

JavaScript

Módulo Básico

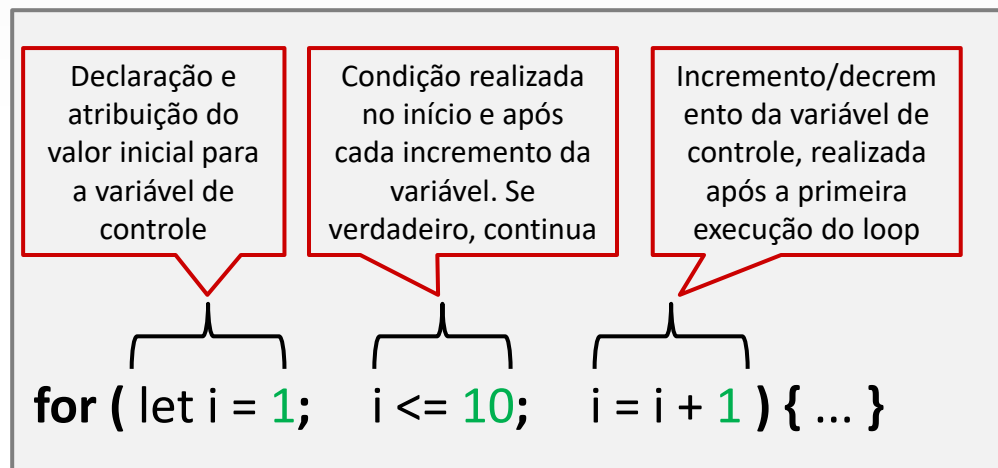




REPETIÇÕES

Repetição

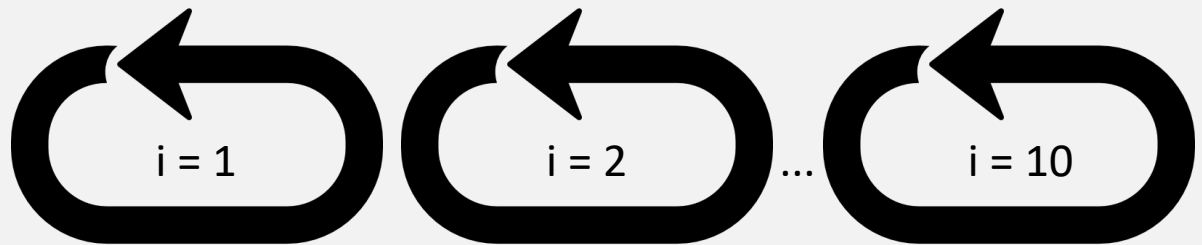
- **Repetição com variável de controle: laço *for***
 - A **sintaxe do comando *for*** é composta de três instruções, que definem:
 - a) o valor inicial da variável de controle
 - b) a condição que determina se a repetição deve ou não continuar
 - c) o incremento ou decremento da variável de controle



Repetição

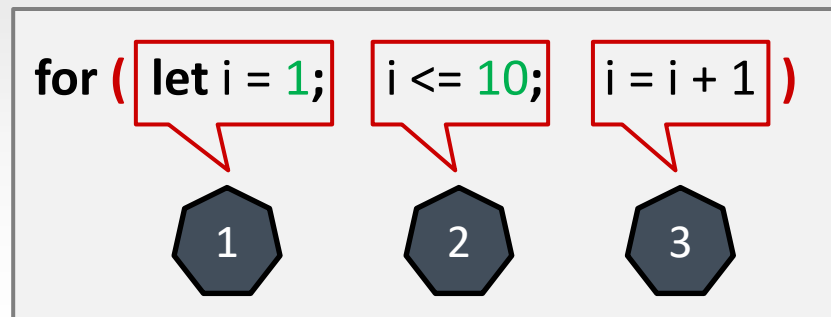
- **Repetição com variável de controle: laço *for***
 - O funcionamento de um **loop for**, que contém inclusive uma condição que avalia o valor da variável *i*, retornando **verdadeiro** nas voltas de número par

```
for (let i = 1; i <= 10; i++) {  
  comando1  
  comando2  
  comando3  
  if (i % 2 == 0) {  
    comando4  
  }  
}
```



Repetição

- **Repetição com variável de controle: laço *for***
 - *Observe a numeração inserida nas instruções que compõem o **for***



- *A sequência de execução das instruções é a seguinte: 1 e 2 (executa), 3 e 2 (executa), 3 e 2 (executa), ... (segue 3 e 2 até a condição ficar falsa)*

Repetição

- **Repetição com variável de controle: laço *for***
 - Para **facilitar a compreensão** do que ocorre **passo a passo** a **execução desse comando em um programa**

```
let numeros = ""  
  
for (let i = 1; i < 4; i = i + 1) {  
    numeros = numeros + i  
}  
resp.innerText = numeros
```

Repetição

- **Repetição com variável de controle: laço *for***
 - Após a variável **numeros** ser declarada e inicializada, as seguintes operações são realizadas pelo laço *for*:
 1. A variável **i** é declarada e recebe o valor **1**
 2. O teste condicional é realizado (**i** < **4**) e retorna **verdadeiro**
 3. O comando do laço é executado: o valor de **i** (**1**) é atribuído à variável **numeros**, que recebe ela mesmo + **i**, ou seja, **numeros** = "**1**"
 4. Volta-se ao comando *for* e a terceira instrução é executada: **i** = **i** + **1**. Logo, **i** = **2**
 5. O teste condicional é novamente realizado (**i** < **4**) e continua **verdadeiro**

Repetição

- **Repetição com variável de controle: laço *for***
 - Após a variável **numeros** ser declarada e inicializada, as seguintes operações são realizadas pelo laço *for*:
 6. Assim o comando do laço é executado, o valor de *i* (2) é atribuído à variável **numeros**, que recebe ela mesmo + 1: **numeros = "12"**
 7. A terceira instrução do comando *for* é novamente executada, *i* = *i* + 1. Logo *i* = 3
 8. O teste condicional é realizado (*i* < 4) e prossegue **verdadeiro**
 9. Mais uma vez, *i* (3) é atribuído a **numeros**, junto ao conteúdo anterior dessa variável. Agora, **numeros = "123"**
 10. Volta-se à execução da terceira instrução do *for*: *i* = *i* + 1. Logo, *i* = 4

Repetição

- **Repetição com variável de controle: laço *for***
 - Após a variável **numeros** ser declarada e inicializada, as seguintes operações são realizadas pelo laço **for**:
 11. O teste condicional é realizado ($i < 4$) e retorna **falso**. O laço é finalizado e se executa o comando após o **for**



- *Repetição com variável de controle: laço **for***
 - *Crie uma pasta **css**, **img** e **js***
 - *Crie um arquivo intitulado de **estilos.css** com o conteúdo abaixo, e salve dentro da pasta **css**:*

```
img.normal { float: left; height: 300px; width: 300px; }  
img.alta { float: left; height: 420px; width: 300px; }  
img.exerc { float: left; height: 200px; width: 200px; }  
h1 { border-bottom-style: inset; }  
pre { font-size: 1.2em; }
```

- **Repetição com variável de controle: laço *for***
 - Definimos **duas regras** para as imagens: **alta** e **normal**
 - Para indicar que uma **imagem** deve ser **estilizada** seguindo uma dessas regras, deve-se utilizar **class="regra"** na **tag** correspondente
 - Caso fôssemos hospedar a **página** em um **provedor de conteúdo**, o **recomendado** seria **trabalhar o tamanho da imagem** para ela **possuir os mesmos valores indicados no estilo**
 - **Designar o tamanho da imagem no CSS é importante** a fim de que o **navegador reserve o espaço adequado** para a **imagem enquanto ela é carregada**

Repetição

- *Repetição com variável de controle: laço **for***
 - *A acrescentamos também a definição de um tamanho de fonte para a tag **pre**, que será utilizada em frequência para exibir a resposta nos programas*



Repetição

- *Repetição com variável de controle: laço **for***
 - *Exemplo (01):*
 - *Ler um número e apresentar a tabuada desse número*

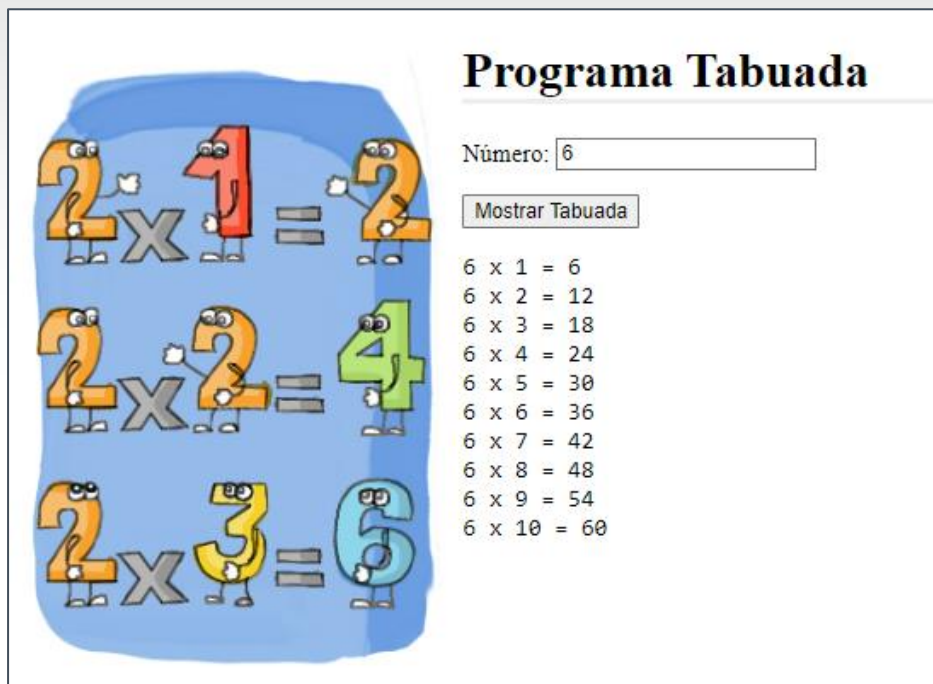


Figura 1 – Renderização do HTML do Exemplo (01)

Repetição

- *Repetição com variável de controle: laço **for***
 - *Exemplo (01):*

```
html > exemplo_1.html > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="../css/estilos.css">
8      <title>Exemplo 01 | bkBank Academy</title>
9  </head>
10 <body>
11     
12     <h1> Programa Tabuada </h1>
13     <form>
14         <p> Número:
15             <input type="number" id="inNumero" required>
16         </p>
17         <input type="submit" value="Mostrar Tabuada">
18     </form>
19     <pre></pre>
20     <script src="../js/exemplo_1.js"></script>
21 </body>
22 </html>
```

Figura 2 – HTML do Exemplo (01)

Repetição


- *Repetição com variável de controle: laço **for***
 - *Exemplo (01):*

```
js > JS exemplo_1.js > ...
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp = document.querySelector("pre")
3
4  frm.addEventListener("submit", (e) => {    // "escuta" evento submit do form
5      e.preventDefault()                    // evita envio do form
6
7      const numero = Number(frm.inNumero.value) // obtém número informado
8
9      let resposta = "" // variável do tipo String, para concatenar a resposta
10
11     // cria um laço de repetição (i começa em 1 e é incrementado até 10)
12     for (let i = 1; i <= 10; i++) {
13         // a variável resposta vai acumulando os novos conteúdos (nos 2 formatos)
14         resposta = resposta + numero + " x " + i + " = " + (numero * i) + "\n"
15         // resposta = `${resposta} ${numero} x ${i} = ${numero * i}\n`
16     }
17
18     // o conteúdo da tag pre é alterado para exibir a tabuada do número
19     resp.innerText = resposta
20 }
```

Figura 3 – JS do Exemplo (01)

Repetição

- **Repetição com variável de controle: laço *for***
 - **Exemplo (02):**
 - **Obter o valor digitado pelo usuário e apresentar todos os números existentes entre o número informado e 1, de forma decrescente**



Programa Números Decrescentes

Número:

Entre 15 e 1: 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

Figura 4 – Renderização do HTML do Exemplo (02)

Repetição

- *Repetição com variável de controle: laço **for***
 - *Exemplo (02):*

```
html > exemplo_2.html > html > body > img.alta
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="../css/estilos.css">
8   <title>Exemplo 02 | bkBank Academy</title>
9 </head>
10 <body>
11   
12   <h1> Programa Números Decrescentes </h1>
13   <form>
14     <p> Número:
15       <input type="number" id="inNumero" required>
16     </p>
17     <input type="submit" value="Decrescer até 1">
18   </form>
19   <h3></h3>
20   <script src="../js/exemplo_2.js"></script>
21 </body>
22 </html>
```

Figura 5 – HTML do Exemplo (02)

Repetição

- *Repetição com variável de controle: laço **for***
 - *Exemplo (02):*

```
js > JS exemplo_2.js > frm.addEventListener("submit") callback
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp = document.querySelector("h3")
3
4  frm.addEventListener("submit", (e) => {
5    e.preventDefault() // evita envio do form
6
7    const numero = Number(frm.inNumero.value) // obtém número informado
8
9    let resposta = `Entre ${numero} e 1: ` // String para montar a resposta
10
11    for (let i = numero; i > 0; i--) { // cria um for decrescente
12      if (i == 1){
13        resposta = resposta + i + "." // ou resposta = `${resposta}${i}.`
14      } else {
15        resposta = resposta + i + ", " // ou resposta = `${resposta}${i},`
16      }
17    }
18    resp.innerText = resposta // exibe a resposta
19  })
```

Figura 6 – JS do Exemplo (02)

Repetição

- **Repetição com teste no início: laços *while***
 - *Um laço de repetição também pode ser criado com o comando **while**, que realiza um teste condicional logo no seu início, para verificar se os comandos do laço serão ou não executado*
 - **Sintaxe:**

```
while (condição) {  
    comandos  
}
```

Repetição

- **Repetição com teste no início: laços *while***
 - No exemplo (01) – Números Decrescentes, poderíamos substituir o comando *for* pelo comando *while*

```
let i = numero           // declara e inicializa a variável i
while (i > 0) {           // enquanto i maior que 0
  resposta = resposta + i + ", " // acumula valores de i
  i--                     // decrementa o i (idem a i = i -1)
}
```

Como o teste é realizado no início, é possível que os comandos do *while* não sejam executados.

Caso o usuário digite o número 0 (zero), a condição já é falsa na primeira verificação e o programa não entra no laço de repetição.

Repetição

- **Repetição com teste no final: laços `do.. while`**
 - Outra alternativa de criar laços de repetição é com a utilização do comando `do.. while`
 - **Sintaxe:**

```
do {  
    comandos  
} while (condição)
```

Com o comando `while`, a condição é verificada no início; enquanto, com o comando `do..while`, a condição é verificada no final.

Com o `do..while`, fica garantido que uma vez, no mínimo, os comandos que pertencem ao laço serão executados.

Repetição

- **Repetição com teste no final: laços `do.. while`**
 - **Exemplo (03):**
 - O programa utiliza o método `prompt()` e o `alert()` para ilustrar o funcionamento do laço criado com o comando `do.. while` na entrada de dados. Após a validação, o programa exibe todos os números pares entre **1** e o número informado pelo usuário

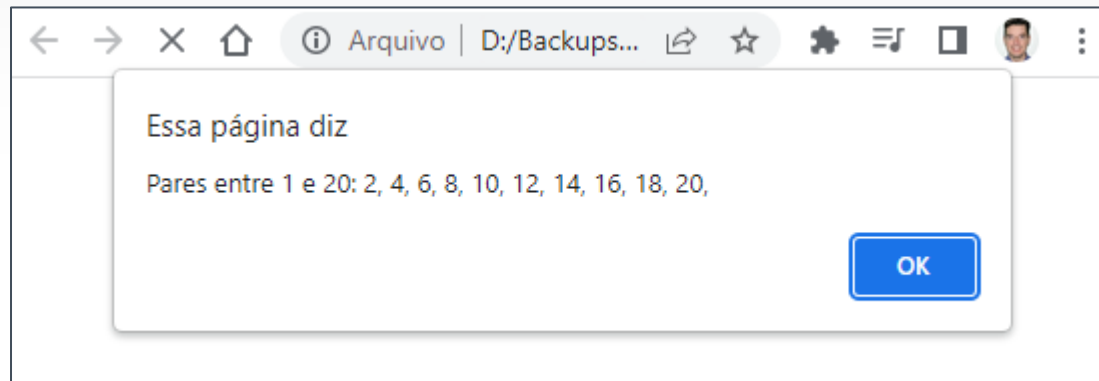


Figura 7 – Lista dos pares entre 1 e 20 (número informado pelo usuário)

Repetição

- *Repetição com teste no final: laços **do.. while***
 - *Exemplo (03):*

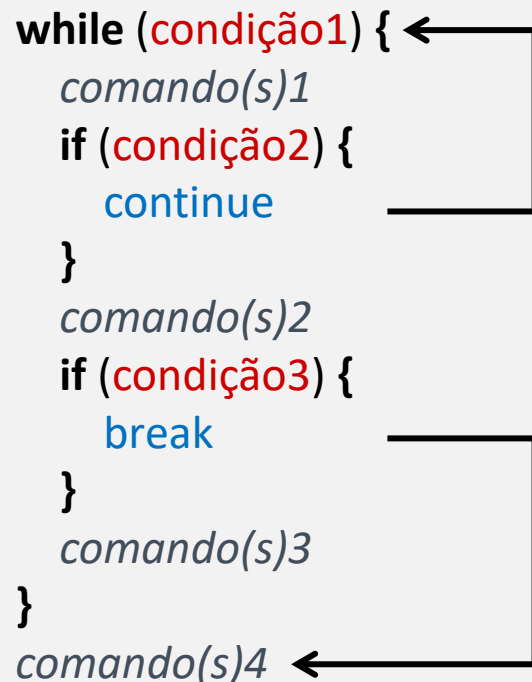
```
10  <script>
11      let num          // declara variável num com let, pois ela pode ser alterada
12                          // e será acessada fora do bloco do..while
13
14      do {              // cria um laço de repetição (faça...)
15          num = Number(prompt("Número: ")) // lê um número
16
17          if (num == 0 || isNaN(num)) {    // se num é inválido
18              alert("Digite um número válido...")
19          }
20      } while (num == 0 || isNaN(num))      // ... enquanto num inválido
21
22      let pares = `Pares entre 1 e ${num}: ` // string que irá conter a resposta
23
24      for (let i = 2; i <= num; i = i + 2) {
25          pares = pares + i + ", "
26      }
27      alert(pares) // exibe lista de números pares
28  </script>
```

Figura 8 – JS do Exemplo (03)

Repetição

- *Interrupções nos laços (**break** e **continue**)*
 - O **break** promove a saída do laço de repetição, enquanto o **continue** retorna ao início do laço (auxiliam no controle de execução dos comandos do loop)

```
while (condição1) {  
    comando(s)1  
    if (condição2) {  
        continue  
    }  
    comando(s)2  
    if (condição3) {  
        break  
    }  
    comando(s)3  
}  
comando(s)4
```



O comando continue faz com que a condição seja novamente testada. Se verdadeiro, continua a execução.

O comando break faz o programa sair do laço de repetição. Se houver comandos após o laço, eles serão executados.

- **Interrupções nos laços (*break* e *continue*)**
 - **Exemplo (04):**
 - **Utiliza os métodos *prompt()* e *alert()* para realizar a leitura de um número e, caso o número for *par* ele exibe o dobro do número e se for *ímpar*, o triplo. A leitura continua até que o usuário informe *0* (ou algum valor inválido)**

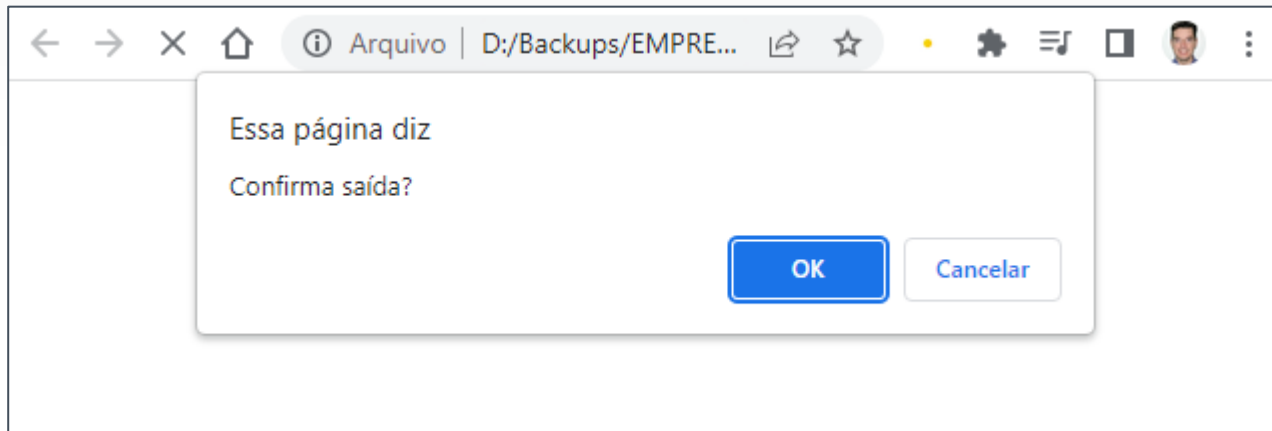


Figura 9 – Retorno do método *confirm()* definirá a execução do *break* ou *continue*

Repetição

- *Interrupções nos laços (**break** e **continue**)*
 - *Exemplo (04):*

```
10  <script>
11    alert("Digite 0 para sair")
12
13    do {
14      const num = Number(prompt("Número: ")) // lê o número
15
16      if (num == 0 || isNaN(num)) {           // se num = 0 ou inválido
17        const sair = confirm("Confirma saída?")
18
19        if (sair) {
20          break                               // sai da repetição
21        } else {
22          continue                           // volta ao início do laço
23        }
24      }
25
26      if (num % 2 == 0) {                     // se par
27        alert(`O dobro de ${num} é: ${num * 2}`) // mostra o dobro
28      } else {                               // senão,
29        alert(`O triplo de ${num} é: ${num * 3}`) // mostra o triplo
30      }
31    } while (true) // enquanto verdadeiro (só sai do laço, pelo break)
32    alert("Bye, bye...")
33  </script>
```

Figura 10 – JS do Exemplo (04)

- **Contadores e Acumuladores**
 - *O uso de **contadores** e **acumuladores** em um programa permite a exibição de contagens e totalizações*
 - *Essas operações são realizadas sobre os dados manipulados pelo programa*
 - *Os **contadores** ou **acumuladores** possuem duas características principais:*
 - *A variável contadora ou acumuladora deve receber uma atribuição inicial (**geralmente zero**)*
 - *A variável contadora ou acumuladora deve receber ela mesma mais algum valor*

- **Contadores e Acumuladores**

- A **diferença** entre os **contadores** e os **acumuladores** é que o **contador** recebe **ele mesmo mais 1** (ou **algum valor constante**), enquanto o **acumulador** recebe **ele mesmo mais uma variável**
- Para fazer uma variável receber ela mesma mais algum valor podemos **repetir o nome da variável na atribuição** ou **utilizar o operador "+="**

```
let soma = 0           // deve ser declarada com let
soma = soma + preco    // em uma repetição, soma irá acumular o preco
soma += preco          // forma simplificada para soma = soma + preco
```

- **Contadores e Acumuladores**

- **Exemplo (05):**

- *Faz a leitura de contas que devem ser pagas por um usuário. As contas são exibidas e no final da listagem o número de contas (**contador**) e a soma dos valores (**acumulador**) são destacados*



Programa Contas do Mês

Descrição da Conta:

Valor a Pagar R\$:

CPFL - R\$: 208.98
SAERP - R\$: 71.89
Escola - R\$: 1340.00
Cartão de Crédito - R\$: 1780.25
Celular - R\$: 121.45

5 Conta(s) - Total R\$: 3522.57

Figura 11 – Renderização do HTML Exemplo (05)

- *Contadores e Acumuladores*
 - *Exemplo (05):*

```
html > exemplo_5.html > html > body
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="../css/estilos.css">
8      <title> Exemplo 05 | bkBank Academy </title>
9  </head>
10 <body>
11     
12     <h1> Programa Contas do Mês </h1>
13     <form>
14         <p>Descrição da Conta:
15             <input type="text" id="inDescricao" required>
16         </p>
17         <p>Valor a Pagar R$:
18             <input type="number" min="0" step="0.01" id="inValor" required>
19         </p>
20         <input type="submit" value="Registrar Conta">
21     </form>
22     <pre id="outResp1"></pre>
23     <pre id="outResp2"></pre>
24     <script src="../js/exemplo_5.js"></script>
25 </body>
26 </html>
```

Figura 12 – HTML Exemplo (05)

- *Contadores e Acumuladores*
 - *Exemplo (05):*

```
js > JS exemplo_5js > ...
1  const frm = document.querySelector("form")           // obtém elementos da página
2  const resp1 = document.querySelector("#outResp1")
3  const resp2 = document.querySelector("#outResp2")
4
5  let numContas = 0  // declara e inicializa contador...
6  let valTotal = 0  // ... e acumulador (variáveis globais)
7  let resposta = ""  // string com a resposta a ser exibida
8
9  frm.addEventListener("submit", (e) => {             // "escuta" evento submit do form
10     e.preventDefault()                             // evita envio do form
11
12     const descricao = frm.inDescricao.value         // obtém dados da conta
13     const valor = Number(frm.inValor.value)
14
15     numContas++                                     // adiciona valores ao contador e acumulador
16     valTotal = valTotal + valor
17
18     resposta = resposta + descricao + " - R$: " + valor.toFixed(2) + "\n"
19
20     resp1.innerHTML = `${resposta}-----`
21     resp2.innerHTML = `${numContas} Conta(s) - Total R$: ${valTotal.toFixed(2)}`
22
23     frm.inDescricao.value = ""                     // limpa campos do form
24     frm.inValor.value = ""
25     frm.inDescricao.focus()                         // posiciona no campo inDescricao do form
26 })
```

- **Contadores e Acumuladores**
 - *Observe que utilizamos **variáveis globais***
 - *Em JS, uma **variável global** continua disponível em memória enquanto a página está ativa*
 - *Para acumular os valores das contas e lista-los sempre que o usuário adicionar uma nova conta, é necessário o uso de **variáveis** com esse **escopo***
 - *A **variável numContas** atua como um **contador**, para apresentar a cada acréscimo de conta o número de contas inseridas pelo usuário*
 - *A **variável valTotal** é um **acumulador**, para somar o valor das contas inseridas no programa*

- **Contadores e Acumuladores**

- **Exemplo (06):**

- **Recebe um número e informa se ele é ou não primo. Um número primo é aquele que possui apenas 2 divisores: 1 e ele mesmo.**



Figura 14 – Renderização do HTML Exemplo (06)

- *Contadores e Acumuladores*
 - *Exemplo (06):*

```
html > 5 exemplo_6.html > html > body > script
1  <!DOCTYPE html>
2  <html Lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="../css/estilos.css">
8      <title> Exemplo 06 | bkBank Academy </title>
9  </head>
10 <body>
11     
12     <h1> Programa Números Primos </h1>
13     <form>
14         <p>Número:
15             <input type="number" id="inNumero" required>
16         </p>
17         <input type="submit" value="Verificar se é Primo">
18     </form>
19     <h3></h3>
20     <script src="../js/exemplo_6.js"></script>
21 </body>
22 </html>
```

Figura 15 – HTML Exemplo (06)

- **Contadores e Acumuladores**
 - **Exemplo (06):**

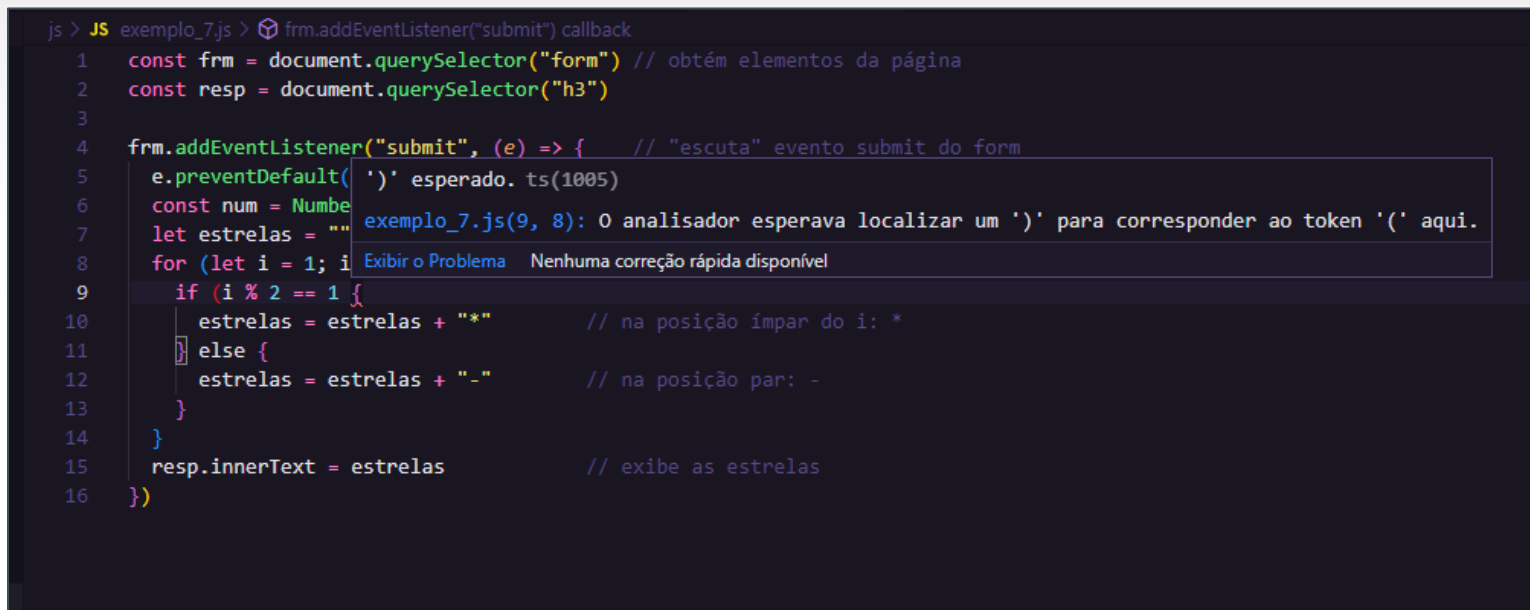
```
js > JS exemplo_6.js > frm.addEventListener("submit") callback
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp = document.querySelector("h3")
3
4  frm.addEventListener("submit", (e) => { // "escuta" evento submit do form
5      e.preventDefault() // evita envio do form
6
7      const num = Number(frm.inNumero.value) // obtém número informado
8      let temDivisor = 0 // declara e inicializa a variável tipo flag
9
10     for (let i = 2; i <= num / 2; i++) { // percorre os possíveis divisores do num
11         if (num % i == 0) { // se tem um divisor
12             temDivisor = 1 // muda o flag
13             break // sai da repetição
14         }
15     }
16
17     if (num > 1 && !temDivisor) { // se num > 1 e não possui divisor
18         resp.innerText = `${num} É primo`
19     } else {
20         resp.innerText = `${num} Não é primo`
21     }
22 })
```

Figura 16 – JS do Exemplo (06)

- ***Depurar Programas (Detectar Erros)***
 - *Em programação, depuração de programas é o nome dado ao processo de detectar e remover erros no código*
 - *Existem **dois tipos** principais de **erros** em um programa: **erros de sintaxe** e **erros de lógica***
 - *Erros de sintaxe impedem o programa de ser executado e referem-se à digitação incorreta de algum comando ou nome de variável*
 - *Em alguns casos, o próprio editor dá indicativos de que existe algum erro no código*

- *Depurar Programas (Detectar Erros)*

- Os **erros** causam um **sublinhado vermelho** no código e o incremento do número exibido ao lado do “x” na barra inferior do Visual Studio Code



```
js > JS exemplo_7.js > frm.addEventListener("submit") callback
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp = document.querySelector("h3")
3
4  frm.addEventListener("submit", (e) => { // "escuta" evento submit do form
5    e.preventDefault( '('' esperado. ts(1005)
6    const num = Number
7    let estrelas = "" exemplo_7.js(9, 8): O analisador esperava localizar um ')' para corresponder ao token '(' aqui.
8    for (let i = 1; i Exibir o Problema Nenhuma correção rápida disponível
9      if (i % 2 == 1 {
10        estrelas = estrelas + "*" // na posição ímpar do i: *
11      } else {
12        estrelas = estrelas + "-" // na posição par: -
13      }
14    }
15    resp.innerText = estrelas // exibe as estrelas
16  })
```

Figura 17 – Alguns erros de sintaxe são detectados pelo próprio editor

- **Depurar Programas (Detectar Erros)**
 - Os recursos de depuração dos browsers permitem identificar tanto **erros** de **sintaxe** quanto **erros** de **lógica**
 - Para analisar os recursos de depuração de programas JS, elaboramos um programa que deverá ler um número que corresponde à quantidade de símbolos que devem ser preenchidos (*em um cheque ou boleto bancário*)
 - O preenchimento deve intercalar os caracteres “*” e “-”



- ***Depurar Programas (Detectar Erros)***
 - ***Exemplo (07):***



Fábrica de Estrelas

Número de Símbolos:

* * * * * * * * * * * * * * * * * * * *

Figura 18 – Renderização do Exemplo (07)

- **Depurar Programas (Detectar Erros)**
 - **Exemplo (07):**

```
html > exemplo_7.html > html > body > form
1  <!DOCTYPE html>
2  <html Lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="../css/estilos.css">
8      <title> Exemplo 07 | bkBank Academy </title>
9  </head>
10 <body>
11     
12     <h1> Fábrica de Estrelas </h1>
13     <form>
14         <p>Número de Símbolos:
15             <input type="number" id="inNumero" min="1" required>
16         </p>
17         <input type="submit" value="Preencher Espaço">
18     </form>
19     <h3 id="outEspacos"></h3>
20     <script src="../js/exemplo_7.js"></script>
21 </body>
22 </html>
```

Figura 19 – HTML do Exemplo (07)

- ***Depurar Programas (Detectar Erros)***
 - ***Exemplo (07):***

```
js > JS exemplo_7.js > frm.addEventListener("submit") callback
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp = document.querySelector("h3")
3
4  frm.addEventListener("submit", (e) => {    // "escuta" evento submit do form
5      e.preventDefault()                    // evita envio do form
6      const num = Number(frm.inNumero.value) // obtém número informado
7      let estrelas = ""                     // variável que irá concatenar as estrelas
8      for (let i = 1; i <= num; i++) {      // cria laço de repetição de 1 até num
9          if (i % 2 == 1) {
10             estrelas = estrelas + "*"      // na posição ímpar do i: *
11         } else {
12             estrelas = estrelas + "-"      // na posição par: -
13         }
14     }
15     resp.innerText = estrelas            // exibe as estrelas
16 })
```

Figura 20 – JS do Exemplo (07)

- **Depurar Programas (Detectar Erros)**
 - Criaremos alguns **erros** no código para verificar o funcionamento do depurador do Google Chrome
 - Modifique o nome da variável **estrelas** para **estrela** na declaração da variável
 - Carregue a página do Exemplo (07) no browser e, no menu superior direito, selecione **Mais Ferramentas > Ferramentas do Desenvolvedor**
 - Na sequência, selecione **Source**
 - Selecione o arquivo **exemplo_7.js**, preencha um valor no campo de formulário da página HTML e clique no botão **Preencher Espaço**

- ***Depurar Programas (Detectar Erros)***
 - *O depurador acusa o erro e exibe a mensagem indicando que a **variável estrelas** não foi definida*
 - *Os **erros de lógica**, por sua vez, são mais difíceis de serem detectados*
 - *Nesse caso, o **programa é executado normalmente**, porém **não apresenta os resultados esperados***
 - *Para isso, o **depurador dispõe de alguns recursos** para nos **auxiliar a identificar os problemas**: **Breakpoints** (pontos de parada) e a **janela Watch** (observador)*

- **Depurar Programas (Detectar Erros)**
 - **Breakpoint:**
 - *define um ponto de parada em uma linha do programa*
 - *o programa é executado até aquela linha e, então, pode-se verificar o valor das variáveis naquele ponto de execução*
 - *Altere novamente o **nome** da **variável** que causou o **erro** anterior (**estrelas**) e atualize a página*
 - *Para criar um **breakpoint**, é necessário apenas clicar na margem esquerda do código, sobre o número da linha onde se deseja definir um ponto de parada*
 - *Pode ser adicionados vários breakpoints no mesmo programa*

- ***Depurar Programas (Detectar Erros)***
 - ***Informe um número no campo de formulário da página HTML e clique no botão para executar o programa JS novamente***
 - ***Observe que a execução irá até a linha onde foi adicionado um **breakpoint*****
 - ***Para prosseguir com a execução do programa, pressione **F8** ou clique na seta da janela de depuração exibida***
 - ***Embora o navegador exiba o conteúdo das variáveis nas linhas em que elas recebem alguma atribuição, também é possível utilizar a janela **Watch** para indicar as variáveis que se deseja observar na execução***

- **Depurar Programas (Detectar Erros)**
 - **Exemplo (07):**

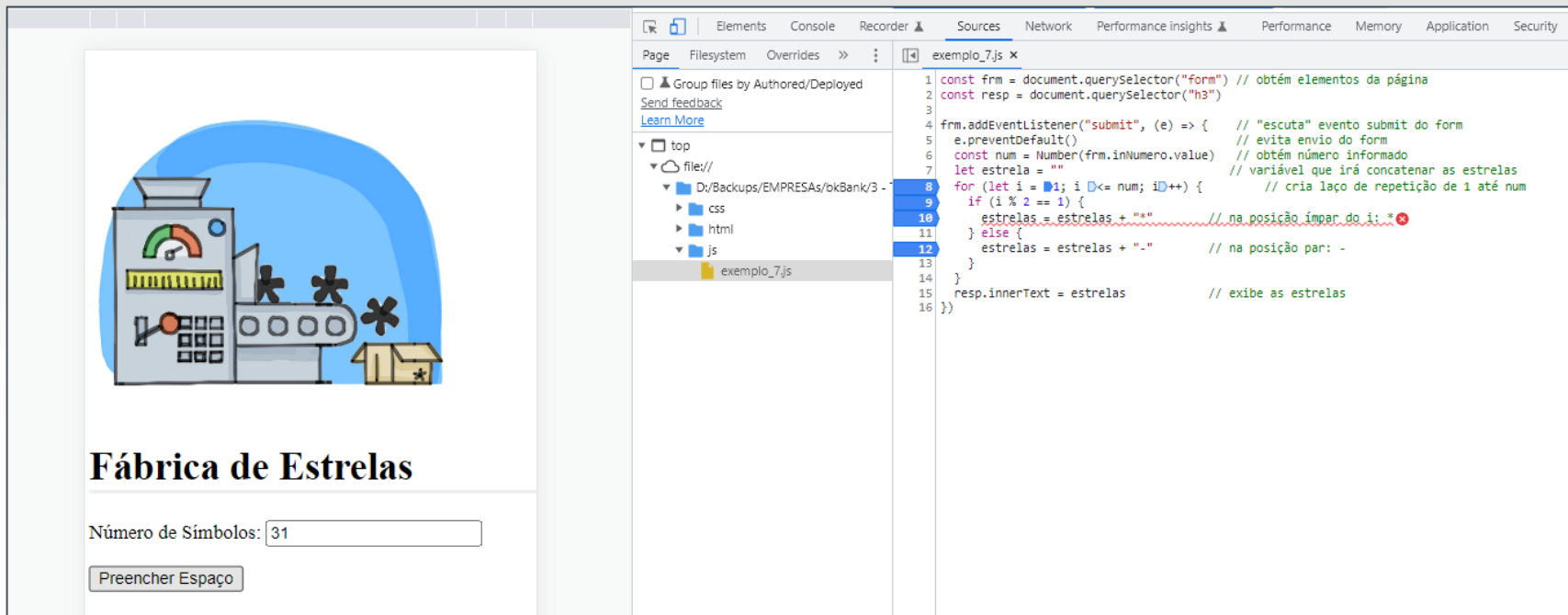


Figura 21 – Depurador posiciona na linha com erro e exibe descrição do problema verificado

- **Depurar Programas (Detectar Erros)**
– **Exemplo (07):**

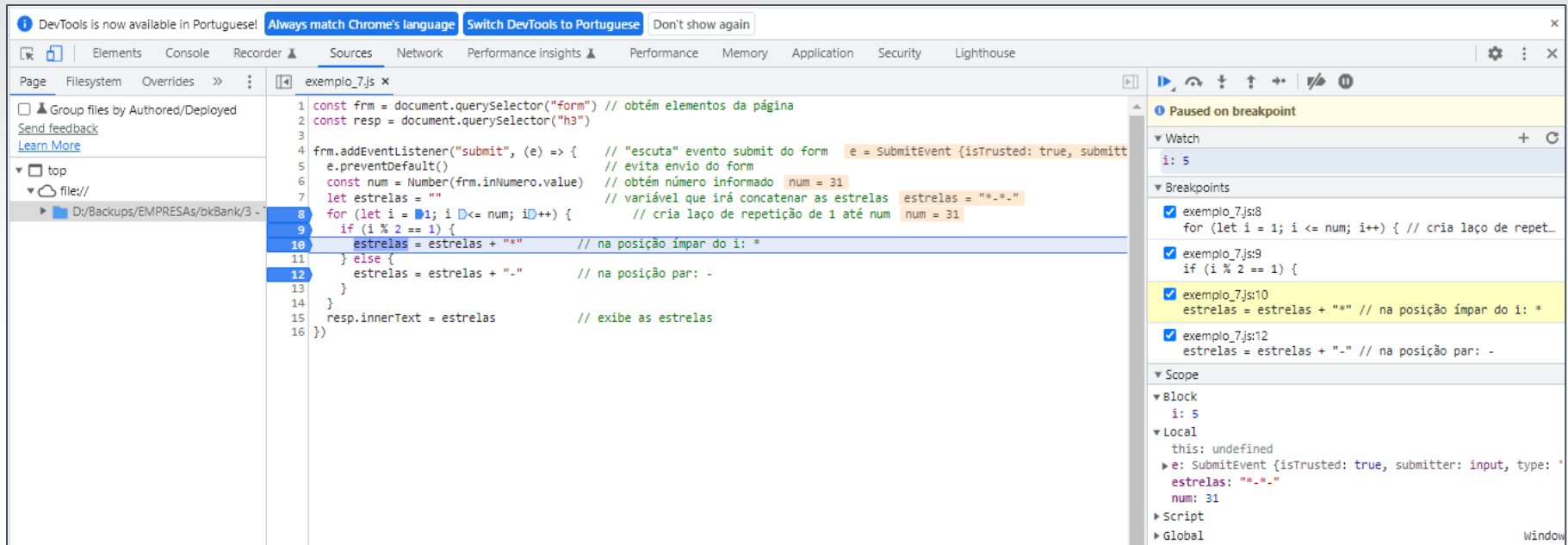
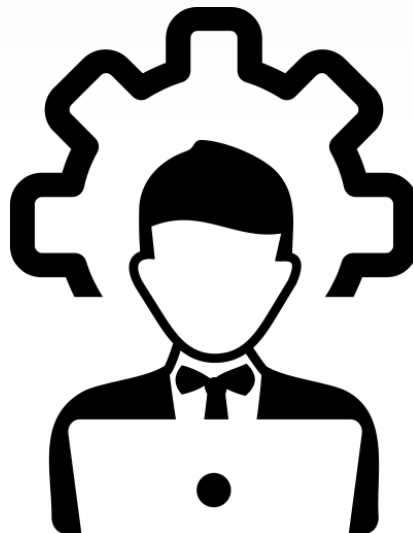


Figura 22 – Depurador do Chrome em ação + Janela Watch

- ***Depurar Programas (Detectar Erros)***
 - *Existem vários outros recursos que podem ser investigados no depurador de programas, como o uso de `console.log()`*
 - *A maioria dos problemas pode ser detectada a partir do uso dos `breakpoints` e pela `observação` do `conteúdo` que as `variáveis` vão assumindo no decorrer do programa*



- **Exemplos com Node.js**

- **Exemplo (08):**

- **A *Copa do Mundo* ocorre de 4 em 4 anos, desde 1930, exceto nos anos de 1942 e 1946 (Segunda Guerra Mundial). Elaborar um programa que repita a leitura de números (anos) até ser digitado 0 (zero). Informe para cada ano se ele é ou não ano de Copa do Mundo**



- **Exemplos com Node.js**

- **Exemplo (08):**

- **A *Copa do Mundo* ocorre de 4 em 4 anos, desde 1930, exceto nos anos de 1942 e 1946 (Segunda Guerra Mundial). Elaborar um programa que repita a leitura de números (anos) até ser digitado 0 (zero). Informe para cada ano se ele é ou não ano de Copa do Mundo**

```
PS D:\Backups\EMPRESAs\bkBank\3 - Treinamentos\5 - JavaScript\src\aula4> node .\js\exemplo_8
Programa Anos de Copa do Mundo. Digite 0 para sair
-----
Ano: 2022
Sim! 2022 é ano de Copa do Mundo!
Ano: 2021
Não... 2021 não é ano de Copa do Mundo.
Ano: 2020
Não... 2020 não é ano de Copa do Mundo.
Ano: 0
```

Figura 23 – Execução do exemplo_8.js

- *Exemplos com Node.js*
 - *Exemplo (08):*

```
js > JS exemplo_8.js > ...
1  const prompt = require("prompt-sync")()
2  console.log("Programa Anos de Copa do Mundo. Digite 0 para sair")
3  console.log("-----")
4  do {
5      const ano = Number(prompt("Ano: "))
6      if (ano == 0) {
7          break // sai da repetição
8      } else if (ano == 1942 || ano == 1946) {
9          console.log(`Não houve Copa em ${ano} (Segunda Guerra Mundial)`)
10     } else if (ano >= 1930 && ano % 4 == 2) {
11         console.log(`Sim! ${ano} é ano de Copa do Mundo!`)
12     } else {
13         console.log(`Não... ${ano} não é ano de Copa do Mundo.`)
14     }
15 } while(true)
```

Figura 24 – Criação do exemplo_8.js

- ***Exemplos com Node.js***

- ***Exemplo (09):***

- ***Elabore um programa que leia o nome de um produto e o número de etiquetas a serem impressas desse produto. Exibir as etiquetas com o nome do produto, com no máximo 2 etiquetas por linha***



- ***Exemplos com Node.js***

- ***Exemplo (09):***

- ***Elabore um programa que leia o nome de um produto e o número de etiquetas a serem impressas desse produto. Exibir as etiquetas com o nome do produto, com no máximo 2 etiquetas por linha***

```
PS D:\Backups\EMPRESAS\bkBank\3 - Treinamentos\5 - JavaScript\src\aula4> node .\js\exemplo_9
Produto: Suco Natural de Uva
Nº de Etiquetas: 7
Suco Natural de Uva      Suco Natural de Uva
Suco Natural de Uva      Suco Natural de Uva
Suco Natural de Uva      Suco Natural de Uva
Suco Natural de Uva
```

Figura 25 – Execução do exemplo_9.js

- *Exemplos com Node.js*
 - *Exemplo (09):*

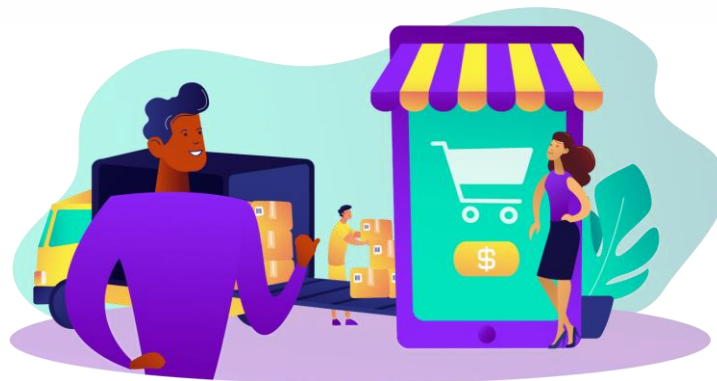
```
js > JS exemplo_9.js > ...
1  const prompt = require("prompt-sync")()
2  const produto = prompt("Produto: ")           // Lê nome do produto e ...
3  const num = Number(prompt("Nº de Etiquetas: ")) // quantidade de etiquetas
4  // Cria um laço de repetição até num/2 (pois imprime 2 etiquetas por linha)
5  for (let i = 1; i <= num / 2; i++) {
6    console.log(`${produto.padEnd(30)} ${produto.padEnd(30)} `)
7  }
8  if (num % 2 == 1) {           // se quantidade solicitada de etiquetas for ímpar...
9    console.log(produto)        // imprime mais uma etiqueta
10 }
```

Figura 26 – Criação do exemplo_9.js

- ***Exemplos com Node.js***

- ***Exemplo (10):***

- ***Elabore um programa para uma loja que leia o valor de uma conta e o número de vezes que um cliente deseja parcelar esse valor (em boletos ou carnê). Para facilitar o troco, o lojista deseja que as parcelas iniciais não tenham centavos, ou seja, centavos apenas na última parcela. Informe como resposta o valor de cada parcela, considerando essa situação***



- ***Exemplos com Node.js***

- ***Exemplo (10):***

- ***Elabore um programa para uma loja que leia o valor de uma conta e o número de vezes que um cliente deseja parcelar esse valor (em boletos ou carnê). Para facilitar o troco, o lojista deseja que as parcelas iniciais não tenham centavos, ou seja, centavos apenas na última parcela. Informe como resposta o valor de cada parcela, considerando essa situação***

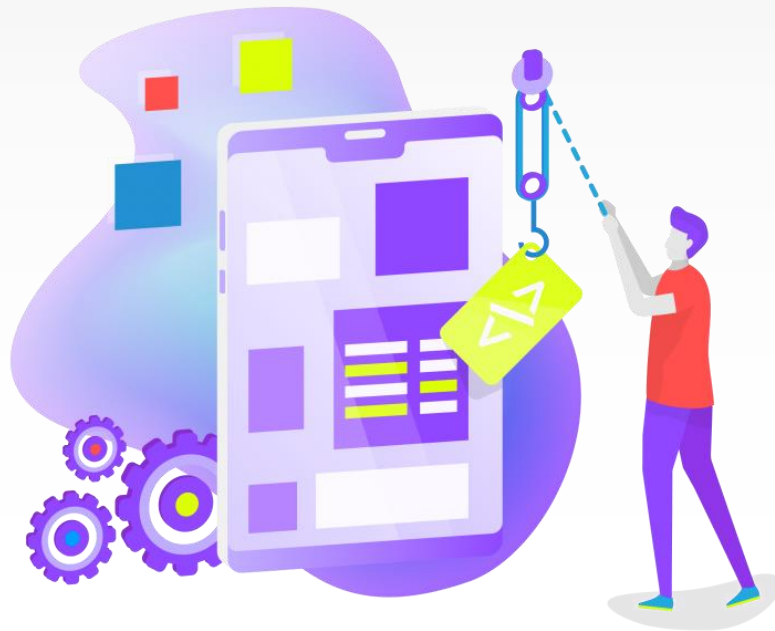
```
PS D:\Backups\EMPRESAS\bkBank\3 - Treinamentos\5 - JavaScript\src\aula4> node .\js\exemplo_10
Valor R$: 91.50
Nº de Parcelas: 3
1ª parcela: R$ 30.00
2ª parcela: R$ 30.00
3ª parcela: R$ 31.50
```

Figura 27 – Execução do exemplo_10.js

- *Exemplos com Node.js*
 - *Exemplo (10):*

```
js > JS exemplo_10.js > ...
1  const prompt = require("prompt-sync")()
2  const valor = Number(prompt("Valor R$: "))           // Lê valor do carnet ...
3  const num = Number(prompt("Nº de Parcelas: "))       // e número de parcelas
4  const valorParcelas = Math.floor(valor / num)        // calcula valor sem decimais
5  const valorFinal = valorParcelas + (valor % num)     // calcula parcela final
6  for (let i = 1; i < num; i++) {                     // enquanto i < num
7    console.log(`${i}ª parcela: R$ ${valorParcelas.toFixed(2)}`)
8  }
9  console.log(`${num}ª parcela: R$ ${valorFinal.toFixed(2)}\n\n`) // última parcela
```

Figura 28 – Criação do exemplo_10.js



EXERCÍCIOS

Referências

Duckett, J.; Javascript e JQuery - Desenvolvimento de interfaces web interativas. Alta Books, 2018.

Flanagan, D.; JavaScript: The Definitive Guide, 7th Edition. O'Reilly Media, Inc. 2020.

Scott A. D., MacDonald M., Powers S.; JavaScript Cookbook, 3rd Edition. O'Reilly Media, Inc. 2021.

