An Array is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. **Arrays** are commonly used in computer programs to organize data so that a related set of values can be easily sorted or searched.

Iteration is the repetition of a process in order to generate a sequence of outcomes.

Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.

The **map()** function is used to iterate over an array and manipulate or change data items.

To render multiple JSX elements in React, you can loop through an array with the .map() method and return a single element.

```
function ReptileListItems() {
  const reptiles = ["alligator", "snake", "lizard"];

  return reptiles.map((reptile) => <li>{reptile}</li>);
}
```

However, if you look at the console, you will see a warning that each child in an array or iterator should have a unique key.

Adding a unique key prevents React from having to re-render the entire component each time there is an update.

To resolve the warning:

```
function ReptileList() {
  const reptiles = ['alligator', 'snake', 'lizard'];

  return (
    <ol>
      {reptiles.map(reptile => (
        <li key={reptile}>{reptile}</li>
      ))}
    </ol>
  );
}
```

State is a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.

Iterables

An iterable is an object capable of **returning** its **members one by one**. Said in other words, an iterable is anything that you can loop over.

Spread syntax (...) The Spread operator lets you expand an iterable like an object, string, or array into its elements.

The spread operator is very useful when you want to make an exact copy of an existing array, you can use the spread operator to accomplish this quickly.

Add Item To Array Hook

To update the state array with a new element at the end of it, we can pass in a callback that returns a new array with the new item added to the end of it.

1. We have the setArr state setter function defined with the useState hook.

2. We have the onClick handler that calls the setArr function with a callback that takes the oldArray parameter, which is the old value of the arr state.

3. And we return a new array with the existing items spread from oldArray with the new item at the end of the array.

```
export default function App() {

const [arr, setArr] = useState(["foo"]);

return (
    <div className="App">

        <button onClick={() => setArr([...arr, "foo"])}>Add</button>

        <div>
          {arr.map((a, i) => (
              <p key={i}>{a}</p>
            )
          )
          }
        </div>
    </div>
  );
}
```

Controlled Components

Event handlers determine what action is to be taken whenever an event is fired. This could be a button click or a change in a text input.

Essentially, event handlers are what make it possible for users to interact with your React app.

Los eventos de React se nombran usando **camelCase**, en vez de minúsculas.

Con JSX pasas una **función** como el manejador del evento, en vez de un string.

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

Guide Event Handlers

https://blog.logrocket.com/a-guide-to-react-onclick-event-handlers-d411943b14dd/

You can also use **synthetic events** directly inside an `onClick` event handler. In the example below, the button's **value** is gotten via `e.target.value` and then used to alert a message.

```
<button value="Hello!" onClick={(e) => alert(e.target.value)}>
```

Other Example:

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

export default function InputExample() {
const [text, setText] = useState('');

return (
 <div>
  <input
    type="text"
    value={text}
    onChange={e => setText(e.target.value)}
/>
   <h1 style={{color: text }}>{text}</h1>
 </div>
 );
```

};

Table Example

Cuando utilizo **onChange o onClick se crea y llama a una función , y el e.target...(ej: value, name...) es para apuntar de donde deseo tomar la variables.**

```
export default function ListOfThings() {
  const [items, setItems] = useState([]);
  const [itemName, setItemName] = useState("");
  const addItem = event => {
    setItems([...items, { id: items.length, name: itemName }]);
    setItemName("");
  };
  return (
    <>
      <label>
        <input
          name="item"
          type="text"
          value={itemName}
          onChange={e => setItemName(e.target.value)}
        />
      </label>
      <button onClick={addItem} >Insertar</button>
      <table>
       <tbody>
          {items.map(item => (
              <tr key={item.id}><td>{item.name}</td></tr>
            )
          )
          }
        </tbody>
      </table>
    </>
  );
}
```