The four types of React components are functional, class, pure, and high-order components.

STATES

* un componente no controlado no usa useState, usa useRef

USEEFFECT:

• useEffect para realizar acciones después del montaje del componente

• el useRef es un hook que se usa para referenciar elementos del DOM (eg, sacar el value de un input)

* solo existe un useEffect por código, de tener varios, solo el último se ejecutara
* puede tener un return, de no tenerlo, se puede generar una fuga de memoria (que llame al useEffect por cada vez que haga referencia al componente, el return termina esa referencia)

HOOKS

* un custom hook es una funcion que empieza por "use"

USEREF

* para detectar un elemento en el DOM (y por ende su valor, como en un input) es util usar "useRef" (ver "general\_practice / EG5"
* un componente no controlado no usa useState, usa useRef

Props

* when we using props, it to send the props name name from the render, and catch them with the word “props” in the component, like: “<Props\_eg3 name={"Byron"} age={22}/> ….. <h1>HI {props.name}, are you {props.age} ?</h1>”
* in order to destructure props to a another component, you have to send each prop to the component (ref. \UdemyBC\05-Hook-App\03-functionalComponents\MultipleCustomHooks.js)

useLayoutEffect

* Works kinda like “useEffect” BUT, the useLayoutEffect runs at the beginning (when the browser rendered the dom but not the elements), and the useEffect after (when the browser rendered the elements)
* it runs after the component has rendered but before the browser paints the screen

Memo

* “Memorize this component, so, only if the parent component has changes, the children component with memo, won’t rewrite”
* This will memorize anything that returns : “const messageMemorize = useMemo(), [];” and it will remain, unless the dependencies of the memo changes
* If has “[]” at the end, it will memorize each time the element inside changes (similar to the “useEffect” where the event occur if the dependencies on the [] changes) IF NOT, it will only memorize it one time

useCallback

* Same as the memo, it memorizes no component, but a function (a callback), that could be in a component or hook, and it will be rendered/memorized if something in it changes
* If we want to use params, check for the “\UdemyBC\05-Hook-App\06-memos\CallBackHook.jsx”

Reducer

* it’s a normal function, actually, has to be a pure function:
  + it needs to resolve itself internally
  + it should (must not ?) call another functions
  + should not have async task
  + must not call the "localStorage / sessionStorage" (are considered side effects even though are sync funcs)
* it will always return an state (the hook)
* an alternative to “useState”
* IN ALL CASES , when using the hook, the hook and the dispatch must have this order:  
  - const [todos, dispatch] = useReducer(ALWAYS THE FILE/FUNCTION FOR DISPACTH, ALWAYS THE STATE TO SEND/RETURN, init);  
  - export const ToDoReducer = (ALWAYS THE STATE TO SEND/RETURN,, ALWAYS “ACTION”) => {}
* S

Routes

* the simbol on the path prop: "/\*", redirects the route if its incorrect or misspelled: "<Route path="/\*" element={<page404/>}" BUT ITS NOT THAT USED, because the URL stays on the misspelled URL, instead, we use the router hook: "Navigate"<  
  Route path="/\*" element={<Navigate to="/about"/>} />

useContext

* Its escencially a high oreder component, what does its use “<Routes>” wich will recive an array of its children (and those are a route array) with “<Route>”