

Estruturas básicas de repetição

Vamos começar a falar sobre repetições e é muito importante que você entenda a condição para poder fazer uma repetição.

Até agora aprendemos o uso de comandos simples, o uso de operadores, os relacionais, os lógicos, os operadores aritméticos aprendemos a fazer algumas contas simples, vimos as condições, nós vimos três tipos de condições e agora vamos começar a aprender essas estruturas que a gente também pode chamar de laços ou iterações.

Em nossa vida existem vários momentos em que a gente faz repetições e um dos exemplos é comer uma pizza.

Vamos supor que você vai querer comer essa pizza inteira.

Criei a function comerPizza, que vai ter um bloco, então abro e fecho chaves, e eu vou começar minha atividade de comer essa pizza inteira.

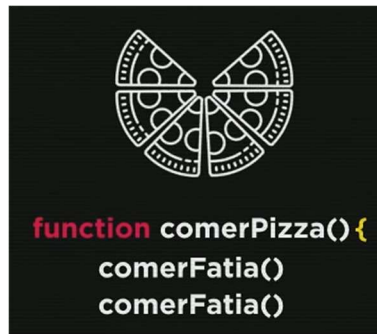
E para comer essa pizza inteira eu tenho que fazer alguns passos.



O primeiro passo que eu vou fazer aqui é comer a primeira fatia. Quando eu como a primeira fatia, já não está mais lá.



O próximo passo da minha atividade é comer outra fatia.



Você percebe que a minha pizza tinha 8 pedaços, tenho 8 comandos comer pizza.

```
function comerPizza() {  
  comerFatia()  
  comerFatia()  
  comerFatia()  
  comerFatia()  
  comerFatia()  
  comerFatia()  
  comerFatia()  
  comerFatia()  
}
```

A minha atividade é partir do primeiro comando até o último comando para cumprir a minha missão, que foi comer essa pizza inteira.

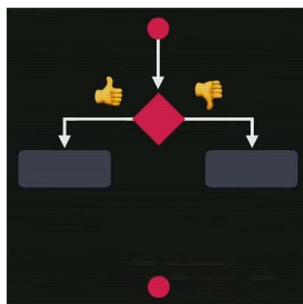
E com a programação também é assim, a gente parte de um ponto A e vai até um ponto B. A sua tarefa na hora de programar é conseguir escrever os programas para que você saia do ponto A. e chega ao ponto B que é o seu objetivo.

E para conseguir atingir os nossos objetivos existem várias formas, várias que a gente chama de **estruturas de controle**.

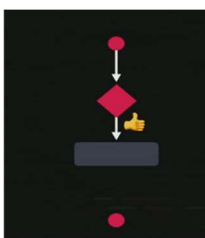
A primeira estrutura de controle, a mais simples de todas, é a sequência, onde eu vou desde o ponto A até o ponto B executando tarefas sequencialmente até chegar ao meu objetivo, que foi exatamente o que aconteceu na hora de comer pizza.

Foram colocados um passo depois do outro, um passo depois do outro, um passo depois do outro, até o momento em que eu comi a pizza inteira.

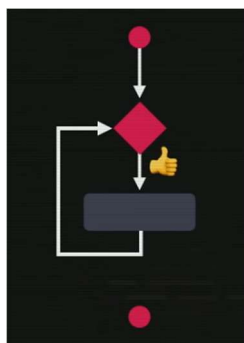
Só que nem tudo é uma maravilha, existem desvios, existem possibilidades, e como vimos nas condições, onde temos um **teste lógico**, que é representado por um losango, e esse teste tem duas possibilidades, verdade ou falso, e de acordo com esse valor, vão ser executadas tarefas específicas.



As repetições ou laços, começam exatamente como uma condição, vai testar uma expressão. E assim como acontecia nas expressões de condição, tenho duas possibilidades, ou esse teste é verdadeiro, ou ele é falso.

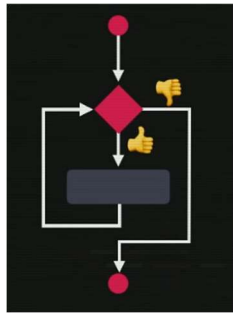


Só que o comportamento é um pouco diferente. Se essa condição, se esse teste lógico for verdade, ele vai executar um bloco, só que ele não vai seguir desse bloco para o ponto final, ele vai voltar ele vai fazer um loop, ele vai fazer um laço, uma repetição, voltando para o losango, ele vai testar de novo essa condição.



Se essa condição for verdadeira de novo, ele vai executar a tarefa e fazer outro laço e vai testar mais uma vez. Testando, sendo verdadeiro, ele vai executar esse bloco de novo e mais uma vez vai voltar. E ele vai fazer isso enquanto essa condição for verdadeira. Percebe na minha frase, **enquanto** essa condição, que é esse losango, for verdade.

A partir do momento em que esse losango não for verdade, o laço é quebrado e eu sigo meu fluxo natural.

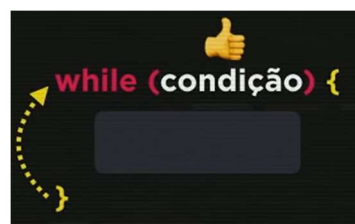


Então, essa é a nossa estrutura de laço principal, ela é a primeira que a gente sempre vê quando a gente vai estudar programação. Existem outras, mas essa é a mais comum de todas.

E para escrever essa estrutura que acabamos de apresentar em JavaScript é exatamente da mesma maneira de qualquer outra linguagem de programação, quando eu li essa estrutura, eu falei **enquanto alguma coisa for verdade**, essa palavra **enquanto** em inglês se escreve **while**.

Vou colocar while, é uma condição, que é exatamente a condição que eu testaria dentro do losango, quando escrevo um while, eu tenho que colocar um bloco, lembrando, em JavaScript blocos são sinais de chaves, tudo que estiver entre chaves é um bloco, então esse bloco que está relacionado a esse enquanto vai acontecer enquanto essa condição for verdadeira, sendo ela verdadeira todo o comando que foi escrito aqui dentro vai ser executado, depois que esse bloco for executado chegando na chave aqui debaixo ele vai voltar para o while, ele vai voltar para o enquanto e vai testar de novo a condição, se ela for verdadeira mais uma vez ele vai executar esse bloco e voltar.

```
while (condição) {  
  
}
```



E a partir do momento em que essa condição deixa de ser verdadeira e passa a ser falsa, aí o fluxo é desviado para o lado de fora.



Vamos voltar ao exemplo de comer pizza, que é uma repetição, vamos montar a nossa estrutura da função mais uma vez. Antigamente era comer fatia, comer fatia, comer fatia, comer fatia, 8 vezes.

E se eu dividir essa pizza em 16 vezes? E se eu dividir essa pizza em 20 vezes? Aquele algoritmo anterior, aquele meu programa anterior onde tinha comer fatia, comer fatia, comer fatia, 8 vezes, já não serve mais. E isso sem falar que tem uma repetição de comandos desnecessária.

Mas agora que você conhece a estrutura enquanto, a estrutura while, sempre que você encontrar a palavra while, leia como enquanto.

```
function comerPizza() {  
  while (temFatia())  
}
```



Enquanto tem fatia, tem fatia? Sim, tem fatia vou criar um bloco e dentro desse bloco vou colocar o comando comer fatia.

Então olhando, tem fatia? Tem. Ele come uma fatia e chegando no final do bloco ele vai fazer o nosso looping, vai fazer a repetição, o teste vai ser feito de novo, ainda tem fatia aqui em cima? Tem. Então ele come a fatia e volta de novo, esse mesmo processo de repetição está acontecendo, enquanto tem fatia ele está comendo as fatias.

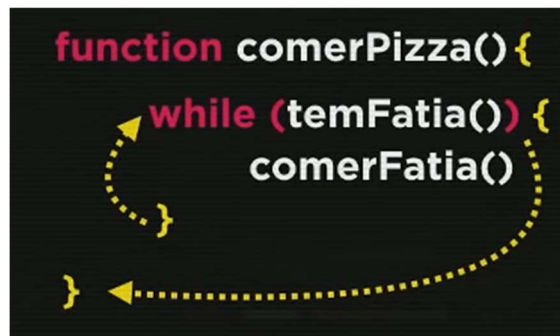
Até chegar o momento em que só tem uma fatia.



```
function comerPizza() {  
  while (temFatia()) {  
    comerFatia()  
  }  
}
```

Vamos fazer o teste de novo. Enquanto tem fatia, tem fatia? Tem, tem uma fatia. Ele vai comer a fatia e voltar de novo.

Pergunto mais uma vez, tem fatia? agora não tem fatia, ficou falso, então ele vai sair e terminou a minha função de comer pizza.



Vamos abrir o Visual Studio Code e fazer os nossos primeiros testes. Primeiro vamos fazer da forma sem repetição.

```
console.log('Tudo bem?')
console.log('Tudo bem?')
console.log('Tudo bem?')
console.log('Tudo bem?')
console.log('Tudo bem?')
console.log('Tudo bem?')
```

Ele escreveu, tudo bem, 6 vezes, mas se você quiser se escrever tudo bem, 500 vezes, você vai ter que ficar copiando sem necessidade.

Vamos fazer utilizando de inteligência e da estrutura que acabamos de aprender fazer o seguinte, vamos criar uma variável **c** que é um contador, esse contador vai começar com 1,

É a primeira vez que eu vou escrever, tudo bem.

Enquanto o contador for menor ou igual a 6, abro o bloco e fecho o bloco. Tudo que eu escrever aqui dentro vai acontecer enquanto isso for verdade, vou colar aquele comando, `console.log`, e vou fazer o seguinte um **c++**.

```
var c = 1
while ( c <= 6) {
  console.log('Tudo bem?')
  c++ // c = c + 1
}
```

Se você quiser, você ainda pode aprimorar mais ainda essa mensagem, vamos usar crases para a gente poder fazer a interpolação por placeholders, e fazer o seguinte:

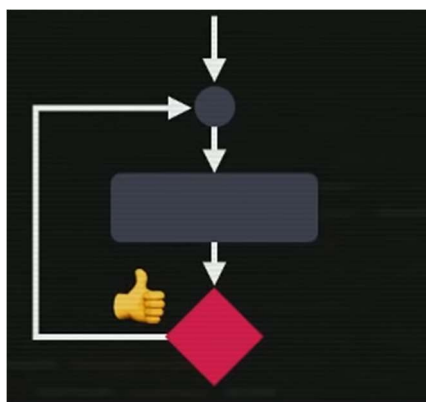
```
var c = 1
while ( c <= 6) {
  console.log(`Passo ${c}`)
  c++ // c = c + 1
}
```

}

Na hora que eu executo, ele escreve passo1, passo2, passo3, ...passo6, porque foram seis passos.

Sendo assim, essa estrutura, que é o **while**, ela é classificada como **estrutura de repetição com teste lógico no início**. Isso porque ele faz o teste, sendo verdadeiro ele faz o bloco.

Mas existe uma outra possibilidade, existe também a possibilidade em vez de eu fazer o teste lógico no início, que é testa, executa e faz looping, posso fazer também a estrutura ao contrário.

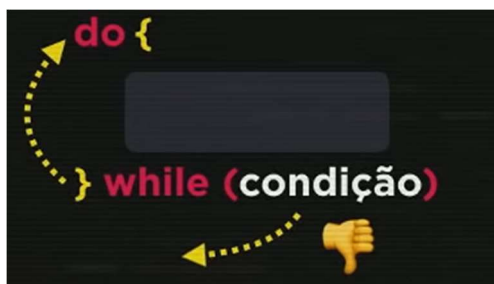


Primeiro eu executo o bloco e depois faço o teste, e a linha de raciocínio é igual, se o teste lógico for verdadeiro, ele faz o looping e executa o bloco de novo, e ele vai ficar nesse looping enquanto esse teste for verdade.

A partir do momento em que esse teste não é mais verdade, é mentira, o fluxo sai da estrutura de repetição.

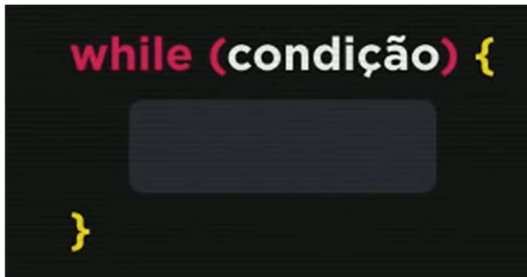
Eles são essencialmente estruturas de repetição, mas eles têm comportamentos diferentes. O primeiro testa e depois executa o bloco. O segundo, executa o bloco, depois ele testa.

E para fazer essa segunda estrutura é muito simples, em vez de **while** alguma coisa, você vai fazer **do**, quer dizer faça, abre e fecha o bloco, faça esse bloco enquanto a condição for verdadeira, vai ficar repetindo.



E a partir do momento em que aquela condição passar a ser falsa, ele sai do bloco e segue na execução do programa.

Dessa maneira, essas são as minhas duas primeiras estruturas de repetição dentro do JavaScript.

A screenshot showing the syntax for a while loop in JavaScript: `while (condição) {` followed by a dark rectangular box representing the loop body, and then `}`.A screenshot showing the syntax for a do while loop in JavaScript: `do {` followed by a dark rectangular box representing the loop body, and then `} while (condição)`.

A primeira, que é a **while**, chamamos **de estrutura de repetição com teste lógico no início**, a segunda, que é a **do while**, chamamos **de estrutura de repetição com teste lógico no final**.

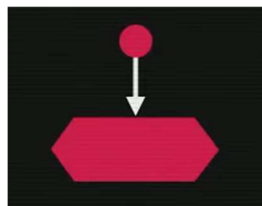
Vamos fazer uns testes nessa segunda estrutura.

Então olha só, vou criar um outro comentário aqui e vou colocar o código que a gente acabou de criar aqui em cima.

```
var c = 1
do {
  console.log(`Passo ${c}`)
  c++ // c = c + 1
}while ( c <= 6)
```

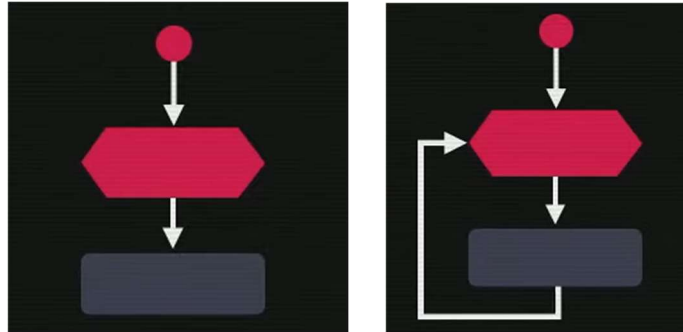
Repetição Com Variável De Controle

Agora vamos conhecer a estrutura de repetição com variável de controle, que é representada pelo hexágono irregular, nesse hexágono são feitas três coisas, inicialização, um teste lógico e o incremento.

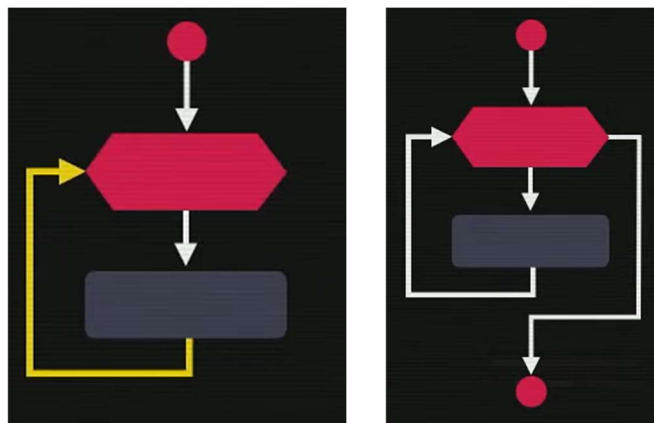


Na primeira passagem, faz a inicialização e o teste lógico, sendo esse teste lógico verdadeiro, vai executar o bloco e no bloco pode ter um ou vários

comandos, como a já vimos, e feito o bloco ele vai automaticamente voltar exatamente como a gente tinha feito nas estruturas while e do while, só que dessa vez no momento do looping ele também vai fazer o incremento uma das três partes que estão nesse hexágono irregular.



O teste lógico então é feito mais uma vez e sendo verdadeiro mais uma vez esse bloco vai ser executado e o looping acontecerá novamente e essa estrutura vai ficar acontecendo a cada vez que voltar eu incremento e faço o teste lógico e quando esse teste lógico ficar falso ele vai seguir o caminho de encerramento.



O comando que vamos utilizar é o comando for, é a estrutura for.

```
for (início ; teste ; incr){  
  
}
```

Como dito anteriormente, a estrutura **for** tem três partes, o **início**, que é a **inicialização**, um **teste lógico** e um **incremento** e tem um **bloco**, abre chave e fecha a chave.

```
for (inicio ; teste ; incr){  
      
}  

```

Na primeira passagem pelo **for**, vai fazer a **inicialização e o teste**, sendo esse **teste verdadeiro**, vai executar um **bloco**, a quantidade de comandos que existir, inclusive posso colocar dentro de um for, um while, um do while, um if, um switch, posso colocar qualquer estrutura que a gente viu das aulas anteriores até aqui, não existe limitação nenhuma para programação.

👍

```
for (inicio ; teste ; incr){  
      
}  

```

Uma vez executado esse bloco, ele vai voltar, só que enquanto ele está voltando, vai fazer o incremento e de novo vai fazer o teste lógico.

👍

```
for (inicio ; teste ; incr){  
      
}
```

Sendo esse teste lógico verdadeiro, ele executa o bloco de novo e volta. No que ele volta, ele faz um incremento de novo e faz o teste lógico mais uma vez. Isso vai acontecer enquanto esse teste lógico for verdadeiro. Se por acaso esse teste lógico for falso, ele vai sair da estrutura e segue seu fluxo natural de execução do nosso script.

👎

```
for (inicio ; teste ; incr){  
      
}
```

Para exemplificar vamos escrever um código que utiliza a estrutura while, que é a estrutura enquanto.

```
var c = 1
while (c <= 10){
  //Bloco
  C++
}
```

Comecei a variável começando com 1, então **c recebe 1**. Vou colocar uma estrutura enquanto e vou um teste lógico, **c** menor igual a 10, enquanto **c** for menor igual a 10, ele vai fazer uma determinada tarefa, vai fazer um bloco e vou colocar ali **c++**, significa **c** mais 1. É menor ou igual a 10, vai fazer, vai virar 3 e vai voltar, e vai fazer isso até passar de 10, quando ele passar de 10 ele sai.

Agora veremos como é que se reescreve esse mesmo código, só que agora utilizando a estrutura **for**. Lembrando que **for**, tem 3 áreas principais. A primeira área é a inicialização, a segunda parte do **for** é o teste lógico, **c** menor igual a 10, a terceira parte é o incremento, no nosso caso, o incremento é o **c++**.

```
for (var c = 1; c <= 10; c++){
  //Bloco
}
```

O código do **while** e o código do **for** tem exatamente a mesma funcionalidade, elas vão começar no 1 e vão terminar no 10, então tanto faz você escrever de uma maneira dou de outra, e muitos programadores, preferem utilizar o tipo de estrutura do **for** para grande maioria das vezes em que sei os limites das minhas execuções.

Vamos para o Visual Studio Code e vamos fazer os nossos testes práticos. Primeiro, utilizando while:

```
var c = 1
while (c <= 10){
  console.log(c)
  c++
}
```

Segundo utilizando do while:

```
var c = 1
do {
  console.log(c)
```

```
    c++  
}while (c <= 10)
```

Terceiro utilizando for:

```
for (var c = 1; c <= 10; c++){  
    console.log(c)  
}
```

Na hora de executar, eles fizeram a mesma coisa.