<LAB365>

JavaScript: Estruturas de repetição

<LAB365> SENAI

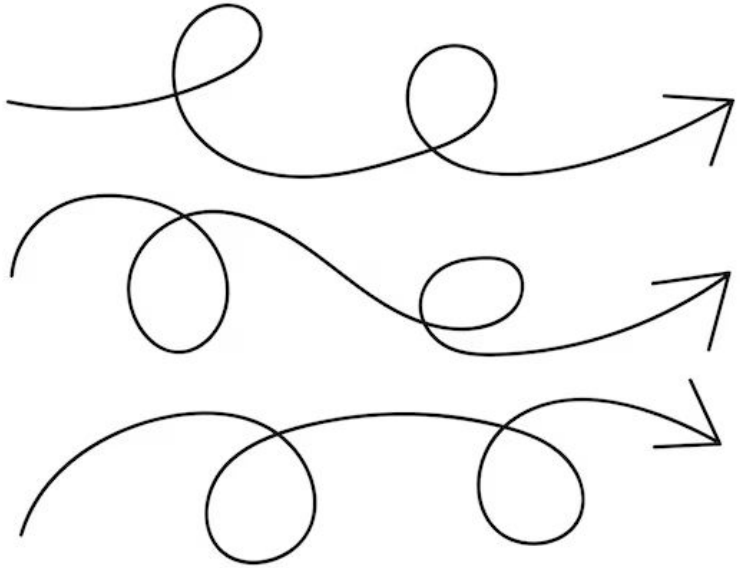
AGENDA | M1S04 - A1

- Estruturas de repetição
  - For
  - While
  - Do while

ESTRUTURAS DE REPETIÇÃO

Em JavaScript, as **estruturas de repetição** (também conhecidas como **loops**) são usadas para **executar um bloco de código várias vezes**. Elas são úteis quando você precisa **repetir** uma **ação** até que uma condição específica seja atendida.

As estruturas que estaremos vendo são a for, while e do while.



FOR

O loop for é uma das estruturas de repetição mais comuns.

Ele é usado quando você sabe **exatamente quantas vezes deseja repetir** um bloco de código.

```
for (inicialização; condição; incremento) {
    // Bloco de código a ser repetido
}
```

FOR

Uma coisa muito interessante na questão da utilização dos loops de repetição é que são feitos para repetir um bloco de código, então podem ser usados para necessidades variadas, um exemplo bem simples do for seria para a exibição de uma tabuada (até o 10).

```
let numero = 2;

console.log(`Tabuada do ${numero}:`);
for (let i = 1; i <= 10; i++) {
    console.log(`${numero} x ${i} = ${numero*i}`);
}
```

WHILE E DO WHILE

O loop while é usado quando você quer repetir um bloco de código enquanto uma **condição for verdadeira**. A condição é **verificada antes** de cada iteração.

```
while (condição) {
    // Bloco de código a ser repetido
}
```

Já loop do... while é semelhante ao while, mas a condição é **verificada após** a execução do bloco de código. Isso garante que o bloco de código seja executado pelo menos uma vez, mesmo que a condição seja falsa desde o início.

```
do {
    // Bloco de código a ser repetido
} while (condição);
```

EXEMPLO DE CÓDIGO

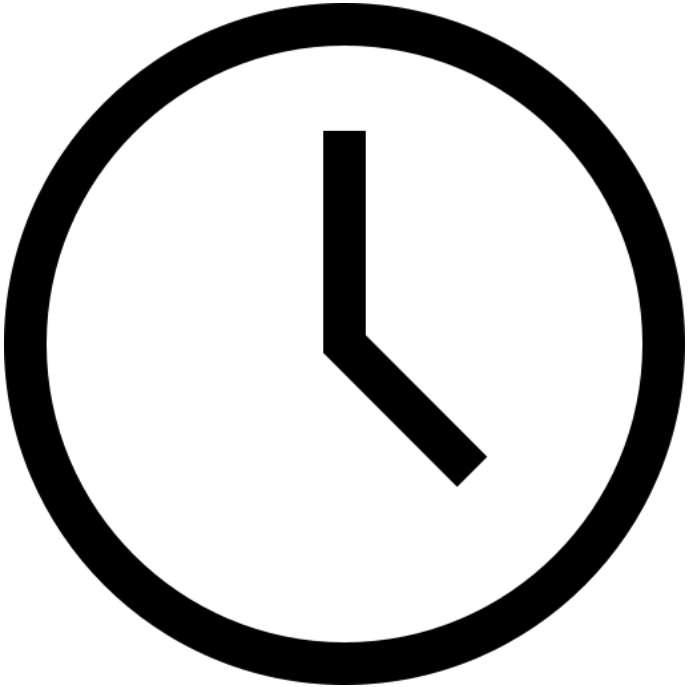
```
let contador = 1;

console.log("Contagem com while:");

while (contador <= 5) {
    console.log(contador);
    contador++; // Incrementa o contador
}

// -- //

contador = 1;

console.log("Contagem com do...while:");

do {
    console.log(contador);
    contador++; // Incrementa o contador
} while (contador <= 5);
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



VAMOS CODAR!

Como uma forma de entendermos mais profundamente a utilização das estruturas de repetição, vamos implementar um estudo de caso.

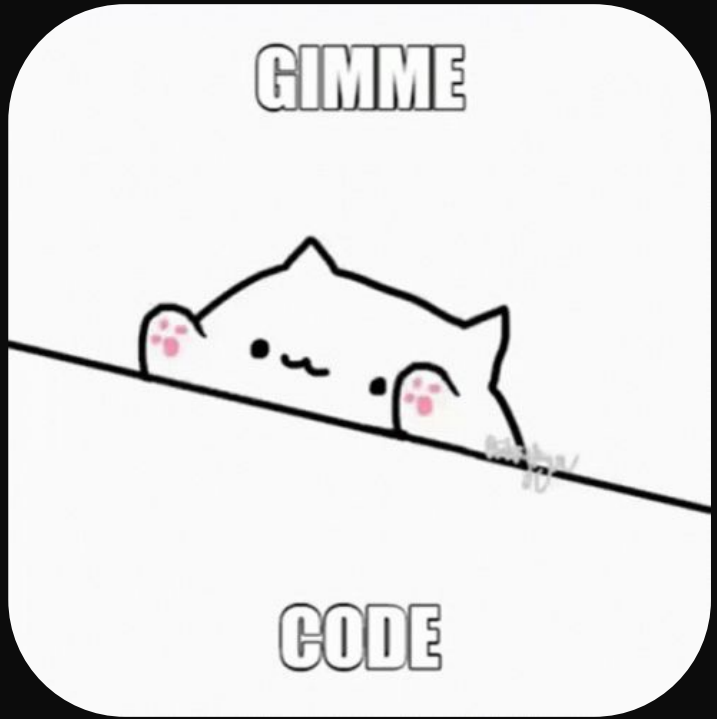
Montaremos um menu para uma aplicação, esse menu terá as seguintes opções:
[1] Qual é nosso curso?
[2] Que módulo estamos?
[3] Quem é o Mentor?
[4] Quem é o Monitor?
[5] Sair

TREINANDO NOSSAS HABILIDADES!

Temos um exemplo bem interessante para demonstrar a utilização de estruturas de repetição, agora ficará para vocês fazerem um menu para o sistemas com as seguintes opções:

[1] Somar
[2] Subtrair
[3] Dividir
[4] Multiplicar
[5] Sair

Sendo que se tentar dividir algo por 0 deverá aparecer “impossível” e que em cada opção você deverá informar dois números para realizar as operações.



<LAB365>

<LAB365>

JavaScript: Arrow Functions e versionamento de código

<LAB365> SENAI

AGENDA | M1S04 - A2

- Arrow Functions
- Versionamento:
  - Git
  - Github
  - Github desktop

ARROW FUNCTIONS

As arrow functions (ou "funções de seta") são uma **sintaxe** introduzida no ECMAScript 6 (ES6) para escrever funções de forma mais concisa em JavaScript.

Elas têm algumas diferenças em relação às funções tradicionais.

```
// Função tradicional
function soma(a, b) {
    return a + b;
}

// Arrow function equivalente
const soma = (a, b) => a + b;
```

CARACTERÍSTICAS DAS ARROW FUNCTIONS

- Sintaxe curta:
  - Se a função tiver apenas uma expressão, o return é implícito.
  - Se houver apenas um parâmetro, os parênteses são opcionais.
- Não podem ser usadas como construtores:
  - Arrow functions não têm a propriedade prototype e não podem ser usadas com new.
- Sem objeto arguments:
  - Arrow functions não têm o objeto arguments como funções tradicionais.

CARACTERÍSTICAS DAS ARROW FUNCTIONS

- Uso comum:
  - São muito usadas em callbacks e funções curtas.

- Quando não usar arrow functions.
  - Em funções que precisam ser usadas como construtores (com new).
  - Quando você precisa do objeto arguments.

EXEMPLO DE CÓDIGO

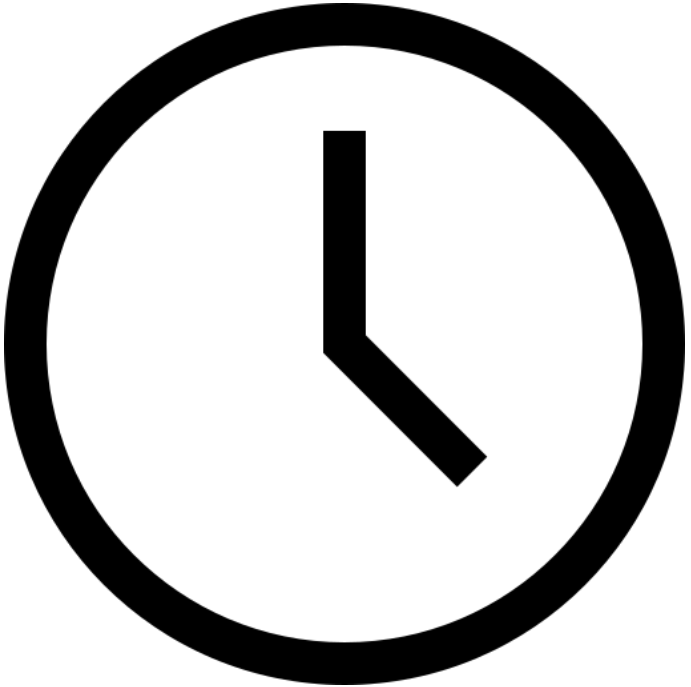
```
// Arrow functions para as operações matemáticas
const soma = (a, b) => a + b;
const subtracao = (a, b) => a - b;
const multiplicacao = (a, b) => a * b;
const divisao = (a, b) => (b !== 0 ? a / b : "Erro: divisão por zero!");

// Função principal
const calculadora = () => {
    let continuar = true;

    while (continuar) {
        // Exibe o menu
        console.log("\nEscolha uma operação:");
        console.log("1 - Soma");
        console.log("2 - Subtração");
        console.log("3 - Multiplicação");
        console.log("4 - Divisão");
        console.log("0 - Sair");

        // Solicita a escolha do usuário
        const escolha = prompt("Digite o número da operação desejada:");

        if (escolha === "0") {
            console.log("Saindo da calculadora...");
            continuar = false;
            break;
        }

        // Solicita os números
        const num1 = parseFloat(prompt("Digite o primeiro número:"));
        const num2 = parseFloat(prompt("Digite o segundo número:"));

        let resultado;

        // Executa a operação escolhida
        switch (escolha) {
            case "1":
                resultado = soma(num1, num2);
                console.log(`Resultado da soma: ${resultado}`);
                break;
            case "2":
                resultado = subtracao(num1, num2);
                console.log(`Resultado da subtração: ${resultado}`);
                break;
            case "3":
                resultado = multiplicacao(num1, num2);
                console.log(`Resultado da multiplicação: ${resultado}`);
                break;
            case "4":
                resultado = divisao(num1, num2);
                console.log(`Resultado da divisão: ${resultado}`);
                break;
            default:
                console.log("Opção inválida! Tente novamente.");
                break;
        }
    }
};

// Inicia a calculadora
calculadora();
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



VERSIONAMENTO DE CÓDIGO

Antes de mais nada, precisamos entender o que é versionamento de código.
Vamos a um exemplo:

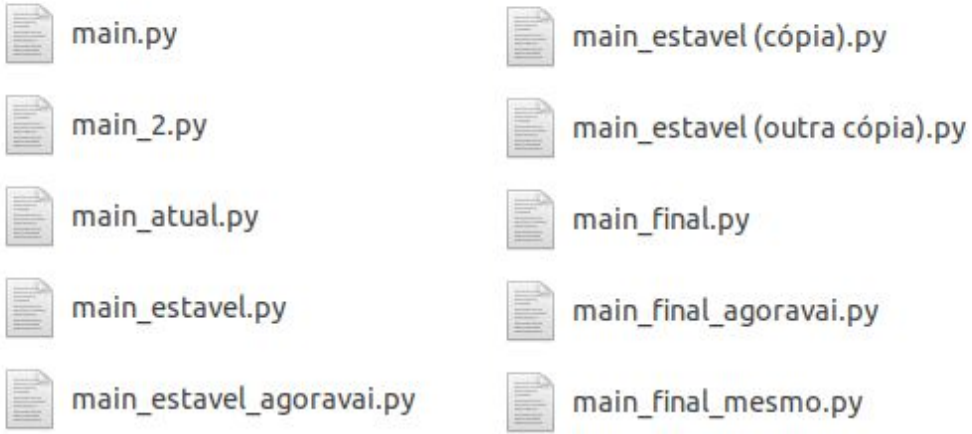
Imagine que você está escrevendo um programa e precisa editar um arquivo, porém, você quer manter uma **cópia da versão anterior** desse arquivo como segurança. Desse modo, você pode retomar seu arquivo da versão anterior, caso algo em sua alteração dê errado, ou por algum motivo você queira voltar para a versão anterior.

Vamos supor que o seu arquivo se chama main.js. Você já construiu bastante coisa, e não gostaria de perder seu progresso caso algo dê errado. Por isso, você cria uma cópia desse arquivo, a chama de main_estavel.js, e continua editando o arquivo main.js.

VERSIONAMENTO DE CÓDIGO

Qual o problema de versionar código criando cópias físicas para cada alteração?

Imagine que você continuou trabalhando nesse mesmo programa por mais alguns dias e foi criando várias cópias desse arquivo. Quando você olha para sua pasta, ela está assim:



VERSIONAMENTO DE CÓDIGO

Alguns dias depois, você volta a mexer nesse mesmo projeto e se depara com alguns problemas:

- Você não sabe mais qual é a versão mais recente do arquivo;
- Você não sabe qual a diferença entre cada versão do arquivo;
- Você não sabe em qual delas está aquela alteração específica que você fez;
- [...]

Agora imagine isso em um sistema de grande porte, com centenas de arquivos e pastas, mantido por uma equipe de vários desenvolvedores.

Para solucionar isso usamos uma ferramenta de versionamento de código!

GIT

De acordo com o site oficial, Git é uma ferramenta de **versionamento**, projetada para suportar de pequenos a grandes projetos com velocidade e eficiência.

Na prática, as pessoas desenvolvedoras têm o Git instalado em seus computadores e o utilizam para versionar o código de um programa em um repositório, integrando seu trabalho com o de seus colegas de equipe.

GIT

É comum em equipes que utilizam Git para versionar código, que esse código seja centralizado em um local que pode ser um servidor próprio, ou **uma plataforma baseada em git**. Vale destacar que a larga maioria prefere utilizar plataformas baseadas em git, que são programas que poupam o esforço de configurar de um servidor git, além de apresentarem diversas outras funcionalidades.

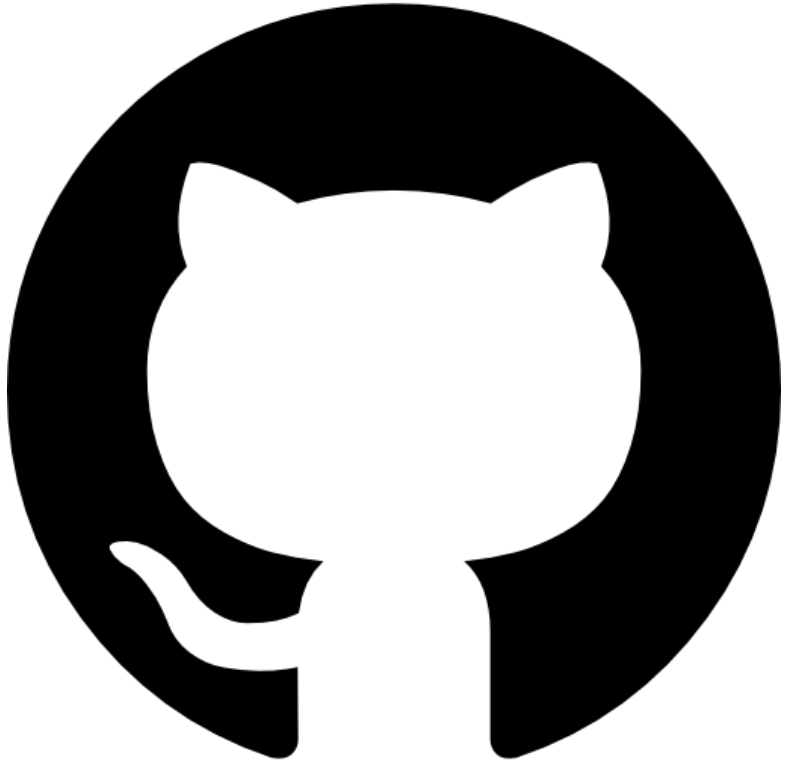
Algumas plataformas baseadas em Git mais populares, utilizadas nas empresas e em demais projetos de software, são:

- Github;
- Gitlab;
- Bitbucket.

GITHUB

É uma plataforma de hospedagem de código-fonte e colaboração baseada em Git, um **sistema de controle de versão** distribuído.

Ele é amplamente utilizado por desenvolvedores para **gerenciar projetos de software**, colaborar com outros desenvolvedores e contribuir para projetos de código aberto.



GITHUB

Principais Recursos:

- Repositórios: Um repositório (ou "repo") é um local onde o código-fonte de um projeto é armazenado. Ele pode ser público (visível para todos) ou privado (acessível apenas para colaboradores autorizados).
- Controle de Versão: O GitHub usa Git para rastrear alterações no código. Isso permite que os desenvolvedores trabalhem em diferentes versões de um projeto simultaneamente e mesclem as alterações de forma eficiente.

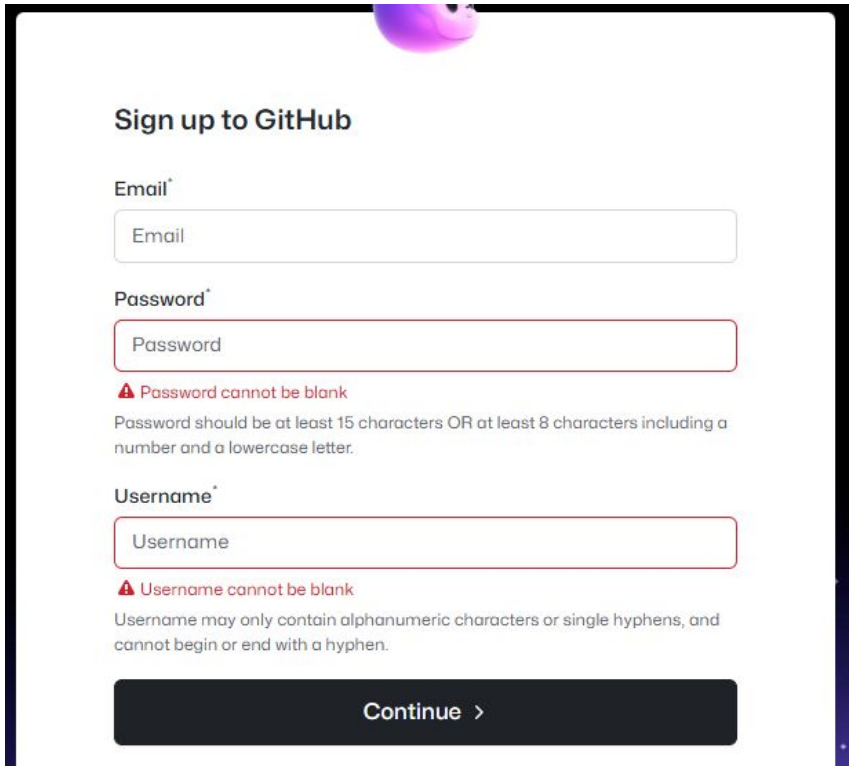
Benefícios:

- Colaboração Global: Facilita a colaboração entre desenvolvedores de todo o mundo.
- Transparência: Projetos de código aberto no GitHub são transparentes, permitindo que qualquer um veja o código e contribua.
- Integração: O GitHub se integra com muitas outras ferramentas de desenvolvimento, como Slack, VSCode, e mais.

GITHUB

Nós inicialmente precisamos criar nossa conta no github!

https://github.com/



GITHUB DESKTOP

O GitHub Desktop é uma **aplicação gráfica** (GUI) desenvolvida pelo GitHub para facilitar o uso do Git, especialmente para quem prefere uma **interface visual em vez de comandos de terminal**.

Ele é projetado para **simplificar o fluxo de trabalho com repositórios Git e GitHub**, tornando-o **acessível** para **iniciantes** e útil para desenvolvedores que desejam uma maneira mais visual de gerenciar seus projetos.

GITHUB DESKTOP

O primeiro passo será fazer o download do github desktop.

https://desktop.github.com/download/

Aqui não temos muito problema, vocês devem baixar o instalador referente ao seu SO, executar o mesmo e seguir a instruções de instalação.

GITHUB DESKTOP : CONFIGURAR

Após o login, o GitHub Desktop pedirá para configurar seu **nome e e-mail**. Esses dados serão usados nos commits.
O nome e e-mail devem ser os mesmos associados à sua conta do GitHub.
Clique em "Finish" (Concluir) para finalizar a configuração.

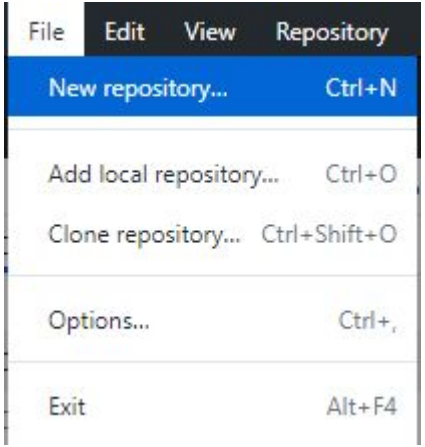
Algo que acho muito pertinente de dar uma ênfase é que estaremos trabalhando com o github desktop, pois sua interface é mais visual e indicada para iniciantes. Você também podem ter contato com muitos desenvolvedores optando por utilizar o github por meio do git, isso é feito por comandos de texto.

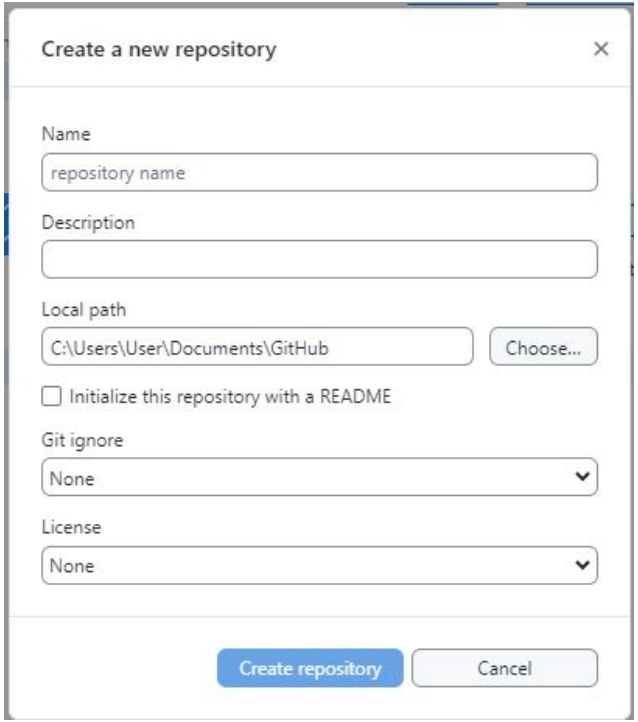
Para simplificar seria algo como colocar comando no cmd/prompt, a única diferença é que usamos um interface visual no local dos comandos digitados.

GITHUB DESKTOP

Para iniciar vamos começar com a **criação** de um **repositório** e a inclusão dos arquivos no nosso repositório, assim temos armazenados nossos arquivos.

Aqui temos a parte inicial referente a criação de um novo repositório, onde temos seu nome, descrição, local, git ignore e license.

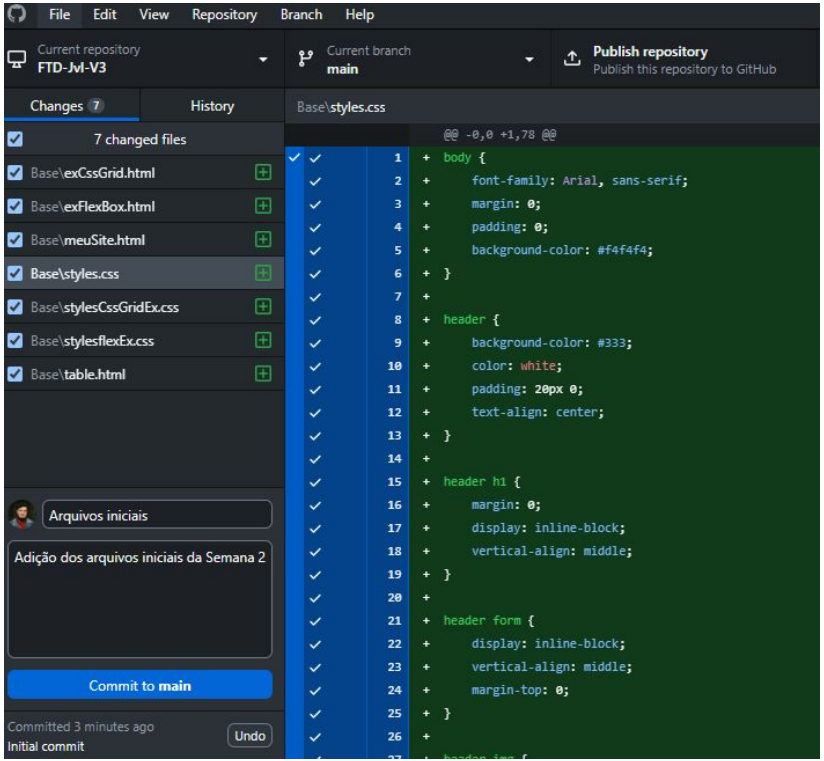




GITHUB DESKTOP

Após a criação do repositório, todo arquivo que for adicionado dentro do nosso repositório local fica disponível como um arquivo modificado e fica elegível a ser hospedado online.

Como ainda não criamos esse repositório online, até o momento ele só existe em nossa máquina, mas caso já existisse o processo seria só de adicionar os novos arquivos.



<LAB365>

<LAB365>

Revisão: HTML + CSS + JavaScript + Versionamento

<LAB365> SENAI

AGENDA | M1S04 - A3

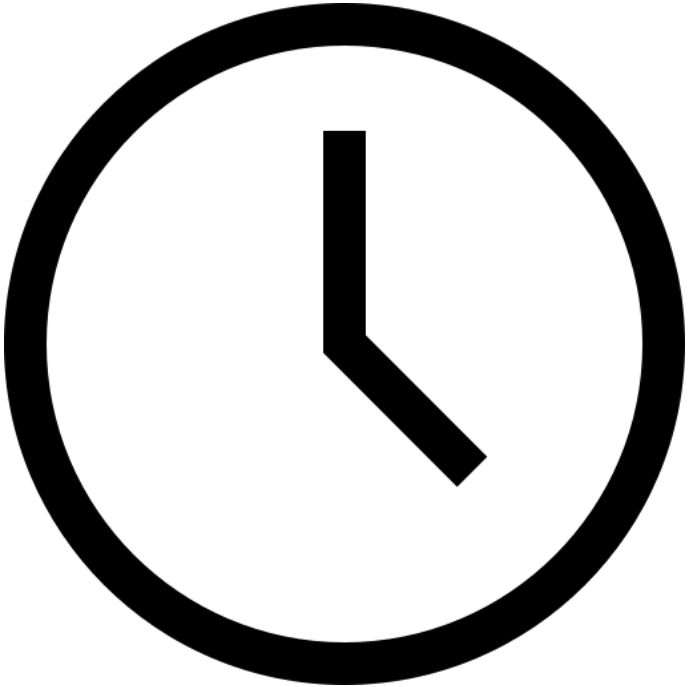
- Vivência
- Finalização de versionamento
- Revisão e correções

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:00
Retorno: 20:30



GITHUB DESKTOP

FTD-Jvl-V3 Private

Rep Teste FTD-Jvl-V3

Updated now

Star

Para enviar as alterações para nosso repositório, após criar o mesmo, temos de clicar no commit (para confirmar as alterações) e utilizar o push, assim subimos os arquivos no nosso repositório online.

GITHUB DESKTOP

Temos também os comandos **Clone e Fork**, ambos servem para manipulação de repositórios.

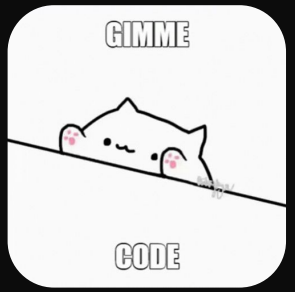
Clone serve para copiar um repositório do GitHub para o seu computador local.

Fork serve para criar uma cópia de um repositório de outra pessoa na sua conta do GitHub.

Clone vs Fork

Clone	Fork
Copia o repositório para sua máquina.	Copia o repositório para sua conta do GitHub.
Usado para trabalhar localmente.	Usado para contribuir em projetos de terceiros.
Não cria um novo repositório no GitHub.	Cria um novo repositório na sua conta.

TREINANDO NOSSAS HABILIDADES!



Resolva os seguintes desafios usando arrow functions:

- Crie uma arrow function que recebe uma temperatura em Celsius e retorna a temperatura convertida para Fahrenheit.
- Crie uma arrow function que:
  - Recebe peso (kg) e altura (m) como parâmetros.
  - Calcula o IMC usando a fórmula:
  - IMC = peso / (altura²)
  - Retorna uma string com o IMC e a categoria, de acordo com a tabela.

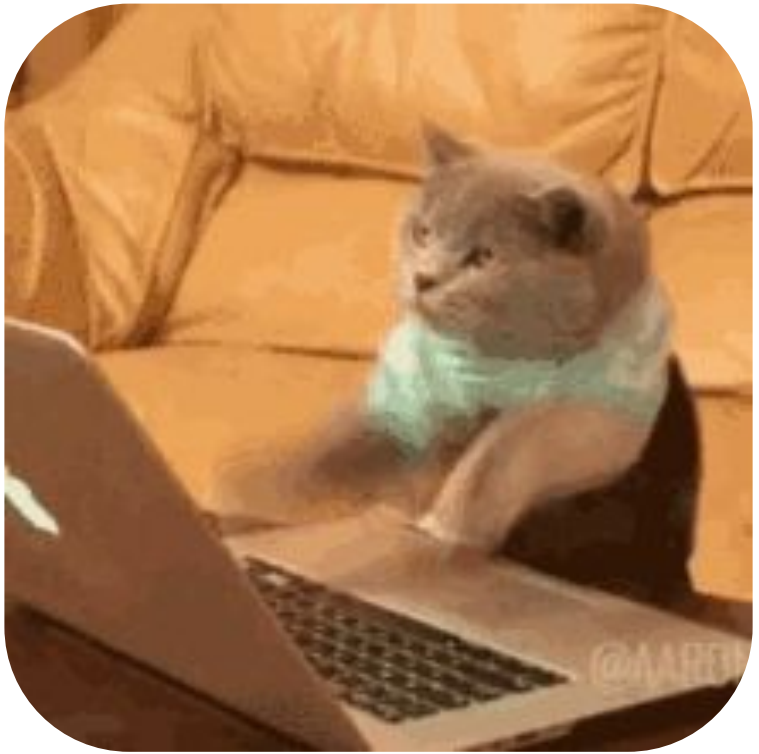
Adicione o código no github!

IMC	Categoria
Abaixo de 18.5	"Abaixo do peso"
18.5 - 24.9	"Peso normal"
25.0 - 29.9	"Sobrepeso"
30.0 ou mais	"Obesidade"

REVISÃO E CORREÇÕES

Nossa revisão será programando!

- HTML
- CSS
- Javascript
  - Estruturas Condicionais
  - Estruturas de repetição
- Arrow function
- Versionamento de código



VAMOS PARA OS EXERCÍCIOS!

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

<LAB365>