 100.

**OBS: Fazendo com “FOR”:** Também é possível realizar isso com “FOR”, mas é mais DIFÍCIL:



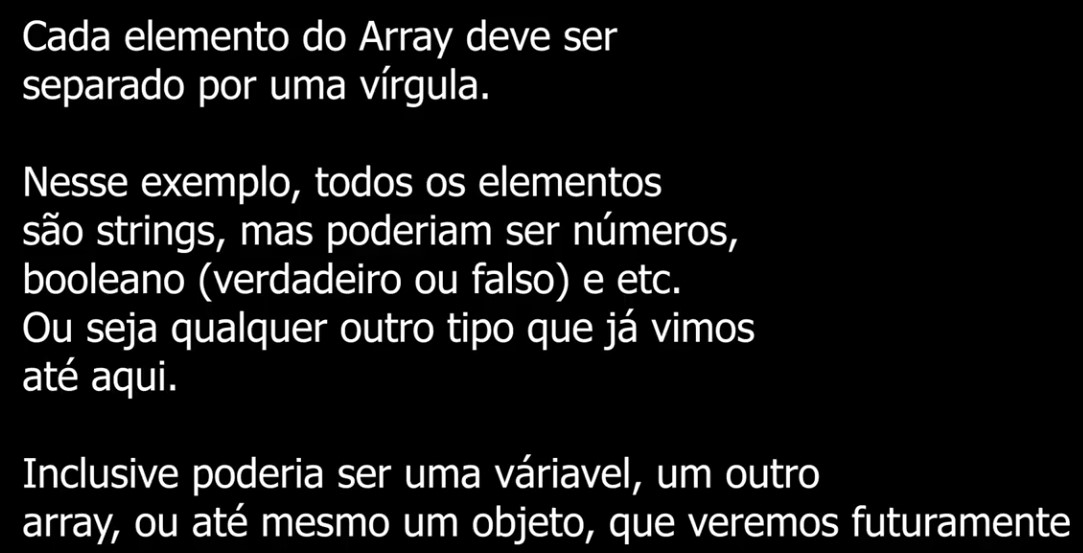
Com o “FOR” nós vamos do ponto “A” até o ponto “B” e para isso, nós precisamos saber qual é o ponto “A” e qual é o ponto “B”. Nesse caso do “WHILE” nós não sabemos, não sabemos quantas vezes ele vai imprimir no “CONSOLE” até que as condições sejam verdade. Pode ser na primeira vez, ou na 10º vez, por exemplo.

➤ **ARRAYS:** É uma estrutura de dados que pode armazenar vários dados em sequência, ou seja, é um conjunto de variáveis sequencial que começa sempre pelo número 0, e vai até o último elemento -1, ou seja, se temos 30 elementos, para chamar o último, você digita 29.

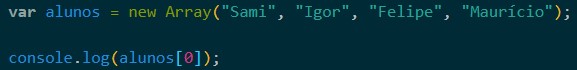


⧫ Vamos supor uma sala com quatro alunos:

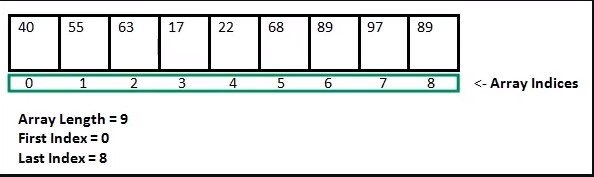


**OBS: **

⧫ Agora vamos supor que queremos chamar apenas um dos alunos:

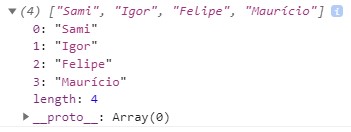


⧫ Porque escrevemos 0? O “Zero” é a primeira posição do Array. Todo array começa na posição zero.

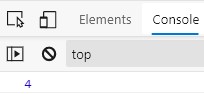
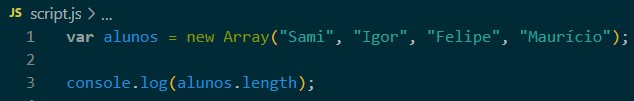


Neste outro exemplo usaram números, mas é a mesma coisa, se você quisesse chamar o maurício, deveria escrever o número 3 para chamá-lo.

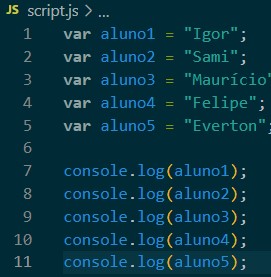
**OBS:** Clicando na setinha do lado do Array, você vê uma tabelinha mostrando todos os elementos do Array, mais um contador dizendo quantos elementos esse Array possui:



⧫ Ou você pode pedir para imprimir apenas o tamanho do “ARRAY”:

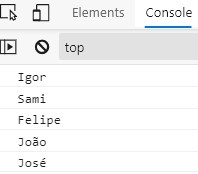
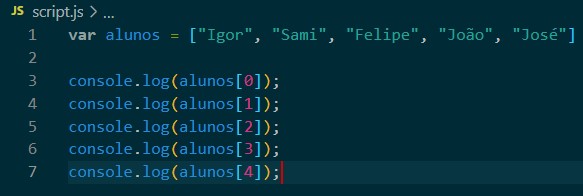


⧫ SEM O ARRAY, a vida seria bem mais complicada:

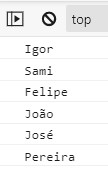


Com pouco alunos, até conseguiríamos fazer, mas e caso tivéssemos 100 alunos?

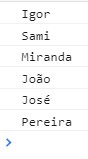
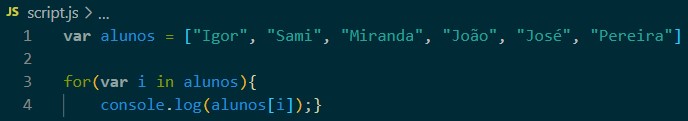
Então fazemos com array:



⧫ **USANDO ARRAY COM “FOR”:** Fica muito mais fácil com o “FOR” pois você deixa tudo preparado, e caso entrem novos alunos nessa escola imaginária, você só vai precisar adicionar o nome dele na variável alunos.



**OU (ABREVIADO):**

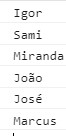
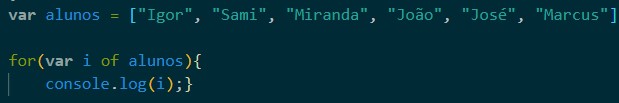
****

⧫ **Se mandarmos imprimir o “i”:** Ele vai mostrar quantos elementos ele está imprimindo:



Lembrando que são 6 elementos, já que o primeiro é o “0”.

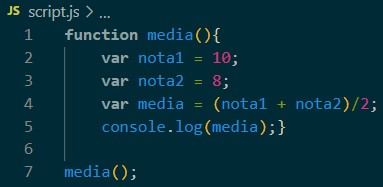
⧫ E SE EM VEZ DE “IN” usarmos “OF” na variável “i”, ele vai me dar os nomes dos alunos do mesmo jeito:



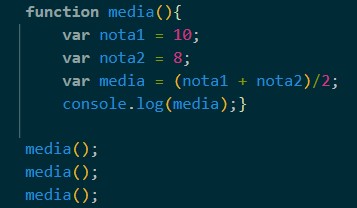
➜ **“FOR” junto com o “IN”:** Vai pegar o INDEX.

➜ **“FOR” junto com o “OF”:** Vai pegar o Valor, o “ELEMENTO DENTRO DO INDEX”, então ele vai varrer o arrey inteiro e pegar o elemento escolhido.

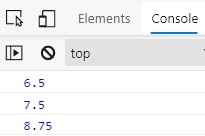
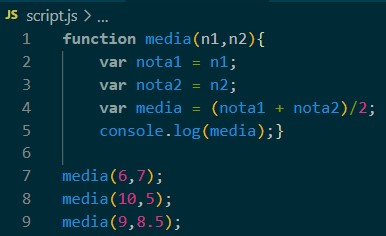
➤ **FUNÇÕES:** Ela dá um apelido para um bloco de comando, aí toda vez que eu chamar esse apelido, o computador vai executar todo esse bloco de comando. Na função, você também precisa abrir e fechar o parênteses, mesmo que ele esteja vazio, para ela poder funcionar! E depois de declarar a função, precisa mandar executá-la com por exemplo: **media();**

****

Poderia chamar a média várias vezes, caso eu quisesse:

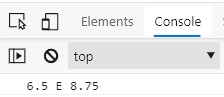
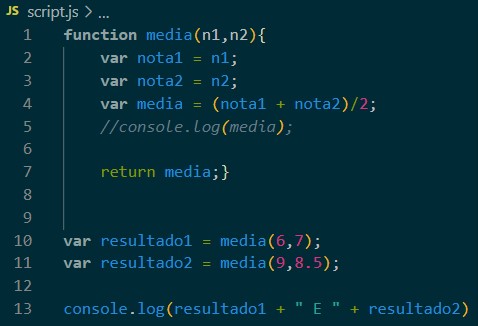


⧫ Nesse exemplo, usamos notas fixas, mas isso não precisa ser dessa maneira, quando criamos uma função, essa **FUNÇÃO** pode receber argumentos, que são valores que vamos passar só quando estiver chamando a função:



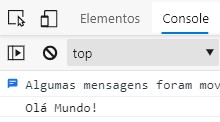
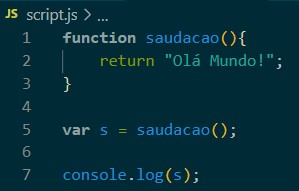
Aí você vai colocando as duas notas dos alunos na frente da média, e ele vai dando a média no final.

⧫ Uma **FUNÇÃO** tem também o poder de retornar um valor: “Toda operação aritmética retorna um valor que é o resultado”. Podemos fazer isso com a **FUNÇÃO** também:



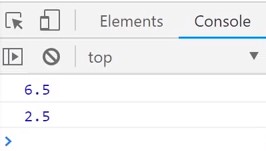
Se não tivéssemos colocado o ‘ “ E “ ‘ no console.log, ele teria somado as duas médias.

➤ **FUNÇÕES (PARTE 2):** Podemos fazer uma função de saudação, por exemplo:

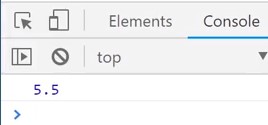
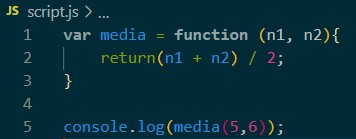


**OBS:** Precisa desses parênteses vazios, ou então não vai funcionar.

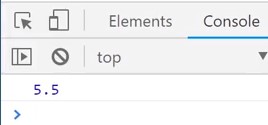
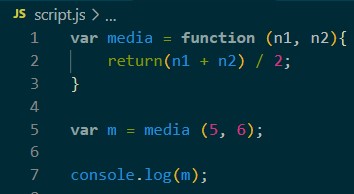
⧫ Voltando à **FUNÇÃO** da aula anterior, com alguns ajustes, temos:



➜ **Função Anônima:** É uma outra maneira de criar função.

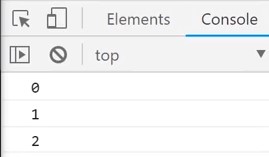
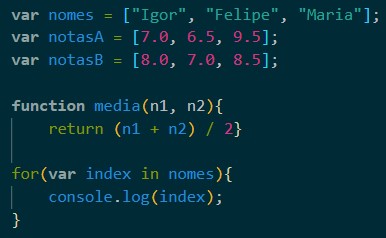


**OU**

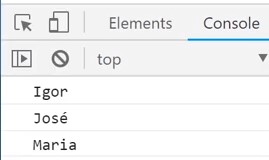
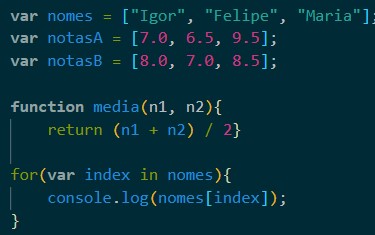


➤ **TUDO VISTO ATÉ AQUI:** Vamos criar um sistema para ver as notas dos alunos de uma determinada escola, e saber se eles estão aprovados ou reprovados.

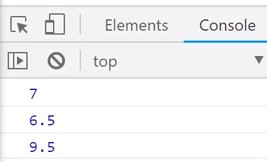
⧫ Vai mostrar quantos alunos tem na **FUNÇÃO**, quando manda imprimir o index:



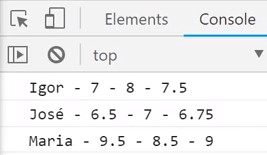
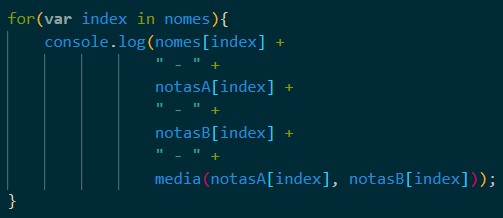
⧫ Podemos pedir os nomes na **FUNÇÃO**, mandando imprimir:



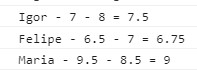
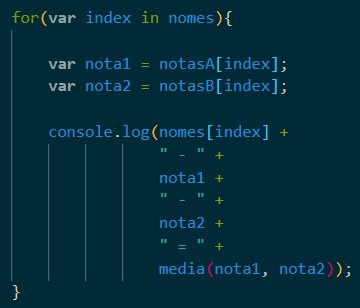
⧫ Ou simplesmente alguma das notas:



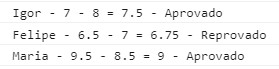
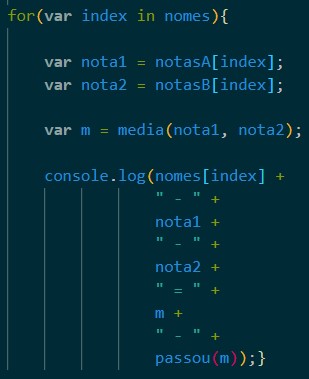
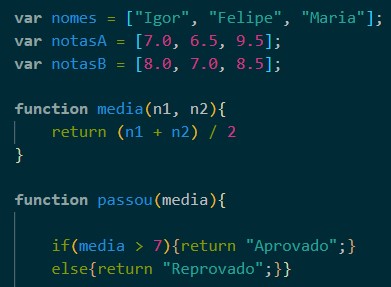
⧫ Ou **TUDO JUNTO**:



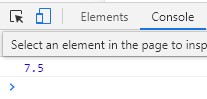
Mas assim fica muito bagunçado, podemos então arrumar para ficar mais ajeitado:



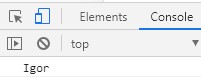
⧫ Com a APROVAÇÃO:



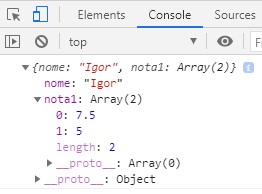
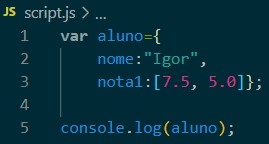
➤ **OBJETOS (Propriedades):** No objeto, diferentemente do ARREIO, em vez de COLCHETES, vamos usar CHAVES, mas essa é a única diferença, o Arreio e o Objeto são muito parecidos:



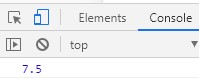
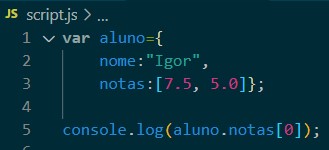
**OU**



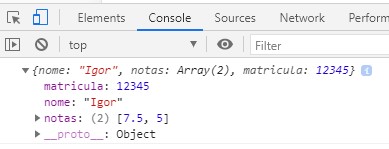
O ARREIO e o OBJETO são muito parecidos, a diferença é que agora, em vez de usar um index, nós vamos usar as CHAVES. E podemos ver que as propriedades funcionam como qualquer variável, mas quando ela está dentro de um objeto, ela muda de nome e se chama PROPRIEDADE, ou seja, sempre que você ouvir alguém falar de um objeto, ele não vai falar: “A variável do objeto”, e sim: “A propriedade do objeto”, mas ali dentro ele funciona como se fosse uma variável, e isso quer dizer que ele aceita todos os tipos de variável: STRING, NUMÉRICO e inclusive ARREIO, por exemplo:



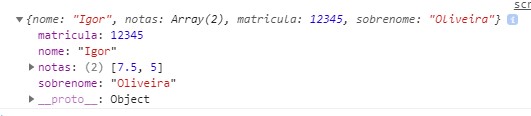
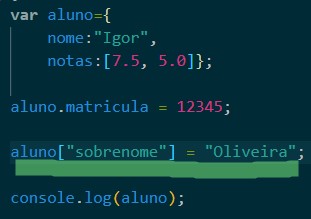
**OU (Nota específica:)**



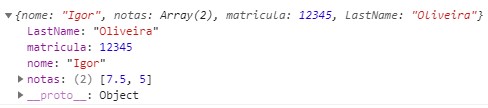
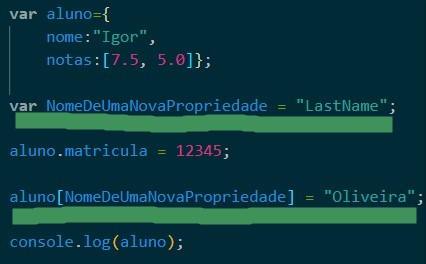
**OBS:** Eu criei o meu aluno, com nome e notas, mas eu posso adicionar outras propriedades depois, mesmo depois dele ter sido criado. E como fazemos isso?



**OU**

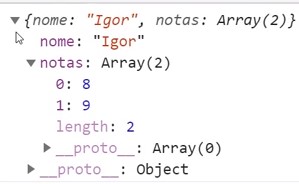
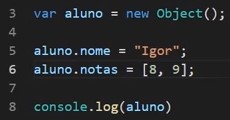
****

**OU (por variável):**

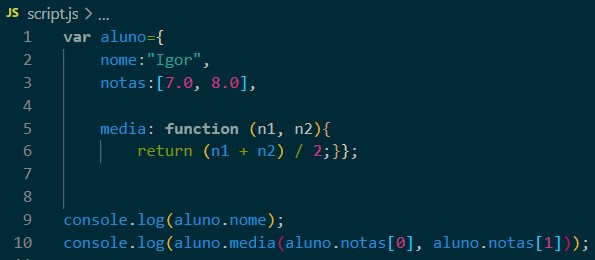
****

➜ **OU SEJA:** Um OBJETO no Javascript é a relação entre uma CHAVE e um VALOR, você tem que entender a sintaxe de que no momento da criação a CHAVE não vai estar entre aspa, mas se você quiser utilizar ela da maneira que fazemos como se fosse um ARREIO, entre colchetes, aí sim essa propriedade vai precisar estar entre aspas. O valor que essa propriedade recebe vai respeitar o tipo dele.

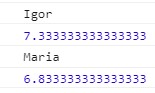
**OBS:** Existe também uma outra forma de criar um objeto:



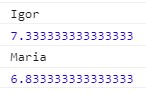
➤ **OBJETOS (Métodos):** Como aprendemos na aula anterior, o OBJETO funciona mais ou menos assim:



**OBS:** Enquanto temos poucos alunos, podemos fazer toda essa programação manualmente, aluno por aluno.



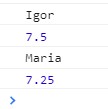
Mas e se tivermos 50 alunos? Como fazer para facilitar a digitação de todo esse conteúdo, sem ter que fazer um por um?



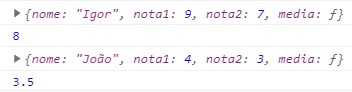
➜ **“THIS”:** This significa ISSO: ele vai depender do contexto, como estamos falando do OBJETO, o THIS vai ser ESSE objeto.



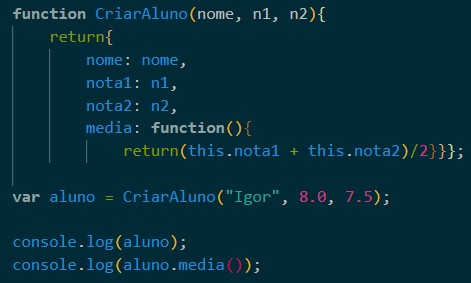
**FACILITANDO AINDA MAIS:**



➤ **OBJETOS CONSTRUTORES:** Aqui vamos misturar todos os conceitos para aprender maneiras mais simples de se criar um objeto, como vimos, quando temos poucos alunos, podemos fazer:



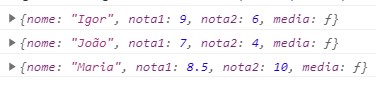
O problema ocorre, quando temos muitos alunos, e não queremos escrever muito para todos, então precisamos encontrar facilitadores, como por exemplo, criar um novo aluno:



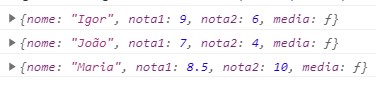
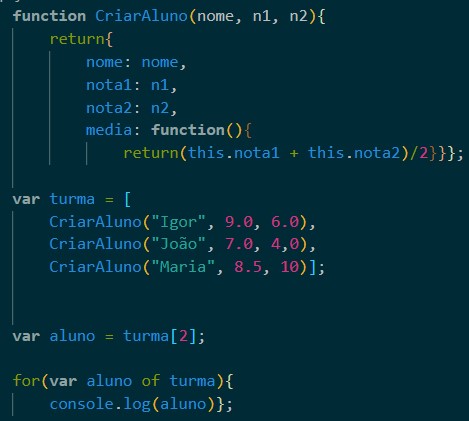
Vai precisar mudar apenas o número relativo ao aluno na “Turma”. Igor = 0, João = 1, Maria = 2;



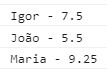
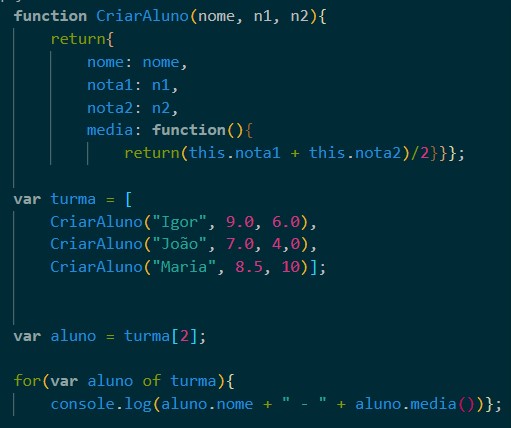
➜ **“forEach”:** Ele vai varrer o ARRAY, vai pegar cada um dos objetos dentro do ARRAY e vai imprimir TODOS ELES.



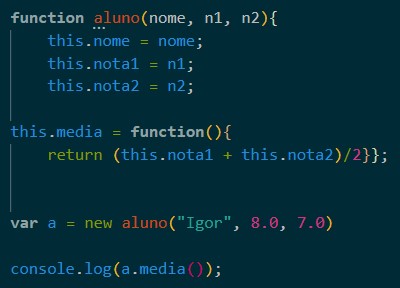
**OU**



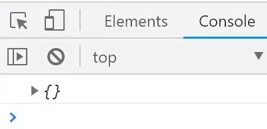
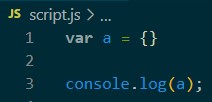
**DEIXANDO MAIS BONITO:**



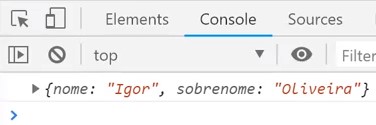
➜ **Criando um OBJETO com um FUNÇÃO:** Mais jeitos de fazer:



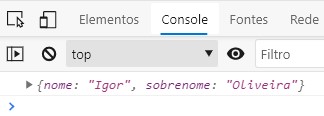
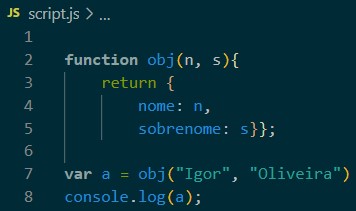
➤ **OBJETOS (RESUMO):** Para se criar um objeto, basta abrir CHAVES! Mesmo não tendo nada dentro (objeto vazio), já é um OBJETO. (Já o ARRAY é identificado por colchetes: [ ] ).



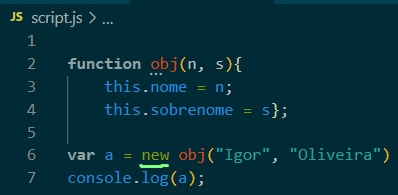
➜ **1º PRIMEIRA:** Primeira maneira de se criar um OBJETO:



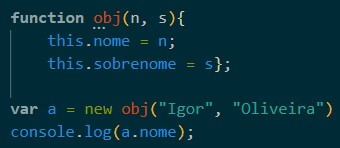
➜ **2º SEGUNDA:** Segunda maneira de se criar um OBJETO:



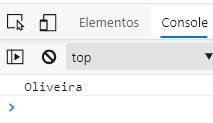
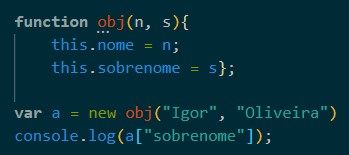
➜ **3º TERCEIRA:** Terceira maneira de se criar um OBJETO: P/ funcionar precisa desse “new”.



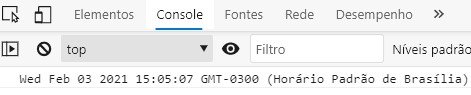
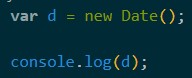
**OBS:** Escolhendo apenas uma das categorias:



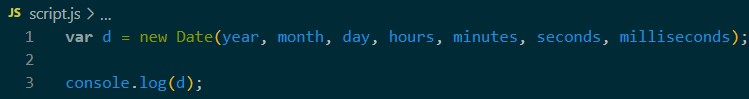
**OU:**



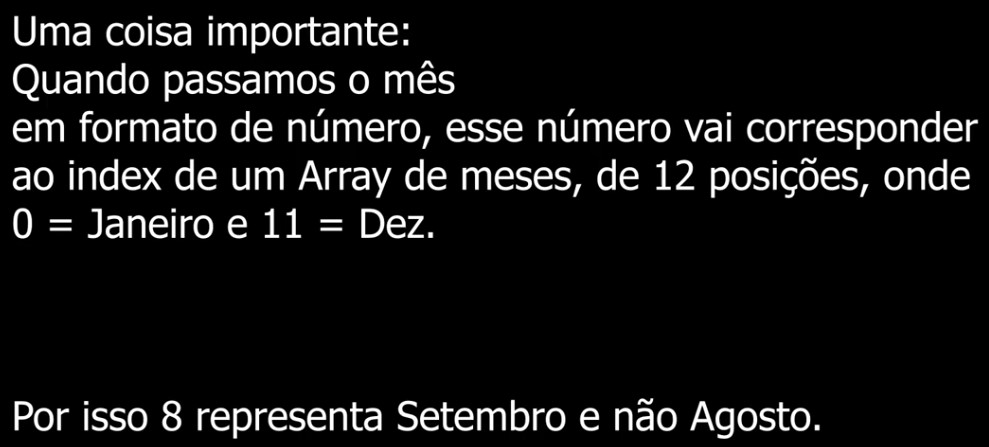
➤ **DATAS:** Datas no JAVASCRIPT são objetos! O D de “Date” tem que ser MAIÚSCULO!

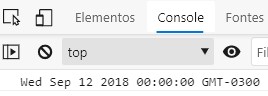
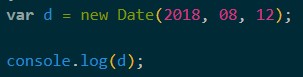


**OBS:** Mas e se eu quiser uma data específica?

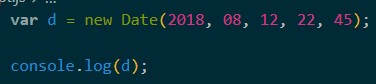


Pode inserir todas essas opções, ou apenas algumas delas.



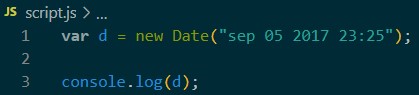


Como não foi passado horas / minutos / segundos, fica tudo 00:00:00.

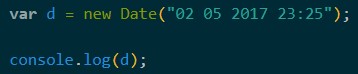


Sempre seguir a ordem: ANO / MÊS / DIA / HORAS / MINUTOS / SEGUNDOS / MILISSEGUNDOS.

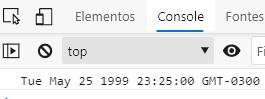
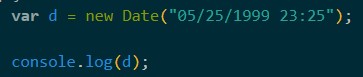
➜ **2º SEGUNDA:** Segunda maneira de se criar DATA, é por STRING:



**OU:**

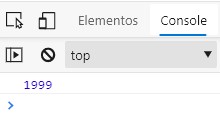
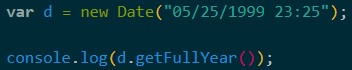


**OU:**

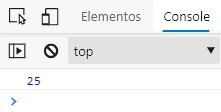
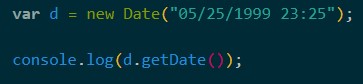


➜ **MÉTODOS DO OBJETO DATA:** Quando você quer escolher apenas uma das variáveis disponíveis.

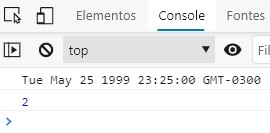
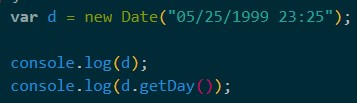
⧫ **ANO**:



⧫ **DIA**:

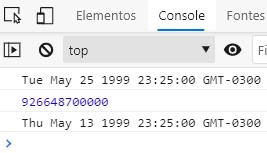
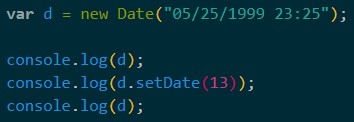


⧫ **DAY**: O getDay pega o dia DA SEMANA: de 0 a 6, 0 = Domingo e 6 = Sabado.



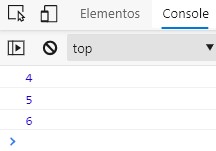
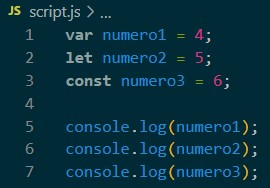
OUTROS: “getMinutes” / “getSeconds” / “getMonth” → Também começa com o 0; / “getMilliseconds”

**OBS:** Você pode também MUDAR algum dado:



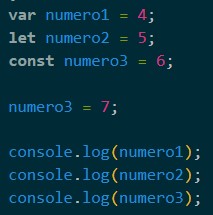
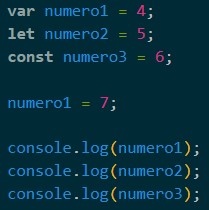
OUTROS: “setFullYear” / “setHours”

➤ **CONST, LET e VAR:** São outras formas de escrever em vez de usar o VAR:



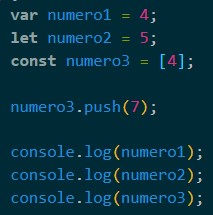
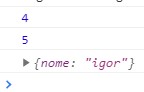
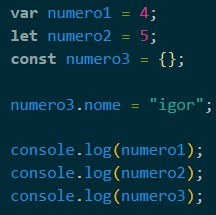
Todos eles funcionam mais ou menos da mesma maneira, mas eles possuem algumas diferenças;

⧫ **CONST**: É uma CONSTANTE, quer dizer que uma vez que foi atribuído um valor a ele, você não pode atribuir mais nada!!!

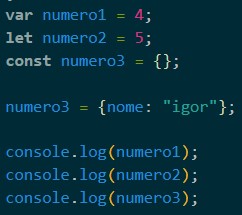


Ou seja, pode-se reatribuir outros valores para o VAR e para o LET, mas para o CONST NÃO!!

→ Mas você pode transformar a CONST em ARRAY e adicionar mais valores a ela, sem mudar:

 Ou Objeto: 

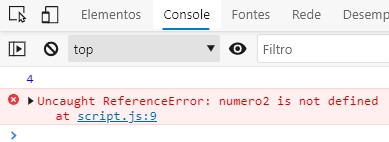
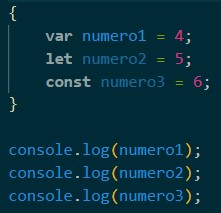
O que não funciona é criar um NOVO OBJETO, tentando substituir o original:



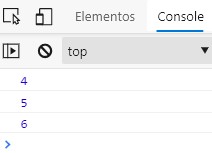
Ou seja a constante pode ser modificada mas não pode ser REATRIBUIDA

⧫ **LET**: A diferença do LET pro VAR vai se dar no ESCOPO;

O que é o ESCOPO? ESCOPO é onde a variável existe e você pode acessar ela e onde ela não existe.



Ou seja, o LET só vai funcionar se estiver tudo dentro do MESMO BLOCO, não aceita blocos separados.



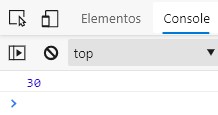
Ou seja, ela é a VARIÁVEL LOCAL, assim como a CONST.

⧫ **VAR**: Uma vez criado o VAR ela é acessível em qualquer lugar do seu código, é a VARIÁVEL GLOBAL.

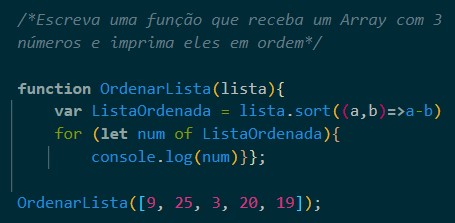
**OBS:** SEMPRE QUE POSSÍVEL iremos usar as VARIÁVEIS LOCAIS em vez da variável global porque assim, se cometermos algum erro dentro do nosso código, fica mais fácil de ENXERGAR esse erro.

➤ **EXERCÍCIOS:** Executando:

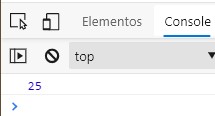
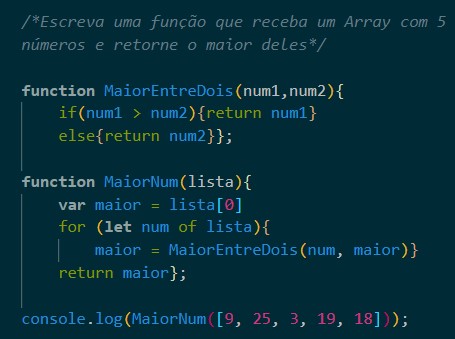
⧫ **01**: Escreva uma função que receba dois números e retorne o maior deles.



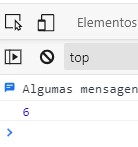
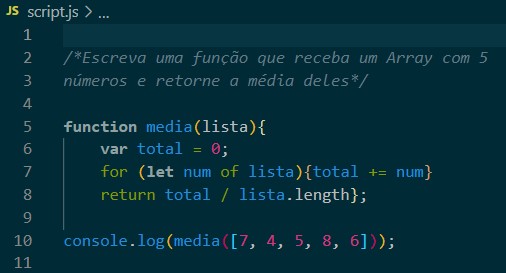
⧫ **02**: Escreva uma função que receba um Array com 3 números e imprima eles em ordem.



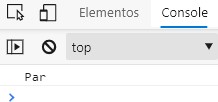
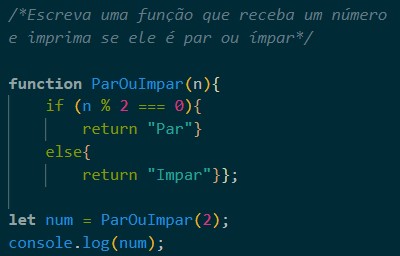
⧫ **03**: Escreva uma função que receba um Array com 5 números e retorne o maior deles.



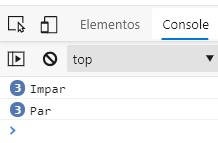
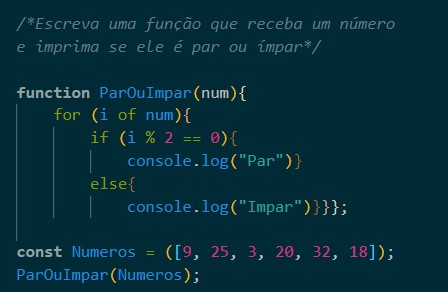
⧫ **04**: Escreva uma função que receba um Array com 5 números e retorne a média deles.



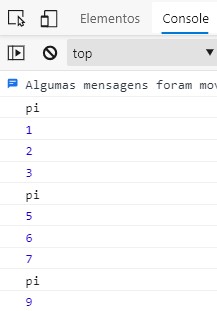
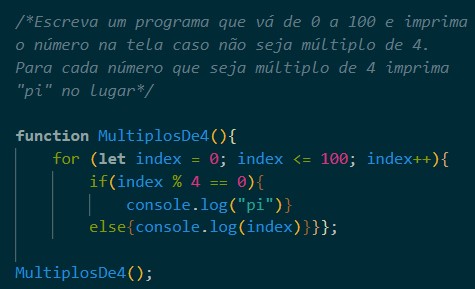
⧫ **05**: Escreva uma função que receba um número e imprima se ele é par ou ímpar:



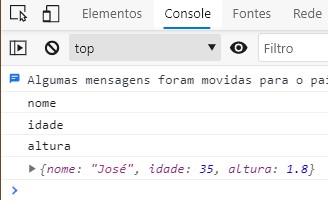
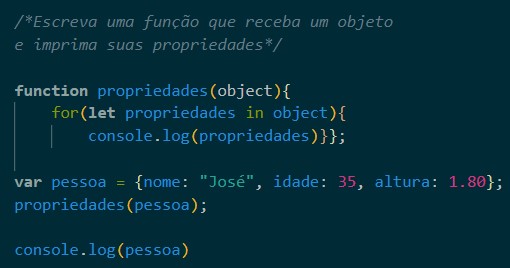
**OU:**



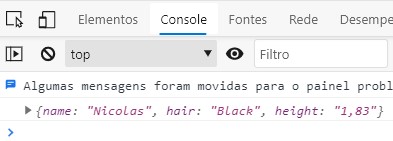
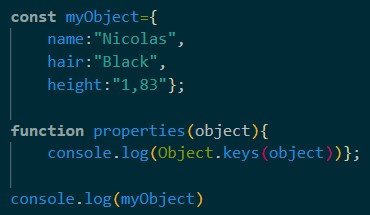
⧫ **06**: Escreva um programa que vá de 0 a 100 e imprima o número na tela caso não seja múltiplo de 4. Para cada número que seja múltiplo de 4 imprima "pi" no lugar.



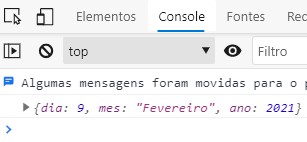
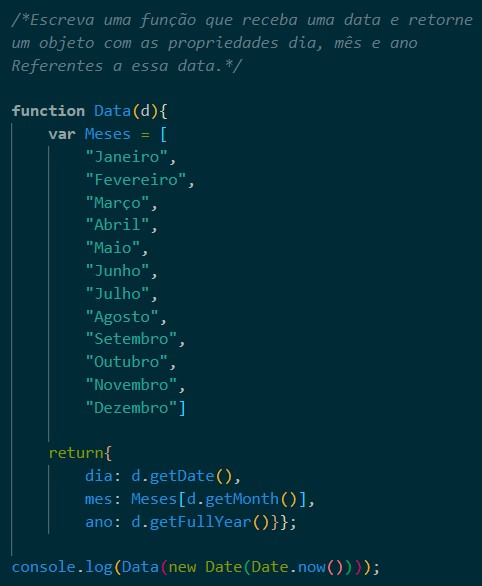
⧫ **07**: Escreva uma função que receba um objeto e imprima suas propriedades.



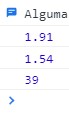
**OU:**



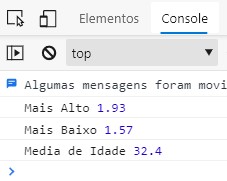
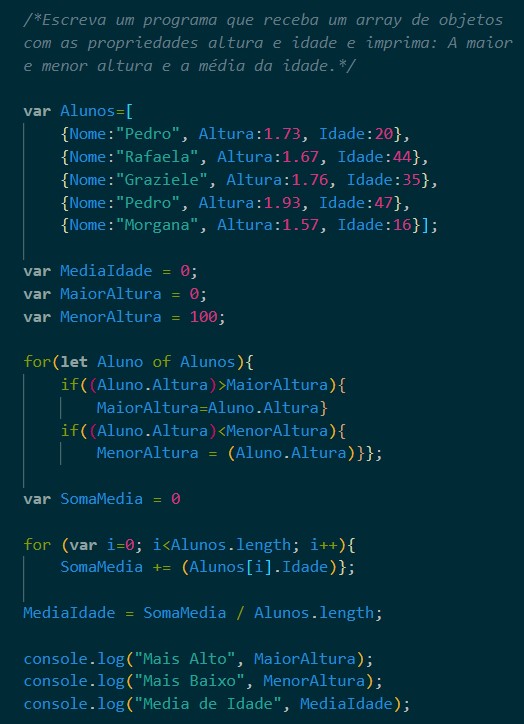
⧫ **08**: Escreva uma função que receba uma data e retorne um objeto com as propriedades dia, mês e ano Referentes a essa data.



⧫ **09**: Escreva um programa que receba um array de objetos com as propriedades altura e idade e imprima: A maior e menor altura e a média da idade.



**OU:**



⧫ **10**: Escreva um programa que imprima o seguinte padrão:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

