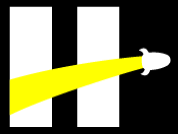
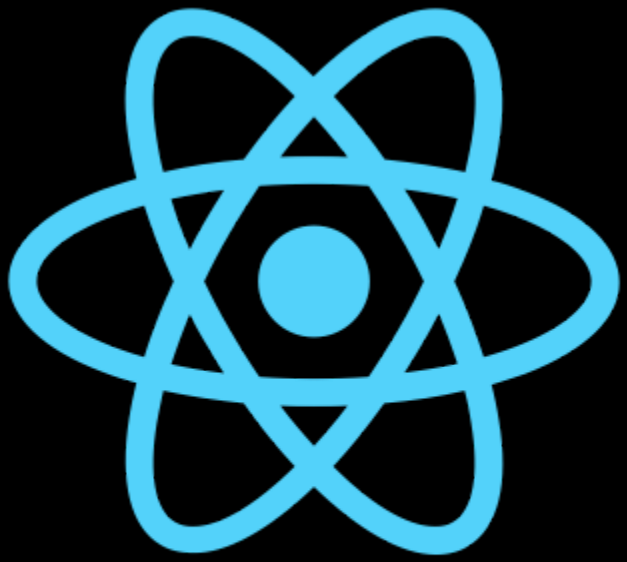
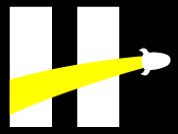

HENRY





Hooks



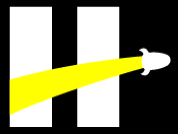
Hook de estado

useState

Permite guardar estados en componentes funcionales.

```
1  function Example() {
2    const [count, setCount] = useState(0);
3    ...
4    function increment() {
5      setCount(count + 1)
6    }
7    return  <div>
8      <button onClick={increment}>Suma 1</button>
9      <p>{count}</p>
10   </div>
11   ...
12 }
```

useState() es una función que devuelve un arreglo con dos valores. El primero es una variable con el valor del estado y el segundo valor es una función se usa para actualizar el estado. Acepta un nuevo valor de estado y sitúa en la cola una nueva renderización del componente.



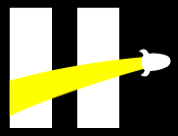
Hook de efecto

useEffect

Permite realizar efectos secundarios en componentes funcionales. Efectos secuendarios pueden ser: Traer data con AJAX, cambiar el DOM manualmente, suscribrise a algo, etc..

```
1 function Example() {
2   const [count, setCount] = useState(0);
3   ...
4   useEffect(() => {
5     // Update the document title using the browser API
6     document.title = `You clicked ${count} times`;
7   });
8   ...
9 }
```

De forma predeterminada, React ejecuta los efectos después del primer renderizado *y* después de cada actualización

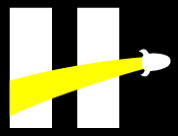


Hook de efecto

componentDidMount con useEffect

```
1 function Example() {
2   const [count, setCount] = useState(0);
3   ...
4   useEffect(() => {
5     // Update the document title using the browser API
6     console.log("this component has been mounted");
7   }, []);
8   ...
9 }
```

Podemos realizar un efecto que actúe de manera similar al método componentDidMount de los class component

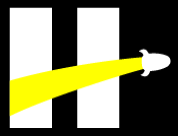


Hook de efecto

componentDidUpdate con useEffect

```
1 function Example() {
2   const [count, setCount] = useState(0);
3   ...
4   useEffect(() => {
5     // Update the document title using the browser API
6     console.log("First time or the count state was updated");
7   }, [count]);
8   ...
9 }
```

Podemos realizar un efecto que actúe de manera similar al método componentDidUpdate de los class component

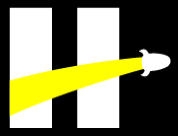


Hook de efecto

componentWillUnmount con useEffect

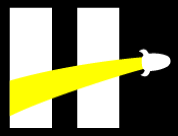
```
1 function Example() {
2   const [count, setCount] = useState(0);
3   ...
4   useEffect(() => {
5     // Update the document title using the browser API
6     return () => console.log("this component has been removed");
7   }, []);
8   ...
9 }
```

Podemos realizar un efecto que actúe de manera similar al método componentWillUnmount de los class component



Hook useReducer

```
1  const initialState = {count: 0};
2
3  function reducer(state, action) {
4    switch (action.type) {
5      case 'increment':
6        return {count: state.count + 1};
7      case 'decrement':
8        return {count: state.count - 1};
9      default:
10       throw new Error();
11   }
12 }
13
14 function Counter() {
15   const [state, dispatch] = useReducer(reducer, initialState);
16   return (
17     <>
18       Count: {state.count}
19       <button onClick={() => dispatch({type: 'decrement'})}>-</button>
20       <button onClick={() => dispatch({type: 'increment'})}>+</button>
21     </>
22   );
23 }
```



Hook useRef

```jsx
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` apunta al elemento de entrada de texto montado
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

IMPORTANTE: podemos manipular el DOM manualmente con este hook, pero es desaconsejado hacerlo.