

**Pratique,
aprenda,
conquiste.**

 **Proz**
Viva sua profissão!

Disciplina | Linguagem de Programação

Téc. em Desenvolvimento de Sistemas



**Aqui
começa
a sua
jornada**

Vamos nessa?



Disciplina | Linguagem de Programação

Téc. em Desenvolvimento de Sistemas

SUMÁRIO

Linguagem De Programação.....	4
Introdução.....	4
TEMA 01.....	5
Introdução ao Javascript.....	5
TEMA 2.....	24
Variáveis Simples.....	24
TEMA 03.....	37
Operadores.....	37
TEMA 04.....	45
Estruturas Condicionais.....	45
TEMA 5.....	53
Laços de Repetição.....	53
TEMA 6.....	61
Array: Vetor (Variável Composta Unidimensional).....	61
TEMA 07.....	84
Funções.....	84
TEMA 08.....	102
Manipulando Dados.....	102
TEMA 09.....	113
Funções Assíncronas.....	113
TEMA 10.....	130
Front-End, Back-End e Full Stack.....	130

Linguagem De Programação



Introdução

A linguagem de programação é a ponte que nos permite comunicar nossas ideias e lógica para os computadores, capacitando-nos a criar aplicativos, sites, jogos e sistemas que impulsionam a tecnologia moderna. A programação é uma habilidade valiosa e versátil, que não apenas abre portas para carreiras emocionantes, mas também potencializa sua capacidade de inovação e resolução de problemas em qualquer campo. De acordo com a Brasscom (Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação), apontou recentemente que a procura por profissionais do mercado tecnológico será mais de 420 mil pessoas nos próximos 2 anos, e com os avanços tecnológicos e inovação nas linguagens de programação, está cada vez mais acessível o conhecimento em programação.

Nosso objetivo é desenvolver uma base sólida para se tornar um programador habilidoso e confiante, independentemente do seu nível de conhecimento prévio. Mergulharemos nas estruturas de controle, variáveis, estruturas de dados, algoritmos e muito mais. Aprender não apenas a escrever código, mas também a pensar de maneira lógica e analítica para resolver problemas complexos de maneira eficiente. Independentemente de você ser um iniciante curioso ou um entusiasta experiente, este curso é projetado para desafiar e expandir seus horizontes de programação.

Vamos explorar uma ampla variedade de exercícios práticos, projetos envolventes e estudos de caso do mundo real que o ajudarão a aplicar os conceitos aprendidos em situações reais. Além disso, nosso material abrangente inclui exemplos detalhados e explicações claras, para que você possa avançar em seu próprio ritmo e superar obstáculos de aprendizado. Vamos mergulhar de cabeça no universo da linguagem de programação e começar a criar um futuro de possibilidades ilimitadas!

Neste curso você encontrará os seguinte temas:

- Introdução ao Javascript.
- Variáveis Simples
- Operadores.
- Estruturas Condicionais
- Laços de Repetição
- Array: Vetor (Variável Composta Unidimensional).
- Funções
- Manipulando dados.
- Funções Assíncronas.
- Front-End, Back-End e Full Stack

TEMA 01

Introdução ao Javascript

Habilidades

- Introdução as regras JavaScript
- Utilizar ambientes de desenvolvimento de software

ONDE TUDO COMEÇOU

O Javascript 1.0 foi lançado em 1995, quando a internet ainda era uma invenção relativamente nova. O primeiro navegador da web, o Mosaic, havia sido lançado há apenas dois anos, e o HTML era a única ferramenta para construir sites - na época, o CSS ainda não tinha sido lançado, apenas em 1996 seria seu lançamento. Em vista disso, a criação de sites ainda era uma habilidade um tanto fácil de aprender, algo que era acessível não apenas a engenheiros de software, mas também a desenvolvedores e designers inexperientes.

Marc Andressen, o fundador do recém-lançado navegador Netscape, queria expandir as capacidades do navegador adicionando mais elementos dinâmicos. Ao mesmo tempo, ele queria que esses elementos estivessem disponíveis para o novo e crescente mercado de desenvolvedores amadores da web. Então, nasceu a ideia de criar uma linguagem de scripting simples e dinâmica, que um dia seria conhecida como Javascript, Andressen contratou Brendan Eich (programador cofundador do Mozilla) para criar essa linguagem de script baseada em navegador. Brendan Eich aceitou o desafio e em um artigo publicado na The First Twenty Years, de coautoria de Eich, ele disse que essa tarefa foi dificultada devido a parceria da Netscape e a Sun Microsystems, que tinham como estratégia conjunta integrar a linguagem de programação Java da Sun ao Netscape 2.0 numa tentativa de superar o Internet Explorer da Microsoft. O lançamento da versão beta do Netscape 2.0 estava programado para setembro de 1995, Brendan Eich sentiu a necessidade de agir rapidamente lançando o protótipo inicial como nome de "Mocha" em 10 de maio de 1995, com algumas diversidades na versão, como a correção de bugs, resposta às solicitações de recursos e projetar APIs que possibilitam a interação do Mocha com o Netscape.

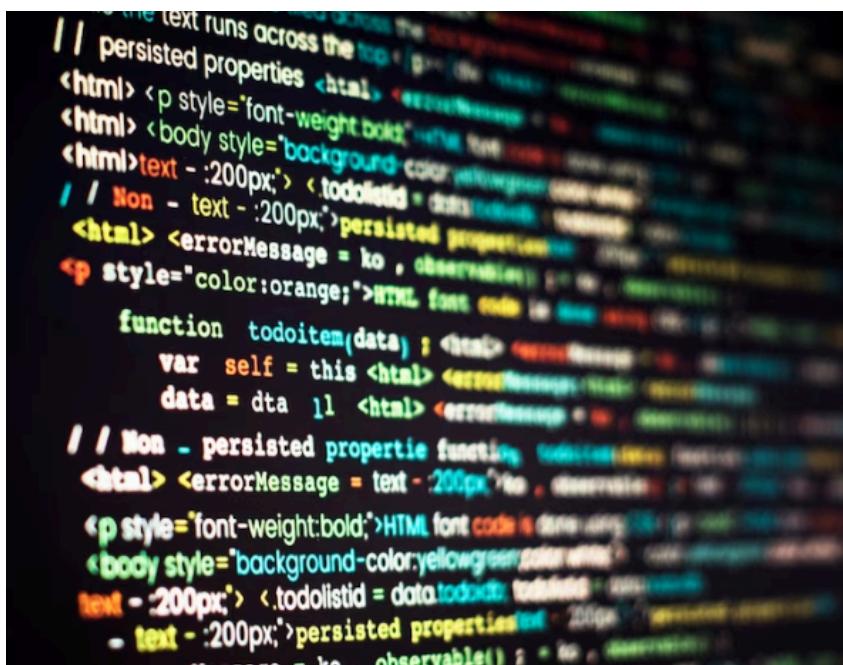
Essas APIs estabeleceram as bases para o que seria conhecido como DOM (Document Object Model), uma interface usada na interação com as linguagens HTML e XML. Essa década foi o período de revolução, pois os navegadores eram estáticos, sem interação com o usuário, com o lançamento do javascript isso mudou, proporcionando uma interatividade Cliente-Servidor que protagonizou o início de uma nova era da WWW. O javascript é usado principalmente no navegador, onde permite os desenvolvedores a manipulação de conteúdo das páginas web através do DOM, por esse entre outros vários motivos o Javascript é uma das linguagens mais utilizadas no mundo, com o crescimento e melhoria de desempenho das APIs disponíveis no navegadores.

Em novembro de 1996, a Netscape se uniu com a ECMA International para definir regras e

padrões para o javascript, e desde então o javascript padronizado é chamado de ECMAScript e com a especificação ECMA-262, documento que rege a normalização da linguagem. Ao contrário de que muitos imaginam o javascript não possui semelhanças ou referência da linguagem java, o javascript e java não são a mesma coisa, não existe nenhuma relação entre as duas linguagens.

JAVASCRIPT

O JavaScript é uma linguagem de programação extremamente versátil, sendo capaz de funcionar em diversas plataformas, o que significa que não está limitada apenas ao ambiente da web. Um exemplo notável desse amplo alcance é a aplicação do JavaScript em motores gráficos avançados, como o Unity 3D, indo além do contexto tradicional da web.



Disponível em: <rawpixel-<https://tinyurl.com/5znz9xdt>>. Acesso em 08 out. 2023.

Além disso, o JavaScript é dotado de uma biblioteca padrão de objetos, acompanhada por um conjunto fundamental de elementos que constituem o cerne da linguagem. Esses elementos incluem operadores, estruturas de controle e declarações. Esse núcleo, por sua vez, pode ser expandido e adaptado para atender a uma vasta gama de finalidades, permitindo que os desenvolvedores personalizem e estendam a funcionalidade da linguagem conforme as necessidades específicas de seus projetos. Isso torna o JavaScript uma linguagem altamente flexível e adaptável, capaz de se adequar a uma diversidade de cenários de desenvolvimento, muito além de sua aplicação inicial na web. O JavaScript é uma linguagem de programação extremamente versátil, sendo capaz de funcionar em diversas plataformas, o que significa que não está limitada apenas ao ambiente da web. Um exemplo notável desse amplo alcance é a aplicação do JavaScript em motores gráficos avançados, como o Unity 3D, indo além do contexto tradicional da web.

Além disso, o JavaScript é dotado de uma biblioteca padrão de objetos, acompanhada por um conjunto fundamental de elementos que constituem o cerne da linguagem. Esses elementos

incluem operadores, estruturas de controle e declarações. Esse núcleo, por sua vez, pode ser expandido e adaptado para atender a uma vasta gama de finalidades, permitindo que os desenvolvedores personalizem e estendam a funcionalidade da linguagem conforme as necessidades específicas de seus projetos. Isso torna o JavaScript uma linguagem altamente flexível e adaptável, capaz de se adequar a uma diversidade de cenários de desenvolvimento, muito além de sua aplicação inicial na web.

Podemos dividir esse núcleo em duas formas e, vamos entender melhor como funcionam:

Front-end e back-end são dois termos essenciais no desenvolvimento de software e na criação de sites e aplicativos. Eles se referem a diferentes aspectos e camadas do sistema, cada um desempenhando um papel fundamental na experiência do usuário e na funcionalidade do software.



Disponível em: <[storyset-https://tinyurl.com/nh9u56fv](https://tinyurl.com/nh9u56fv)>. Acesso em 08 out. 2023.

Acesse o QR Code do vídeo Classes em Java - Fonte do vídeo: <<https://tinyurl.com/2bzu4cem>>

O front-end, também conhecido como Client-side (lado do cliente), é a parte visível e interativa de um aplicativo ou site com a qual os usuários interagem diretamente. Ele abrange o design, a interface do usuário (UI), a disposição de elementos gráficos, a usabilidade e a experiência do usuário. Os desenvolvedores front-end são responsáveis por traduzir o design e as funcionalidades em código, garantindo que o usuário final tenha uma experiência agradável e intuitiva. As tecnologias comuns utilizadas no front-end incluem HTML (linguagem de marcação), CSS (folhas de estilo em cascata) e JavaScript.

O back-end, também conhecido como Server-side (lado do servidor), por outro lado, é a

parte do sistema que opera nos bastidores e lida com a lógica de negócios, processamento de dados, armazenamento e gerenciamento de informações. Os desenvolvedores back-end são responsáveis por criar e manter os servidores, bancos de dados e aplicativos que suportam as funcionalidades do front-end. Eles lidam com a segurança dos dados, a autenticação de usuários, o processamento de formulários, a comunicação com APIs (interfaces de programação de aplicativos) e outras operações complexas que não são diretamente visíveis para o usuário.

Em resumo, enquanto o front-end trata da apresentação visual e da interação com o usuário, o back-end lida com a lógica e o processamento dos dados que permitem que o aplicativo funcione corretamente. Ambas as partes são igualmente importantes para a criação de sistemas eficazes e funcionais, e os desenvolvedores front-end e back-end frequentemente colaboram para construir experiências completas e coesas para os usuários.

O documento JavaScript, muitas vezes referido como "script" ou "arquivo de script", desempenha um papel crucial no desenvolvimento de páginas web interativas e dinâmicas. Ele é um arquivo contendo código JavaScript que é incorporado ou vinculado a uma página HTML. A importância do documento JavaScript reside em sua capacidade de adicionar interatividade, funcionalidade e comportamento à página, melhorando significativamente a experiência do usuário.

Algumas razões pelas quais o documento JavaScript é fundamental:

Interatividade do Usuário: O JavaScript permite criar elementos interativos na página, como botões que respondem a cliques, formulários que validam dados antes do envio e elementos de interface que mudam com base nas ações do usuário. Isso torna a experiência do usuário mais envolvente e responsiva.



Disponível em: <[freepik-https://tinyurl.com/yc3k9fht](https://tinyurl.com/yc3k9fht)>. Acesso em 08 out. 2023.

Manipulação do DOM: O Document Object Model (DOM) é uma representação da estrutura da página em forma de árvore. O JavaScript pode ser usado para acessar, modificar e manipular elementos do DOM em tempo real, alterando conteúdo, estilo e comportamento da página sem a necessidade de recarregar a página inteira.

Requisições Assíncronas: O JavaScript permite que as páginas façam solicitações assíncronas

ao servidor, obtendo ou enviando dados sem a necessidade de recarregar a página. Isso é fundamental para a criação de aplicativos web dinâmicos, como feeds de notícias em tempo real ou sistemas de mensagens.

Validação de Dados: Com JavaScript, é possível validar dados inseridos pelos usuários em formulários antes de enviá-los ao servidor. Isso ajuda a melhorar a qualidade dos dados recebidos e a oferecer uma experiência de usuário mais amigável.

Manipulação de Cookies e Armazenamento Local: JavaScript permite manipular cookies no navegador do usuário, o que é útil para rastreamento de sessões e armazenamento temporário de informações. Além disso, com o uso de APIs modernas, como Web Storage ou IndexedDB, é possível armazenar dados localmente no navegador.

Carregamento Dinâmico de Conteúdo: JavaScript é frequentemente usado para carregar conteúdo adicional em uma página sem recarregar a página inteira. Isso é especialmente útil para atualizações parciais de conteúdo em sites e aplicativos de página única (SPA).

Animações e Efeitos Visuais: Com JavaScript, é possível criar animações e efeitos visuais impressionantes, como sliders, carrosséis, transições suaves e muito mais, tornando a aparência da página mais atrativa.

Em resumo, o documento JavaScript é uma ferramenta poderosa que capacita os desenvolvedores a criar páginas web mais interativas, responsivas e funcionais. Ele desempenha um papel fundamental em aprimorar a experiência do usuário, fornecendo recursos que vão além das capacidades estáticas do HTML e do CSS.

Toda linguagem de programação possui um documento que explica e exemplifica os comandos que possuem as linguagens de programação. É recomendado que saibamos onde encontrar esse documento quando necessário.



Para acessar o documento de JavaScript Acesse o QR Code do vídeo Classes em Java - Fonte do vídeo: <https://www.w3schools.com/js/>

Sintaxe

A sintaxe JavaScript é o conjunto de regras, como os programas JavaScript são construídos. Assim como nas línguas humanas, seguir a sintaxe correta é fundamental para que o computador comprehenda e execute suas instruções corretamente. A sintaxe do JavaScript é relativamente flexível, mas há regras que devem ser seguidas para garantir que seu código seja válido e funcional.

Você pode usar as palavras-chave var, let ou const para declarar variáveis. A declaração é seguida pelo nome da variável, um sinal de = (igual), e o valor que você deseja atribuir. Lembre-se de que as variáveis podem conter letras, números e sublinhados, mas não podem começar com um número.

Os comentários são usados para explicar o código e não são executados pelo computador. Comentários de linha única começam com //, enquanto comentários de várias linhas são

delimitados por /* e */.

Uma instrução é uma ação que o computador deve executar. Instruções podem ser uma única linha de código ou blocos de código delimitados por chaves {}. Expressões são avaliadas para produzir um valor. As instruções podem conter expressões.

As estruturas de controle, como if, for e while, permitem que você controle o fluxo do seu programa. Elas são usadas para executar blocos de código condicionalmente ou repetidamente.

Funções são blocos de código reutilizáveis que podem ser chamados com um nome. Elas podem receber parâmetros (entradas) e retornar valores.

Em JavaScript, objetos são coleções de pares chave-valor. Eles são usados para agrupar informações relacionadas.

Arrays são listas ordenadas de valores, usados para armazenar múltiplos itens em uma única variável.

Em JavaScript existem diversos operadores para realizar operações matemáticas, de comparação e lógicas.

Esses são apenas alguns dos elementos básicos da sintaxe do JavaScript. À medida que você se familiarizar com esses conceitos, poderá criar códigos mais complexos e eficientes para realizar uma ampla variedade de tarefas.

Os comandos que fazem parte da biblioteca do JavaScript são em inglês, por uma série de razões históricas, técnicas e práticas. Algumas das principais razões pelas quais a linguagem de programação JavaScript e seus comandos foram desenvolvidos em inglês tem relação com as origens históricas. Como já vimos a linguagem JavaScript foi criada nos anos 90 na Netscape Communications Corporation por Brendan Eich. Na época, a programação em inglês era dominante na comunidade de desenvolvimento de software e na indústria de tecnologia como um todo.

O inglês também é amplamente reconhecido como uma língua internacional de comunicação, como padrão internacional, especialmente na área de tecnologia da informação. Ter comandos em inglês facilita a colaboração entre desenvolvedores de diferentes partes do mundo, permitindo que eles compartilhem código e recursos mais facilmente.

Muitas das tecnologias e protocolos da web são originalmente em inglês, como o próprio HTML e os URLs (Uniform Resource Locators). Ao manter os comandos em inglês, a linguagem JavaScript se integra de forma mais eficiente com outras tecnologias da web e retém compatibilidade e interoperabilidade.

O inglês é usado amplamente em documentação, tutoriais e recursos de aprendizado. Manter os comandos em inglês torna mais fácil para os desenvolvedores entenderem e aprenderem a linguagem, uma vez que muitos recursos educacionais estão disponíveis neste idioma, tornando maior consistência e clareza. Também é suportado por padrão em muitos sistemas operacionais e ambientes de desenvolvimento. Isso simplifica a implementação da linguagem em diferentes plataformas, facilitando a implementação.

Muitas bibliotecas e frameworks populares em JavaScript são desenvolvidos em inglês. Usar os mesmos nomes e convenções facilita a integração dessas ferramentas no desenvolvimento. Manter os comandos em inglês ajuda a linguagem a evoluir e se adaptar às mudanças tecnológicas, garantindo uma linguagem coesa e consistente ao longo do tempo.

Embora a escolha de ter comandos em inglês possa criar uma barreira inicial para algumas pessoas que não são fluentes nesse idioma, essa abordagem globalmente aceita e utilizada, tem demonstrado vantagens significativas para a comunidade de desenvolvimento como um todo.

O foco deste módulo é trabalhar somente sobre a linguagem JavaScript, mas para se tornar um bom desenvolvedor de aplicativos móveis, é preciso aprender as principais linguagens de programação. Tornar-se um desenvolvedor de aplicativos móveis envolve a aquisição de habilidades específicas para criar aplicativos que funcionem em dispositivos móveis, como smartphones e tablets. Aqui estão as principais áreas de conhecimento e habilidades que você precisa aprender para se tornar um desenvolvedor de sucesso para dispositivos móveis:

Linguagens de Programação



Disponível em: <svstudioart-<https://tinyurl.com/yrk7ssd9>>. Acesso em 08 out. 2023.

Você precisará aprender as linguagens de programação usadas para desenvolver aplicativos móveis. As principais linguagens incluem:

Swift: Usado para desenvolver aplicativos iOS (iPhone e iPad).

Kotlin: Usado para desenvolver aplicativos Android.

JavaScript: Usado com frameworks como React Native e NativeScript para desenvolvimento multiplataforma.

Dart: Usado para desenvolver aplicativos com o framework Flutter.

Plataformas e Ambientes de Desenvolvimento:

Familiarize-se com as plataformas e ambientes de desenvolvimento específicos para cada sistema operacional:

Xcode: Ambiente de desenvolvimento integrado (IDE) para iOS.

Android Studio: IDE para desenvolvimento de aplicativos Android.

Visual Studio Code: Pode ser usado para desenvolver aplicativos multiplataforma com frameworks como React Native e Flutter.

Frameworks e Bibliotecas:

Aprenda frameworks e bibliotecas relevantes para o desenvolvimento móvel, como:

React Native: Para desenvolvimento multiplataforma usando JavaScript e React.

Flutter: Para desenvolvimento multiplataforma usando a linguagem Dart.

NativeScript: Para desenvolvimento multiplataforma usando JavaScript e tecnologias web.

Design Responsivo e UI/UX:

Entenda os princípios de design responsivo e criação de interfaces de usuário (UI) e experiências do usuário (UX) eficazes para dispositivos móveis.

Consumo de APIs e Dados:

Aprenda a integrar aplicativos com serviços externos, consumir APIs e lidar com dados provenientes da web.

Armazenamento de Dados Local:

Saiba como armazenar dados localmente no dispositivo, como bancos de dados SQLite e armazenamento local.

Testes e Depuração:

Desenvolva habilidades em testes unitários, testes de integração e depuração de aplicativos para garantir que funcionem corretamente.

Publicação e Distribuição:

Aprenda como preparar, compilar e publicar seu aplicativo nas lojas de aplicativos, como a App Store da Apple e o Google Play Store.

Atualizações e Manutenção:

Entenda como manter e atualizar seus aplicativos para fornecer correções de bugs, melhorias e novos recursos.

Tendências e Novas Tecnologias:

Fique atualizado com as tendências atuais e emergentes em desenvolvimento móvel, como inteligência artificial, realidade aumentada, aprendizado de máquina e mais.

Lembrando que o desenvolvimento para dispositivos móveis pode envolver escolhas entre desenvolvimento nativo (usando as linguagens e ferramentas específicas de cada plataforma) ou desenvolvimento multiplataforma (usando frameworks que permitem criar um único código para várias plataformas). Portanto, suas escolhas podem depender do seu interesse, das demandas do mercado e das suas preferências pessoais.

Ambiente de desenvolvimento

Para iniciar nossa jornada na criação de aplicações com JavaScript, é essencial que tenhamos à nossa disposição um ambiente de programação adequado. Existem diversas opções de editores de código disponíveis, cada um com suas características e vantagens. Algumas das alternativas populares incluem o Sublime, Atom, Notepad++, VScode, entre outros.

No entanto, para este projeto em particular, optamos por utilizar o editor code.org. A escolha desse editor se deve a uma série de razões que tornam nossa experiência de programação mais acessível e eficaz. O code.org se destaca por ser uma ferramenta de programação extremamente amigável, especialmente para iniciantes. Ele é acessível online e, o melhor de tudo, é gratuito.

Uma das características notáveis do code.org é sua poderosa funcionalidade que permite a criação de aplicativos e jogos de forma intuitiva. O editor oferece uma variedade de laboratórios e

recursos que simplificam o processo de escrever códigos em JavaScript, tornando-o mais acessível mesmo para aqueles que estão começando a aprender a programar.

Além disso, o code.org inclui seu próprio emulador, o que é um grande benefício para nosso projeto. Esse emulador nos permite realizar testes e experimentações diretamente no ambiente de desenvolvimento, facilitando a depuração e verificação do funcionamento de nossos programas. Em última análise, essa escolha do code.org como nosso editor de código para esta jornada de criação de aplicações JavaScript promete simplificar e aprimorar nossa experiência de programação.

Podemos criar aplicativos, animações, jogos, entre outros utilizando o editor de código do code.org online, sem a necessidade de configuração do ambiente.

Para criar a conta, no canto superior direito da tela clique em Entrar.



Fonte: Elaborado pelo autor (2023).

Insira seu melhor email e uma senha segura ou entre com uma das contas, que você já possui sugeridas abaixo:

Já tens uma conta? Inicia sessão

The screenshot shows the login interface for code.org. On the left, there are fields for 'Nome utilizador ou e-mail' and 'Palavra-passe'. Below the password field is a link 'Esqueceste-te da palavra-passe?'. Underneath the password field are two buttons: 'Iniciar sessão' (purple) and 'Criar uma conta' (white). To the right, there is a section titled 'OU' (Or) with three social sign-in options: 'Continuar com Google' (green), 'Continuar com Microsoft' (yellow), and 'Continuar com Facebook' (blue). Above these options is a placeholder 'Insere o teu código de secção de 6 letras' with a 'Código (ABCDEF)' input field and a 'Go' button.

Fonte: Elaborado pelo autor (2023)

Ao criar uma conta não é necessário utilizar um email real, poderá ser um fictício com no exemplo a seguir:

Inscreve-te na Code.org

Cadastre-se para ter uma conta e acompanhar seu progresso ou gerenciar sua sala de aula. [Você pode explorar diversas fases e desafios](#) sem uma conta, mas precisará fazer login para salvar seu progresso e seus projetos.

Já criaste uma conta? [Iniciar sessão](#)

G Continuar com Google

M Continuar com Microsoft

f Continuar com Facebook

OU

Regista-te com o teu endereço de email

E-mail *	aluno@proz.com
Palavra-passe	*****
Confirmação da palavra-passe	*****

Criar conta

* Os endereços de email não são armazenados num formulário que nos permita

Fonte: Elaborado pelo autor (2023).

Para finalizar preencha os dados abaixo:

Terminar de criar a tua conta

Preencha as informações a seguir para concluir a criação da sua conta Code.org para [aluno@proz.com](#). [Cancelar](#)

Tipo de conta	Aluno Explore todos os nossos cursos e atividades, e ainda: <ul style="list-style-type: none"> Salvar seu progresso e projetos Junte-se à secção de sala de aula do seu professor 	Professor Tudo a que os alunos têm acesso, e ainda: <ul style="list-style-type: none"> Criar secções de sala de aula Atribuir tarefas e acompanhar o trabalho dos alunos Matricular-se em aprendizagem profissional Interagir no nosso fórum de professores
<input type="checkbox"/> Sou pai/responsável, atuando em nome do(a) meu(minha) filho(a)		
Nome de exibição (ex: João da Silva ou Maria Alves)		
Idade		
Género (opcional)		

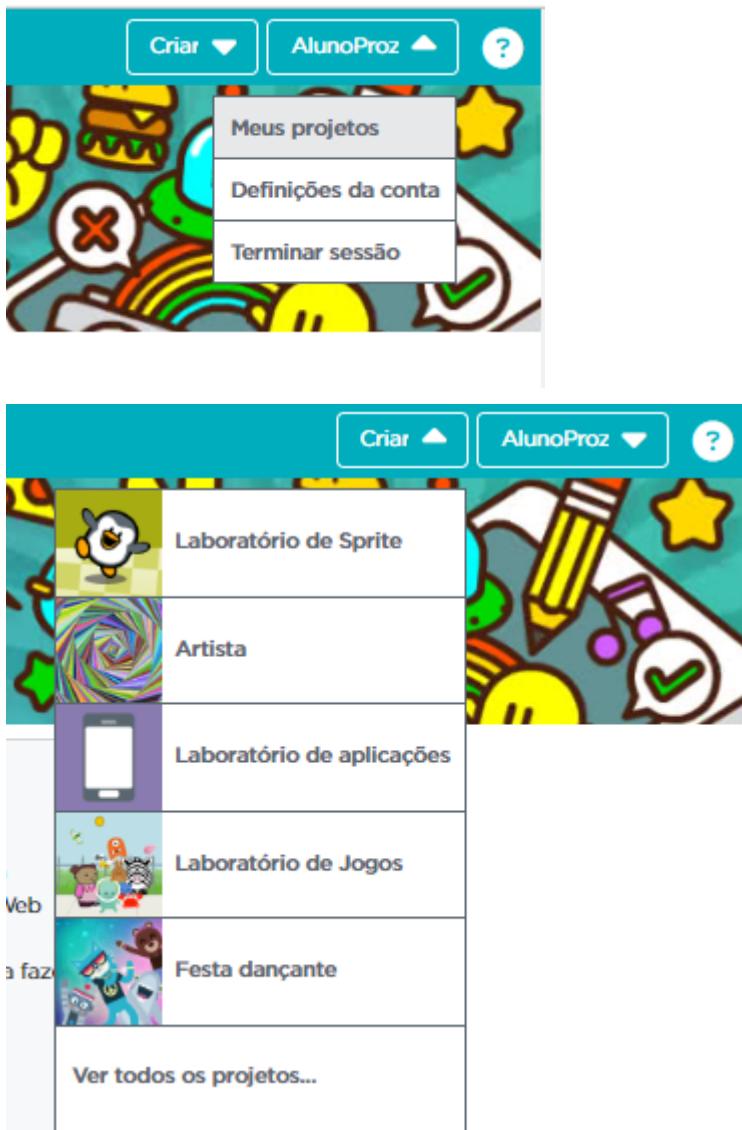
Ao se inscrever no Code.org, você concorda com nossos [Termos de Serviço](#) e [Política de Privacidade](#).

Ir para a minha conta

Fonte: Elaborado pelo autor (2023).

Coloque idade de 21+, mesmo se não tiver, para que o site não bloqueeie acesso aos laboratórios.

Depois de criar sua conta é possível acessar, no canto direito da tela, os laboratórios clicando na palavra criar, e os seus projetos ficarão salvos na área de projetos, clicando no seu nome de usuário você poderá encontrar, entre outras opções.



Fonte: Elaborado pelo autor (2023)

Iniciando o desenvolvimento

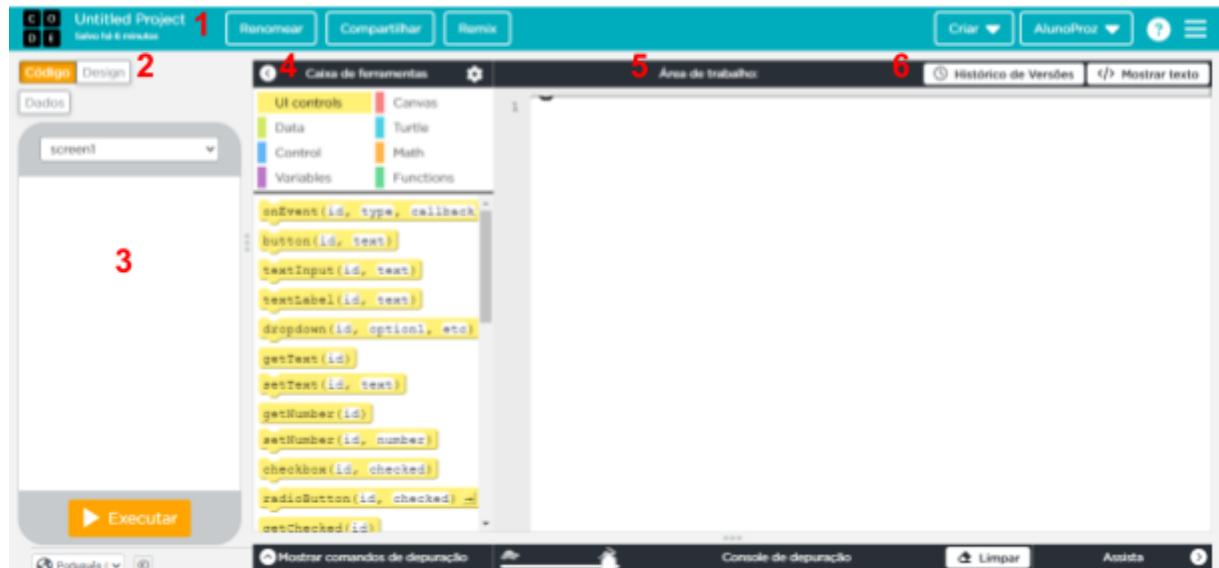
O Code.org é uma organização sem fins lucrativos e site homônimo liderado por Hadi e Ali Partovi, em 2013, que visa incentivar as pessoas, principalmente estudantes de unidades de ensino básico, a aprender programação.

Vamos começar acessando o laboratório de aplicações para criar um aplicativo simples usando JS. Clique na palavra criar e selecione Laboratório de aplicações:



Fonte: Elaborado pelo autor (2023).

No laboratório teremos a seguinte Visão:

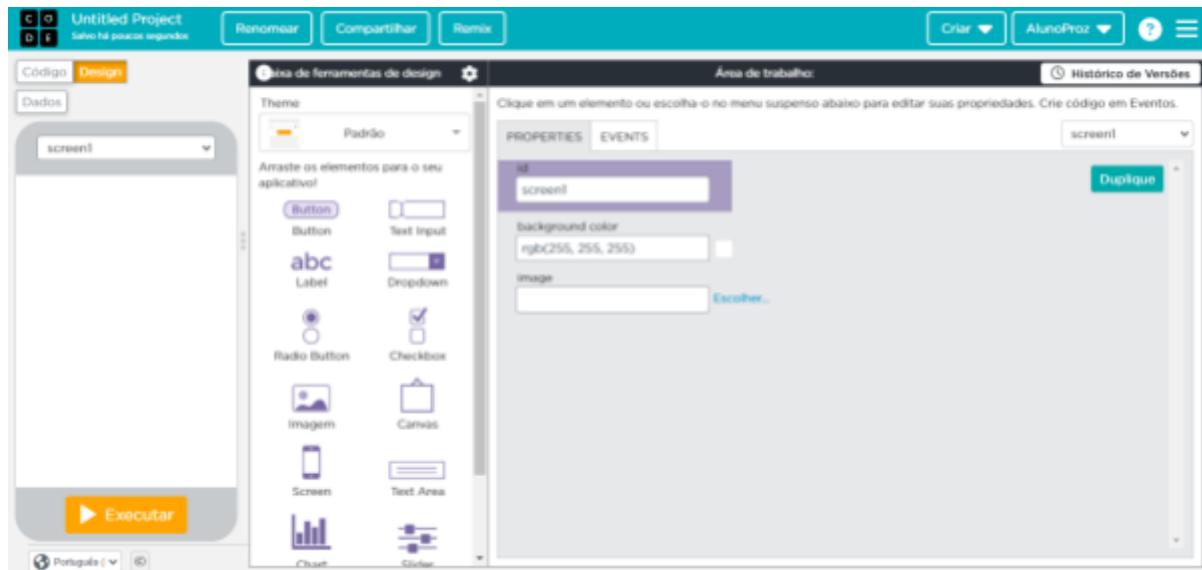


Fonte: Elaborado pelo autor (2023).

1. RENOMEAR/COMPARTILHAR/REMIX- botões usados para configurações do projeto;
2. CÓDIGO/DESIGN/DADOS- botões usados para mudar a função do laboratório;
3. EMULADOR- dispositivo móvel que usaremos para desenvolver e testar nossas aplicações;
4. CAIXA DE FERRAMENTAS- onde encontraremos os blocos de programação JS;

5. ÁREA DE TRABALHO- espaço para criação do código;
6. HISTÓRICO DE VERSÕES/ MOSTRAR TEXTO- botões para ver o histórico de edição dos código, e mudar a versão de blocos para texto.

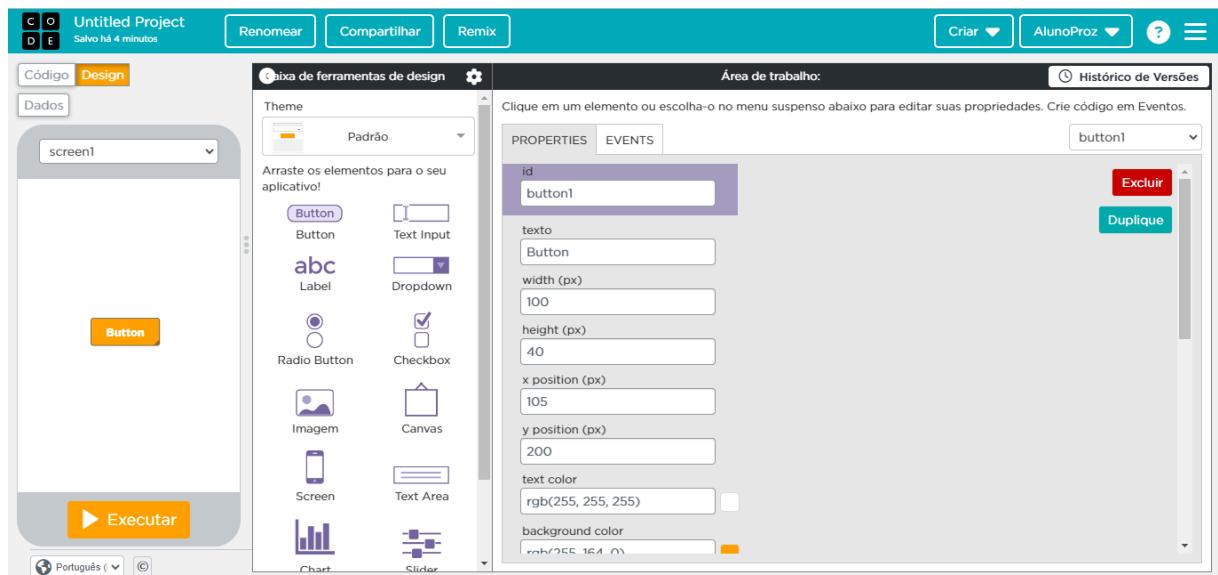
O design



Fonte: Elaborado pelo autor (2023).

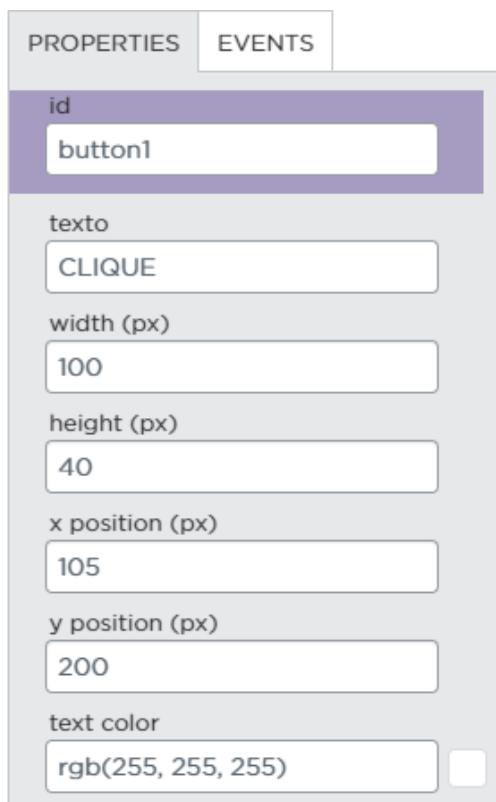
O desafio é criar um aplicativo simples onde um botão terá a função de modificar a cor de fundo da tela. Para isso é preciso criar o design do aplicativo clicando no botão Design no canto esquerdo da tela:

A caixa de ferramentas e a área de trabalho agora estão modificadas para função design. Na caixa de ferramentas encontramos os elementos de design necessários para criar um aplicativo, vamos precisar do button. Clique, segure, arraste e solte o botão dentro do emulador em qualquer lugar que desejar. Desta Maneira:

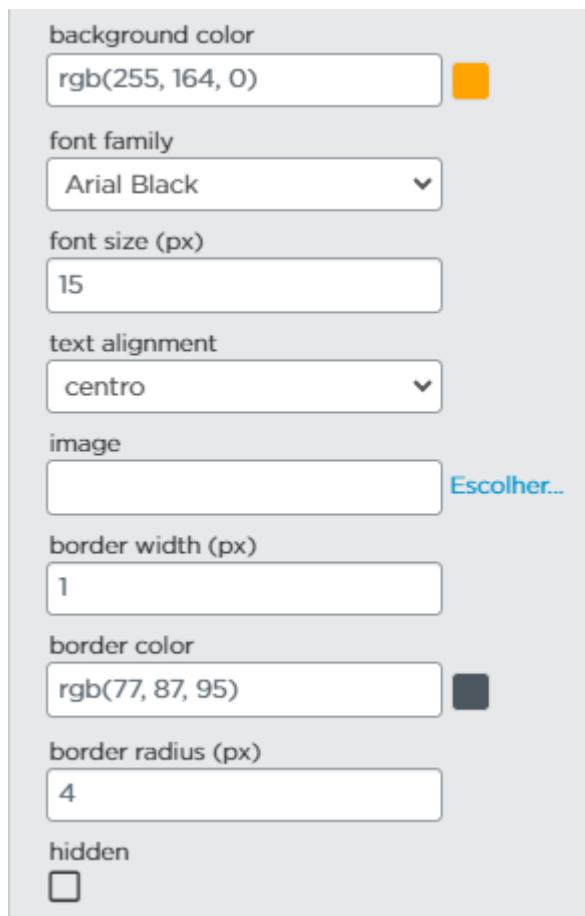


Fonte: Elaborado pelo autor (2023).

A área de trabalho agora mostra as propriedades dos elementos que podem ser alteradas. Mude a propriedade *text* do button para uma palavra que chame a atenção do usuário para clicar no botão. Altere a cor de fundo do button na propriedade *background color*. Poderá testar outras propriedades para descobrir o que elas são responsáveis.



Fonte: Elaborado pelo autor (2023).

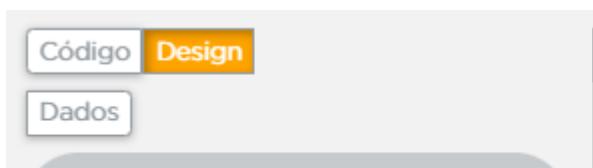


Depois que alterar as propriedades do botão lembre-se de verificar o id do elemento. O ID precisa ser único no documento, e geralmente é utilizado para identificar o elemento que será responsável por alguma funcionalidade do aplicativo. Volte para a tela de código clicando na palavra código no canto esquerdo da tela.

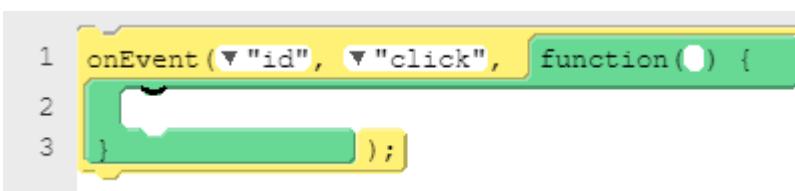
Fonte: Elaborado pelo autor (2023).

O código

Para escrever o código é preciso clicar e arrastar os blocos que estão na caixa de ferramentas, para dentro da área de trabalho, é importante o encaixe correto e a sequência para que o código funcione, além da escrita correta. Vamos criar um evento de click, para que quando o botão for clicado alterar a cor de fundo da tela para vermelho.

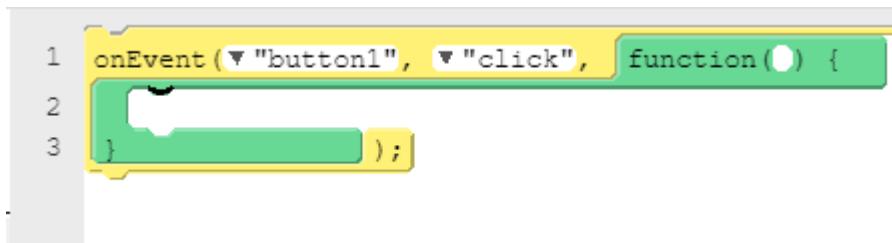


Fonte: Elaborado pelo autor (2023).



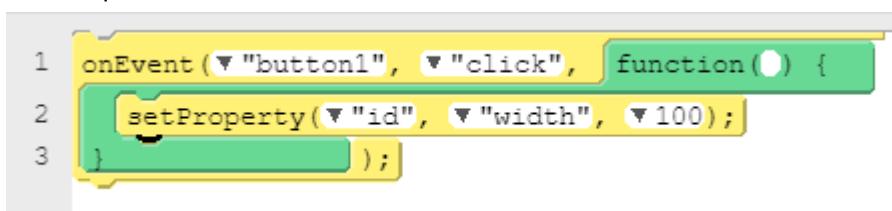
Fonte: Elaborado pelo autor (2023).

O primeiro bloco `onEvent`, que fica na sessão UI CONTROLS, marca o evento de click, dentro dele é necessário incluir o id do elemento que será responsável e a ação, que será click:



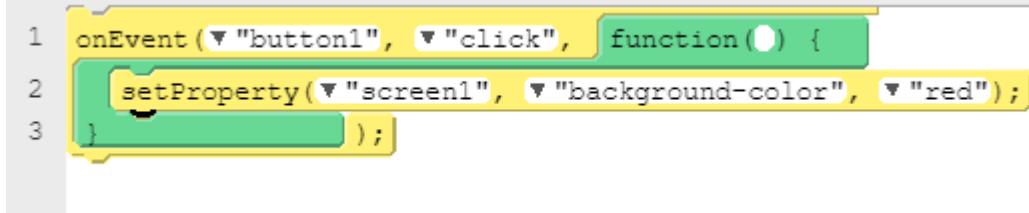
Fonte: Elaborado pelo autor (2023).

A palavra *function* significa "faça", todo comando que for incluído dentro do bloco `onEvent` será executado quando o botão for clicado.



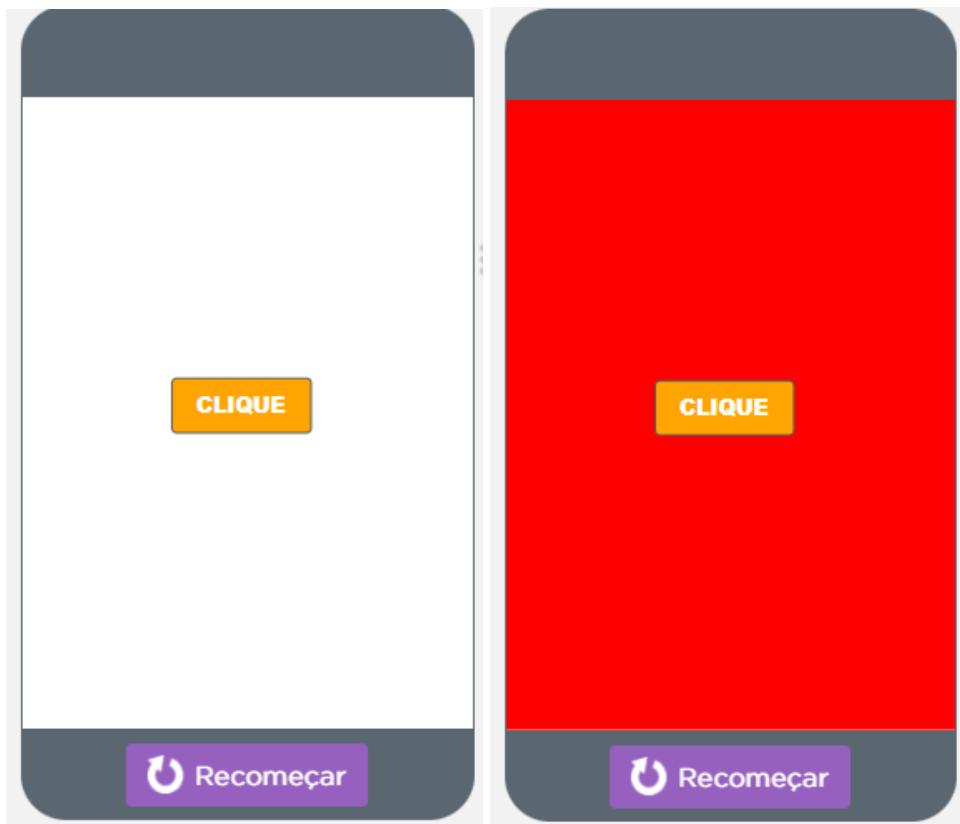
Fonte: Elaborado pelo autor (2023).

O comando que queremos encaixar no `onEvent` é o `setProperty`, que significa “altere a propriedade”, fica na sessão UI CONTROLS. Dentro dele é necessário acrescentar o id do elemento que será alterado, no caso a tela id “`screen1`”, a propriedade que será alterada, no caso “`background-color`”, e o valor, no caso a cor vermelho, colocada em inglês, “`red`”, todos parâmetros são escritos entre aspas, pois é uma regra da linguagem JS.



Fonte: Elaborado pelo autor (2023).

Clique no botão EXECUTAR do emulador e teste o aplicativo simples que acabamos de montar



Fonte: Elaborado pelo autor (2023).

Se caso algo estiver errado, será apontado no código qual a linha que contém erro de sintaxe, ou com um quadrado vermelho ou um triângulo amarelo. Se passar o cursor do mouse em cima do triângulo ou quadrado verá o erro identificado no formato de um balão.

```

1 onEvent(button, click, function() {
2     The onEvent() id parameter refers to an id ("button") which does not exist.
3 })

```

Comandos de depuração

Console de depuração

WARNING: Line: 1: The onEvent() id parameter refers to an id ("button") which does not exist.

Fonte: Elaborado pelo autor (2023).

Observe o console no canto inferior da tela e o erro estará descrito, você poderá pesquisar qual o erro encontrado. Se não souber inglês, usando o google tradutor poderá traduzir.

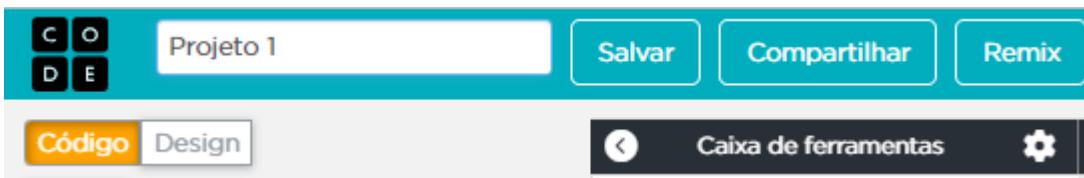
Neste exemplo é apontado erro no parâmetro id, dado como não existente. Verifique se o id está corretamente escrito, para relembrar o id do elemento volte na sessão design e clique no elemento para rever suas propriedades.

```
button1
1 onEvent ("button", "click", function () {
2    setProperty ("screen1", "background-color", "red");
3 })
```

Fonte: Elaborado pelo autor (2023).

Ajuste o código e teste novamente.

Clique em renomear no canto superior esquerdo, atribua um nome ao primeiro projeto e clique em salvar.



Fonte: Elaborado pelo autor (2023).



RESUMO

Neste tema você aprendeu que a linguagem JavaScript é uma linguagem versátil e indispensável para o desenvolvimento web moderno. Sua capacidade de adicionar interatividade, dinamismo e funcionalidade a sites e aplicativos o torna uma ferramenta poderosa nas mãos dos desenvolvedores. Seus comandos são em inglês e as regras de sintaxe estão descritas em um documento que é necessário ter conhecimento.

Também conhecemos um ambiente de programação interativo onde construímos um aplicativo simples e que pode ser codificado por js, além da criação do design e testes no emulador.



ATIVIDADE DE FIXAÇÃO

1. Qual é a importância do Js, e para que finalidade ele foi criado?
2. Pesquise e descreva quais as principais regras de sintaxe do JS.
3. Crie um tutorial com 4 passos para criação de aplicativo:
4. Recrie o aplicativo simples agora usando 4 botões e 4 cores diferentes
5. Qual o comando utilizado em JS para criar um evento?
6. Pesquise quais eventos podem ser utilizados na criação de aplicativos móveis
7. Explique o que é a Sintaxe do código e elenque as principais regras de diagramação do js.

8. O que significa o comando `setProperty` ?
9. Mude para o modo texto clicando no botão mostrar texto no canto superior direito da tela, reescreva o código digitalmente e teste.
10. Analise o código abaixo e identifique

```
1 onEvent("buttonClick", "click", function() {  
2     setProperty("screen1", "background-color", "yellow");  
3 } );  
4
```

- a. o id do botão
- b. o parâmetro de propriedade que vai ser alterada
- c. qual o valor que receberá a screen1
- d. explique com suas palavras o que irá acontecer quando o código for executado

TEMA 2

Variáveis Simples

Habilidades:

- Declarar e atribuir valor a variáveis
- Manipular dados na programação;
- Aplicação mobile variáveis na programação js;



Disponível em: <[freepik-https://tinyurl.com/365zdj7c](https://tinyurl.com/365zdj7c)>. Acesso em 08 out. 2023.

Variáveis em JavaScript são como containers ou espaços de armazenamento que permitem que os programadores guardem e manipulem dados de diferentes tipos. Elas permitem que os programadores armazenem informações temporárias ou permanentes e, em seguida, utilizem esses valores em cálculos, exibições e processos de tomada de decisão dentro do código.

Existem alguns tipos de variáveis no JavaScript, que podem conter diferentes tipos de

dados, como números inteiros e de ponto flutuante, strings (textos), arrays, objetos e muito mais. Imagine uma caixa onde você pode colocar coisas diferentes: livros, brinquedos, roupas etc. Cada item tem um nome, certo? As variáveis funcionam de maneira semelhante. Elas têm nomes exclusivos que você escolhe para identificar o tipo de informação que irá armazenar. Por exemplo, você pode ter uma variável chamada "idade" para armazenar a idade de uma pessoa ou uma variável chamada "mensagem" para armazenar uma saudação.

A magia acontece quando você atribui um valor a uma variável. Isso significa que você está colocando um conteúdo específico dentro do contêiner. Uma vez que um valor é atribuído, você pode reutilizá-lo, modificá-lo, combiná-lo com outros valores e muito mais. É como se você pudesse pegar coisas da sua caixa mágica e fazer coisas incríveis com elas!

As variáveis em JavaScript são criadas usando palavras-chave como var, let ou const, seguidas pelo nome que você escolhe para a variável. Depois de criar uma variável, você pode usá-la para armazenar informações temporárias, fazer cálculos, controlar o fluxo do programa e muito mais.

Neste tema entraremos em uma jornada de aprendizado sobre JavaScript, vamos explorar as nuances das variáveis, como declará-las, atribuir valores, entender seus escopos e tipos de dados. À medida que nos aprofundamos, você descobrirá que as variáveis são os blocos de construção essenciais para criar aplicativos interativos e dinâmicos. Portanto, prepare-se para desvendar o poder das variáveis e levantar o véu do mundo da programação em JavaScript!

Declaração

A ação de declarar uma variável em um programa desempenha um papel de extrema relevância, uma vez que sua omissão pode resultar em erros e inconsistências potenciais no código. É amplamente considerado uma prática recomendada na programação sempre efetuar a declaração de variáveis antes de utilizá-las em qualquer parte do programa.

Entretanto, é importante destacar que o JavaScript apresenta uma particularidade que não é comum a todas as linguagens de programação, mas que desempenha um papel significativo na forma como o código é interpretado. Essa característica é conhecida como "Case Sensitive". Em outras palavras, o JavaScript diferencia entre letras maiúsculas e minúsculas no nome das variáveis, o que significa que variáveis com nomes semelhantes, mas com diferenças de maiúsculas e minúsculas, são tratadas como variáveis distintas.

Por exemplo, se declararmos uma variável com o nome "Média" e outra com o nome "média" em algumas linguagens de programação, essas variáveis seriam reconhecidas como a mesma entidade. No entanto, no JavaScript, essas variáveis são interpretadas como entidades separadas devido à sensibilidade ao uso de letras maiúsculas e minúsculas. Isso tem implicações importantes para a identificação e utilização de variáveis no JavaScript, uma vez que o programador deve estar ciente de que nomes de variáveis que diferem apenas no uso de letras maiúsculas ou minúsculas são tratados como variáveis distintas pelo interpretador da linguagem. Portanto, a atenção aos detalhes na nomenclatura das variáveis é essencial para garantir o funcionamento adequado dos programas em JavaScript.

Variáveis JavaScript podem ser declaradas de 4 maneiras:

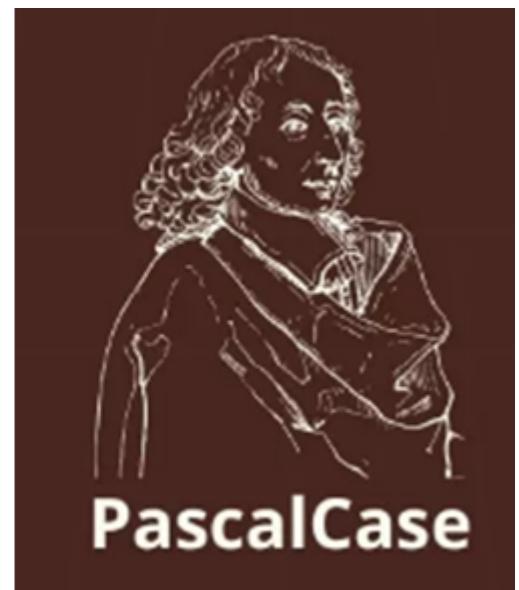
- Automaticamente
- Usando var- Faz a declaração de uma variável, opcionalmente, inicializando-a com um valor.
- Usando let- Faz a declaração de uma variável de escopo local, opcionalmente, inicializando-a com um valor.
- Usando const - Faz a declaração de uma variável de escopo global, que será usada apenas para leitura, ou seja, não podemos subscrever o seu identificador no decorrer do programa.

A palavra-chave var, foi usada em todo o código JavaScript de 1995 a 2015. As palavras-chave let e const foram adicionadas ao JavaScript em 2015. Quando devemos usar cada uma das palavras chaves?

Sempre use const para declarar variáveis que não podem ser alteradas após a atribuição inicial, e se o tipo de variável não deve ser alterado (Arrays e Objetos). Só use let se não puder usar const. Use apenas var se você deve oferecer suporte a navegadores antigos.

A regra Camelcase é uma convenção de nomenclatura utilizada em programação, incluindo JavaScript, para criar nomes de variáveis, funções e identificadores compostos por mais de uma palavra. A ideia é que o nome se assemelhe à forma de uma corcova de camelo, onde cada palavra subsequente começa com uma letra maiúscula, exceto a primeira palavra.

A regra Camelcase tem diferentes variações, mas as duas mais comuns são:



Fonte: Elaborado pelo autor (2023).

Camel Case: Nesse formato, todas as palavras após a primeira começam com uma letra maiúscula. A primeira palavra não tem letra maiúscula.

Exemplo: nomeDoUsuario, idadeDaPessoa, totalDeProdutos

Pascal Case: Similar ao Camelcase, mas a primeira letra da primeira palavra também é maiúscula.

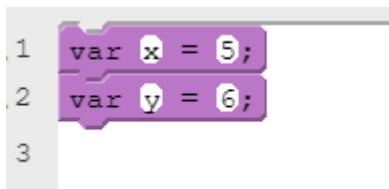
Exemplo: NomeDoUsuario, IdadeDaPessoa, TotalDeProdutos

A regra Camelcase é frequentemente usada em JavaScript para dar nomes a variáveis, funções e propriedades de objetos. Isso ajuda a tornar o código mais legível e organizado, especialmente quando os nomes são compostos por várias palavras.

Atribuir Valores

Em JavaScript, a atribuição de valor a variáveis é um processo simples e poderoso que permite que você coloque dados em um espaço de armazenamento identificado por um nome exclusivo. Para atribuir um valor a uma variável, você utiliza o operador de atribuição =. O valor à direita do operador é colocado na variável à esquerda.

Assim como na álgebra, as variáveis contêm valores:



Fonte: Elaborado pelo autor (2023).

Assim como na álgebra, variáveis são usadas em expressões:

Fonte: Elaborado pelo autor (2023).

Uma vez que um valor é atribuído a uma variável, você pode usá-lo de várias maneiras. Por exemplo, você pode exibir o valor no console, combiná-lo com outros valores ou utilizá-lo em cálculos. Veja um exemplo:

A atribuição de valor a variáveis não é apenas para valores fixos. Você pode atribuir valores resultantes de expressões, funções ou até mesmo valores obtidos de entradas do usuário.

Além disso, é importante mencionar que as variáveis podem ser reatribuídas com novos valores durante a execução do programa, permitindo que os dados sejam atualizados e reutilizados conforme necessário.

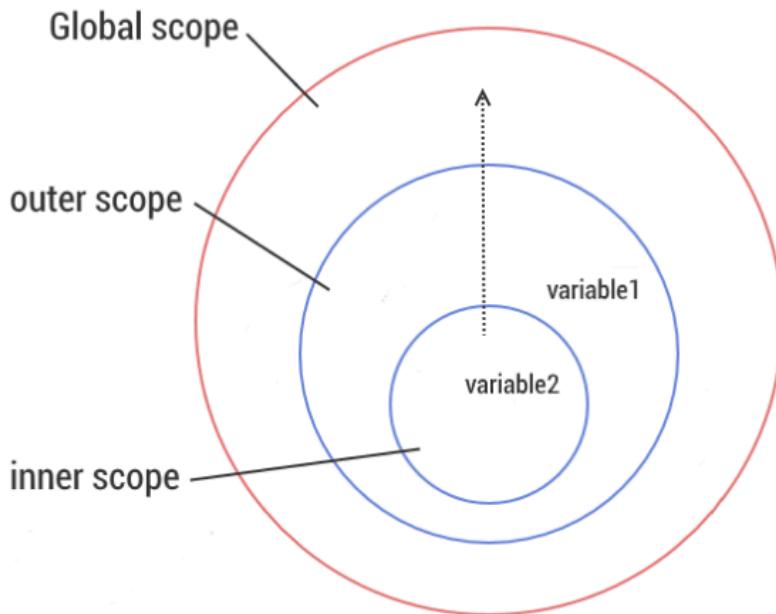
Identificador

Um identificador é uma palavra usada para “identificar” uma variável, possuem algumas regras ao declarar variáveis que devem ser consideradas. Identificar uma variável é muito importante para referenciar o propósito para o qual ela foi criada. Em javascript não seria diferente quanto aos identificadores, vamos conhecer algumas regras para uso de identificador de uma variável:

- O primeiro caractere é uma letra(a-z ou A-Z) ou os caracteres especiais _(sublinhado) ou \$ (cifrão).
- Os próximos caracteres podem ser os mesmos vistos na primeira regra acrescido de caracteres numéricos de 0 a 9.
- Não podem ser identificadores palavras reservadas da linguagem. Exemplo: a palavra var é uma palavra reservada para declaração de uma variável, outra array é uma palavra reservada para uma variável composta. Portanto, muito cuidado ao definir um identificador

de uma variável, o recomendado é sempre deixar o nome da sua variável intuitivo.

Escopo de Variáveis



Disponível em: <<https://javascript.reativa.dev/interview-14/>>. Acesso em 08 out. 2023.

Na programação temos um termo técnico chamado de escopo (local de atuação de uma determinada variável) que determina a acessibilidade (visibilidade) das variáveis. Temos 3 tipos de escopo: Escopo do bloco, escopo da função, Escopo global.

Escopo do Bloco

```
function testBlock(){
  if(true) {
    let z = 5
  }
  console.log(z)
}

testBlock() // Throws a ReferenceError "z" is not defined
```

Disponível em: <<https://javascript.reativa.dev/interview-14/>>. Acesso em 08 out. 2023.

O escopo do bloco foi introduzido com o ES6 (ECMAScript 2015) e se refere ao escopo de variáveis declaradas usando as palavras-chave `let` e `const` dentro de um bloco de código, como um

loop for ou uma estrutura condicional if. Variáveis declaradas nesse escopo são visíveis somente dentro desse bloco e são destruídas após o bloco ser executado.

Escopo da Função

```
function myFavoriteFunc(a) {
    if (true) {
        var b = "Hello " + a
    }
    return b
}

myFavoriteFunc("World")

console.log(a) // Throws a ReferenceError "a" is not defined
console.log(b) // does not continue here
```

Disponível em: <<https://javascript.reativa.dev/interview-14/>>. Acesso em 08 out. 2023.

O escopo da função é o escopo onde as variáveis declaradas com var, let ou const dentro de uma função são acessíveis. Essas variáveis só podem ser vistas dentro da função onde foram declaradas. Variáveis declaradas fora da função têm escopo global.

Escopo Global

O escopo global é o escopo onde as variáveis declaradas fora de qualquer bloco ou função são visíveis. Variáveis declaradas nesse escopo podem ser acessadas e modificadas em qualquer parte do código, dentro de funções, blocos ou no escopo global.

```
//global namespace
var g = "global"

function globalFunc(){
    function innerFunc(){
        console.log(g) // G é global, então é acessível aqui
    }
    innerFunc()
}
```

Disponível em: <<https://javascript.reativa.dev/interview-14/>>. Acesso em 08 out. 2023.

A compreensão dos diferentes tipos de escopo é fundamental para evitar conflitos de variáveis e problemas de acesso indesejado. A preferência pelo uso de let e const ao invés de var

auxilia na definição de escopos mais previsíveis e bem definidos.

Lembrando que, ao escolher onde declarar suas variáveis, você determina onde elas serão visíveis e, consequentemente, onde podem ser usadas e modificadas. Portanto, o conhecimento sobre os diferentes escopos é fundamental para escrever código organizado e evitar comportamentos inesperados.

Tipos de dados JavaScript

Na programação, os tipos de dados são um conceito importante. Para poder operar em variáveis, é importante saber algo sobre o tipo. Sem tipos de dados, um computador não pode resolver problemas com segurança. O js tem 8 tipos de dados principais

String

As strings são usadas para armazenar dados que são representados em forma de texto, ou seja, qualquer variável que será armazenada palavras ou frases podemos declarar como string. Uma das manipulações mais comuns com strings é verificar o tamanho e concatená-las(juntar palavras). Por definição as strings são sequências de caracteres alfanuméricos amplamente usadas na programação, no javascript uma string será determinada entre aspas:

```
1 var carName = "Volvo XC60";
2
3
4
```

Fonte: Elaborado pelo autor (2023).

Número

As variáveis do tipo number são variáveis com valores numéricos, no qual podemos separar em dois subtipos Inteiro(Integer) ou Real(Float). Ao contrário de muitas outras linguagens de programação, JavaScript não define diferentes tipos de números, como inteiros, curtos, longos, ponto flutuante, etc.

```
1 var x = 3.14;
2 var y = 3;
3
4
5
```

Fonte: Elaborado pelo autor (2023).

Os números JavaScript são sempre armazenados como números de ponto flutuante de precisão dupla, seguindo o padrão internacional IEEE 754.

Este formato armazena números em 64 bits, onde o número (a fração) é armazenado nos bits 0 a 51, o expoente nos bits 52 a 62 e o sinal no bit 63.

Números inteiros (números sem ponto ou notação de expoente) são precisos até 15 dígitos. Já a aritmética de ponto flutuante nem sempre é 100% precisa:

```
1 var x = 0.2+0.1;  
2  
3  
4
```

Fonte: Elaborado pelo autor (2023).

O comando `parseInt()` e `parseFloat()` são funções em JavaScript usadas para converter strings em números inteiros e de ponto flutuante, respectivamente. Essas funções são úteis quando você precisa trabalhar com valores numéricos armazenados como strings ou quando deseja validar a entrada do usuário para garantir que ela seja numérica.

A função `parseInt()` converte uma string em um número inteiro. Ela analisa a string até encontrar um caractere não numérico ou o final da string e retorna o número resultante.

```
1 var numeroString = "42";  
2 var numeroInteiro = parseInt(numeroString);  
3 console.log(numeroInteiro);  
4  
5
```

Fonte: Elaborado pelo autor (2023).

// Saída: 42

A função `parseFloat()` é semelhante ao `parseInt()`, mas é usada para converter strings em números de ponto flutuante (números com casas decimais).

```
1 var numeroHexadecimal = "1A";  
2 var numeroDecimal = parseFloat(numeroHexadecimal, 16);  
3 console.log(numeroDecimal);
```

Fonte: Elaborado pelo autor (2023).

// Saída: 26

Ambas as funções tentam extrair um número da string fornecida, ignorando espaços em branco no início e no final da string. Se a string não puder ser convertida em um número válido, as funções retornarão NaN (Not-a-Number), indicando que a conversão falhou.

O comando `console.log()` é uma função em JavaScript que é usada para exibir informações no console do navegador ou em ambientes de desenvolvimento, como o console do Node.js. É uma ferramenta útil para depuração e desenvolvimento, pois permite que os desenvolvedores visualizem valores de variáveis, mensagens de status e outras informações relevantes durante a execução de um programa.

Bigint

As variáveis JavaScript `BigInt` são usadas para armazenar valores inteiros grandes que são muito grandes para serem representados por um JavaScript normal `Number`. Para criar a `BigInt`, acrescente `n` ao final de um número inteiro ou chame `BigInt()`:

Exemplos

```
1 var x = 9999999999999999;
2
3 var x = BigInt(9999999999999999) [ ] [ ];
4
```

Fonte: Elaborado pelo autor (2023).

Booleano

As variáveis do tipo `boolean` são variáveis lógicas que possuem apenas dois valores, `true`(Verdadeiro)ou `false`(Falso), onde esses valores devem ser definidos sem o uso das aspas.

```
1 var booleano = true;
2
3
4 var booleano = false;
```

Fonte: Elaborado pelo autor (2023).

Indefinido e Nulo

As variáveis dos tipos `null` e `undefined` representam variáveis que não possuem nenhum valor ou que estão incompletas. Para criar uma variável nula, devemos especificar seu valor como "`null`", como vemos no exemplo abaixo:

```
1 var valorNulo = null;
2
```

Fonte: Elaborado pelo autor (2023).

Podemos também criar uma variável e não definir o valor inicialmente, declarando-a como indefinida usando o "undefined", vemos no exemplo abaixo:

```
1 var semValor = "";
```

Fonte: Elaborado pelo autor (2023).

Símbolo

Em JavaScript, o tipo de dado Symbol é uma adição relativamente recente à linguagem (introduzido no ECMAScript 2015, também conhecido como ES6). Símbolos são valores únicos e imutáveis que podem ser usados como identificadores para propriedades de objetos. Eles são frequentemente usados para criar propriedades "privadas" em objetos ou para definir comportamentos especiais em APIs.

API (Interface de Programação de Aplicativos) é um conjunto de regras e protocolos que permite que diferentes softwares se comuniquem e interajam entre si. As APIs fornecem uma maneira padronizada para que os desenvolvedores acessem e usem os recursos ou funcionalidades de um software, serviço ou plataforma sem precisar entender detalhes internos de como esses recursos são implementados.

A principal vantagem das APIs é que elas permitem que os desenvolvedores aproveitem funcionalidades de outras aplicações ou serviços sem precisar reinventar a roda. Isso promove a reutilização de código, acelera o desenvolvimento de novos aplicativos e facilita a integração entre diferentes sistemas.

Objeto

Você já aprendeu que variáveis JavaScript são contêineres para valores de dados. Objetos em JavaScript são estruturas de dados complexas que permitem armazenar e organizar informações relacionadas em um único local. Eles são usados para representar entidades, conceitos ou até mesmo abstrações do mundo real. Um objeto é composto por pares chave-valor, onde cada

chave (também chamada de propriedade) é uma string que identifica um valor associado.

Exemplo:

```
1 var pessoa = { nome: "Ana", idade: 25, profissao: "Engenheira" };  
2
```

Fonte: Elaborado pelo autor (2023).

Criar objetos é como colocar algo na memória ou guardar algo em um guarda-roupa. Então, para podermos ver isso, precisamos colocar esse objeto em exibição. Alguns objetos possuem propriedades e funções que podem ser alteradas, e para acessar qualquer propriedade ou função de um objeto, usamos a notação de ponto.

A notação de ponto é uma forma de acessar propriedades e métodos de um objeto em JavaScript. É uma das principais maneiras de interagir com objetos e manipular seus valores e comportamentos. A notação de ponto é simples e amplamente utilizada na programação JavaScript.

Para acessar uma propriedade ou método de um objeto usando a notação de ponto, você utiliza a seguinte sintaxe:

```
1 objeto.propriedade  
2 objeto.metodo( )
```

Fonte: Elaborado pelo autor (2023).

Acesso a Propriedades

Você pode usar a notação de ponto para acessar as propriedades de um objeto. As propriedades são valores associados a um objeto. Por exemplo:

```

1 var pessoa = {
2   nome: "Ana",
3   idade: 25,
4   profissao: "Engenheira"
5 }
6
7 console.log(pessoa.nome); // Acesso à propriedade "nome"
8 console.log(pessoa.idade); // Acesso à propriedade "idade"
9 console.log(pessoa.profissao); // Acesso à propriedade "profissao"
10
11
12
13

```

Fonte: Elaborado pelo autor (2023).

Chamada de Métodos:

Métodos são funções associadas a um objeto. Eles podem ser chamados utilizando a notação de ponto. Por exemplo:

```

1 var calculadora = {
2   somar: function(a, b) {
3     return a + b;
4   },
5   subtrair: function(a, b) {
6     return a - b;
7   }
8 }
9
10
11 console.log(calculadora.somar(5, 3)); // Chamada do método "somar"
12 console.log(calculadora.subtrair(10, 2)); // Chamada do método "subtrair"
13

```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools with the 'Console de depuração' (Debug Console) tab selected. The output window displays two lines of text: '8' and '8'. This indicates that the code was run and the results of the console.log statements were printed.

Fonte: Elaborado pelo autor (2023).



RESUMO

Neste tema nos aprofundamos nas regras de declaração e atribuição de valor de uma variável em js, descobrimos novas regras que fazem parte da sintaxe dessa linguagem de programação, como por exemplo a Camelcase e a notação de ponto. Reforçamos a importância de seguir boas práticas de programação, como escolher nomes descritivos para variáveis, documentar código e usar comentários para explicar a lógica por trás das operações.

Além de fornecer uma base sólida para entender os elementos fundamentais da programação em JavaScript e como eles se encaixam no desenvolvimento de aplicativos e sistemas complexos. Com esses conhecimentos, estamos prontos para enfrentar desafios mais avançados e criar soluções inovadoras.



ATIVIDADE DE FIXAÇÃO

1. O que são variáveis e para que elas são utilizadas?
2. Elenque quais os tipos de variável mais utilizadas
3. Declare uma variável chamada nome e atribua a ela o valor do seu nome. Escreva abaixo como seria o código escrito.
4. Explique a regra CamelCase e quando ela deve ser utilizada.
5. Explique o que é a notação de ponto e dê 2 exemplos dessa regra.
6. Explique a diferença entre os tipos de dados strings e números em js
7. Qual a maneira de fazer um comentário no código?
8. Crie um exemplo que utiliza uma variável com valor numérico e um exemplo com valor string
9. Em qual momento do meu código , preciso usar o parseInt(), ou o parseFloat()?
10. Explique o tipo de dado booleano e em que situação ele será utilizado.
11. Crie um aplicativo cujo objetivo é inserir seus dados pessoais e armazenar em cada variável determinada e depois exibidos na tela

TEMA 03

Operadores

Habilidades

- Compreender a lógica da matemática na linguagem de programação
- Conhecer os símbolos importantes na sintaxe e suas funções

Podemos classificar os operadores de várias formas, a mais conhecida e de acordo com o número de operandos que eles possuem. Em javascript temos diversas classificações de operadores, vamos conhecer as mais conhecidas e como utilizar.

Operadores de Atribuição

O operador de atribuição faz o que o próprio nome sugere, atribui um valor ao operando à sua esquerda baseando-se no operando à direita, o sinal de “ = ” (igual) por exemplo, atribui o valor do operando à direita ao operando à esquerda ($X = Y$) , atribui o valor de X a Y. Temos alguns outros tipos de operadores de atribuição em javascript, vamos conhecer os principais, veja na tabela abaixo:

Nome	Operador encurtado	Significado
Atribuição	$x = y$	$x = y$
Atribuição de adição	$x += y$	$x = x + y$
Atribuição de subtração	$x -= y$	$x = x - y$
Atribuição de multiplicação	$x *= y$	$x = x * y$
Atribuição de divisão	$x /= y$	$x = x / y$
Atribuição de resto	$x \% y$	$x = x \% y$
Atribuição exponencial	$x **= y$	$x = x ** y$

Operadores de Comparação

Esse tipo de operador faz a comparação de seus operandos e retorna um valor lógico caso a comparação for verdadeira. Vamos um exemplo:

```
1 var numero1 = 4; var numero2 = 8;
2 console.log(numero1 == numero2);
3
```

Fonte: Elaborado pelo autor (2023).

saída:

The screenshot shows a browser's developer tools console window titled "Console de depuração". It contains a single line of code: "false". The text is displayed in blue, which typically indicates a boolean value in many programming environments.

Fonte: Elaborado pelo autor (2023).

No exemplo acima temos duas variáveis com valores distintos, o sinal de comparação “==” faz a comparação para retornar se essa comparação é verdadeira, caso for essa expressão retorna como true, caso ao contrário retorna false. Vamos conhecer mais alguns operadores de comparação:

Operador Descrição

<code>==</code>	Retorna verdadeiro caso os operandos sejam iguais.
<code>Igual</code>	
<code>!=</code>	Retorna verdadeiro caso os operandos sejam diferentes.
<code>Diferente</code>	
<code>==</code>	Retorna verdadeiro caso os operandos sejam do mesmo valor e do mesmo tipo.
<code>Igual</code>	
<code>></code>	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.
<code>Maior que</code>	
<code><</code>	Retorna verdadeiro caso o operando da esquerda seja menor que o da direita.
<code>Menor que</code>	
<code>>=</code>	Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita.
<code>Maior que ou igual</code>	
<code><=</code>	Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita.
<code>Menor que ou igual</code>	

Operadores Aritméticos

Os operadores aritméticos usam valores numéricos, quer sejam literais ou variáveis, como seus operando e retornam um único valor numérico. Na programação os operadores aritméticos, em sua maioria se assemelham muito de linguagem para linguagem, como adição (+), subtração (-), divisão (/) e multiplicação (*), mas temos algumas diferenças que é importante conhecermos. Vejamos alguns exemplos de operadores aritméticos em javascript:

Adição	+
--------	---

Subtração	-
Multiplicação	*
Divisão	/
Exponencial	**
Resto da Divisão	%
Incremento	++
Decremento	--

Operadores Lógicos

Esse tipo de operador é utilizado tipicamente com valores booleanos(lógicos), retornando assim um valor booleano. Entretanto, os operadores **||**(OU) e **&&** (E) retornam o valor de um dos operandos em específico, de modo que esses operadores façam uso de valores não booleanos, eles possam também retornar um valor não booleano, então podemos entender que não é uma regra absoluta, vamos conhecer os operadores lógicos:

AND (& &) O operador “ E ” recebe dois operandos e retorna verdadeiro somente se ambos os valores forem verdadeiros, caso contrário retorna falso.

OR (||) O operador “ OU ” recebe dois operandos e retorna verdadeiro caso os dois ou um dos operandos for verdadeiro.

NOT (!) - O operador de negação “ NÃO ” é um operador unário, ou seja, ele recebe apenas um operando e nega, inverte o valor lógico do operando.

Operador de String

Os operadores de string são usados para manipular e transformar dados de texto, permitindo que os programadores realizem tarefas específicas com strings de maneira eficiente e flexível.

Concatenação (+)

A concatenação em JavaScript refere-se ao processo de combinar (juntar) duas ou mais strings para criar uma nova string. Isso é feito usando o operador de concatenação **+**, que é usado para unir diferentes valores de string em uma única string.

```
1 var nome = "João";
2 var sobrenome = "Silva";
3
4 var nomeCompleto = nome + " " + sobrenome;
5 console.log(nomeCompleto);
```

Fonte: Elaborado pelo autor (2023).

Aqui está um exemplo simples de concatenação:

Saída:

The screenshot shows a browser's developer tools console window titled "Console de depuração". It displays the output of the code execution, which is the string "João Silva".

Fonte: Elaborado pelo autor (2023).

No exemplo acima, a variável nomeCompleto é criada pela concatenação das variáveis nome e sobrenome, com um espaço entre elas.

Outra maneira que a concatenação também pode ser usada com valores de variáveis e strings literais:

```
var idade = 30;
console.log("A idade de João é " + idade + " anos.");
// Saída: "A idade de João é 30 anos."
```

```
1 var idade = 30;
2 console.log("A idade de João é " + idade + " anos.");
3
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

Você pode concatenar quantos valores quiser, e o JavaScript os unirá em ordem. Lembre-se de que a concatenação é uma operação que cria uma nova string e não altera as strings originais.

Operador Ternário (Condicional)

O operador ternário é o único operador javascript que usa três operandos, dado o nome ternário. O operador pode ter um de dois valores baseados em uma condição, vamos conhecer sua sintaxe:

condição ? valor1 : valor2

Se a condição for verdadeira, o operador terá o valor de valor1, senão terá o valor de valor2. Este operador pode ser usado em qualquer lugar onde você usaria um operador padrão, vamos para mais um exemplo:

```
var status = (idade >= 18) ? "Adulto" : "Menor de Idade";
```

A expressão acima atribui o valor “Adulto” à variável idade caso seja maior ou igual a 18, senão ela atribui o valor “Menor de Idade”.

Operador de tipos

O operador de tipo em javascript (typeof) vai analisar qual o tipo primitivo de uma determinada variável e retorna seu tipo. Como exemplo:

```
typeof "John"           // Returns "string"  
typeof 3.14             // Returns "number"  
typeof NaN              // Returns "number"  
typeof false            // Returns "boolean"  
typeof [1,2,3,4]         // Returns "object"
```

Operadores Relacionais

Um operador relacional faz a comparação de seus operandos e retorna como valor booleano se baseando caso a comparação seja verdadeira. O operador in é muito usado para verificar se um

elemento faz parte de um array(variável composta). Vamos ver alguns exemplos:

```

1 var x = 10;
2 var y = 5;
3
4 console.log(x == y); // false
5 console.log(x === y); // false
6 console.log(x != y); // true
7 console.log(x !== y); // true
8 console.log(x > y); // true
9 console.log(x >= y); // true
10 console.log(x < y); // false
11 console.log(x <= y); // false
12

```

Fonte: Elaborado pelo autor (2023).

Saída:

```

false
false
true
true
true
true
false
false

```

Fonte: Elaborado pelo autor (2023).

Precedência

A precedência de um operador é a ordem no qual as operações serão realizadas em uma expressão, por exemplo se tenho a seguinte expressão; $6 + 6 / 2 = ?$ Existe uma ordem de precedência no qual serão realizados esses cálculos, e se caso você pensou que o resultado da expressão é 6 , infelizmente você errou! O resultado correto da expressão é 9 , pois é calculado primeiro a divisão , com o resultado da divisão é somado o 6. Vamos conhecer a ordem de precedência de operadores em javascript:

1 - Operadores Aritméticos

() Parênteses

** Exponenciação

*, / e % Multiplicação, Divisão e Resto da divisão

- e + Subtração e Adição

2 - Operadores Relacionais

Não possuem ordem de precedência, são feitos o que estiver primeiro contando da esquerda para a direita.

3- Operadores Lógicos

! - NÃO

&& - E

|| - OU



Neste tema nós aprendemos sobre as regras dos operadores. Os operadores desempenham um papel crucial na programação, permitindo que os programadores realizem diversas operações e tomem decisões com base nas relações entre os valores. Eles são símbolos ou palavras-chave que indicam ações a serem realizadas em variáveis e valores. Os operadores são classificados em várias categorias, incluindo operadores aritméticos, operadores de atribuição, operadores de comparação, operadores lógicos e muito mais.



1. Utilize os operadores adequados para comparar os valores abaixo:

- a. 1 ____ 4
- b. 5 ____ 7 ____ 3
- c. “morango” ____ “abacaxi”
- d. 1 ____ false
- e. 12 / 4 ____ x / y
- f. 6 * 4 ____ 24

2. Cite 2 tipos de operadores de atribuição. Apresenta um trecho de código que os use
3. Além de cálculo, para que mais podemos usar o operador + ? Exemplifique
4. Dado o cálculo $12 * 3 + 4 - 8 / 2 \% 3$, qual o resultado segundo a precedência de

operadores?

5. Cite 2 tipos de operadores lógicos. Apresente um trecho de código que os use
6. Cite 2 tipos de operadores aritméticos. Apresente um trecho de código que os use
7. Crie um mapa conceitual com os tipos de operadores e seus símbolos representativos.
8. Crie um programa que peça ao usuário para inserir dois números. Verifique se eles são iguais e exiba uma mensagem indicando o resultado.
9. Crie um aplicativo de uma calculadora cujo objetivo é de calcular o desconto sobre o preço dos produtos no ambiente code.org.
10. Crie um aplicativo que calcule o Índice de Massa Corporal (IMC) de uma pessoa. Peça o peso e a altura, e calcule o IMC usando a fórmula: $IMC = \text{peso} / (\text{altura} * \text{altura})$.

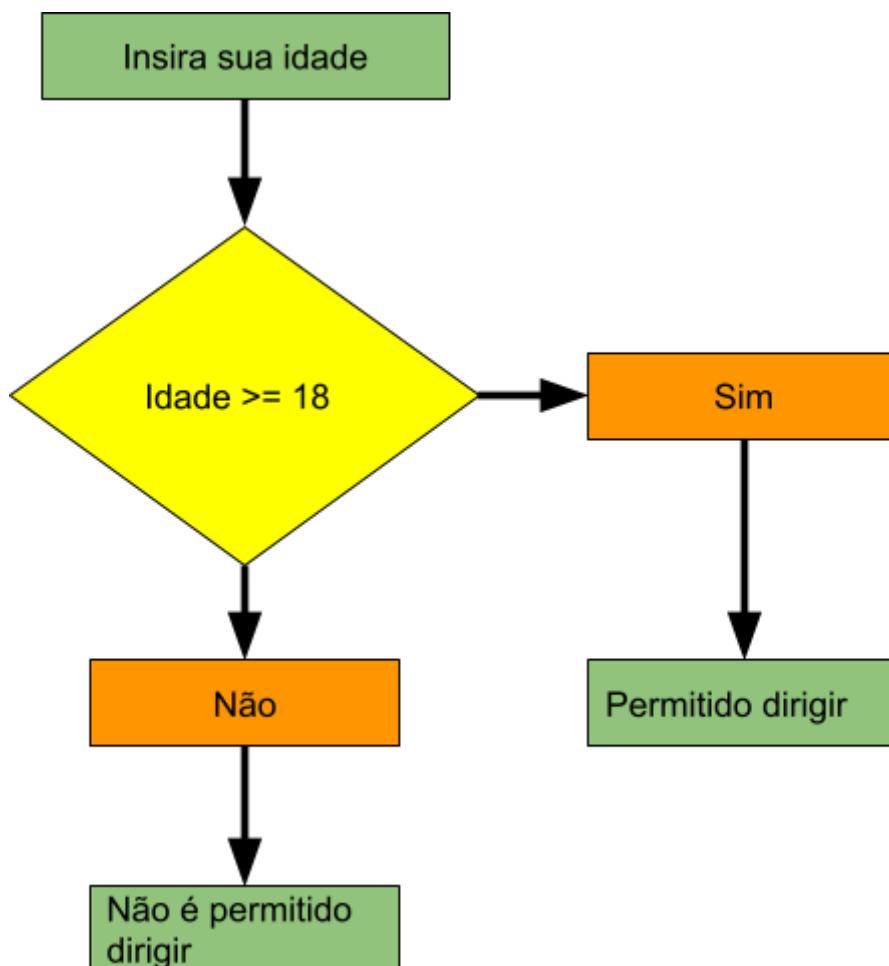
TEMA 04

Estruturas Condicionais

Habilidades

- Compreender os mecanismos de condicionais
- Aplicar na prática as estruturas de condicionais no ambiente de desenvolvimento.
- Conhecer a lógica das condicionais.

Estruturas condicionais são condições impostas para que determinada ação aconteça. Por exemplo, quando sua mãe diz: Você só irá jogar videogame quando arrumar seu quarto, ela está impondo uma condição para você jogar videogame , caso contrário você não irá jogar videogame. Vamos ver outro exemplo, agora em gráfico para ficar mais claro:



Fonte: Elaborado pelo autor (2023).

Nesse exemplo mostra uma condição verificando a idade do usuário, se é ou não maior de 18 anos. Nós seres humanos tomamos decisões o tempo todo que afetam nossas vidas, desde criança, como por exemplo vou assistir desenho ou jogar videogame?, e na fase adulta, como por exemplo, fazer um curso em uma faculdade ou abrir meu próprio negócio?. Na programação não é diferente, usamos as estruturas condicionais para definir o comportamento do nosso programa de acordo com a decisão que especificamos no código, assim podemos definir ações para o código de analisar através de condições , e para cada condição uma determinada ação é feita.

Em javascript temos alguns tipos de estruturas condicionais semelhantes às encontradas em outras linguagens, usamos a estrutura IF (SE) para iniciar o bloco condicional (também pode ser chamado de desvio condicional) e determinamos uma condição, caso esta condição seja verdadeira um bloco de comando é executado. Vamos conhecer as estruturas condicionais usadas no javascript.

As condicionais são amplamente utilizadas em vários casos. Para exemplificar os tipos de condicionais vamos pegar como exemplo um aplicativo em que o aluno verifica o status que está depois de obter sua média de notas.

Para acessar link : <https://tinyurl.com/bdfv4uft>



Fonte: Elaborado pelo autor (2023).

Estrutura Condicional Simples

Esse tipo de estrutura faz a verificação de apenas uma condição, e caso a condição seja verdadeira ela executa um bloco de comando, caso contrário o programa sai da estrutura condicional:

```
var media = 8
if ( media >= 7 ){
    console.log ("Aluno Aprovado");
}
```

```
1 var media = 0;
2 onEvent("text_input1", "input", function() {
3     media = getText("text_input1");
4     if (media >= 7) {
5         setText("label3", "Você está aprovado ");
6     }
7 });
8
```

Fonte: Elaborado pelo autor (2023).

Estrutura Condicional Composta

A estrutura condicional composta é a verificação de uma condição, e caso seja verdadeira ela executa um bloco de comando, caso for falsa é executado outro bloco de comando:

```
var media = 8
if ( media >= 7 ){
    console.log ("Aluno Aprovado");
} else{
    console.log ("Aluno Reprovado");
}
```

```
1 var media = 0;
2 onEvent("text_input1", "input", function() {
3     media = getText("text_input1");
4     if (media >= 7) {
5         setText("label3", "Você está aprovado ");
6     } else {
7         setText("label3", "Você está reprovado ");
8     }
9 });
10
```

Fonte: Elaborado pelo autor (2023).

Estrutura Condicional Aninhada

Essa estrutura é a que normalmente usamos em nosso cotidiano, pois raramente temos apenas uma condição para uma tomada de decisão, geralmente são mais de duas condições, por isso é muito importante o entendimento dessa estrutura para a prática no javascript:

```
var media = 8
if ( media >= 7 ){
    console.log ("Aluno Aprovado");
} else if ( media > 4 || media <= 6 ){
    console.log ("Aluno em Recuperação");
}else{
    console.log ("Aluno Reprovado");
}
```

```
1 var media = 0;
2 onEvent (text_input1, input, function() {
3     media = getText (text_input1);
4     if (media >= 7) {
5         setText (label3, "Você está aprovado ");
6     } else if ((media > 4) || (media <= 6)) {
7         setText (label3, "Você está de recuperação ");
8     } else {
9         setText (label3, "Você está reprovado ");
10    }
11 });
12
```

Fonte: Elaborado pelo autor (2023).

Podemos analisar várias condições em uma estrutura aninhada, não existe um número exato de condições que podem ser usadas.

```
if (condição) {
// Ação
} else if (outra condição) {
// Ação
} else if (outra condição) {
// Ação
```

```
 } else if (outra condição) {  
 //Ação  
 } else if (quantas condições quiser) {  
 //Ação  
 } else {  
 //Ação final se nenhuma condição for verdadeira  
 }
```

Estrutura de Múltiplas Escolhas (Switch Case)

A estrutura de múltiplas escolhas, também conhecida como switch case, é uma construção de controle em linguagens de programação que permite executar diferentes blocos de código com base no valor de uma expressão específica. Ela é uma alternativa ao uso repetido de estruturas if...else quando você precisa avaliar uma variável ou expressão em relação a múltiplas opções.

A estrutura switch case funciona da seguinte maneira:

É fornecida uma expressão (geralmente uma variável) que será avaliada em relação a diferentes casos. Cada caso contém um valor que a expressão será comparada. Se o valor da expressão for igual a um dos valores do caso, o bloco de código associado a esse caso será executado. Se nenhum caso coincidir, você pode fornecer um caso default que será executado se nenhum dos outros casos corresponder.

Aqui está um exemplo de como a estrutura switch case é usada em JavaScript:

```
1 var diaDaSemana = 3;  
2 var mensagem;  
3 switch (diaDaSemana) {  
4     case 1:  
5         mensagem = "Hoje é domingo.";  
6         break;  
7     case 2:  
8         mensagem = "Hoje é segunda-feira.";  
9         break;  
10    case 3:  
11        mensagem = "Hoje é terça-feira.";  
12        break;
```

Fonte: Elaborado pelo autor (2023).

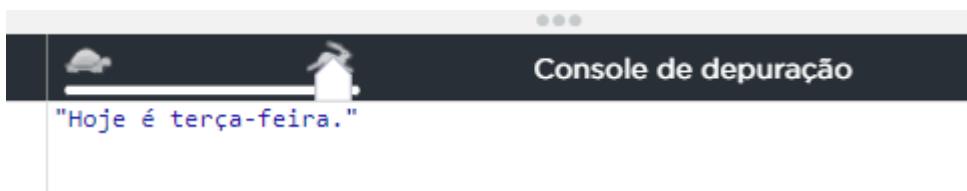
```
13 case 4:  
14     mensagem = "Hoje é quarta-feira."  
15 break;  
16 case 5:  
17     mensagem = "Hoje é quinta-feira."  
18 break;  
19 case 6:  
20     mensagem = "Hoje é sexta-feira."  
21 break;  
22 case 7:  
23     mensagem = "Hoje é sábado."  
24 break;
```

Fonte: Elaborado pelo autor (2023).

```
25 default:  
26     mensagem = "Dia inválido."  
27 break;  
28 }  
29 console.log(mensagem);
```

Fonte: Elaborado pelo autor (2023).

Saída



Fonte: Elaborado pelo autor (2023).

Neste exemplo, a expressão diaDaSemana é avaliada e comparada com os diferentes casos. Como o valor de diaDaSemana é 3, o bloco de código associado ao caso 3 é executado.

A estrutura switch case é útil quando você precisa comparar uma expressão com vários valores diferentes e executar diferentes ações com base na correspondência. No entanto, é importante lembrar de usar o break para evitar a execução de casos subsequentes após encontrar um caso correspondente.



RESUMO

Neste contexto de aprendizado, exploramos a importância dos operadores no desenvolvimento de programas. Os operadores desempenham um papel fundamental na construção da lógica, na tomada de decisões e na realização de cálculos. Eles oferecem aos desenvolvedores as ferramentas necessárias para controlar o fluxo de execução de um programa e criar comportamentos dinâmicos que respondem às diversas situações que podem surgir.

Um domínio sólido dos diferentes tipos de operadores é um pré-requisito essencial para a criação de programas eficientes e funcionalmente robustos. Os operadores, neste contexto, são representados por símbolos ou palavras-chave que indicam as ações a serem executadas em variáveis e valores. Eles são categorizados em várias classes, abrangendo operadores aritméticos, operadores de atribuição, operadores de comparação, operadores lógicos e uma série de outras categorias.

Portanto, a compreensão dos operadores é fundamental para os desenvolvedores, pois eles fornecem as ferramentas necessárias para criar lógica coerente, tomar decisões criteriosas e executar cálculos precisos em seus programas, permitindo que eles atinjam níveis mais elevados de eficiência e funcionalidade.



ATIVIDADE DE FIXAÇÃO

1. Explique o que são condicionalismos.
2. Cite um exemplo de condicional que usamos no dia-a-dia.
3. Transforme o exemplo de condicional dado no exercício acima em um esquema de fluxograma.
4. O que são condicionais aninhadas?
5. Qual a palavra chave para criar uma condicional simples no js ? E para criar uma condicional composta?
6. Crie um script que leia três números inteiros, em seguida mostre o maior e o menor deles.
7. Você começou a estudar economia financeira e quer ter mais responsabilidade sobre seus gastos, então decidiu criar um app onde você insere o valor do seu saldo disponível, e o valor do produto que você deseja comprar e então ele te diz se você pode ou não comprar esse produto naquele momento, caso contrário retorne uma mensagem de saldo insuficiente. Crie um algoritmo para explicar o funcionamento desse programa.
8. Sua escola soube o quanto você é bom em criar programas, e lhe pediu para criar um programa para divulgar o resultado da competição de futsal que terá na escola. O programa deve receber os valores dos saldos de pontos das 6 partidas. Se o time ganha a partida recebe 3 pontos, independente do número de gols. Se o time perder a partida recebe 1 ponto pela participação.
 - a. crie um algoritmo para demonstrar o funcionamento do programa.
 - b. qual a estrutura de condicional que mais se encaixa com esse programa?
 - c. quantas partidas o time deve ganhar, pelo menos, para ser o campeão?

9. Crie um programa que solicite ao usuário um número inteiro. Verifique se o número é par ou ímpar e exiba o resultado.

10. Desenvolva um aplicativo que leia duas notas digitadas pelo usuário e mostre a média e a situação de um aluno(de 0 a 4,5 – Aluno Reprovado, de 5 a 6,5 – Aluno em Recuperação e de 7 a 10 – Aluno Aprovado).

TEMA 5

Laços de Repetição

Habilidades

- Conhecer as estruturas de repetição
- Analisar criticamente a aplicação do loop pertinente a cada situação.
- Identificar e reconhecer as palavras chaves

Os laços de repetição, são muito utilizados em praticamente todo programa, podemos utilizar os laços para repetir praticamente tudo, começando por repetições simples, como uma simples contagem de 0 a 10, indo para repetições mais avançadas, e até exibir propriedades de um objeto ou índices de um array.

Os laços de repetição permitem automatizar tarefas repetitivas, como processar listas de itens, validações e manipulações de dados. No entanto, é fundamental garantir que a condição do loop se torne falsa em algum ponto para evitar loops infinitos. Além disso, escolher o tipo de loop correto depende da situação e do conhecimento prévio do número de iterações.

Ao usar laços de repetição, é importante otimizar o código para evitar processamentos desnecessários e garantir a eficiência do programa.

Em javascript temos alguns laços já conhecidos em outras linguagens, e alguns laços característicos da própria linguagem, como exemplo da tabela abaixo:

- ☒ **For**
- ☒ **While**
- ☒ **Do While**

For (Para)

Esse tipo de laço é usado geralmente quando vamos utilizar uma finita e conhecida de repetições, pois existem casos em que não sabemos o número de repetições, depende de uma resposta de um usuário ou servidor. Entretanto podemos definir como um laço infinito mantendo a condição ou uma alteração de uma variável em branco.

Esse laço é conhecido também como laço como contador, devido usarmos uma variável que irá “controlar” o número de repetições. Ele possui três partes: inicialização, condição e expressão final em sua sintaxe:

```
for (valor inicial; condição; incremento++ ou decremento--) { Ações;  
}
```

```
1 for (var i = 0; i < 4; i++) {  
2        
3 }
```

Fonte: Elaborado pelo autor (2023).

Inicialização: Essa parte é executada apenas uma vez, no início do loop. Geralmente é usada para inicializar variáveis que controlam a interação do loop.

Condição: A condição é verificada antes de cada iteração do loop. Se a condição for verdadeira, o bloco de código dentro do loop é executado; caso contrário, o loop é encerrado.

Expressão Final: Essa parte é executada após cada iteração do loop. Normalmente é usada para atualizar ou modificar as variáveis do loop.

Vamos fazer um contador de 0 a 10 para entender melhor como funciona esse laço:

```
for (var i = 0; i < 11; i++) {  
    console.log(i);  
}
```

```
1 for (var i = 0; i < 11; i++) {  
2     console.log(i);  
3 }  
4
```

Fonte: Elaborado pelo autor (2023).

Saída:

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Fonte: Elaborado pelo autor (2023).

A inicialização é var i = 0, onde a variável i é iniciada com o valor 0.

A condição é `i < 11`, o que significa que o loop continuará enquanto `i` for menor que 11.

A expressão final é `i++`, que incrementa o valor de `i` após cada iteração.

O loop `for` irá executar o bloco de código dentro dele repetidamente, incrementando `i` de 0 até 10 (pois quando `i` for 11, a condição não será mais verdadeira).

Existem 2 variações de loop `for`, que são usados para fins diferentes. O `for...in` e o `for...of` são duas formas de laços de repetição em JavaScript que são utilizados para percorrer elementos em uma estrutura de dados. Eles têm diferentes propósitos e são adequados para diferentes tipos de objetos. Vamos entender a diferença entre eles:

for...in

O laço `for...in` é usado para iterar sobre as propriedades enumeráveis de um objeto, como chaves de objetos ou índices de arrays. Ele é mais adequado para percorrer propriedades de objetos.

for...of

O laço `for...of` é usado para iterar sobre elementos iteráveis, como arrays, strings, objetos Set, objetos Map, entre outros. Ele é mais adequado para percorrer os valores de coleções.

Diferenças

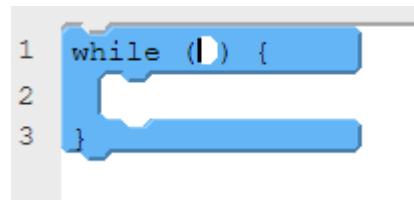
- O `for...in` itera sobre as chaves ou índices do objeto, enquanto o `for...of` itera sobre os valores da coleção.
- O `for...in` é adequado para objetos, enquanto o `for...of` é mais adequado para coleções iteráveis.
- O `for...in` também itera sobre as propriedades herdadas do objeto, enquanto o `for...of` não.

WHILE (Enquanto)

O laço `while` (Enquanto) é um dos mais simples na programação, ele realiza o teste de uma condição(teste lógico) no início de cada repetição, e continua fazendo enquanto a condição for verdadeira(true). É importante lembrar que no `while` não é feito o incremento ou decremento na declaração inicial, dentro do bloco de comando a variável deve receber um incremento ou decremento, caso contrário seu programa irá num loop infinito.

Sintaxe:

```
while ( condição ) { bloco de comando;
}
```



Fonte: Elaborado pelo autor (2023).

Vamos fazer como um exemplo a mesma contagem do laço anterior de 0 a 10, e perceber algumas diferenças na sintaxe, mas sabendo que o resultado será o mesmo.

```
1 var i = 0;
2 while ( (i < 11) ) {
3     console.log(i);
4     i++;
5 }
6 console.log("fim");
7
```

Fonte: Elaborado pelo autor (2023).

Nesse exemplo a variável i é declarada no início de cada repetição, e enquanto i menor a 11 (teste lógico) o bloco de comandos é feito (nesse caso usamos para mostrar a contagem), esse laço só para de repetir quando chegar no valor definido no bloco (c++), caso não digite essa linha o laço repete o número zero infinitamente, pois não foi adicionado valor para variável c.

The screenshot shows a browser's developer tools with the 'Console de depuração' (Debug Console) tab selected. The output window displays the following text:
0
1
2
3
4
5
6
7
8
9
10
"fim"

Saída:

Fonte: Elaborado pelo autor (2023).

Do while (Faça Enquanto)

Esse laço é praticamente o mesmo que o while, a única diferença, é que a condição (teste lógico) é feita depois do bloco de comando for digitado. Vamos usar o mesmo exemplo do laço anterior para entendermos as diferenças:

Código:

```
1 var i = 0;
2 do{
3     console.log(i);
4     i++;
5 }
6 while (i<11);
7 console.log("fim");
8
```

Fonte: Elaborado pelo autor (2023).

A instrução “break” é usada para pausar “saltar” de um laço, semelhante ao exemplo visto no comando Switch Case, o break faz a interrupção do bloco de comando em que está inserido.

Código:

```
1 var i = 0;
2 while (i<11){
3     if (i==11){
4         break;
5     }
6     i++;
7     console.log(i);
8 }
9 console.log("fim");
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools with the "Console de depuração" (Debug Console) tab selected. The console window displays the following output:
1
2
3
4
5
6
7
8
9
10
11
"fim"

Fonte: Elaborado pelo autor (2023).

Temos outros tipos de laços, conhecidos como laços de iterações que vamos conhecer no tema 7 na seção Eventos e Iterações.

Usar loops em JavaScript oferece várias vantagens significativas, especialmente quando se trata de automatizar tarefas repetitivas e realizar iterações sobre coleções de dados. Aqui estão algumas das principais vantagens de utilizar loops em JavaScript:

Automação de Tarefas Repetitivas

Os loops permitem executar um bloco de código repetidamente, poupando o esforço de escrever a mesma sequência de instruções várias vezes. Isso é particularmente útil para realizar tarefas que requerem a mesma operação em múltiplos itens ou situações.

Eficiência de Código

Em vez de replicar o mesmo código manualmente, um loop pode executar o código várias vezes com base em uma condição específica. Isso torna o código mais enxuto, fácil de manter e menos propenso a erros.

Iteração em Coleções de Dados

Loops são frequentemente usados para percorrer arrays e objetos, permitindo a manipulação de cada elemento individualmente. Isso é essencial para realizar operações em múltiplos itens, como cálculos, transformações ou filtragens.

Escalabilidade

Quando você precisa realizar a mesma operação em um grande conjunto de dados, usar um loop é uma abordagem escalável. A quantidade de código necessário não aumenta proporcionalmente ao tamanho dos dados.

Flexibilidade

Loops podem ser controlados por variáveis e expressões, o que permite modificar o comportamento do loop dinamicamente. Isso é útil para cenários em que a lógica de repetição pode mudar.

Redução de Duplicação de Código

A utilização de loops reduz a necessidade de duplicar o mesmo código, o que melhora a manutenção do código e reduz a probabilidade de erros.

Aumento da Produtividade

Automatizar tarefas repetitivas por meio de loops economiza tempo e esforço, permitindo que os desenvolvedores se concentrem em aspectos mais criativos e complexos do projeto.

Legibilidade do Código

Usar loops em vez de repetir manualmente o mesmo bloco de código melhora a legibilidade do código, tornando-o mais claro e fácil de entender.

No entanto, é importante usar loops com cuidado e entender a lógica por trás deles. Looping excessivo ou inadequado pode levar a problemas de desempenho, resultando em programas lentos. Além disso, loops infinitos devem ser evitados, garantindo que a condição do loop se torne falsa em algum momento.



RESUMO

Neste tema nós aprendemos tipos de repetição que são mais utilizados e alguns exemplos de códigos, os laços de repetição são ferramentas poderosas para lidar com tarefas que requerem a execução repetida de um bloco de código. A escolha entre for e while depende das necessidades específicas de cada situação. É fundamental entender a lógica por trás dos laços e aplicá-los de maneira eficaz para criar programas eficientes e funcionais.



ATIVIDADE DE FIXAÇÃO

1. O que são estruturas de repetição ?
2. Qual a diferença entre o loop for e while ?
3. Escreva a sintaxe do loop for, e com setas indique o que são cada parâmetro e qual sua função.
4. O loop for possui três parâmetros, quais são eles?
5. Crie um mapa conceitual com os tipos de estrutura de repetição e suas principais informações
6. Cite 2 exemplos de tarefas cotidianas em que utilizamos a repetição no processo.
7. Crie um script para uma contagem regressiva de 10 a 0 usando os laços for
8. Crie um script para uma contagem regressiva de 10 a 0 usando os laços while
9. Explique com suas palavras explique a principal diferença entre os loops.
10. Observe o código abaixo e descreva o que acontecerá quando executado

```
1 for (var carneirinhos = 2; carneirinhos < 21; carneirinhos++) {  
2     console.log(carneirinhos + " carneirinhos");  
3 }  
4 console.log("fim");
```

Fonte: Elaborado pelo autor (2023).

11. Crie um script que exiba a tabuada do 7, do 1 ao 10, usando o loop for;
**** imagem para consulta somente do professor****

```
1 for ( var i = 0; i < 11; i++ ) {  
2     var resultado = i * 7;  
3     console.log( "7 x" + i + "= " + resultado );  
4 }  
5 console.log( "fim" );  
6
```

Fonte: Elaborado pelo autor (2023).

TEMA 6

Array: Vetor (Variável Composta Unidimensional)

Habilidades

- Entender a manipulação de dados em js
- Compreender a lógica da sintaxe de js
- Praticar criando projetos aplicáveis no ambiente de desenvolvimento

Em programação, a organização e manipulação de conjuntos de dados é uma tarefa essencial. Para isso, o JavaScript oferece uma estrutura de dados chamada "array", também conhecida como "vetor". Arrays permitem armazenar múltiplos valores em uma única variável, tornando a manipulação de dados mais eficiente e organizada.

Os arrays em JavaScript são coleções ordenadas de elementos, onde cada elemento pode ser de qualquer tipo de dado, como números, strings, objetos e até mesmo outros arrays. Eles são usados para armazenar listas de informações relacionadas, como nomes, números, datas, entre outros.

A principal característica de um array é a indexação, onde cada elemento é associado a um índice numérico. Os índices começam em 0 para o primeiro elemento, 1 para o segundo, e assim por diante. Esses índices permitem o acesso rápido e preciso aos elementos do array.

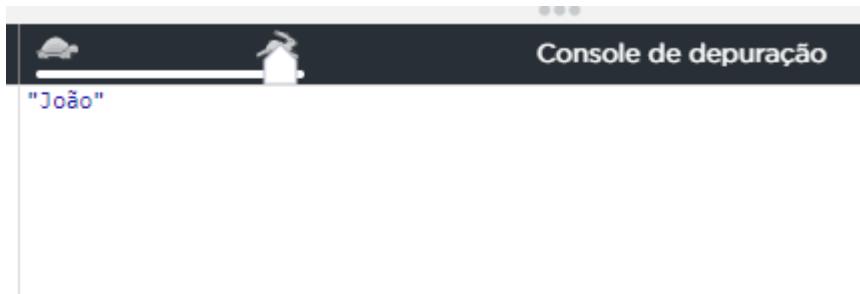
Arrays oferecem uma variedade de métodos e operações que facilitam a adição, remoção, pesquisa e modificação de elementos. Eles são usados em uma ampla gama de cenários, desde processamento de dados até armazenamento temporário de informações.

Sabemos o que é uma variável e qual sua utilização, mas até aqui falamos apenas de variáveis simples, ou seja, que armazenam apenas um valor. Por exemplo preciso armazenar o nome de um aluno, assim é simples, declaro uma variável e faço a atribuição do nome do aluno, veja o exemplo:

```
1 var aluno = "João";
2 console.log(aluno);
3
```

Fonte: Elaborado pelo autor (2023).

Saída:



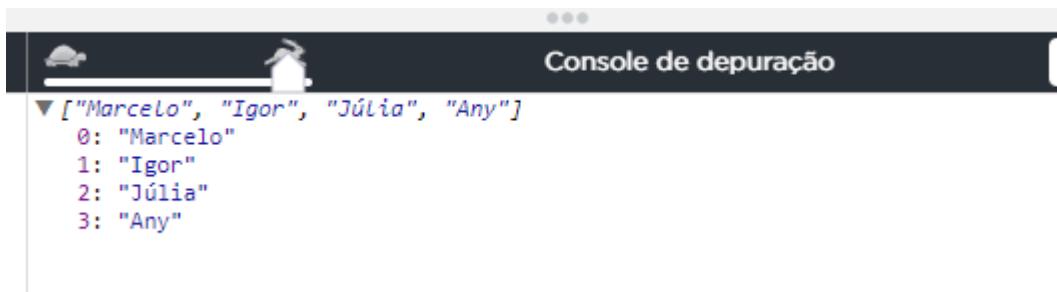
Fonte: Elaborado pelo autor (2023).

O resultado é mostrado no console o nome João, mas se precisarmos armazenar o nome de 10 alunos usando esse mesmo método teria que criar 10 variáveis diferentes para armazenar o nome de todos, e isso não parece ser muito prático, não é verdade? Por isso existem as variáveis compostas, que são variáveis que armazenam mais de um valor. Usando o mesmo exemplo anterior usando uma variável composta:

```
1 var alunos = ["Marcelo", "Igor", "Júlia", "Any"];  
2 console.log(alunos);  
3
```

Fonte: Elaborado pelo autor (2023).

Saída:



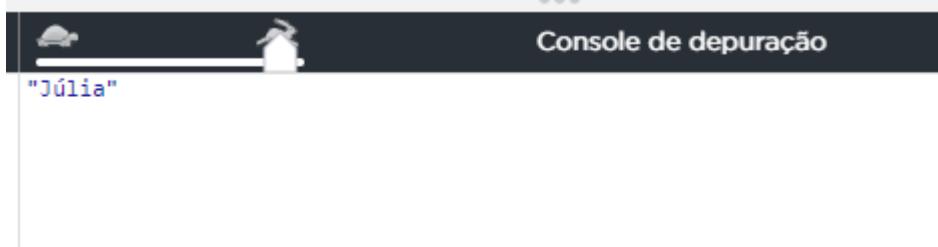
Fonte: Elaborado pelo autor (2023).

Nesse exemplo usando uma variável composta armazenamos nomes de 4 alunos na mesma variável “alunos”. Deste modo podemos armazenar quantos valores forem precisos dentro de uma mesma variável e referenciá-los pelas posições dentro da variável, essas posições são chamadas de índices.

```
1 var alunos = ["Marcelo", "Igor", "Júlia", "Any"];
2 console.log(alunos[2]);
3
```

Fonte: Elaborado pelo autor (2023).

Temos o resultado:



Fonte: Elaborado pelo autor (2023).

Pois dentro da minha variável alunos, o índice 2(primeiro índice) tem o valor “Júlia”. Os índices dentro de um array sempre começam com 0 (primeira posição) e são representados por números inteiros, você não irá encontrar um índice 2,5 por exemplo.

Temos 2 formas de criar um array em Javascript

Deste modo criamos um array vazio, e em seguida inserimos os valores:

```
1 var frutas = new Array ();
2 frutas[0] = "Maça";
3 frutas[1] = "Melância";
4 frutas[2] = "Abacaxi";
5 frutas[3] = "Morango";
6 frutas[4] = "Uva";
7 console.log(frutas);
8
9
```

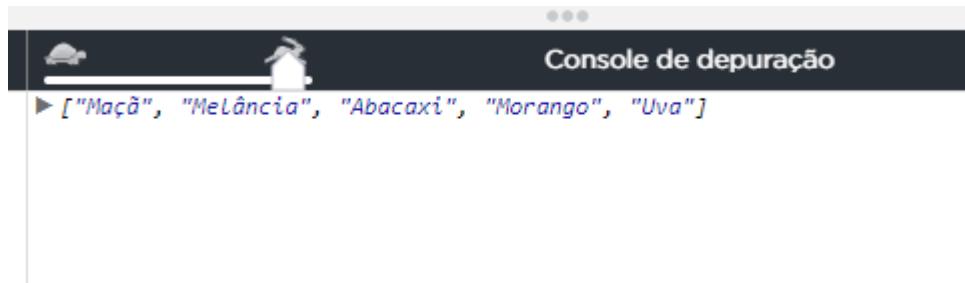
Fonte: Elaborado pelo autor (2023).

Deste modo criamos um array com elementos.

```
1 var frutas = ["Maçã", "Melância", "Abacaxi", "Morango", "Uva"];
2 console.log(frutas);
3
```

Fonte: Elaborado pelo autor (2023).

Estes dois modos têm como resultado a mesma saída de dados



Fonte: Elaborado pelo autor (2023).

Manipulando dados em Arrays

Quando trabalhamos com arrays nos abre um leque de possibilidades enorme quanto a manipulação de dados. Podemos contar quantos elementos possuem no array, como adicionar ou remover valores entre outras opções. Usando o mesmo array (frutas), vamos conhecer as principais manipulações que podemos fazer com arrays no javascript.

Acesso ao item do array

Quantidade de itens dentro do array - Propriedade length

A propriedade length de um array em JavaScript é uma propriedade integrada que indica a quantidade de elementos presentes no array. Essa propriedade é acessada usando a notação de ponto, como nomeDoArray.length, onde nomeDoArray é o nome da variável que armazena o array.

A propriedade length é muito útil para determinar o tamanho de um array dinamicamente, sem a necessidade de contar manualmente os elementos. Aqui estão algumas características importantes da propriedade length

Exemplo:

```
1 var frutas = ["Maçã", "Melancia", "Abacaxi", "Morango", "Uva"];
2 console.log(frutas.length);
3
4
5
6
7
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

Características:

Base 1:

A propriedade `length` é baseada em 1, ou seja, ela contará a quantidade de elementos presentes no array começando do índice 1. Portanto, se um array tiver 5 elementos, a propriedade `length` será igual a 5.

Contagem de Elementos:

A propriedade `length` considera todos os elementos do array, incluindo os valores `undefined` e `null`.

Dinâmico:

A propriedade `length` é dinâmica, o que significa que ela reflete automaticamente as mudanças no número de elementos do array. Quando você adiciona ou remove elementos, a propriedade `length` se ajusta automaticamente.

Não é um Índice:

A propriedade `length` não é um índice válido no array. Se você tentar acessar `array[length]`, obterá `undefined`.

Uso Comum:

A propriedade `length` é frequentemente usada em loops para percorrer todos os elementos de um array, pois fornece o ponto de parada natural para a iteração.

Laço de repetição com array - Laço forEach

O laço de repetição `forEach` é uma forma conveniente de percorrer os elementos de um array em JavaScript. Ele foi projetado especificamente para percorrer arrays e executar uma função de callback em cada elemento do array. O `forEach` é uma alternativa elegante ao uso do laço `for` convencional quando se trata de percorrer elementos em um array.

Aqui está a sintaxe básica do `forEach`:

```
1 array.forEach( function(elemento, indice, arrayCompleto) {  
2     // código a ser executado para cada elemento  
3 } );
```

Fonte: Elaborado pelo autor (2023).

Aqui estão os parâmetros passados para a função de callback:

- `elemento`: O elemento atual sendo processado no array.
- `índice`: O índice do elemento atual no array.
- `arrayCompleto`: O próprio array que está sendo percorrido.

Características do `forEach`:

```
1 var numeros = [1,2,3,4,5];  
2  
3 numeros.forEach( function (item, index){  
4     console.log([index], item);  
5 } );
```

Fonte: Elaborado pelo autor (2023).

saída:

```
"[0]
1"
"[1]
2"
"[2]
3"
"[3]
4"
"[4]
5"
```

Fonte: Elaborado pelo autor (2023).

Iteração Simplificada:

O `forEach` simplifica a interação sobre elementos de um array, eliminando a necessidade de controlar manualmente um contador e acessar elementos por índices.

Síncrono:

O `forEach` é executado de forma síncrona, o que significa que ele percorrerá o array na ordem em que os elementos foram adicionados.

Imutabilidade do Array Original:

O `forEach` não modifica o array original durante a iteração. Ele apenas itera sobre os elementos existentes.

Não Pode Ser Interrompido:

Ao contrário de um loop `for` convencional, o `forEach` não pode ser interrompido usando `break` ou `return`. Ele sempre percorrerá todos os elementos do array.

Boa Prática com Funções Anônimas:

Embora você possa usar funções nomeadas como callbacks, é comum usar funções anônimas ou arrow functions para maior concisão.

Retorno:

O `forEach` não retorna um novo array. Ele é usado principalmente para executar operações nos elementos do array.

O `forEach` é uma ferramenta útil quando você precisa percorrer todos os elementos de um array e realizar uma operação específica em cada um deles. No entanto, se você precisa realizar transformações nos elementos ou criar um novo array com base nos elementos existentes, outras funções como `map`, `filter` e `reduce` podem ser mais apropriadas.

Adicionar um item ao final do array - Método push

O método `push()` é uma função integrada em JavaScript que é usada para adicionar um ou mais elementos ao final de um array existente. Ele modifica o array original, aumentando seu tamanho conforme os novos elementos são inseridos. O `push()` é amplamente utilizado para atualizar arrays com novos dados ou expandi-los dinamicamente.

Exemplo:

```
1 var frutas = ["Maçã", "Melância", "Abacaxi", "Morango", "Uva"];
2 frutas.push("Laranja");
3 console.log(frutas);
4
5
```

Fonte: Elaborado pelo autor (2023).

Outra maneira também o appendItem();

```
1 var frutas = ["Maçã", "Melância", "Abacaxi", "Morango", "Uva"];
2 appendItem(frutas, "Laranja");
3 console.log(frutas);
4
5
```

Fonte: Elaborado pelo autor (2023).

Saída:

```
...  
Console de depuração  
▶ ["Maçã", "Melância", "Abacaxi", "Morango", "Uva", "Laranja"]
```

Fonte: Elaborado pelo autor (2023).

Características do push()

Modifica o Array Original

O método push() modifica diretamente o array original, adicionando os novos elementos ao final dele.

Retorno: O retorno do método push() é o novo comprimento do array após a adição dos elementos.

Aceita Múltiplos Elementos: Você pode passar vários elementos separados por vírgulas como argumentos para o push(). Todos esses elementos serão adicionados ao final do array na ordem em que foram fornecidos.

Dinâmico: O método push() é útil para adicionar novos elementos a um array dinamicamente, sem precisar definir antecipadamente o tamanho do array.

Efeito Colateral: Lembre-se de que o método push() não retorna um novo array, mas sim o novo comprimento do array. Portanto, se você desejar criar um novo array com elementos

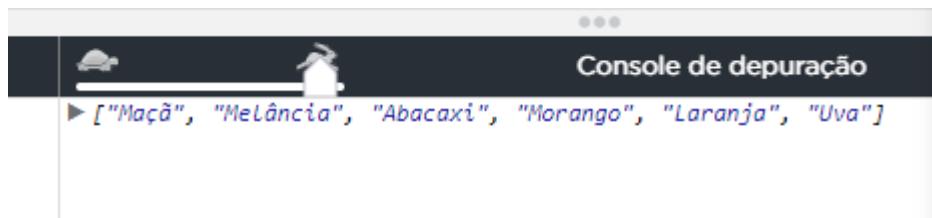
adicionados, pode precisar usar outras abordagens, como a concatenação de arrays.

Outra forma de inserir algum elemento no array é com `insertItem(list, index, item)`; que permite inserir um item na matriz em uma posição específica.

```
1 var frutas = ["Maçã", "Melância", "Abacaxi", "Morango", "Uva"];
2 insertItem(frutas, 4, "Laranja");
3 console.log(frutas);
4
5
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

No exemplo acima foi inserido o item “Laranja”, no índice 4.

Remover o último item do array - Método pop()

O método `pop()` é uma função integrada em JavaScript que é usada para remover o último elemento de um array. Ele modifica o array original, diminuindo seu tamanho após a remoção do elemento. O `pop()` é útil quando você precisa remover o último item de um array e não precisa mais dele.

Aqui está a sintaxe básica do método `pop()`:

```
1 var ultimoElemento = array.pop();
```

Fonte: Elaborado pelo autor (2023).

- `array`: O array do qual você deseja remover o último elemento.
- `ultimoElemento`: A variável que receberá o valor do último elemento removido.

Aplicação:

```
1 var frutas = ['maçã', 'banana', 'laranja'];
2 var ultimoFruta = frutas.pop();
3 console.log(frutas); // Saída: ['maçã', 'banana']
4 console.log(ultimoFruta); // Saída: 'laranja'
```

Fonte: Elaborado pelo autor (2023).

Saída:

```
▶ ["maçã", "banana"]
"laranja"
```

Fonte: Elaborado pelo autor (2023).

Características do pop():

Modifica o Array Original:

O método `pop()` modifica diretamente o array original, removendo o último elemento dele.

Retorno:

O retorno do método `pop()` é o valor do último elemento removido. Esse valor pode ser atribuído a uma variável para posterior uso, se necessário.

Efeito Colateral:

O método `pop()` remove o último elemento do array, o que pode afetar diretamente a estrutura do array, diminuindo seu tamanho.

Array Vazio:

Se o array estiver vazio, o `pop()` retornará `undefined`, pois não há elementos para serem removidos.

O `pop()` é uma maneira eficaz de remover o último elemento de um array quando você precisa atualizar a estrutura do array. No entanto, se você precisa remover elementos de outras posições ou deseja manter um registro dos elementos removidos, pode ser necessário usar outros métodos como `splice()` ou `shift()`.

Remover o primeiro item do array - Método shift

O método `shift()` é uma função integrada em JavaScript que é usada para remover o primeiro elemento de um array. Assim como o `pop()`, o `shift()` também modifica o array original, diminuindo seu tamanho após a remoção do elemento. O `shift()` é útil quando você precisa remover o primeiro item de um array e não precisa mais dele.

Aqui está a sintaxe básica do método `shift()`:

```
1 var primeiroElemento = array.shift();  
2
```

Fonte: Elaborado pelo autor (2023).

- `array`: O array do qual você deseja remover o primeiro elemento.
- `primeiroElemento`: A variável que receberá o valor do primeiro elemento removido.

Aplicação:

```
1 var frutas = ['maçã', 'banana', 'laranja'];  
2 var primeiraFruta = frutas.shift();  
3 console.log(frutas); // Saída: ['banana', 'laranja']  
4 console.log(primeiraFruta); // Saída: 'maçã'  
5
```

Fonte: Elaborado pelo autor (2023).

Saída:

Console de depuração

```
▶ ["banana", "laranja"]
"maçã"
```

Fonte: Elaborado pelo autor (2023).

Características do `shift()`:

Modifica o Array Original:

O método shift() modifica diretamente o array original, removendo o primeiro elemento dele.

Retorno:

O retorno do método shift() é o valor do primeiro elemento removido. Esse valor pode ser atribuído a uma variável para posterior uso, se necessário.

Efeito Colateral:

O método shift() remove o primeiro elemento do array, o que afeta diretamente a estrutura do array, diminuindo seu tamanho.

Array Vazio:

Se o array estiver vazio, o shift() retornará undefined, pois não há elementos para serem removidos.

Assim como o pop(), o método shift() é útil quando você precisa remover elementos de um array e atualizar sua estrutura. No entanto, lembre-se de que, se você precisar remover elementos de outras posições ou desejar manter um registro dos elementos removidos, outros métodos como splice() ou pop() podem ser mais adequados.

Adicionar um item ao início do array - Método unshift()

O método unshift() é uma função integrada em JavaScript que é usada para adicionar um ou mais elementos ao início de um array existente. Ao contrário do push(), que adiciona elementos ao final do array, o unshift() insere elementos no início do array. Ele modifica o array original, aumentando seu tamanho conforme os novos elementos são inseridos no início.

```
1 var frutas = ['maçã', 'banana', 'laranja'];
2
3 frutas.unshift('uva', 'morango');
4 console.log(frutas); // Saída: ['uva', 'morango', 'maçã', 'banana', 'laranja']
```

Fonte: Elaborado pelo autor (2023).

Características do unshift()

Modifica o Array Original:

O método unshift() modifica diretamente o array original, adicionando os novos elementos no início dele.

Retorno:

O retorno do método unshift() é o novo comprimento do array após a adição dos elementos.

Aceita Múltiplos Elementos:

Você pode passar vários elementos separados por vírgulas como argumentos para o unshift(). Todos esses elementos serão adicionados ao início do array na ordem em que foram fornecidos.

Dinâmico:

O método `unshift()` é útil para adicionar novos elementos ao início de um array dinamicamente.

Efeito Colateral:

Lembre-se de que o método `unshift()` modifica o array original, o que pode afetar a ordem e a estrutura dos elementos.

O `unshift()` é uma maneira conveniente de adicionar elementos ao início de um array, especialmente quando você deseja manter a ordem dos elementos existentes e inserir novos elementos no início.

Encontrar o índice de um item no array - Método `indexOf()`

O método `indexOf()` é uma função integrada em JavaScript que é usada para encontrar o índice de um determinado elemento em um array. Ele retorna o primeiro índice em que o elemento é encontrado ou `-1` se o elemento não estiver presente no array. O `indexOf()` é muito útil para verificar a presença de um elemento e obter sua posição dentro do array.

Aqui está a sintaxe básica do método `indexOf()`:

```
1 var indice = array.indexOf(elemento);  
2
```

Fonte: Elaborado pelo autor (2023).

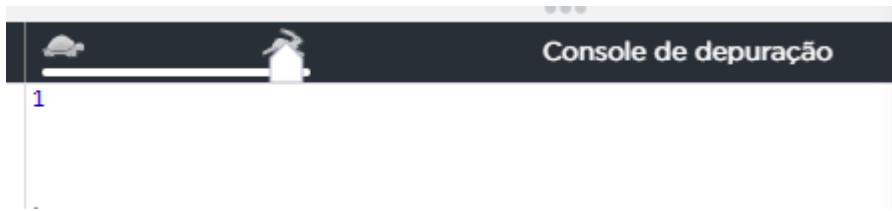
- `array`: O array no qual você deseja procurar o elemento.
- `elemento`: O elemento que você deseja encontrar no array.

Exemplo:

```
1 var frutas = ['maçã', 'banana', 'laranja'];  
2 var indiceBanana = frutas.indexOf('banana');  
3 console.log(indiceBanana); // Saída: 1  
4
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

Se o elemento não estiver presente no array, o método `indexOf()` retornará -1:

Características do `indexOf()`

Retorno do Índice:

O método `indexOf()` retorna o índice do primeiro elemento encontrado no array. Se o elemento não for encontrado, ele retornará -1.

Busca Linear:

O `indexOf()` realiza uma busca linear no array, começando do início até encontrar o primeiro elemento correspondente.

Valor Estrito (==):

O `indexOf()` compara os elementos usando o operador de igualdade estrita (==). Isso significa que ele verifica tanto o valor quanto o tipo de dados do elemento.

Busca a Partir de um Índice:

O `indexOf()` também aceita um segundo parâmetro opcional que especifica o índice a partir do qual a busca deve começar.

Use `includes()` para Verificação de Presença:

Se você estiver apenas interessado em verificar se um elemento está presente no array, o método `includes()` pode ser uma alternativa mais simples.

O `indexOf()` é uma ferramenta valiosa para verificar a presença de um elemento em um array e obter seu índice. Lembre-se de que, se você estiver interessado em encontrar todos os índices de um elemento repetido, pode precisar usar um loop ou outros métodos mais avançados.

Remover um item pela posição do índice - Método `Splice(indice, quantidade)`

O método `splice()` é uma função integrada em JavaScript que é usada para remover ou substituir elementos de um array, com base em sua posição de índice. Ele também pode ser usado para inserir novos elementos no array. O `splice()` modifica o array original, alterando sua estrutura conforme os elementos são adicionados ou removidos.

Aqui está a sintaxe básica do método `splice()` para remover elementos:

```
1 array.splice(indice, quantidade); ← →  
2
```

Fonte: Elaborado pelo autor (2023).

- array: O array do qual você deseja remover elementos.
- índice: O índice do primeiro elemento a ser removido.
- quantidade: O número de elementos a serem removidos a partir do índice.

Exemplo:

```
1 var frutas = ['maçã', 'banana', 'laranja', 'uva', 'morango']; ← →  
2  
3 frutas.splice(2, 2); ← →  
4 console.log(frutas); // Saída: ['maçã', 'banana', 'morango'] ← →  
5
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools with the 'Console de depuração' (Debug Console) tab selected. The console window displays the output of the JavaScript code: `["maçã", "banana", "morango"]`.

Fonte: Elaborado pelo autor (2023).

No exemplo acima, o método `splice()` remove dois elementos a partir do índice 2, ou seja, remove 'laranja' e 'uva'.

Você também pode usar o método `splice()` para substituir elementos:

```
1 var frutas = ['maçã', 'banana', 'laranja', 'uva', 'morango']; ← →  
2  
3 frutas.splice(1, 2, 'abacate', 'kiwi'); ← →  
4 console.log(frutas); // Saída: ['maçã', 'abacate', 'kiwi', 'uva', 'morango'] ← →  
5
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools console titled "Console de depuração". It displays the command `["maçã", "abacate", "kiwi", "uva", "morango"]` followed by its output, which is another array: `["maçã", "abacate", "kiwi", "uva", "morango"]`. This indicates that the splice operation did not change the original array.

Fonte: Elaborado pelo autor (2023).

Nesse exemplo, o método `splice()` substitui 'banana' e 'laranja' por 'abacate' e 'kiwi', respectivamente.

Características do `splice()`

Modifica o Array Original:

O método `splice()` modifica diretamente o array original, adicionando, removendo ou substituindo elementos.

Retorno dos Elementos Removidos:

O `splice()` retorna um novo array contendo os elementos removidos, ou um array vazio se nenhum elemento for removido.

Inserir Elementos:

Além de remover e substituir elementos, você pode usar o `splice()` para inserir novos elementos no array.

Efeito Colateral:

Lembre-se de que o método `splice()` afeta diretamente a estrutura do array, adicionando, removendo ou substituindo elementos.

O `splice()` é uma ferramenta poderosa para manipular elementos em um array, permitindo que você remova, substitua ou insira elementos com base em índices específicos.

Array - Matriz (*Variável Composta Multidimensional*)

Uma matriz em JavaScript é uma estrutura de dados que armazena uma coleção de elementos organizados em linhas e colunas. Ao contrário de um array unidimensional, que é uma lista simples de elementos, uma matriz é uma estrutura bidimensional que permite armazenar informações em um formato de tabela ou grade. Cada elemento em uma matriz é acessado por um par de índices: um para a linha e outro para a coluna.

Sintaxe da matriz:

```

1 var matriz = [
2   [elemento1, elemento2, elemento3],
3   [elemento4, elemento5, elemento6],
4   [elemento7, elemento8, elemento9]
5 ] ;
6

```

Fonte: Elaborado pelo autor (2023).

Cada conjunto de colchetes internos representa uma linha da matriz, e os elementos individuais em cada linha são separados por vírgulas.

Acesso aos elementos

Para acessar um elemento específico em uma matriz, você usa o índice da linha e o índice da coluna:

```

1 var matriz = [
2   [1, 2, 12],
3   [4, 5, 6],
4   [7, 8, 9]
5 ] ;
6
7 console.log(matriz[1][2]); // Saída: 6 (linha 1, coluna 2)
8

```

Fonte: Elaborado pelo autor (2023).

Saída

The screenshot shows the Scratch IDE's debug console window titled "Console de depuração". It displays the number "6" in blue text, indicating the result of the console log statement.

Fonte: Elaborado pelo autor (2023).

Neste exemplo, matriz[1][2] retorna o valor 6, que está na segunda linha e terceira coluna da matriz.

Aplicações de Matrizes

As matrizes são frequentemente usadas para representar dados tabulares, como tabelas de registros ou imagens pixeladas. Por exemplo, você pode usar uma matriz para representar uma planilha de dados, uma grade de pixels em uma imagem ou até mesmo coordenadas em um sistema de coordenadas cartesianas.

Iteração em Matrizes

A iteração em matrizes refere-se ao processo de percorrer todos os elementos de uma matriz e realizar alguma operação em cada um deles. Isso é feito usando loops, como for ou while, para visitar cada linha e coluna da matriz e executar um determinado bloco de código para cada elemento.

A iteração em matrizes é essencial para realizar tarefas como processamento de dados, cálculos estatísticos, busca por elementos específicos ou qualquer outra manipulação que envolva todos os elementos da matriz.

De modo geral, iterar é repetir uma ação por um determinado de espaço de tempo ou até uma condição for alcançada. Esses tipos de repetições fazem o laço repetir por toda uma sequência mostrando os resultados, é muito usado para pesquisar listas de dados, ou até mesmo um array. Por exemplo, vamos entender que vou usar um array com dados de alunos (usado na seção anterior) onde temos de início apenas o nome dos alunos, com um laço iterativo consigo mostrar todos os elementos desse array.

Exemplos de Iteração em Matrizes

Vamos supor que temos a seguinte matriz que representa uma tabela de notas de alunos.

```
1 var notas = [
2     [8, 7, 9],
3     [6, 5, 8],
4     [9, 9, 10]
5 ]
6
7
```

The Scratch script shows a variable named "notas" set to a list of three lists. The inner lists contain the values [8, 7, 9], [6, 5, 8], and [9, 9, 10] respectively. The script ends with a control block to stop the loop.

Fonte: Elaborado pelo autor (2023).

Imprimir todos os elementos:

```
1 var notas = [ [8, 7, 9], [6, 5, 8], [9, 9, 10] ];
2 
3 for (var i = 0; i < notas.length; i++) {
4     for (var j = 0; j < notas[i].length; j++) {
5         console.log(notas[i][j]);
6     }
7 }
```

Fonte: Elaborado pelo autor (2023).

Console de depuração

```
8
7
9
6
5
8
9
9
10
```

Saída:

Fonte: Elaborado pelo autor (2023).

Calcular a média de cada aluno

```

for (var i = 0; i < notas.length; i++) {
    var soma = 0;
    for (var j = 0; j < notas[i].length; j++) {
        soma += notas[i][j];
    }
    var media = soma / notas[i].length;
    console.log("Média do aluno " + i + media.toFixed(2));
}

```

Fonte: Elaborado pelo autor (2023).

Saída:

```

Média do aluno08.00
Média do aluno16.33
Média do aluno29.33

```

Fonte: Elaborado pelo autor (2023).

Em resumo, a iteração em matrizes envolve usar loops aninhados para percorrer cada linha e coluna da matriz e realizar operações específicas em cada elemento. Isso é uma parte fundamental da programação quando se trabalha com dados organizados em estruturas bidimensionais.

Manipulando dados de um array multidimensional (matriz)

Já vimos que a manipulação de dados em arrays é uma parte fundamental da programação, permitindo que você organize, modifique, insira ou remova elementos de uma coleção de forma dinâmica. Em JavaScript, os arrays são uma das estruturas de dados mais versáteis e amplamente usadas, e existem várias operações que você pode realizar para manipular seus dados.

A semelhança entre uma matriz comum e uma matriz multidimensional é que ambas são estruturas de dados que permitem armazenar coleções de elementos em uma estrutura organizada. Ambos os tipos de matriz compartilham alguns conceitos básicos, mas a diferença chave é a dimensão e a estrutura dos elementos armazenados.

As semelhanças estão na organização Estruturada. Tanto as matrizes comuns quanto as multidimensionais permitem a organização de dados de maneira estruturada, onde os elementos são acessados por meio de índices.

Em ambos os tipos de matrizes, você pode acessar os elementos individuais usando índices numéricos. Os índices indicam a posição de um elemento na matriz. E também em ambas as matrizes, você pode realizar operações como inserção, remoção e atualização de elementos, dependendo das capacidades da linguagem de programação.

A principal diferença é a dimensão das matrizes. Uma matriz comum é unidimensional, contendo uma única linha de elementos. Uma matriz multidimensional tem duas ou mais dimensões, formando uma estrutura de linhas e colunas (ou até mais dimensões).

Além de que em uma matriz comum, os elementos são organizados em uma única sequência linear. Em uma matriz multidimensional, os elementos são organizados em uma estrutura hierárquica de linhas e colunas (ou mais dimensões).

Em matrizes comuns, você acessa os elementos diretamente usando um único índice. Em matrizes multidimensionais, você usa vários índices correspondentes às várias dimensões para acessar elementos específicos.

Matrizes multidimensionais são mais complexas do que matrizes unidimensionais, pois envolvem mais índices e níveis de aninhamento. Isso pode aumentar a complexidade ao acessar, percorrer ou manipular os elementos.

As matrizes unidimensionais são frequentemente usadas para armazenar listas simples de itens, enquanto matrizes multidimensionais são mais adequadas para representar estruturas mais complexas, como tabelas, imagens, dados 2D/3D, entre outras.

Portanto, embora as matrizes comuns e multidimensionais compartilhem alguns conceitos básicos, a principal diferença entre elas é a quantidade de dimensões e a forma como os elementos são organizados e acessados. As matrizes multidimensionais oferecem maior flexibilidade na representação de estruturas de dados mais complexas.



RESUMO

Neste tema sobre arrays em JavaScript, exploramos os conceitos fundamentais relacionados a essa estrutura de dados versátil. Descobrimos como criar, acessar e manipular elementos em arrays, além de explorar métodos avançados que facilitam a tarefa de lidar com conjuntos de dados de maneira eficaz. Com os arrays em mãos, você estará preparado para organizar e otimizar suas tarefas de programação de forma mais eficiente e estruturada.

Também aprendemos como aplicar esses conceitos de forma eficaz e eficiente. Arrays são ferramentas poderosas para armazenar e manipular conjuntos de dados em linguagens de programação. As operações de manipulação de dados permitem que os programadores acessem, modifiquem e processem informações de maneira eficiente e flexível.



ATIVIDADE DE FIXAÇÃO

1. O que são arrays?

2. Quais as palavras e símbolos utilizados na sintaxe da estrutura de uma array?
3. Complete a sintaxe abaixo com os símbolos corretos:

```
// Criando um array de números  
var numeros = __ 1, 2, 3, 4, 5] ;  
  
// Imprimindo o array no console  
console.__ ("Array de números:" __ numeros);  
  
// Acessando elementos do array  
console.__("Primeiro elemento:"+ numeros[0]);  
console.__("Segundo elemento:" __ numeros[1]);  
  
// Adicionando um elemento ao final do array  
numeros.push(6);  
console.log("Array após adicionar 6:"+ numeros);  
  
// Removendo o último elemento do array  
numeros.pop();  
console.log("Array após remover o último elemento:" + numeros);  
  
// Verificando o tamanho do array  
_____ .log("Tamanho do array:" + numeros.length); // Saída: 5
```

// Iterando sobre o array usando um loop

```
for (var i __ 0; i < numeros.length; i++) __  
    console.log("Elemento" + i + " : " + numeros[i]);  
}
```

4. Crie um array de números e calcule a soma de todos os elementos
5. Crie um loop for para concatenar as arrays “primeirosNomes” e “ultimosNomes”

```
1 //Concatene os arrays 'primeirosNomes' e 'ultimosNomes'  
2 //para criar um array 'nomesCompletos'.  
3 var primeirosNomes = ['Maria', 'João', 'Pedro'] __ __;  
4 var ultimosNomes = ['Silva', 'Santos', 'Ferreira'] __ __;  
5 __ ]
```

Fonte: Elaborado pelo autor (2023).

The screenshot shows a browser's developer tools with a dark-themed interface. At the top, there are two small icons: a gear and a house. To the right of the icons, the title "Console de depuração" is displayed. Below the title, there is a list of three items, each preceded by a blue right-pointing arrowhead: `["Maria Silva"]`, `["João Santos"]`, and `["Pedro Ferreira"]`.

Fonte: Elaborado pelo autor (2023).

6. Crie um programa com uma array com a finalidade de armazenar o nome e a idade das pessoas que fazem parte da sua família.

7. Qual comando é utilizado para acrescentar um item no final da array da Família ? escreva a sintaxe abaixo com um exemplo.

8. Qual comando é utilizado para acrescentar um item no início da array Família ? escreva a sintaxe abaixo com um exemplo.

9. Qual comando é utilizado para excluir um item da array Família? escreva a sintaxe abaixo com um exemplo.

10. Analise o seguinte código escrito na linguagem Javascript:

```
var frutas = ["banana", "laranja", "limão"];
console.log(frutas[2] + " | ");
```

O resultado correspondente apresentado como saída é:

- a) banana | laranja | limão
- b) limão |
- c) | laranja |
- d) laranja | uva | limão

TEMA 07

Funções

Habilidades

- Conhecer e entender a estrutura modular da função e suas aplicações
- Definir funções usando a palavra-chave function e a chamá-las com argumentos
- Resolução de problemas para dividir problemas complexos em partes menores

As funções são blocos de construção muito usados na programação. Uma função é um conjunto de instruções que executa uma determinada tarefa e devolve como retorno um resultado. Em algumas linguagens as funções também são conhecidas como módulos, a ação de modularizar um código é otimizar o código deixando mais legível e limpo, visando sempre a integridade e segurança, em outras palavras é separar o código em pequenos blocos, assim deixando o código mais fácil de entender. Vamos ver um exemplo, vou criar uma função para verificar um número qualquer e me devolver se esse número é par ou ímpar. Posso fazer esse procedimento de duas maneiras.

Fazer tudo no código principal:

```
1 var n = promptNum ("digite um número");
2 if (n%2 ==0 ){
3     console.log("Par");
4 }else{
5     console.log("Impar");
6 }
```

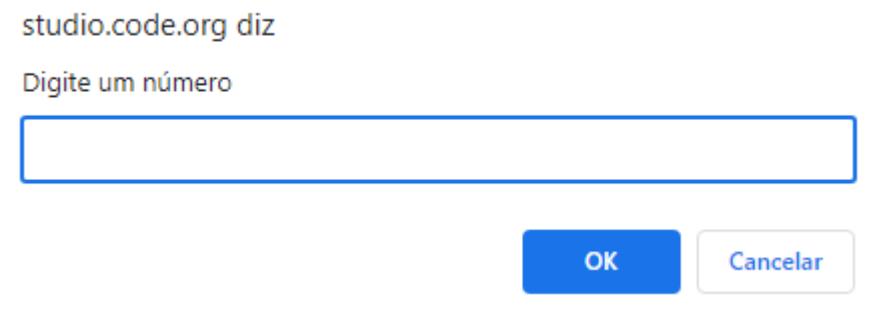
Fonte: Elaborado pelo autor (2023).

```
1 function parImpar(n) { ← →
2     if ((n%2)==0) {
3         console.log("Par");
4     }else{
5         console.log("Impar");
6     }
7 }
8 var resposta = parImpar (promptNum ("digite um número")) ← →;
```

Fonte: Elaborado pelo autor (2023).

Esse é o código dentro da estrutura principal do programa, imagine programas mais complexos, onde temos milhares de funções, a leitura desses códigos seria cansativa e estressante e muito improdutiva. Agora vamos usar o mesmo programa acima , mas separando a função do programa principal:

SAÍDA:



Fonte: Elaborado pelo autor (2023).

O número deve ser digitado pelo usuário e a saída será de acordo com o respectivo número escolhido.

Nesse exemplo ficou claro que separar nossas “rotinas” em funções e deixá-las separadas do código principal. A declaração de uma função é muito importante pois é através dela que os comandos são executados.

Resumindo, para criar uma função é preciso atribuir um nome a ela de acordo com sua tarefa específica e chamá-la no momento certo.

Sintaxe:

*Nome da Função (parâmetros){ Bloco de comando
}*

Exemplo:

```
function dobro(num) { return num * 2  
}
```

A função dobro recebe um parâmetro chamado “num”, no qual é passado através da chamada da função, veja no exemplo, a chamada para a função dobro.

The screenshot shows a code editor with the following content:

```
1 function dobro(num) {  
2     return num * 2;  
3 }  
4 console.log(dobro(5));
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser developer tools console titled "Console de depuração". It displays the number "10" as the output of the function call.

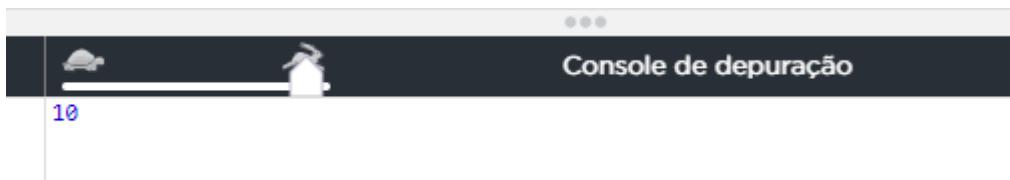
Fonte: Elaborado pelo autor (2023).

A declaração acima chama a função e passa como parâmetro o número cinco, nesse caso a função executa as instruções e retorna o valor dez. Parâmetros (número, por exemplo) são passados para as funções por valor, como o exemplo o número cinco foi passado para a função, poderia fazer de outras formas pedindo para um usuário digitar e armazenar em uma variável e passar essa variável como parâmetro. Como mostra o exemplo abaixo:

```
1 var dobro = function (num) {  
2     return num*2 ;  
3 } ;  
4 var resposta = dobro(5) ;  
5 console.log(resposta);  
6
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

Em javascript podemos outros tipos de funções , como a expressão de função, onde a função não possui nome, ou seja, ela é uma função anônima, veja um exemplo com a função dobro:

```
1 var dobro = function (num) {  
2     return num*2 ;  
3 } ;  
4 var numero = prompt("Digite um número");  
5 var resposta = dobro(numero) ;  
6 console.log(resposta);  
7
```

Fonte: Elaborado pelo autor (2023).

Saída:

studio.code.org diz

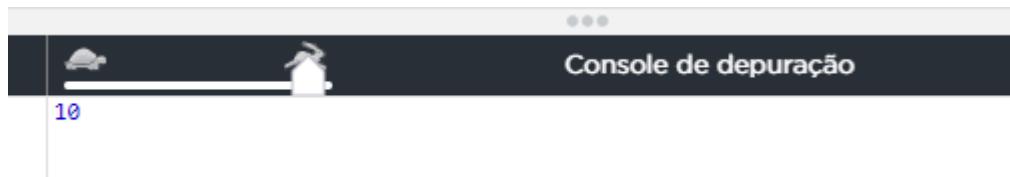
Digite um número

5

OK

Cancelar

Fonte: Elaborado pelo autor (2023).



Fonte: Elaborado pelo autor (2023).

Podemos ver no exemplo acima a função dobro foi declarada dentro do programa principal, inclusive em uma única linha sem precisar de chamada por ser uma função anônima. Em alguns casos não precisamos deixar uma função separada pois é uma tarefa muito simples e rápida, então quem decide qual momento usar ou não uma função ou até mesmo se usar uma função nominal ou anônima é o desenvolvedor, por isso é importante conhecer e praticar essas funções para entender e tomar as melhores decisões ao programar.

Função de Flecha (Arrow Function)

A função de flecha nada mais é que uma simplificação da expressão de função, como geralmente usamos bastante funções em nossos programas, nada melhor e mais inteligente do que criar atalhos e sintaxes menos formais, não é? Por este motivo, a criação de funções de flecha facilita a criação e o uso das funções permitindo a criação de maneira resumida. Vamos ao exemplo abaixo:

```
1 // Função com a biblioteca Math
2 // para gerar um numero aleatório Random(aleatório)
3 var numeroAleatorio = function() {
4     return Math.random();
5 }
6 // Programa principal
7 console.log(numeroAleatorio());
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools with the title "Console de depuração". It displays the output of a JavaScript command: "0.8305006983278638".

Fonte: Elaborado pelo autor (2023).

No exemplo mostrado criamos uma função para gerar um número aleatório, é fácil notar que usamos mais de uma linha de código para declarar essa função. Com a função em flecha podemos simplificar a mesma função, declarando-a na mesma linha, elas foram introduzidas na especificação ECMAScript 6 (ES6) e oferecem uma maneira mais sucinta de criar funções, tornando o código mais legível e fácil de entender. Aqui está uma explicação mais detalhada sobre as arrow functions:

The screenshot shows a code editor with the following content:

```
1 // Função com a biblioteca Math
2 // para gerar um numero aleatório Random(aleatório)
3 // de forma simplificada com a função flecha */
4 var numeroAleatorio = (Math.random());
5
6 // Programa principal
7 console.log(numeroAleatorio);
8
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools with the title "Console de depuração". It displays the output of a JavaScript command: "0.043474267670150724".

Ficou claro que é muito mais simples usar a função em flecha, repare que removemos o nome da função e a palavra “return” e as chaves {}. Caso o corpo da função de flecha tenha apenas uma linha podemos omitir as chaves, e não precisamos utilizar a palavra return, podemos removê-lo, pois a primeira linha será executada e retornada automaticamente.

Função construtora

Uma função construtora é definida e declarada como qualquer outra função, a forma de utilizar é a mesma, a diferença está na situação de uso e o que ela retorna. A função construtora é normalmente iniciada com letra maiúscula, para diferenciar de uma função padrão.

```
function Carro() { bloco de comando;}
```

Nesse exemplo declaramos uma função construtora que irá criar um objeto “Carro”. Para entender o que é um objeto em javascript, devemos lembrar que o javascript é uma linguagem com paradigma orientado a objeto, ou seja, um objeto é uma coleção de atributos que fazem uso de métodos. Vamos a um exemplo, um carro é um objeto onde tem propriedades(motor, câmbio, rodas etc.) e usam métodos(ligar motor, acender faróis etc.), entendemos então que um objeto contém propriedades, que também podem ser chamadas de atributos, e usam métodos(funções). Objetos em javascript podem ser comparados com objetos na realidade, pois possui propriedades que são associadas a ele, e que também podem ter propriedades que definem suas próprias características. Uma propriedade pode ser entendida semelhante a uma variável ligada ao objeto. Acessamos a propriedade de um objeto com uma simples notação de ponto:

nomeObjeto.nomePropriedade

Vamos ver o exemplo de como usar criar um objeto, vamos criar o objeto carro com as propriedades marca, modelo, cor e ano:

```
1 var carro = new Object() →;
2 carro.marca = "Toyota";
3 carro.modelo = "Corolla";
4 carro.ano = 2023;
5 console.log(carro);
6
7
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools open to the 'Console de depuração' (Debug Console) tab. The console displays the output of the JavaScript code execution. The output is a single line of text: 'Object {marca: "Toyota", modelo: "Corolla", ano: 2023}'.

Fonte: Elaborado pelo autor (2023).

Criamos um objeto “carro” com essas três propriedades, as propriedades que não foram definidas inicialmente são consideradas indefinidas “undefined”. Podemos acessar ou alterar as propriedades de um objeto em javascript usando [] (colchetes):

```
1 var carro = new Object()
2 carro.marca = "Toyota";
3 carro.modelo = "Corolla";
4 carro.ano = 2023;
5 carro["marca"] = "Ford";
6 console.log(carro);
7
```

Fonte: Elaborado pelo autor (2023).

Saída:

```
Object {marca: "Ford", modelo: "Corolla", ano: 2023}
```

Fonte: Elaborado pelo autor (2023).

Agora que temos o conhecimento do que é um objeto podemos voltar às funções construtoras.

Vamos criar o mesmo objeto “carro” usando uma função construtora:

```
1 function Carro (marca, modelo, ano) {
2     this.marca = marca;
3     this.modelo = modelo;
4     this.ano = ano;
5 }
6 console.log(Carro);
7
```

Fonte: Elaborado pelo autor (2023).

Usamos a palavra “this” para fazer a atribuição de valores às propriedades do objeto “carro”,

usando de base os valores passados na função. Vamos criar outro objeto carro, agora que temos uma função construtora definida:

Esse código cria um objeto carro e faz a atribuição a ele de valores definidos para suas propriedades, seguindo a mesma ordem da função, notamos que “Volkswagen” é a marca, “Camaro” é o modelo e ano 2018.

```
1 function Carro (marca, modelo, ano){  
2     this.marca = marca;  
3     this.modelo = modelo;  
4     this.ano = ano;  
5 }  
6 var carro = new Carro("Volkswagen", "Camaro", 2018);  
7 console.log(carro);  
8  
9
```

Fonte: Elaborado pelo autor (2023).

Saída:

The screenshot shows a browser's developer tools with the title "Console de depuração". It displays the output of a JavaScript command: "Object {marca: "Volkswagen", modelo: "Camaro", ano: 2018}".

Fonte: Elaborado pelo autor (2023).

DOM (Document Object Model)

O DOM (Document Object Model) em JavaScript é uma representação estruturada em forma de árvore de todos os elementos e conteúdos presentes em um documento HTML ou XML. Ele permite que scripts em JavaScript interajam e manipulem os elementos, estilos e conteúdo de uma página da web de forma dinâmica. O DOM é uma das partes fundamentais da programação web e é amplamente utilizado para criar páginas interativas e responsivas.

Como o DOM Funciona:

O DOM representa cada elemento HTML como um "nó" na árvore, onde o nó raiz representa o documento inteiro. Cada elemento, como tags HTML, atributos e texto, é representado como um nó no DOM. Isso significa que cada parte da página, incluindo elementos, atributos e até mesmo o próprio texto, pode ser acessada e manipulada por meio de código JavaScript.

Manipulação do DOM:

A manipulação do DOM envolve a interação com esses nós usando JavaScript para realizar ações como:

Acessar Elementos: Usando seletores (como getElementById, querySelector, getElementsByTagName, etc.), você pode acessar elementos específicos da página.

Alterar Conteúdo: Você pode alterar o texto, atributos e conteúdo HTML dos elementos.

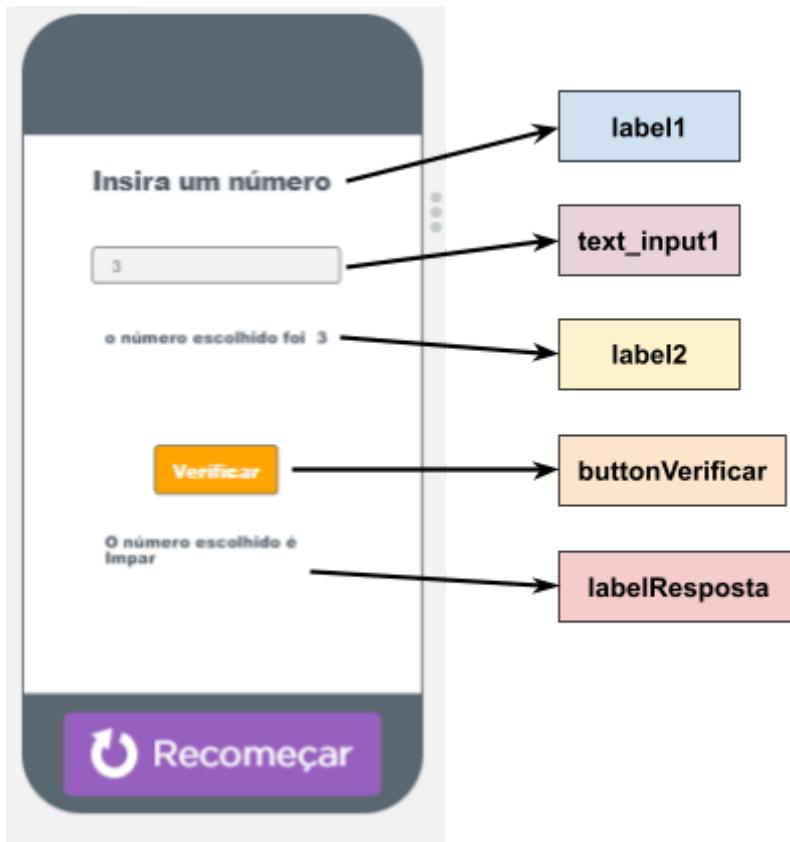
Manipular Estilos: É possível modificar as propriedades de estilo (como cor, tamanho, margens, etc.) dos elementos.

Adicionar e Remover Elementos: Você pode adicionar novos elementos à página ou remover elementos existentes.

Responder a Eventos: Você pode definir manipuladores de eventos para executar ações quando eventos, como cliques ou submissões de formulários, ocorrerem.

Vamos observar um exemplo em que um app recebe do usuário um número no elemento text_input1 e retorna uma resposta. Um elemento é geralmente definido como um “id”, usado para identificar exclusivamente o elemento dentro do documento.

Design:



Fonte: Elaborado pelo autor (2023).

Código:

```
1 onEvent("text_input1", "input", function() {
2     setText("label2", "o número escolhido foi " + getText("text_input1"));
3 });
4 onEvent("buttonVerificar", "click", function() {
5     if (getText("text_input1") % 2 == 0) {
6         setText("labelresposta", "O número escolhido é Par");
7     } else {
8         setText("labelresposta", "O número escolhido é Ímpar");
9     }
10});
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

No exemplo acima podemos observar a manipulação dos elementos através dos comandos

`setText id() ("text")` e `getText (id)`. O conteúdo da entrada de texto é recuperado usando `getText("text_input1")` e o resultado é exibido na etiqueta com o ID "label2" usando `setText("label2", ...)`.

Em outras linguagem com o HTML , o javaScript também acessa os elemento do documento pelo id como o exemplo `document.getElementById();`

Eventos e Closures

Um evento, no contexto de uma página web, pode ser compreendido como uma sequência de ações que ocorrem dentro de um elemento específico, como um pedaço de texto, uma imagem ou até mesmo uma divisão (div). Essas ações estão intrinsecamente relacionadas às interações do usuário com um documento web, o que significa que cada interação do usuário é potencialmente um evento.

Por exemplo, quando um usuário pressiona uma tecla enquanto está focado em um elemento da página, isso desencadeia um evento associado a esse elemento, conhecido como "keydown" (tecla pressionada). Quando o usuário solta a tecla, outro evento denominado "keyup" (tecla liberada) é ativado. Da mesma forma, quando um usuário clica em um botão na página, isso resulta na ativação de um evento específico para esse botão, chamado "click" (clique).

Os eventos podem representar uma ampla gama de interações, desde as interações básicas mencionadas nos exemplos acima até notificações automáticas e respostas a ações do usuário. Eles são fundamentais para a dinâmica de uma página web, permitindo que a página responda de forma ágil e interativa às ações dos usuários, melhorando assim a experiência geral do usuário na navegação na web. Existe uma grande variedade de eventos definidos para uso em Javascript, vamos conhecer os principais e mais utilizados:

onBlur	remove o foco do elemento
onChange	muda o valor do elemento
onClick	o elemento é clicado pelo usuário
onFocus	o elemento é focado
onKeyPress	o usuário pressiona uma tecla sobre o elemento
onLoad	carrega o elemento por completo
onMouseOver	define ação quando o usuário passa o mouse sobre o elemento
onMouseOut	define ação quando o usuário retira o mouse sobre o elemento

Existem algumas formas de se aplicar esses e v e n to s em javaScript. Vamos usar o exemplo dado anteriormente:

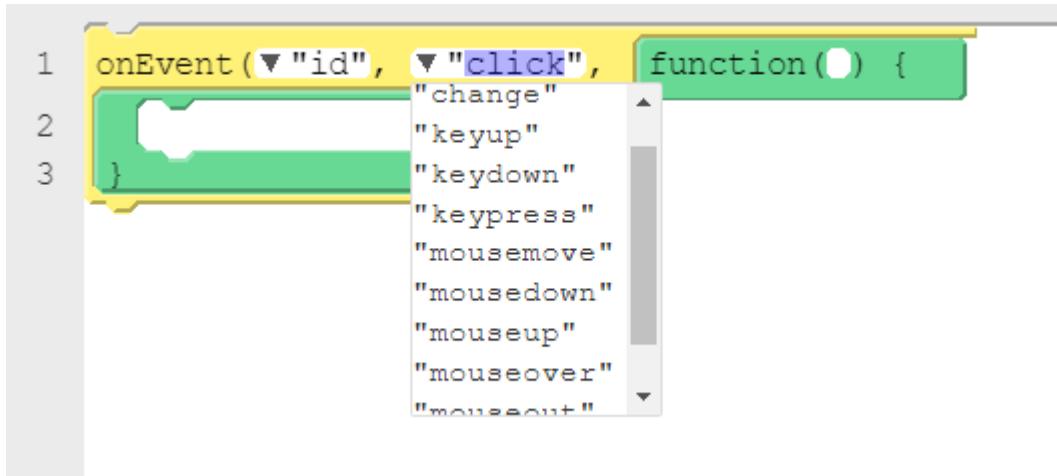
```

1 onEvent (▼ "text_input1", ▼ "input", function () {
2     setText (▼ "label2", "o número escolhido foi " + getText (▼ "text_input1"));
3 }
4 onEvent (▼ "buttonVerificar", ▼ "click", function () {
5     if (getText (▼ "text_input1") % 2 == 0) {
6         setText (▼ "labelresposta", "O número escolhido é Par");
7     } else {
8         setText (▼ "labelresposta", "O número escolhido é Ímpar");
9     }
10 }

```

Fonte: Elaborado pelo autor (2023).

Neste exemplo observamos que foi utilizado o `onEvent id "click"` e `onEvent id "input"`. O comando `onEvent id type callback` executa o código de retorno de chamada quando ocorre um tipo de evento específico para o id do elemento de interface do usuário especificada. Os aplicativos interativos precisam de elementos de UI `button`, `textInput`, `textLabel`, `dropdown`, `checkbox`, `radioButton`, `image` e manipuladores de eventos para esses elementos de UI e cada tipo de interação do usuário necessária. O elemento UI, com id exclusivo, deve existir antes que a função `onEvent` possa ser usada.



Fonte: Elaborado pelo autor (2023).

onChange

O evento `onChange` é usado para que determinada ação seja realizada após alguma mudança no documento. Por exemplo, vamos fazer um código, onde ao clicarmos fora do “`input text`” acontecer, nesse caso o texto escrito no `input` irá ficar todo em caixa alta.

onMouseOver e onMouseOut

Esses dois eventos são responsáveis pelas ações que o usuário realiza com o mouse, quando o mouse está em cima do elemento e quando está fora do elemento, respectivamente

onMouseDown, onMouseUp e onMouseClick

Com a combinação desses eventos podemos trabalhar com o clique do mouse, e quando soltar o clique. Existe uma lista muito ampla de eventos em javascript, nessa lista estão subdivididos em categorias, as mais comuns são:

- ❖ Recursos
- ❖ Rede
- ❖ Foco
- ❖ WebSocket
- ❖ Histórico de Sessão
- ❖ Animações CSS
- ❖ Transições CSS
- ❖ Formulários Impressão
- ❖ Composição de TextoTela
- ❖ Área de Transferência
- ❖ TecladoMouse
- ❖ Arrastar e Soltar
- ❖ Mídia Progressão Armazenamento

A delegação de eventos é uma técnica em que você registra um único manipulador de evento em um elemento pai, em vez de em cada elemento filho. Isso é útil para lidar com muitos elementos sem precisar de muitos manipuladores de eventos individuais.

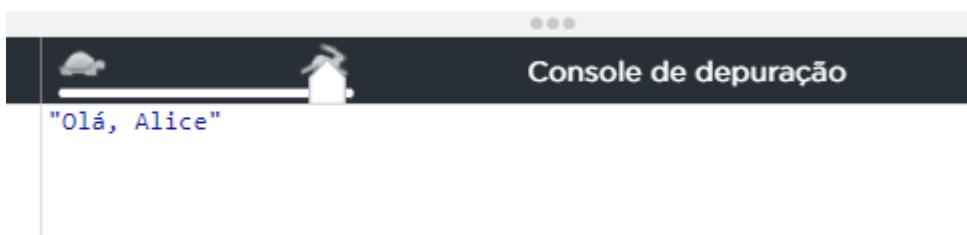
Closures

Closures em JavaScript são um conceito avançado e poderoso que envolve a combinação de funções e escopo. Uma closure ocorre quando uma função é definida dentro de outra função e mantém acesso ao escopo da função externa, mesmo após a função externa ter sido executada e seu escopo ter sido encerrado. Isso permite que a função interna continue a acessar variáveis, parâmetros e outros elementos da função externa.

```
1 function saudacao(nome) { ↵ ↶
2     var mensagem = "Olá, " + nome;
3
4     function mostrarMensagem() { ↶
5         console.log(mensagem);
6     }
7
8     return mostrarMensagem;
9 }
10
11 var cumprimento = saudacao("Alice") ↵ ↶;
12 cumprimento(); ↶ // Irá exibir "Olá, Alice"
13
```

Fonte: Elaborado pelo autor (2023).

Saída:



Fonte: Elaborado pelo autor (2023).

A função inicio cria uma variável local chamada “saudacao”, logo depois define uma função chamada mostrarMensagem(). A função mostrarMensagem() é uma função aninhada(um closure) - ela é definida dentro da função saudacao(), e está disponível apenas dentro do corpo desta função. Ao contrário da função saudacao(), a mostrarMensagem() não tem variáveis locais próprias, e ao invés disso ela reusa a variável browser declarada na função saudacao(), também chamada de função pai. Este é um exemplo de escopo léxico em javascript. O escopo de uma variável é definido por sua localização dentro do código fonte, e funções aninhadas têm acesso às variáveis declaradas em seu escopo externo.

Características das Closures

Função Interna: A função que é definida dentro de outra função é chamada de função interna ou função fechada (closure function).

Função Externa: A função que envolve a função interna é chamada de função externa ou função envolvente (outer function).

Escopo: A função interna tem acesso ao escopo da função externa, incluindo as variáveis, parâmetros e outros elementos definidos nesse escopo.

Retenção de Escopo: Uma closure retém o escopo da função externa mesmo após a função externa ter terminado a execução.

Alteração de escopo

Em algumas situações precisamos alterar o escopo de uma função, ampliando sua utilização, vamos entender no exemplo mostrado abaixo algumas situações em que isso pode acontecer. A alteração do escopo, em JavaScript, se refere à capacidade de uma função interna (função aninhada) acessar e manipular as variáveis definidas no escopo de sua função externa (função envolvente), mesmo após a função externa ter terminado de ser executada.

Contexto do Problema:

Normalmente, quando uma função é chamada e sua execução é concluída, todas as variáveis locais criadas dentro dessa função são destruídas e o espaço de memória associado a essas variáveis é liberado. No entanto, quando uma função interna é definida dentro de outra função, a função interna mantém um vínculo com as variáveis da função externa, mesmo após a execução da função externa ter terminado.^{3e}

```
1 function externa() {  
2     var x = 10;  
3  
4     function interna() {  
5         console.log(x); // A função interna acessa a variável 'x' da função externa  
6     }  
7  
8     return interna;  
9 }  
10  
11 var minhaFuncao = externa();  
12  
13 minhaFuncao(); // Irá imprimir 10  
14
```

Fonte: Elaborado pelo autor (2023).

Neste exemplo, a função interna é retornada como resultado da função externa. Embora a função externa tenha terminado sua execução e a variável x esteja fora de escopo, a função interna ainda pode acessar a variável x porque ela mantém uma referência a esse escopo.

Utilização Prática:

A capacidade de alterar o escopo é amplamente usada para criar closures, que são úteis para

encapsular dados, criar funções personalizadas e gerenciar eventos. Ao usar closures, você pode criar funções que operam com dados específicos sem expor esses dados a todo o escopo global. Isso ajuda a evitar conflitos de nomes e a manter o código organizado.

Importante Notar:

Enquanto closures e alteração de escopo são recursos poderosos e úteis, eles também podem levar a problemas se não forem usados com cuidado. Por exemplo, closures podem levar a vazamentos de memória se as referências não forem gerenciadas adequadamente. Além disso, a alteração de escopo pode complicar a depuração do código se não for compreendida corretamente.

Em resumo, a alteração de escopo é uma característica crucial das closures em JavaScript, permitindo que funções internas acessem e manipulem variáveis da função externa, mesmo após a conclusão da execução da função externa. Isso é amplamente usado para criar funções encapsuladas, personalizadas e eventos de gerenciamento, mas requer uma compreensão sólida para ser usado com sucesso.



RESUMO

Neste tema nós aprendemos que as funções em JavaScript desempenham um papel fundamental na criação de código modular, reutilizável e eficiente. Elas são blocos de código que podem ser definidos e executados quando necessário, permitindo que você agrupe uma série de instruções em uma única unidade lógica. Elas podem receber parâmetros como entrada e retornar um valor como saída. As funções podem ser definidas usando a palavra-chave `function`.

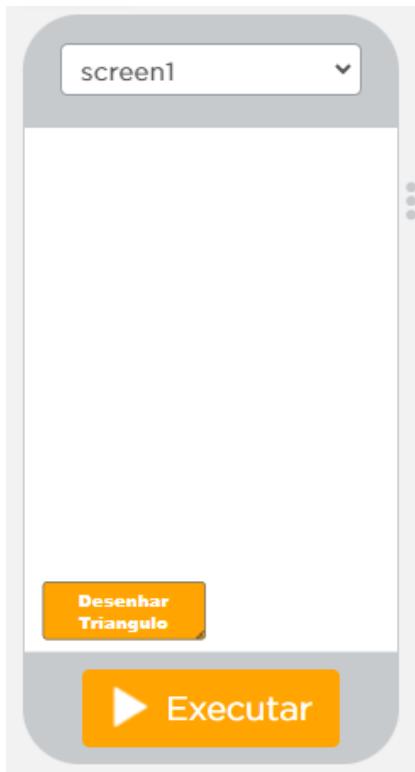
Também entende-se que as funções em JavaScript desempenham um papel central na criação de aplicativos interativos e eficientes. Ao entender como usar funções, você pode escrever código mais organizado, reutilizável e de fácil manutenção.



ATIVIDADE DE FIXAÇÃO

1. O que é uma função
2. Quais são as regras para criar uma função?
3. O que é um evento? Para que são usados?
4. Quais são os tipos de eventos utilizados no exemplo dado?
5. Por que na função construtora usamos a palavra `this` ?
6. Crie um programa com a função de desenhar um triângulo quando clicar em um botão. Você

deverá utilizar a sessão turtle do laboratório de aplicações e o design ficará desse jeito:



Fonte: Elaborado pelo autor (2023).

7. Adicione um botão para desenhar um quadrado, no exercício acima.
8. Crie um programa em que o usuário insere 2 números e exiba qual é o maior deles.
9. Crie um programa com a função construtora que exiba o nome, a idade e o gênero de 4 pessoas;
10. Crie um programa, usando como base o desafio anterior, e faça uma contagem de pessoas cadastradas usando uma closure.

TEMA 08

Manipulando Dados

Habilidades

- Compreender a manipulação de dados e uso de funções e métodos.
- Aprender a coletar, armazenar e exibir dados inseridos pelos usuários, bem como a realização de operações com esses dados.

A manipulação de dados em JavaScript refere-se à capacidade de acessar, modificar e interagir com dados em um programa JavaScript. Isso é fundamental para qualquer linguagem de programação, pois os programas geralmente precisam armazenar, recuperar e processar informações.

Inserção e exclusão de dados envolve a coleta e inserção de informações em uma estrutura de dados, como um banco de dados, uma matriz ou uma lista que permite a remoção, ordenação, filtragem, validação e transformação dessas informações armazenadas. Essas operações são fundamentais para armazenar dados para uso futuro e é útil para manter registros precisos e atualizados.

Além de poder recuperar Dados e permitir a criação de Relatórios de forma comprehensível e significativa pois relatórios são frequentemente usados para tomada de decisões, é imprescindível manter a limpeza de Dados, Backup e Restauração e a Segurança de Dados com a implementação de medidas de segurança para proteger os dados contra acesso não autorizado ou ameaças à sua integridade.

A Integração de Dados é o processo de combinar dados de diferentes fontes para criar uma visão unificada e completa. Isso é comum em sistemas que utilizam várias fontes de dados.

E a mineração de Dados envolve a descoberta de padrões, tendências e informações valiosas nos dados. É usado em análises avançadas e tomada de decisões.

Esses são apenas alguns exemplos dos tipos de manipulação de dados que os desenvolvedores e analistas de dados realizam rotineiramente. A manipulação de dados é uma habilidade essencial em muitas áreas, incluindo desenvolvimento de software, análise de dados, negócios e pesquisa.

Manipulação de datas

É o processamento, transformação e interpretação de informações relacionadas a datas e horários. Isso é crucial em muitos contextos, como desenvolvimento de aplicativos, análise de dados, automação de tarefas e muito mais.

Em javascript a data é um objeto que nos mostra a data atual, em milissegundos percorridos desde 1º de janeiro de 1970. Existem Diferentes formas de trabalhar com datas em javascript, vamos ver os exemplos das principais e mais conhecidas manipulações com datas.

Conceitos-chave sobre a manipulação de dados de datas:

Representação de Datas e Horários

Em programação, datas e horários são representados como objetos que contêm informações específicas, como dia, mês, ano, hora, minuto e segundo. Essas informações são armazenadas em uma estrutura de dados que permite a manipulação.

Formatos de Data e Hora

As datas podem ser representadas em diversos formatos, como "dd/mm/aaaa", "aaaa-mm-dd", "mm/dd/aaaa", entre outros. Além disso, os horários podem ser representados em formatos de 12 horas ou 24 horas, com ou sem fuso horário.

exemplo:

```
1 // Criando um objeto de data
2 var data = new Date();
3
4 // Formatando a data em um formato específico
5 var formato = { year: 'numeric', month: 'long', day: 'numeric' };
6
7 var dataFormatada = data.toLocaleDateString('pt-BR', formato);
8
9 console.log("Data formatada:" + dataFormatada);
10
```

Fonte: Elaborado pelo autor (2023).

Saída

The screenshot shows a browser's developer tools with the 'Console de depuração' (Debug Console) tab selected. The console window displays the output of the JavaScript code. The output is a single line of text: "Data formatada:1 de setembro de 2023". This indicates that the current date in Brazil (pt-BR) is being displayed in a localized string format.

Fonte: Elaborado pelo autor (2023).

Com o código acima temos a data atual vinda do sistema é mostrada em forma de mensagem.

Operações Básicas

A manipulação de datas envolve operações básicas, como adição ou subtração de dias, meses ou anos a uma data, formatação de datas em um formato específico e cálculos de diferenças entre duas datas.

Exemplo

```

1 // Duas datas para cálculo de diferença
2 var data1 = new Date('2023-12-31');
3 var data2 = new Date('2022-05-15');
4
5 // Calculando a diferença em milissegundos
6 var diferença = data1 - data2;
7
8 // Convertendo para dias
9 var dias = Math.floor(diferença / (1000 * 60 * 60 * 24));
10
11 console.log("Diferença em dias:" + dias);
12

```

Fonte: Elaborado pelo autor (2023).

Saída

The screenshot shows a browser's developer tools with the title "Console de depuração". It contains the following text:
"Diferença em dias:595"

Fonte: Elaborado pelo autor (2023).

Tratamento de Fusos Horários

Em um contexto global, é importante considerar os fusos horários ao manipular datas. Isso envolve converter datas e horários de um fuso horário para outro, o que é particularmente relevante em aplicativos que atendem a usuários em diferentes regiões do mundo.

Fonte: Elaborado pelo autor (2023).

Saída:

```

1 // Data atual
2 var dataAtual = new Date();
3
4 // Adicionando 5 dias à data atual
5 dataAtual.setDate(dataAtual.getDate() + 5);
6
7 console.log("Data após adicionar 5 dias:" + dataAtual);
8
9 // Subtraindo 2 dias da data atual
10 dataAtual.setDate(dataAtual.getDate() - 2);
11
12 console.log("Data após subtrair 2 dias:" + dataAtual);
13

```

```
Console de depuração
Limpar
>Data após adicionar 5 dias:Wed Sep 06 2023 08:26:36 GMT-0300 (Horário Padrão de Brasília)
>Data após subtrair 2 dias:Mon Sep 04 2023 08:26:36 GMT-0300 (Horário Padrão de Brasília)
```

Fonte: Elaborado pelo autor (2023).

Análise de Séries Temporais

Em análise de dados, a manipulação de datas é essencial para a análise de séries temporais, que envolve o estudo de dados que variam ao longo do tempo. Isso é comum em campos como finanças, meteorologia, saúde e muito mais.

Validação e Formatação

A manipulação de datas também inclui a validação de datas para garantir que estejam em um formato correto e a formatação de datas para exibição em um formato legível para os usuários.

Tarefas Recorrentes

Muitas aplicações lidam com tarefas que ocorrem em datas e horários específicos, como agendamento de eventos, lembretes e notificações. A manipulação de datas é essencial para gerenciar essas tarefas.

Lembre-se de que, ao trabalhar com datas em JavaScript, é importante considerar as limitações e peculiaridades da linguagem, como a contagem de meses a partir de 0 (janeiro é 0, fevereiro é 1, etc.) e as diferenças de fusos horários, se aplicáveis ao seu caso de uso. O uso de bibliotecas como o Moment.js pode facilitar muitas tarefas de manipulação de datas.

CallBack

Uma callback (função de retorno de chamada), é uma função passada para outra função como um argumento, que será invocado dentro da função externa para completar alguma rotina ou ação.

Bibliotecas

Para facilitar a manipulação de datas em programação, muitas linguagens de programação oferecem bibliotecas ou módulos dedicados que fornecem funções predefinidas para realizar operações comuns de manipulação de dados. Em JavaScript, por exemplo, a biblioteca Date é usada para trabalhar com datas e horários.

No code.org podemos acessar a biblioteca de dados através do botão localizado no canto superior esquerdo da tela

The screenshot shows the Code.org Data Library interface on the left and a Data Navigator interface on the right. The Data Library lists various categories with their respective table counts. The Data Navigator allows users to create their own tables with columns and rows.

Fonte: Elaborado pelo autor (2023).

Você pode utilizar as bibliotecas que já possuem no arquivo ou criar as suas próprias usando o excel.

Como adicionar uma planilha

Passo 1. Formate a planilha Em um editor de planilha de sua escolha, certifique-se de que os dados estejam formatados com os nomes das colunas na primeira linha e os dados da tabela abaixo dela. Os dados da sua planilha devem ser formatados assim, mas com valores de sua escolha (como nosso exemplo de comida favorita):

coluna 1	coluna 2	coluna N
linha1 dados col1	linha1 col2 dados	linha1 dados col N
linha2 dados col1	linha2 dados col2	linha2 dados col N
linha3 dados col1	linha3 dados col2	linha3 dados col N

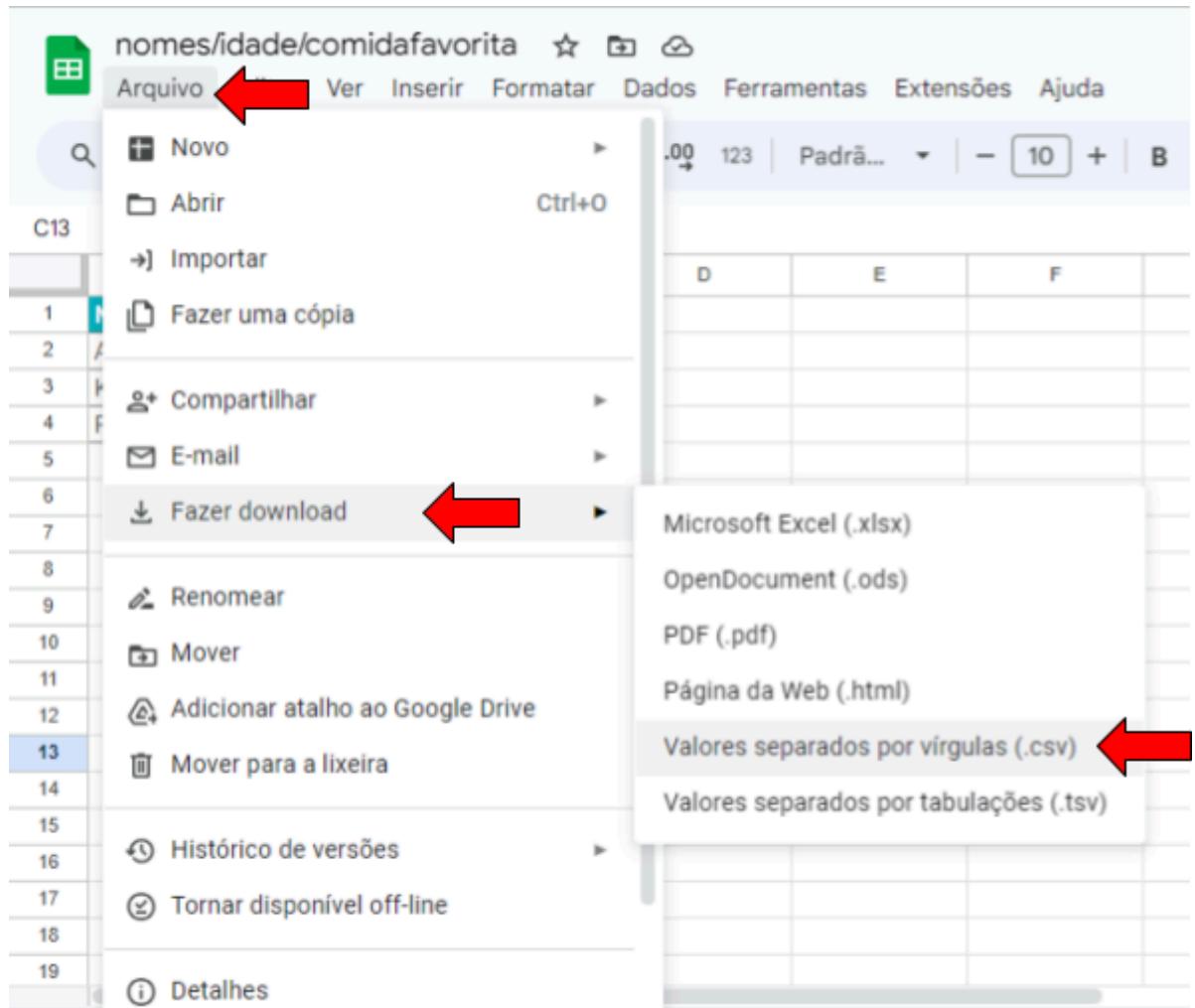
Exemplo:

The screenshot shows a Microsoft Excel spreadsheet titled "nomes/idade/comidafavorita". The table has columns labeled "Nome", "Idade", and "Comida". The data includes three rows: Abby (17, Ravioli), Kamara (15, Sushi), and Rachel (16, Salada). The table is styled with a blue header row.

Fonte: Elaborado pelo autor (2023).

Etapa 2. Salve a planilha como um arquivo csv Um arquivo separado por vírgula, ou arquivo csv, é um formato de arquivo simples que a maioria dos

editores de planilhas deve suportar para salvar seu arquivo. Salve o arquivo como csv e baixe-o para o seu computador.



Fonte: Elaborado pelo autor (2023).

Etapa 3. Importe o arquivo para o armazenamento de dados da tabela do App Lab. Para importar o arquivo para uma tabela no App Lab:

Clique no botão “Visualizar dados” no App Lab enquanto visualiza seu código no “Modo de código”.

Clique no nome de uma tabela existente ou adicione uma nova tabela ao seu aplicativo.

Crie tabelas de dados para armazenar linhas de dados com múltiplas colunas para diferentes campos.

Nomes	Adicionar
-------	------------------

Fonte: Elaborado pelo autor (2023).

Clique no botão Importar e selecione o arquivo csv da Etapa 2.

Navegador de dados

Voltar aos dados

Visualização de depuração

Nomes

Zerar a tabela Importar csv Exportar para csv

id	Nome	Ações
#	column1	Adicionar linha
#	digite o texto	

Fonte: Elaborado pelo autor (2023).

Clique em 'Substituir e importar' para que os dados da planilha substituam os dados atuais da tabela. Nota: Os dados existentes serão substituídos, portanto, certifique-se de que não pretendia manter os dados atuais. Se precisar adicionar ou excluir algum item é só editar na planilha.

Navegador de dados

Voltar aos dados

Visualização de depuração

Nomes

Zerar a tabela Importar csv Exportar para csv

id	Nome	Idade	Comida	Ações
#	digite o texto	digite o texto	digite o texto	Adicionar linha
1	"Abby"	17	"Ravioli"	Editar Excluir
2	"Kamara"	15	"Sushi"	Editar Excluir
3	"Rachel"	16	"Salada"	Editar Excluir

Fonte: Elaborado pelo autor (2023).

Etapa 4. Revise os dados importados Se seus dados forem importados com sucesso, revise a tabela para ter certeza de que tudo está correto. Você pode fazer ajustes rápidos nas linhas clicando no botão "Editar" dessa linha e editando os valores embutidos.

Para criar, ler, atualizar e excluir registros de tabela do seu aplicativo, use as seguintes funções do App Lab Data:

- ❖ Para criar uma linha em uma tabela: `createRecord()`

- ❖ Para ler uma ou várias linhas em uma tabela: `readRecords()`
- ❖ Para atualizar uma linha existente em uma tabela: `updateRecord()`
- ❖ Para excluir uma linha existente de uma tabela: `deleteRecord()`

Vamos observar um exemplo de um aplicativo que recolhe dados e armazena nas planilhas criadas.

```

1 // Collect favorite food data from friends and store it in a table.
2
3
4
5
6

```

Fonte: Elaborado pelo autor (2023).

```

7 onEvent("doneButton", "click", function() {
8     var favFoodData = {};
9     favFoodData.name = getText("nameInput");
10    favFoodData.age = getNumber("ageInput");
11    favFoodData.food = getText("foodInput");
12    createRecord("fav_foods", favFoodData, function(record) {
13        console.log("Record created with id:" + record.id);
14        console.log("Name:" + record.name + " Age:" + record.age + " Food:" + record.food);
15    });
16 });

```

Fonte: Elaborado pelo autor (2023).

Coletando Dados

Existem três campos de entrada de texto na interface: um para nome, um para idade e outro para comida favorita.

- Há também um botão "Done" (Concluído) na interface.
- Evento de Clique:

O código registra um evento de clique no botão "doneButton". Isso significa que quando o botão "Done" for clicado, a função definida no evento será executada.

- Criação de um Objeto de Dados:

Dentro da função de clique, um objeto chamado `favFoodData` é criado. Este objeto será usado para armazenar os dados inseridos pelo usuário.

Os valores dos campos de entrada de texto são obtidos usando as funções `getText()` e `getNumber()`, e esses valores são atribuídos às propriedades do objeto `favFoodData`.

- Envio de Dados para uma Tabela:

A função `createRecord()` é usada para enviar os dados do objeto `favFoodData` para uma tabela chamada "fav_foods". Esta tabela é usada para armazenar os dados inseridos pelos usuários.

Manipulação de Callback:

A função `createRecord()` aceita uma função de retorno de chamada (callback) que será executada após os dados serem enviados com sucesso para a tabela.

Dentro da função de retorno de chamada, as informações do registro recém-criado são exibidas no console do navegador, incluindo o ID do registro, o nome, a idade e a comida favorita.

Esse código cria uma interface onde os usuários podem inserir seu nome, idade e comida favorita. Quando o botão "Done" é clicado, os dados são coletados, armazenados em um objeto e enviados para uma tabela chamada "fav_foods". Em seguida, as informações do registro recém-criado são exibidas no console. Esse código é um exemplo de como coletar e armazenar dados de usuário em uma aplicação.

Design:



Fonte: Elaborado pelo autor (2023).

Saída:

```
Console de depuração
"Record created with id:6"
"Name:Julia Age:15 Food:Goiaba"
```

Fonte: Elaborado pelo autor (2023).

Tabela:

Navegador de dados

TABELA DE DADOS **CHAVE/PARES DE VALORES**

Crie tabelas de dados para armazenar linhas de dados com múltiplas colunas.

<input type="text" value="Nome da tabela"/>	Adicionar				
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #007bff; color: white; text-align: left; padding: 5px;">Nome da tabela</th> <th style="background-color: #007bff; color: white; text-align: left; padding: 5px;">Ações</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">fav_foods</td> <td style="padding: 5px; text-align: center;"> Excluir </td> </tr> </tbody> </table>		Nome da tabela	Ações	fav_foods	Excluir
Nome da tabela	Ações				
fav_foods	Excluir				

Fonte: Elaborado pelo autor (2023).

Navegador de dados

← Voltar aos dados **Visualização de depuração**

fav_foods

[Visualizar dados](#)
[Zerar a tabela](#)
[Importar csv](#)
[Exportar para csv](#)

id	name	age	food	Ações
#	<input type="text" value="digite o texto"/>	<input type="text" value="digite o texto"/>	<input type="text" value="digite o texto"/>	Adicionar linha
7	"Julia "	15	"Goiaba"	Editar Excluir

Fonte: Elaborado pelo autor (2023).



RESUMO

Neste tema, exploramos a manipulação de dados em JavaScript e como coletar informações de usuário em uma aplicação interativa. Aprendemos como criar campos de entrada de texto em uma interface de usuário para que os usuários possam inserir informações, como nome, idade e comida favorita.

Usamos a função `createRecord()` (ou método semelhante) para enviar os dados coletados para uma tabela ou armazenamento de dados. Implementamos uma função de retorno de chamada

para lidar com a resposta após o envio bem-sucedido dos dados. Isso inclui a exibição das informações do registro recém-criado. Vimos como criar uma aplicação interativa que permite aos usuários inserir dados, armazená-los e exibi-los em resposta a eventos do usuário.



ATIVIDADE DE FIXAÇÃO

1. O que é manipulação de datas ?
2. A manipulação de datas envolve cálculos matemáticos, exemplifique com um problema real o que é possível realizar com a manipulação de datas.
3. Quais maneiras possíveis de se trabalhar com datas ?
4. Crie um programa que peça ao usuário sua data de nascimento e, em seguida, calcule e exiba sua idade.
5. O que é mineração de dados?
6. O que é manipulação de dados e quem é o profissional responsável por esse setor ?
7. Qual a diferença entre o comando `createRecord()` e `readRecords()` ?
8. A manipulação de dados exige maior conhecimento das estruturas de js, acesse o qr Code e explique com suas palavras o aplicativo criado.
9. Crie um aplicativo de lista de contatos onde o usuário pode adicionar novos contatos, excluir contatos existentes e pesquisar por contatos.
10. Desenvolva um aplicativo de lista de tarefas que permita ao usuário adicionar, marcar como concluídas e excluir tarefas.

TEMA 09

Funções Assíncronas

Habilidades

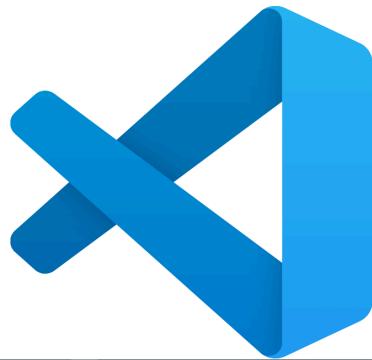
- Utilizar ambientes de desenvolvimento de software para aplicativos móveis.
- Aplicar técnicas corretivas em aplicativos e sistemas
- Conhecer a estrutura das linguagens de programação de alto nível.

Uma função assíncrona, também conhecida como `async function` ou função assíncrona em JavaScript, representa um tipo especial de construção de código que facilita a execução de operações assíncronas de maneira mais conveniente e legível. Essas operações assíncronas geralmente incluem tarefas como fazer solicitações de rede, ler arquivos, aguardar temporizadores e outras atividades que podem levar algum tempo para serem concluídas.



Disponível em: <FREEPIK-<https://tinyurl.com/ms9tdnkj>>. Acesso em 08 out. 2023.

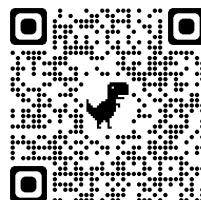
Essencialmente, as funções assíncronas permitem que o código JavaScript execute várias tarefas simultaneamente, sem bloquear a execução do programa. Isso é especialmente valioso em situações em que é necessário lidar com atrasos causados por operações que não são instantâneas, como o carregamento de dados de um servidor remoto.



Disponível em: <<https://tinyurl.com/bddyhbxa>>. Acesso em 06 out. 2023.
Acesse o sistema online pelo link <https://vscode.dev/> ou pelo qrCode.

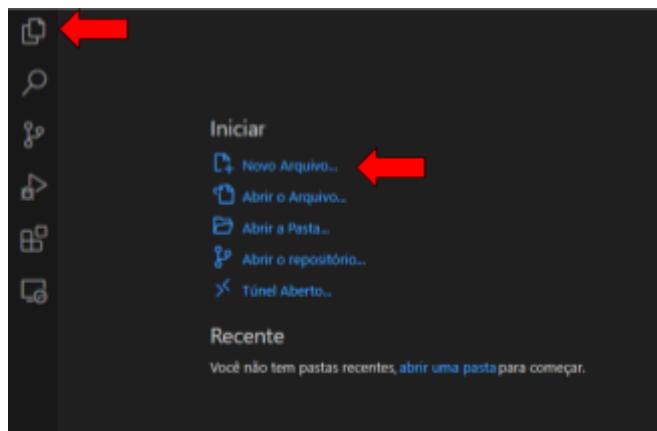


Para instalação do VsCode acesse o QrCode:



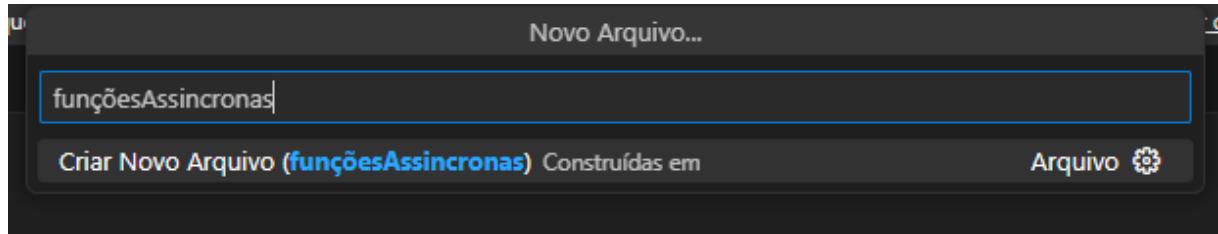
Para aprofundar nosso entendimento sobre o uso de funções assíncronas, é necessário criar um ambiente de desenvolvimento mais profissional. É por isso que optamos por utilizar o Visual Studio Code (VSCode), uma ferramenta altamente reconhecida e eficiente para escrever, depurar e testar código JavaScript e outras linguagens de programação. O VSCode oferece uma ampla gama de recursos e extensões que tornam o desenvolvimento de aplicações JavaScript, especialmente aquelas que se baseiam em funções assíncronas, mais eficaz e produtivo. Portanto, ao explorar as funções assíncronas neste ambiente, estaremos preparados para trabalhar de maneira mais competente com tarefas assíncronas em nossos projetos JavaScript.

Clique no símbolo da esquerda e em seguida novo arquivo.

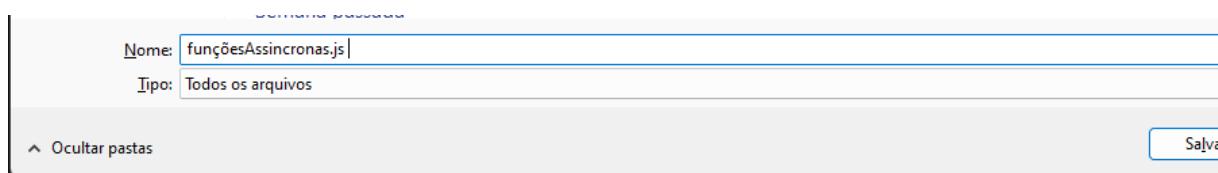


Fonte: Elaborado pelo autor (2023).

salve o arquivo como funçõesAssincronas.js



Fonte: Elaborado pelo autor (2023).



Fonte: Elaborado pelo autor (2023).

Para entender o que é uma função assíncrona (*async function*), precisamos entender como são constituídas. As funções assíncronas são baseadas em *promises*, ou seja, recebendo o retorno tanto se forem executadas ou não. Dessa forma podemos dizer que as funções assíncronas são uma espécie de *promise*, pois condicionam a sua execução na espera de alguns retornos, usando a expressão *await*.

Toda função assíncrona faz o uso da expressão *async* e *await*, vamos entender sua sintaxe:

```
1  function primeiraFuncao() {  
2    console.log("Esperou isso")  
3    }async function segundaFuncao() {  
4      console.log("Iniciou")  
5      await primeiraFuncao()  
6      console.log("Agora executou isso!")  
7    }  
8    segundaFuncao()  
  
CONSOLE ✘  
  
Iniciou  
Esperou isso  
Agora executou isso!
```

Fonte: Elaborado pelo autor (2023).

Note que a função *primeiraFuncao()*, não precisa ser assíncrona para usarmos ela dentro de uma assíncrona, mas estamos esperando o seu resultado dentro da *segundaFuncao()*. Ao executá-la, a última linha só irá rodar quando a resposta da *primeiraFunção()* for exibida. Um erro muito comum, principalmente com os iniciantes de programação, é utilizar uma função assíncrona, é não definir a função com “*async*” e utilizar o “*await*” na função. Neste caso será exibida uma mensagem de erro:

Uncaught SyntaxError: await is only valid in async functions and the top level bodies of modules

Ao ver essa mensagem, lembre-se de declarar a função com *async*. Vamos ver outro exemplo:

```
1  function getUser(id) {
2    return fetch(`https://reqres.in/api/users?id=${id}`)
3      .then(data => data.json())
4      .catch(err => console.log(err))
5  }
6  async function showUserName(id) {
7    const user = await getUser(id)
8    console.log(`O nome do usuário é ${user.data.first_name}`)
9  }
10 showUserName(1) // O nome do usuário é George
```

CONSOLE ×

O nome do usuário é George

Fonte: Elaborado pelo autor (2023).

Neste exemplo foi feita uma requisição em uma API, e aguardou o retorno, para assim depois continuar a execução da função. Temos uma requisição para uma API utilizando o *Fetch API*. Como visto no capítulo anterior, sabemos que uma requisição não é instantânea, ou seja, de alguma forma temos que aguardar os dados retornarem do servidor da API.

Promises

Uma *promise* (promessa) é uma classe que nos possibilita a construção de funções de processamento assíncrono (*async, await*), representando um valor que poderá estar disponível futuramente. *Promises* são representadas em um *proxy*, onde o valor não é necessariamente conhecido quando a *promise* é criada, ou seja, permitindo a associação de métodos de tratamento de eventos da ação assíncrona, num caso eventual de sucesso ou falha. Toda *promise* está em alguns destes três estados abaixo:

- **Pendente** - Estado inicial, ou seja, não foi realizada nem rejeitada.
- **Realizada** - Sucesso na operação.
- **Rejeitado** - Falha na operação.

A *promise* pendente se torna realizada com um valor ou rejeitada por um motivo (erro), e quando ocorre alguns destes estados, o método “*then*” do *promise* é acionado, ele chama o método de tratamento associado ao estado.

Sintaxe:

`new Promise ((resolve: Function, reject: Function) => void)`

Resolve - Função para retornar o resultado da promise.

Reject - Função para retornar o erro da promise.

Propriedade:

Constructor - Função construtora que recebe um *callback* criando uma função assíncrona.

Métodos:

Then - Permite definir o bloco executado mediante o cumprimento de uma *promise*. **Catch** -

Permite definir o bloco executado mediante a rejeição de uma *promise*.

Funções:

Resolve - Cria uma *promise* resolvida.

Reject - Cria uma *promise* rejeitada com o resultado igual ao argumento recebido.

All - Faz a união de várias *promises* em um *array* e retorna o valor quando todas tiverem sido resolvidas.

Vamos ver na prática o uso de *promises*:

```
1 const timeout = (duration) => {
2   return new Promise((resolve, reject) => { setTimeout(resolve, duration)
3   })
4 }
5 timeout(5000)
6 .then(function() { // executa o bloco após 5 segundos
7   console.log('Passou 5 segundos')
8 })
```

CONSOLE ×

Passou 5 segundos

Fonte: Elaborado pelo autor (2023).

Podemos ver o código acima, onde a mensagem é mostrada após 5 segundos da execução da função, com o disparo da função *timeout*.

```
1 function getUser(userId) {
2   const userData = fetch(`https://api.com/api/user/${userId}`)
3   .then(response => response.json())
4   .then(data => console.log(data.name))
5 }
6 getUser(1); // "Nome Sobrenome"
```

Fonte: Elaborado pelo autor (2023).

Nesse exemplo acima temos a função `getUser()` recebendo um *id* de usuário como parâmetro, para que seja passado para o *endpoint* REST. O método `fetch()` recebendo como parâmetro o *endpoint* e retornando uma *promise*. Note que usamos o método `then()`, que recebe uma função *callback* e retorna um “*promise object*”. Deste modo devemos entender que uma *promise* não é um retorno dos dados, é a promessa do retorno destes dados.

Temporizadores

Os temporizadores são funções em javascript (*built-in timer functions* - funções embutidas) que são usadas para o retorno após um determinado tempo. Podemos usar essas funções para que um *script* seja executado em intervalos de tempo, ou até mesmo aguarde um determinado tempo para dar algum retorno ao usuário. Em javascript temos duas funções onde podemos definir intervalos de tempo, são elas:

`setTimeout()` - Essa função executa um específico bloco, uma vez depois de determinado tempo. Ela tem os seguintes parâmetros:

- Zero ou mais valores que representam quaisquer parâmetros que você quiser passar para a função quando ela for executada.
- Uma função a ser executada, ou uma referência de uma função definida em outro lugar.
- Um número representando o intervalo de tempo em milissegundos, ou seja, 1000 milissegundos equivalem a 1 segundo, para esperar antes de executar o código.

No exemplo abaixo vamos entender como funciona a função `setTimeout`:

```
1 let saudacao = setTimeout(function() {  
2   console.log("Seja bem-vindo!");  
3 }, 2000)  
  
CONSOLE ×  
  
Seja bem-vindo!
```

Fonte: Elaborado pelo autor (2023).

Com esse código, é esperado 2 segundos para aparecer uma janela de alerta com a mensagem “Seja bem-vindo”.

`setInterval()` - Essa função executa um específico bloco, com um intervalo fixo entre cada chamada. Essa função funciona perfeitamente quando precisamos executar algum código após um período. Podemos passar parâmetros para uma função `setTimeout()`:

```
1 // Função para usar o timeout
2 function saudacao(nome){
3   console.log("Bem-vindo" + nome);
4 }
5 // Criando uma variável e passando os parâmetros(function, tempo, nome)
6 let msg = setTimeout(saudacao, 2000, " José de Assis");

CONSOLE ×

Bem-vindo José de Assis
```

Fonte: Elaborado pelo autor (2023).

É possível também cancelar o tempo limite acima, com a seguinte expressão:

clearTimeout(msg);

No exemplo acima temos um *script* onde mostra um relógio com o formato tradicional de horas, minutos e segundos.

Parando o tempo:

```
1 <html>
2 <body>
3 <h2>Relógio com botão de parada</h2>
4 <p id="relogio"></p>
5 <button onclick="clearInterval(intervalo)">Parar relógio</button>
6 <script>
7 let intervalo = setInterval(meuTimer ,1000); function meuTimer() {
8   const d = new Date();
9   document.getElementById("relogio").innerHTML = d.toLocaleTimeString();
10 }
11 </script>
12 </body>
13 </html>

WEBSITE VIEW ×

Relógio com botão de parada
22:38:00
Parar relógio
```

Fonte: Elaborado pelo autor (2023).

Podemos ver o exemplo acima que adicionamos um botão para dar o *stop* no relógio. Podemos manipular os intervalos de várias maneiras, o importante é saber usar essas funções para otimizar nossos *scripts*.

WebStorage

Com o lançamento do HTML5, o armazenamento de dados tornou-se bem mais prático e eficiente. Foram adicionados dois objetos para o controle de tal funcionalidade: **localStorage** e **sessionStorage**. As funcionalidades desses objetos são bem semelhantes, enquanto o **localStorage** armazena dados sem data de expiração, o **sessionStorage** armazena apenas para a seção atual, ou seja, ao fechar o navegador, os dados do **sessionStorage** são apagados automaticamente, já os do **localStorage** são mantidos por tempo indeterminado. Os dados digitados pelo usuário, são armazenados no navegador no formato chave/valor, deste modo, cada item deve possuir um único identificador que diferencia dos demais e através do qual ele é localizado. Temos algumas particularidades do **WebStorage** que são de suma importância, sua propriedade e métodos abaixo mostram um pouco mais sobre esse recurso.

Propriedade

`Storage.length` - Retorna um número inteiro que é representado pelo número de itens de dados armazenados no objeto storage.

Métodos

- ④ `Storage.key()` - Esse método é usado quando passado um número “n”, ele retorna o nome da enésima chave no dado objeto.
- ④ `Storage.getItem()` - Esse método é usado para adicionar a chave passada, ou alterar essa chave, caso ela já exista.
- ④ `Storage.removeItem()` - Esse método remove a chave do armazenamento.
- ④ `Storage.clear()` - Esse método limpa todos os itens do armazenamento.

Vamos entender de modo prático, como funciona essa API **WebStorage**:

Com esse código base, podemos fazer as operações com `storage`. Primeiramente vamos inserir uma chave e um valor nos respectivos campos do documento.

```

1 <html>
2 <head>
3 <script type="text/javascript">
4 //inserir as funções aqui, uma abaixo da outra
5 </script>
6 </head>
7 <body>
8 Chave: <input type="text" id="txtChave"/>
9 Valor: <input type="text" id="txtValor"/>
10 <hr/>
11 <button id="btnAdicionar" onclick="adicionar()">Adicionar item</button>
12 <button id="btnLer" onclick="ler()">Ler valor</button>
13 <button id="btnRemover" onclick="remover()">Remover item</button>
14 <button id="btnLimpar" onclick="limpar()">Remover todos</button>
15 </body>
16 </html>

```

WEBSITE VIEW

Chave: | Valor: |

Adicionar item | Ler valor | Remover item | Remover todos

Fonte: Elaborado pelo autor (2023).

Se esse item foi inserido corretamente, a mensagem “item adicionado”, será mostrada na tela.

Agora temos a função de leitura de um item. O usuário deve inserir a chave que está buscando no primeiro campo e clicar no botão “Ler valor”. Se o item existir, seu valor será exibido, caso não exista uma mensagem será mostrada pelo *alert*.

```

function ler(){
var obj = localStorage.getItem(txtChave.value);
if(obj != null){
alert(obj)
}else{
alert("O item procurado não existe.")
}
    
```

Fonte: Elaborado pelo autor (2023).

Nesse código atribuímos o retorno do método *getItem* a uma variável “obj”. A próxima função faz a remoção de um item específico, a partir da chave digitada no campo:

```
function remover()
{
localStorage.removeItem(txtChave.value);
alert("Item removido.");
}
```

Fonte: Elaborado pelo autor (2023).

No código acima, dispensamos a verificação da existência do item, pois a intenção é removê-lo. E por último, temos a função clear(), para excluir todos os itens do *storage*:

```
function limpar(){
localStorage.clear();
alert("Itens Removidos Com Sucesso!!");
}
```

Fonte: Elaborado pelo autor (2023).

A maioria dos *browsers* em suas versões mais atuais já suporta o *webstorage* do HTML5, mas como desenvolvedores dedicados devemos sempre confirmar para evitar possíveis erros. Com um *script* simples, podemos verificar se o browser suporta essa funcionalidade e, em caso negativo, as operações que fariam uso dela, são abortadas. Veja o *script* abaixo:

```
1 if(typeof(Storage)!=="undefined"){
2 //o browser suporta HTML 5 Web Storage
3 }else{
4 //o browser não suporta HTML 5 Web Storage
5 }
```

Fonte: Elaborado pelo autor (2023).

Alguns navegadores oferecem ferramentas de auxílio ao desenvolvedor, o *Google Chrome*, por exemplo, permite visualizar os dados armazenados do *webstorage*, bem como editá-los e removê-los. Abra uma página da internet usando o *Google Chrome*, e pressione a tecla F12, na parte inferior da janela, será aberta uma aba com algumas opções. Alguns cuidados e limitações quando usamos o *localStorage*:

💡 Você só pode usar *strings* no *localStorage*, e isso não é muito legal, porque limita a gravação de dados mais complexos e se você tentar fazer qualquer conversão se transforma numa gambiarra sem limites.

💡 O uso do *localStorage* é síncrona, ou seja, toda execução só é feita uma depois da outra.

💡 Em qualquer *browser*, o *localStorage* é limitado a guardar apenas 5Mb de dados. Muito mais que os 4Kb dos *cookies*.

💡 Não use o *localStorage* para guardar dados sensíveis.

☒ Não pode ser usado por *web workers*. Isso significa que se você quiser fazer aplicações mais complexas, usando processamento em *background* para melhorar performance, você não poderá usar *localStorage* porque não está disponível.

☒ Os dados que estão gravados não têm camada nenhuma de proteção de acesso, ou seja, todos os dados gravados ali podem ser acessados por qualquer código da sua página.

Novas funcionalidades do ES6

O ECMAScript está em constante evolução, então é muito comum que ao fazer referência à linguagem, chamamos de Javascript, mas ao se referir a uma determinada versão, chamamos de ECMA ou ES + ano. Embora essas especificações da ES6 não sejam recentes, é importante saber o tipo de suporte que utilizamos para programação em javascript. Usamos muitas delas anteriormente, porém vamos olhar com mais cuidado e para que serve esses recursos.

Palavras-Chave

As palavras *let* e *const*, destina-se para uso das variáveis, *let* é usada para variáveis comuns, e a *const* para variáveis de valores constante. Notamos que usamos as variáveis com a palavra “*var*” que são consideradas de escopo global, as variáveis declaradas com *let* e *const* ficam confinadas no escopo do bloco no qual foram declaradas.

Funções *helper* para arrays

Funções para facilitar a manipulação de *arrays*, é um dos recursos de mais relevância com essa versão do ES6. Usamos a função “*forEach*” para executar cada elemento dentro de um *array*, veja o exemplo abaixo:

```
1 var cores = ['vermelho', 'verde', 'violeta']
2 function mostraCor(v){
3   console.log(v);
4 }
5 cores.forEach(mostraCor);

CONSOLE ×

vermelho
verde
violeta
```

Fonte: Elaborado pelo autor, 2023.

Map - Essa função cria um *array* contendo o mesmo número de elementos de um determinado array, ou seja, converte o *array* original em outro, renomeando seus elementos.

```
1 var cores = ['vermelho', 'verde', 'violeta']
2 function maiusculas(v){
3   return v.toUpperCase();
4 }var coresMaiusculas = cores.map(maiusculas)
5 console.log(coresMaiusculas);

CONSOLE ×

(3) [ "VERMELHO", "VERDE", "VIOLETA" ]
```

Fonte: Elaborado pelo autor (2023).

Filter - Essa função cria um *array* contendo subconjunto dos elementos do *array* original. O subconjunto de elementos do novo *array* é determinado por uma função que retorna como verdadeiro (*true*) ou falso (*false*) quando verifica uma determinada condição para cada elemento do *array* original.

```
1 var valores = [2, 45, 67, 54, 12, 5]
2 // função que retorna valores menores que 50
3 function novaFuncao(v) {
4   return v < 50
5 }
6 var valorMenor50 = valores.filter(novaFuncao)
7 console.log(valorMenor50);

CONSOLE ×

(4) [ 2, 45, 12, 5 ]
```

Fonte: Elaborado pelo autor (2023)..

Find - Essa função encontra o primeiro elemento de um *array* que satisfaz determinada condição imposta por uma função que retorna verdadeiro (*true*) ou falso (*false*), para cada elemento do *array* original.

```
1 var galera = [
2   {nome: 'Afonso', idade: 8},
3   {nome: 'Thiago', idade: 31},
4   {nome: 'Diego', idade: 29},
5   {nome: 'Lucia', idade: 15},
6   {nome: 'Monique', idade: 24}
7 ]
8 function jovem(pessoa) {
9   return pessoa.idade > 10 && pessoa.idade < 18
10 }
11 var primeiraPessoa = galera.find(jovem)
12 console.log('Primeiro adolescente encontrado:',primeiraPessoa.nome, 'com', primeiraPessoa.idade, 'anos.')

CONSOLE ×

Primeiro adolescente encontrado: Lucia com 15 anos.
```

Fonte: Elaborado pelo autor (2023).

Every - Essa função faz a varredura em cada elemento de um array, para uma determinada condição imposta por uma função com o retorno de verdadeiro(true) e falso(false). Veja um exemplo no código a seguir:

```
1  var galera = [
2    {nome: 'Afonso', idade: 8},
3    {nome: 'Thiago', idade: 31},
4    {nome: 'Diego', idade: 29},
5    {nome: 'Lucia', idade: 15},
6    {nome: 'Monique', idade: 24}
7  ]
8  function jovem(pessoa) {
9    return pessoa.idade > 10 && pessoa.idade < 18
10 }
11 var todosSaoAdolescentes = galera.every(jovem)
12 console.log('Todo mundo é adolescente.', todosSaoAdolescentes)

CONSOLE ×
Todo mundo é adolescente. false
```

Fonte: Elaborado pelo autor (2023).

Reduce - Essa função permite o uso de dois parâmetros e se destina a interação dos elementos de um array executando uma *callback* primeiro parâmetro é uma *callback* que acumula valores do resultado de uma operação feita a cada iteração, e o segundo parâmetro é o valor inicial do acúmulo.

```
1  var array = [1, 2, 3, 4]
2  function soma(somar, valor) {
3    return somar + valor
4  }
5  function produto(mult, valor) {
6    return mult * valor
7  }
8  var somaDosArrays = array.reduce(soma, 0)
9  var produtoDosArrays = array.reduce(produto, 1)
10 console.log('A soma dos elementos do array', array, 'é ', somaDosArrays)
11 console.log('O produto dos elementos do array', array, 'é', produtoDosArrays)

CONSOLE ×
A soma dos elementos do array
(4) [1, 2, 3, 4]
é 10
O produto dos elementos do array
(4) [1, 2, 3, 4]
é 24
```

Fonte: Elaborado pelo autor (2023).

Arrows Functions (Funções de Seta) - Essas funções se encontram na versão do ES6, uma nova forma de criar funções usando setas () => { }.

```
1 var array = [1, 2, 3, 4]
2 var somaDosArrays = array.reduce((somar, valor) => somar + valor, 0)
3 var produtoDosArrays = array.reduce((produto, valor) => produto * valor, 1)
4 console.log('A soma dos elementos do array', array, 'é ', somaDosArrays)
5 console.log('O produto dos elementos do array', array, 'é', produtoDosArrays)

CONSOLE ×

A soma dos elementos do array
(4) [1, 2, 3, 4]
é 10
O produto dos elementos do array
(4) [1, 2, 3, 4]
é 24
```

Fonte: Elaborado pelo autor (2023).

Objetos literais incrementados - Nessa versão do ES6 ficou muito mais simples atribuir campos às variáveis de mesmo nome, escrever funções e efetuar cálculos com propriedades dinâmicas.

```
1 const cor = 'Azul'
2 const num = {
3   x: 5,
4   y: 10,
5   cor, toString() {
6     return 'X = ' + this.x + ', Y = ' + this.y + ', cor = ' + this.cor
7   },
8   [ 'prop_' + 42 ]: 42
9 }
10 console.log('Os valores ' + num)
11 console.log('A propriedade dinâmica é ' + num.prop_42)

CONSOLE ×

Os valores X = 5, Y = 10, cor = Azul
A propriedade dinâmica é 42
```

Fonte: Elaborado pelo autor (2023).

Promises - As *promises* como já vimos anteriormente, temos o resultado ou erro em potencial. Para receber o resultado de uma *promise*, podemos utilizar uma função *callback* como parâmetro da função “*then*”, e para tratamento de erros usamos como parâmetro a função “*catch*”. Abaixo temos um exemplo simples de entender melhor uma *promise*.

No exemplo mostrado acima a saída foi diferente a cada execução, pois a função faz a chamada de um número aleatório a cada execução.

```
1  function asyncFunc() {
2    return new Promise((resolve, reject) => { setTimeout(() => {
3      const resultado = Math.random();
4      resultado > 0.5 ? resolve(resultado) : reject('Não posso calcular.')
5    }, 1)
6  });
7 }
8 for (let i=0; i<10; i++) { asyncFunc()
9   .then(resultado => console.log('Resultado: ' + resultado))
10  .catch(resultado => console.log('Erro: ' + resultado))
11 }
```

CONSOLE ×

```
Erro: Não posso calcular.
Erro: Não posso calcular.
Erro: Não posso calcular.
Erro: Não posso calcular.
Resultado: 0.968555428838048
Erro: Não posso calcular.
Resultado: 0.6146160942763597
Erro: Não posso calcular.
Erro: Não posso calcular.
Resultado: 0.7751167626763489
```

Fonte: Elaborado pelo autor (2023)..

As funções assíncronas são amplamente utilizadas em desenvolvimento JavaScript para tornar o código mais legível e gerenciável quando se lida com operações assíncronas. Elas são especialmente úteis em aplicações web, onde tarefas como solicitações de API, manipulação de eventos e animações podem ocorrer de forma assíncrona. O uso correto de funções assíncronas ajuda a evitar bloqueios na interface do usuário e melhora a experiência do usuário.



RESUMO

Nesse tema falamos muito sobre *Promises*, que são um conceito importante em programação assíncrona, especialmente em JavaScript, que permitem gerenciar operações assíncronas de forma mais limpa e eficaz. Se uma Promise representa um valor que pode não estar disponível imediatamente, mas que será resolvido (com sucesso) ou rejeitado (com erro) em algum momento no futuro, é possível utilizar as funções de temporizador em JavaScript.

Vimos também como utilizar a tecnologia *WebStorage* para armazenamento de dados de forma eficiente com suas propriedades e métodos e as novidades da última versão do ES6 com exemplos de funções para facilitar a manipulação de *arrays*.



ATIVIDADE DE FIXAÇÃO

1. Explique, com suas palavras, o uso das Promises. E quais os estados de uma *promise*?
2. Para que serve a propriedade *constructor*?

3. Explique, com suas palavras, como funciona o temporizador nesse conceito de promise em JavaScript.
4. O que faz a função `setInterval()`?
5. Pensando em `WebStorage`, explique, com suas palavras, as diferenças entre as funcionalidades da função `localStorage` e `sessionStorage`.
6. Cite pelo menos 3 limitações da utilização do `localStorage`.
7. Explique, com suas palavras, os benefícios da novidade do ES6, as Funções `helper` para `arrays`.
8. Explique pelo menos 3 funções diferentes de `helper` para `arrays`.
9. Que função pode ser usada como parâmetro da função “`then`” em JavaScript?
10. Explique, com suas palavras, o que é e como funciona a função `callback`?
11. Explique, com suas palavras, o uso das Promises. E quais os estados de uma promise?

TEMA 10

Front-End, Back-End e Full Stack

Habilidades

- Identificar as áreas de atuação de um desenvolvedor
- Identificar o melhor software de desenvolvimento de acordo com a necessidade
- Utilizar ambientes de desenvolvimento de software para aplicativos móveis.

É uma ocorrência frequente em nosso quotidiano enquanto programadores ouvirmos expressões como "Front-end," "Back-end" ou "Full Stack," mas qual é o significado subjacente desses termos e como eles podem impactar sua carreira futura? Essas designações se aplicam a indivíduos que trabalham no campo do desenvolvimento de software e são utilizadas para classificar de forma mais precisa o tipo de desenvolvedor em questão. Para obter um entendimento mais profundo, exploraremos o significado de cada um desses termos e discutiremos as responsabilidades inerentes a esses profissionais, complementando com exemplos de linguagens de programação e softwares comumente empregados em suas respectivas funções.

Aprofundando o conhecimento, vamos desvendar o que cada um destes rótulos realmente representa e qual é o papel desempenhado por aqueles que se identificam com essas categorias. Além disso, vamos examinar como esses profissionais interagem e colaboram para criar sistemas de software funcionais e eficientes.

Para começar, vamos explorar o universo do desenvolvimento Front-end, mergulhando nas tarefas e ferramentas específicas que esses especialistas utilizam para criar a interface do usuário e a experiência do usuário. A seguir, mergulharemos nas complexidades do desenvolvimento Back-end, que lida com a lógica e o funcionamento interno dos aplicativos e sistemas. Por fim, abordaremos o conceito de desenvolvedor Full Stack, que possui habilidades tanto no Front-end quanto no Back-end, e discutiremos como essa versatilidade pode abrir portas para oportunidades profissionais fascinantes.

Front-End

O profissional que trabalha com *front-end* é responsável pelo desenvolvimento de toda a parte visual de um site ou aplicativo, para que haja a interação com o usuário. Quando você entra em um site, por exemplo, você pode visualizar o menu com botões, imagens, vídeos, entre outros elementos, posicionados em determinadas áreas da página. Toda essa parte gráfica que o usuário visualiza e interage, é o chamado *front-end*.



Disponível em: <FREEPIK-<https://tinyurl.com/4w3zdd4r>>. Acesso em 06 out. 2023.

Para o desenvolvedor *front-end* é muito importante ter conhecimentos sobre códigos de interface gráfica, como HTML, para estruturar o conteúdo das páginas, e as linguagens de programação, CSS, para estilizar e posicionar os elementos utilizados e Javascript, para programar interações dinâmicas na página, como animações e manipulação de dados. Ter familiaridade com bibliotecas que simplificam o desenvolvimento de interfaces de usuário interativas e reativas, e capacidade de usar ferramentas de desenvolvimento do navegador, para depurar problemas, otimizar o desempenho e testar o código também são diferenciais que ajudam muito o trabalho do desenvolvedor.



Disponível em: <[vector4stock-https://tinyurl.com/yjs9z7nb](https://tinyurl.com/yjs9z7nb)>. Acesso em 06 out. 2023.

Back-End

O profissional que trabalha com *back-end* fica responsável por fazer a conexão entre dados que vem do navegador rumo ao banco de dados e vice-versa. Nesse trajeto é necessário aplicar regras de negócio, validações e garantias para não ter problemas de acesso ao banco de dados. É o profissional que trabalha ao lado do servidor, como MySQL, PostgreSQL, MongoDB, ou outros, para armazenar e gerenciar dados de maneira eficiente, em termos populares, nos bastidores. O desenvolvedor *back-end* precisa conhecer algumas linguagens para desenvolver projetos como um todo, algumas delas são : PHP, Python, Javascript, Java, Ruby, entre outras. Não é uma regra absoluta conhecer todas as linguagens, mas o desenvolvedor que tem o conhecimento de maior número de linguagens, está um passo à frente de outros que não o tem.

Alguns diferenciais que também auxiliam o trabalho com *back-end* são, habilidade em criar e consumir APIs (Interfaces de Programação de Aplicativos) para conectar diferentes sistemas e serviços, e compreensão de práticas de segurança cibernética para proteger os dados e a infraestrutura, bem como conhecimento sobre como dimensionar a aplicação para lidar com o aumento da demanda.

Full Stack



Disponível em: <[pressfoto-https://tinyurl.com/2w9h92xk](https://tinyurl.com/2w9h92xk)>. Acesso em 06 out. 2023.

O profissional *Full Stack* é o desenvolvedor que tem domínio no front-end e no back-end. Esse desenvolvedor pode contribuir em qualquer formato com uma equipe de desenvolvimento de produto digital, conforme necessário, pois consegue entender todas as etapas do processo de criação da aplicação. Uma das vantagens em ser desenvolvedor *Full Stack* é entender como as coisas funcionam por completo, isso abre uma variedade de possibilidades com a participação ativa em uma equipe, mas nem tudo são flores, vale lembrar que as tecnologias evoluem e com elas suas linguagens. Por isso o desenvolvedor *Full Stack* precisa se atualizar nas linguagens e novas tecnologias em ambas as situações (*Front-End, Back-End*).

As habilidades necessárias nesse caso, envolvem familiaridade com frameworks que cobrem tanto o Front-End quanto o Back-End, como MEAN (MongoDB, Express.js, Angular, Node.js) ou MERN (MongoDB, Express.js, React, Node.js) e utilizar ferramentas como Git e GitHub para gerenciar o código-fonte e colaborar com outros desenvolvedores de forma eficaz.

Softwares

O mercado de desenvolvimento de *softwares* é um dos principais fatores por levar automação e mobilidade para as empresas, afinal, os *softwares* estão cada vez mais presentes no cotidiano das pessoas, graças ao aumento exponencial de *softwares Mobile*. Um desenvolvedor por trabalhar com diversas possibilidades no mercado de T.I, é muito importante a pesquisar e fazer testes para escolher um editor de código que facilite o processo de desenvolvimento. Vamos conhecer os principais *softwares* de desenvolvimento Javascript.

Node Js

O Node Js é uma plataforma de aplicação que tem como sua linguagem de desenvolvimento o Javascript. O Node Js por usar uma máquina virtual V8, faz etapas de pré-compilação e otimização antes mesmo do código ser executado.

Disponível em: <<https://www.shutterstock.com/pt/image-vector/node-js-1021480693>>. Acesso em 06 out. 2023.

Faça download do Node JS acessando o QR Code: <https://nodejs.org/pt-br/download>



Vue JS

O Vue JS é um *framework* progressivo usado na construção de *interfaces de usuário*. A biblioteca principal do Vue tem como foco na camada visual exclusivamente, assim facilitando adotar e integrar outras bibliotecas ou até mesmo outros projetos.

Disponível em: <<https://tinyurl.com/48v3vnse>>. Acesso em 06 out. 2023.

Faça download do Vue Js acessando o QR Code:<https://br.vuejs.org/v2/guide/installation.html>



Angular JS

O Angular Js é muito conhecido e usado entre os desenvolvedores Javascript. Seu código fonte é *open source*, ou seja, código aberto, possibilitando a comunidade de programadores à atualização e correção de erros, assim como novas atualizações e iterações que possam desenvolver. A vantagem do Angular Js é a possibilidade de integrar o código em HTML nas aplicações Javascript, tornando o desenvolvimento mais simples e em um único *software*.

Disponível em: <<https://tinyurl.com/5bh2mnkv>>. Acesso em 06 out. 2023.

Faça download do Angular JS acessando o QR Code<https://angularjs.org/>



RESUMO

Esse tema explora os termos "*Front-End*", "*Back-End*" e "*Full Stack*" no desenvolvimento, destacando suas responsabilidades e relevância profissional.

O *Front-End* lida com a interface visual, utilizando HTML, CSS e JavaScript. O *Back-End* gerencia a lógica e a conexão com bancos de dados, usando linguagens como PHP, Python e Java. O *Full Stack* engloba ambos. Também é mencionado neste tema alguns exemplos de *softwares* que podem ser utilizados para facilitar e agilizar o trabalho do desenvolvedor.

ATIVIDADE DE FIXAÇÃO

1. Desenvolva um programa para ajudar o departamento pessoal de uma empresa. O programa

deverá ler o nome do funcionário, salário atual e quantidade de dependentes. Faça o cálculo do novo salário desse funcionário, usando como base os dados abaixo:

1, 2, ou 3 Dependentes aumento de 12% do salário

4, 5, ou 6 Dependentes aumento de 15% do salário Acima de 6 Dependentes aumento de 21%

Como resultado, o programa deverá mostrar uma mensagem com o nome do funcionário e o novo salário.

2. Escreva um script para calcular o preço da passagem de uma determinada empresa de transportes. O script deve ler a quantidade de quilômetros da viagem que foi digitado pelo usuário.

A passagem é calculada de acordo com a distância percorrida, segue a tabela de preço abaixo:

Até 50 KM => R\$ 0,65/km

De 51 até 100 KM => R\$ 0,90/km

De 101 até 200 KM => R\$1,00/km

Acima de 200 KM => R\$1,50/km

Calcule o preço da passagem usando a tabela acima, e mostre na tela.

3. Numa promoção para o Dia dos Pais, uma loja irá dar descontos para todos os clientes, mas especialmente para os pais. Desenvolva um programa que leia o nome do cliente, sexo e o valor total da compra. Faça o cálculo do desconto de acordo com a tabela abaixo:

Homens recebem 15% de desconto Mulheres recebem 10% de desconto

Mostre como resultado o valor total da compra com o desconto aplicado.

4. Faça um formulário em HTML com dois campos, um que irá receber comprimento, e o outro a largura de um terreno. Deverá ter um botão “Calcular Área” neste formulário, e quando acionado deve disparar uma função que irá fazer o cálculo e mostrar numa janela a área total do terreno e sua classificação:

Área até 125m² => TERRENO PADRÃO

Área de 125 a 250m² => TERRENO INTERMEDIÁRIO Área de 250 a 350m² => TERRENO LUXO

Área acima de 350m² => TERRENO VIP

5. Desenvolva um cronômetro inteligente, esse cronômetro irá receber o início, final e o passo da contagem. Deverá mostrar numa janela a contagem.

6. Faça um programa que leia 10 números inteiros e mostre quantos deles são pares e quantos são ímpares.

7. Crie um programa que faça o sorteio de 20 números inteiros aleatórios entre 0 e 20 e mostre

numa janela:

Quais números foram sorteados

Quantos números estão acima de 10

Quantos números são divisíveis por 5

8. Desenvolva um script que leia a idade e o sexo de várias pessoas, perguntando se o usuário deseja continuar para cada pessoa. Mostre os seguintes resultados:

Qual maior idade Qual menor idade

Qual a mulher mais jovem Qual o homem mais velho

Qual a média da idade entre as mulheres

9. Crie um programa que leia o nome e idade de 10 pessoas e armazene-os num array. Mostre como resultado:

Qual o nome da pessoa mais velha

Qual o nome da pessoa mais jovem

Qual a média de idade

10. Usando como base o exercício 5, acrescente ao script as seguintes funções: Iniciar, parar e resetar (zerar cronômetro).

sua classificação:

Área até 125m² => TERRENO PADRÃO

Área de 125 a 250m² => TERRENO INTERMEDIÁRIO Área de 250 a 350m² => TERRENO LUXO

Área acima de 350m² => TERRENO VIP

11. Desenvolva um cronometro inteligente, esse cronometro irá receber o início, final e o passo da contagem. Deverá mostrar numa janela a contagem.

12. Faça um programa que leia 10 números inteiros e mostre quantos deles são pares e quantos são ímpares.

13. Crie um programa que faça o sorteio de 20 números inteiros aleatórios entre 0 e 20 e mostre numa janela:

Quais números foram sorteados

Quantos números estão acima de 10

Quantos números são divisíveis por 5?

14. Desenvolva um script que leia a idade e o sexo de várias pessoas, perguntando se o usuário deseja continuar para cada pessoa. Mostre os seguintes resultados:

Qual maior idade

Qual menor idade

Qual a mulher mais jovem

Qual o homem mais velho

Qual a média da idade entre as mulheres

- 15.** Crie um programa que leia o nome e idade de 10 pessoas e armazene-os num array. Mostre como resultado:

Qual o nome da pessoa mais velha

Qual o nome da pessoa mais jovem

Qual a média de idade

- 16.** Usando como base o exercício 5, acrescente ao script as seguintes funções:

Iniciar

Parar

Resetar(zerar cronometro)

- 17.** Faça um programa que leia um número inteiro digitado pelo usuário e numa janela mostre seu fatorial. Ex. $4 \times 3 \times 2 \times 1 = 24$

- 18.** Desenvolva um script que simule um caixa eletrônico. Comece perguntando o valor a ser sacado pelo usuário(número inteiro) e o script irá mostrar a quantidade de cédulas que serão entregues. Considere um caixa eletrônico com as notas de 200, 100, 50, 20, 10, 5 e 2.



REFERÊNCIAS

BROWN, E. **Web Development with Node.js, Express, and MongoDB.** O'Reilly Media,2014.

CROCKFORD, D. **JavaScript: The Good Parts.** O'Reilly Media.2008.

DUCKETT, J. **Desenvolvimento Web com HTML, CSS e JavaScript.** Alta Books, 2014.

FLANAGAN, D. **JavaScript: The Definitive Guide.** O'Reilly Media,2011.

HAVERBEKE, M. **Eloquent JavaScript: A Modern Introduction to Programming.** No Starch Press, 2018.

MALDONADO, José Carlos, et al. **Padrões e frameworks de software.** Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil: 28,2002.

MARTIN, R. C. **Clean Code: A Handbook of Agile Software Craftsmanship.** Prentice Hall,2008.

NIXON, R. **Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5.** O'Reilly Media, 2014.

ROBBINS, J. N. **Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics.** O'Reilly Media,2018.

VEROU, L. **CSS Secrets: Better Solutions to Everyday Web Design Problems.** O'Reilly Media,2015.

W3Schools. HTML CSS Tutorial. Disponível em: https://www.w3schools.com/html/html_css.asp Acesso em: 05 de Julho de 2023.

W3Schools. SQL Tutorial. Disponível em: <https://www.w3schools.com/sql/default.asp>. Acesso em: 05/07/2023



Viva sua profissão!