Tornando o seu programa um robô incansável

Até agora nós vimos as estruturas básicas de um programa que são: variáveis, fluxo de execução, estrutura de decisão (IF) e também os operadores.

E chegou a hora de tornar o nosso programa mais avançado dando a ele mais poderes e conhecendo o que chamamos de estrutura de repetição.

Para que a gente possa entender direito, imagine que você aprontou todas na escola e ficou de castigo, então a professora nervosa decide a punição: "escrever 150 vezes na lousa".

Se você for "preguiçoso" como eu, logo pensaria em uma forma de automatizar isso, deixar um robô fazer o serviço as 150 vezes.

A estrutura de repetição (no contexto da programação) é necessário pelo mesmo motivo: executar comandos repetitivos sob uma determinada condição imposta pelo programador.

Esta condição (de parada) junto com seus operadores funciona idêntico ao IF.

A grosso modo seria algo dessa forma:

```
ENQUANTO x < 150 FAÇA
  ESCREVER("Nunca mais irei morder o cachorro da diretora!");
  x + 1;
FIM ENQUANTO
```

O pseudocódigo acima funciona assim: enquanto a estrutura não atingir a condição de parada ela fica executando (no caso escrevendo na tela).

A estrutura de repetição mais básica (WHILE)

Dentro da especificação do JavaScript podemos utilizar diversos tipos de estrutura de repetição. Como veremos o que varia é a finalidade de cada uma.

E das três mais básicas começaremos com o `while`. Que na prática é o "enquanto" do pseudocódigo acima.

Sua sintaxe básica é da seguinte forma:

```
while (x < 150) {
    console.log("Nunca mais irei morder o cachorro da diretora!");
    x = x + 1; // X assume o valore dele mesmo + 1.
}
```

Como você já percebeu, caso a condição de parada não seja atingida o *looping* ficará infinito. Ou seja, o robô executará isso até ocorrer algum evento externo (acabar a energia, ser desligado ou cair um meteoro em cima).

Essas condições infinitas ocorrem por 3 motivos principais:

1. O programador não adicionou um contador para cada passo do *looping*.
2. A condição de parada está malfeita ou é complexa demais (foi adicionado condição em cima de condição tornando-a impossível de ser atingida).
3. É intencional.

Na prática ficaria assim...

Erro 1: esquecer do contador de parada.

```
while (x < 150) {
    console.log("Nunca mais irei morder o cachorro da diretora!");
    // aqui iria o contador para o valor ser incrementado cada vez que for executado, por isso o valor de X nunca sairá de zero e a condição de parada nunca será satisfeita.
}
```

Ou até:

```
while (y < 150) {
    console.log("Nunca mais irei morder o cachorro da diretora!");
    x = x + 1; // o critério de avaliação está diferente do meu contador (variável trocada)
}
```

Erro 2: condições confusas e complexas

```
// veja a condição de parada alienígena, isso nunca será atingido...
while ((x < 150) && (y > 10) || (z == true)) {
    console.log("Nunca mais irei morder o cachorro da diretora!");
    x = x + 1;
}
```

Operadores do WHILE

Como você percebeu as condições de paradas obedecem exatamente a mesma estrutura e "jeitão" do IF. Ou seja, utilizam variáveis, valores, operadores de comparação e se você quiser até os operadores lógicos.

Por isso o `while` é uma das estruturas de repetição mais básicas e importantes da programação.

```
while ("<sua condição de parada aqui>") {
    // comando
    // contador
}
```

Nas condições de parada você pode utilizar o:

- Igual (`==`)
- Diferente (`!=`)
- Menor (`<`) ou Menor igual (`<=`)
- Maior (`>`) ou Maior igual (`>=`)

E:

- AND (`&&`)
- OR (`||`)
- NOT (`!`)

A grande sacada para entrar na sua cabeça é pensar assim: "ENQUANTO <CONDIÇÃO> EXECUTE". O segredo é ler e pensar com a palavra **enquanto** 😉

O contador do WHILE

A segunda parte do `while` mais importante é o contador. Ele que irá dar o ritmo da contagem (a cada *n* passos).

Para criar um contador na programação utilizamos o conceito de acumuladores de valores. Ficando assim:

```
var x = 0;

x = x + 1; // x era 0 e passou a ser 1
x = x + 1; // x era 1 e passou a ser 2
x = x + 1; // x era 2 e passou a ser 3
x = x + 1; // x era 3 e passou a ser 4...
```

A cada execução do *looping* você vai incrementando o seu valor.

Normalmente executamos/incrementamos 1 passo de cada vez. Mas também podemos a cada 2, 3 ou até decrescer de valor.

Veja:

```
while (x < 10) {
    // comando
    x = x + 1; // soma de 1 em 1
}
```

Ou...

```
while (x < 10) {
    // comando
```

```
    x = x + 2; // soma de 2 em 2
}
```

Ou até...

```
x = 10;

while (x > 0) {
    // comando
    x = x - 1; // diminuí de 1 em 1
}
```

Uma forma mais "esperta" que você irá também ver nos meus programas é o contador escrito dessa forma:

```
while (x < 10) {
    // comando
    x++; // isso é um atalho para x = x + 1
}
```

Ou quando for de 2 em 2 (ou mais):

```
while (x < 10) {
    // comando
    x+=2; // isso é um atalho para x = x + 2
}
```

Ou quando for para decrescer de 1 em 1:

```
x = 10;

while (x > 0) {
    // comando
    x--; // isso é um atalho para x = x - 1
}
```

Ou quando for para decrescer de 2 em 2:

```
x = 10;

while (x > 0) {
    // comando
    x-=2; // isso é um atalho para x = x - 2
}
```

O contador também é útil para ser utilizado como valor no comando. Veja:

```
while (x < 10) {
    console.log("Valor de X: " + x);
    x++;
}
```

Ou:

```
while (x < 10) {
    y = x *2;
    console.log("Valor de Y: " + y);
    x++;
}
```

Criando um looping infinito de propósito

As vezes precisamos que o *looping* seja infinito. Ou seja, que o tal robô fique em execução analizando alguma coisa que não tem um tempo certo para acabar.

Por exemplo, um forno elétrico. O programinha do termostato deve ficar em execução até a condição de temperatura de 180 graus seja atingida:

```
ligar(); // inicia

while(temperatura < 180) {
    console.log("Temperatura atual: " + temperatura);
    temperatura = ler_temperatura(); // a leitura da temperatura faz o
papel do contador
}

desligar(); // desliga o forno após a temperatura ser atingida (while sair
do looping)
```

Em outros casos o nosso *looping* infinito pode estar mais explicito, sendo assim:

```
while(true) {
    // comando
}
```

Como a condição de parada se comporta como o IF, ela só precisa ter um retorno verdadeiro, ou seja, `true` para estar apta a continuar. Que nesse caso está fixa.

Criando uma parada forçada

Muitas vezes (em *looping* infinito principalmnente) precisamos forçar uma parada. Essa interrupção é executada pelo comando `break`.

Vejamos um exemplo prático envolvendo o forno elétrico:

```
ligar(); // inicia

while(temperatura < 180) {
    console.log("Temperatura atual: " + temperatura);
    temperatura = ler_temperatura(); // a leitura da temperatura faz o
papel do contador
    minutos = ler_tempo(); // a leitura do tempo faz o papel de saída
forçada

    if (minutos >= 60) {
        console.log("O forno pode estar com problemas...");
        break; // <<<-- aqui executa a parada forçada do while
    }
}

desligar(); // desliga o forno após a temperatura ser atingida (while sair
do looping)
```

O comando `break` sempre deve estar dentro de algum *looping* e é muito importante para ser como uma condição de saída alternativa (normalmente por erro).

Desafio

No nosso exemplo da calculadora criar uma rotina que fique sempre perguntando os valores até que seja digitado a letra N em qualquer valor (indicando uma condição de parada).

Resposta

```
while(true) {
    var x = prompt("Digite o primeiro número ou N");
    var y = prompt("Digite o segundo número ou N");

    if (x == "N" || y == "N") {
        break;
    }

    var resultado = Number(x) + Number(y);

    alert("O resultado da soma é: " + resultado);
}
```