React: State

En nuestro componente App tenemos el siguiente código:

App.js
```
let numero = 1;

function App() {
  return (
    <>
      <h1>{numero}</h1>
      <button onClick={function () {
        window.alert('Hola')
      }}>Click</button>
    </>
  )
}
```

La variable número está declarada fuera de la función, por lo que su valor podrá ser leído y podríamos pensar que su valor se mantendrá a lo largo de la ejecución de la aplicación.

En el botón ponemos onClick en vez de onclick, ya que queremos que sea React quien maneje lo que ocurra cuando hacemos click.

Dentro del onClick, entre llaves, definimos la función que queremos que se ejecute cuando hagamos click en el botón. De esta manera, cada vez que hagamos click se mostrará una alerta con el mensaje 'Hola'.

Podríamos escribirlo también de la siguiente manera:

```
let numero = 1;

function App() {

  function mostrarMensaje(){
    window.alert('Hola');
  }
  return (
    <>
      <h1>{numero}</h1>
      <button onClick={mostrarMensaje}>Click</button>
    </>
  )
}
```

Dentro de la función App hemos definido la función mostrarMensaje(), que muestra una alerta.

En el onClick, en este caso, escribimos el nombre de la función, **sin paréntesis**. Estamos diciendo a React que ejecute esa función **cuando** hagamos click en el botón. Si pusiéramos onClick={mostrarMensaje()}, React ejecutaría esa función al iniciar la aplicación en vez de cada vez que hagamos click.

Cambiamos la función para que sume 1 a la variable **número** y muestre el valor en consola.

```
let numero = 1;

function App() {

  function mostrarMensaje(){
    numero++;
    console.log(numero);
  }
  return (
    <>
      <h1>{numero}</h1>
      <button onClick={mostrarMensaje}>Click</button>
    </>
  )
}
```

Si hacemos click en el botón, veremos que en la consola se está incrementando el valor de número pero **no** en el HTML.

Cuando React renderiza un componente, pinta ese componente tal y como está en ese momento. Cuando ese componente se modifica, React renderiza únicamente la parte que ha cambiado. Entonces, podríamos pensar que, al cambiar el valor de ese número, debería actualizarse el HTML renderizado por React. Pero no ocurre eso porque la variable número es ajena al componente, no forma parte de él así que React no tiene en cuenta el cambio en el valor. Para que React modifique el HTML cuando una variable cambia de valor, usaremos el estado de un componente o **state**: el estado de un componente son las variables asociadas a ese componente.

Antes de continuar con React, vamos a JavaScript normal. Tenemos un array de números:

```
let numeros = [1,2,3]
```

Podemos escribir lo siguiente para asignar los valores de ese array a variables independientes:

```
let [primero, segundo, tercero] = numeros;
```

Tendremos 3 variables: **primero**, **segundo** y **tercero** con los valores en las posiciones 0,1 y 2 del array.

Imaginaos que tenemos una función que recibe un valor y modifica un valor previamente declarado:

```
let numero = 5;
function cambiarNumero(nuevoNumero){
  valor = nuevoNumero;
}
```

En la función, podríamos comprobar si el valor que nos llega cumple unas condiciones para especificar el comportamiento de esa función. Por ejemplo, lo sumaremos únicamente si el valor es mayor que 0:

```
unction cambiarNumero(nuevoNumero){
  if(nuevoNumero>0){
    numero = nuevoNumero;
  }
```

De este modo, no solo tendríamos una función para modificar el valor, si no que también estamos controlando el comportamiento de esa función.

Si volvemos a React, haremos esto usando el módulo **useState**. Para ello, en el import de

App.js

```
import React from 'react';
```

le indicaremos que utilice también el módulo **useState**.

App.js

```
import React, {useState} from 'react';
```

Entonces, en la función App pondremos lo siguiente:

App.js

```
let [numero,cambiarNumero] = useState(5);
```

Cuando declaramos el estado, el valor que ponemos entre paréntesis es el valor inicial que queremos que tenga y lo guardamos en la variable **numero**. La función useState(5) nos devuelve un array con una variable **número** y una función **cambiarNumero.** A esta función le podemos poner el nombre que queramos, pero por convención se usa el nombre **set** seguido del nombre de la variable. En este caso **setNumero**

Ahora tendremos que definir el comportamiento del onClick

App.js

```
<button onClick={function(){
      setNumero(8)
    }}>Click</button>
```

Decimos que, al hacer click, llame a la función **setNumero**y le pasamos un nuevo valor entre paréntesis. Lo que hará esta función es modificar la variable **número** del estado y React, al ver que se ha modificado, actualiza automáticamente el HTML.

En este caso, cuando hagamos click, el valor inicial de 5 cambia a 8. Podemos decir que al hacer click en botón, coja el valor que tiene la variable **numero** en ese momento y le sume 1:

App.js

```
<button onClick={function () {
      setNumero(numero+1)
    }}>Click</button>
```

Podemos también sacar esa función a una función aparte y llamarla al hacer click en el botón:

App.js

```
function sumar(){
   setNumero(numero+1);
  }

  return (
    <>
      <h1>{valor}</h1>
      <button onClick={sumar}>Click</button>
    </>
  )
```

De este modo, cada vez que hagamos click en el botón, el número se actualiza y pintamos el nuevo valor en el HTML.

Usando las herramientas de desarrollador de React (**React Developer Tools**) podremos ver toda la información sobre nuestra aplicación y nuestros componentes: su estado, los props que recibe y muchos otros datos.

Vamos a crear un nuevo componente Persona. Importamos React y **useState** de 'react'

Persona.js
```
import React, {useState} from 'react';

function Persona(){
   return (
      <p>En componente Persona</p>
   )
}

export default Persona;
```

En el componente App, importamos el componente Persona:

App.js
```
import Persona from './Persona';

function App() {

  let [numero, setNumero] = useState(5);

  function sumar(){
    setNumero(numero+1);
  }

  return (
    <>
      <Persona />
      <h1>{numero}</h1>
      <button onClick={sumar}>Click</button>
    </>
  )
}
```

Si miramos las herramientas de desarrollador de React en el navegador, veremos que nos sale el componente App y, dentro de él, saldrá el componente Persona. Ahora podemos pasar unas propiedades al componente Persona:

App.js
```
function App() {

  let [numero, setNumero] = useState(5);
  function sumar(){
    setNumero(numero+1);
  }

  return (
    <>
      <Persona nombre ="Iker" edad={40}/>
      <h1>{numero}</h1>
      <button onClick={sumar}>Click</button>
    </>
  )
}
```

Podremos ver en las herramientas de desarrollador que el componente Persona recibe propiedades: nos indica las claves y los valores de esas propiedades.

Ahora vamos a cambiar el componente Persona para decir que pinte en el HTML los valores que recibe en props:

Persona.js
```
function Persona(props){
    return (
        <div>
            <p>Nombre: {props.nombre}</p>
            <p>Edad: {props.edad}</p>
        </div>
    )
}
```

Le añadimos un botón para que muestre un mensaje cada vez que hagamos click.

Persona.js
```
function Persona(props) {
    function felisitatu(){
        window.alert("sorihonac");
    }
    return (
        <div>
            <p>Nombre: {props.nombre}</p>
            <p>Edad: {props.edad}</p>
            <button onClick={felisitatu} > Cumplir años</button>
        </div>
    )
}
```

Añadimos un estado al componente Persona y hacemos que la función **felisitatu** sume 1 a la edad que tenemos en el estado.

Persona.js
```
function Persona(props) {
   let [edad, setEdad] = useState(5);

   function felisitatu(){
      setEdad(edad+1);
   }
   return (
      <div>
         <p>Nombre: {props.nombre}</p>
         <p>Edad: {props.edad}</p>
         <button onClick={felisitatu} > Cumplir años</button>
      </div>
   )
}
```

SI hacemos click en el botón veremos que la página no se actualiza, pero si miramos las herramientas de desarrollador veremos que el estado sí se está modificando. Esto ocurre porque en el HTML estamos mostrando directamente el valor que recibimos por props.

Para que se actualice automáticamente, haremos dos cosas: por una parte, pondremos como valor inicial del estado el valor que recibimos por props y, por otra, en el **return** tendremos que poner que queremos mostrar el valor **edad** que tenemos guardado en el estado. El componente quedaría así:

Persona.js
```
function Persona(props) {
   let [edad, setEdad] = useState(props.edad);

   function felisitatu(){
      setEdad(edad+1);
   }
   return (
      <div>
         <p>Nombre: {props.nombre}</p>
         <p>Edad: {edad}</p>
         <button onClick={felisitatu} > Cumplir años</button>
      </div>
   )
}
```