JavaScript Básico

Carlos Dias da Silva Júnior

Formação Inicial e Continuada

+IFMG

Carlos Dias da Silva Júnior

JavaScript Básico

1ª Edição

Belo Horizonte

Instituto Federal de Minas Gerais

2022

Pró-reitor de Extensão	Carlos Bernardes Rosa Júnior
Diretor de Projetos de Extensão	Niltom Vieira Junior
Coordenação do curso	Carlos Dias da Silva Júnior
Arte gráfica	Ângela Bacon
Diagramação	Eduardo dos Santos Oliveira

S586j Silva Júnior, Carlos Dias da.

JavaScript básico [recurso eletrônico] / Carlos Dias da Silva Júnior. – Belo Horizonte : Instituto Federal de Minas Gerais, 2022.

52 p. : il. color.

Material didático para Formação Inicial e Continuada.

ISBN 978-65-5876-040-5

1. JavaScript (Linguagem de programação de computador). 2. WEB (Linguagem de programação). I. Título.

CDD 005.133
CDU 004.438

Catalogação: Viviane Barbosa Andrade - Bibliotecária - CRB-6/2819

Índice para catálogo sistemático:

1. Programação 005.133

2022

Direitos exclusivos cedidos ao
Instituto Federal de Minas Gerais
Avenida Mário Werneck, 2590,
CEP: 30575-180, Buritis, Belo Horizonte – MG,
Telefone: (31) 2513-5157

Sobre o material

Este curso é autoexplicativo e não possui tutoria. O material didático, incluindo suas videoaulas, foi projetado para que você consiga evoluir de forma autônoma e suficiente.

Caso opte por imprimir este *e-book*, você não perderá a possiblidade de acessar os materiais multimídia e complementares. Os *links* podem ser acessados usando o seu celular, por meio do glossário de Códigos QR disponível no fim deste livro.

Embora o material passe por revisão, somos gratos em receber suas sugestões para possíveis correções (erros ortográficos, conceituais, *links* inativos etc.). A sua participação é muito importante para a nossa constante melhoria. Acesse, a qualquer momento, o Formulário “Sugestões para Correção do Material Didático” clicando nesse *link* ou acessando o QR *Code* a seguir:

Para saber mais sobre a Plataforma +IFMG acesse

mais.ifmg.edu.br

Palavra do autor

A linguagem JavaScript é uma das mais utilizadas atualmente sendo, entre as linguagens disponíveis, a mais cogitada para o desenvolvimento web. Além disso, ela se faz presente em diversas aplicações industriais, no mercado de trabalho e até mesmo no ambiente acadêmico, sendo, portanto extremamente proveitosa sua compreensão para aperfeiçoamento dos conhecimentos de programação e dinamicidade dos códigos e interações cliente-servidor.

Desta forma, por meio desta apostila buscou-se desenvolver os conhecimentos de JavaScript de uma forma dinâmica e de fácil entendimento, trazendo na primeira parte da apostila a introdução da linguagem, sua história e o seu ambiente de programação.

Ademais, na segunda parte da apostila, os conteúdos são voltados para o contato do aluno com a linguagem JavaScript, a partir de códigos básicos para entendimento da sintaxe, criação de variáveis, as possibilidades e condições para executar código e repetições de código, tendo como foco trabalho com objetos dinâmicos e manipulação de dados.

Desejo que vocês tenham sucesso em sua jornada, não somente no curso de JavaScript, mas como futuros desenvolvedores!

O autor.

Apresentação do curso

Este curso está dividido em quatro semanas, cujos objetivos de cada uma são apresentados, sucintamente, a seguir.

SEMANA 1	Tem como objetivo introduzir a linguagem de programação JavaScript. Serão apresentados os conceitos gerais da linguagem, sua história e preparar o ambiente de programação.
SEMANA 2	Com os primeiros testes de códigos e exibindo mensagens ao usuário, nesta semana aprenderá o básico da sintaxe de JS, criação e tipos de variáveis.
SEMANA 3	Aprenda a trabalhar com condições para executar código e repetições de código em JavaScript
SEMANA 4	Com foco em tratar dados e objetos em JS, nesta semana entenderá sobre funções e como usar alguns recursos preparando para o avançado!

Carga horária: 40 horas.

Estudo proposto: 5 horas (1 hora por dia, 5 dias na semana) na primeira e segunda semana, e 15h (de 2 a 3 horas por dia, 5 dias na semana) na terceira e quarta semana.

Sumário

Semana 1 – Conceitos gerais da linguagem

Objetivos

Tem como objetivo introduzir a linguagem de programação JavaScript. Serão apresentados os conceitos gerais da linguagem, sua história e preparar o ambiente de programação.

Mídia digital: Os passos descritos aqui estão disponíveis na vídeo aula “Preparando o ambiente”. Se estiver com dúvida, talvez lá fique mais claro!

1.1 Introdução e histórico da evolução de JavaScript;

Criada pela empresa Netscape Communications Corporation, a linguagem JavaScript (JS) foi primeiramente pensada como uma forma de manipular elementos no HTML a fim de tornar as páginas web “vivas”. Desde sua criação a linguagem JS variou em diversas nomenclaturas, sendo sua primeira designação “Mocha”, seguida de “LiveScript”, entretanto, pouco tempo depois optou-se pelo termo JavaScript, devido a popularidade atribuída a sintaxe Java na época. Porém, apesar de contribuir para a nomeação deste linguagem, as linguagens “Java” e “JavaScript” não possuem semelhanças, visto que, a medida em que o JavaScript evoluiu, evoluíram também suas próprias especificações conhecidas como ECMAScript (European Computer Manufacturers Association), que são independentes e não possuem nenhuma relação com a linguagem de programação Java.

A linguagem em JavaScript é vista por muitos como uma das mais dinâmicas e de propósito geral dentre as linguagens interpretadas, sendo considerada portanto como "multiparadigmas". Tal fator, se deve a sua simplicidade e capacidade de realizar diversificadas aplicações, visto que, a partir dela se pode escrever programas diretamente em uma página da web, conhecidos como scripts, que são executados como texto simples sem preocupações como a indentação durante a escrita do código, ou até mesmo com a compatibilidade com o ambiente de produção. Além disso, uma de suas mais notáveis características é a possibilidade de editar um ambiente à medida em que se o utiliza, criando processamento de dados a partir da criação de pequenos programas capazes de processar dados, promovendo interatividade dinâmica entre sites à medida que se utiliza.

Basicamente, com o JavaScript é possível criar aplicações para os “dois lados”: tanto para o servidor quanto para o cliente. Para as aplicações Client-side ou lado do cliente, se tem o usuário acessando um site por meio do navegador, enquanto que o Server-side ou lado servidor, se tem o software em JS sendo executado diretamente no computador ou servidor, sem necessidade de navegador da web.

Isto só é possível graças aos chamados “engine”, ou seja, os interpretadores, responsáveis por executar o código em JS. Dependendo do local em que está sendo executado, no servidor ou no cliente, o interpretador pode dar ao desenvolvedor diferentes ferramentas, como acesso a arquivos locais, por exemplo. Um exemplo de engines ou interpretador utilizado em JS enquanto navega-se com o Chrome, é o V8 foi desenvolvido em linguagem C + + e é responsável por aumentar a performance de uma aplicação compilando o código Javascript para o formato nativo de máquina antes de executá-lo, possibilitando a execução com velocidade de um código binário compilado, outro engine amplamente utilizado é o SpiderMonkey do Firefox, também desenvolvido em linguagem C++.

Atualmente existem diversas bibliotecas, ou seja, conjuntos de códigos com funções e objetivos específicos, que facilitam e aceleram o processo de desenvolvimento de softwares em JS. Essas bibliotecas oferecem, desde recursos para simplificar o processo de construção de interfaces visuais até mesmo para trabalhar com banco de dados, quando executado do lado do servidor.

Neste curso, você aprenderá os conceitos básicos de programação em JavaScript, sem utilizar bibliotecas, apenas o que a linguagem oferece. O objetivo é entender como se pode criar softwares simples baseados na web de forma dinâmica.

1.2 Preparação do ambiente de programação;

Como explicado anteriormente, o JavaScript é uma linguagem interpretada, o que permite versatilidade, necessitando apenas de um interpretador. Todos os navegadores modernos têm os seus interpretadores para JS, que nos permite acessar páginas interativas, ver animações e dinamizar a navegação na web. Enquanto usuário de navegadores como o Google Chrome ou do Firefox, você pode acessar as chamadas “Ferramentas do desenvolvedor”, que oferecem, entre outros recursos, a possibilidade de executar códigos diretamente na página que está navegando. Por exemplo, no Google Chrome, pressione a tecla F12 (ou Cmd+Opt+J se estiver no Mac) com qualquer página aberta e uma janela com será aberta com uma aba “Console”, conforme figura a seguir.

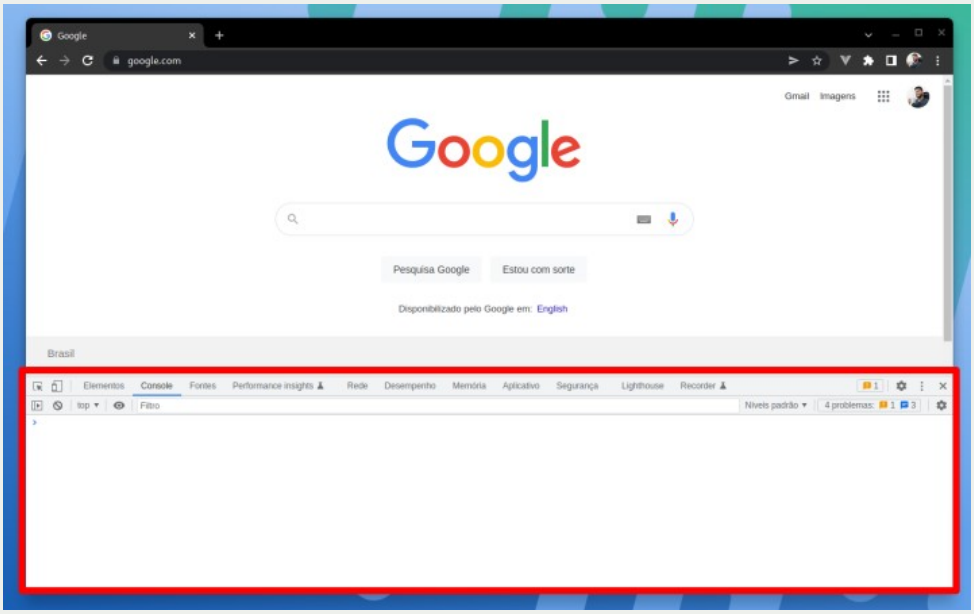


Figura 1 – Navegador Google Chrome com destaque para as “Ferramentas do desenvolvedor”.
Fonte: O próprio autor.

O mesmo procedimento pode ser feito em outros navegadores. No caso do Firefox, na imagem abaixo, pode ser que apareçam algumas mensagens que se parecem com erros. Isto acontece devido ao fato de que cada navegador tem suas próprias políticas de tratamento de erros e de engine JavaScript. Porém, os comandos serão os mesmos.

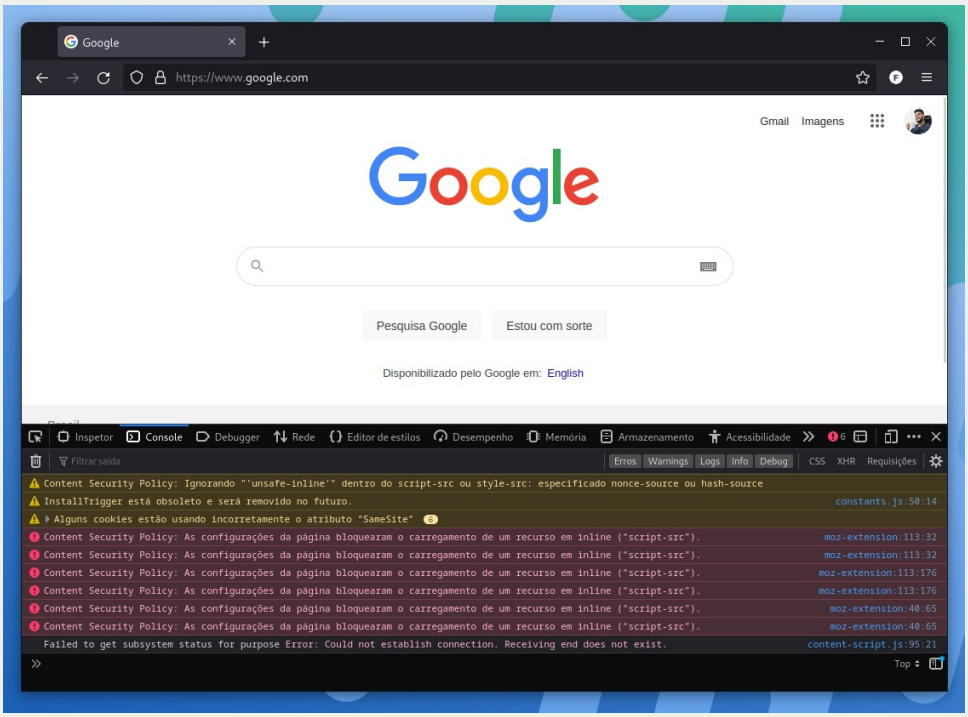


Figura 2 – Navegador Firefox com destaque de erros que aparecem no console.
Fonte: O próprio autor.

No Console, é possível escrever comandos em JavaScript, que serão interpretados e executados na página que está aberta, permitindo que interaja diretamente com elementos da mesma.

Porém, para começar a desenvolver códigos mais complexos, precisamos de um editor de texto. Por exemplo, podemos utilizar o Gedit, em sistemas baseados em Linux, ou mesmo o Bloco de Notas, no Microsoft Windows.

Porém, é mais vantajoso utilizar softwares chamados “Integrated Development Environment” (IDE), ou, em português, Ambiente de desenvolvimento integrado. Estes são editores de texto com foco na produtividade de escrita de código, trazendo recursos como facilidade de navegação entre vários arquivos, integração com gerenciadores de versão, auxílio de autocompletar em certos trechos de códigos, entre outros.

Existem vários editores desse tipo, cada um com suas vantagens e recursos que são, no fim das contas, únicas para cada desenvolvedor. Sendo assim, não há uma “Melhor IDE” e sim a que melhor se adapta ao seu estilo de trabalho. Neste livro iremos utilizar o Visual Studio Code, por ser um editor popular, gratuito e de código aberto, além de estar disponível para todos os sistemas operacionais.

Para adquiri-lo, basta ir para https://code.visualstudio.com/, fazer o download da versão compatível com seu sistema operacional, e instalá-lo seguindo as instruções que irão aparecer na tela.

Ao abrir o Visual Studio pela primeira vez, observe que temos vários elementos visuais, para acessar as ferramentas disponibilizadas pelo editor. Devemos dar destaque à barra lateral esquerda, com vários ícones, como o de “Explorador” para navegar entre os arquivos em uma pasta de projeto, o “Pesquisar” para realizar buscas em arquivos, “Controle de código-fonte” que dá acesso à ferramenta de controle de versão do código e “Extensões” que permite instalar novas ferramentas e funcionalidades para o editor. É recomendado que tente explorar um pouco a janela do Visual Studio Code, de forma a descobrir um pouco do que essa IDE pode oferecer. Não se preocupe em decorar onde estão, ou saber quais as funcionalidades de tudo: a medida que for utilizando, com certeza irá entender cada “pedacinho” disponível.

Com o editor já instalado e conhecendo a possibilidade de utilizar o Console no navegador, já podemos iniciar os testes e programar em JavaScript!

1.3 Primeiro teste: “Olá mundo!”

Uma espécie de “padrão” no ensino de programação, é a introdução da linguagem com um código que mostre ao usuário “Hello world!” ou, em português, “Olá mundo!”. Por aqui, não seria diferente! Vamos iniciar os nossos testes utilizando o console do navegador: como explicado anteriormente, abra uma página qualquer da web (o “google.com”, é um exemplo) e aperte a tecla F12 (ou Cmd+Opt+J se estiver no Mac). No console, observe que há um campo de texto, que você irá utilizar para executar comandos. Digite: console.log(“Olá mundo”) e veja a frase aparecer logo em seguida, como na figura a seguir.

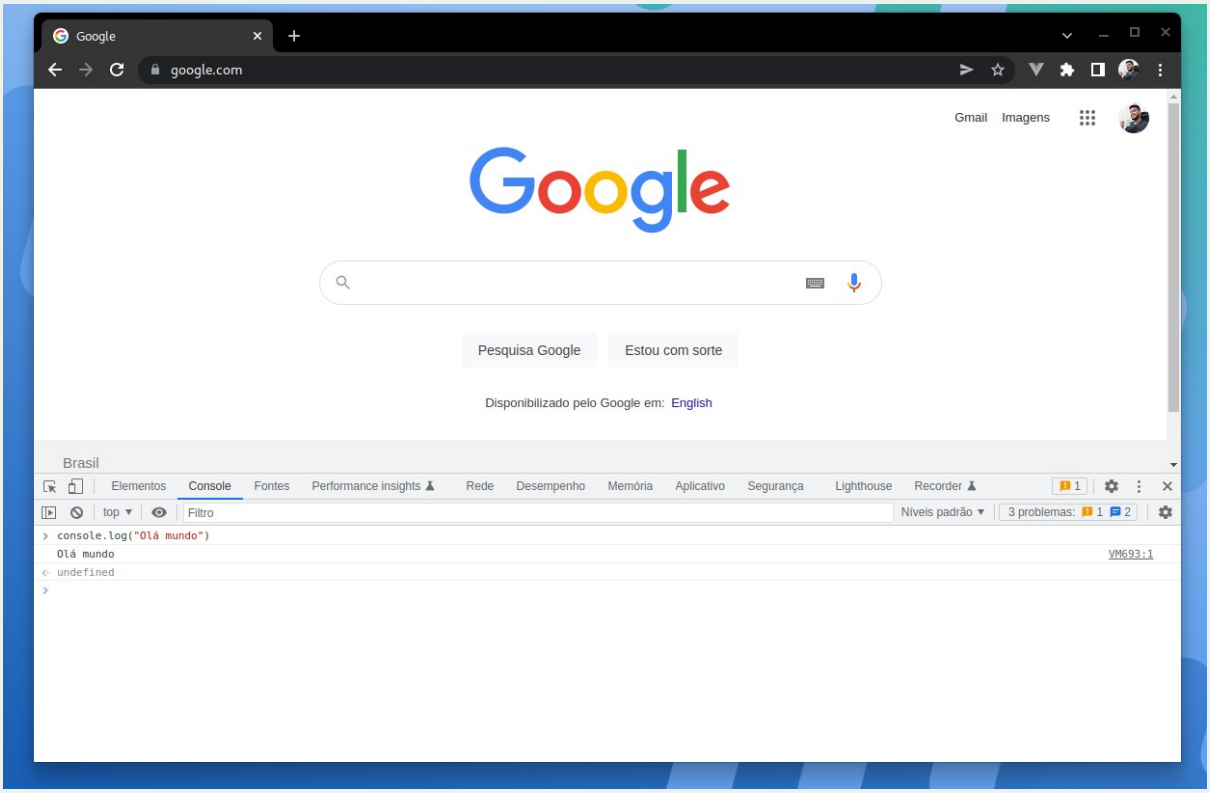


Figura 3 – Teste de “console.log” no Google Chrome.
Fonte: O próprio autor.

Agora vamos entender o que está acontecendo: você acabou de usar o seu primeiro comando no JavaScript! Em todos os navegadores, o Console é uma ferramenta para identificar falhas no código (também chamadas de bugs) ou mesmo exibir informações úteis para o desenvolvedor. Neste caso, usamos a função “log” do console, que permite exibir na tela do Console, textos.

Outra coisa que podemos aprender com esse exercício simples é sobre a sintaxe, ou seja, o padrão de como devemos escrever comandos: observe que a frase “Olá mundo” foi digitada entre aspas e dentro de um parênteses., logo depois do nome da função. Iremos estudar nos próximos capítulos sobre o motivo disso acontecer, mas de início, tente modificar o comando para que exiba a frase “Curso de JS”, no console.

Após testar diretamente no console, está na hora preparar uma página própria para testes de códigos. Basicamente, teremos uma página HTML em branco, com os códigos inseridos na mesma. Para isto, vá até o Visual Studio Code, e nos menus superiores “Arquivo”, depois em “Novo arquivo de texto”. Uma página em branco será exibida, copie e cole o código HTML abaixo e salve o arquivo como “cursoJS.html”.

```
<html>
                                                    <body>
                                <h1>Curso          JS          Básico</h1>
```

```
                                                    <script>
                                    console.log("Olá        Mundo");
                                                    </script>
                                                    </body>
</html>
```

Logo em seguida, abra o arquivo salvo em seu navegador. Para isto, vá pelo gerenciador de arquivos do seu sistema operacional até a pasta em que salvou o mesmo, e clique duas vezes sobre o arquivo para abri-lo em seu navegador padrão. Você verá apenas uma página em branco com um texto “Curso JS Básico”. Porém, se abrir novamente o console, como vimos anteriormente, verá a frase “Olá Mundo”. Como na imagem abaixo.

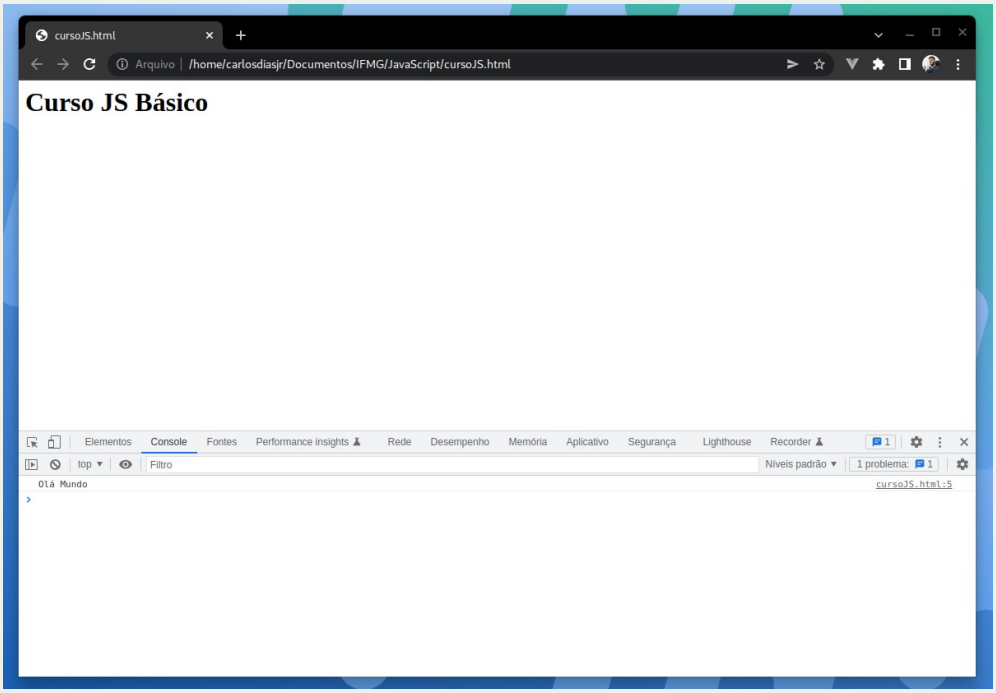


Figura 4 – “Olá Mundo” a partir de arquivo HTML personalizado.
Fonte: O próprio autor.

Mas, como funciona esse código? Como explicado anteriormente, o JS foi, inicialmente, criado para ser uma linguagem que é interpretada no navegador da web, para criar páginas dinâmicas. O arquivo que acabou de criar é um código HTML básico: primeiramente indicamos que é um código HTML com a tag “<html>” e no corpo desta página, demarcado pelas tags “<body>” e </body>”, inserimos um texto grande, com “<h1>”, e um bloco de código, marcado com “<script>” e “</script>”. Será entre essas marcações de script que iremos inserir todos os códigos que estaremos trabalhando nos próximos capítulos.

Por exemplo, vamos tentar uma coisa nova: ao invés de mostrar o resultado no Console, vamos interagir com o usuário mostrando uma janela de alerta. Para isto,

modifique o código anterior, removendo a linha de JS “console.log("Olá Mundo");” alterando para “alert("Olá Mundo");”. Agora, sua página de testes ficará assim:

```
<html>
                                                    <body>
                                <h1>Curso      JS      Básico</h1>
                                                    <script>
                                          alert("Olá      Mundo");
                                                    </script>
                                                    </body>
</html>
```

Agora abra a página novamente em seu navegador.

Dica do Professor: Se a página de testes já estiver aberta, basta apertar a tecla F5, e o navegador irá atualizar o conteúdo da mesma.

Como você pode perceber, ao carregar a página lhe é exibido uma janela de alerta, com um botão de “ok”. Isto é muito útil quando precisamos informar ao usuário erros, coisas importantes, ou apenas dar um “oi” como fizemos agora.

Estes foram os nossos primeiros passos com JavaScript, e a jornada de aprendizado está apenas começando! Nos próximos capítulos, iremos entender como funciona uma linguagem de programação como o JavaScript, como manipular variáveis (o que são variáveis), verificações de condições, entre outras coisas úteis para se tornar um programador. Além disso, vamos mostrar como funciona a sintaxe da linguagem, ou seja, como é o padrão que iremos utilizar para programar, ao longo de todos os capítulos. Espero que goste, bons estudos!

Atividade: Para concluir, vá até a sala virtual e responda o questionário desta semana.

Objetivos

Com os primeiros testes de códigos e exibindo mensagens ao usuário, nesta semana aprenderá o básico da sintaxe de JS, criação e tipos de variáveis.

2.1 Variáveis e tipos primitivos

Nesta seção, vamos entender o que são variáveis e como utilizá-las em JavaScript. É importante ressaltar que esse conceito é comum a outras linguagens de programação, e é super útil. Imagine a seguinte situação: você precisa guardar dados de contato de uma pessoa, como nome, telefone, e endereço de email. Como você faria isso? Talvez um caderno ou um papel, como na imagem a seguir…

Figura 5 – Exemplo de anotação, para criação de variáveis.
Fonte: O próprio autor.

Observe o que fizemos aqui, de forma bem simplificada: nós salvamos os dados, colocando rótulos neles para que eu saiba o que cada um representa. Em programação de computadores, variáveis é justamente isso: uma maneira de armazenar dados, colocando nomes que identificam o que está sendo salvo.

Para criar uma variável em JavaScript, deve-se utilizar a palavra chave “let”: da mesma maneira que quando deseja-se exibir uma mensagem de alerta na tela, é utilizado o comando “alert”, essa palavra chave é a maneira de indicar ao interpretador que deverá ser criada uma variável.

Como exemplo, vamos criar as três variáveis apresentadas anteriormente: nome, telefone e e-mail. Para isto, deverá haver três linhas de código, uma para cada variável que se deseja criar. Observe o código da nossa página HTML, abaixo:

```
<html>
                                                    <body>
                                    <h1>Curso JS Básico</h1>
                                                    <script>
    let nome = “Carlos”;
    let telefone = “(31)98765-4321”;
                                    let email = “email@email.com”;
                                                    </script>
                                                    </body>
</html>
```

Observe novamente a sintaxe utilizada para criar estas variáveis: no início da linha, deve-se aparecer a palavra reservada “let”, indicando que irá ser criada uma variável, depois vem o nome da variável e, por último, um igual que indica que a seguir será digitado o valor que será salvo naquela variável.

É importante salientar que os valores a serem salvos (por exemplo o nome “Carlos”) ficam entre aspas, pois indica que aquilo é um dado, no caso, todos os dados são textos. Mas e se o valor a ser salvo for um número? Por exemplo, gostaria de salvar um valor que seria a altura do meu contato. Poderíamos fazer da seguinte maneira:

```
<html>
                                                    <body>
                                    <h1>Curso JS Básico</h1>
                                                    <script>
    let nome = “Carlos”;
    let telefone = “(31)98765-4321”;
    let email = “email@email.com”;
                                    let altura = 1.86;
                                                    </script>
                                                    </body>
</html>
```

Neste caso, foi utilizada “let” novamente, mas o valor a ser salvo na variável “altura” não está mais entre aspas, sendo apenas um número com ponto, pois, em geral, não se utiliza vírgula para separar os decimais em linguagens de programação. Com isso, temos mais um conceito em programação: o tipo da variável. As variáveis podem armazenar dados de diferentes tipos (como números inteiros, números com decimais e textos) e, de acordo com o que será salvo, a forma de escrever esse dado se modifica. É por isso que o texto que salvamos na variável “email” está entre aspas, por ser um texto, e a altura não, por ser um número.

Em outras linguagens de programação (como em C, por exemplo) o programador precisa dizer no código o tipo da variável que será armazenada. Em JavaScript, que é um tipo de linguagem chamada “não tipada”, não existe necessidade de marcar os tipos. Porém, no JavaScript, existem 6 tipos primitivos:

- **String**: para representar conjuntos de caracteres, ou seja, textos.
- **Number**: que engloba número inteiros ou não
- **Boolean**: para lógica Booleana (verdadeiro ou falso, 1 ou 0)
- **Null**: quando a variável tem um valor nulo (sem valor)
- **undefined**: no caso da variável não existir ainda na memória
- **Symbol:** novo no ECMAScript 6, é um tipo para armazenar valores únicos e otimização do uso de memória

Devemos observar, então, a importância de se escrever corretamente e manipular as variáveis com os tipos que envolvem um pouco de atenção, mas que irá se acostumar à medida que desenvolver programas.

Agora vamos testar como usar essas variáveis: assim como já foi feito anteriormente, podemos exibir um texto para o usuário através de uma janela de alerta ou no console. Neste teste, será exibido “O email salvo no contato de Carlos é email@email.com”, utilizando os dados salvos nas variáveis. Para isso, vamos modificar o código anterior, adicionado a seguinte linha de código:

```
alert("O email salvo no contato de " + nome + " é " + email);
```

Neste momento, pode-se destacar mais uma lição de sintaxe da linguagem: o texto, que não é variável, ficou entre aspas enquanto os nomes das variáveis não. Isso acontece para que o interpretador entenda que você quer utilizar a variável chamada “nome” e não o texto, escrito nome. Além disso, já conseguimos perceber como juntamos variáveis e textos: usando o operador “+”. Um pouco mais a frente, vamos explicar como funcionam outros operadores. O código final então da nossa página ficou assim:

```
<html>
                                                                    <body>
                                                <h1>Curso          JS          Básico</h1>
                                                                    <script>
    let nome = “Carlos”;
    let telefone = “(31)98765-4321”;
    let email = “email@email.com”;
    let altura = 1.86;
            alert("O email salvo no contato de " + nome + " é " + email);
                                                                    </script>
                                                                    </body>
</html>
```

Ao abrir a página HTML no navegador, será exibido um alerta, como esperado. Porém, e se você quiser entrar com seus dados? Neste curso, o foco é que você aprenda os conceitos básicos de JS, sem mexer com o HTML (que é chamado manipular o DOM). Por isso, vamos utilizar uma espécie de alerta, mas para receber informações digitadas pelo usuário chamado “prompt”. Veja o código completo abaixo:

```
<html>
                                                                    <body>
                                                <h1>Curso          JS          Básico</h1>
                                                                    <script>
    let nome = prompt(“Por favor insira seu nome”);
    let telefone = prompt(“Por favor insira seu telefone”);
    let email = prompt(“Por favor insira seu email”);
            alert("O email salvo no contato de " + nome + " é " + email);
                                                                    </script>
                                                                    </body>
</html>
```

Desta maneira, será perguntado ao usuário os dados, e depois será exibido uma mensagem de alerta com os dados que foram obtidos.

2.2. “let”, “const” e variáveis globais

Agora imagine a seguinte situação: o Carlos, esse contato que salvamos, mudou de telefone. Como vamos modificar o número de telefone dele? Em programação isso é possível modificando o valor da função.

Antes de modificar, vamos entender como funciona o armazenamento de variáveis na memória do computador: imagine que a memória volátil (a memória em que os dados são perdidos quando se desliga a energia, que no caso do PC é a RAM, não confunda com o disco rígido, que é uma memória não volátil, e que seus arquivos estão guardados) do computador é um armário com gavetas. Quando você cria uma variável, é como se o sistema estivesse colocando uma etiqueta nestas gavetas (veja a figura a seguir) e quando você coloca um valor, o dado é armazenado nessa gaveta.

Figura 6 – Analogia entre a memória de um computador, e a criação de variáveis, com gavetas.
Fonte: O próprio autor.

Observe que a identificação é de acordo com o nome da variável. Então, chegamos a uma conclusão lógica: não podemos criar variáveis com o mesmo nome em um código ou, então, como o sistema saberá qual variável quer modificar?

Mídia digital: No vídeo “Variáveis” falamos um pouco sobre isso, com demonstração direto no editor de código. Assista o vídeo e revise aqui!

Falando em modificar o valor das variáveis, imagine a seguinte situação: Carlos, o contato que salvamos nas variáveis anteriormente, mudou o número de telefone. Para entender como modificar esta variável, veja o código a seguir:

```
<html>
                                                                    <body>
                                        <h1>Curso JS Básico</h1>
                                                                    <script>
    let nome = “Carlos”;
    let telefone = “(31)98765-4321”;
    let email = “email@email.com”;
    let altura = 1.86;
    telefone = “(31)98877-6543”;
            alert("O email salvo no contato de " + nome + " é " + email);
                                                                    </script>
                                                                    </body>
</html>
```

Agora vamos analisar o código: observe que adicionamos a linha marcada de verde, nesta linha não digitamos a palavra reservada “let”, pois não vamos criar a variável “telefone” novamente, queremos apenas modificar o valor da mesma. Por fim, o código irá gerar um alerta e o número de telefone que irá aparecer não é “(31)98765-4321” e sim “(31)98877-6543”, pois o mesmo foi alterado.

O código é executado de cima para baixo, ou seja, as primeiras linhas são executadas primeiro que as de baixo. Por isso, quando criamos a variável “telefone” e na linha abaixo modificamos o valor desta, quando a linha com o “alert” é chamada, o valor já foi alterado.

Mas vamos a um caso diferente: gostaria de que o nome não pudesse ser alterado em nenhum momento do código, ou seja, ela não será mais uma variável e sim uma constante. Para isto, precisamos utilizar da palavra reservada “const” ao invés de “let” ao criar o “nome”. Veja o código abaixo, com o “const” adicionado.

```
<html>
                                                                    <body>
                                        <h1>Curso JS Básico</h1>
                                                                    <script>
    const nome = “Carlos”;
    let telefone = “(31)98765-4321”;
    let email = “email@email.com”;
    let altura = 1.86;
```

```
    telefone = “(31)98877-6543”;
            alert("O email salvo no contato de " + nome + " é " + email);
                                                                    </script>
                                                                    </body>
</html>
```

Atividade: Que tal testar algumas coisas no JS? Adicione uma linha no código anterior para tentar alterar o nome para “Teste”. Veja no console do navegador: a mensagem aparece como esperava?

2.3 “String”

Como já vimos anteriormente, o JavaScript tem tipos primitivos de variáveis. Um tipo em especial que já utilizamos anteriormente, foi a “String”. Esta pode ser utilizada para salvar dados em forma de texto ou qualquer sequência de caracteres. Como assim “sequência de caracteres”? Isso quer dizer que em uma String podemos guardar textos como o nome completo de uma pessoa, ou o seu endereço, e esses dados na verdade nada mais são do que uma sequência de símbolos/caracteres (no caso do nome é uma sequência de letras com espaços). Isto é particularmente interessante, pois uma String permite que um caractere específico do dado que está salvo seja acessado.

Veja o exemplo a seguir: nele criamos uma variável chamada “email” e exibimos ao usuário.

```
<html>
                                                                    <body>
                                        <h1>Curso      JS      Básico</h1>
                                                                    <script>
    let email = “meu@email.com”;
            alert("O caractere na posição 3 do email é " + email[3])
                                                                    </script>
                                                                    </body>
</html>
```

Observe que o caractere que aparece no alert é o “@”. Por que isso acontece? cada caractere dentro da String “email” tem um número, a posição em que ele está, começando pelo primeiro caractere, que é o 0 (zero). Mais uma vez, devemos ficar atentos à sintaxe: primeiro vem o nome da variável (no caso é “email”) e depois entre colchetes (“[“ e “]”) o

número da posição que deseja acessar. Também é possível fazer a mesma coisa utilizando a função “charAt”, como mostrado a seguir:

```
<html>
                                                                    <body>
                                        <h1>Curso          JS          Básico</h1>
                                                                    <script>
    let email = “meu@email.com”;
            alert("O caractere na posição 3 do email é " + email.charAt(3))
                                                                    </script>
                                                                    </body>
</html>
```

Agora, ao invés de utilizar os colchetes, adicionamos um “.charAt” (atente-se ao ponto, que indica que queremos trabalhar com a variável que vem logo antes dele) e com a posição entre parênteses.

Outra manipulação interessante que podemos fazer com uma String é o método “replace”, para substituir um determinado texto em uma variável. Veja o exemplo a seguir:

```
<html>
                                                                    <body>
                                        <h1>Curso          JS          Básico</h1>
                                                                    <script>
    let email = “meu@email.com”;
    email = email.replace(“meu”, “javascript”)
                                    alert("    Email:      " + email)
                                                                    </script>
                                                                    </body>
</html>
```

O que fizemos foi substituir uma parte do texto, e o alerta irá exibir “Email: javascript@email.com”. Novamente, a sintaxe, e como usar a função aqui é muito importante: quando colocamos um “replace”, o código irá retornar o texto modificado apenas, não salvando na memória o valor modificado. Por isso, na linha destacada em verde, fizemos que a variável email é ela mesma, mas substituindo “meu” por “javascript”.

2.4 Operadores

Agora que já entendeu a dinâmica de armazenar dados na memória, seja através de variáveis ou utilizando de constantes,pode pensar “como posso fazer modificar tudo isso?”. Para isso vamos utilizar operadores, que são símbolos para indicar uma certa operação/manipulação das variáveis. Por exemplo, os símbolos matemáticos: é possível realizar cálculos utilizando as quatro operações básicas, com seus símbolos (+ para adição , - para subtração, * para multiplicação e / para divisão). Se desejar realizar operações matemáticas mais complexas, como potenciação, raiz quadrada ou cosseno, deve-se utilizar recursos disponíveis na biblioteca “Math”, padrão em interpretadores JS. Veja os exemplos a seguir:

```
<html>
                                                    <body>
                                <h1>Curso      JS      Básico</h1>
                                                    <script>
    let soma = 5 + 2;
    let subtracao = 3 - 1;
    let multiplicacao = 8*9;
    let divisao = 10/5;
    let raizQuadrada = Math.sqrt(81);
    let potenciacao = Math.pow(2,3);
    let cosseno = Math.cos(Math.PI*2);
                                                    </script>
                                                    </body>
</html>
```

No código de exemplo anterior, “raizQuadrada” foi calculada utilizando a função “sqrt” da biblioteca “Math”, entre parênteses, temos o número que desejamos calcular a raiz quadrada. Já na variável “potenciacao” é calculado o valor de 2^3 (que é igual a 8) utilizando a função “pow” que recebe dois argumentos entre parênteses: primeiro o número da base, uma vírgula e o expoente. A variável “cosseno” é um pouco mais complexa: nela vamos utilizar a função “cos” que fornece uma maneira de calcular o cosseno de um ângulo em radianos. O número PI pode ser entregue através de “Math.PI” que é uma constante dentro da biblioteca “Math”.

Atividade: Calcule o cosseno de π/2 utilizando “Math.cos” e a constante “Math.PI” e mostre o resultado no console. Depois faça o mesmo cálculo aproximando π = 3.14 utilizando uma constante que você criou.

Além disso, há outros operadores úteis para lógica de programação. Por exemplo, imagine que você quer incrementar o valor de uma variável chamada “idade”: você poderia fazer da seguinte maneira:

```
let idade = 27;
idade = idade +1;
```

Ou seja, o novo valor de idade é o valor atual, mais 1. Mas você poderia também fazer da seguinte maneira:

```
let idade = 27;
idade += 1;
```

Esta forma de operador de atribuição funciona para todas as operações matemáticas, apresentados anteriormente, e simplifica a escrita do código.

Atividade: Teste todos os operadores de atribuição: “+=”, “-=”, “/=” e “*=”.

Outros operadores, como os condicionais, serão apresentados na próxima semana de estudos, junto a questões de lógica.

Objetivos

Aprenda a trabalhar com condições para executar código e repetições de código em JavaScript

3.2 Condições

É tão comum em nosso dia a dia, trabalharmos com condições e decisões que nem percebemos: se o semáforo de pedestres está verde, vou atravessar a rua, se o estou com sede, devo tomar água ou se quero utilizar meu computador, vou ligá-lo. Em programação, também é necessário estabelecer algumas condições, que deverão ser analisadas pelo código, como por exemplo: se o CPF digitado não for válido, exibir alerta ao usuário ou se o usuário e senha não estiverem válidos, bloquear o acesso ao sistema.

Sabendo disso, volte no parágrafo anterior e veja a palavra que mais se repetiu, para descrever cada condição apresentada: “se”. Sempre que desejar que, para que ocorra algo (efeito) , seja necessário uma condição (causa) você irá usar “se”.

Em JavaScript, assim como em várias linguagens de programação, este “se” torna-se “if”: um bloco de condição que irá permitir que tome decisões baseado em estados no seu código.

Essa estrutura deverá ser assim:

```
if (condição) {
    Código, caso a condição seja verdade
}
```

A sintaxe aqui apresentada será bem comum daqui para frente: sempre que houver um bloco de instruções (linhas de código), iremos colocá-los entre chaves (“{“ e “}”) para indicar que aquele trecho está “dentro” da linha anterior. Para construir as condições, iremos utilizar os operadores lógicos de comparação que podem ser:

Operador	Descrição	Exemplo
==	Igualdade ou seja testa se “ ‘a’ é igual a ‘b’ “	1==2 nome == “carlos”
!=	Desigualdade, ou seja “ ‘a’ é	1 != 2

	diferente de ‘b’ “	idade != 27
>	“Maior que”	2 > 3 altura > 1.8
<	“Menor que”	3 < 1 saldo < 0
>=	“Maior ou igual que”	2 >= 3 altura >= 1.8
<=	“Menor ou igual que”	3 <= 1 saldo <= 0

Atividade: Faça um programa que mostre na tela se o valor de uma variável “altura” é maior ou igual a 1.8 e se a variável “cidade” é diferente de “Ibirité”

É importante salientar que, como apresentado anteriormente, as variáveis em JS não têm tipo definidos ao serem criadas, o que significa que comparações como a mostrada a seguir são verdadeiras:

```
let a = 2;
let b = “2”;
console.log(a==b)
```

Este trecho de código irá retornar “true” no console, pois é ignorado o fato de que a variável “b” é uma String, e o valor numérico dela é comparado. Se desejar comparar estritamente, considerando o tipo da variável, deverão ser utilizados os operadores “===” e “!==”. Assim, o código para retornar que “a” é diferente de “b” devido ao fato de serem de tipos diferentes é:

```
let a = 2;
let b = “2”;
console.log(a===b)
```

Com tudo isso em mente, voltemos à construção da condição na programação em JS. Imagine a seguinte situação: um gerente de um supermercado tem um banco de dados com as quantidades de cada produto. Queremos gerar um alerta toda ao usuário, se a quantidade de laranjas for menor que 5. Veja o código de exemplo abaixo:

```
let qtdLaranjas = 4;
if(qtdLaranjas <5){
    alert(“AVISO: o estoque de laranjas está baixo!”);
}
```

E se desejarmos mostrar uma mensagem caso o número de laranjas não for menor que 5? Podemos fazer um código com outro “if” em que a condição seja “qtdLaranjas > 5”, e funcionaria corretamente. Porém, em JS e outras linguagens de programação, há a opção e usar “else”.

Atividade: Faça um código com dois blocos “if”, que chamem um alerta caso o número de laranjas seja menor que 10 e outro alerta caso seja maior ou igual a 10.

O “else” nada mais é do que dizer “se não…”, ou seja, caso a condição anterior não seja verdadeira. É importante salientar que só existe “else” depois de uma condição, o que significa que sempre teremos blocos encadeados de “if” e “else”. Veja o código a seguir:

```
let qtdLaranjas = 5;
if(qtdLaranjas <5){
    alert(“AVISO: o estoque de laranjas está baixo!”);
```

```
}
else{
    alert("Estoque de laranjas está OK!");
}
```

Mais uma vez, devemos estar atentos à sintaxe: “else” abre um bloco de linhas de código que deverão estar entre chaves, assim como no “if”. Além disso, o bloco de “else” só é iniciado, depois que o bloco de “if” é fechado.

Atividade: Faça um código com “if-else” para validar se o dado de uma variável chamada “senha” é “1234abc”.

3.3 Laços de repetição

Às vezes, se faz necessário realizar tarefas repetitivas, em que o mesmo trecho de código será executado várias vezes até uma determinada condição. Por exemplo: imagine que nossa aplicação necessita de mostrar ao usuário múltiplos de um número de 1 a 10. Por exemplo: queremos mostrar no console para o usuário, números entre 0 e 5, como no código a seguir:

```
console.log(0);
console.log(1);
console.log(2);
console.log(3);
console.log(4);
```

Parece fácil, certo? Mas e se desejarmos todos os números entre 0 e 100? É perceptível como a dificuldade e a repetição de código é inevitável. Para resolver este tipo de problema, devemos recorrer a estruturas de repetição na programação.

Iniciemos por uma estrutura chamada “for”: para (que é o que significa “for” em inglês) uma determinada condição inicial, enquanto uma condição não for atingida é repetido um loop, como no exemplo a seguir:

```
for(condição inicial; condição de parada; expressão final){

    código a ser repetido

}
```

Novamente, atentemos à sintaxe exibida anteriormente: é comum que a condição inicial seja a criação de uma variável que será utilizada para verificar quantas vezes o laço já foi repetido, além de que a expressão final, geralmente é apenas o incremento dessa variável. Já a condição de parada, segue a mesma sintaxe, utilizando operadores condicionais, apresentados anteriormente.

Vamos agora desenvolver o código para mostrar no console todos os números de 0 a 100, utilizando uma estrutura de repetição com “for”:

```
for(let i = 0; i< 100; i++){
    console.log(i);
}
```

Como pode ser visto, com apenas 3 linhas de código, no console será exibido 100 saídas (começando no número 0 e terminando com 99). Esta é uma clara demonstração de como laços de repetição simplificam códigos e agilizam o processo de desenvolvimento.

Além do “for”, há a opção de se utilizar um outro tipo de laço: o “while”. Em tradução literal, “while” significa “enquanto”, ou seja, “enquanto uma condição for verdadeira faça isso…”. Como por exemplo, vamos refazer o código anterior, feito com “for”:

```
let i = 0;
while(i< 100){
    console.log(i);
    i++;
}
```

Atividade: Faça códigos que mostram no console números entre 1990 e 2022 de duas maneiras: utilizando o “for” e utilizando “while”.

Neste caso,não temos mais uma sintaxe com vários argumentos: o “while” recebe apenas um argumento que é a condição de manter a repetição. Leia a linha “while(i< 100){” como “enquanto i for menor do que 100, faça…”. Uma forma alternativa de se fazer o “while” é utilizar uma estrutura “do… while”. Por exemplo, você quer realizar uma ação uma vez, e repetir ela enquanto uma situação seja verdadeira.Veja o código abaixo:

```
let i = 0;
do{
    console.log(i);
    i++;
} while(i< 100);
```

Atividade: Faça um teste no código utilizando apenas “while”: faça i = 100. O código dentro do laço de repetição é executado? E se fizer o mesmo em um laço do tipo “do… while”?

A aplicação de “while” e “for” depende do caso, de como você enquanto desenvolvedor acha que será mais válido, e saberá a medida que utilizar, e entender melhor. Mais aplicações de laços de repetição poderão ser vistas na seção a seguir, “Trabalhando com arrays”.

3.4 Trabalhando com arrays

Até agora, vimos que uma variável pode guardar um dado, seja ele uma String, seja um número… Mas, às vezes, precisamos representar um conjunto de dados maior, por exemplo: você está fazendo uma aplicativo para ajudar a gerenciar sua lista de compras, logo precisa de alguma forma de salvar vários dados, que são os nomes dos itens que precisa comprar. Da maneira que sabemos criar variáveis até agora, teríamos que criar várias variáveis, uma para cada item que iria comprar, certo? Mas isso seria inviável, pois sua lista estaria limitada à quantidade de variáveis que você criou.

Para este tipo de problema, o JavaScript dispõe de um tipo de objeto muito interessante: o array. Neste tipo de objeto, semelhante a uma lista, temos vários dados

armazenados em sequência, ou seja, assim como com variáveis que são Strings, que é uma lista de caracteres, nos arrays conseguimos acessar os itens de forma independente. Porém, a grande diferença da String, é que um array não tem um tipo fixo de variável: podemos ter uma lista de números e Strings, ou só de números, ou só de Strings…

Para começar, vamos criar um array com a nossa lista de compras, veja o código a seguir.

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
console.log(“O primeiro item da lista é ” + lista[0]);
```

O que nos é mostrado no console, é o primeiro item da lista: “arroz”. Isto pois, assim como com Strings, em um array, a contagem se inicia em zero. Assim, se desejo ver o segundo item da lista, devo acessar “lista[1]”, e assim por diante. O interessante é que, conseguimos alterar o valor do dado salvo em cada posição do array da mesma maneira que uma variável comum, utilizando também a sintaxe de “nome_da_variavel [posição]”, como neste exemplo:

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
lista[0] = “feijão”;
console.log(“O primeiro item da lista é ” + lista[0]);
```

Porém, pode ser que precisemos adicionar itens à nossa lista de compras: agora eu preciso comprar também “pasta dental”. Para adicionar um item, podemos utilizar o método “push”, e o novo item irá para o final da lista. Com o código a seguir, no console, será exibida a lista completa, agora com o item que adicionei no final.

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
lista.push(“pasta dental”);
console.log(lista);
```

Atividade: Faça com laço de repetição e prompt um código que peça ao usuário mais 3 itens para adicionar à lista.

Além disso, se deseja adicionar um item no começo da array, pode-se utilizar “unshift”.

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
lista.unshift(“pasta dental”);
console.log(lista);
```

Mídia digital: No vídeo "Condições e laços de repetição” é feito um exemplo bem legal de lista de compras. Veja como ficou e tente fazer a sua versão!

E se deseja remover um item, deverá utilizar o método “pop”, caso deseje apagar o último, e o primeiro pode ser removido com “shift”. Veja esses dois exemplos:

Remover o último item

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
lista.pop();
console.log(lista);
```

Remover o primeiro item

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
lista.shift();
console.log(lista);
```

Como pode perceber, o tamanho de um array é variável, podendo mudar a qualquer momento, como queira. Pra saber a quantidade de itens que estão armazenados neste objeto, há uma propriedade: “length”, que literalmente quer dizer “tamanho”. Esta propriedade pode ser útil para verificar quantos itens ainda estão na lista, ou para fazer laços de repetição, por exemplo.

```
let lista = [“arroz”, “batata”, “leite”, “tomate”, “sabão em pó”];
console.log(“Na lista há ” + lista.length + “ itens”);
```

Atividade: Agora que já conhece laços de repetição, que tal fazer um laço para mostrar no console todos os itens que estão na lista de compras? **Dica:** o critério de repetição é a quantidade de itens que há no array

Além disso, como dito anteriormente, arrays aceitam vários tipos de variáveis, e isso significa até mesmo outros arrays. Isto é muito útil para fazer tabelas, em que nos interessamos não apenas na posição unidimensional, mas em duas dimensões (linha e coluna). Por exemplo, vamos criar um tabuleiro de xadrez: precisamos indicar onde cada peça está em linha e coluna, isto pode ser feito com um array que contém vários arrays que representam as linhas do tabuleiro. Veja o código a seguir:

```
let board =
    [ ['T','C','B','Q','K','B','C','T'],
      ['P','P','P','P' ,'P','P','P','P'],
      [' ' , ' ' , ' ' , ' ' , ' ', ' ', ' ' , ' ' ],
      [' ' , ' ' , ' ' , ' ' , ' ', ' ', ' ' , ' ' ],
      [' ' , ' ' , ' ' , ' ' , ' ', ' ', ' ' , ' ' ],
      [' ' , ' ' , ' ' , ' ' , ' ', ' ', ' ' , ' ' ],
      ['p','p' ,'p' ,'p' ,'p','p', 'p','p'],
      ['t' ,'c' ,'b' ,'q' ,'k' ,'b','c', 't' ] ];

console.log(board.join('\n'));

console.log(“\n\n Fazendo o peão avançar”);
board[4][4] = board[6][4];
board[6][4] = ' ';
console.log(board.join('\n'));
```

Vamos agora entender vários conceitos envolvidos neste trecho de código. O primeiro é o “\n”, isto é um caractere especial que indica quebra de linha, ou seja, indica que é para o cursor de texto, passar para a linha logo abaixo. Usamos esse recurso para deixar melhor formatadas as Strings que exibimos ao usuário. Segundo conceito: “board.join('\n')”. Este método, “join”, une todos os elementos do array em uma única string. No nosso caso, ele juntou os elementos internos de cada linha no array “board”. Observe que, entre parenteses, há um “\n”: o método “join” recebe um argumento o separador entre os elementos do array que ele irá juntar. Em resumo, o comando “board.join('\n')” fez com que o JS criasse um novo array, com os elementos (no caso, os elementos de “board” são as linhas) separados por uma quebra de linha.

Por fim, podemos aprender com esse código como acessar um elemento em uma “linha” e “coluna” específica do array, assim como feito em “board[6][4]”: acessando o elemento na linha 6, coluna 4.

Atividade: Mova um cavalo das letras maiúsculas para frente, lembre-se das regras do xadrez: o cavalo se move em “L”

Agora, finalizamos mais uma semana! Na próxima, vamos aprender como utilizar variáveis em blocos de código, chamados funções. Até lá!

Atividade: Para concluir, vá até a sala virtual e responda o questionário desta semana.

Objetivos

Com foco em tratar dados e objetos em JS, nesta semana entenderá sobre funções e como usar alguns recursos da linguagem para a web.

4.1 Conceito e criação de funções

Esta semana é a última do curso, o que significa que vamos usar tudo que aprendemos até agora para construir programas mais complexos e, por consequência, mais legais! Essa aventura começa com um conceito muito interessante: funções. Talvez você conheça essa palavra, por causa da matemática, por exemplo, temos uma função de segundo grau "$f(x) = x^2 + 2x + 1$".

Em linguagens de programação, uma função é um subconjunto de linhas de códigos, como se fosse um subprograma, que pode ser chamado por outro. Ou seja, assim como nos laços de repetição, que nos permite ter blocos de código que vão ser repetidos, nas funções também podemos fazer blocos, que serão utilizados em diferentes locais do meu código e até mesmo repetidas vezes. Além disso, as funções podem receber argumentos, ou seja, receber dados para usar dentro dos blocos de código.

Fazendo analogia com a função matemática de segundo grau, o argumento é um valor para "x". Quando fazemos por exemplo "f(2)" esse valor "2" será substituído nos lugares que aparece um "x", certo? Além disso, essa função "f" vai nos retornar um valor, que é o valor equivalente para "x=2".

Aqui, em JS, podemos fazer a mesma coisa: uma função, ou seja, um bloco de código, pode receber um valor (argumento) e retorna outro valor. Veja o exemplo:

```
function f (x){
    return Math.pow(x,2) +2*x + 1;
}

let f2 = f(2);
console.log("f(2) = " + f2);
```

Como fizemos em todas as semanas do curso, até aqui, vamos observar a sintaxe de criação de funções: assim como para criar uma variável, que utilizamos a palavra reservada “let”, para criar uma função precisamos utilizar primeiramente “function”, em seguida o nome da função, e entre parênteses os argumentos dessa função, ou seja, os dados que ela pode receber.

É importante salientar que uma função pode ter várias linhas e não necessariamente precisa retornar algo ou receber algum argumento. Como exemplo, vamos fazer uma função que, dado três dimensões de um paralelepípedo, ela calcula o volume e mostra ao usuário diretamente no console:

```
function logVolume (x,y,z){
    let v = x*y*z;
    console.log(“Dimensões: ”+x+ “,” + y + “,” + z);
    console.log(“Volume: ”+v);
}

logVolume(2,2,2);
```

Neste caso, a função não retorna nenhum valor, então não precisamos criar uma variável para armazenar o valor do volume fora da função. Porém, agora temos um conceito muito importante: escopo da variável.

Quando criamos esses blocos de código, seja através de um laço de repetição, seja através de uma função, as variáveis criadas dentro desse bloco, existem apenas enquanto ele está sendo executado. Em palavras mais técnicas, as variáveis criadas dentro de blocos são chamadas de variáveis locais, e não podem ser acessadas fora de seu escopo (fora do bloco em que foram criadas).

Isto significa que não podemos chamar a variável “v”, fora da função que ela foi criada, “logVolume”, além de que, assim que a última linha dessa função for executada, essa variável deixa de existir, e seu valor é apagado.

Atividade: Crie uma função chamada “printArray” que recebe um array como argumento e mostra no console, todos os elementos do mesmo.

4.3 Manipulando objetos em JavaScript

Até agora, cada variável armazenava um único dado. Porém, muitas vezes, os dados devem estar agrupados, de forma a simplificar o acesso aos mesmos e a busca por informações. Por exemplo, quando tentamos salvar dados de um contato, nós criamos uma variável para salvar o nome, uma para o telefone, uma para o e-mail… Ou seja, várias variáveis diferentes, para uma única pessoa. Se quisermos armazenar os dados de 2 contatos, deveríamos fazer algo como o código abaixo:

```
let nome = “Carlos”;
let email = “carlos@email.com”;
let nome2 = “Dias”;
let email2 = “dias@email.com”;
```

Atividade: Faça o código anterior novamente, utilizando arrays para nome, email e telefone.

Porém há uma maneira mais fácil de organizar os dados: utilizando objetos. Em JS, objetos são um conjunto de dados, ou seja, variáveis, ou funções. Os dados têm identificadores chamados de chaves, como no exemplo abaixo:

```
let contato = { "nome":"Carlos",
                "email": "email@email.com",
                "telefone": "(31)98765-4321" };
alert("O email salvo no contato de " + contato.nome + " é " + contato.email);
```

Neste exemplo, criamos as chaves “nome”, “email” e “telefone”, para armazenar os dados. Veja que criamos apenas uma variável que é “contato”, e conseguimos acessar os dados internos colocando “**nome_variavel.chave**”.

Como dito anteriormente, um objeto em JS pode ter funções internas também, que chamaremos de métodos, para isto, deve-se criar uma chave, assim como para as variáveis, e escrever uma função diretamente.

```
let contato = { "nome":"Carlos",
                "email": "email@email.com",
                "telefone": "(31)98765-4321",
                "dizerOi": function() { alert("Oi, eu sou o " + this.nome); }
};
contato.dizerOi();
```

Observe que para acessar a função "dizerOi", utilizamos a mesma metodologia que fizemos com acessar as variáveis, utilizando "**nome_variavel.método**". Entenda o ponto como se estivéssemos acessando um conteúdo dentro do contexto à esquerda do ponto (no caso, o nome da variável).

Também é possível fazer um array de objetos e, para demonstrar, vamos montar um exemplo de agenda com dois contatos:

```
let agenda = [ { "nome":"Carlos",
                "email": "email@email.com",
                "telefone": "(31)98765-4321" },
                { "nome":"Dias",
                "email": "dias@email.com",
                "telefone": "(31)12345-6789" }]
alert("O contato 0 é o " + contato[0].nome + " com email " + contato[0].email);
```

Assim como em arrays trabalhados anteriormente, para acessar cada elemento, devemos especificar entre colchetes a posição, retornando assim o objeto que criou. Depois, os valores dentro do objeto podem ser acessados através das chaves criadas.

4.4 Introdução ao JSON e finalização

Agora que conhece mais sobre objetos em JavaScript, está na hora de aprender sobre o que mais irá ouvir falar no mundo de desenvolvimento web: JSON (leia como se fosse o nome do ator que faz o Aquaman, Jason Momoa).

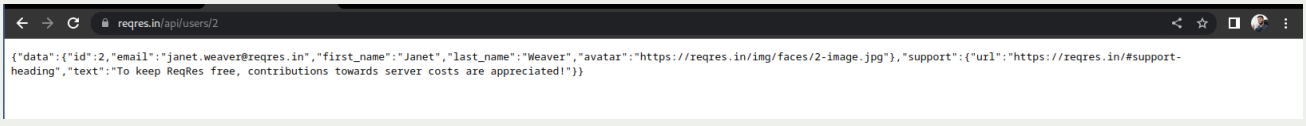
JavaScript Object Notation, ou JSON para os íntimos, é uma forma de representar dados como texto, seguindo a base da sintaxe de objetos em JS. Este é muito utilizado para transmitir dados entre aplicações web pois, com ele, dados complexos podem ser enviados de forma padronizada como uma String, através das chamadas API's ("Application Programming Interface" ou "Interface de Programação de Aplicação") que são justamente interfaces para padronizar a comunicação.

Dica do Professor: API's são muito interessantes e um assunto que não trataremos profundamente neste curso mas, se procurar um pouco na internet, verá muito material de base para entender como estes recursos são tão importantes.

Por exemplo, um sistema na web disponibiliza dados de temperatura, umidade e probabilidade de chuva na sua região, para que seu aplicativo mostre ao usuário se hoje é um bom dia para sair com guarda-chuvas, Ou você precisa de uma lista de usuários, que está salva em outro servidor, responsável por gerenciar todo o banco de dados. Uma maneira de se realizar estas tarefas é que a aplicação que detém os dados, te forneça um link para que acessar uma string JSON com as informações que precisa.

Mídia digital: Veja o vídeo "Trabalhando com Objetos", na sala virtual, com um exemplo prático de consulta de API.

Para testar tudo isso, vamos utilizar o *reqres.in* que disponibiliza diferentes dados de maneira simples para teste. Para começar, faça um teste: acesse em seu navegador o link https://reqres.in/api/users/2, e verá uma página parecida com a da imagem abaixo.



Veja que esse texto é parecido com algo que você já estudou neste curso: objetos JS. Essa página que acabou de acessar retorna uma String com os dados do usuário 2, de um sistema fictício. Agora, precisamos acessar os dados desse sistema em nosso software e, para consegui-lo, utilizaremos o "fetch", que significa "buscar" do JS. Veja o código abaixo:

```
let dados = “”,
fetch("https://reqres.in/api/users/2").then(
    function(response) {
        response.json().then(function(dados) {
        console.log(dados);
    });
}).catch(
    function(err) {
        console.error('Falhou ao buscar dados', err);
    }
);
```

No “fetch”, passamos como argumento a URL que desejamos buscar as informações entre parenteses. O método “then”, significa “então”, será executado assim que os dados forem adquiridos. Já o “catch”, que significa “capturar”, literalmente captura o estado de erro, caso isso ocorra, impedindo o programa de travar se não conseguir buscar os dados na URL. Ou seja, estamos dizendo “busque em tal URL então (then) faça isso, mas se der errado pegue esse erro (catch) e me mostre”. Quando olhar no console, verá que os dados foram formatados como objeto do JS e consegue interagir com eles perfeitamente.

Atividade: Modifique o código anterior para exibir o e-mail do usuário em um alert. **DICA:** o e-mail está em “dados.data.email”

Eis que nossa jornada chega ao fim! Foi uma aventura longa, mas muitas habilidades foram adquiridas até aqui. Espero que seja apenas o primeiro de vários aprendizados que terá daqui para frente, e que tenha muito sucesso!

Atividade: Para concluir o curso e gerar o seu certificado, vá até a sala virtual e responda ao Questionário “Avaliação final”. Este teste é constituído por 10 perguntas de múltipla escolha, que se baseiam em todo o conteúdo estudado.

Um abraço,

Carlos Dias

Referências

GRILLO, Filipe Del Nero; FORTES, RENATA PONTIN DE MATTOS. **Aprendendo JavaScript**. São Carlos: USP, 2008.

MDN. **Primeiros passos com JavaScript**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps>. Acesso em: 9 abr. 2022.

ROSSETTO, A. G. DE M. **Apostila de Linguagem de Programação Web**. Rio Grande do Sul: Universidade Aberta do Brasil/Instituto Federal Sul-rio-grandense, 2012.

w3Schools. **JavaScript tutorial**. Disponível em: <https://www.w3schools.com/js/>. Acesso em: 9 abr. 2022.

HAVERBEKE, Marijn. **Eloquent javascript: A modern introduction to programming**. No

Starch Press, 2018.

Microsoft. **JavaScript na Microsoft**. Disponível em: <https://docs.microsoft.com/pt-br/javascript/>. Acesso em: 9 abr. 2022.

Currículo do autor

Técnico em Eletrônica(2011) e Engenheiro de Automação Industrial(2017) ambos pelo Centro Federal de Educação Tecnológica de Minas Gerais. Experiência em pesquisa e desenvolvimento na área de engenharia elétrica. Durante a graduação participou de diversas competições de robótica, além de programas de iniciação científica. Foi professor de disciplinas na área de controle e automação, como "Modelamento de sistemas de controle", "Sistemas Multivariável", "Sistemas microprocessados", "Programação de computadores" e "Segurança e confiabilidade na automação industrial". Além disso, orientou equipes do ensino médio para participar nas Olimpíadas Brasileira de Robótica, e do ensino superior em competição internacional do INPE, o Cubedesign. Atualmente é professor do ensino básico, técnico tecnológico no Instituto Federal de Minas Gerais, campus Ibirité e membro do grupo de pesquisa Robotics and Intelligent Systems - EPIIBOTS.

Currículo Lattes: http://lattes.cnpq.br/4034444219840272

Feito por professor	Data	Revisão de *layout*	Data	Versão
Carlos Dias da Silva Junior	~~xx/xx/xxxx~~01/08/2022	Desginado pela proex	~~xx/xx/xxxx~~02/08/2022	1.0

Glossário de códigos QR (*Quick Response*)

Mídia digital

Preparação do Ambiente de Trabalho

Mídia Digital

Variáveis

Dica do professor

Condições e laços de repetição

Mídia digital

Funções, objetos e JSON

Plataforma +IFMG

Formação Inicial e Continuada EaD

A Pró-Reitoria de Extensão (Proex), desde o ano de 2020, concentrou seus esforços na criação do Programa +IFMG. Esta iniciativa consiste em uma plataforma de cursos *online*, cujo objetivo, além de multiplicar o conhecimento institucional em Educação a Distância (EaD), é aumentar a abrangência social do IFMG, incentivando a qualificação profissional. Assim, o programa contribui para o IFMG cumprir seu papel na oferta de uma educação pública, de qualidade e cada vez mais acessível.

Para essa realização, a Proex constituiu uma equipe multidisciplinar, contando com especialistas em educação, *web design*, *design* instrucional, programação, revisão de texto, locução, produção e edição de vídeos e muito mais. Além disso, contamos com o apoio sinérgico de diversos setores institucionais e também com a imprescindível contribuição de muitos servidores (professores e técnico-administrativos) que trabalharam como autores dos materiais didáticos, compartilhando conhecimento em suas áreas de atuação.

A fim de assegurar a mais alta qualidade na produção destes cursos, a Proex adquiriu estúdios de EaD, equipados com câmeras de vídeo, microfones, sistemas de iluminação e isolação acústica, para todos os 18 *campi* do IFMG.

Somando à nossa plataforma de cursos *online*, o Programa +IFMG disponibilizará também, para toda a comunidade, uma Rádio *Web* Educativa, um aplicativo móvel para Android e IOS, um canal no Youtube com a finalidade de promover a divulgação cultural e científica e cursos preparatórios para nosso processo seletivo, bem como para o Enem, considerando os saberes contemplados por todos os nossos cursos.

Parafraseando Freire, acreditamos que a educação muda as pessoas e estas, por sua vez, transformam o mundo. Foi assim que o +IFMG foi criado.

O +IFMG significa um IFMG cada vez mais perto de você!

Professor Carlos Bernardes Rosa Jr.
Pró-Reitor de Extensão do IFMG