Imperative React

We'll cover the following

- Exercises:

React is inherently declarative, starting with JSX and ending with hooks. In JSX, we tell React *what* to render and not *how* to render it. In a React side-effect Hook (useEffect), we express when to achieve *what* instead of *how* to achieve it. Sometimes, however, we'll want to access the rendered elements of JSX imperatively, in cases such as these:

- read/write access to elements via the DOM API:
  - measure (read) an element's width or height
  - setting (write) an input field's focus state
- implementation of more complex animations:
  - setting transitions
  - orchestrating transitions
- integration of third-party libraries:
  - D3 is a popular imperative chart library

Because imperative programming in React is often verbose and counterintuitive, we'll walk only through a small example for setting the focus of an input field imperatively. For the declarative way, simply set the input field's `autofocus` attribute:

```
const InputWithLabel = ({ ... }) => (
  <>
    <label htmlFor={id}>{children}</label>
     
    <input
      id={id}
      type={type}
      value={value}

      autoFocus
      onChange={onInputChange}
    />
```

```
  </>
);
```

src/App.js

This works, but only if one of the reusable components is rendered once. For instance, if the App component renders two InputWithLabel components, only the last rendered component receives the auto-focus on its render. However, since we have a reusable React component here, we can pass a dedicated prop and let the developer decide whether its input field should have an `autofocus` or not:

```
const App = () => {
  ...

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

      <InputWithLabel
        id="search"
        value={searchTerm}
        isFocused

        onInputChange={handleSearch}
      >
        <strong>Search:</strong>
      </InputWithLabel>

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

src/App.js

Using just `isFocused` as an attribute is equivalent to `isFocused={true}`. Within the component, use the new prop for the input field's `autoFocus` attribute:

```
const InputWithLabel = ({
  id,
  value,
  type = 'text',
  onInputChange,

  isFocused,

  children,
}) => (
  <>
    <label htmlFor={id}>{children}</label>
     
    <input
      id={id}
      type={type}
```

```
      value={value}

      autoFocus={isFocused}


      onChange={onInputChange}
    />
  </>
);
```

src/App.js

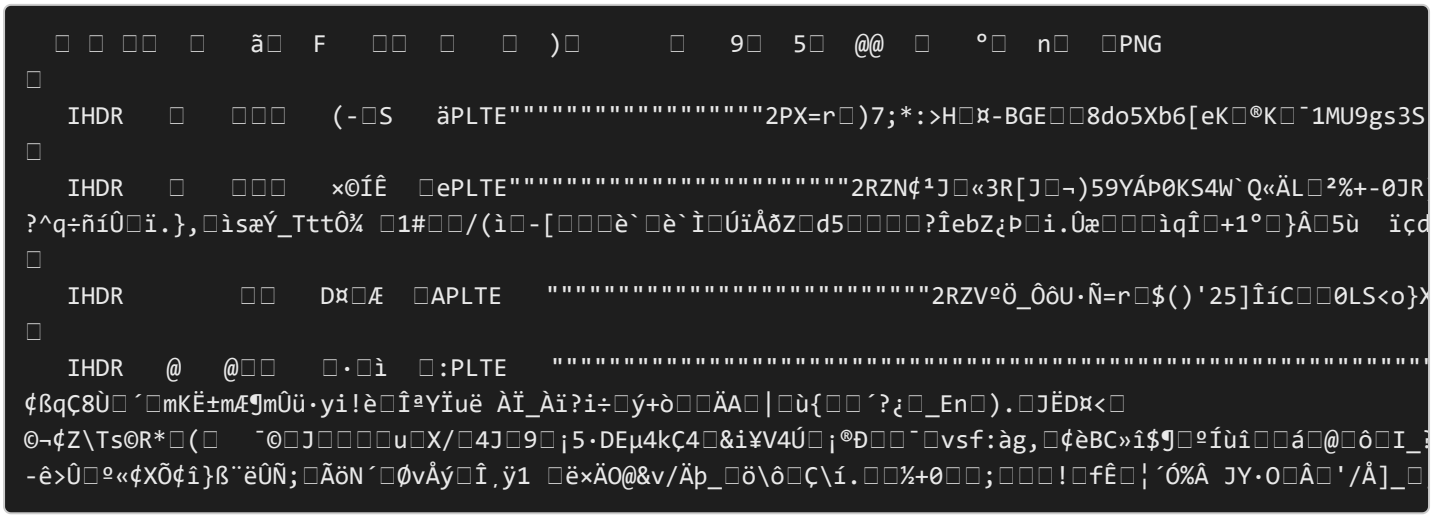
The feature works, yet it's still a declarative implementation. We are telling React *what* to do and not *how* to do it. Even though it's possible to do it with the declarative approach, let's refactor this scenario to an imperative approach. We want to execute the `focus()` method programmatically via the input field's DOM API once it has been rendered:

```
const InputWithLabel = ({
  id,
  value,
  type = 'text',
  onInputChange,
  isFocused,
  children,
}) => {

  // A
  const inputRef = React.useRef();

  // C
  React.useEffect(() => {
    if (isFocused && inputRef.current) {
      // D
      inputRef.current.focus();
    }
  }, [isFocused]);

  return (
    <>
      <label htmlFor={id}>{children}</label>
       

       {/* B */}

      <input
        ref={inputRef}

        id={id}
        type={type}
        value={value}
        onChange={onInputChange}
      />
    </>
  );
};
```

src/App.js

All the essential steps are marked with comments that are explained step by step:

1. (A) First, create a `ref` with **React's useRef hook**. This `ref` object is a persistent value which stays intact over the lifetime of a React component. It comes with a property called `current`, which, in contrast to the `ref` object, can be changed.
2. (B) Second, the `ref` is passed to the input field's JSX-reserved `ref` attribute and the element instance is assigned to the changeable `current` property.
3. (C) Third, opt into React's lifecycle with React's useEffect Hook, performing the focus on the input field when the component renders (or its dependencies change).
4. (D) And fourth, since the `ref` is passed to the input field's `ref` attribute, its `current` property gives access to the element. Execute its focus programmatically as a side-effect, but only if `isFocused` is set and the `current` property is existent.



This was an example of how to move from declarative to imperative programming in React. It's not always possible to go the declarative way, so the imperative approach is necessary. This lesson was for the sake of learning about the DOM API in React.

Exercises:

- Confirm the changes from the last section.
- Read more about React's useRef Hook.