Componentes
Class vs Funcional

Componente de Class vs Funcional

React

```
import { StrictMode } from "react";
import ReactDOM from "react-dom";

import ArrowOla, { ClassOla, FunctionOla } from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(
      <StrictMode>
            <ArrowOla />
            <ArrowOla />
            <ClassOla />
            <FunctionOla />
      </StrictMode>,
      rootElement
);
```

Olá, Linguagens Script!

Olá, Linguagens Script!

Olá, Linguagens Script!

Olá, Linguagens Script!

Componente de Class vs Funcional

React

```
import React from "react";
class ClassOla extends React.Component {
     render() {
          return <h1>Olá, Linguagens Script!</h1>;
     }
}
```

Componente de Class

```
const FunctionOla = function () {
     return <h1>Olá, Linguagens Script!</h1>;
};
```

Componentes Funcionais

```
const ArrowOla = () => <h1>Olá, Linguagens Script!</h1>;

export default ArrowOla;
export { ClassOla, FunctionOla };
```


Componente de Class vs Funcional

React

```
import { StrictMode } from "react";
import ReactDOM from "react-dom";
import ArrowOla , { ClassOla} from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(
     <StrictMode>
          <ArrowOla nome="Linguagens Script" />
          <ArrowOla nome="JavaScript" />
          <ArrowOla nome="React" />
     </StrictMode>,
     rootElement
);
```

Olá, Linguagens Script

Olá, JavaScript

Olá, React

Comp. Class e Funcional > Props

React

```
import React from "react";

class ClassOla extends React.Component {
      render() {
            return <h1>Ola, {this.props.nome}</h1>;
      }
}

const ArrowOla = (props) => <h1>Ola, {props.nome}</h1>;

export default ArrowOla;
export { ClassOla };
```

Ola, Linguagens Script!

Componentes Puros

React

```
import React from "react";

class ClassOla extends React.PureComponent {
      render() {
            return <h1>Ola, {this.props.nome}</h1>;
      }
}

const ArrowOla = React.memo(
     (props) => <h1>Ola, {props.nome}</h1>
);

export default ArrowOla;
export { ClassOla };
```

Ola, Linguagens Script!

Componentes Puros

```
import React from "react";

class
    r
    }
}

const
    (p
);

export
export {
```

Componentes puros criados desta forma são armazenados em cache com base nas suas *props* e *valores* de estado interno.

Isso significa que, ao invés do componente ser inicializado sempre, o React irá reutilizá-lo melhorando a performance.

Componente de Class vs Funcional

- **Sintaxe**
  - Escolha depende de gostos e características do programador;
  - O componente de class usa a sintaxe ES6 e extende a componentes React com o método render que retorna elementos React. Componentes funcionais com hooks, são funções JavaScript puras que também retornam elementos React;
- ***Estado e Métodos Lifecycle***
  - Antes do React 16.8 os componentes funcionais eram stateless;
  - Em termos de ciclo de vida, é possível usar o *useEffect* hook em componentes funcionais para obter o mesmo efeito que métodos como *componentDidMount*, *componentDidUpdate* e *componentWillUnmount* dos componentes de classse.
- O **this** e métodos de *binding* deixam de ser necessários em componentes funcionais; Partilha de estados entre componentes class é mais tediosa;

Considerações de Desempenho

- Adicionar keys a lista de elementos

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

- Construa componentes pequenos!
  - Caso contrário… muitas variáveis de estado no componente, implica, renderização de tudo!
- React.memo
  - Muito útil para componentes que são completamente de apresentação e apenas obtem as props e mostram alguma interface ao utilizador. Não se deve abusar…

Considerações de Desempenho

- Evitar montar e desmontar componentes

```
import React, { useState } from "react";
const MountingComponent = () => {
  const [show, setShow] = useState(false);
  return (
    <main>
      <button onClick={() => setShow((show) => !show)}>Show the text</button>
      {show ? <p>I am the text</p> : null}
    </main>
  );
};
```

```
const CSSWay = () => {
  const [show, setShow] = useState(false);
  return (
    <main>
      <button onClick={() => setShow((show) => !show)}>Show the text</button>
      <p style={{ opacity: show ? "1" : "0" }}>I am the text</p>
    </main>
  );
};
```

Gestão de Estados

Comunicação entre Componentes

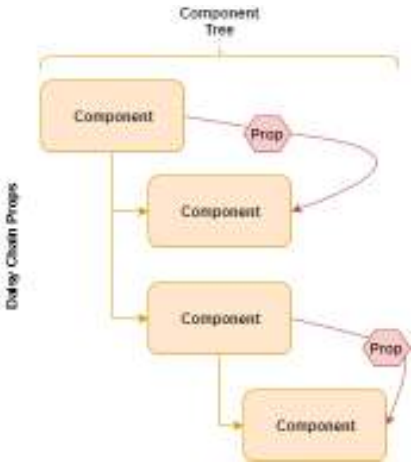
- Passar informação do element pai para o filho: ***Usar props*** (atributos)
  - `<ComponentFilho param={infoParaComponenteFilho} />`
- Passar informação do filho para o pai: ***Callbacks***
  - `paiCallback = (infoDoFilho) =>`

    `{ /* processInfoDoFilho*/};`
  - `<ComponentFilho callback={paiCallback}> />`
- ***React Context / useContext***
  - Variáveis globais para a subárvore de componentes
  - Fornece uma forma de passar dados pela àrvore de componentes sem necessidade de passar propriedades manualmente em todos os níveis*
- ***Redux / useReducer ****

**não será explorado no contexto prático em 2021/2022*

State and props

- Permite armazenar informação que influencia o resultado da renderização.
  - **Props** são uma característica básica do React e permite encadear informações de pai para filho. Portanto, são passados para o componente como parâmetros de funções.
  - **State** gerido dentro do componente, como variáveis declaradas dentro da função.



Gestão de Estados

- Existem dois locais principais no qual os estados podem residir:
  - Nos componentes:
  - Armazenamento central
- Nos componentes
  - Fluxo de dados facil de compreender
  - Trabalha de forma faciitada quando a aplicação não é demasiado complexa de forma a que os componentes não precisem partilhar dados e interagir
- Exite um conjunto de outras técnicas para manipulação de dados
  - render props
  - the Context API
  - Redux.