EXERCÍCIO DIRIGIDO
Estruturas de Repetição

1) Por que utilizar laços?

A programação de computadores tem como principal meta a execução de tarefas repetitivas. Muitas vezes há pedaços de códigos que são repetidos inúmeras vezes. Assim, faz-se necessário estruturas de codificação que permitam executar essas repetições, sem que o programador precise reescrever códigos incessantemente.

Veja um exemplo a seguir de uma codificação repetitiva:

Vamos escrever um script que mostre na saída padrão os números de 1 a 5.

```
console.log("Numero: 1")
console.log("Numero: 2")
console.log("Numero: 3")
console.log("Numero: 4")
console.log("Numero: 5")
```

Perceba que as linhas do *script* são repetidas 5 vezes com apenas o número variando de 1 até 5. Todo o conteúdo restante é repetido. Aqui estamos falando de apenas 5 linhas. Agora imagine se tivéssemos centenas delas?

Vejamos agora como ficaria a codificação utilizando uma estrutura de repetição:

```
var cont:number;

for(cont=1;cont<=5;cont++){
     console.log("Numero: "+cont);
}
```

Ao invés de digitarmos as 5 linhas do comando “console.log”, utilizamos um comando de laço chamado “*for*” para fazê-lo. A finalidade deste exemplo é apenas mostrar como um laço funciona e o quanto pode ser útil. Na sequência analisaremos a estrutura “for” comando a comando, demonstrando como o computador “pensa” para executar um laço.

Inicialmente focaremos a utilização dos laços apenas para contar números. Ao final mostraremos o quanto mais poderá ser feito utilizando-se laços.

1) **Exercício:** Tente alterar o script do segundo exemplo para que sejam mostrados números de 1 até 11 na tela.

Veja a seguir a sintaxe básica do comando “for”:

```
for(var cont:number=1;cont<=11;cont++){
      <seu código a repetir escrito aqui>
}
```

Todo comando “**for**” utiliza uma variável contadora. Aqui, a variável utilizada foi “**cont**” (mas poderíamos ter escolhido qualquer nome para ela). Essa variável tem muitas funções. A primeira parte do laço diz ao computador que atribua o valor inicial de 1 à variável. Em seguida testa esse valor nos limites atribuídos, que neste caso são de 1 até 11, ou seja, os valores possíveis que serão atribuídos à variável serão inteiros entre 1 e 11, inclusive. Se o valor atribuído à variável estiver dentro dos limites estabelecidos as linhas dentro do laço, que estarão entre as chaves do bloco {}, serão executadas.

2) **Exercício:** Execute novamente o script 2, só que desta vez altere o início do laço para 5 e o final para 11. Veja o que acontece.

Até aqui já sabemos como controlar o início de um laço "for", vejamos agora como ele termina.

Neste exemplo nosso laço iniciará o valor de **cont** em 5 e continuará executando o código inserido até que seja igual a 11. Assim, enquanto o valor de **cont** for 5, 6, 7 e assim, sucessivamente até que seja 11, o laço continuará sua execução. No entanto, assim que o valor de **cont** for maior que 11, o laço encerrará sua execução e o script continuará na linha seguinte após a chave de fechamento do laço.

```
var i:number;

for(i=5;i<=11;i++){
     console.log(i);
}
```

3) **Exercício:** Sabemos que o exemplo anterior contará o valor de "i" de 5 até 11. Altere o exemplo para que inicie a contagem em 4 e vá até o valor de 23, inclusive. Ou seja, não queremos que o valor mostrado no console seja acima de 23.

Agora que já sabemos controlar os limites do comando "for", que tal aprendermos a controlar o que acontece dentro dos laços?

Nos exemplos que vimos até agora, o valor das variáveis de controle ou contadoras variava de 1 em 1, do início até o limite estabelecido.

Vejamos agora como mudar esse incremento e como utilizar o laço para decrementar um valor nessas variáveis:

a) Para incrementar com valores diferentes de 1, utilizamos o operador de incremento (+=x), onde x, é o incremento que queremos para a variável contadora;
b) Se quisermos decrementar os valores possíveis para a variável contadora, iniciamos a contagem em um número maior que o limite e utilizamos o operador de decremento (-=x), atribuindo a "x" o valor a ser decrementado. Veja os exemplos:

```
var i:number;

// Exemplo 4
for(i=0;i<=50;i+=2){   // valores de i de 2 em 2 => {0,2,4,6,8,...,48,50}
     console.log(i);
}

// Exemplo 5
for (i=10; i>=1;i--) { // valores de i de 1 em 1 em ordem decrescente =>
                       // {10,9,8,...,3,2,1}
     console.log(i);
}
```

4) **Exercício:** Altere o script "Exemplo 4" para que conte de 5 até 50 de 5 em 5.

5) **Exercício:** Altere o script "Exemplo 5" para que conte de 20 até 0 de -2 em -2.

Até aqui vimos três etapas da sintaxe do comando “for”. Mas como será que realmente o comando controla o computador. Ou seja, como será que o computador “pensa” ao executar esse código?

Vamos imaginar que o computador executará o seguinte script:

```
var i:number;

for (i=2;i<=13;i++){
     console.log(i);
}
```

1. Inicia com i = 2;
2. Testa a condição para saber se i está entre os limites estabelecidos pelo laço, que estão compreendidos entre 2 e 13, inclusive. Assim, como i = 2, isso é verdadeiro e o laço continua sua execução;
3. A incrementação não é feita neste instante. Ao invés disso, se a condição foi verdadeira o código dentro do laço é executado. Neste caso, o comando console.log(i) será executado e o valor de “i” será mostrado no console;
4. Uma vez terminada a execução do código dentro do laço, será acionada a incrementação/decrementação da variável contadora. Neste caso, como se utilizou o padrão de incrementação de 1 em 1, o valor da variável “i” será adicionado de 1 e passará ser i = 3;
5. Agora i = 4. O comando “for” checará se o valor é menor ou igual ao limite, nesse caso 13. Se for verdadeiro, ou seja, o valor menor que 13, continuará a rodar o código dentro do laço;
6. Com o código dentro do laço executado, passará à fase de incremento/decremento, e assim serão repetidos esses passos, sucessivamente, até que o valor de “i” seja maior que 13;
7. Sendo o valor de “i” maior que o limite estabelecido, o laço é encerrado e o script continuará sua execução logo após a chave de fechamento, se existir.

6) **Exercício:** Faça um script que inicie a contagem em 8 e vá até 120, inclusive, com incremento de 12 em 12, mostrando o valor da variável a cada volta no console.

Nem sempre teremos uma variável contadora que controle os limites de execução de um laço. Em algumas situações talvez seja necessário interromper as voltas de um laço com alguma condição que não seja um valor. Para esse casos, existe outro tipo de laço de repetição chamado **while**.

O laço **while** é uma estrutura de controle que permite executar repetidamente um bloco de código enquanto uma condição especificada for verdadeira.

A sintaxe básica é a seguinte:

```
while (condição) {
    // bloco de código a ser executado
}
```

- condição: Uma expressão booleana que é avaliada antes de cada iteração do laço.
- bloco de código: O código que será executado repetidamente enquanto a condição for verdadeira.

Funcionamento

1. Avaliação da condição: antes de cada iteração, a condição é avaliada.
2. Execução do Bloco de Código: se a condição for verdadeira, o bloco de código é executado.
3. Reavaliação: após a execução do bloco de código, a condição é reavaliada.
4. Terminação: O laço termina quando a condição se torna falsa.

Vejamos um exemplo simples onde usamos um laço **while** para contar de 0 a 4 e mostrar esses números no console:

```
let contador: number = 0;

while (contador < 5) {
    console.log(contador);
    contador++;
}
```

1. Inicialização: a variável contador é inicializada com o valor 0.
2. Condição: o laço **while** verifica se contador é menor que 5.
3. Execução: se a condição for verdadeira, o valor de contador é mostrado no console.
4. Incremento: o valor de contador é incrementado em 1.
5. Repetição: o processo se repete até que contador seja igual a 5, momento em que a condição se torna falsa e o laço termina.

Vejamos agora um exemplo de **while** que não usa uma variável contadora para determinar a continuidade do laço.

Neste exemplo, usaremos um *array* e removeremos elementos até que esteja vazio:

```
let frutas: string[] = ["maçã", "banana", "abacaxi", "laranja"];

while (frutas.length > 0) {
    console.log(frutas.pop()); // remove e retorna o último elemento
                               // do array
}
```

Neste código, o laço **while** continua a executar enquanto o *array* “frutas” tiver elementos. A cada iteração, ele remove o último elemento do *array* usando o método “pop()” e o mostra no console. O laço termina quando o *array* estiver vazio.

7) **Exercício**: Calcule a soma dos inteiros de 1 a 10 utilizando o laço de repetição while.

Quando tentamos utilizar a rotina de entrada de dados, percebemos um problema com o o código que é o laço **while** não esperar a resposta do usuário antes de continuar para a próxima iteração. Isso ocorre porque a função *question* do módulo *readline* é assíncrona, mas o laço **while** é síncrono. Como resultado, o laço **while** continua executando indefinidamente sem esperar pela entrada do usuário.

Para resolver isso, podemos usar uma função recursiva para lidar com a entrada do usuário de forma assíncrona. Aqui está um exemplo de como fazê-lo:

```
var readline = require('readline');
var leitor = readline.createInterface({
    input: process.stdin,
```

```
    output: process.stdout
});

let continuar = true;

function perguntar() {
    if (continuar) {
        leitor.question("Digite uma palavra (ou 'sair' para
            terminar): ", function(answer) {
            if (answer.toLowerCase() === 'sair') {
                continuar = false;
                leitor.close();
            } else {
                console.log('Você digitou: ${answer}');
                perguntar(); // Chama a função novamente para
                             // continuar perguntando
            }
        });
    }
}

perguntar(); // Inicia o processo de perguntas
```

Explicação

1. **Função Recursiva**: A função “perguntar” é chamada recursivamente após cada resposta do usuário, garantindo que o programa espere pela entrada do usuário antes de continuar.
2. **Condição de Parada**: Se o usuário digitar “sair”, a variável “continuar” é definida como *false* e a interface *readline* é fechada.
3. **Chamada Inicial**: A função perguntar é chamada inicialmente para iniciar o processo de perguntas.

Dessa forma, o programa espera pela entrada do usuário antes de continuar, resolvendo o problema do laço **while** síncrono.

8) **Exercício**: Peça ao usuário que digite uma palavra qualquer. Mostre no console a quantidade de vogais e consoantes existentes na palavra informada.

9) **Exercício**: Escreva um script que aceite a entrada de um número inteiro do usuário e determine se o número é primo ou não. Se o número não for primo, exiba os únicos fatores primos do número. Lembre-se de que os fatores de um número primo são somente 1 e o próprio número primo. Cada número que não é primo tem uma fatoração em primos única. Por exemplo, considere o número 54. Os fatores primos de 54 são 2, 3, 3 e 3. Quando os valores são multiplicados, o resultado é 54. Para o número 54, a saída dos fatores primos deve ser 2 e 3.

10) **Exercício**: Peça ao usuário para digitar 10 números inteiros de 1 a 100. Ao término da digitação, mostre no console, qual o menor e o maior número informado, a somatória dos 10 números, listando-os em ordem crescente. O script não deve aceitar o mesmo número duas vezes.