TYPESCRIPT

Uma informação inicial importante é que o TypeScript nos alerta sobre possíveis erros antes mesmo de colocá-lo em um ambiente de execução final.

TEXTCONTENT

textContent é uma propriedade que retorna ou define o conteúdo textual de um nó no DOM como uma string. Ele não é adequado para cálculos diretamente com valores numéricos, pois sempre retorna o texto como uma string. Além disso, textContent é responsável por ler todo o texto presente no elemento HTML referenciado, incluindo os elementos filhos, e permite modificar esse texto, substituindo todo o conteúdo textual pelo valor fornecido.

elementoSaldo.textContent = saldo.toString(); -

Ao usar **toString**, estamos convertendo a variável para uma string. Caso queiramos convertê-la para um valor numérico inteiro, podemos usar **parseInt**.

Se a intenção era lidar com números decimais, o correto seria mencionar **parseFloat.** exemplos:

tostring = “texto”

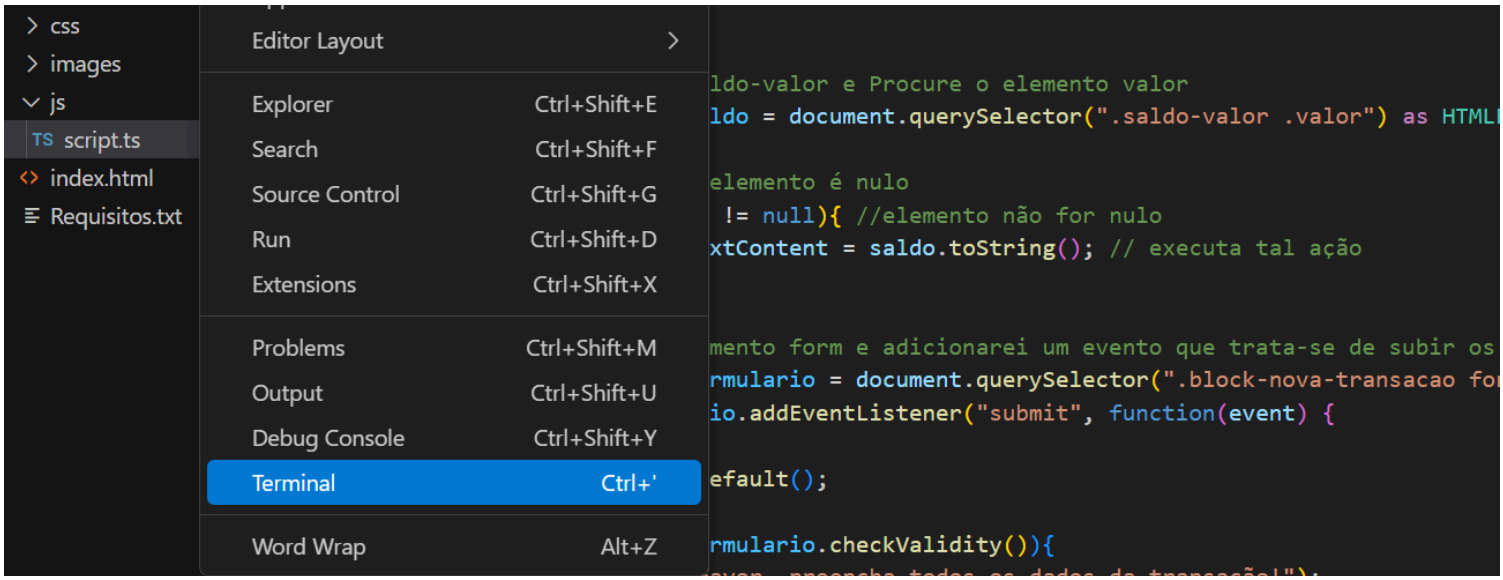
parseInt = “200"

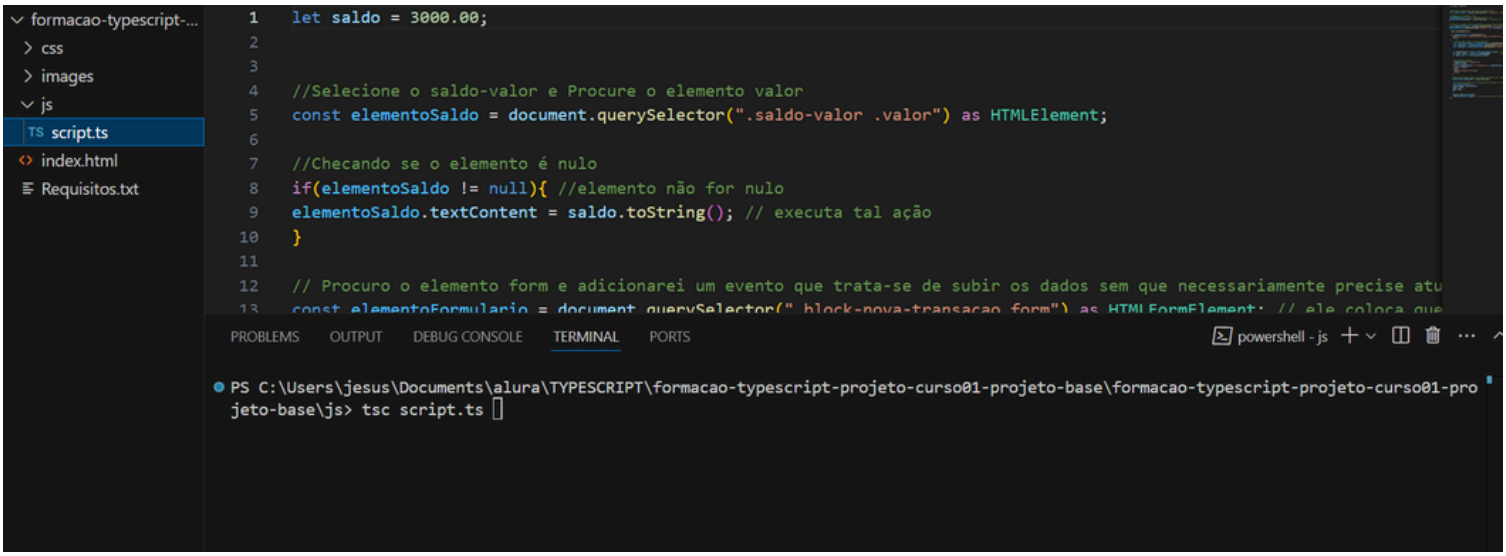
parseFloat= “2,00”

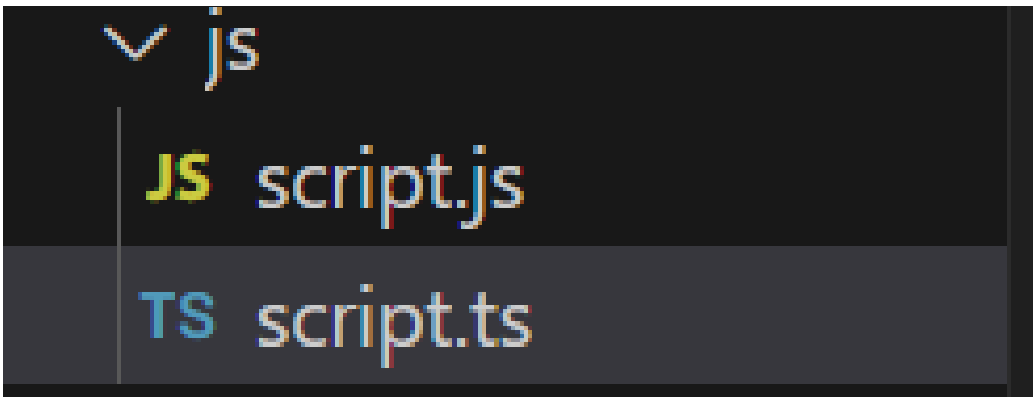
Olá, estudante! Vamos entender a diferença entre HTMLSelectElement e HTMLInputElement.

Imagine que você está construindo uma casa. **HTMLInputElement** é como um tijolo: um elemento básico e versátil que pode ser usado para várias coisas, como criar uma parede (um campo de texto), uma porta (um botão) ou uma janela (um campo de senha). Já o **HTMLSelectElement** é como uma porta específica, uma porta de entrada com opções pré-definidas, como uma lista suspensa (dropdown).No nosso código, **inputTipoTransacao** é uma lista suspensa (um <select> no HTML), onde o usuário escolhe um tipo de transação (depósito, transferência, etc.). Como é uma lista suspensa, precisamos usar **HTMLSelectElement** para que o TypeScript entenda que estamos lidando com um elemento que possui propriedades e métodos específicos de uma lista suspensa, como as opções disponíveis.Por outro lado, **inputValor e inputData** são campos de texto simples (<input type="number"> e <input type="date">, respectivamente). Eles são elementos de entrada de dados, mas não são listas suspensas. Por isso, usamos HTMLInputElement, que é o tipo mais genérico para campos de entrada de texto, abrangendo vários tipos de input.Usar os tipos corretos (HTMLSelectElement e HTMLInputElement) ajuda o TypeScript a entender o que cada variável representa e a fornecer sugestões de código (IntelliSense) mais precisas, evitando erros e tornando o desenvolvimento mais eficiente.Consegue pensar em um exemplo de como você usaria HTMLInputElement de forma diferente dos exemplos do código? Que tipo de input você criaria?

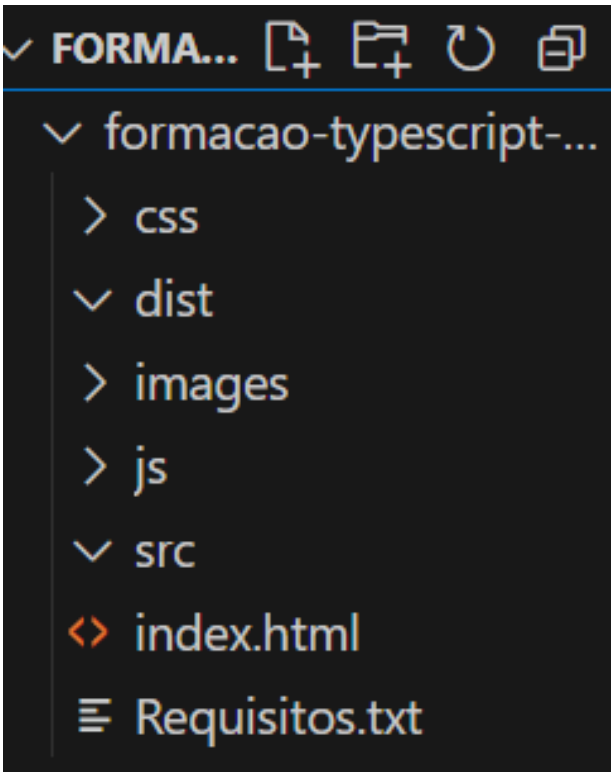
PARA FAZER O TYPESCRIPT INTERPRETAR E RODAR O CÓDIGO EM JAVASCRIPT (JS).







ORGANIZAÇÃO DAS PASTAS



ORGANIZAÇÃO

para manter a organização da aplicação, inseri duas pastas: dist e src.

DIST - denomina como distribuição, ou seja, qualquer código que corresponda ao ambiente de desenvolvimento estará lá, como: CSS, IMAGENS E HTML

SRC- significa como pasta de desenvolvimento, que irá sofrer alterações no processo, como: typescript

COMPILAÇÃO

tsc -w - Corresponde à um acompanhador que monitora qualquer arquivo TS

ESTRUTURA DE TIPOS (PRIMITIVOS)

let saldo2: number = 3000;

```
let nome: string =  "Rian";

let pago: boolean = false;

let qualquerVariavel: any = 22;  // o ANY correspondente significa que aceita qualquer tipo de variável
```

ENUMS - ENUMERAÇÕES

Trata-se de um conjunto de valores fixos, definidos dentro do código, que tornam a escrita mais polida e simplificam a especificação desses valores. Segue um exemplo:

```
enum Status {

  Pendente = "PENDENTE",

  EmProgresso = "EM_PROGRESSo",

  Concluido = "CONCLUIDO"

}


// USANDO O ENUM

function atualizarStatus(status: Status) {

  console.log(`O status atualizado é: ${status}`);

}

atualizarStatus(Status.Concluido); // SAÍDA: O STATUS ATUALIZADO É: CONCLUIDO
```

um de seus objetivos é trazer um padrão para que as letras não sejam colocadas de forma incorreta. exemplo:

```
const novaTransacao: transacao = {

    tipoTransacao: " transferência", -------> No cenário certo é para colocar “Transferência”

    data: new Date(),

    Valor: 0,
}
```

Const - valores que nao podem ser mudados

let - valores que podem ser mudados

obs. O typescript é uma ferramenta muito importante para garantir que o código com erro no ambiente de desenvolvimento não vá errado para o ambiente de desenvolvimento

PARA APRENDER

```
//Declarando variaveis primitivas

//let saldo2: number = 3000;

//let nome: string =  "Rian";

/* let pago: boolean = false;

let qualquerVariavel: any = 22;  // o ANY correspondente significa que aceita qualquer tipo de variavel


// arrays em TS

const lista: number[] = []; //Essa lista só pode conter números

lista.push(2, 3, 6, 8,) // O "push" Serve para inserir novas informações nesta array



//Tipos personalizados (type Alias)

  //Expicitando o que uma transacao tem que ter

type transacao = {

    tipoTransacao: TipoTransacao;

    data: Date;

    Valor: number;

}
```

```
// Enum


enum bipoTransacao{


DEPOSITO = "Depósito",

TRANSFERÊNCIA = "Transferência",

PAGAMENTO_BOLETO = "Pagamento de boleto"

}
const novaTransacao: transacao = {

    tipoTransacao: TipoTransacao.DEPOSITO, // exemplo de busca da enum

    data: new Date(),

    Valor: 0,
}
console.log(novaTransacao)
*/
```

EXIBIÇÃO DE DATA PADRÃO NA APLICAÇÃO E VALOR NUMÉRICO EM MOEDA

Para que isso seja possível, utilizaremos um método que já executa este comando:

ELEMENTOSALDO.TEXTCONTENT = **saldo.toLocaleString("pt-br", {currency: "BRL", style: "currency" })**

toLocaleString("pt-br", {currency: "BRL", style: "currency" }) = este comando faz com que a string se transforme em moeda e que seja brasileira

OUTRA POSSIBILIDADE:

Quando o valor usado é uma data e precisa ser modificado, veja:

```
  elementoDataAcesso.textContent = dataAcesso.toLocaleDateString("pt-br", {
```

weekday: "long", - Semana

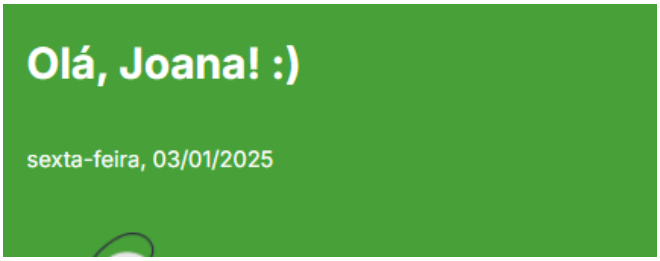
day: "2-digit", - dia

month: "2-digit", - mês

year: "numeric" - ano completo

});

resultado:



RESOLUÇÃO DE PROBLEMAS COM OS MÓDULOS ES6

O módulo ES6 (ou ES2015) é uma funcionalidade do JavaScript que permite organizar e reutilizar código de forma mais eficiente. Ele introduz as palavras-chave **import** e **export** para trabalhar com arquivos separados, possibilitando a importação e exportação de funções, objetos, ou variáveis entre diferentes módulos. exemplo:

```
export function soma(a, b) {
  return a + b;
}
```

```
import { soma } from './math.js';

console.log(soma(2, 3)); // 5
```

o módulo ES6 (ECMAScript 2015) é considerado uma forma de **encapsulamento** em JavaScript. Ele foi introduzido para organizar melhor o código, isolando partes dele em unidades independentes, chamadas módulos. Isso oferece diversos benefícios relacionados à manutenção, **reusabilidade e segurança** do código.

ORGANIZAÇÃO PARTE 2

Com finalidade de organizar os ts, criamos as seguintes pastas:

Components - Tudo que tenha cesso direto a aplicação, exemplo o arquivo **nova-transacao-components.ts**

Utils - Uma pasta que contenha funções livres na aplicação, como exemplo o arquivo: **formatador.ts**

Types - Representa os types e enums da aplicação, como exemplo o arquivo: **FormatoData.ts**

MÓDULOS

<script type= "module" src="js/main.ts"></script>

----> Esse código está inserindo um script no HTML com as seguintes características:

1. **type="module"**: Indica que o arquivo é um módulo ES6, permitindo o uso de **import** e **export** para organização do código. Também garante que o código seja executado em escopo próprio (não polui o escopo global).
2. **src="js/main.ts"**: Especifica o caminho do arquivo que será carregado. No caso, é um arquivo **TypeScript** (.ts), que provavelmente será transpilado para JavaScript antes de ser executado.

SEPARAÇÃO DE MÓDULOS E ALGUMAS DEFINIÇÕES

É importante que os módulos contenham apenas as funções que lhes foram atribuídas. Em caso de sobrecarga de demandas, deve-se criar um novo módulo. Exemplo:

No módulo ***novoSaldo,*** há uma condicional que, dependendo do tipo de transação, faz o saldo aumentar ou diminuir. No entanto, isso não é responsabilidade do módulo ***novaTransação***.

LANÇAMENTO DE ERROS

throw new Error

Em TypeScript (e também em JavaScript), o comando throw new Error é usado para lançar uma exceção do tipo Error. Isso significa que o programa interromperá a execução normal naquele ponto e passará o controle para o bloco catch mais próximo (se houver um bloco try-catch para tratar o erro). Caso não haja tratamento, a exceção resultará no encerramento do programa ou no comportamento padrão do ambiente (por exemplo, logar o erro no console do navegador).

- **throw new Error**:
  - Usado para sinalizar que algo deu errado na execução do programa.
  - Interrompe o fluxo normal do código ao lançar uma exceção.
  - Geralmente utilizado em lógica de validação, detecção de erros ou situações excepcionais.
- **alert**:
  - Usado para exibir mensagens ao usuário final em uma interface gráfica.
  - Não interrompe o fluxo do código, mas pausa a execução até que o usuário interaja (feche a janela do alerta).
  - É mais adequado para comunicação simples, mas não é recomendado para aplicações modernas devido à má experiência de usuário.

TRY CATCH

O try-catch em TypeScript (e JavaScript) é um bloco de código usado para **lidar com erros ou exceções** que podem ocorrer durante a execução do programa. Ele permite que você execute uma lógica específica caso um erro ocorra, em vez de permitir que o programa falhe abruptamente.

exemplo:

```
try {
  // Código que pode gerar um erro
} catch (erro) {
  // Código para lidar com o erro
}
```

const transacoes: Transacao[] = JSON.parse(localStorage.getItem("transacoes"));

. **localStorage.getItem("transacoes")**

- localStorage: É uma API do navegador que permite armazenar dados localmente no formato de string.
- .getItem("transacoes"): Recupera o valor associado à chave "transacoes" no localStorage. Se essa chave não existir, retornará null.

3. **JSON.parse(...)**

- O método JSON.parse() converte uma string JSON (como a que pode ser armazenada no localStorage) de volta em um objeto JavaScript ou array.
- Nesse caso, ele transforma a string recuperada de "transacoes" em um **array de objetos**, assumindo que o conteúdo armazenado é um JSON válido.

4. **Atribuição**

- O resultado da conversão (JSON.parse) é atribuído à variável transacoes.
- Caso o valor no localStorage seja null (ou seja, a chave "transacoes" não exista ou esteja vazia), isso pode resultar em null, o que pode causar erros dependendo de como o código manipula transacoes.

Em Resumo

Essa linha de código:

1. Recupera do localStorage a string associada à chave "transacoes".
2. Converte essa string de volta em um array de objetos do tipo Transacao usando JSON.parse.
3. Armazena o array convertido na constante transacoes.

ANALISANDO CÓDIGO

localStorage.setItem("saldo", saldo.toString());

1. **localStorage.setItem(key, value)**
   - A função localStorage.setItem é usada para armazenar um par chave-valor no **localStorage**.
   - **key**: É o nome da chave usada para identificar o valor armazenado. Nesse caso, é "saldo".
   - **value**: É o valor que será armazenado. Ele deve ser uma string.

2. **saldo.toString()**
   - O método toString() é chamado no valor da variável saldo.
   - Isso converte o valor de saldo (seja ele um número, objeto, ou outro tipo compatível) para uma string, porque o localStorage só aceita strings como valores.
3. **Armazenamento no localStorage**
   - O localStorage salva permanentemente o valor, até que seja explicitamente removido ou sobrescrito.
   - O valor ficará acessível mesmo após recarregar a página ou fechar e reabrir o navegador.

EXTRATO

const transacoesOrdenadas: Transacao[] = listaTransacoes.sort((t1, t2) => t2.data.getTime() - t1.data.getTime()); },

Contexto

1. **listaTransacoes**: É um array de objetos representando transações.
2. **Transacao[]**: Indica que transacoesOrdenadas é um array de objetos do tipo Transacao. Esse tipo pode ter sido previamente definido com propriedades como data (provavelmente do tipo Date) e outros detalhes de uma transação.

Explicação

O código usa o método .sort() para ordenar os elementos do array listaTransacoes. O método .sort() recebe uma função de comparação com dois argumentos (t1 e t2), que representam dois elementos do array. A função de comparação deve retornar:

- **Um valor negativo**: Se t1 deve vir antes de t2.
- **Zero**: Se a ordem entre t1 e t2 não precisa mudar.
- **Um valor positivo**: Se t1 deve vir depois de t2.

No caso, a função de comparação é:

typescript

Copiar código

(t1, t2) => t2.data.getTime() - t1.data.getTime();

- **t2.data.getTime()**: Converte a data do segundo elemento (t2) para um número de milissegundos desde 1º de janeiro de 1970.
- **t1.data.getTime()**: Faz o mesmo para o primeiro elemento (t1).
- A subtração **t2.data.getTime() - t1.data.getTime()**retorna um número:
  - Positivo, se a data de t2 for mais recente que a de t1 (fazendo t2 vir antes de t1).
  - Negativo, se a data de t1 for mais recente que a de t2 (fazendo t1 vir antes de t2).

Resultado

As transações são ordenadas em ordem decrescente de data (mais recente primeiro).

Exemplo

Se listaTransacoes for:

typescript

```
[
  { id: 1, data: new Date('2025-01-05') },
  { id: 2, data: new Date('2025-01-07') },
  { id: 3, data: new Date('2025-01-03') },
]
```

Após a ordenação:

typescript

```
[
  { id: 2, data: new Date('2025-01-07') },
  { id: 1, data: new Date('2025-01-05') },
  { id: 3, data: new Date('2025-01-03') },
```

novo grupo

```
if(labelAtualGrupoTransacao != labelGrupoTransacao) {
        labelAtualGrupoTransacao = labelGrupoTransacao;
        gruposTransacoes.push({
         label: labelGrupoTransacao,
         transacoes: []
        });
    }
```

O código verifica se o rótulo do grupo de transações atual é diferente do rótulo anterior. Se for, isso indica que estamos iniciando um novo grupo. Nesse caso, o código cria um novo objeto para representar esse grupo e o adiciona ao array gruposTransacoes.