FUNDAMENTOS DA LINGUAGEM

Unidade 01

Sumário

Olá, **Fullturist,**

Seja bem-vindo(a)!

Ao navegar pela internet, seja pelo seu computador ou utilizando seu smartphone, você já deve ter acessado sites como Facebook e Instagram e ficado admirado com funcionalidades fantásticas. Deve ter se deparado também com jogos divertidos e interativos que te prendeu por um bom tempo. Você já se perguntou como tudo isso é feito?

Para desenvolver algo parecido você precisa dominar linguagens que vão além do HTML e CSS. Nesta trilha você irá aprender a linguagem JavaScript, a linguagem de scripts que irá possibilitar que você crie sites e aplicativos semelhantes ao Facebook, Instagram e Tik Tok.

JavaScript é uma linguagem de programação muito utilizada na web. A partir de agora, você será capaz de desenvolver funcionalidades mais complexas. Quando uma página web, como Facebook e Instagram, faz mais do que apenas mostrar informações estáticas e, em vez disso, mostram em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, entre outros, você pode ter certeza de que o JavaScript está por traz de tudo. Ficou empolgado(a)? Então, vamos lá!

Unidade 01

Fundamentos da linguagem

Olá, Fullturist!

Esta é a Unidade 1 da Trilha de Aprendizagem JavaScript. Aqui, você terá a oportunidade de conhecer e/ou se aprofundar em temas como o que é o JavaScript e para que o utilizamos, declarações de variáveis, tipos de dados, estruturas de controle, funções e arrays.

Vamos lá?

Objetivos de aprendizagem

Ao final do estudo desta unidade, você será capaz de:

- Compreender o funcionamento da linguagem de programação JavaScript;
- Diferenciar declaração de variáveis com var, let e const e saber exatamente quando utilizar;
- Interpretar condições lógicas e repetição;
- Criar blocos de códigos reaproveitáveis com definição e programação de funções;
- Manipular estruturas de dados array.

1.1 Introdução

Olá Fullturist, que bom que você chegou à Trilha de Aprendizagem 4!

Nossa jornada com programação JavaScript está prestes a começar. Sabe o que isso significa? Que iniciaremos uma jornada sensacional onde você irá adquirir a experiência básica para decolar como profissional Full Stack JavaScript. Nesta trilha, vamos aprender o que é JavaScript, para que serve o JavaScript, vamos dar os primeiros passos, entender os fundamentos da linguagem, as variáveis, os formulários, DOM, as estruturas de decisão e de controle, as funções, os eventos e muito mais. E, no final da trilha, iremos desenvolver uma aplicação prática para fixar todos esses conhecimentos estudados. Ficou curioso(a) ou empolgado(a)? Então, vamos decolar!

Antes de iniciarmos o nosso mergulho nos conhecimentos sobre JavaScript é importante você conhecer a ferramenta e o ambiente que iremos utilizar para desenvolver e testar nossos códigos.

1.2 Configurando o ambiente para trabalhar

Em toda a trilha, vamos utilizar a ferramenta Visual Studio Code (VSCode) para desenvolver e testar nossos códigos e exercícios em JavaScript.

O Visual Studio Code (VSCode) é um editor de código-fonte, desenvolvido pela Microsoft, muito utilizado pelos desenvolvedores. A maioria dos desenvolvedores opta por utilizar o VCode porque ele inclui suporte para depuração, controle de versionamento Git incorporado, realce de sintaxe, complementação inteligente de código, snippets e refatoração de código. Outro fator importante na escolha do VSCode é que ele permite ao desenvolvedor programar em várias linguagens diferentes, entre elas o JavaScript.

Caso você ainda não possua o VSCode instalado, clique aqui e veja como realizar a instalação.

Saiba mais sobre o Visual Studio Code e conheça todo o poder que esta ferramenta possui.

Vamos utilizar o VSCode para desenvolver os códigos JavaScript e, também, os códigos HTML. Em determinados momentos, você irá testar os códigos JavaScript, utilizando o console do VSCode. Em outros, quando você desenvolver códigos JavaScript para manipular o arquivo HTML, usará um navegador de internet. Por este motivo, vale conhecer algumas particularidades do JavaScript em navegadores como Chrome, Firefox e Internet Explorer.

Os navegadores Chrome e Firefox são navegadores mais modernos e que não possuem problemas na execução das versões mais atuais do JavaScript. Inclusive, esses dois navegadores possuem ferramentas que auxiliam o desenvolvedor. Uma dessas ferramentas é o console, onde é possível verificar se existe erros no seu código e, também, a saída de dados, as mensagens e os resultados de processamentos que você programa para serem exibidos. Para usar essa ferramenta no Chrome, acesse o menu e, em seguida, a opção Mais Ferramentas e selecione a opção Ferramentas do Desenvolvedor. Já o Internet Explorer possui uma série de problemas ao executar códigos JavaScript, que variam dependendo da versão do navegador. Então, recomendamos que você não utilize esse navegador.

1.3 Como funciona o javascript

O JavaScript é uma linguagem que funciona em conjunto com o HTML e o CSS.

Podemos programar os nossos códigos de duas formas:

- JavaScript interno – é quando programamos os códigos JavaScript dentro do próprio arquivo HTML utilizando uma tag <script>, como pode ser observado na Figura 1.

- Arquivo externo – é quando programamos os códigos JavaScript em um arquivo separado do arquivo HTML. O nome do arquivo é composto pela extensão .js, como exemplo, index.js. Quando utilizamos essa forma de programação, é necessário informar ao HTML qual arquivo com o código JavaScript será vinculado e, para isso, utilizamos a tag script com o parâmetro src com o diretório e nome do arquivo, conforme pode ser observado na Figura 2.

```
1   <!DOCTYPE html>
2   <html lang="pt-br">
3   <head>
4       <meta charset="UTF-8">
5       <title>Minha Página</title>
6
7       <script>
8           // código Javascript aqui
9       </script>
10  </head>
11  <body>
12
13  </body>
14  </html>
```

Fonte: Elaborada pelo autor (2021).

```
1   <!DOCTYPE html>
2   <html lang="pt-br">
3   <head>
4       <meta charset="UTF-8">
5       <title>Minha Página</title>
6   </head>
7   <body>
8
9
10      <script src="index.js"></script>
11
12  </body>
13  </html>
14
```

Fonte: Elaborada pelo autor (2021).

O JavaScript é uma linguagem executada pelos navegadores. Isso quer dizer que os navegadores fazem o download dos códigos para serem executados no dispositivo do usuário. Então, como a execução é feita no próprio navegador, não existe a necessidade de uma conexão de internet.

Quando uma linguagem é executada pelo navegador, dizemos que a linguagem é client--side. Com ela, um código desenvolvido em JavaScript é executado sempre que um usuário interage com o site, como, por exemplo, quando um botão na tela é pressionado.

EXEMPLO

Imagine que você está realizando a compra de alguns produtos em um e-commerce e, ao selecionar o carrinho de compra, todos os produtos são apresentados, bem como o valor total a pagar. As ações de seleção de produtos e cálculo do valor total são executados pelo JavaScript no computador do usuário. Uma conexão de internet e o envio de dados para um servidor só ocorrem quando o pedido é finalizado.

Agora que você entendeu como o JavaScript funciona, vamos começar a jornada de aprendizagem dessa tecnologia?

1.4 Fundamentos da linguagem JavaScript

Javascript é uma linguagem de programação que utilizamos para adicionar interatividade aos sites, criar animações, executar alguma funcionalidade quando você pressiona um botão ou quando você preenche dados em um formulário.

Com JavaScript também podemos criar jogos!

Por essa você não esperava, não é?

JavaScript é uma linguagem de programação que roda (é executada) no navegador do usuário (front-end).

Quando nos referimos a front-end, estamos falando de tudo que está nas mãos dos usuários, tudo o que ele pode visualizar e interagir por meio do navegador de internet. O javascript também é executado no back-end em servidores, mas, originalmente, no início do javascript, ele foi criado apenas para ser executado no navegador.

1.4.1 O que podemos fazer com JavaScript?

Como dito anteriormente, podemos adicionar interatividade e ações às páginas web, e até mesmo criar aplicações web completas. Também podemos criar aplicativos usando React Native, que é um framework que vai nos ajudar a construir aplicativos para funcionar no smartphone, e aplicativos desktop usando uma biblioteca chamada eléctron, um framework que nos ajudará a construir aplicações para este tipo de plataforma.

Aliás, você conseguiu perceber que JavaScript é multiplataforma?

Isso quer dizer que podemos utilizá-la em nosso smartphone, em nosso computador e na Web por meio dos navegadores.

Fonte: diy13 / Shutterstock ID 1912729303

Existem várias empresas famosas que utilizam o JavaScript, dentre elas estão o Facebook e sua rede de aplicativos, como Instagram, Whatsapp e TikTok, o Google, com o Youtube, Gmail e Driver, o Uber, a Netflix, enfim, a lista é grande: praticamente 99,99% dos sites na web usam JavaScript! Concluímos, então, que JavaScript é uma linguagem obrigatória para quem quer se tornar um programador de sucesso.

1.5 Variáveis

Antes de começarmos a falar de variáveis, que tal recordarmos os componentes básicos de um computador?

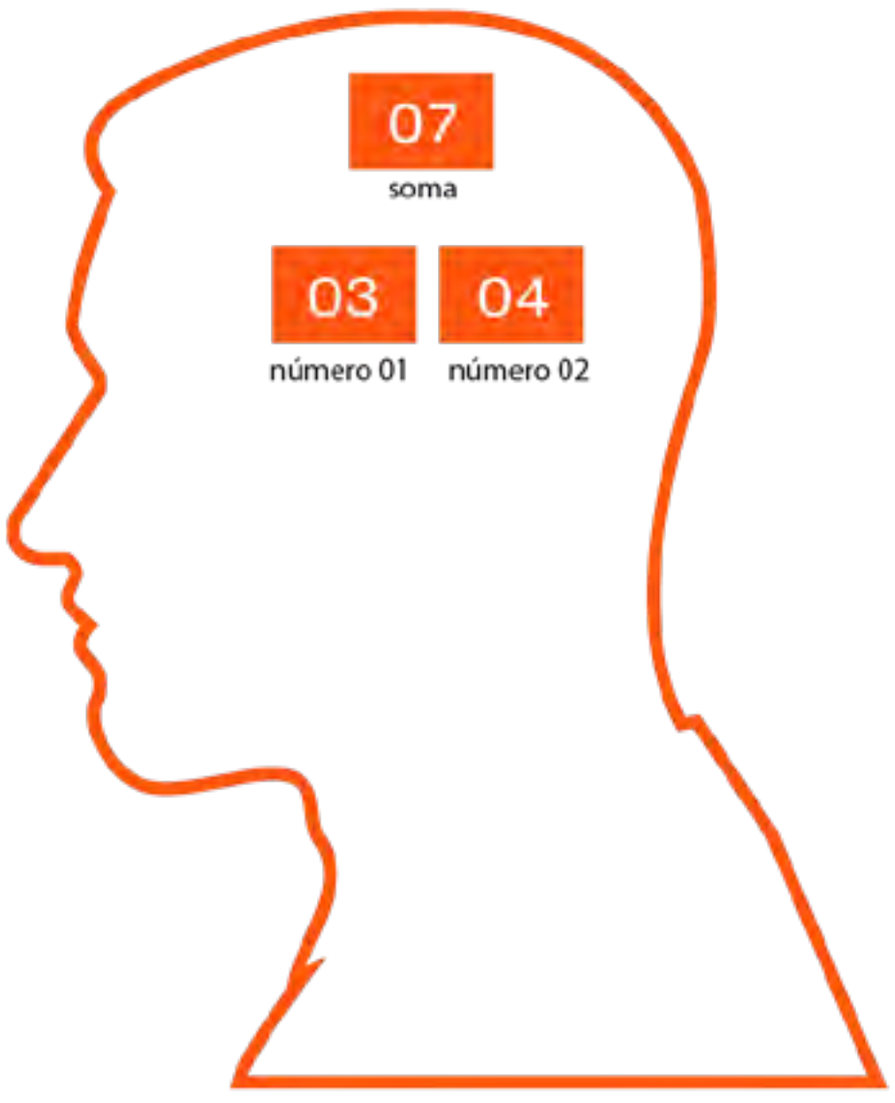
Você se recorda quais são?

Os periféricos de entrada de dados, como teclado e mouse; a memória, que guarda (armazena) temporariamente esses dados; e o processador, que executa os cálculos ou tarefas com esses dados. Recordou? Nesse momento vamos focar na memória. Para que a memória guarde temporariamente os dados, ela precisa reservar locais, espaços que são como caixas. Essas caixas são as variáveis que irão guardar as informações, entendeu?

Para ficar ainda mais claro, vamos analisar o que são variáveis por meio de um exercício.

EXEMPLO

Primeiramente, pense no número 3 e, agora, guarde esse número na cabeça. Guardou?
Em seguida pense em um outro número, o 4, e guarde ele também na cabeça.
Agora, você está com os números 3 e 4 guardados em sua mente.
Para finalizar o exercício, calcule a soma dos dois números. Qual o resultado?
O resultado, provavelmente, também está guardado em sua cabeça. Isso quer dizer que nosso cérebro é como um computador, onde primeiro guardamos informações na memória para, depois, realizarmos alguma ação com essas informações.



Fonte: Elaborada pelo autor (2021).

No computador, o mesmo acontece. Você consegue perceber a semelhança?

Uma variável é um espaço na memória do computador, uma caixa onde os dados são guardados temporariamente para que o processador consiga executar ações ou cálculos com esses dados.

Para facilitar a utilização de uma variável damos um nome a ela, colocamos uma etiqueta na nossa caixinha. Por exemplo, no exercício anterior, poderíamos nomear uma variável como “numero1” para guardar o número 3 e nomear como “numero2” para guardar o valor 4.

Agora ficou fácil, não?

1.5.1 Declaração de variáveis

Agora que já entendemos o que são variáveis, que tal começarmos a praticar?

Para que possamos programar uma variável, primeiramente, precisamos criá-la. O ato de criar uma variável em um código é chamado de declaração de variável.

No código abaixo apresentamos a declaração de três variáveis que armazenam diferentes valores.

Código 1 - Exemplo de criação de variáveis

```
var nome;

var idade;

nome = “João”;

idade = 19;

console.log(nome);

console.log(idade);
```

Fonte: Elaborada pelo autor (2021).

No código acima, nas linhas 1 e 2 estamos declarando duas variáveis chamadas nome e idade. Uma vez declarada uma variável, você pode inicializá-la com valores. Na linha 3, estamos inicializando a variável nome com o texto “João”, e inicializando a variável idade com o número 19 na linha 4. Abra o seu VSCode, teste o código acima e veja que o resultado exibido será exatamente os valores “João” e 19.

Vamos entender um pouco melhor o que está acontecendo na declaração das variáveis nome e idade. Para a declaração das variáveis está sendo usada a palavra

reservada “var”, e o operador “=” está atribuindo os valores que estão sendo armazenados, ou seja, está armazenando o texto “João” na variável nome e o valor 19 na variável idade.

Você pode declarar e inicializar as variáveis ao mesmo tempo.

Inicializar significa atribuir o valor inicial de uma variável. Declarar e inicializar variáveis ao mesmo tempo é muito comum e, provavelmente, você irá fazer isso na maioria das vezes em seus códigos.

EXEMPLO

Código 2 - Declaração e inicialização de variáveis

```
var nome = “João”;

var idade = 19;

console.log(nome);

console.log(idade);
```

Fonte: Elaborada pelo autor (2021).

Teste o código acima no VSCode e veja o resultado exibido. Como exercício, declare uma variável chamada escola e atribua a ela o texto “Fullture”. Em seguida, exiba o seu valor no console.

Existem algumas regras para criação dos nomes das variáveis em Javascript. O nome de uma variável deve começar com uma letra, ou com um underline (_), ou com um símbolo cifrão ($). Os demais caracteres do nome da variável podem ser números (0-9) ou letras, ok? Não devem existir espaços nos nomes das variáveis! Se houver necessidade de criar uma variável com nome composto, utilize underline (_), por exemplo: "nome aluno" pode ser criado como nome_aluno. Outro ponto muito importante é que Javascript é case-sensitive, isso quer dizer que uma variável nome (escrita inteira com fonte minúscula) é diferente de uma variável Nome (escrita com a primeira letra maiúscula).

Vimos até agora que as variáveis são locais onde iremos armazenar informações que serão utilizadas posteriormente. Porém, você precisa entender que cada informação possui o seu tipo bem definido para que tudo funcione corretamente. Por isso, no próximo tópico, vamos te apresentar quais são esses tipos. Assim, continue com a leitura para que o uso de variáveis fique ainda mais interessante.

1.5.2 Tipagem

Em JavaScript as variáveis são dinamicamente tipadas, o que significa que você não precisa declarar seu tipo. Quando declaramos e atribuímos valores a uma variável, o JavaScript define o seu tipo de acordo com o valor que atribuímos a ela. Por exemplo, no código 2, quando atribuímos o texto "João" à variável nome, o Javascript automaticamente define o tipo como String. Assim como ocorre com a variável idade que assume o tipo número.

Vamos conhecer os tipos de variáveis?

1.5.3 Tipos de variáveis

O JavaScript define seis tipos de dados de uma variável. São eles:

- String - qualquer valor entre aspas simples ou aspas duplas, incluindo números. Exemplos: var nome = “João”; var cidade = “São Paulo”; var ano = ‘2021’;
- Number - são números inteiros e fracionados. Exemplos: var idade = 19; var troco = 10.5;
- Booleano - são os valores true (verdadeiro) ou false (falso);
- Null - é uma palavra-chave que indica que um valor de uma variável é nulo;
- Undefined - quando apenas declaramos uma variável e não inicializamos com algum valor, seu valor padrão é indefinido, ou seja, undefined;
- Object - variáveis que armazenam arrays, objetos ou funções. Veremos mais sobre elas posteriormente em outras seções.

1.5.4 Var x Let x Const

Vimos, anteriormente, que para declarar uma variável utilizamos a palavra reservada “var” e, logo em seguida, inicializamos, ou não, atribuindo algum valor a essa variável. Mas existem outras formas que você pode declarar uma variável.

Em JavaScript você pode declarar uma variável de três formas:

- Utilizando a palavra reservada “var” - como visto nos exemplos anteriores, você pode declarar uma variável utilizando a palavra reservada “var”. Por exemplo, var nome = “João”, var escola_top = “Fullture”, var idade = 19;
- Utilizando a palavra reservada “let” - você pode declarar variáveis utilizando a palavra reservada “let”, também podendo inicializá-la ou não ao mesmo tempo que a declaração. Por exemplo: let nome = “João”; let idade = 19;

> Utilizando a palavra reservada “const” - você também pode declarar variáveis utilizando a palavra reservada “const”. Uma variável declarada com “const” é uma variável especial, uma constante. Ou seja, o valor dessa variável é constante e nunca poderá ser mudado.

Neste ponto, temos a certeza de que você entendeu que existem três maneiras de declarar variáveis: var, let e const. Mas você deve estar se perguntando: qual a diferença entre as formas?

1.5.6 Declaração de variáveis com var

Utilizamos “var” tanto para declarar variáveis locais como variáveis globais. As variáveis declaradas com “var” podem ser utilizadas antes da sua declaração pois o JavaScript executa algo chamado Hoisting. Em outras palavras, todas as variáveis declaradas com “var” são elevadas para o início do código em tempo de execução, indiferente de onde foram declaradas.

Além disso, uma variável declarada com “var” pode ser redeclarada em outras partes dos códigos sem causar erros na execução do JavaScript.

Para entender melhor, analise o exemplo abaixo:

EXEMPLO

Código 3 - Declaração de variáveis com var

```
var nome = “João”;
console.log(nome);
console.log(idade);
Var idade = 19;
var nome = “Maria”;
console.log(nome);
```

Fonte: Elaborada pelo autor (2021).

No código de exemplo acima, na linha 1 está sendo declarada uma variável nome. A String “João” está sendo atribuído a ela e, logo em seguida, na linha dois, ela está sendo utilizada e seu valor exibido no console. Na linha 4, a variável idade está sendo utilizada e seu valor sendo exibido no console antes mesmo que ela seja declarada. A declaração da variável idade só ocorre na linha 5.

Em outras linguagens isso geraria um erro, mas, em JavaScript, este erro não ocorre porque todas as variáveis declaradas com “var” são movidas para o início do código em tempo de execução (Hoisting). Para finalizar o nosso exemplo, a variável nome está sendo declarada novamente na linha 7 e nenhum erro ocorre. Abra o seu VSCode e teste o código do exemplo. Analise o que ocorre com a variável idade e veja o resultado no console da exibição da variável nome.

O JavaScript permite o uso de variáveis antes da sua declaração quando esta variável é declarada com “var”, mas isso não é recomendado e nem é uma boa prática de programação pois complica a manutenção do código e pode gerar bugs inesperados. Bugs também podem ocorrer com as redeclarações de variáveis, como ocorreu com a variável nome – outra prática não recomendada!

1.5.7 Declaração de variáveis com let

A declaração de variáveis com a palavra reservada “let” foi criada nas versões mais modernas do JavaScript e o comportamento dessas variáveis é um pouco diferente das variáveis declaradas com “var”. Primeiramente, o Hoisting não funciona com variáveis declaradas com “let”, isso quer dizer que primeiro você precisa declarar a variável para depois utilizá-la em seu código. Esse comportamento já evita que ocorram bugs inesperados como ocorria com variáveis declaradas com “var”. Uma variável declarada com “var” pode ser declarada novamente quantas vezes você quiser, mas isso não ocorre com “let”. O JavaScript não permite que variáveis declaradas com “let” sejam declaradas novamente, mas o seu valor pode mudar quantas vezes forem necessárias durante o código. E por fim, uma variável declarada

com “let” possui um escopo de bloco, ou seja, ela só existe dentro do bloco de código que a criou.

Para você entender melhor tudo isso vamos a um outro exemplo:

EXEMPLO

Código 4 - Declaração de variáveis com let

```
let nome = “João”;
console.log(nome);
nome = “Maria”;
console.log(nome);
function exibe_nome(){
    let nome = “José”;
    console.log(nome);
}
exibe_nome();
console.log(nome);
let nome = “Pedro”;
console.log(idade);
let idade = 19;
```

Fonte: Elaborada pelo autor (2021).

Vamos analisar o código acima. Na linha 1, declaramos a variável nome com “let” e inicializamos o seu valor com a String “João”. Seu valor foi exibido no console na linha 2. Na linha 4, foi atribuído um novo valor à variável nome, a String “Maria” e, na linha 5, o seu novo valor está sendo exibido no console.

Até agora nada de novo, certo?

Na linha 7 foi criado uma função com o nome exibe_nome (falaremos mais sobre funções logo adiante). Uma função é um bloco de código. Como variáveis declaradas com “let” possuem escopo de bloco, não vão ocorrer erros com a declaração da variável nome na linha 8. Isso porque a variável nome declarada na linha 8 só

existe dentro desta função, dentro deste bloco de código, ou seja, entre a abertura e fechamento das chaves “{}”, por isso não ocorrem erros. A variável nome declarada na linha 8 é diferente da variável nome declarada na linha 1. Por fim, o JavaScript vai informar que existem dois erros no código. O primeiro ocorre na linha 15, onde a variável nome está sendo declarada novamente. O outro ocorre na linha 17, pois a variável idade está sendo usada antes de ser declarada. Teste o código no VSCode e confira todas essas questões. Primeiro teste os códigos entre as linhas 1 a 13. Depois, faça um segundo teste com o código completo.

É recomendado que você sempre declare variáveis utilizando “let” para evitar que ocorram bugs inesperados no seu código. Utilize declaração de variáveis com “var” somente se for realmente necessário.

1.5.8 Declaração de Variáveis com const

Outra forma de declaração de variáveis é utilizar a palavra reservada “const”. Uma variável declarada com “const” deixa de ser uma variável e se torna uma constante! Uma constante é uma variável que não pode ter seu valor redeclarado, por isso o nome constante. Seu valor será sempre o mesmo, não pode ser alterado como ocorre com os valores das variáveis declaradas com “var” e “let”.

Teste o código abaixo no VSCode e veja o erro emitido pelo JavaScript:

EXEMPLO

Código 5 - Declaração de variáveis com const

```
const nome = “José”;
console.log(nome);
nome = “Maria”;
const nome = “Pedro”;
```

Fonte: Elaborada pelo autor (2021).

No código acima, dois erros serão exibidos pelo JavaScript. O primeiro erro ocorre na linha 3, pois constantes não podem ter seu valor alterado. O outro erro ocorre na linha 4, pois, assim como ocorre com as variáveis declaradas com “let”, constantes também podem ser declaradas apenas uma vez.

Você utilizará constantes quando estiver programando arrays, objetos e componentes em React e React Native.

1.6 Estruturas de controle

Durante o desenvolvimento de um programa, muitas vezes precisamos programar ações que serão executadas, dependendo de determinadas condições, e pensar na melhor forma de controlar o fluxo de execução para obter o melhor resultado.

O JavaScript disponibiliza algumas estruturas de controle que permite a você executar o código baseado em condições ou repetir um bloco de código várias vezes, dependendo da necessidade. Essas estruturas são: IF...THEN, SWITCH, FOR e WHILE. IF e SWITCH, que são estruturas de decisão; e FOR e WHILE, que são estruturas de repetição.

Vamos estudar como essas estruturas funcionam e para que servem?

1.6.1 Estruturas de decisão IF e SWITCH

Vamos começar com a estrutura IF. Ela executa um bloco de código se uma determinada condição especificada for verdadeira. Além disso, nela também podemos determinar a execução de um fluxo alternativo de código caso a condição especificada seja falsa. Neste caso estamos nos referindo ao ELSE.

EXEMPLO

Vamos verificar se uma determinada pessoa tem idade mínima para possuir uma habilitação de motorista. Nesse caso temos que, se uma pessoa tem idade igual ou superior a 18 anos, essa pessoa poderá adquirir uma habilitação de motorista; se não, ela não tem idade mínima para isso.

Código 6 - Estrutura de decisão IF

```
let idade = 18;
if ( idade < 18 ){
console.log(“Você ainda não possui idade mínima”);
} else {
console.log(“Você possui idade mínima, pode adquirir   habilitação!”);
}
```

Fonte: Elaborada pelo autor (2021).

No código acima, na linha 3, temos uma condição para que o seu bloco de código seja executado (o console da linha 4 que está entre as chaves). A mensagem da linha 4 só será exibida no console caso a idade armazenada na variável for menor que 18. Como o valor 18 é o valor armazenado na variável, então a condição será falsa. Nesse caso, o bloco de código do else (linha 5) será executado exibindo a mensagem “Você possui idade mínima, pode adquirir habilitação!” no console. No seu VSCode, teste o código modificando os valores da variável idade. Primeiramente, teste como está, depois teste com um valor menor que 18.

Para programar as condições na estrutura IF, precisamos fazer uso de alguns operadores lógicos. Esses operadores são listados a seguir:

Tabela 01 – Operadores lógicos

Operador	Significado
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
===	Igual a (o tipo da variável deve ser igual, ou seja, “10” === 10 retornaria false)
==	Igual a (independentemente do tipo de dados da variável, ou seja, “10” == 10) retornaria true);
!==	Diferente de (o tipo da variável deve ser igual, ou seja, “10” !== 10 retornaria true)
!=	Diferente de (independentemente do tipo de dados da variável, ou seja, “10” != 10 retornaria false)

Fonte: MDN / developer.mozilla.org/pt-BR/docs/web/JavaScript/Guide/Expressions_and_operators.

Alterando o código anterior, vamos verificar se a idade da pessoa é maior ou igual a 18.

O código ficaria assim:

Código 7 - Estrutura de decisão IF com o código alterado

```
let idade = 18;
if ( idade >= 18 ){
      console.log(“Você possui idade mínima, pode adquirir habilitação!”);
} else {
       console.log(“Você ainda não possui idade mínima”);
}
```

Fonte: Elaborada pelo autor (2021).

Algumas vezes podemos ter mais de uma condição a ser testada e mais de dois blocos de código para serem executados dependendo da condição. Nesse caso, a estrutura IF não é necessária, pois ela valida se a condição é verdadeira ou falsa. Então, é mais recomendado utilizar a estrutura SWITCH.

A estrutura SWITCH também é uma estrutura condicional. SWITCH permite executar um bloco de código diferente de acordo com cada condição (case) especificada. Utilizamos switch sempre que temos valores predefinidos a serem analisados.

Veja a escolha de uma opção no menu de um aplicativo:

EXEMPLO

Código 8 - Estrutura de decisão SWITCH

```
let opcao = 1;
switch(opcao){
      case 0:
        sair();
       breack;
      case 1:
enviar_mensagem();
```

```
breack;
  case 2:
enviar_foto();
   breack;
  case 3:
enviar_video();
   breack;
  default:
exibir_mensagens();
   }
```

Fonte: Elaborada pelo autor (2021).

No código acima foi utilizada a estrutura de decisão SWITCH para executar funções diferentes dependendo do valor da variável opção. É muito usual utilizar essa estrutura na programação de menus como está sendo simulado nesse código entre as linhas 3 a 18. O código acima é apenas ilustrativo, porém, lanço aqui um desafio: para testar o código no VSCode, você terá que criar as funções enviar_mensagem(), eviar_foto(), enviar_video() e exibir_mensagem(). Crie as funções e, dentro de cada uma delas, utilize o console para exibir uma mensagem referente a cada uma. Troque o valor da variável opção para ver a execução.

1.6.2 Estruturas de repetição FOR e WHILE

Estruturas de repetições são utilizadas quando é necessário que determinado bloco de código seja executado mais de uma vez. A utilização dessas estruturas nos permite não repetir códigos, facilitando sua manutenção e evitando bugs.

Vamos começar pela estrutura de repetição FOR. Utilizamos essa estrutura quando sabemos o número de vezes em que um bloco de código deve ser repetido. Para entendermos melhor, vamos analisar o exemplo de um código que calcula a tabuada do número 5.

Primeiramente, vamos analisar um código sem a utilização da estrutura de repetição FOR.

Código 9 - Exemplo de código sem a estrutura de repetição FOR

```
let numero = 5;
let resultado = numero * 1;
console.log(resultado);
let resultado = numero * 2;
console.log(resultado);
let resultado = numero * 3;
console.log(resultado);
let resultado = numero * 4;
console.log(resultado);
let resultado = numero * 5;
console.log(resultado);
let resultado = numero * 6;
console.log(resultado);
let resultado = numero * 7;
console.log(resultado);
let resultado = numero * 8;
console.log(resultado);
let resultado = numero * 9;
console.log(resultado);
let resultado = numero * 10;
console.log(resultado);
```

Fonte: Elaborada pelo autor (2021).

Perceba que, no código acima, os códigos da linha 2 e da linha 3 se repetem para os demais números da tabuada até chegar no número 10. Esse código, além de longo, dificulta a manutenção ou uma possível correção, pois seria necessário alterar todas as linhas no caso de algum erro. O próximo código apresenta o cálculo de uma tabuada do número 5 com a utilização da estrutura de repetição FOR.

Código 10 - Código alterado utilizando a estrutura de repetição FOR

```
let numero = 5;
let resultado;
for( let i = 0; i <=10; i++){
    resultado = i * numero;
    console.log(resultado);
}
```

Fonte: Elaborada pelo autor (2021).

O código ficou menor e bem mais simples de manter, não acha? Vamos entender o que está acontecendo. Na linha 1, declaramos e inicializamos a variável número com o valor 5. Em seguida, declaramos a variável resultado para ser utilizada dentro do bloco de código estrutura de repetição FOR. Na linha 3, iniciamos a estrutura de repetição FOR. Essa estrutura é dividida em três partes separadas por ponto e virgula (;). Na primeira parte declaramos uma variável i. Esta variável é, na verdade, um índice que irá mudar o seu valor a cada vez que o bloco de código for executado; Na segunda parte, após o primeiro ";", especificamos a condição de parada da repetição do bloco de código. Nesse caso, o bloco de código será repetido enquanto o valor do índice i for menor ou igual a 10; e, na última parte, após o segundo ";", é incrementado o valor de i a cada repetição. Na primeira vez que o bloco de código for executado, uma multiplicação será calculada com os valores da variável i e da variável número. O resultado dessa multiplicação é guardado na variável resultado, que é exibida no console na linha 5.

No exemplo acima, sabíamos a quantidade de vezes que o bloco de código deveria ser executado, por isso utilizamos a estrutura de repetição FOR. Mas e quando não soubermos esse número de repetição? O que fazer? Neste caso, utilizaremos a estrutura de repetição WHILE.

A estrutura de repetição WHILE executa um bloco de instrução até quando sua condição de parada seja avaliada como verdadeira.

Para entender melhor vamos analisar o código abaixo.

EXEMPLO

Código 11 - Exemplo de código com a estrutura de repetição WHILE

```
let total_alunos = 4;
let alunos = 1;
while( alunos < total_alunos ){
console.log(“Aluno numero: “, alunos);
alunos++;
}
```

Fonte: Elaborada pelo autor (2021).

Neste momento, você deve estar se perguntando: mas qual é a diferença entre a estrutura WHILE e a estrutura FOR? Por que existe uma estrutura que faz a mesma coisa? Vamos começar entendendo a sintaxe da estrutura WHILE.

Primeiramente, no WHILE temos apenas uma condição de parada que é avaliada toda vez que o bloco é repetido; inclusive a condição é avaliada antes de a repetição iniciar. Quando esta condição for falsa, a repetição termina. Para a repetição terminar, algo no bloco de código que está sendo executado repetidamente deve acontecer. No caso do exemplo, a cada repetição do código, na linha 5 é adicionado 1 ao valor da variável alunos.

Mas uma grande diferença da estrutura WHILE em comparação com a estrutura FOR é que a estrutura WHILE pode ser executada infinitamente. Vamos ver um exemplo para entendermos melhor:

Código 12 - Exemplo de código com o problema de loop infinito

```
let acumulador = 0;
while( true ){
    console.log(“valor do acumulador: “, acumulador);
    acumulador++;
}
```

No código do exemplo acima, o valor do acumulador vai ser executado infinitamente ou até esgotar a memória do computador. Isso porque a condição de parada do WHILE, na linha 2, foi estabelecida como true, ou seja, a condição de parada sempre vai ser verdadeira e o acumulador vai seguir acumulando infinitamente. Como toda estrutura de repetição precisa de uma condição de parada, para resolver o problema adicionamos, na linha 5 do código abaixo, uma condição de parada utilizando a estrutura de decisão IF. O WHILE irá parar de executar quando o acumulador alcançar o valor 100, então, a condição do IF será verdadeira e a instrução break da linha 6 irá ser efetuada, interrompendo a execução.

Código 13 - Solução do problema do loop infinito com condição de parada interna

```
let acumulador = 0;
while( true ){
    console.log(“valor do acumulador: “, acumulador);
    acumulador++;
    If ( acumulador === 100){
        break;
    }
}
```

Fonte: Elaborada pelo autor (2021).

A estrutura de repetição WHILE é muito utilizada para executar menus de seleção em que o menu só para de ser exibido quando os usuários selecionam a opção “sair”.

1.7 Funções

Funções são blocos de códigos reutilizáveis que realizam tarefas ou cálculos.

Para você entender melhor o que é uma função, para que ela serve e por que devemos utilizá--las, vamos citar um exemplo do seu dia a dia.

EXEMPLO

Vamos supor que você queira criar um aplicativo como o WhatsApp. Na lista de conversas, você quer exibir o nome e a foto das pessoas com quem você conversou. No código abaixo, vamos simular, de forma simples, a exibição desses dados. Teste o código no seu VSCode e veja que o resultado será a exibição do texto “foto” e do texto “João”, que seria o nome do seu contato.

Código 14 - Exibição de dados no console

```
console.log(“foto”);
console.log(“João”);
```

Fonte: Elaborada pelo autor (2021).

Agora, suponha que você precise exibir a foto e nome do seu contato em outras partes da aplicação que você está criando. Seguindo o exemplo do código 14, bastaria duplicar o código. No exemplo do código 15, está sendo simulada a exibição dos dados em partes diferentes.

Código 15 - Exemplo de repetição de código sem uso de funções

```
// Exibir na lista de conversas
console.log(“foto”);
console.log(“João”);

// Exibir no Status
console.log(“foto”);
console.log(“João”);
// Exibir na lista de contatos

console.log(“foto”);
console.log(“João”);
```

Fonte: Elaborada pelo autor (2021).

Nessa etapa, imagine um cenário em que o código precise ser repetido mais vezes. A situação fica mais complexa e você acaba espalhando código repetido por toda a aplicação. Claro, fazer isso seria muito mais simples, pois bastaria usar o CTRL+C e CTRL+V.

Mas duplicar código pode causar bugs no sistema e complicar a manutenção do aplicativo que você está desenvolvendo. Existe uma solução para que você não precise duplicar esse código toda vez que for necessário, e essa solução é utilizar funções.

Para utilizar funções, primeiro você precisa declará-la, criá-la, dando um nome relacionado com a tarefa ou cálculo que ela executa e, posteriormente, quando você precisar utilizar aquele bloco de código, você precisará executar essa função. Você vai ouvir muitos Devs dizendo que vão rodar a função, chamar a função ou invocar a função em vez de dizer que vão executar a função, mas considere como sendo a mesma coisa.

EXEMPLO

Código 16 - Exemplo de funções

```
function exibeDadosDoContato(){
console.log(“foto”);
console.log(“João”);
}
```

No código acima, estamos declarando (criando) uma função com o nome exibeDadosDoContato. Perceba que esse nome está relacionado com a tarefa que a função executa. Teste o código no seu VSCode. Não aconteceu nada, correto? Isso porque é necessário executar a função! Altere o seu teste como no código abaixo:

Código 17 - Exemplo de execução de funções

```
function exibeDadosDoContato(){
        console.log(“foto”);
        console.log(“João”);
}
exibeDadosDoContato();
console.log(“a função já foi executada e os dados exibidos”);
```

Fonte: Elaborada pelo autor (2021).

No código acima, na linha 6, estamos executando a função. Agora, se você testar o código, o resultado será a exibição dos textos na saída do console. Vamos entender melhor o que está acontecendo?

O JavaScript começa a execução do código a partir da linha 1 e vem executando, sequencialmente, linha a linha. Quando o JavaScript encontra a declaração da função, ele armazena o bloco de código na memória e, com o nome dado à função, ele define um atalho para o acesso a esse código. Quando o JavaScript chega na linha 6 e encontra a chamada da função, ele executa o código que está armazenado na memória, os dados são exibidos, a execução continua e o código da linha 8 é executado.

1.7.1 Declaração de funções

Fullturist, até agora você já testou os códigos anteriores e começou a entender como funciona uma função, a sua finalidade e como se declara (cria) uma função. Vamos aprender mais um pouco?

Nessa parte você irá entender pontos importantes da sintaxe de declaração de funções e as formas existentes para isso.

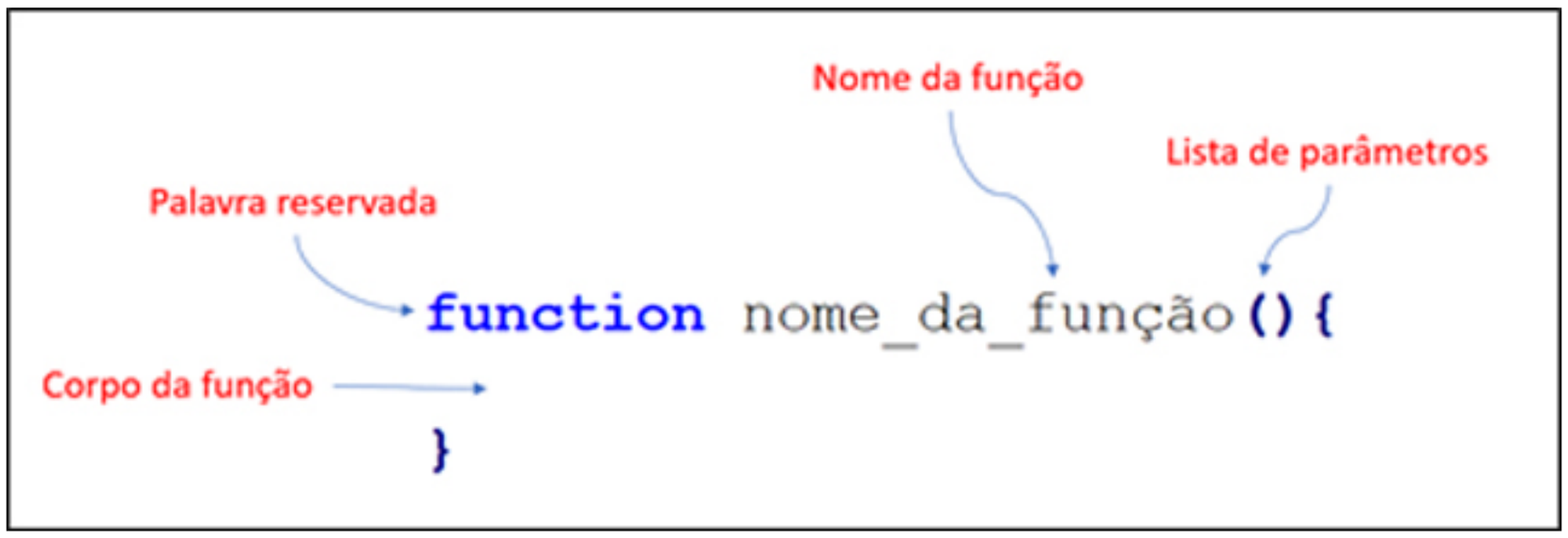
Em JavaScript, existem cinco formas em que você pode declarar uma função, que são:

- Functions declaration - função de declaração de funções apresentadas nos exemplos até agora;
- Functions expression - função de expressões que são armazenadas em uma variável;
- Arrow functions - função de flecha que é uma forma mais moderna e mais simplificada de declaração de funções;
- Functions constructor - função construtora;
- Generator functions - função geradora.

As formas de definições mais utilizadas no mercado são Function declaration, Functions expression e Arrow functions. Vamos estudar um pouco mais sobre essas formas de declaração de função?

1.7.2 Functions declaration

Essa forma de declaração de funções é a mais básica. Todas as funções declaradas assim possuem uma sintaxe completa começando pela palavra-chave function. Em seguida, aparece o nome da função e uma lista de parâmetros (opcional) separados por vírgula e dentro de parênteses. O corpo da função, ou seja, o algoritmo, o código da função fica entre abre e fecha chaves “{}”. A figura a seguir exemplifica a sintaxe de uma forma mais clara:



Fonte: Elaborada pelo autor (2021).

Confira, na sequência, a configuração de declaração na forma Functions expression.

1.7.3 Functions expression

É uma forma de declaração de funções como uma expressão em que atribuímos essa função a uma variável. Para executar essa função basta utilizar a variável em conjunto com o abre e fecha parênteses “()”. Essa forma de declaração de funções também é chamada de funções anônimas, pois não possuem um nome.

Para ficar mais clara a definição acima, veja a seguir:

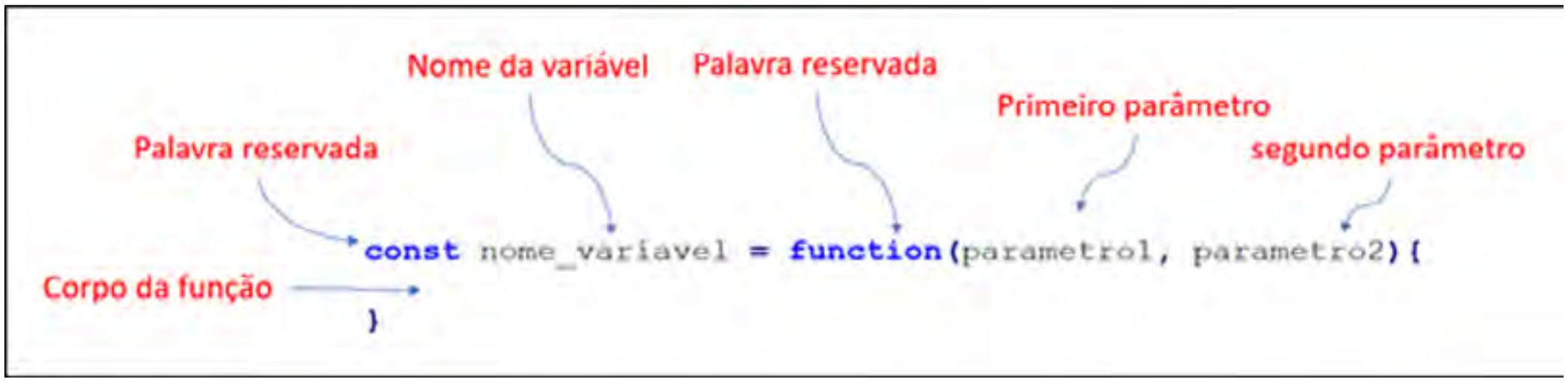
EXEMPLO

Código 18 - Exemplo de função declarada no formato de Functions expression

```
const exibeMensagens = function(){
console.log(“Seja bem-vindo a trilha JavaScript!”);
console.log(“JavaScript é a linguagem mais utilizada na web!”);
};
exibeMensagens();
```

Fonte: Elaborada pelo autor (2021).

No código acima, na linha 1, uma função é declarada e atribuída à constante exibeMensagem e, para executar essa função, é utilizado o nome da variável junto com os parênteses “()”, conforme pode ser visto na linha 5. Dois pontos importantes que você deve se atentar nessa forma de declaração de funções: primeiro, a função não possui um nome, como você pode observar na linha 1 – ela é atribuída a uma variável e você deve utilizar a variável para executar a função; segundo, ao final da função, depois do fecha chave, você deve colocar ponto e vírgula (veja na linha 4). Na imagem abaixo, é exibida a sintaxe completa da declaração de funções por expressão. Não se esqueça de testar o código no VSCode para reforçar o aprendizado!



Fonte: Elaborada pelo autor (2021).

Você deve estar curioso(a) para entender o motivo de atribuir uma função a uma variável, não está?

No momento certo, mais à frente no curso, vamos estudar mais sobre isso.

1.7.4 Arrow Functions

É outra forma de declaração de funções muito utilizada e que provavelmente você vai se deparar e utilizar em vários códigos durante a sua vida profissional. A declaração de funções na forma de Arrow Functions é uma maneira simplificada e resumida de declarar funções em JavaScript.

Arrow Function foi adicionado nas versões mais atuais do JavaScript conhecida como ECMAScript 6 ou ES6. As arrow functions são sempre funções anônimas e esse é um dos motivos pelos quais elas são muito utilizadas como funções de callback. De forma simples, uma função callback é uma função passada como argumento para outra. Para você entender melhor, um bom exemplo é quando você manipula ações de clique.

No código abaixo, na função addEventListener, o primeiro parâmetro é o evento que está sendo manipulado e, o segundo, é uma função de callback que será executada quando o evento de clique for acionado.

Código 19 – Exemplo de utilização de functions Exepression como parâmetro.

```
botao.addEventListener('click', function(){
});
```

As arrow functions são simplificações para as funções declaradas como functions expression, como pode ser visto a seguir.

EXEMPLO

Código 20 – Comparativo entre declaração de função functions expression x arrow functions

(a) Função declarada na forma de function expression.

```
1. const exibeMensagens = function(){
2.         console.log("Seja bem-vindo!");
3. };
4. exibeMensagens();
```

(b) Função declarada na forma de arrow functions

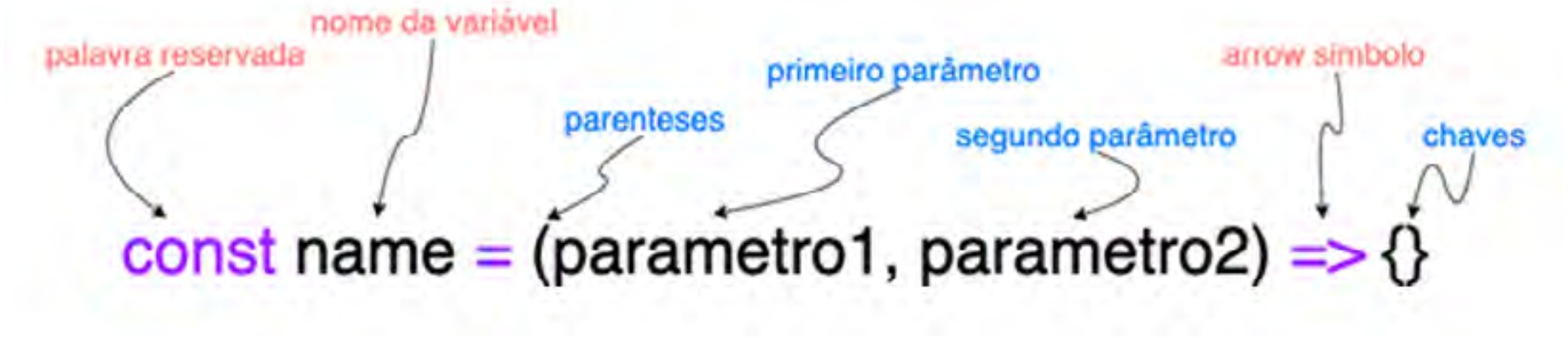
```
1. const exibeMensagens = () => {
2.         console.log("Seja bem-3. vindo!");
3. };
4. exibeMensagens();
```

Fonte: Elaborada pelo autor (2021).

No exemplo acima, no código (a) é declarada uma função na forma de function expression e, no código (b), é declarada uma função na forma de arrow function. Perceba na linha 1 que, em uma arrow function, a palavra reservada "function" é substituída pelo sinal "=>" precedido por abre e fecha parênteses "()", onde é declarada a lista de parâmetros separados por vírgula. Arrow function tem uma particularidade em relação às demais formas de declaração de funções: não possui o parâmetro this como as demais funções; elas assumem o this do contexto em que elas foram criadas/definidas. Para facilitar o entendimento, vamos voltar ao exemplo de manipulação do evento de um botão. Quando você manipula o evento clique de

um botão por meio da função addEventListener(), a função de callback assume o this da função addEventListener(), ou seja, this é o próprio botão. Implemente e teste o código (b) no VSCode para aprender ainda mais.

A imagem abaixo ilustra a sintaxe completa de declaração de funções como arrow functions:



Fonte: Elaborada pelo autor (2021).

1.8 Argumentos e parâmetros de uma função

O grande ponto agora é você entender como usar uma função para executar o código sobre valores diferentes. Para você entender melhor, analise o código abaixo com a implementação de uma função que executa a soma de dois números:

Código 21 - Código com funções sem uso de parâmetros e argumentos

```
function soma(){
        let valor1 = 10;
        let valor2 = 5;
        let soma = valor1 + valor2;
        console.log("O resultado da soma é: ", soma);
}
soma();
```

Fonte: Elaborada pelo autor (2021).

Implemente o código no VSCode e teste para fixar o conhecimento e verificar o resultado apresentado no console. Executou? O resultado exibido no console foi 15, correto? Vamos entender o que está acontecendo na execução do código. O JavaScript começa a ler o script acima e, logo na linha 1, ele encontra a declaração de uma função. O JavaScript, então, armazena o código na memória e cria um atalho para ser usado quando for necessária a execução deste código. Este atalho é o nome da função, lembra? Quando o JavaScript lê a linha 8, ele encontra o código que invoca, chama, executa a função soma. O que está sendo executado na função soma? Primeiramente são criadas duas variáveis chamadas valor 1, que armazena o número 10 (linha 2), e valor2, que armazena o número 5 (linha 3). Na linha 4 está sendo realizada a soma com os números armazenados nas variáveis valor1 e valor2, e o valor resultante dessa soma está sendo armazenado na variável soma. Para finalizar, na linha 5 está sendo exibida no console a mensagem com o valor armazenado na variável soma: “O resultado da soma é: 15”. Resumindo, a função realiza a soma de dois números e exibe no console o resultado dessa soma. O ponto agora é, e se você precisar realizar a soma com dois números diferentes? Neste caso, para que o código da função seja realmente repetido para valores diferentes, você tem que utilizar parâmetros na função.

Vamos a um exemplo para facilitar o entendimento:

EXEMPLO

Código 22 - Exemplo de função declarada com parâmetros e execução com argumentos

```
function soma(valor1, valor2){
let soma = valor1 + valor2;
console.log(“O resultado da soma é: “, soma);
}
soma(2, 5);
soma(3, 4);
soma(10, 10);
```

Fonte: Elaborada pelo autor (2021).

O que mudou no código?
Agora a função soma pode ser utilizada para realizar adição com números diferentes e, para que isso aconteça, estão sendo usados parâmetros e argumentos. Ou seja, agora o código da função soma pode ser repetido para valores diferentes. Vamos entender melhor. Na linha 1, na declaração da função, estamos criando dois parâmetros: valor1 e valor2. Parâmetros é uma outra nomenclatura para variáveis. Usamos a nomenclatura "parâmetro" no contexto de funções e para diferenciar das variáveis comuns declaradas no corpo da função, como por exemplo, a variável soma declarada na linha 2. Perceba que, para se declarar um parâmetro, ele deve estar dentro dos parênteses junto ao nome da função e não necessita do uso de let, var ou const. Continuando, na linha 6 o JavaScript executa a função soma e passa como argumentos dos parâmetros os valores 2 e 5. Nesse caso, o código da função soma será executado com os valores 2 e 5. Argumentos é uma nomenclatura usada para os valores que são passados para a função e armazenados nos parâmetros.
No caso do nosso exemplo, na linha 6 o argumento 2 está sendo passado para o parâmetro valor1, e o argumento 5 está sendo passado para o parâmetro valor2. Em outras palavras, o número 2 está sendo armazenado, atribuído ao parâmetro valor1, e o número 5 está sendo armazenado, atribuído ao parâmetro valor2. Teste o código e veja o resultado exibido para as chamadas da função soma nas linhas 6, linha 8 e linha 10.

1.9 Retornando valores dentro da função

Nos exemplos acima, onde está sendo implementada a função soma, no corpo desta função está sendo executada a soma de dois números, e o resultado dessa soma é exibido no console. Existem muitos casos e é muito comum que o resultado da execução da função seja aproveitado no restante da execução do script. Por exemplo, imagine que você precise utilizar, no restante do script, o resultado da soma de dois números executado pela função soma. Para que isso seja possível, a função tem que retornar o resultado calculado. Analise o código abaixo para você entender melhor. Não se esqueça de implementar e testar o código no VSCode.

Código 23 - Exemplo de código com retorno da função

```
function soma( valor, valor ){
        Let soma = valor1 + valor2;
        return soma;
}
let resultado = soma(2, 2);
console.log("O resultado foi: ", resultado);
```

Fonte: Elaborada pelo autor (2021).

Ao ler o código na linha 8, o JavaScript executa a função soma passando o argumento 2 para o parâmetro valor1 e o segundo argumento 2 para o parâmetro valor 2. Na linha 3 é executada a soma dos dois valores armazenados nos parâmetros e o resultado dessa soma é atribuído, armazenado na variável soma. A grande mudança acontece na linha 5: a novidade aqui é a palavra reservada return. Na linha 5, o valor do cálculo armazenado na variável soma é retornado para o ponto onde a função foi executa, retornado para a linha 8. O que acontece na linha 8 é que, após a execução da função, depois que a função retorna ao valor da soma, este valor retornado é atribuído, armazenado na variável resultado. Agora podemos utilizar o valor no restante do script. Neste caso, como é um exemplo simples, o valor retornado e armazenado na variável resultado está sendo apenas exibido no console na linha 10. No dia a dia de um programador JavaScript é muito utilizado o retorno das funções.

1.10 Arrays

Uma das principais funções das linguagens de programação é lidar com dados. Com JavaScript não seria diferente. Até agora você viu e testou uma série de exemplos em que foram manipulados dados com variáveis. Vamos ver mais um? No código abaixo será armazenado números de 10 a 15 em diferentes variáveis e, depois, esses dados serão manipulados executando uma soma.

Código 24 - Exemplo de código sem a utilização de Arrays

```
let valor = 10;
let valor2 = 11;
let valor3 = 12;
let valor4 = 13;
let valor5 = 14;
let valor6 = 15;
let soma = valor + valor2 + valor3 + valor4 + valor5 + valor6;
console.log(“Resultado: ”, soma);
```

Fonte: Elaborada pelo autor (2021).

O código acima é bem simples. Na linha 1 é declarada uma variável com o nome de valor e o número 10 é atribuído (armazenado) a ela, assim como ocorre para as variáveis valor2, valor3 e demais variáveis. Na linha 8 é calculada a soma com os números armazenado nas variáveis e o resultado é atribuído à variável soma. O valor armazenado na variável soma é exibido no console na linha 10. Tranquilo, não é? Agora, imagine o seguinte: você precisa adicionar à soma o número 16. Então, você provavelmente criaria uma nova variável com o nome valor7, atribuiria o número 16 a essa variável e depois adicionaria essa variável à soma, correto? Tranquilo também, não é? Então imagine um novo cenário, onde você precisará adicionar à soma os números 17, 18, 19, 20, 21, 22, 23 e 24. Começou a complicar, certo? Começou a ficar trabalhoso! Para piorar, em um software, quem informa os valores a serem somados e a sua quantidade os são as pessoas que utilizam o software. Neste caso, como você saberia a quantidade de variáveis que você deveria criar? E os nomes dessas variáveis? Complicou mais um pouco, não complicou? Para resolver esse problema e facilitar a sua vida, o JavaScript permite o uso de Arrays.

Um array é um tipo variável especial que pode conter mais de um valor por vez. Isso mesmo! Um array pode armazenar vários valores com um único nome de variável, é como uma espécie de lista. Para acessar os valores armazenados nesta variável especial utilizamos índices.

Como sempre, um bom exemplo vale mais que mil palavras!

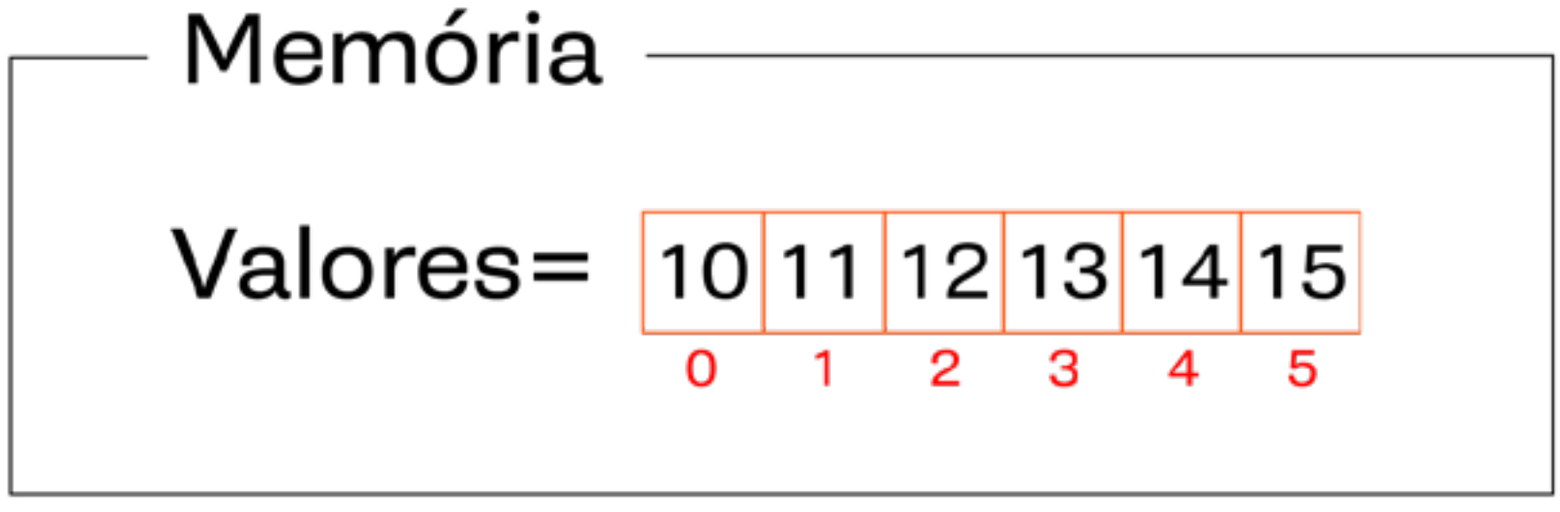
Código 25 - Código 24 alterado com utilização de arrays

```
let valores = [10, 11, 12, 13, 14, 15];
let soma = valores[0] + valores[1] + valores[2] + valores[3] + valores[4] +
valores[5] + valores[6];

console.log("Resultado: ", soma);
```

Fonte: Elaborada pelo autor (2021).

O que está acontecendo no código acima? Na linha 1, está sendo declarado um array com o nome de valores e está sendo atribuída a ele uma lista de valores começando de 10 até 15. Ou seja, essa variável especial, o array valores, está armazenando todos esses números! Na linha 4, para realizar a soma de todos esses números, cada valor está sendo acessado individualmente pelo seu índice. A imagem abaixo ajudará você a entender melhor como um array armazena esses valores e como funcionam esses índices.



Fonte: Elaborada pelo autor (2021).

Com o auxílio da imagem acima, analise o array valores como se fosse uma caixa composta por pequenas divisões ou caixinhas internas. Cada divisão guarda separadamente o seu valor. Para identificar essas divisões e acessar o seu valor, cada divisão possui um rótulo que são os índices representados na imagem pelos números em vermelho. Os índices de um array são criados automaticamente pelo JavaScript conforme ele cria as divisões. Os índices sempre iniciam a partir do valor 0.

No código acima foi utilizado um array para resolver o problema da utilização de várias variáveis. Mas ainda continuamos com o problema de inclusão de novos valores. Array é um objeto e possui uma série de funções prontas para facilitar a sua vida de programador. Algumas dessas funções permite que você adicione novos valores no array sem se preocupar em como isso é feito. A função push() é utilizada para adicionar um elemento (novo valor) ao final de um array. Vamos alterar o código do exemplo anterior criando uma função com o nome de soma. Esta função irá somar os valores existentes dentro array utilizando a estrutura de repetição FOR. No exemplo também será utilizado a função push() para adicionar novos valores ao array.

Código 26 - Manipulação de arrays com estrutura de repetição FOR

```
function soma(arrayValores){
        let resultado = 0;
        for(let i = 0; i < arrayValores.length(); i++){
                resultado = resultado + arrayValores[i];
        }
        return resultado;
}
const valores = [10, 11, 12, 13, 14];
let res = soma(valores);
console.log("Resultado da soma é:", res);
valores.push(15);
res = soma(valores);
console.log("Resultado da soma é:", res);
```

Fonte: Elaborada pelo autor (2021).

Implemente o código no VSCode, teste e veja os resultados exibidos no console. Para ajudar você a entender melhor, segue a explicação do código. Na linha 1 foi declarada uma função denominada soma com um parâmetro de nome arrayValores. Como arrays são variáveis especiais, eles podem ser passados como argumento para uma função assim como uma variável comum. Na linha 3, foi declarada uma variável com nome resultado e seu valor foi inicializado com zero. Essa variável é um acumulador que irá guardar a soma interativa dos valores do array.

Queremos chamar a sua atenção, neste momento, para a declaração da estrutura de repetição for. O for irá auxiliar no acesso iterativo dos valores armazenados no array através do índice de cada elemento. Para o for executar corretamente e repetir o código Na quantidade de vezes necessária para acessar todos os valores do array, ele precisa conhecer o tamanho deste array. O tamanho de um array é a quantidade de valores nele armazenados, é a quantidade de caixinhas, de divisões existentes. A função length() é responsável por retornar o tamanho de um array. Na linha 5 foi

utilizada a função length() do arrayValores como condição de parada do for, ou seja, o for vai repetir o código até acessar todo os valores do array. A variável i é o contador do for e também vai servir como o valor do índice de acesso as posições do array como foi feito na linha 7. Ao final da execução do for, todos os valores foram somados e armazenados na variável resultado, e ao final do código, na linha 11, esse valor é retornado. Continuando a execução, na linha 13 o JavaScript encontra a declaração do array Valores inicializados com os números 10, 11, 12, 13 e 14. Neste momento, o array possui 5 caixas e cada uma armazena o seu respectivo valor. Na linha 15 a função soma é chamada e, para sua execução, o array Valores é passado como argumento. O resultado retornado pela função soma é armazenado na variável res e logo em seguida, na linha 17 o seu valor é exibido no console. Na linha 19, um novo valor é adicionado ao final do array utilizando a função push(). Agora o array possui um novo elemento e um novo tamanho.

1.11 Função map(), reduce() e filter()

No código 24, apresentado anteriormente, para calcular a soma dos valores armazenados no vetor foi utilizada a construção de uma função. No corpo desta função foi utilizada a estrutura de repetição for para percorrer e somar todos os valores. Existe uma maneira mais simples e fácil para programar ações como a soma dos valores do vetor. Essa maneira é utilizar a função reduce() disponibilizada pelo objeto array. A ideia da função reduce() é produzir um único valor a partir de um array. Para ficar mais claro, analise, implemente e teste o código de exemplo abaixo. Neste exemplo estamos reescrevendo o exemplo do código anterior sem utilizar funções e a estrutura de repetição for.

Código 27 - Exemplo de manipulação de array com a função reduce()

```
const valores = [10, 11, 12, 13, 14];
let res = valores.reduce(  (resultado, x)  =>  {
        console.log(`${resultado }+${x} = ${ resultado +x}`);
        return resultado + x;
});
console.log("Resultado da soma é:", res);
```

Fonte: Elaborada pelo autor (2021).

A função reduce() recebe um parâmetro que é outra função que irá executar a soma dos elementos. Na linha 3, como argumento da função reduce(), está sendo passada uma arrow function. Esta arrow function possui dois parâmetros: o primeiro é um contador de nome resultado que irá acumular a soma dos números e, o segundo, de nome x, armazena temporariamente valor a valor armazenado no array. A função reduce() executa a arrow function passando o valor acumulado e o próximo item do array.

Agora, imagine que você precise duplicar um a um os valores armazenados em um array? Como você implementaria o código para essa solução? Os objetos array também disponibilizam a função map() que resolve este problema de uma forma simples e fácil. A função map() permite que você execute uma transformação para cada valor armazenado em seu array, gerando um novo array como resultado. No exemplo abaixo está sendo gerado um novo array e atribuído à variável res.

Código 28 - Exemplo de manipulação de array com a função map()

```
const valores = [10, 11, 12, 13, 14];
let res = valores.map( function(num) {
        console.log(`${num } x ${2} = ${ num * 2}`);
        return num * 2;
});
console.log("Resultado da soma é:", res);
```

Fonte: Elaborada pelo autor (2021).

Implemente, execute, veja o resultado e analise o código acima para entender melhor o seu funcionamento.

A função map() funciona da mesma forma que a função reduce(). O que difere é que a função map() retorna um array com as modificações realizadas nos itens, enquanto a função reduce() retorna apenas um valor.

1.12 Promises

Promises (Promessa) é um recurso da linguagem JavaScript muito utilizado no desenvolvimento de aplicações web e aplicativos para smartphones.

Promise é um objeto que utilizamos para executar processamentos assíncronos. Esse objeto armazena um valor resultante de um processamento que poderá estar disponível agora, no futuro ou nunca. Dessa forma, podemos fazer o tratamento de eventos que acontecem em caso de sucesso ou falha de forma assíncrona.

Neste momento, imaginamos que você esteja com uma interrogação enorme em sua mente: o que é tudo isso? Não entendi nada!

Bom, vamos explicar alguns conceitos para você conseguir compreender melhor o que é Promises.

O primeiro passo é entender o que é um processamento assíncrono.

1.12.1 Processamento assíncrono em JavaScript

Todo mundo sabe que o JavaScript sempre foi síncrono, ou seja, ele sempre executava os comandos, as funções, os blocos de códigos em sequência, um após o outro. Dessa forma, não era possível você executar os códigos de forma assíncrona. Não era até lançarem as novas versões do JavaScript! Essa nova versão criou as Promises para ajudar a resolver esse problema de execução assíncrona. Em uma execução de código assíncrona, um comando, uma função ou um bloco de código não precisa esperar que outro seja finalizado para ele ser executado.

A grande sacada das Promises é que permitem executar duas partes do seu código ao mesmo tempo, por exemplo.

Para ficar mais claro como funciona uma Promise e o processamento de código assíncrono, vamos conferir um exemplo. Este exemplo é bem básico, mas servirá para que você possa compreender melhor esse funcionamento. Primeiramente, o exemplo vai ser criado para ser executado de forma síncrona e, posteriormente, vamos alterá-lo para a execução assíncrona, utilizando Promise.

1.12.2 Processamento síncrono

No exemplo de código a seguir está sendo programada uma função que calcula uma soma, duas funções de callback (um tipo de função que só é executada após o processamento de outra função) e uma exibição de uma mensagem no console. Vamos analisar primeiro o código e deixemos as explicações para a sequência. Utilize o VSCode para testar o código abaixo:

Código 29 – Código com processamento síncrono

```
function somaNumeros(){
    let resultado = 1+1;
    if (resultado == 2){
        sucessoCallBack();
    }else{
        errorCallBack();
    }
}

function sucessoCallBack(){
    console.log("Sucesso!!!!!");
}
function errorCallBack(){
    console.log("Opa!!! Algo deu errado :( ");
}
somaNumeros();
console.log("Código executado após a execução da função somaNumeros");
```

Fonte: Elaborada pelo autor (2021).

Vamos entender o código acima. Na linha 1 está sendo declarada uma função que realiza a soma de dois números. Neste caso, os valores da soma estão fixos, como pode ser observado na linha 2. Na linha 11 é declarada uma função de nome sucessoCallBack() e, na linha 15, outra função denominada errorCallBack(). A execução da função somaNumeros() está acontecendo na linha 19, sendo que a partir desta linha você deve ter mais atenção. Quando a função somaNumeros() é executada, é realizada a soma de dois números 1 e o resultado 2 é armazenado na variável resultado (linha 2). Então, na estrutura de decisão if esse resultado é comparado (linha 4), e se a comparação for positiva, a função sucessoCallBack() é executada, exibindo no console a mensagem "Sucesso!!!!!". A execução da função

somaNumeros() é encerrada e a da linha 21 “Código executado após a execução da função somaNumeros” é exibida também no console. A execução do código acima é síncrona. Isso quer dizer que a execução ocorreu linha a linha, uma linha executada após o término da execução da outra. Se você testou o código, verificou que, no console, primeiro foi exibida a mensagem “Sucesso!!!!!” e, logo depois, a mensagem “Código executado após a execução da função somaNumeros”.

Além de demonstrar uma execução síncrona, o exemplo também apresenta de forma didática o funcionamento de uma Promise. Uma Promise executa um código contido dentro dela e o resultado desse processamento pode obter sucesso ou não, assim como o código que foi programado na função somaNumeros(). Agora vamos alterar o código utilizando uma Promise.

1.12.3 Processamento Assíncrono

Como dito anteriormente, em uma execução de código assíncrona, um comando, uma função ou um bloco de código não precisa esperar outro finalizar para ele ser executado. No VSCode, teste o código abaixo e analise o resultado da utilização de Promises.

Código 30 – Código com processamento assíncrono utilizando Promises

```
let p = new Promise( (resolve, reject) => {
   let resultado = 1 + 1;
   if( resultado == 2){
      resolve(“sucesso!!!”);
   }else{
      reject(“Opa!!! Algo deu errado :( “);
   }
});
```

```
p.then( (message) =>{
console.log("Mesangem no then " + message);
}).catch( (err) => {
console.log("Mesangem no catch " + err);

});
console.log("Código executado após a execução da função somaNumeros");
```

Fonte: Elaborada pelo autor (2021).

Bom, para começar, você precisa lembrar que a Promise é um objeto e, para utilizá-lo, precisamos instanciar este objeto utilizando o comando new (linha 1). Não se preocupe se você não entendeu o que é um objeto, você ainda estudará muito sobre isso no decorrer no curso. Para criar um objeto Promise, você tem que passar como parâmetro uma função de callback que contém o código que irá ser processado quando você utilizar essa Promise. Esta função é uma função anônima que recebe dois parâmetros: o resolve e o reject. Vamos utilizar, nesse caso, uma notação arrow functions. Quando eu tenho o resultado positivo (linha 2), está sendo passado para o callback resolve (linha 4) a mensagem "Sucesso!!"; e quando eu tenho um resultado que falha, está sendo passada a mensagem "Opa!!! Algo deu errado :(" para um callback reject (linha 6). Na linha 10, a nossa promise criada e armazenada na variável p é executada. Para executar essa promise é chamada a função/método then(). O método then() irá executar se o resultado do código da promise obtiver sucesso, ou seja, a comparação na linha 3 for verdadeira. Então, será exibida no console a mensagem de sucesso (linha 11). Agora, se o resultado não for positivo, o método catch() na linha 12 será executado e uma mensagem de erro será exibida no console (linha 13). Se você testou e executou este código, você percebeu que a mensagem da linha 16 foi exibida no console antes da mensagem da promise. Isso porque o código da linha 16 não precisou esperar a promise finalizar a execução para ser executado. Os dois foram executados ao mesmo tempo: isso é execução assíncrona.

Esse é o princípio básico das Promises.

Acessando o site da MDN você poderá aprender mais sobre Promise.

1.12.4 Por que utilizar Promise?

Durante a sua carreira como desenvolvedor, você irá programar aplicativos que acessam dados disponíveis na internet. Para acessar esses dados, o aplicativo deverá realizar uma conexão. Se essa conexão falhar por algum motivo, ou o acesso ao dado levar um certo tempo, o aplicativo não poderá travar. Para evitar o travamento do aplicativo, essa execução é feita através de Promises. Outro exemplo são os navegadores de internet: você não precisa esperar todo o conteúdo de uma página ser exibido para navegar por ela.

Até aqui você já aprendeu várias coisas importantes sobre o JavaScript e absorveu conhecimentos que, apesar de básicos, são de extrema importância para a sua evolução durante o curso. Você aprendeu o que é e para que serve o JavaScript, estudou sobre a utilização de variáveis para armazenar dados que serão utilizados no processamento e viu que existem estruturas que se auxiliam na tomada de decisão e na execução de código. Você aprendeu, ainda, o que são funções e como organizar o seu código, aprendeu sobre objetos arrays e que os códigos JavaScript podem ser executados de forma assíncrona.

Mas você está pensando que acabou? Nada! Ainda tem muita coisa interessante para você aprender, como manipular dinamicamente os conteúdos existentes no arquivo HTML, adicionar eventos a esses conteúdos, programar formulários e até mesmo salvar dados no seu navegador. Ficou curioso(a)? Aposto que sim! Então, nos vemos na próxima unidade!

Síntese

Nesta unidade, você estudou e aprendeu os principais conceitos sobre a linguagem JavaScript. Confira a seguir os itens mais importantes!

- Vimos o que é JavaScript e a finalidade de utilizarmos essa linguagem de programação.

- Você aprendeu que a utilização de variáveis é um recurso para armazenar dados que serão utilizados no processamento.

- Constatamos que, utilizando as estruturas de decisão, é possível programar execuções diferentes dependendo do resultado de um processamento.

- Você aprendeu que, executar um mesmo bloco de código repetidamente através das estruturas de repetição permite que você não necessite duplicar código.

- Vimos que as funções são recursos que permitem criar e organizar blocos de códigos que podem ser executados/chamados em diferentes pontos do seu código, e permite que você reaproveite-os sem a necessidade de duplicidade.

- Você aprendeu que as funções possuem formas diferentes de serem declaradas, cada uma com suas particularidades específicas.

- Constatamos que os arrays são variáveis especiais, semelhantes a uma lista de dados, que permitem a manipulação desses dados sem a necessidade de criação de múltiplas variáveis. Além disso, os arrays possuem uma série de métodos, funções que agilizam o processo de desenvolvimento.

- Vimos o que é execução de código assíncrono e que JavaScript permite essa execução de forma assíncrona através das Promises, e que a grande sacada das promises é a possibilidade de executar mais de um bloco de código ao mesmo tempo.

Fullture Insights

- GUIA JavaScript – Sintaxe e tipos. MDN Web Docs, [s. l.], 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 10 out. 2021.

- VISUAL Studio Code. Disponível em: https://code.visualstudio.com. Acesso em: 10 out. 2021.

FU
LL
TU
RE

www.fullture.com