Handler Function in JSX

Learn to use onchange handlers with a React component's JSX.

We'll cover the following

- Exercises:

The App component still has the input field and label, which we haven't used. In HTML outside of JSX, input fields have an onchange handler. We're going to discover how to use onchange handlers with a React component's JSX. First, refactor the App component form a concise to block body so we can add implementation details.

```
const App = () => {
  // do something in between

  return (
    <div>
      <h1>My Hacker Stories</h1>

      <label htmlFor="search">Search: </label>
      <input id="search" type="text" />

      <hr />

      <List />
    </div>
  );
};
```

src/App.js

Next, define a function – which can be normal or arrow – for the change event of the input field. In React, this function is called an **(event) handler**. Now the function can be passed to the `onChange` attribute (JSX named attribute) of the input field.

```
const App = () => {
  const handleChange = event => {
    console.log(event);
  };

  return (
    <div>
      <h1>My Hacker Stories</h1>

      <label htmlFor="search">Search: </label>
      <input id="search" type="text" onChange={handleChange} />

      <hr />

      <List />
    </div>
  );
};
```

src/App.js

After opening your application in a web browser using the app link in the SPA widget, open the browser's developer tools to see logging occur after you type into the input field. This is called a **synthetic event** defined by a JavaScript object. Through this object, we can access the emitted value of the input field:

```
const App = () => {
  const handleChange = event => {
    console.log(event.target.value);
  };

  return ( ... );
};
```

src/App.js

The synthetic event is essentially a wrapper around the browser's native event, with more functions that are useful to prevent native browser behavior (e.g. refreshing a page after the user clicks a form's submit button). Sometimes you will use the event, sometimes you won't need it.

This is how we give HTML elements in JSX handler functions to respond to user interaction. Always pass functions to these handlers, not the return value of the function, except when the return value is a function:

```
// don't do this
<input
  id="search"
  type="text"
  onChange={handleChange()}
/>

// do this instead
<input
  id="search"
  type="text"
  onChange={handleChange}
```

```
/>
```

src/App.js

The following code demonstrates the execution of the above concepts:

```
  □ □ □□  □   ã□  F   □□  □   □  )□     □   9□  5□  @@  □   °□  n□  □PNG
□
   IHDR   □   □□□   (-□S   äPLTE""""""""""""""""""2PX=r□)7;*:>H□¤-BGE□□8do5Xb6[eK□®K□¯1MU9gs3S
□
   IHDR   □   □□□   ×©ÍÊ  □ePLTE"""""""""""""""""""""""""2RZN¢¹J□«3R[J□¬)59YÁÞ0KS4W`Q«ÄL□²%+-0JR
?^q÷ñíÛ□ï.},□ìsæÝ_TttÔ¾ □1#□□/(ì□-[□□□è`□è`Ì□ÚïÅðZ□d5□□□□?ÎebZ¿Þ□i.Ûæ□□□iqÎ□+1°□}Â□5ù  ïçd
□
   IHDR       □□   D¤□Æ  □APLTE   """""""""""""""""""""""""""""2RZVºÖ_ÔôU·Ñ=r□$()'25]ÎíC□□0LS<o}X
□
   IHDR   @   @□□   □·□ì  □:PLTE   """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
¢ßqÇ8Ù□´□mKË±mÆ¶mÛü·yi!è□ÎªYÏuë ÀÏ_Àï?i÷□ý+ò□□ÄA□|□ù{□□´?¿□_En□).□JËD¤<□
©¬¢Z\Ts©R*□(□  ¯©□J□□□□□u□X/□4J□9□¡5·DEµ4kÇ4□&i¥V4Ú□¡®Ð□□¯□vsf:àg,□¢èBC»î$¶□ºÍùî□□á□@□ô□I_
-ê>Û□º«¢XÕ¢î}ß¨ëÛÑ;□ÃöN´□ØvÅý□Î¸ÿ1 □ë×ÄO@&v/Äþ_□ö\ô□Ç\í.□□½+0□□;□□□!□fÊ□¦´Ó%Â JY·O□Â□'/Å]_□
```

HTML and JavaScript work well together in JSX. JavaScript in HTML can display objects, can pass JavaScript primitives to HTML attributes (e.g. `href` to `<a>`), and can pass functions to an element’s attributes for handling events.

I prefer using arrow functions because of their concision as event handlers, however, in a larger React component I see myself using the function statements too because it gives them more visibility in contrast to other variable declarations within a component’s body.

Exercises:

- Confirm the changes from the last section.
- Read more about React’s events.