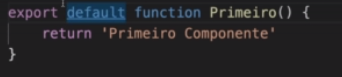
* Npx create-react-app ele irá criar temporariamente a função create-react-app, e excluí-la (sem cria-la globalmente) após realizar seus script
* ReactDOM.render() recebe dois parâmetros
  + 1º - O que eu quero renderizar (pode ser uma string)
  + 2º - Elemento o qual quero renderizar (pode ser uma div)
* <noscript>Mensagem<noscript/> - Caso o usuário desabilite o javascript no navegador, a mensagem irá aparecer para ele habilitar
* Div com id=root – É a Div onde será injetado todo o código react
* JSX – Parece html mas não é, ele transforma o “html” em Javascript. Sempre que usamos o jsx em um código no react, temos que importar “import React from ‘react’”
* Para inserirmos código puramente javascript dentro de JSX, colocamos o código entre parênteses

Carregando CSS

* O arquivo CSS tem que ter o mesmo nome do arquivo js. Exemplo: index.js e index.css
* É feita uma importação indireta = import ‘./index.css’
* Ele irá puxar diretamente o estilo CSS

Primeiro Componente

* Para conseguirmos exportar a função para fora do arquivo (no exemplo, do arquivo Primeiro,js) temos que colocar o export:



* Posso referenciar esse componente como uma tag:



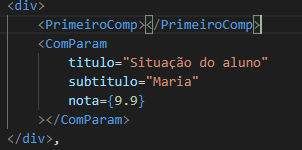
* Eu posso referenciar por qualquer nome esse “Primeiro” caso eu exporte por padrão (export default):

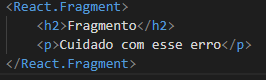
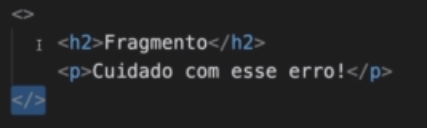


Componente com Propriedade

* Dá utilizarmos componentes com .js e .jsx (no ponto de vista do React, não há diferença)
* Utilizou-se Jsx apenas por conta da IDE “ajudar” durante a programação (a completar alguma função, por exemplo)
* 
* Propriedades é somente leitura!!!!

**React Fragment**

* Não conseguimos passar dois elementos jsx sem estarem “encapsulados”, por isso usamos div em render()
* Com o React.Fragment, não precisamos enclausular os Jsx’s em uma div, ele irá juntar as JSX e deixar elas individuais



* Quando eu uso o <React Fragment> eu consigo passar uma propriedade, por exemplo, um <React Fragment key=””>, diferente de usar só o <>

**Componente App**

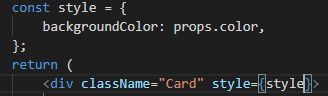
* Se eu tiver export default, posso criar uma função anônima:
  + Export default function(props){}
* Se eu não tiver export default, não posso criar:
  + Export default function App(props){}
* Podemos usar uma arrow function:
  + Export default (props) =>{}

**Componente Card**

* class é uma função de javascript, e não se refere a mesma função que a de html, para ser a mesma de html, usa-se className
* 

O props.children irá pegar todos os Filhos (ou conteúdos dentro da tag Card) dentro do Pai “Card”

* Passar propriedade de cor por parâmetro



* Caso não seja passado nenhum parâmetro, podemos usar uma cor padrão usando:



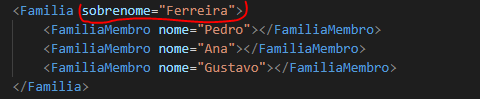
**Componente Filho**

* Para não passar todos os atributos um por vez, fazemos o seguinte:

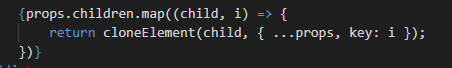


Inserimos o {...props}

* Quando quisermos passar a informação de pai para filho assim:



Iremos fazer o seguinte (dentro de Familia.jsx) :



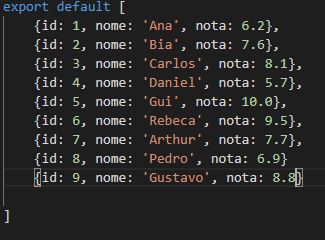
Ele irá pegar todos os elementos do filho, e realizar um clone deles passando as propriedades do pai em cloneElement

Map() funciona para mapearmos um array em outro array. Como props.children é uma lista (um array de filhos), podemos usar a função map

A função irá gerar um índice i que pode ser passado também para os filhos.

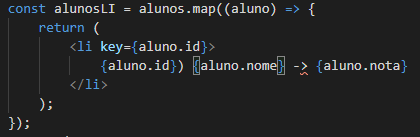
**Repetição (mostrar lista)**

* Iremos criar um arquivo .js que exporte um array com os dados de um aluno



Depois é só eu importar na no componente que eu criei

* Depois é só realizar o map() transformando os dados do array alunos.js em <li>



Usa-se o key pois o react utiliza o parâmetro key da lista para que se caso ocorra alguma mudança naquela lista, ele rapidamente perceba isso e altere a página.

Podemos usar também o aluno.nome, porém se tiver 2 nomes iguais, irá dar problema

* No desafio, quisemos aplicar o conceito de denominar um className condicionalmente, ou seja, se o id da linha fosse par, tinha o className ‘Par’



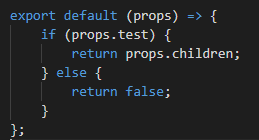
Quando eu uso === eu vou comparar o valor e o TIPO da variável

**Renderização Condicional**

* Podemos instanciar o componente no App.jsx passando o parâmetro “numero” para ele, aí dentro do arquivo do componente, inserimos a renderização condicional dessa forma:



* Podemos criar de outra forma, criando um arquivo If.js separado



E chamando no jsx assim:



* Podemos criar um else: Vê no código, mais fácil

**Comunicação direta entre componentes**

* Comunicação direta é usado o props direto, como utilizamos anteriormente:

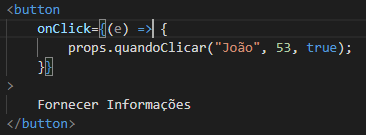


**Comunicação indireta entre componentes**

* Queremos que o filho passe informações para o componente Pai
* Exemplo: Um botão quando apertado, envia informações para o componente Pai
* Passamos uma função do pai para o filho



* Colocamos a função aonde desejamos:



**Componentes com Estado**

* Se você tentar mudar uma variável no componente, ele não irá mudar na interface, você precisa do state, para o React entender que houve mudança e atualizar a interface!
* Iremos utilizar o React.useState e instanciaremos as variáveis const da seguinte forma:



Porém o useState irá retornar um array, onde na primeira posição haverá o valor, e a segunda será uma função para alterar o valor (assim como utilizado em classes em Java) e a interface

* Então na função chamada no Filho, iremos setar o novo valor para nome:



**Componente Controlado**

* O que o sistema chama de verdade absoluta são os seus dados! E você não consegue mudar o valor desse dado diretamente pela interface. Você tem que setar o dado, para apenas assim esse novo dado ser refletido na interface gráfica
* onChange irá pegar qualquer mudança que ocorra no input (apagar uma letra, escrever outra, etc)
* Iremos inserir uma função que dê um set nos dados com o que você digitou:



* Quando queremos colocar um componente não controlado (que não mude nenhum dado, inserimos o parâmetro readOnly nele:



* Quando queremos colocar um componente não controlado por nada, inserimos o valor undefined nele:
* 