

AULA 1 – IF/ELSE E OPERADOR TERNÁRIO

OBJETIVO DA AULA

Conhecer, diferenciar e aplicar as estruturas condicionais.

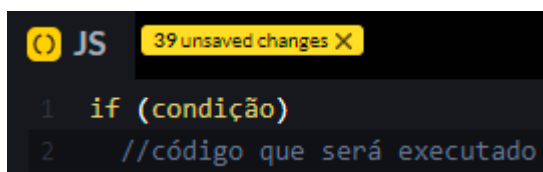
APRESENTAÇÃO

Nesta aula aprenderemos o que é uma estrutura condicional e como podemos aplicá-la ao nosso código. Basicamente, fazemos uso desta estrutura para verificarmos uma condição e definirmos o que será feito caso a condição seja verdadeira ou falsa. Você sabia que também recorremos a estruturas condicionais para nos comunicarmos no nosso dia a dia? Observe a frase: “Se fizer sol, vou à praia”, neste exemplo temos uma condição (se fizer sol) para executar uma ação (vou à praia). Resumindo, se a condição for verdadeira, a ação será executada. Na programação seguimos a mesma lógica. Aqueles que já conhecem outra linguagem de programação vão conseguir perceber uma certa semelhança de sintaxe.

1. IF/ELSE

A estrutura condicional mais utilizada é o *if* (se); tal estrutura permite ao JavaScript testar uma ou mais condições pré-estabelecidas para daí executar o trecho de código referente aquele teste. Para criar uma condição é necessário utilizar os operadores (aritméticos, relacionais e lógicos). Podemos utilizar o *if* de 3 formas diferentes. Vejamos:

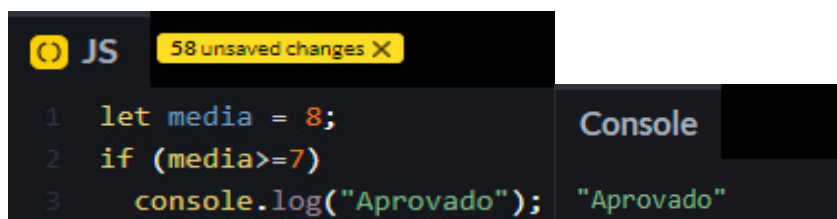
- **if**: executa o código caso a condição seja verdadeira.



```

JS 39 unsaved changes X
1  if (condição)
2    //código que será executado
    
```

Como ficaria na prática?

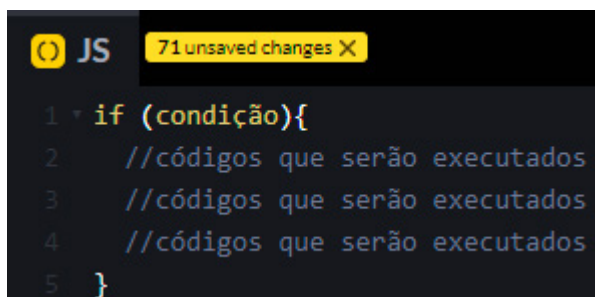


```

JS 58 unsaved changes X
1  let media = 8;
2  if (media >= 7)
3    console.log("Aprovado");
    
```

Console
"Aprovado"

No exemplo acima criamos uma variável *media* (*linha 1*) e atribuímos um valor a ela. Em seguida, utilizamos a palavra *if* (*linha 2*) seguido da condição *media >= 7*, caso a variável *media* tenha uma nota maior ou igual a 7 (verdadeiro) a linha abaixo (*linha 3*) será executada, ou seja, o comando *console.log* exibirá a mensagem “Aprovado” no console. Quando temos mais de uma linha de código a ser executada devemos usar *{ }* para delimitar o trecho de código.

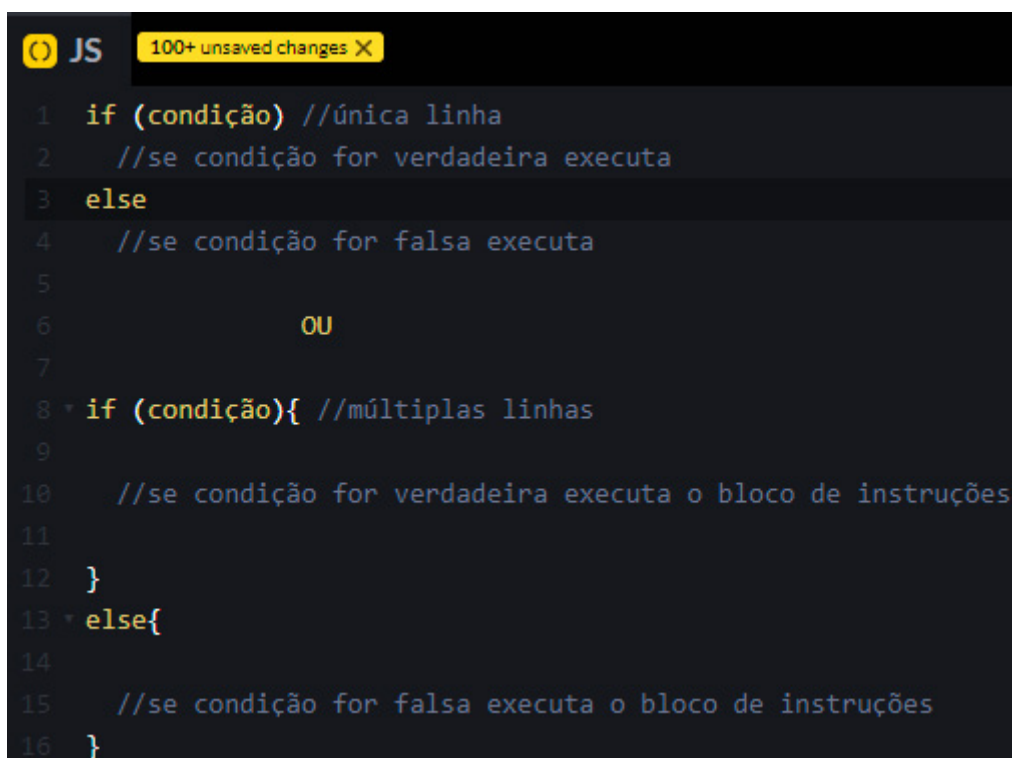


```

1 * if (condição){
2   //códigos que serão executados
3   //códigos que serão executados
4   //códigos que serão executados
5 }
  
```

Neste caso, caso a condição seja verdadeira, o JavaScript executará o bloco de instruções que está dentro das chaves *{ }*.

- **if-else:** executa o código caso a condição seja verdadeira ou falsa. Para isso utilizamos a palavra-chave *else*.



```

1 if (condição) //única linha
2   //se condição for verdadeira executa
3 else
4   //se condição for falsa executa
5
6           OU
7
8 * if (condição){ //múltiplas linhas
9
10   //se condição for verdadeira executa o bloco de instruções
11
12 }
13 * else{
14
15   //se condição for falsa executa o bloco de instruções
16 }
  
```

Seguindo o nosso exemplo, como ficaria na prática?

```

1 let media = 6;
2 if (media >= 7)
3   console.log("Aprovado");
4 else
5   console.log("Reprovado");

```

Console

"Reprovado"

Na nossa versão melhorada, caso a condição seja verdadeira, a *linha 3* será executada e exibirá a mensagem "Aprovado" no console, caso contrário, a *linha 4* será executada e exibirá a mensagem "Reprovado" no console.

- **else if:** quando há casos em que duas opções (verdadeiro/falso) não são o suficiente, devemos utilizar o *else if* para testar uma nova condição.

```

1 * if ( condição ) {
2   //se condição for verdadeira, executa.
3 * } else if ( outra condição ) {
4   //se condição for verdadeira, executa.
5 * } else if ( outra condição ) {
6   //se condição for verdadeira, executa.
7 * } else if ( quantas condições quiser ) {
8   //se condição for verdadeira, executa.
9 * } else {
10  // Ação final se nenhuma condição for verdadeira
11 }

```

Seguindo o nosso exemplo, como ficaria na prática?

```

1 let media = 3;
2 if (media >= 7)
3   console.log("Aprovado");
4 else if (media >= 5 & media < 7)
5   console.log("Recuperação");
6 else
7   console.log("Reprovado");

```

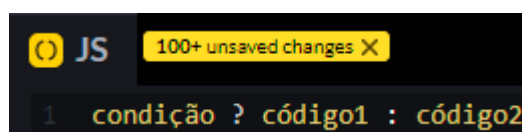
Console

"Reprovado"

Na nossa versão melhorada, **se** a condição for verdadeira, a *linha 3* será executada e exibirá a mensagem “Aprovado” no console, **senão se** a nova condição *for* verdadeira a *linha 5* será executada e exibirá a mensagem “Recuperação” no console, **senão** a *linha 7* será executada e exibirá a mensagem “Reprovado” no console. Chamamos esta estrutura de **se encadeado** ou **se aninhado**.

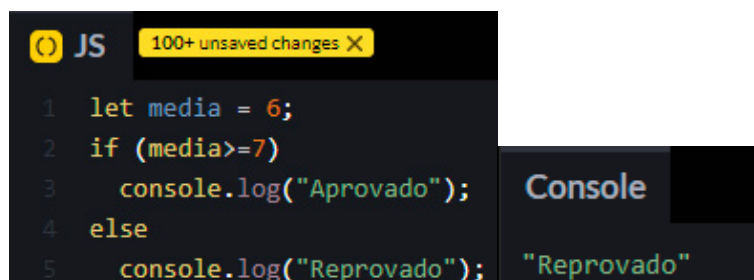
2. OPERADOR TERNÁRIO

O operador ternário nada mais é do que uma versão mais compacta do **if-else**, com ele podemos deixar nosso código mais limpo, pois um **if** de 4 linhas é substituído por apenas 1 linha. É possível encadear diversas verificações de condição, porém não é recomendado, pois a leitura do código fica mais difícil do que um **if** convencional. Vamos conhecer a sua sintaxe:



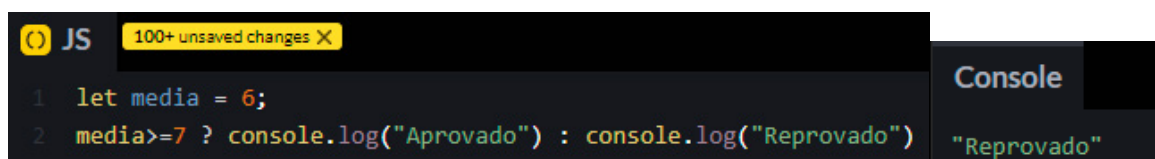
```
1 condição ? código1 : código2
```

Caso a condição seja verdadeira, o código1 é retornado, caso contrário, o código2 é retornado. O retorno pode ser atribuído a uma variável ou ao retorno de uma função. Vamos comparar o uso do **operador ternário** com o uso do **if** para que você possa perceber a diferença entre eles.



```
1 let media = 6;
2 if (media >= 7)
3   console.log("Aprovado");
4 else
5   console.log("Reprovado");
```

Console
"Reprovado"



```
1 let media = 6;
2 media >= 7 ? console.log("Aprovado") : console.log("Reprovado");
```

Console
"Reprovado"

Repare que no primeiro exemplo escrevemos o código em 5 linhas e no segundo exemplo escrevemos o mesmo código em apenas duas linhas. Logo após o ponto de interrogação (?) temos a instrução que representa a verdade e depois dos dois pontos (:) temos a instrução que representa a falsidade.

CONSIDERAÇÕES FINAIS

Nesta aula aprendemos a utilizar as estruturas condicionais **if**, **if-else** e **else if** e também o **operador ternário**. Você deve estar se perguntando: “São tantas variações da estrutura condicional **if**, qual delas devo utilizar?” Você decidirá qual utilizar segundo o contexto da aplicação. Além do **if**, vimos também o **operador ternário**, utilizado como atalho para o **if**, que pode deixar o código mais enxuto e mais limpo. Os conceitos abordados nesta aula são fundamentais para o seu desenvolvimento em qualquer linguagem de programação.

MATERIAIS COMPLEMENTARES

Operador ternário:

<https://youtu.be/uAyfj46vOJE>

If/Else:

<https://youtu.be/8UXQ6S0KURk>

REFERÊNCIAS

FLANAGAN, David. *JavaScript: O guia definitivo*. Porto Alegre, RS: Bookman, 2013.

AULA 2 – SWITCH CASE

OBJETIVO DA AULA

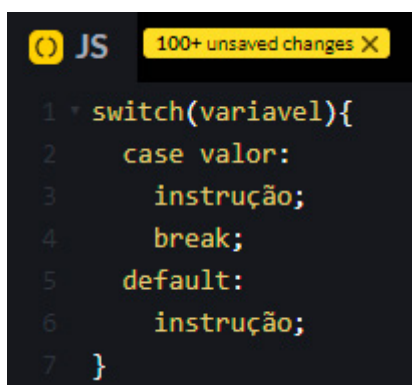
Conhecer, diferenciar e aplicar as estruturas condicionais.

APRESENTAÇÃO

Na aula anterior você viu como aplicar as diferentes formas de utilização da estrutura condicional **if** e o operador ternário. Nesta aula, aprenderemos outra estrutura condicional chamada **switch case**, utilizada para analisar diversos valores diferentes para a mesma variável. Podemos dizer que sua estrutura é mais organizada e de fácil compreensão se comparado com a versão **else if**, dependendo do contexto. Além disso, é uma excelente alternativa para a execução de códigos com muitas comparações. Vamos conhecê-la? Vem comigo!

1. SWITCH CASE

O **switch case** é uma estrutura condicional assim como o **if**, que vimos na aula anterior; tal estrutura permite que um conjunto de instruções seja executado de acordo com cada opção que chamamos de **case**. Vamos entender a sua sintaxe:



```

1 switch(variavel){
2   case valor:
3     instrução;
4     break;
5   default:
6     instrução;
7 }
  
```

Onde, **variável** representa a expressão que será avaliada, caso (case) seja verdadeiro, ele executa a instrução contida nele. O **break** (uso opcional) é utilizado para encerrar a leitura do **switch**, caso o break seja omitido, o programa continua a execução para a próxima instrução. O **default** será executado quando nenhum dos cases corresponder ao valor especificado. Veja como ficaria o nosso exemplo que exibe a situação de um aluno baseado na sua média, porém, como no JavaScript o switch não opera sobre intervalos (case media >=7) precisaremos fazer algumas modificações. Veja:

```

JS 100+ unsaved changes X
1 var media=6
2 switch (media) { //Recebe a média...
3     case 0: case 1: case 2: case 3: case 4: //Médias entre 0 e 4...
4         console.log('Reprovado');
5         break; //Desvio de fluxo - sai do laço se a condição for atendida
6     case 5: case 6: //Médias entre 5 e 6
7         console.log('Recuperação');
8         break;
9     case 7: case 8: case 9: case 10: //Médias entre 7 e 10
10        console.log('Aprovado');
11        break;
12    default: //Caso nenhuma das condições seja satisfeita...
13        console.log('Nota inválida');
14 }

```

Console
"Recuperação"

Neste caso, o ideal seria trabalhar com conceitos A, B, C e D. Um exemplo clássico de utilização do switch é criar um programa que exiba o nome do mês a partir do seu respectivo número. Veja:

```

JS
1 let mes = 8;
2 switch (mes) {
3     case 1:
4         nomeMes = "Janeiro";
5         break;
6     case 2:
7         nomeMes = "Fevereiro";
8         break;
9     case 3:
10        nomeMes = "Março";
11        break;
12    case 4:
13        nomeMes = "Abril";
14        break;
15    case 5:
16        nomeMes = "Maio";
17        break;
18    case 6:
19        nomeMes = "Junho";
20        break;

```



```

21     case 7:
22         nomeMes = "Julho";
23         break;
24     case 8:
25         nomeMes = "Agosto";
26         break;
27     case 9:
28         nomeMes = "Setembro";
29         break;
30     case 10:
31         nomeMes = "Outubro";
32         break;
33     case 11:
34         nomeMes = "Novembro";
35         break;
36     case 12:
37         nomeMes = "Dezembro";
38         break;
39     default:
40         nomeMes = "Mês inexistente";
41 }
42 console.log("O mês escolhido é: " + nomeMes);

```

Console

"O mês escolhido é: Agosto"

Neste caso, a variável nomeMes é alterada conforme o mês escolhido, porém, ao invés de colocar o método console.log() dentro de cada case, como mostrado no exemplo da média, optamos por deixá-lo fora do bloco switch, evitando a repetição e deixando o código mais limpo.

DESTAQUE

Cuidado! Falamos anteriormente que o uso do break é opcional. Fique atento, pois, ao optar por não o declarar, dependendo do caso, podemos obter um resultado inesperado no nosso projeto. Vamos imaginar a seguinte situação: precisamos criar um script que consiga calcular o salário de um professor universitário conforme a sua titulação (especialista – 5%, mestre – 10%, doutor – 15%). Vejamos:


```

JS 100+ unsaved changes X
1  var titulacao = "mestre";
2  var salario = 2000;
3
4  switch (titulacao) {
5      case "especialista":
6          salario *= 1.05; // é o mesmo que salario = salario * 1.05
7      case "mestre":
8          salario *= 1.10; // é o mesmo que salario = salario * 1.10
9      default:
10         salario *= 1.15; // é o mesmo que salario = salario * 1.15
11     }
12     console.log(salario);

```

Console
2530

Repare que a omissão do break neste código produziu o resultado 2530, quando o resultado deveria ser 2200, pois 10% de 2000 é 200. O que aconteceu? Sem o break, ele executou o a instrução do case "mestre" e a instrução do default, resultando em 2530. Vamos adicionar o break no código abaixo para perceber a diferença.

```

JS 100+ unsaved changes X
1  var titulacao = "mestre";
2  var salario = 2000;
3
4  switch (titulacao) {
5      case "especialista":
6          salario *= 1.05; // é o mesmo que salario = salario * 1.05
7          break;
8      case "mestre":
9          salario *= 1.10; // é o mesmo que salario = salario * 1.10
10         break;
11     default:
12         salario *= 1.15; // é o mesmo que salario = salario * 1.15
13         break;
14     }
15     console.log(salario);

```

Console
2200

Percebeu a diferença? Com o break ele executou somente a instrução referente ao case "mestre", resultando o valor correto 2200. Resumindo, o uso do break se faz necessário quando se deseja que apenas um bloco seja executado e não os demais que vem abaixo dele.

CONSIDERAÇÕES FINAIS

Ao final desta aula chegamos à conclusão de que precisamos usar o switch com muita cautela. Ele é semelhante ao *if*, porém não é possível substituí-lo em todas as situações. Isso se dá pelo fato dele ser um pouco mais restrito que o *if*. O switch trata os casos de uma única entrada enquanto o *if* não te restringe a quantas entradas serão avaliadas, ou seja, você pode comparar diretamente duas ou mais condições em um único *if*, por exemplo, *if (media >= 7 && frequência > 75)*.

MATERIAIS COMPLEMENTARES

JavaScript switch – o que é e como utilizar:

<https://youtu.be/KO6TVNuL0yc>

Como funciona o Switch Case no Javascript:

<https://youtu.be/NsRgcbqt1YI>

REFERÊNCIAS

FLANAGAN, David. *JavaScript: O guia definitivo*. Porto Alegre, RS: Bookman, 2013.

AULA 3 – WHILE E DO... WHILE

OBJETIVO DA AULA

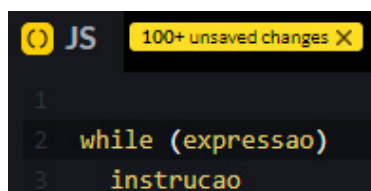
Implementar scripts com laços de repetições.

APRESENTAÇÃO

Nas aulas anteriores aprendemos como implementar diferentes tipos de estruturas condicionais no nosso código. Nesta aula aprenderemos as estruturas de repetição *while* e *do... while*. Por que utilizar estruturas de repetição? Basicamente, para evitar repetição de trechos de código. Uma estrutura de repetição, seja ela qual for, permite que um trecho de código seja executado repetidamente até que ocorra condição de parada. A partir de agora conseguiremos solucionar problemas cada vez mais complexos.

1. WHILE

A estrutura de repetição *while* (enquanto) funciona da seguinte forma: um bloco de código é executado até que o teste condicional se torne falso. Veja como é a sintaxe desta estrutura:

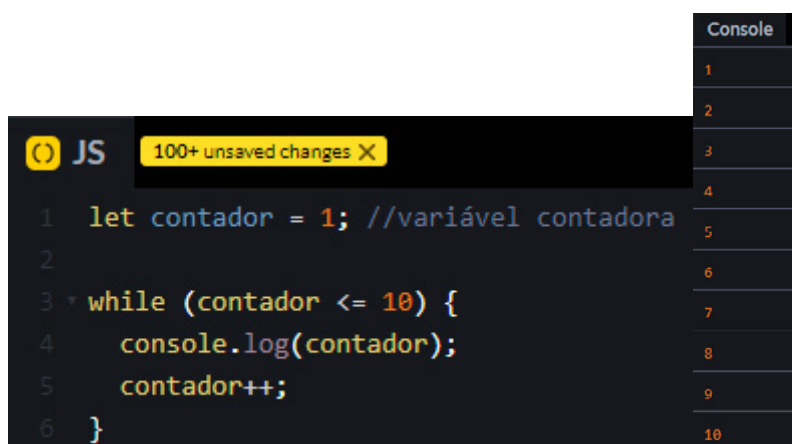


```

1 while (expressao)
2   instrucao

```

Onde **instrução** representa o bloco de instruções que será executado dentro do *while* e **expressão** representa o teste condicional para que o código seja executado. Por exemplo, criaremos um pequeno script para exibir na tela os números de 1 a 10.



```

1 let contador = 1; //variável contadora
2
3 while (contador <= 10) {
4   console.log(contador);
5   contador++;
6 }

```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ed7460889e221c50/Unitade2Aula3a>

Optei por exemplificar apenas com 10 números para que o *print* da imagem coubesse na apostila, mas você pode acessar o código no link acima e testar com o número que quiser.

Agora vamos analisar o código. Na *linha 1*, temos uma variável chamada contador, esta variável dentro de uma estrutura de repetição se comporta como uma variável contadora. Calma! Já vamos entender o que isso significa. Na *linha 3*, temos a estrutura de repetição *while* testando a seguinte condição `contador<=10`, podemos traduzir esta linha da seguinte forma: "enquanto a variável contador for menor ou igual a 10, faça." Caso a condição seja verdadeira, as instruções dentro do *while* são executadas. No nosso caso, a condição é verdadeira, portanto, a *linha 4* será executada, ou seja, a variável contadora será exibida na tela. Em seguida, a *linha 5* será executada, nela temos `contador++` (é o mesmo que `contador = contador + 1`), ou seja, a variável contador está sendo incrementada, logo o valor da variável deixa de ser 1 e passa a ser 2. Na *linha 6*, temos o final do bloco de instruções representado pelo fechamento da `}`, ao se deparar com o final do bloco ele entende que deve voltar para a linha 3 e repetir todo o processo até que a condição seja falsa, por isso chama-se estrutura de repetição. Repare que em cada rodada a variável contador é incrementada, por esse motivo é considerada uma variável contadora. Vale ressaltar que, ser contadora não significa contar de um em um, podemos contar de dois em dois (`contador+=2` ou `contador=contador + 2`), de três em três (`contador+=3` ou `contador=contador + 3`) e assim sucessivamente.

Vamos melhorar o nosso exemplo exibindo também o somatório dos números de 1 a 10, para isso, utilizaremos uma variável que se comportará como uma variável acumuladora dentro da nossa estrutura de repetição. Diferentemente da variável contadora que recebe ela mesma + 1 (caso você esteja contando de 1 em 1), a variável acumuladora recebe ela mesma + o que ela quer acumular. Veja:

The screenshot shows a code editor with a dark theme. On the left, there's a JavaScript file named 'JS' with 100+ unsaved changes. The code is as follows:

```

1 let contador = 1; //variável contadora
2 let soma = 0;
3
4 while (contador <= 10) {
5   console.log(contador);
6   soma+=contador; //variável acumuladora (soma = soma + contador)
7   contador++;
8 }
9 console.log("O somatório é: " + soma);

```

On the right, the 'Console' panel shows the output of the code. It displays the numbers 1 through 10, each on a new line, followed by the final sum: "O somatório é: 55".

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Agora vamos analisar o código. Inserimos aqui uma variável nova chamada soma (*linha 2*) e a mesma está recebendo o valor inicial 0. Dentro da nossa estrutura de repetição inserimos a nossa variável acumuladora (*linha 6*) `soma+=contador` (`soma = soma + contador`), que a cada rodada acumulará os valores da variável contador que varia de 1 a 10, ou seja, a nossa variável acumuladora somará $1+2+3+4+5+6+7+8+9+10$. Ao final exibirá a seguinte mensagem: "O somatório é: 55", através do método `console.log` (*linha 9*). Veja na tabela abaixo os valores assumidos pelas variáveis contador e soma em todas as rodadas do nosso script.

	contador	soma
Valor Inicial	1	0
Rodada 1	1	1
Rodada 2	2	3
Rodada 3	3	6
Rodada 4	4	10
Rodada 5	5	15
Rodada 6	6	21
Rodada 7	7	28
Rodada 8	8	36
Rodada 9	9	45
Rodada 10	10	55

VOCÊ SABIA?

Esta técnica de montar uma tabelinha para testar programas de computador é chamada de teste de mesa ou chinês.

2. DO... WHILE

Diferentemente do *while*, a estrutura de repetição *do... while* (faça... enquanto) testa a condição somente no final do bloco de instruções, com isso, mesmo que a condição seja FALSA, o bloco de instruções será executado ao menos uma vez. Veja como é a sintaxe desta estrutura:

```

1  do {
2
3      instrucoes
4
5  }while(condicao);
  
```

Nela as instruções são executadas uma vez e reexecutadas cada vez que a condição (linha 5) for verdadeira, quando a condição for falsa, a estrutura de repetição é abandonada. Vamos refazer os mesmos exemplos só que agora usando a estrutura *do... while*. Vejamos:

```

1 let contador = 1;
2
3 do {
4   console.log(contador);
5   contador++;
6 }while (contador <= 10);
  
```

Console

1
2
3
4
5
6
7
8
9
10

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Repare que trocamos de estrutura, mas obtivemos o mesmo resultado. Agora vamos ver como fica o exemplo do somatório.

```

1 let contador = 1; //variável contadora
2 let soma = 0;
3
4 do{
5   console.log(contador);
6   soma+=contador; //variável acumuladora (soma = soma + contador)
7   contador++;
8 } while (contador <= 10);
9 console.log("O somatório é: " + soma);
  
```

Console

1
2
3
4
5
6
7
8
9
10
"O somatório é: 55"

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Reparou que a lógica é a mesma? A diferença é que o teste que antes era feito na *linha 4*, agora está sendo feito na *linha 8*. Outra diferença também é que a condição do *do... while* é seguida de ponto e vírgula (;).

CONSIDERAÇÕES FINAIS

A estrutura repetição é utilizada quando queremos executar mais de uma vez um mesmo bloco de instruções. Nesta aula, conhecemos duas estruturas de repetição bem semelhantes, são elas: *while* e *do... while*. Você enquanto programador é que deve decidir qual delas irá utilizar. A diferença entre elas é bem sutil, porém, a estrutura *do... while* não é muito utilizada, pois não é comum escrevermos um código considerando que um bloco de instruções será executado mesmo que a condição testada seja falsa. Portanto, use parcimônia!

MATERIAIS COMPLEMENTARES

Tutorial *do... while*:

<https://youtu.be/y8°tiWoVfRI>

Tutorial *while*:

<https://youtu.be/buZQknMhPKo>

Laços e interações:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Loops_and_iteration

REFERÊNCIAS

FLANAGAN, David. *JavaScript: O guia definitivo*. Porto Alegre, RS: Bookman, 2013.

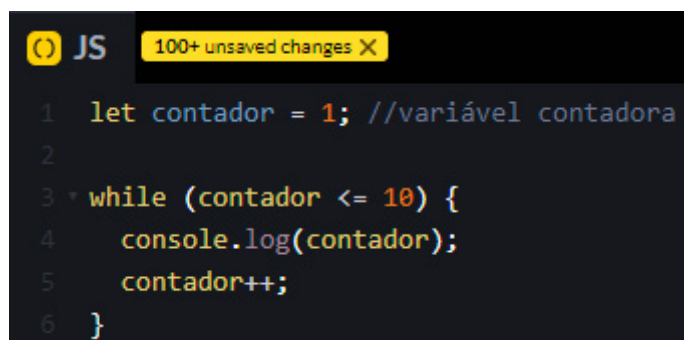
AULA 4 – FOR, FOR... IN E FOR... OF

OBJETIVO DA AULA

Implementar scripts com laços de repetições.

APRESENTAÇÃO

Na aula anterior aprendemos duas estruturas de repetição muito parecidas: *while* e *do... while*. Nesta aula, aprenderemos a estrutura de repetição *for* e suas variações *for... in* e *for... of*. Dentre todas as estruturas de repetição, ela é a mais utilizada, pois é mais simples que as demais. Por exemplo, na estrutura *while* a variável contadora é inicializada antes da estrutura começar (linha 1) e é incrementada antes da estrutura terminar (linha 5). Veja:



```

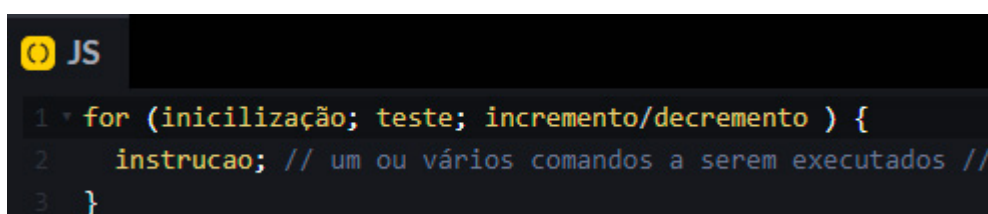
JS 100+ unsaved changes X
1 let contador = 1; //variável contadora
2
3 while (contador <= 10) {
4   console.log(contador);
5   contador++;
6 }
  
```

Na estrutura de repetição *for*, a inicialização, o teste e o incremento é feito numa única linha e por esse motivo acaba sendo a estrutura queridinha dos desenvolvedores.

1. FOR

A estrutura de repetição *for* é muito utilizada quando temos estruturas incrementais, sendo assim, uma das suas principais aplicações é percorrer objetos ou variáveis do tipo *array*, assunto este que será aprofundado na próxima unidade.

Como já foi dito anteriormente, o *for* concentra tudo numa única linha (inicialização, teste e incremento) e isso facilita “entender o que o laço está fazendo e evita erros, como esquecer de inicializar ou incrementar a variável de laço”, que é a (variável contadora) (FLANAGAN, 2013). Veja como é a sintaxe desta estrutura:



```

JS
1 for (inicialização; teste; incremento/decremento ) {
2   instrução; // um ou vários comandos a serem executados //
3 }
  
```

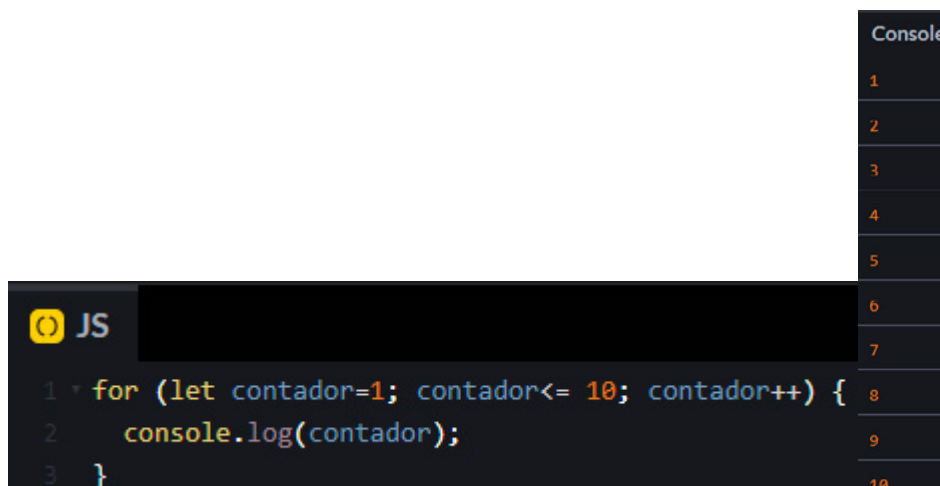
Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Onde **inicialização** corresponde a declaração da variável contadora com seu valor inicial; **teste** corresponde a condição que será avaliada para então executar ou sair da estrutura de repetição; e incremento/decremento corresponde a alteração da variável contadora que tanto pode ser incrementada (++) quanto decrementada (--).

DESTAQUE

É importante destacar que a “expressão de inicialização é avaliada apenas UMA vez, antes que o laço comece” (FLANAGAN, 2013). Além disso, a inicialização e o teste não são obrigatórios, ou seja, podem ser omitidos. Porém, é preciso tomar muito cuidado para evitar que a estrutura se torne um laço infinito.

Vamos entender como o *for* funciona, utilizando os mesmos exemplos da aula de *while* e *do... while*. Primeiro vamos ver como escrever os números de 1 a 10 na tela na ordem crescente e decrescente, em seguida vamos fazer o somatório destes números. **Exemplo 1:**



```

JS
1 for (let contador=1; contador<= 10; contador++) {
2   console.log(contador);
3 }
  
```

Console

1
2
3
4
5
6
7
8
9
10

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Repare que na expressão inicial é feita a declaração da variável contador inicializada com o valor 1. No teste é criada uma expressão para avaliar se o valor da variável contador é menor ou igual 10. Por fim, há o a atualização (incremento) da expressão inicial representada pelo comando contador++, que significa adicionar 1 ao valor de contador. Viu como é simples? Utilizamos apenas 3 linhas para fazer a mesma coisa que o *while* faz em 5 linhas. Observe a diferença:

```

JS
1 * for (let contador=1; contador<= 10; contador++) {
2   console.log(contador);
3 }

JS
1 let contador = 1; //variável contadora
2
3 while (contador <= 10) {
4   console.log(contador);
5   contador++;
6 }

```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Agora vamos decrementar ao invés de incrementar a variável contadora (contador). Veja:

```

JS
1 * for (let contador=10; contador>=1; contador--) {
2   console.log(contador);
3 }

```

Console
10
9
8
7
6
5
4
3
2
1

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Repare que na expressão inicial é feita a declaração da variável contador inicializada com o valor 10. No teste é criada uma expressão para avaliar se o valor da variável contador é maior ou igual 1. Por fim, há a atualização (decremento) da expressão inicial representada pelo comando contador--, que significa retirar 1 do valor de contador.

Agora vamos ver o nosso exemplo do somatório. **Exemplo 2:**

```

JS
1 soma=0;
2 * for (let contador=1; contador<=10; contador++) {
3   console.log(contador);
4   soma+=contador; //variável acumuladora (soma = soma + contador)
5 }
6 console.log("O somatório é: " + soma);

```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Mais uma vez economizamos duas linhas se comparado com o mesmo código feito com o *while*.

1.1. FOR... IN

A versão *for... in* é muito utilizada para percorrer as propriedades de um objeto, porém também pode ser utilizado para percorrer *arrays*. Lembra que falamos sobre objeto, propriedade e método na unidade 1? Não! Então vamos relembrar... Dentro do paradigma POO (Programação Orientada a Objetos), um objeto nada mais é que uma representação do mundo real e um possui propriedades e métodos. Para entender melhor, vamos fazer a seguinte analogia: vamos considerar o objeto **PESSOA**. Uma pessoa possui as seguintes **propriedades**: *altura*, *peso*, *sexo* e *naturalidade*. Essas propriedades representam as características deste objeto. Vamos considerar também que este objeto também pode exercer algumas ações (*andar*, *dormir*, *comer*, *trabalhar* etc.), que no JavaScript chamamos de **métodos ou funções**. Para acessar a propriedade de um objeto basta colocarmos um ponto (.) entre o nome do objeto e o nome da propriedade (**pessoa.altura**). Para acessar um método de um objeto também colocamos um ponto (.) entre o nome do objeto e o nome do método, porém o método é seguido de parênteses (**pessoa.andar()**). Agora que já relembramos a diferença entre propriedade e método, vamos ver como é a sintaxe do *for... in*:

```
JS
1 for (variável in objeto)
2   instrução
```

Onde a **variável** representa a variável que você vai criar para percorrer **objeto** escolhido. Vamos ver como percorrer as propriedades do nosso objeto *pessoa*:

```
JS
1 * const pessoa = {
2   altura: 1.65,
3   peso: 60,
4   sexo: "feminino",
5   naturalidade: "RJ",
6 };
7 * for (const prop in pessoa){
8   console.log(prop);
9 }
```

Console

```
"altura"
"peso"
"sexo"
"naturalidade"
```

Primeiramente, criamos o objeto pessoa (*linha 1 a linha 6*) e depois criamos um *for* para percorrer o objeto criado, para isso, criamos uma variável cujo nome é *prop* que irá percorrer as propriedades do objeto pessoa. Repare que o *for... in* não percorre o valor que cada propriedade possui, mas, sim, cada propriedade declarada para o objeto pessoa. Se quisermos acessar os valores (atributos) de cada propriedade, basta usarmos *prop* com índice do nosso objeto (*pessoa[prop]*). Vejamos:

```

JS
1 * const pessoa = {
2   altura: 1.60,
3   peso: 60,
4   sexo: "feminino",
5   naturalidade: "RJ",
6 };
7 * for (const prop in pessoa){
8   console.log(pessoa[prop]);
9 }

```

Console

```

1.6
60
"feminino"
"RJ"

```

Se quisermos que o nome da propriedade aparece ao lado do seu respectivo valor, basta montarmos a *string* da seguinte forma:

```

JS
1 * const pessoa = {
2   altura: 1.60,
3   peso: 60,
4   sexo: "feminino",
5   naturalidade: "RJ",
6 };
7 * for (const prop in pessoa){
8   console.log(prop + "= " + pessoa[prop]);
9 }

```

Console

```

"altura= 1.6"
"peso= 60"
"sexo= feminino"
"naturalidade= RJ"

```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/ec06942681daf31c23c466b318e6a72aa232fa2c/Unidade2Aula4a>

O sinal de "+" foi utilizado para concatenar a propriedade juntamente com o seu valor, separados pelo sinal "=". Acesse o link acima e teste todas os exemplos vistos.

1.2. FOR... OF

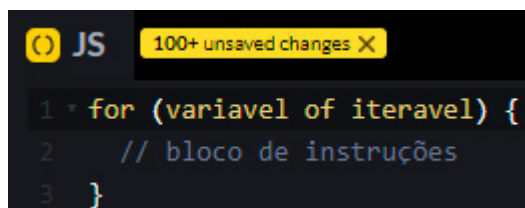
A estrutura *for... of* permite que você percorra um objeto iterativo (*Array, Map, Set*) e execute um bloco de instruções. É muito utilizado para percorrer *arrays* (vetores). Não se preocupe!

O conteúdo deste livro eletrônico é licenciado para GLEITON - 08303020692, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

Vamos aprender um pouco mais sobre *arrays* na próxima unidade. Saiba que o *for* por si só também percorre *arrays*, porém, com o *for... of*:

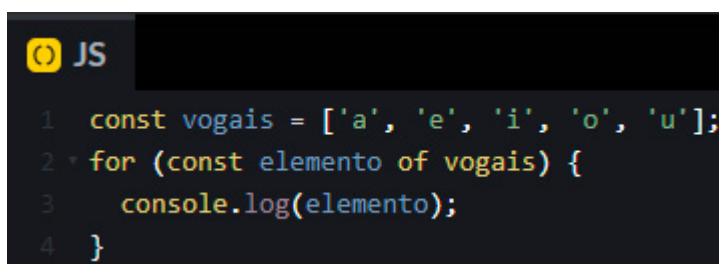
(...) você passará diretamente o *array* no qual será percorrido, e também uma variável que conterá o valor de cada posição do *array*. Ele sempre percorrerá o *array* por completo, sem você precisar fazer nenhuma verificação de tamanho do *array* ou colocar alguma condição específica (CERO, 2021).

Além disso, o uso do *for... of* torna o código mais simples e limpo. Veja como é a sua sintaxe:



```
JS 100+ unsaved changes X
1 for (variavel of iteravel) {
2   // bloco de instruções
3 }
```

No exemplo abaixo, vamos utilizar a estrutura *for... of* para percorrer um vetor chamado vogais:



```
JS
1 const vogais = ['a', 'e', 'i', 'o', 'u'];
2 for (const elemento of vogais) {
3   console.log(elemento);
4 }
```

Na *linha 1*, criamos um *array* chamado *vogais* e atribuímos os seguintes valores a ele: a, e, i, o, u. Na *linha 2*, criamos um *for... of* com uma variável chamada *elemento* que irá percorrer o *array* *vogais*. O método *console.log()* é responsável por exibir na tela os valores assumidos pela variável *elemento* em cada interação (clique ou rodada).

Também podemos utilizar o *for... of* para percorrer *strings*:

```

JS
1 const curso = "Gran Cursos";
2 for (const letra of curso) {
3   console.log(letra);
4 }
  
```

Console

```

"G"
"r"
"a"
"n"
" "
"C"
"u"
"r"
"s"
"o"
"s"
  
```

No exemplo acima conseguimos ter cada letra da string separada, usando o *for... of*.

CONSIDERAÇÕES FINAIS

Repetir um bloco de instruções é muito importante e faz parte da vida de um desenvolvedor. Nesta aula aprendemos mais uma estrutura de repetição (*for*) e suas variações. Vimos que o *for* é uma estrutura muito mais simples que o *while*. Podemos utilizar suas variações (*for... in* e *for... of*) para deixar o código mais legível e elegante, pois com elas não é necessário inicializar uma variável para controlar o fluxo de repetição e nem a incrementar ou decrementá-la para que a iteração ocorra. O próprio JavaScript se encarrega de saber quantos itens existem dentro do objeto/objeto iterativo a ser percorrido e, com base nisso, repete até que essa lista de itens seja toda percorrida.

MATERIAIS COMPLEMENTARES

Estruturas de repetição *for*, *for... in* e *for... of* Javascript:

<https://youtu.be/VGOhchtuQAc>

Diferenças entre *for... of* e *for... in*:

<https://youtu.be/CRYH34yc99U>

Loops *FOR IN* e *FOR OF* – Curso de Javascript – Aula 99:

https://youtu.be/nGt_9znTQoU

For... of:

O conteúdo deste livro eletrônico é licenciado para GLEITON - 08303020692, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...of>

For... in:

[https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for... in](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...in)

REFERÊNCIAS

CERON, Vitor. *Estruturas de repetição for, for... in e for... of Javascript*, 2021. Disponível em: <https://programandosolucoes.dev.br/2021/03/23/estruturas-de-repeticao/>. Acesso em: 12 de nov. de 2022.

FLANAGAN, David. *JavaScript: O guia definitivo*. Porto Alegre, RS: Bookman, 2013.

AULA 5 – IMERSÃO JAVASCRIPT – ESTRUTURAS DE SELEÇÃO E REPETIÇÃO (LINHA DE BOOTCAMPS)

OBJETIVO DA AULA

Praticar os conceitos vistos até o momento.

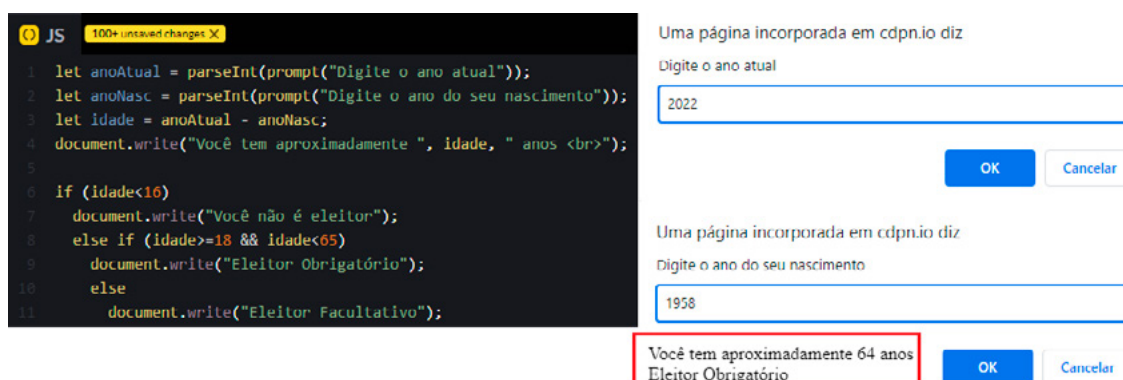
APRESENTAÇÃO

Nesta aula iremos colocar a mão na massa. Isso mesmo! Vamos fazer exemplos práticos juntos para testar os nossos conhecimentos em relação aos conceitos aprendidos nas aulas anteriores. Basicamente, vamos fazer um exercício de estrutura de seleção, um mesmo exercício utilizando as três estruturas de repetições aprendidas (*while*, *do... while* e *for*) e também vamos misturar a estrutura de seleção com a estrutura de repetição. Preparados? Vamos lá!

1. EXERCÍCIO 1 – ELSE... IF

Vamos começar fazendo um exercício clássico de estrutura condicional *if* encadeada: criar um *script* para solicitar o ano atual e o ano de nascimento de uma pessoa. Calcular sua idade aproximada e exibir a seguinte mensagem:

- Não eleitor (abaixo de 16 anos);
- Eleitor obrigatório (entre a faixa de 18 e menor de 65 anos);
- Eleitor facultativo (de 16 até 18 anos e maior de 65 anos, inclusive).



Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/c369aff823904a105fbcc4c5a6bb60790da6234f/Unidade2Aula5a>

Este mesmo exercício poderia ter sido feito usando as outras variações do *if* (*if*, *if-else*). Veja:

Livro Eletrônico

```

6  if (idade<16)
7      document.write("Você não é eleitor");
8  if (idade>=18 && idade<65)
9      document.write("Eleitor Obrigatório");
10 if (idade>=26 && idade<18 || idade>=65)
11     document.write("Eleitor Facultativo");

```

```

6  if (idade<16)
7      document.write("Você não é eleitor");
8  if (idade>=18 && idade<65)
9      document.write("Eleitor Obrigatório");
10 else
11     document.write("Eleitor Facultativo");

```

Você decidirá qual estrutura irá utilizar, lembrando que nem sempre podemos substituir uma variação pela outra, dependendo do caso a estrutura encadeada se faz necessário.

2. EXERCÍCIO 2 – SWITCH CASE

Agora vamos fazer um exercício usando a estrutura condicional *switch case*: criar um *script* que peça dois valores ao usuário e a operação que ele deseja executar (+, -, *, /). Execute a operação desejada e mostre o resultado na tela.

The screenshot displays a web application with a dark-themed code editor on the left and a light-themed user interface on the right. The code editor shows a JavaScript script using a `switch` statement to handle different operations. The UI consists of three input prompts: 'Digite o primeiro número:' (with value 10), 'Digite o segundo número:' (with value 2), and 'Digite a operação desejada:' (with value '/'). Each prompt has an 'OK' button and a 'Cancelar' button. The result of the operation, 'Resultado da operação 10/2 = 5', is displayed in a red-bordered box.

```

1  let num1 = parseInt(prompt("Digite o primeiro número:"));
2  let num2 = parseInt(prompt("Digite o segundo número:"));
3  let op = prompt("Digite a operação desejada: \n (+, -, *, /)");
4  switch (op){
5      case "+":
6          var resultado = num1 + num2;
7          break;
8      case "-":
9          var resultado = num1 - num2;
10         break;
11        case "*":
12            var resultado = num1 * num2;
13            break;
14        case "/":
15            var resultado = num1 / num2;
16            break;
17        default:
18            document.write("Opção Inválida");
19    }
20    document.write("Resultado da operação ", num1 + op + num2, " = ", resultado);

```

Uma página incorporada em cdpn.io diz
 Digite o primeiro número:
 10
 OK Cancelar

Uma página incorporada em cdpn.io diz
 Digite o segundo número:
 2
 OK Cancelar

Uma página incorporada em cdpn.io diz
 Digite a operação desejada:
 (+, -, *, /)
 /
 OK Cancelar

Resultado da operação 10/2 = 5

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/ec06942681daf31c23c466b318e6a72aa232fa2c/Unidade2Aula5b>

O conteúdo deste livro eletrônico é licenciado para GLEITON - 08303020692, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.



DESTAQUE

Neste exemplo, a variável resultado foi declarada como var, pois estamos usando a mesma variável em cada caso. Lembre que a variável declarada como var pode ser atualizada, desde que seja dentro do seu escopo? Se você não se lembra deste conceito, dê uma passadinha lá na unidade 1, combinado?

Agora vamos ver o mesmo exercício utilizando a estrutura condicional *if*:

```

JS 100+ unsaved changes X
1 let num1 = parseInt(prompt("Digite o primeiro número:"));
2 let num2 = parseInt(prompt("Digite o segundo número:"));
3 let op = prompt("Digite a operação desejada: \n (+, -, *, /)");
4 if(op=="+")
5     var resultado = num1 + num2;
6 else if (op=="-")
7     var resultado = num1 - num2;
8     else if (op=="*")
9         var resultado = num1 * num2;
10        else if (op=="/")
11            var resultado = num1 / num2;
12        else
13            document.write("Opção Inválida");
14 document.write("Resultado da operação ", num1 + op + num2, " = ", resultado);

```

Com o *if* encadeado obtivemos o mesmo resultado e economizamos 6 linhas no nosso código.

3. EXERCÍCIO 3 – ESTRUTURAS DE REPETIÇÃO

Agora vamos fazer um mesmo exercício utilizando todas as estruturas de repetição (*while*, *do... while* e *for*): crie um script que seja capaz de gerar a tabuada de qualquer número inteiro digitado pelo usuário. O resultado deve ser exibido da seguinte forma:

Tabuada:

2 X 1 = 2

2 X 2 = 4

...

2 X 10 = 20

- Utilizando o *while*:

```
JS 100+ unsaved changes X
1 var numero = parseInt(prompt("Digite um número:"));
2 var cont=1;
3 document.write("Tabuada: <br>");
4 while(cont<=10){
5     document.write(numero + " X " + cont + " = " + numero*cont + "<br>");
6     cont++;
7 }
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/a00bb37d4800c5f61e5c8a94310bd6e4ec48b89a/Unidade2Aula5c>

- Utilizando o *do... while*:

```
JS 100+ unsaved changes X
1 var numero = parseInt(prompt("Digite um número:"));
2 var cont=1;
3 document.write("Tabuada: <br>");
4 do{
5     document.write(numero + " X " + cont + " = " + numero*cont + "<br>");
6     cont++;
7 }while(cont<=10);
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/a00bb37d4800c5f61e5c8a94310bd6e4ec48b89a/Unidade2Aula5c>

- Utilizando o *for*:

```
JS 100+ unsaved changes X
1 var numero = parseInt(prompt("Digite um número:"));
2 document.write("Tabuada: <br>");
3 for(var cont=1; cont<=10 ; cont++)
4     document.write(numero + " X " + cont + " = " + numero*cont + "<br>");
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/a00bb37d4800c5f61e5c8a94310bd6e4ec48b89a/Unidade2Aula5c>

Todos os exemplos acima produzem o mesmo resultado:

```
Tabuada:
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Abaixo temos o mesmo exemplo da tabuada, só que agora utilizando uma caixa de texto, por isso a necessidade de utilizar o HTML.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Apostila JavaScript Progressivo</title>
5 <script>
6     function tabuada(){
7         var numero = parseInt(document.getElementById("numero").value);
8         var resposta = document.getElementById('resposta');
9         var tabuada='';
10
11         for(var cont=1; cont<=10 ; cont++)
12             tabuada += numero + " x " + cont + " = " + numero*cont + "<br />";
13
14         resposta.innerHTML = tabuada;
15     }
16 </script>
17 </head>
18 <body>
19     Tabuada do número: <input id="numero" type="text"><br>
20     <button onclick="tabuada()">Exibir</button><br>
21     Tabuada: <br>
22     <div id='resposta'></div>
23 </body>
24 </html>
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/a00bb37d4800c5f61e5c8a94310bd6e4ec48b89a/Unidade2Aula5c>

4. EXERCÍCIO 4 – EXERCÍCIO MISTO (CONDIÇÃO + REPETIÇÃO)

Para finalizar, vamos fazer um exercício que envolva as duas estruturas: crie um *script* que exiba na tela os 10 primeiros múltiplos de 3. Para isso, faremos uso do operador **MOD (%)**.

Este operador devolve como resposta o resto de uma divisão inteira. Por exemplo, ao dividir 5/2 o resultado é 2.5, correto? Ao utilizarmos o MOD o resultado seria 1, vamos entender...

$$\begin{array}{r} 5 \overline{)2} \\ 1 2 \\ \hline \end{array}$$

MOD

Como se trata de uma divisão inteira, o resultado é 2 e não 2.5. Ao multiplicar 2 por 2 temos como resultado 4 e para 5 falta 1. Deu para entender a aplicação do MOD? Ele é muito utilizado para saber se um número é par ou ímpar ou para saber se um número é múltiplo ou divisível por outro. Vamos ao nosso exercício:

JS 8 unsaved changes X

```

1  var inicio = 3;
2  var fim = 30;
3  while (inicio<=fim){
4      if (inicio % 3 == 0)
5          console.log(inicio);
6  inicio++;
7  }
```

JS 33 unsaved changes X

```

1  var fim = 30;
2  for (var inicio=3; inicio<=fim; inicio++)
3      if (inicio % 3 == 0)
4          console.log(inicio);
```

Fizemos o exercício usando o *while* e o *for*. Nos dois casos usamos a estrutura condicional *if* para testar se o resultado do MOD (%) era igual a zero, caso o resultado seja verdadeiro ele imprime na tela o valor da variável início naquela rodada, projetando o seguinte resultado.

Console
3
6
9
12
15
18
21
24
27
30



PRA PRATICAR

Faça um *script* para uma loja de tintas. O *script* deverá pedir o tamanho em metros quadrados da área a ser pintada. Considere que a cobertura da tinta é de 1 litro para cada 3 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam R\$ 80,00 ou em galões de 3,6 litros, que custam R\$ 25,00. Informe ao usuário as quantidades de tinta a serem compradas e os respectivos preços em 3 situações:

- comprar apenas latas de 18 litros;
- comprar apenas galões de 3,6 litros;
- misturar latas e galões, de forma que o preço seja o menor.

Acrescente 10% de folga e sempre arredonde os valores para cima, isto é, considere latas cheias.

Tente fazer sozinho, caso não consiga acesse este link para se basear: <https://gist.github.com/marcellobenigno/54c80d8f9886f262d94298c899f40183>.

Nele o código está em Python daí é só converter para o JavaScript.

CONSIDERAÇÕES FINAIS

Nesta aula, tivemos a oportunidade de colocar a mão e aprender uma pouco mais sobre as estruturas condicionais e de repetição. A ideia é que a partir de agora você consiga dar os seus próprios passos. Aproveite para acessar, explorar e editar os códigos que estão disponíveis na apostila. JavaScript não é difícil, mas a prática se faz necessária.

Resumindo, aprendemos nesta aula como usar as variações do *if*, como usar as estruturas de repetição e como usar as duas estruturas juntas. Além disso, vimos também como utilizar o operador MOD, que pode ser um grande aliado nos seus estudos daqui para frente.

MATERIAIS COMPLEMENTARES

Lista de exercícios 1:

https://oprogramador.bsb.br/aprenderjs_exercicios.php?page=1

Lista de exercícios 2:

<https://www.javascriptprogressivo.net/2018/12/Lista-Exercicios-Lacos-Loopings-WHILE-FOR.html>

REFERÊNCIAS

FLANAGAN, David. *JavaScript: O guia definitivo*. Porto Alegre, RS: Bookman, 2013.