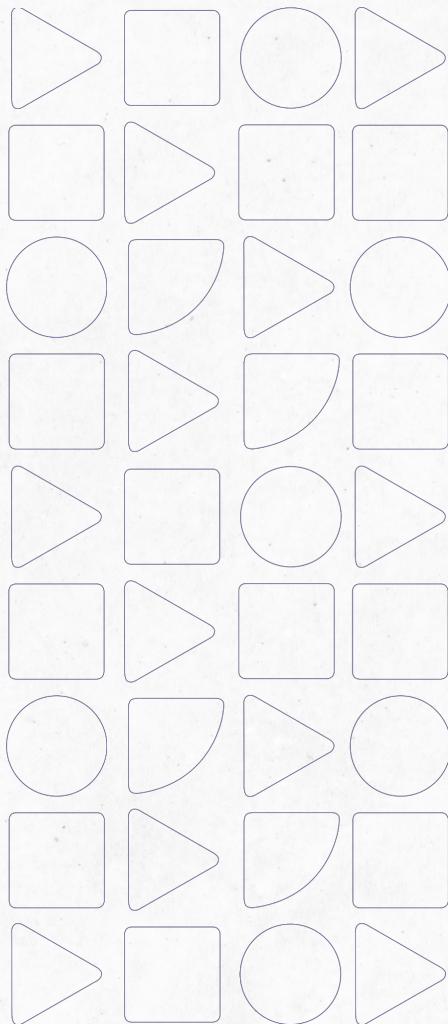


# Laços de Repetição

**Disciplina:** Linguagem de Programação



## Conteúdos:

Laços de repetição.

## Habilidade(s):

- Conhecer as estruturas de repetição;
- Analisar criticamente a aplicação do *loop* pertinente à cada situação;
- Identificar e reconhecer as palavras-chave.

# Bloco 1

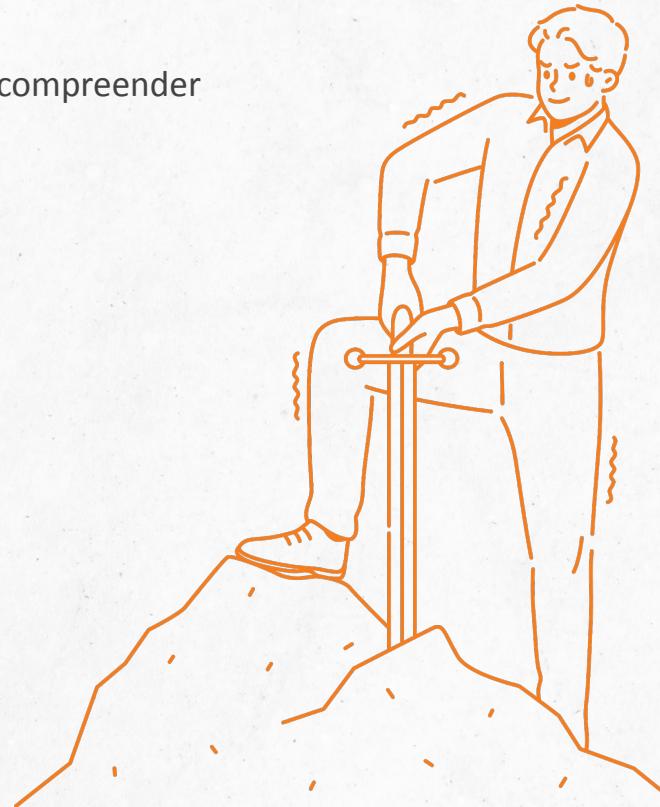
---

Entendendo o que são laços de repetição.

## Iniciando com um desafio!

Nesta aula, vocês participarão de uma dinâmica para que possam compreender mais facilmente o conceito da aula. Preparados?

Sem usar a calculadora, encontre a soma de todos os números pares de 1 a 20.



A resposta correta é:

110!



# Gostaram do desafio?

Sentiram dificuldade em realizar a dinâmica?

Agora, imagine ter que...

Calcular a soma de uma série geométrica complexa;

Descobrir a sequência de Fibonacci, na qual cada termo é a soma dos dois termos anteriores;

Encontrar a sequência de números que são quadrados perfeitos.



# Para fazer isso, existem os laços de repetição!

Eles permitem tornar automáticas as tarefas repetitivas e que levariam muito tempo para serem processadas.

Seja em uma simples contagem de 1 a 10 ou buscando descobrir os primeiros 100 números primos, é possível utilizar os **laços de repetição**.



# Benefícios dos laços de repetição

Permitem que você execute o mesmo conjunto de instruções várias vezes sem ter que escrever o código repetidamente;

São ideais para percorrer elementos em listas, objetos ou outras estruturas de dados. Isso é fundamental para processar informações em conjuntos de dados, como listas de contatos, registros de banco de dados e mais;

Podem ser usados para criar interfaces de usuário interativas, permitindo que os usuários forneçam entradas e executem ações repetitivas em resposta;

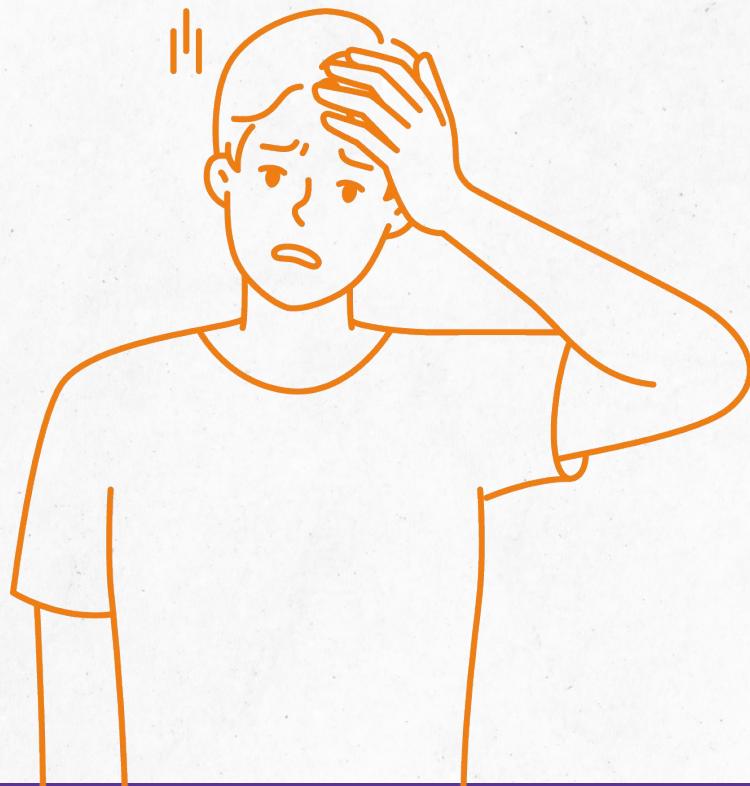
Em vez de duplicar código, você pode encapsular a lógica repetitiva em um *loop*, o que torna seu código mais limpo, legível e fácil de manter.



## É importante permanecer atento

Imagine que você quer assistir um filme de romance. Em vez de procurar na internet, decide criar um código que te recomende todos os filmes românticos possíveis. No entanto, por um descuido, você esquece de finalizar o código depois de, pelo menos, 20 repetições.

Isso cria um *loop infinito*, que ocorre quando uma condição nunca é satisfeita. O seu computador recomendará todos os filmes e repetirá o processo eternamente, o que pode gerar travamento e quedas de desempenho.



## Entretanto, um *loop* pode ser útil!

Quando bem executado, ele pode facilitar alguns processamentos de dados no código.

**Por exemplo**, você pode usar *loops* para verificar se a entrada de um usuário atende a certos critérios e continuar pedindo entrada até que esses critérios sejam satisfeitos. Isso acontece quando alguém tenta fazer um *login*, mas suas credenciais estão incorretas.





# Como criar um bom laço de repetição?

Existem três laços que são utilizados para criar bons laços de repetição, sendo eles:

*for*

*while*

*do while*

# Bloco 2

---

Explorando o laço de repetição *for*.

# Casa inteligente

Vamos realizar uma dinâmica?

Imagine que a turma está construindo uma casa inteligente. Nela, precisam ser desenvolvidas ideias de eletrodomésticos que facilitem a vida usuários.

O objetivo é realizar um *brainstorming* inspirador e criativo!



## Uma casa perfeita

Imagine que você está saindo para o trabalho e tem apenas trinta minutos para se arrumar e preparar o café da manhã.

Uma casa inteligente seria útil para fazer a sua torrada, preparar compras e, de quebra, lavar a sua louça, certo?





Mas...

Para tornar isso possível, é necessário programar um número finito de repetições. Já imaginou quantas torradas seriam preparadas se não houvesse um comando que interrompesse depois de um intervalo?



## Para isso, existe a estrutura *for* (para)

É uma estrutura de controle que permite executar um bloco de código repetidamente por um número específico de vezes.

Ela é frequentemente usada quando você sabe exatamente quantas vezes deseja que um determinado conjunto de instruções seja executado.



# Sintaxe da estrutura *for*

```
for (inicialização; condição; incremento/decremento) {  
    // Código a ser repetido  
}
```

**Inicialização:** é a parte na qual você configura uma variável de controle, atribuindo um valor inicial a ela;

**Condição:** é uma expressão booleana que é avaliada antes de cada iteração do *loop*. Se a condição for avaliada como verdadeira (*true*), o *loop* continua a ser executado. Se for avaliada como falsa (*false*), o *loop* é interrompido e a execução continua após o *loop*;

**Expressão final:** nesta parte, você atualiza a variável de controle, incrementando-a ou decrementando-a.

# Criando um laço de repetição na prática

O objetivo é criar um contador de 1 a 5.

1

**Inicialização:** configuramos uma variável de controle chamada “i” e atribuímos a ela um valor inicial. Aqui, vamos definir i como 1, pois queremos começar a contagem a partir de 1;

2

**Condição:** definimos a condição que será avaliada antes de cada iteração do *loop*. Queremos que o *loop* continue enquanto i for menor ou igual a 5. Portanto, nossa condição é *i <= 5*;

3

**Expressão final:** queremos que i seja incrementado em 1 após cada iteração. Isso é feito usando *i++*, que é uma forma abreviada de *i = i + 1*.

```
for (let i = 1; ...)
```

```
for (let i = 1; i <= 5; ...)
```

```
for (let i = 1; i <= 5; i++) {  
    console.log(i); // Isso imprimirá os números de 1  
    a 5 no console.  
}
```



### Se liga no que acontece!

A condição é avaliada como verdadeira (1 é menor ou igual a 5), então o código dentro do *loop* é executado, que é `console.log(i)`, imprimindo o valor de *i* (1) no console.

Em seguida, *i* é incrementado em 1, tornando-se 2.

O processo se repete até que *i* seja igual a 6, momento em que a condição se torna falsa (6 não é menor ou igual a 5) e o *loop* é encerrado.

# Estruturas *for...in* e *for...of*

Existem duas variações de *loop for* e elas são utilizadas para fins diferentes.

*for in*

É usado para percorrer as propriedades nomeáveis de um objeto. É como nomear diversos tipos de cartas de baralho como, por exemplo, “rei” e “rainha”.

*for of*

É usado para iterar sobre possíveis elementos e é mais adequado para percorrer os valores de coleções. É como enumerar as cartas de baralho como “1” e “2”.

```
for (variável in objeto) {  
    // Código a ser executado para cada  
    propriedade/enumerável  
}
```

```
for (variável of iterável) {  
    // Código a ser executado para cada  
    elemento do iterável  
}
```

# Observando na prática

*for in*

```
const pessoa = {  
    nome: 'João',  
    idade: 30,  
    cidade: 'São Paulo'  
};  
  
for (let chave in pessoa) {  
    console.log(`#${chave}: ${pessoa[chave]}`);  
}
```

*for of*

```
const numeros = [1, 2, 3, 4, 5];  
  
for (const numero of numeros) {  
    console.log(numero);  
}
```

No exemplo, o *for...in* percorre as propriedades do objeto *pessoa*, imprimindo o nome, a idade e a cidade.

No exemplo, percorremos os valores dos elementos do array “*numeros*” usando *for of*.

# Bloco 3

---

Vamos praticar o que vimos até aqui?

# Mão na massa

## Festa das celebridades

Separem-se em grupos e peguem um computador ou um *notebook*. Em seguida, desenvolvam um código em JavaScript para, em um painel, percorrer e imprimir a lista de famosos convidados para uma festa.

## Desafio

Como você utilizaria a estrutura *for* para solucionar esse problema?



# Bloco 4

---

Compreendendo o laço de repetição *while*.

# Jogo dos erros

Essa dinâmica se parece com o exercício “**jogos dos sete erros**”.

Aqui, você verá um código e terá que identificar quais são os seus principais erros. Vamos lá?

X

```
for (let i = 1; i <= 7; i++) {  
  console.log("Erro " + i);  
}
```

# Gabarito

E aí, acertou?



```
for (let i = 1; i <= 7; i++) {  
    console.log("Erro " + i);  
}
```



```
for (let i = 0; i < 7; i++) {  
    console.log("Erro " + (i + 1));  
}
```

O *loop* deve começar com *i* = 0 em vez de *i* = 1;

Falta um ponto e vírgula após *i* = 1;

A condição do *loop* deve ser *i* <= 7 para que ele repita sete vezes;

O *console.log* está exibindo "Erro" mais o valor de *i*, mas isso não é um erro real.

# Estrutura *while*

Diferente da estrutura *for*, ela executa um bloco de comandos enquanto uma condição especificada for verdadeira.

## Sintaxe básica

```
while (condição) {  
    // Bloco de comandos a ser executado enquanto a condição for verdadeira  
}
```

## Vamos observar um exemplo

Imagine que você deseja calcular a soma de números inteiros consecutivos até que a soma exceda um valor limite.

```
let soma = 0;  
let numero = 1;  
const limite = 100;  
  
while (soma < limite) {  
    soma += numero;  
    numero++;  
}  
  
console.log("A soma dos números consecutivos é: " + soma);  
console.log("O último número adicionado foi: " + (numero - 1));
```

O *loop* continuará a ser executado até que a condição `soma < limite` seja falsa. Assim que a condição não for mais atendida, o *loop while* será interrompido.

A large orange line-art illustration of a woman with long hair, wearing a white t-shirt and pants, shouting into a megaphone. Her mouth is wide open, and there are three sets of three short horizontal lines above her head, indicating sound or volume. The megaphone has a large orange handle and a white body. The background behind the woman is white, while the area behind the megaphone is a solid orange color.

## Importante

Para evitar *loops* infinitos, é fundamental sempre garantir que a condição de saída do *loop* seja alcançada em algum momento. Isso pode ser feito garantindo que a variável de controle (usada na condição do *loop*) seja modificada de forma que ela permita que a condição seja eventualmente falsa. Caso contrário, o *loop* continuará indefinidamente.

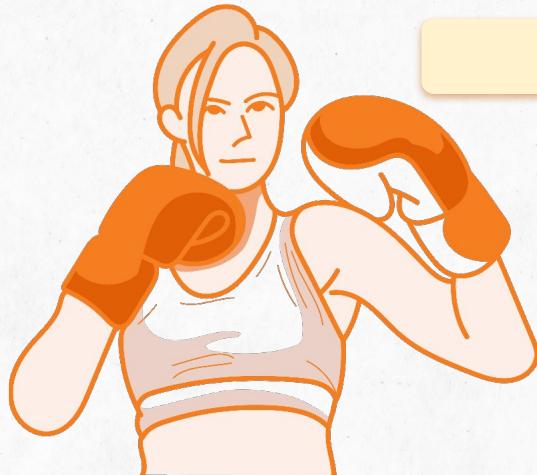
# Bloco 5

---

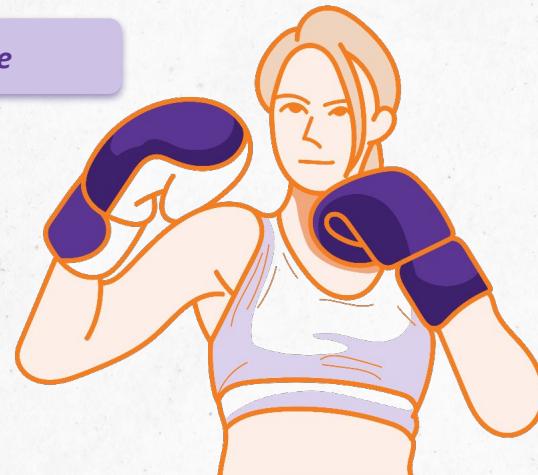
Compreendendo o laço de repetição *do while*.

# Estruturas *for* e *while*: qual é a diferença?

As estruturas *while* e *for* são duas estruturas de controle de fluxo que permitem a repetição de um bloco de código em JavaScript (e em muitas outras linguagens de programação), mas elas têm diferentes propósitos e sintaxes.



**VS**



# Gabarito

*for*

Usada quando você sabe exatamente quantas vezes deseja repetir;

Limitada pelo estado de repetição;

Tem uma sintaxe mais estruturada e é mais adequada para iterações com contagem fixa;

Útil quando você sabe o número exato de iterações no código.

**VS**

*while*

Usada quando você precisa repetir com base em uma condição;

Você precisa controlar a variável de condição explicitamente dentro do bloco de código;

Possui uma sintaxe fácil e flexível;

Útil quando o número de iterações não é conhecido antecipadamente.

## Estrutura do *while*

É semelhante ao *while*, mas com uma diferença fundamental: a condição é verificada após a execução do bloco de comandos.

Isso significa que o bloco de comandos sempre será executado pelo menos uma vez, mesmo que a condição seja falsa desde o início.

### Sintaxe

```
do {  
    // Bloco de comandos a ser executado  
} while (condição);
```

## Exemplo de uma estrutura *do while*

Um usuário precisa digitar um número positivo até que um número positivo seja inserido.

```
let numero;  
  
do {  
    numero = parseInt(prompt("Digite um número positivo:"));  
}  
    while (numero <= 0);  
  
console.log("Você digitou um número positivo: " + numero);
```

O bloco de comandos dentro do *do while* solicita que o usuário digite um número positivo repetidamente. O *loop* continuará a ser executado até que o usuário insira um número positivo (ou seja, a condição `número <= 0` seja falsa).

## Uso de *break*

É usado para interromper a execução de um *loop*. Pode ser usado em um *do while* da mesma forma que em outros tipos de *loops*, visando sair antecipadamente do *loop* se a condição for atendida. Aqui está um exemplo:

```
let contador = 1;

do {
    console.log(contador);
    contador++;

    if (contador === 4) {
        break; // Este loop será interrompido quando contador for igual a 4
    }
} while (contador <= 5);
```

# Bloco 6

---

Hora de praticar!

# Analisando redes sociais

Separem-se em cinco grupos e acessem os computadores e *notebooks* em sala de aula.

O desafio é que cada grupo desenvolva um *script* baseado em uma situação-problema apresentada, a fim de resolvê-la corretamente.

Cada grupo pode se sentir livre para escolher os temas a seguir.

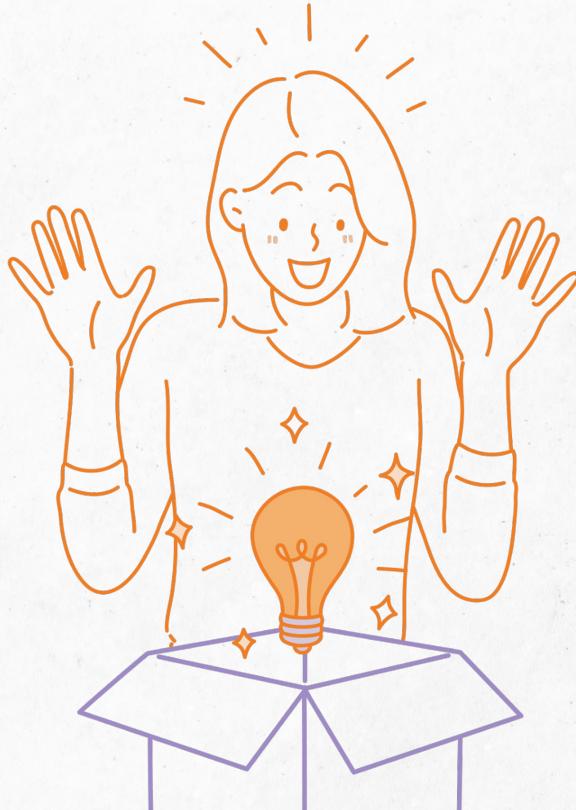
Você tem uma lista de notas de alunos e deseja calcular a média das notas.

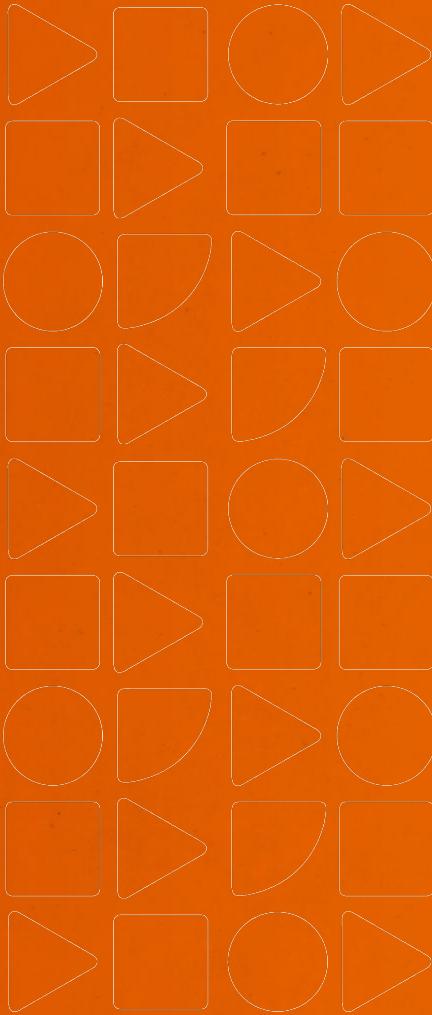
Você está lendo dados de um arquivo e deseja processar cada linha do arquivo.

Você está construindo uma interface de usuário interativa, na qual as ações do usuário precisam ser verificadas e processadas repetidamente.



Agora, o que você sabe sobre **laços de repetição**?





## Referências Bibliográficas

PROZ EDUCAÇÃO. *Apostila de Desenvolvimento para Dispositivos Móveis I.* 2023.