6. Estruturas de Repetição

Neste capítulo você aprenderá a:

- Entender a funcionalidade de uma estrutura de repetição.
- Utilizar comandos de repetição em *javascript*.
- Aplicar condições de parada de um fluxo de repetição.

6.1 Repetição de Fluxo

Em muitas situações é interessante que consigamos repetir um fluxo, reaproveitando, assim, um código. Seja para repetir tudo, por exemplo: (1) fazer um jogo ter várias partidas, ou (2) fazer a tabuada da multiplicação do 9, onde cada linha o código poderia imprimir o valor da multiplicação do 9 por um número de 1 a 10. A ideia é a seguinte, pensando no exemplo (2), ao invés de colocar um `console.log()` para imprimir cada multiplicação de 9 pelos números de 1 a 10, podemos reaproveitar o que se repete e alterar somente os números que serão multiplicados. Esse fluxo de repetição é usualmente chamado de *laço*.

Vamos estudar três estruturas de repetição que possuem características bem semelhantes: **enquanto**, **faça-enquanto**, **desde que-até**. Da mesma forma que fiz com as estruturas de decisão, irei colocar os fluxogramas para que consiga compreender melhor essas estruturas.

6.2 Estrutura Enquanto

A estrutura **enquanto** tem a característica de verificar uma condição, como no **se**, e caso seja verdade seu escopo será executado. Após finalizar a execução do escopo o fluxo volta para antes da verificação da condição para ser verificado novamente, como ilustra a Figura 6.1. A ideia dessa estrutura é voltar a executar o escopo sempre que a condição for verdadeira, logo, você terá que ter um *critério de parada*, caso contrário seu programa ficará executando a estrutura de repetição indefinidamente. Falarei desse *critério* em um exemplo.

Vamos trabalhar com o exemplo da tabuada de multiplicação de nove, mas vamos deixar mais genérica essa aplicação. O usuário irá informar qual número ele quer que o

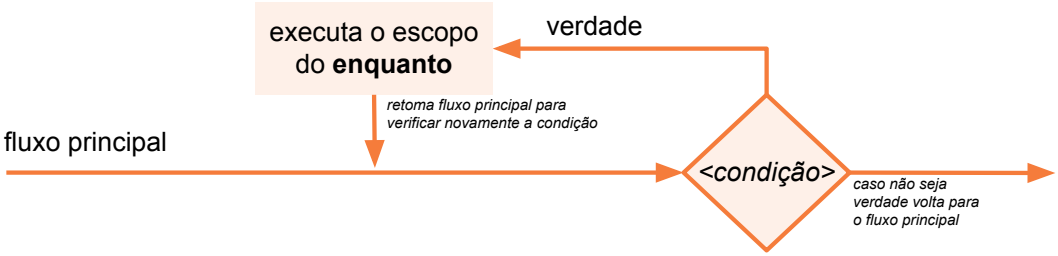

Figura 6.1: Fluxograma da estrutura do **enquanto**.

programa faça a tabuada e então mostramos na tela a tabuada de multiplicação deste número pelo número 1 até 10, certo? Vamos ver o fluxograma disso na Figura 6.2.

Observe na Figura 6.2 que a variável `multiplicador` é incrementada dentro do fluxo do **enquanto**. Em um determinado momento o laço será encerrado, ou seja, quando `multiplicador` for igual a 11. Isso é o tal *critério de parada*, um critério no qual a condição do **enquanto** não será satisfeita. Visto que cada critério vai depender exclusivamente do seu problema/solução. Certo!?

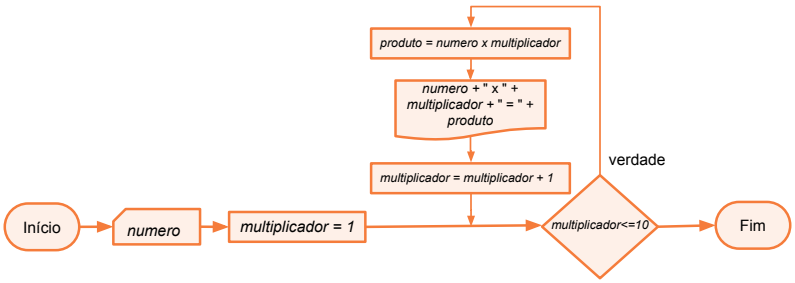

Figura 6.2: Fluxograma da estrutura do **enquanto** do exemplo da tabuada.

Agora vamos ver essa estrutura no *javascript*:

```
1 while (<condição>) {
2     <instruções>
3 }
```

Nosso **enquanto** equivale ao comando `while`, a condição fica logo após o inserirmos no código e as chaves definem o seu escopo. Da mesma forma que nas estruturas condicionais as chaves delimitam o escopo. Bem simples, né? Vejamos agora o código completo da tabuada no *javascript*.

■ **Código 6.1** Tabuada com `while`[1] (Fluxograma da Figura 6.2).

```
1 var numero = Number(prompt("Informe qual tabuada de multiplicação você
      quer (entre 1 e 10)"));
2 var multiplicador = 1;
3 while(multiplicador<=10){
4   produto = numero * multiplicador;
5   console.log(`${numero} x ${multiplicador} = ${produto}`);
6   multiplicador = multiplicador + 1; // pode substituir por
        multiplicador++;
7 }
```

■

[1] <https://replit.com/@DaniloBorges/EstruturaWhile-Ex1>

No Código 6.1 você pode perfeitamente substituir a linha 6 por `multiplicador++`, pois produzirá o mesmo efeito. Execute esse código no Replit para ver o funcionamento dessa estrutura.

6.3 Estrutura Faça-Enquanto

Esta estrutura tem um funcionamento similar ao `while` a diferença é que o que desejamos repetir será executado pelo menos uma vez, já no `while` o que queremos repetir só será executado se a condição for verdadeira. Na Figura 6.3, podemos ver que é isso que acontece no **faça-enquanto**. Primeiro ele faz o que queremos, depois ele verifica a condição e se for verdade ele executa o escopo novamente.

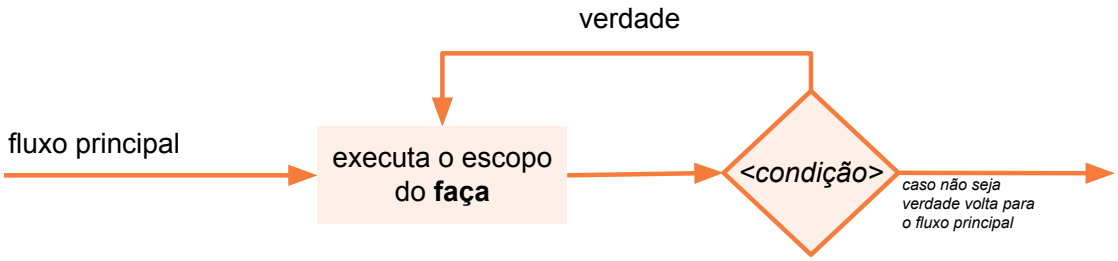

Figura 6.3: Fluxograma da estrutura do **faça-enquanto**.

Essa estrutura no *javascript* também é simples e semelhante ao `while`:

```
1 do {
2     <instruções>
3 } while (<condição>);
```

Observe que o `do` equivale ao nosso **faça**, ele entra direto no escopo e depois avalia se entra novamente no `while` (**enquanto**). Caso seja verdade ele repete o escopo do `do`.

Você pode optar por usar essa estrutura sempre que tiver certeza que um determinado trecho de código deverá ser executado pelo menos uma vez. Na Figura 6.4 temos o fluxograma da estrutura **faça-enquanto** da tabuada e no Código 6.2 temos o sua implementação.

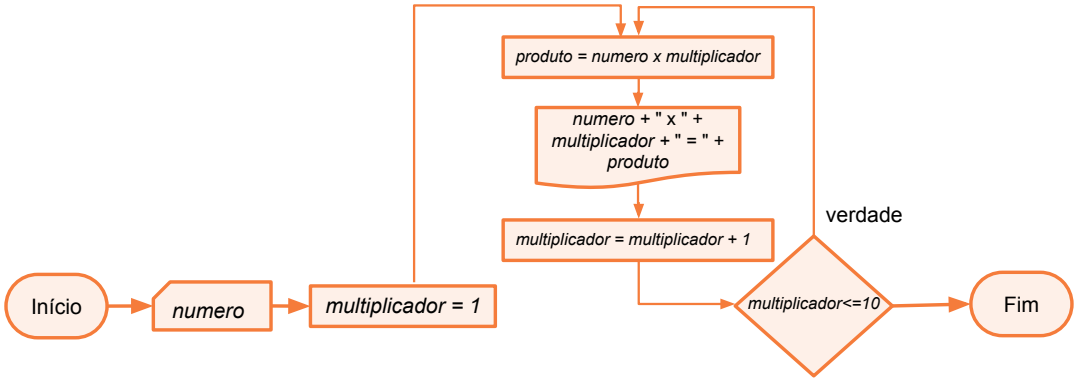

Figura 6.4: Fluxograma da estrutura do **faça-enquanto** do exemplo da tabuada.

■ **Código 6.2** Tabuada com `do-while`[2] (Fluxograma da Figura 6.4).

```
1  var numero = Number(prompt("Informe qual tabuada de multiplicação você
       quer (entre 1 e 10)"));
2  var multiplicador = 1;
3  do {
4    produto = numero * multiplicador;
5    console.log(`${numero} x ${multiplicador} = ${produto}`);
6    multiplicador++;
7  }while(multiplicador<=10);
```

■

No Código 6.4, observe que como não existe nenhuma condição após o `do` o trecho das linhas 4 até 6 é executado. Aí é que a *condição de parada* é verificada, se for verdade o fluxo volta para linha 4, senão a repetição para. Ou seja, pelo menos uma vez o escopo da repetição `do-while` será executado.

6.4 Estrutura Desde que-Até

Essa estrutura é bem interessante, e simplifica muito o trabalho quando temos que incrementar uma variável, como é o caso da tabuada. No fluxograma da Figura 6.5 temos o seu funcionamento.

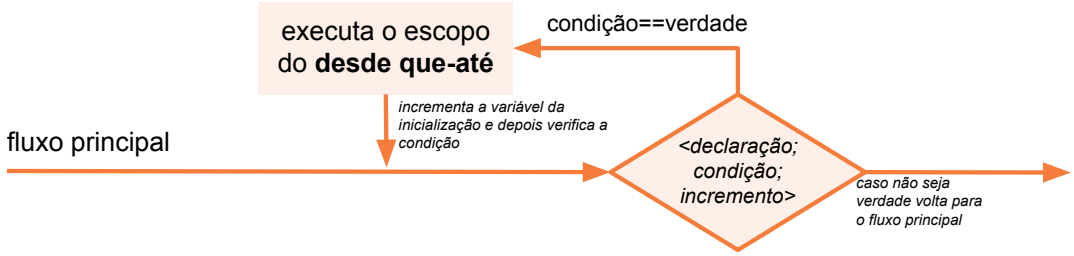

Figura 6.5: Fluxograma da estrutura do **desde que-faça**.

A estrutura **desde que-até** é constituída por três partes: (1) declaração, (2) condição e (3) incremento[3]. Em (1) você irá declarar, e inicializar, a(s) variável(eis) que será(ão) incrementada(s), ou decrementada(s), ao final do escopo; no (2) você irá definir qual condição deve ser satisfeita para entrada no escopo; e em (3) você adiciona o(s) incremento(s) (ou decremento(s)) na(s) variável(eis) declarada(s).

Agora vamos ver o funcionamento dessas três partes na repetição do escopo. Na primeira vez que a estrutura começa sua execução irá ocorrer a inicialização da(s) variável(eis) (1) e o teste da condição (2). Se a condição for satisfeita, então depois de cada laço (escopo), antes de iniciar o escopo novamente, (3) o(s) incremento(s), ou decremento(s), são realizados. O **desde que** para **até** que a condição (2) não seja satisfeita. Logo, já dá para perceber que pode ser que o escopo do **desde que-até** nunca seja executado, caso a condição já falhe após a inicialização.

Essa estrutura no *javascript* se chama `for`:

```
1  for (<declaração(ões)>;<condição>;<incremento(s) ou decremento(s)>) {
2      <instruções>
3  }
```

[2] <https://replit.com/@DaniloBorges/EstruturaDoWhile-Ex1>

[3] Pode ser colocado também um decremento, ou os dois: uma variável com incremento e outra com decremento.

No `for` temos as três partes da estrutura, cada uma delas separadas por ponto e vírgula. Da mesma forma que o `while`, esse comando só será executado se a condição for verdadeira. A declaração (com inicialização) ocorre uma única vez, e o(s) incremento(s) só acontece no final do escopo; é como se existisse o comando de incremento, ou decremento, no final do laço. A entrada no laço só será feita enquanto a condição for verdadeira.

Vamos lá, ficará bem mais claro ao praticarmos. Na Figura 6.6, temos o fluxograma da tabuada utilizando essa estrutura. Olha como ficou mais simples...

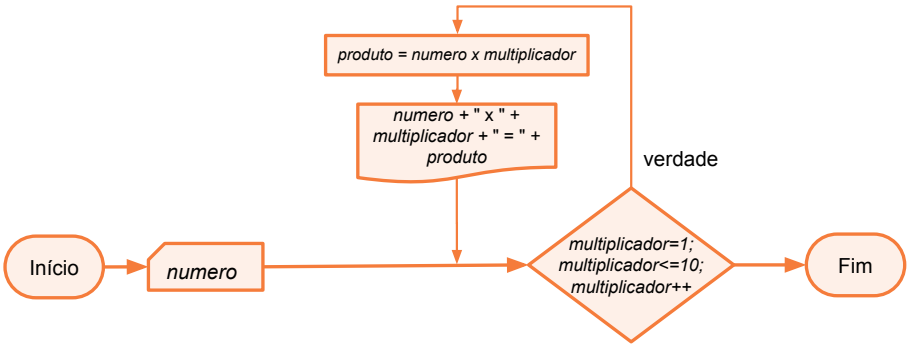

Figura 6.6: Fluxograma da estrutura do **desde que-até** do exemplo da tabuada.

No Código 6.3 temos a implementação da tabuada usando essa estrutura. Observe que não precisamos mais declarar o `multiplicador` antes da repetição, colocamos isso na área de declaração (`var multiplicador=1;`); a condição ficou no meio (`multiplicador<=10;`); e no caso dessa aplicação teremos o incremento da variável `multiplicador` que colocávamos no final do escopo (no `while` e `do-while`) agora ficará na estrutura na terceira parte do `for` (`multiplicador++`).

■ **Código 6.3** Tabuada com `for`[4] (Fluxograma da Figura 6.6).

```
1 var numero = Number(prompt("Informe qual tabuada de multiplicação você
      quer (entre 1 e 10)"));
2 for (var multiplicador=1;multiplicador<=10;multiplicador++) {
3   produto = numero * multiplicador;
4   console.log(`${numero} x ${multiplicador} = ${produto}`);
5 }
```

■

Pronto. Mostrei as estruturas básicas de repetição que poderemos usar para solucionar vários problemas. Lembre-se, sempre que for utilizar um desses comandos, de definir o *critério de parada*. Uma outra alternativa para parar o programa, dentro de uma estrutura de repetição, é usar o comando `break`.

6.5 Parar a Repetição

Como havia dito a repetição para quando o *critério de parada* estiver presente, ou seja, quando a condição da repetição for `false`. Porém podemos parar o fluxo da repetição com o comando `break`, similar ao `break` que temos em cada `case` do `switch`. Ao executar esse comando o fluxo para imediatamente.

Para ilustrar esse comportamento farei o seguinte exemplo: o usuário poderá ver quantas tabuadas quiser, porém, se ele colocar um número que estiver fora dos números entre 1 e

[4] <https://replit.com/@DaniloBorges/EstruturaFor-Ex1>

10 o programa para. Você já deve ter percebido que vamos trabalhar com uma estrutura condicional para verificar se o número está entre 1 e 10. E com dois laços de repetição: (1) para repetir a impressão da tabuada e (2) para imprimir a tabuada em si. Optei por utilizar em (1) um `do-while` e no (2) um `for`. Vejamos o Código 6.4.

■ **Código 6.4** Tabuada com `do` Infinito[5]. A palavra infinito se refere a uma repetição na qual a condição é sempre verdadeira, o critério de parada depende totalmente do `break`.

```
1  do{
2    var numero = Number(prompt("Informe qual tabuada de multiplicação voc
         ê quer (entre 1 e 10)"));
3    if(numero<1 || numero>10){
4      console.log(`Você digitou um número inválido: ${numero}`);
5      console.log("Fim do Programa");
6      break;
7    }
8    for (var multiplicador=1;multiplicador<=10;multiplicador++) {
9      produto = numero * multiplicador;
10     console.log(`${numero} x ${multiplicador} = ${produto}`);
11   }
12 } while (true);
```

A estrutura `do` compreende o escopo da linha 2 até a linha 11. E nossa condição, linha 12, será sempre satisfeita, pois, coloquei um `true` dentro dela. Somente se o usuário colocar um número inválido é que essa repetição irá parar, ou seja, se a condição do `if` da linha 3 for satisfeita; será impresso a informação de número inválido, sinalizado o fim do programa, e o `break` irá parar a estrutura de repetição. A repetição feita pelo `for` será responsável por imprimir a tabuada do número informado. ■

Coloquei um exemplo que mostra o uso do `break` bem simples, mas saiba que você pode utilizar sempre que quiser. Na maioria dos usos, dentro de uma repetição, o colocamos dentro de um condicional. Então, se quiser parar a repetição de outra forma, sem ser com a condição, é só utilizá-lo.

6.6 Estudos de Caso

Nesta seção coloquei alguns problemas práticos e os resolvo utilizando estrutras de repetição com *javascript*. Colocarei também na definição do problema qual a estrutura mais indicada desenvolver a solução.

Problema 6.6.1 — While ou Do-While ou For. Escreva um programa que encontre o quinto número maior que 1000, cuja divisão por 11 tenha resto 5.

O Problema 6.6.1 pode ser resolvido utilizando qualquer um dos algoritmos de repetição que vimos. O que devemos prestar atenção é no *critério de parada*. Qual seria o *critério de parada* desse problema? Resposta correta: ter encontrado o quinto número maior que 1000, cuja divisão por 11 tenha resto 5. Então vamos utilizar uma variável para armazenar a quantidade de números que encontrarmos maiores que 1000 cuja divisão por 11 tenha resto 5, e caso tenhamos encontrado cinco números a gente para. Certo?

Nos Códigos 6.5 ao 6.7, chamaremos essa variável de `quantidade_do_criterio` que começa inicialmente com `0`. E à medida que vamos encontrando os números que queremos vai sendo incrementada.

Você pode colocar então como critério de parada `quantidade_do_criterio<5`, e dentro da repetição caso encontre `quantidade_do_criterio==5` sinalizar que encontrou o tal do

[5] <https://replit.com/@DaniloBorges/EstruturaBreak-Ex1>

quinto número.

Coloquei em cada um dos códigos uma estrutura de repetição diferente, para que você consiga observar as semelhanças e o porquê de cada uma funcionar como esperado.

■ **Código 6.5** Solução do Problema 6.6.1 com `while`[6].

```
1  var numero = 1000;
2  var quantidade_do_criterio = 0;
3  while (quantidade_do_criterio<5){
4    numero++;
5    if(numero%11==5){
6      quantidade_do_criterio++;
7    }
8    if(quantidade_do_criterio==5){
9      console.log("O quinto número maior que 1000, cuja divisão por 11
           tenha resto 5 é",numero);
10   }
11 }
```

■

■ **Código 6.6** Solução do Problema 6.6.1 com `do-while`[7].

```
1  var numero = 1000;
2  var quantidade_do_criterio = 0;
3  do {
4    numero++;
5    if(numero%11==5){
6      quantidade_do_criterio++;
7    }
8    if(quantidade_do_criterio==5){
9      console.log("O quinto número maior que 1000, cuja divisão por 11
           tenha resto 5 é",numero);
10     break;
11   }
12 } while (true);
```

■

■ **Código 6.7** Solução do Problema 6.6.1 com `for`[8].

```
1  var quantidade_do_criterio = 0;
2  for (var numero=1000;quantidade_do_criterio<5;numero++){
3    if(numero%11==5){
4      quantidade_do_criterio++;
5    }
6    if(quantidade_do_criterio==5){
7      console.log("O quinto número maior que 1000, cuja divisão por 11
           tenha resto 5 é",numero);
8    }
9  }
```

■

Problema 6.6.2 — While ou Do-While ou For. Foi feita uma pesquisa entre um grupo de estudantes e coletados os dados de altura e sexo (0=masculino, 1=feminino). Faça um programa que leia 10 dados diferentes e informe:

[6] <https://replit.com/@DaniloBorges/EstudoCaso-Ex1-ComWhile>
[7] <https://replit.com/@DaniloBorges/EstudoCaso-Ex1-ComDoWhile>
[8] <https://replit.com/@DaniloBorges/EstudoCaso-Ex1-ComFor>

1. a maior e a menor altura encontradas;
2. a média de altura dos estudantes de sexo feminino;
3. a média total de altura dos estudantes;
4. o percentual de estudantes do sexo masculino;

Calma, parece muita coisa nesse Problema 6.6.2, mas é bem simples de resolver. Primeiro temos que ver como nossa repetição irá agir, você tem alguma sugestão? Vamos agir na quantidade de dados! Caso cheguemos a quantidade de dados (variável `quantidade_dados`) esperada a gente para (`quantidade_dados==10`). Em cada laço da repetição devemos capturar o sexo do estudante (variável `sexo`), e sua altura (variável `altura`). E devemos também criar variáveis para armazenar: maior altura (variável `maior_altura`), menor altura (variável `menor_altura`), quantidade de estudantes do sexo masculino (variável `quantidade_masculino`), quantidade de estudantes do sexo feminino (variável `quantidade_feminino`). Com essas variáveis conseguiremos resolver o problema. Embora seja possível fazer este algoritmo utilizando qualquer estrutura de repetição mostrarei somente a que utiliza o `while`, Código 6.8.

Código 6.8 Solução do Problema 6.6.2 com `while`[9].

```
1  var soma_altura_feminino = 0, soma_altura_masculino = 0;
2  var quantidade_feminino = 0, quantidade_masculino = 0;
3  var altura,sexo;
4  var maior_altura = -Infinity, menor_altura = Infinity;
5  var quantidade_dados = 1;
6  while (quantidade_dados<=10){
7    altura = Number(prompt(`Informe a altura do estudante ${
         quantidade_dados}:`));
8    sexo = Number(prompt(`Informe o sexo do estudante ${quantidade_dados}
         (0 para masculino e 1 para feminino):`));
9    if(sexo==0){ //sexo masculino
10     quantidade_masculino++;
11     soma_altura_masculino += altura;
12   }else if(sexo==1){
13     quantidade_feminino++;
14     soma_altura_feminino += altura;
15   }else{
16     console.log("Sexo inválido, faça novamente.")
17   }
18   if ((sexo==0) || (sexo==1)){
19     if(altura > maior_altura){
20       maior_altura = altura;
21     }
22     if(altura < menor_altura){
23       menor_altura = altura;
24     }
25     quantidade_dados++;
26   }
27 }
28 console.log("Resultado:");
29 console.log("Maior Altura: ",maior_altura);
30 console.log("Menor Altura: ",menor_altura);
31 console.log("Média de altura do sexo feminino",soma_altura_feminino/
       quantidade_feminino);
32 console.log("Média de altura dos estudantes",(soma_altura_feminino+
       soma_altura_masculino)/10);
33 console.log("Sexo masculino corresponde a",((quantidade_masculino)/(
       quantidade_feminino+quantidade_masculino))*100,"%");
```

[9] <https://replit.com/@DaniloBorges/EstudoCaso-Ex2-ComWhile>

■

No Código 6.8, você pode observar que inicialmente declarei todas essas variáveis citadas, algumas já com iniciazação (linhas 1, 2, 4 e 5). Um caso especial está nas variáveis `maior_altura` e `menor_altura` que foram inicializadas com `-Infinity` e `Infinity`, repectivamente. Mas porque coloquei isso aqui? Você acertou quando pensou que não existe esse número. Essa palavra reservada é uma variável que possui o maior valor possível que o programa pode possuir. Logo, estou inicializando essas variáveis com o menor e maior valor para que na primeira comparação (linhas 19 e 22) elas sejam atualizadas. Uma atenção especial na linha 18, só irei atualizar o contador de dados, e maior altura se foi inserido o código do sexo de forma correta, caso contrário informo que o usuário deve tentar novamente (linha 16). Da linha 28 até a linha 33 temos a exposição dos resultados do que foi requisitado no problema.

Para finalizar nossos estudos de caso, vejamos o Problema 6.6.3.

Problema 6.6.3 — While ou Do-While ou For. Escreva um programa que lê um valor n inteiro e positivo e que calcula a seguinte soma:

$$\frac{1}{1}+\frac{2}{3}+\frac{3}{5}+\frac{4}{7}+\cdots+\frac{n-1}{2(n-1)-1}+\frac{n}{(2n-1)}$$

Nesse último problema a estrutura mais indicada de repetição para resolvê-la é o `for`. Devido ao fato de que a soma está sempre incrementando o numerador e o denominador, então o `for` é perfeito para isso. Precisamos somente definir a regra da formação do numerador e do denominador em cada laço. Na própria definição do problema já nos foi dado o formato da atualização do numerador (variável `numerador`) e denominador (variável `denominador`): o numerador soma de 1 em 1, e o denominador depende do numerador considerado no laço. Em cada laço iremos adicionando o valor do cálculo da fração à soma (variável `soma`). Confira tudo isso no Código 6.9.

■ **Código 6.9** Solução do Problema 6.6.3 com `for`[10].

```
1 var n = parseInt(prompt("Informe o número n, inteiro e positivo:"));
2 var soma = 0;
3 for (var i=1,numerador=1,denominador=1;i<=n;i++,numerador++,denominador
      =2*numerador-1){
4   soma += numerador/denominador;
5 }
6 console.log("Soma =",soma);
```

■

Perceberam algo de diferente no `for` do Código 6.9? Na parte da declaração coloquei a variável que controla o laço, `i`, e adicionei as variáveis `numerador` e `denominador` (separadas por vírgula). E na seção de incremento, ou decremento, coloquei a atualização de todas essas três variáveis. Perceba que o incremento não se aplica somente aos operadores `++` e `--`, você pode colocar outra forma de incremento como a que coloquei neste exemplo, `denominador=2*numerador-1`.

6.7 Resumo

Vimos três modos de repetir um fluxo em um algoritmo e utilizando *javascript*, um resumo dos comandos podem ser observados na Tabela 6.1. Com a prática você irá percebendo

[10] <https://replit.com/@DaniloBorges/EstudoCaso-Ex3-ComFor>

qual estrutura utilizar, em alguns casos uma estrutura será melhor que outra e em outros casos podemos usar qualquer uma das estruturas. Agora é a sua vez, pratique bastante!

Comando	Descrição
`while(<condição>) {<instruções>}`	Enquanto a condição for satisfeita (verdade) as instruções serão executadas.
`do{<instruções>} while(<condição>);`	Nessa estrutura as instruções serão executadas pelo menos uma vez, após avaliada a condição o bloco do `do` repete caso a condição seja verdade.
`for(<declaração>;<condição>;<incremento>) {<instruções>}`	Declaração: será a inicialização das variáveis que serão incrementadas ou decrementadas no final do laço; Condição: é a avaliação que será feita para que o laço possa ser executado; e Incremento: será inserida a operação de incremento, ou decremento, da variável declarada.
`break`	Para o fluxo da repetição.

Tabela 6.1: Resumo dos comandos das estruturas de repetição em *javascript*.

6.8 Exercícios

Exercício 6.1 Faça um programa que imprime a tabuada da soma, dado um número n. Exemplo, caso $n = 4$, seu programa deve imprimir:

```
4 + 0 = 4
4 + 1 = 5
4 + 2 = 6
4 + 3 = 7
4 + 4 = 8
4 + 5 = 9
4 + 6 = 10
4 + 7 = 11
4 + 8 = 12
4 + 9 = 13
4 + 10 = 14
```

■

Exercício 6.2 Faça um programa que imprime a tabuada da divisão, dado um número n. Exemplo, caso $n = 3$, seu programa deve imprimir:

```
3 / 3 = 1
6 / 3 = 2
9 / 3 = 3
12 / 3 = 4
```

```
15 / 3 = 5
18 / 3 = 6
21 / 3 = 7
24 / 3 = 8
27 / 3 = 9
30 / 3 = 10
```

Exercício 6.3 Faça um programa que calcula a seguinte soma:

$$\frac{1}{1}+\frac{3}{2}+\frac{5}{3}+\frac{7}{4}+\cdots+\frac{99}{50}$$

Exercício 6.4 Faça um programa que escreve os números ímpares entre 1 e 100.

Exercício 6.5 Faça um programa lê cinco números e exibe o maior e o menor número.

Exercício 6.6 Faça um programa que leia um número, n, e imprima se ele é primo ou não. (um número primo tem apenas 2 divisores: 1 e ele mesmo! O número 1 não é primo!!!)

Exercício 6.7 Escreva um programa que recebe um número entre 1 e 1000. Entre todos os números de 1 até o número informado você deve exibir aqueles que divididos por 11 dão resto 5.

Exercício 6.8 Faça um programa que leia um número, n, (informado pelo usuário) e mostre na tela os n primeiros números primos. Exemplo.: se for inserido o número 6, deverá ser impresso os números: 2, 3, 5, 7, 11 e 13.

Exercício 6.9 Escreva um programa que calcule e mostre a média aritmética dos números lidos entre 13 e 73.

Exercício 6.10 Escreva um programa que gera e escreve os 5 primeiros números perfeitos. Um número perfeito é aquele que é igual a soma dos seus divisores. (Ex.: $6 = 1+2+3$; $28 = 1+2+4+7+14$).

Exercício 6.11 Faça um algoritmo que calcule o fatorial de um número. (Ex.: $5! = 5\times4\times3\times2\times1 = 120$).

Exercício 6.12 Faça um programa que leia uma quantidade não determinada de números positivos. Calcule a quantidade de números pares e ímpares, a média de valores pares e a média geral dos números lidos. O número que encerrará a leitura será um número menor ou igual a zero.

Exercício 6.13 Faça um programa que leia uma quantidade desconhecida de números e conte quantos deles estão nos seguintes intervalos: [0-25], [26-50], [51-75] e [76-100]. Você deve parar de ler os números caso o número informado esteja fora do intervalo (0 a 100). ■

Exercício 6.14 Faça um programa que lê um valor n inteiro e positivo e que calcula e escreve o valor de E.

$$E = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{(n-1)!} + \frac{1}{n!}$$

Observação: $0! = 1$. ■