SENAI Serviço Nacional de Aprendizagem Industrial

Estruturas de Repetição

Introdução às Estruturas de Repetição

Imagine que você precisa executar a mesma tarefa várias vezes, como imprimir os números de 1 a 100, ou processar uma lista de itens. Fazer isso manualmente, escrevendo o código para cada repetição, seria tedioso e ineficiente. É aí que as estruturas de repetição entram em jogo. Elas nos permitem executar um bloco de código repetidamente, economizando tempo e tornando nossos programas mais concisos e poderosos.

Em JavaScript, as principais estruturas de repetição são `for` , `while` e `do while` . Cada uma delas tem suas particularidades e é mais adequada para diferentes cenários. Nesta aula, vamos explorar cada uma delas em detalhes, com exemplos práticos e aulas guiadas para você consolidar seu aprendizado.

A Estrutura `for`

O laço `for` é uma das estruturas de repetição mais comuns e versáteis. Ele é ideal quando você sabe de antemão quantas vezes um bloco de código precisa ser executado. O `for` é composto por três partes principais, separadas por ponto e vírgula, dentro dos parênteses:

1. **Inicialização:** Executada apenas uma vez no início do laço. Geralmente, é onde você declara e inicializa uma variável de controle (contador).
2. **Condição:** Avaliada antes de cada iteração. Se a condição for verdadeira, o bloco de código é executado. Se for falsa, o laço termina.
3. **Expressão Final (Incremento/Decremento):** Executada após cada iteração do bloco de código. Geralmente, é onde você atualiza a variável de controle (incrementa ou decrementa).

Sintaxe:

```
for (inicializacao; condicao; expressaoFinal) {
    // Bloco de código a ser executado repetidamente
}
```

Exemplo Básico: Contando de 1 a 5

```
for (let i = 1; i <= 5; i++) {
    console.log("Contagem: " + i);
}
```

Explicação:

`let i = 1;` : A variável `i` é inicializada com `1` .

`i <= 5;` : A condição é verificada. Enquanto `i` for menor ou igual a `5` , o laço continua.

`i++` : Após cada execução do bloco de código, `i` é incrementado em `1` . **Saída esperada:**

```
Contagem: 1
Contagem: 2
Contagem: 3
Contagem: 4
Contagem: 5
```

Exemplo: Iterando sobre um Array

O laço `for` é frequentemente usado para percorrer elementos de um array.

```
let frutas = ["Maçã", "Banana", "Laranja", "Uva"];

for (let i = 0; i < frutas.length; i++) {
    console.log("Eu gosto de " + frutas[i]);
}
```

Explicação:

`frutas.length` retorna o número de elementos no array (neste caso, 4).

O laço começa com `i = 0` (primeiro índice do array) e vai até `i < frutas.length` (ou seja, `i` será 0, 1, 2, 3).

Saída esperada:

```
Eu gosto de Maçã
Eu gosto de Banana
Eu gosto de Laranja
Eu gosto de Uva
```

A Estrutura `while`

O laço `while` é usado quando você quer repetir um bloco de código **enquanto uma condição específica for verdadeira**. Diferente do `for`, onde geralmente sabemos o número de iterações, o `while` é mais adequado quando o número de repetições é incerto e depende de uma condição que pode mudar durante a execução do laço.

Sintaxe:

```
while (condicao) {
    // Bloco de código a ser executado repetidamente
    // É crucial que a condição eventualmente se torne falsa
    // para evitar um loop infinito.
}
```

`condicao`: É uma expressão que é avaliada antes de cada iteração. Se for `true`, o bloco de código é executado. Se for `false`, o laço termina.

Exemplo Básico: Contagem Regressiva

```
let contador = 5;

while (contador > 0) {
    console.log("Contagem regressiva: " + contador);
    contador--; // Decrementa o contador para que a condição eventualmente se torne falsa
}
console.log("Lançar!");
```

Explicação:

A variável `contador` é inicializada com `5`.

O laço `while` continua enquanto `contador` for maior que `0`.

Dentro do laço, `contador` é decrementado em `1` a cada iteração. Isso é fundamental para que o laço não se torne infinito.

Saída esperada:

```
Contagem regressiva: 5
Contagem regressiva: 4
Contagem regressiva: 3
Contagem regressiva: 2
Contagem regressiva: 1
Lançar!
```

Cuidado com Loops Infinitos!

Se a condição de um laço `while` nunca se tornar falsa, o programa entrará em um **loop infinito**,

consumindo recursos e travando a aplicação. Certifique-se sempre de que há uma maneira de a condição se tornar falsa dentro do bloco do `while` .

```
 // Exemplo de loop infinito (NÃO EXECUTE ISSO SEM SABER COMO PARAR!) // let x = 1;
// while (x > 0) {
// console.log("Isso vai rodar para sempre!");
// // Faltou x--; ou alguma forma de x se tornar <= 0
// }
```

A Estrutura `do while`

O laço `do while` é semelhante ao `while` , mas com uma diferença crucial: o bloco de código é executado **pelo menos uma vez**, e a condição é avaliada **depois** da primeira execução. Isso significa que, mesmo que a condição seja falsa desde o início, o código dentro do `do` será executado uma vez.

Sintaxe:

```
 do {
    // Bloco de código a ser executado pelo menos uma vez
    // e repetidamente enquanto a condição for verdadeira.
} while (condicao);
```

`condicao` : É uma expressão que é avaliada após cada iteração. Se for `true` , o laço continua. Se for `false` , o laço termina.

Exemplo Básico: Execução Mínima Garantida

```
 let i = 0;

do {
    console.log("O valor de i é: " + i);
    i++;
} while (i < 5);

console.log("\n--- Exemplo com condição falsa inicial ---");

let j = 10;
do {
    console.log("O valor de j é: " + j);
    j++;
} while (j < 5); // Condição falsa desde o início
```

Explicação:

No primeiro `do while` , `i` começa em `0` . O bloco é executado, `i` se torna `1` , e a condição `1 < 5` é verdadeira, então o laço continua. Isso se repete até `i` ser `5` , quando a condição `5`

`< 5` se torna falsa e o laço termina.

No segundo `do while`, j começa em `10`. O bloco é executado uma vez (imprimindo `10`), j se torna `11`, e a condição `11 < 5` é falsa. O laço termina imediatamente após a primeira execução.

Saída esperada:

```
 O valor de i é: 0
O valor de i é: 1
O valor de i é: 2
O valor de i é: 3
O valor de i é: 4

--- Exemplo com condição falsa inicial ---
O valor de j é: 10
```

Quando usar `do while` ?

Use `do while` quando você precisa garantir que o bloco de código seja executado pelo menos uma vez, independentemente da condição inicial. Um caso de uso comum é para validação de entrada do usuário, onde você precisa pedir uma entrada e só então verificar se ela é válida.

`break` e `continue` : Controlando Laços

Dentro de laços de repetição (`for` , `while` , `do while`), podemos usar duas palavras chave especiais para controlar o fluxo de execução: `break` e `continue` .

`break`

A instrução `break` é usada para **encerrar imediatamente** o laço mais interno em que ela está. Assim que o `break` é encontrado, o laço é interrompido e a execução do programa continua na primeira instrução após o laço.

Exemplo com `break` :

```
 for (let i = 1; i <= 10; i++) {
    if (i === 5) {
        console.log("Número 5 encontrado! Encerrando o laço.");
        break; // Sai do laço quando i for 5
    }
    console.log("Contando: " + i);
}
console.log("Fim do programa.");
```

Saída esperada:

```
Contando: 1
Contando: 2
Contando: 3
Contando: 4
Número 5 encontrado! Encerrando o laço.
Fim do programa.
```

`continue`

A instrução `continue` é usada para **pular a iteração atual** do laço e ir para a próxima iteração. Quando o `continue` é encontrado, o restante do código dentro do bloco do laço para a iteração atual é ignorado, e o laço avança para a próxima avaliação da condição (no caso do `for`, para a expressão final; no `while` / `do while`, para a condição).

Exemplo com `continue`:

```
for (let i = 1; i <= 5; i++) {
    if (i === 3) {
        console.log("Pulando o número 3.");
        continue; // Pula esta iteração quando i for 3
    }
    console.log("Processando: " + i);
}
console.log("Fim do programa.");
```

Saída esperada:

```
Processando: 1
Processando: 2
Pulando o número 3.
Processando: 4
Processando: 5
Fim do programa.
```

Laços Aninhados

Assim como as estruturas de decisão, os laços de repetição também podem ser **aninhados**, ou seja, um laço pode estar dentro de outro laço. Isso é útil quando você precisa iterar sobre coleções de dados multidimensionais (como matrizes) ou quando a lógica do seu programa exige repetições dentro de outras repetições.

Quando você tem laços aninhados, o laço interno completa todas as suas iterações para cada uma das iterações do laço externo.

Exemplo: Imprimindo Coordenadas (Matriz 2x2)

Vamos usar laços aninhados para imprimir todas as combinações de coordenadas em uma grade 2x2.

```
 for (let linha = 0; linha < 2; linha++) { // Laço externo para as linhas for (let coluna = 0; coluna < 2;
    coluna++) { // Laço interno para as colunas console.log(`Coordenada: ($`{linha}, `${coluna})`);
    }
}
```

Explicação:

O laço externo (`linha`) executa duas vezes (para `linha = 0` e `linha = 1`).

Para cada vez que o laço externo executa, o laço interno (`coluna`) executa duas vezes (para `coluna = 0` e `coluna = 1`).

Saída esperada:

```
 Coordenada: (0, 0)
Coordenada: (0, 1)
Coordenada: (1, 0)
Coordenada: (1, 1)
```

Exemplo: Gerando um Padrão de Estrelas

Laços aninhados são frequentemente usados para gerar padrões.

```
 let numLinhas = 5;

for (let i = 1; i <= numLinhas; i++) {
    let linhaEstrelas = \'\';
    for (let j = 1; j <= i; j++) {
        linhaEstrelas += \'* \';
    }
    console.log(linhaEstrelas);
}
```

Saída esperada:

```
 *
* *
* * *
* * * *
* * * * *
```