REACT - STATE

El estado es el núcleo principal de React, éste nos va a permitir que nuestra interfaz cambie y algo MUY IMPORTANTE A RECORDAR ES:

“SI ALGO DEBE CAMBIAR EN NUESTRA INTERFAZ, TIENE QUE SER UN ESTADO*”

* Ésta norma no se aplica si el cambio es estético como resultado de un :hover en CSS

Para usar el estado de React haremos uso del primer hook que vamos a ver, useState()

REACT - useState()

Los hooks son funciones que nos van a permitir interactuar con React más allá del JSX, digamos que la API de React está compuesta por JSX y los Hooks.

useState es una función que recibe como parámetro el estado inicial de nuestro estado, éste estado inicial puede ser cualquier dato válido en JavaScript, un número, un string, un booleano o incluso una función que devuelva un valor.

Al utilizar useState, la función nos devuelve un array de dos posiciones:

```
const state = useState(0);
console.log(state);
```

```
▼(2) [0, f]
  0: 0
  ▶1: f ()
  length: 2
```

REACT - useState()

La función para cambiar el estado es necesaria porque NO PODEMOS cambiar el estado de ninguna otra forma, ésto es así porque react repinta el componente al que pertenece el estado con el nuevo valor, gestionar esos cambios si pueden suceder desde cualquier sitio es muy complicado a nivel estructural, por eso react optó por hacerlo de esta forma.

Gracias al destructuring podemos nombrar el estado y su función como necesitemos, la única norma que debemos seguir es que la función empiece por “set” y el nombre del estado.

```
const [state, setState] = useState(0);
```

REACT - useState() - Demo

Esto es una representación de cómo sería el comportamiento interno de react al utilizar useState.

```
let state = undefined;

const setState = newValue => {
  state = newValue;
};

const useState = initialValue => {
  setState(initialValue);
  return [state, setState];
};
```

REACT - useState() - Ciclo de vida

Entender el ciclo de vida de un componente es clave para entender react.

```jsx
const Counter = () => {
  const [counter, setCounter] = useState(0);
  const [step, setStep] = useState(1);

  const handleCounter = () => {
    setCounter(counter + step);
    console.log(counter);
  };

  const handleStep = () => {
    setStep(step + 1);
    console.log(step);
  };

  return (
    <div>
      <h1>Counter: {counter}</h1>
      <h2>Step: {step}</h2>
      <button onClick={handleCounter}>Increment Counter</button>
      <button onClick={handleStep}>Increment Step</button>
    </div>
  );
};
```

Cuando hacemos un click en uno de los dos botones llamamos a la función que “maneja” ese evento.

Es MUY IMPORTANTE entender que hasta que esa función no termine su trabajo, el componente no se va a repintar con los cambios del estado.

El console.log SIEMPRE mostrará el estado actual, no el que se produce tras el cambio.