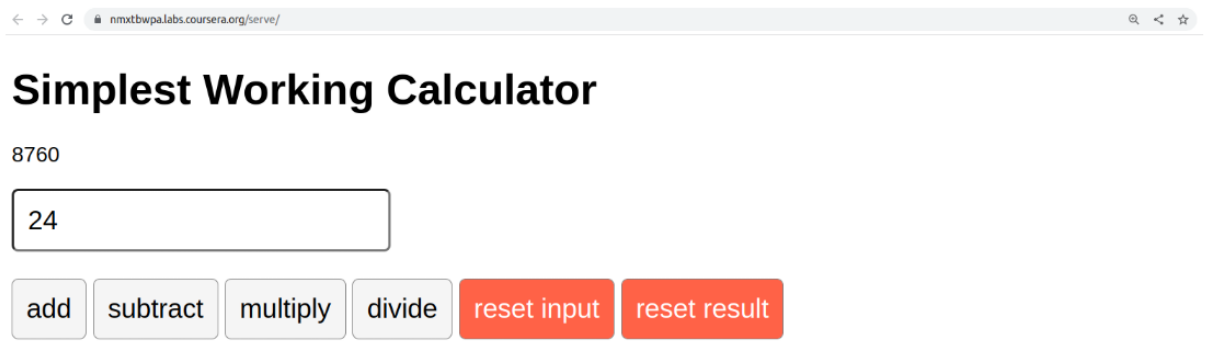
Assignment: Simplest Working Calculator

Date: 3/9/2025

Author: Lorraine Figueroa

This Coursera Assignment (the end assignment for the React Basics course) is to build a simple working calculator:

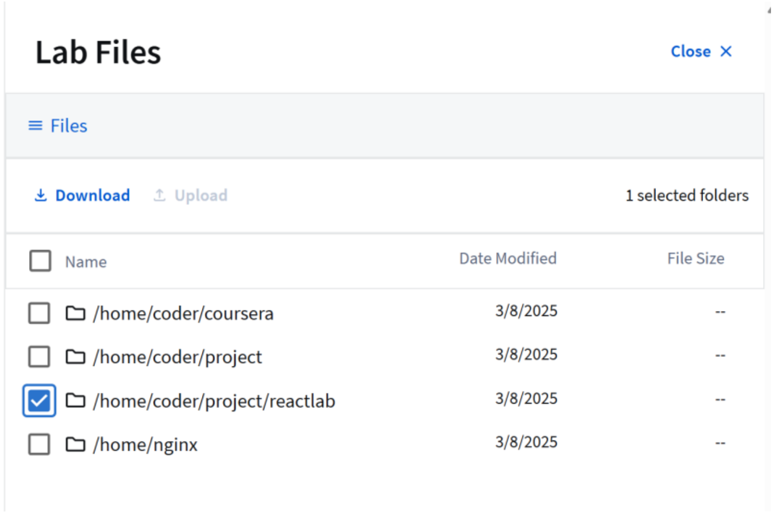

nmxtbwpa.labs.coursera.org/serve/
Simplest Working Calculator
8760
24
add
subtract
multiply
divide
reset input
reset result

Note: These notes include the steps I follow for downloading a lab and getting it to work on my desktop machine.

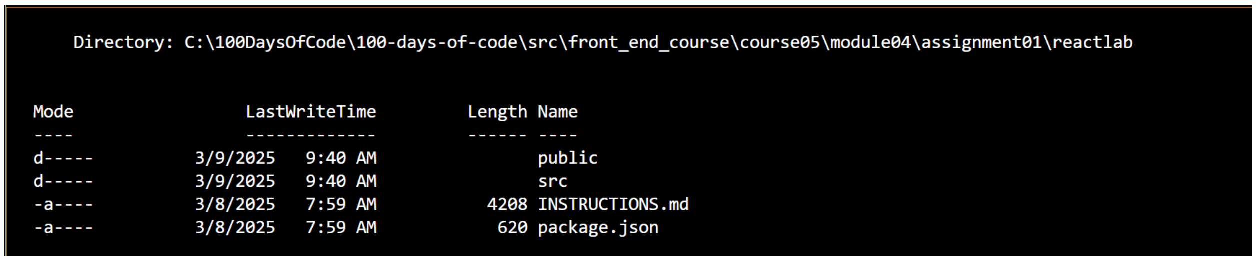
Provided Files

The assignment was already packaged as a React project.

- I downloaded the files under the reactlab folder.



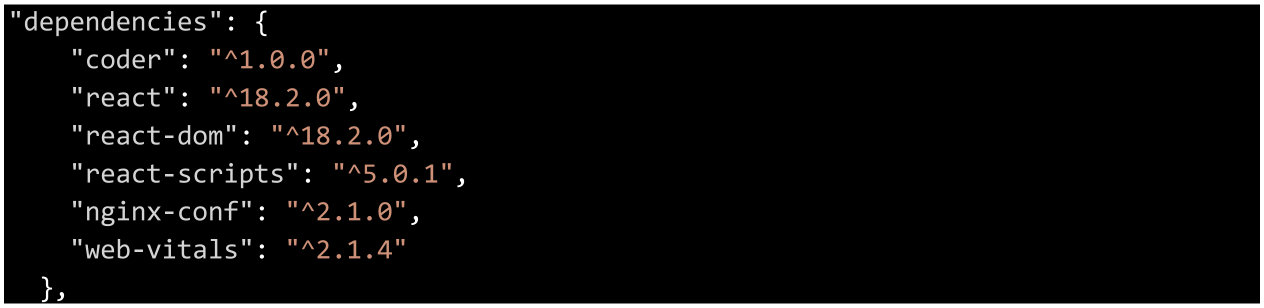
Name	Date Modified	File Size
/home/coder/coursera	3/8/2025	--
/home/coder/project	3/8/2025	--
✓ /home/coder/project/reactlab	3/8/2025	--
/home/nginx	3/8/2025	--



```
    Directory: C:\100DaysOfCode\100-days-of-code\src\front_end_course\course05\module04\assignment01\reactlab


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          3/9/2025   9:40 AM                public
d-----          3/9/2025   9:40 AM                src
-a----          3/8/2025   7:59 AM           4208 INSTRUCTIONS.md
-a----          3/8/2025   7:59 AM            620 package.json
```

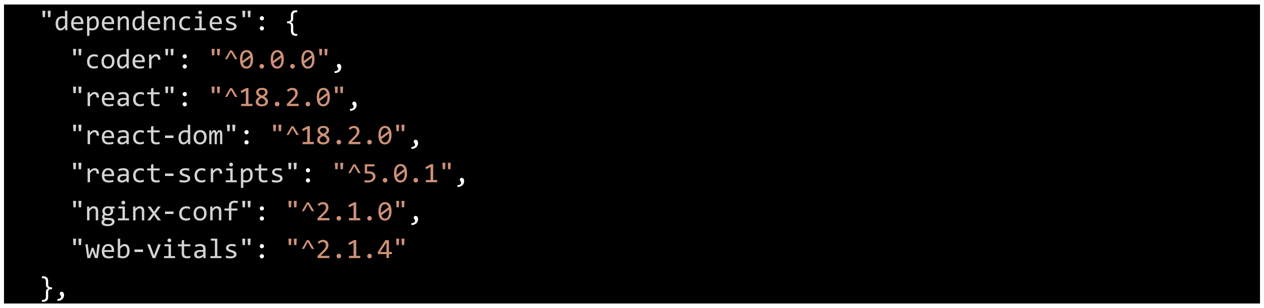
- I edited the `package.json` to change the version number for `coder`

FROM:



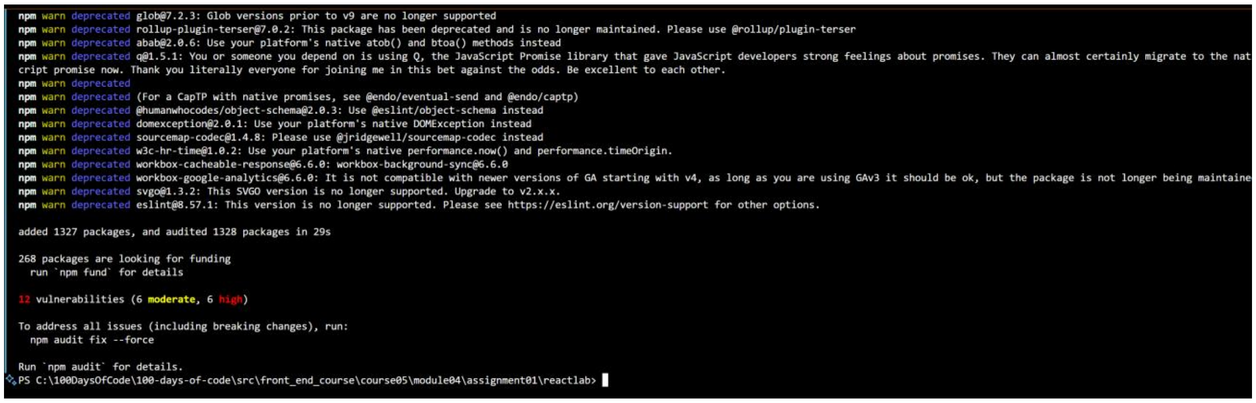
```
"dependencies": {
    "coder": "^1.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "^5.0.1",
    "nginx-conf": "^2.1.0",
    "web-vitals": "^2.1.4"
  },
```

TO:



```
  "dependencies": {
    "coder": "^0.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "^5.0.1",
    "nginx-conf": "^2.1.0",
    "web-vitals": "^2.1.4"
  },
```

- Navigated to `reactlab` folder for this project in a Terminal and entered: **`npm install`**

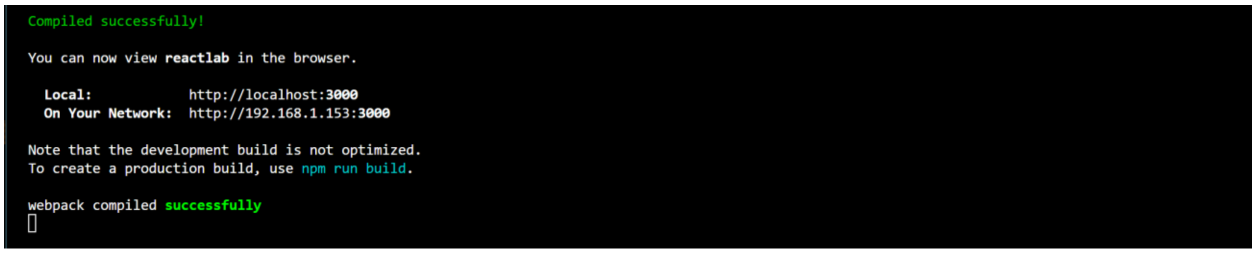


```
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated rollup-plugin-terser@7.0.2: This package has been deprecated and is no longer maintained. Please use @rollup/plugin-terser
npm warn deprecated abab@2.0.6: Use your platform's native atob() and btoa() methods instead
npm warn deprecated q@1.5.1: You or someone you depend on is using Q, the JavaScript Promise library that gave JavaScript developers strong feelings about promises. They can almost certainly migrate to the nat
cript promise now. Thank you literally everyone for joining me in this bet against the odds. Be excellent to each other.
npm warn deprecated
npm warn deprecated (For a CapTP with native promises, see @endo/eventual-send and @endo/captp)
npm warn deprecated @humanwhocodes/object-schema@2.0.3: Use @eslint/object-schema instead
npm warn deprecated domexception@2.0.1: Use your platform's native DOMException instead
npm warn deprecated sourcemap-codec@1.4.8: Please use @jridgewell/sourcemap-codec instead
npm warn deprecated w3c-hr-time@1.0.2: Use your platform's native performance.now() and performance.timeOrigin.
npm warn deprecated workbox-cacheable-response@6.6.0: workbox-background-sync@6.6.0
npm warn deprecated workbox-google-analytics@6.6.0: It is not compatible with newer versions of GA starting with v4, as long as you are using GAv3 it should be ok, but the package is not longer being maintaine
npm warn deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
npm warn deprecated eslint@8.57.1: This version is no longer supported. Please see https://eslint.org/version-support for other options.

added 1327 packages, and audited 1328 packages in 29s

268 packages are looking for funding
  run `npm fund` for details

12 vulnerabilities (6 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\100DaysOfCode\100-days-of-code\src\front_end_course\course05\module04\assignment01\reactlab>
```

Everything seems to have worked out but for the vulnerabilities.

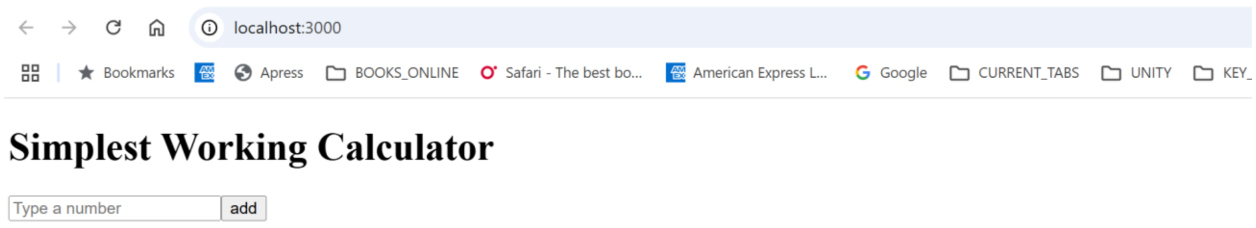
- Enter: npm start

I wanted to see what an untouched version of the project looks like.



```
Compiled successfully!

You can now view reactlab in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.153:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

SCREEN:



localhost:3000

Bookmarks Apress BOOKS_ONLINE Safari - The best bo... American Express L... Google CURRENT_TABS UNITY KEY_

Simplest Working Calculator

Instructions

Task

For this assessment, you will be building a calculator in React. You are provided with code snippets, and your task is to use these, plus any of your code to complete a calculator app that can perform the four basic mathematical operations: addition, subtraction, multiplication, and division.

Before you begin

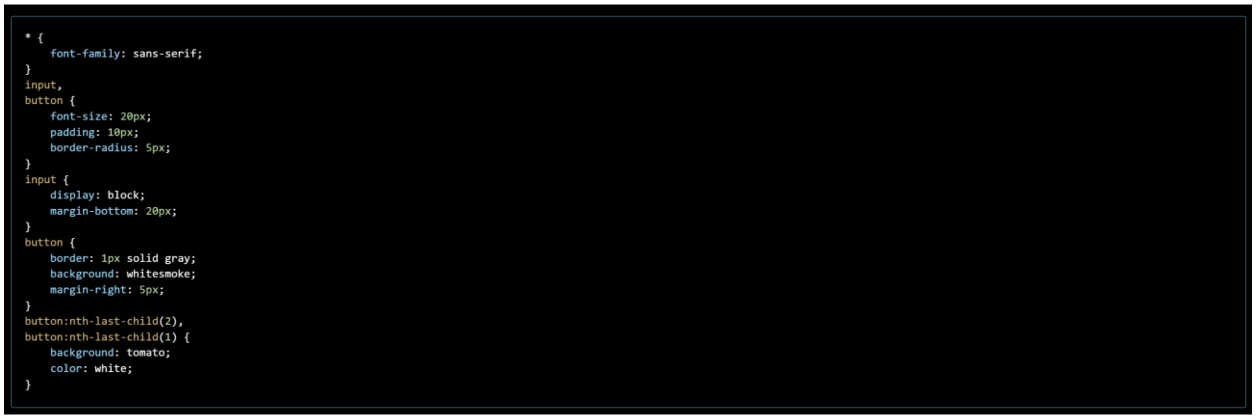
If you execute the `npm start` command before changing any code in this task, you'll get the notification in the code lab's terminal that reads *"webpack compiled successfully"*.

You are now ready to start working on your app. Follow the comments in the code! They are your instructions on what you need to do to make this app work. For example, once you've added the imports, you should be able to serve the app. It still needs work, but at least you'll have the app showing in the browser.

Step 1.

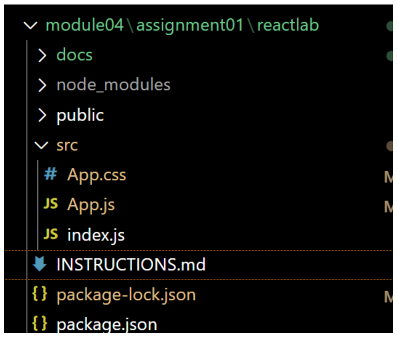
In this assessment, your goal is to build a simple calculator app.

The app should be fully functional. However, since you do not have the `App.css` file in the root folder, you need to add it, with the following code:



```css
* {
    font-family: sans-serif;
}
input,
button {
    font-size: 20px;
    padding: 10px;
    border-radius: 5px;
}
input {
    display: block;
    margin-bottom: 20px;
}
button {
    border: 1px solid gray;
    background: whitesmoke;
    margin-right: 5px;
}
button:nth-last-child(2),
button:nth-last-child(1) {
    background: tomato;
    color: white;
}
```

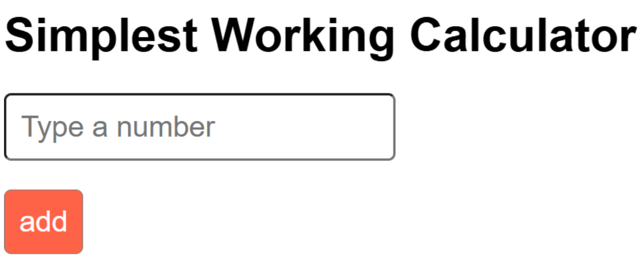
Note: The file `App.css` already was in the src folder!



```
module04 \ assignment01 \ reactlab
  docs
  node_modules
  public
  src
    App.css        M
    App.js         M
    index.js
  INSTRUCTIONS.md
  package-lock.json   M
  package.json
```

- Add the css code provided to the file `App.css`

SCREEN:



The CSS code makes the last two buttons in the DOM stand out in tomato color with white text. Our “add” button has the same look because there is only one button at this time.

Step 2. Here is the app's starting code:

App.js:

```
import {
  useState,
  useRef
} from "react";
import "./App.css";

function App() {
  const inputRef = useRef(null);
  const resultRef = useRef(null);
  const [result, setResult] = useState(0);

  function plus(e) {
    e.preventDefault();
    setResult((result) => result + Number(inputRef.current.value));
  };

  function minus(e) {
    // Add the code for the minus function
  };

  function times(e) {
    // Add the code for the plus function
  };

  function divide(e) {
    // Add the code for the divide function
  };

  function resetInput(e) {
    // Add the code for the resetInput function
  };

  function resetResult(e) {
    // Add the code for the resetResult function
  };

  return (
    <div className="App">
      <div>
        <h1>Simplest Working Calculator</h1>
      </div>
      <form>
        <p ref={resultRef}>
```

```
          {/* add the value of the current total */}
        </p>
        <input
          pattern="[0-9]"
          ref={inputRef}
          type="number"
          placeholder="Type a number"
        />
        <button onClick={plus}>add</button>
        {/* Add the subtract button */}
        {/* Add the multiply button */}
        {/* Add the divide button */}
        {/* Add the resetInput button */}
        {/* Add the resetResult button */}
      </form>
    </div>
  );
}

export default App;
```

Your goal is to use this starting code and extend it with missing pieces to make the app.

- Addition (plus): This function is already provided. It adds the value from the input field to the current result.
- Subtraction (minus): Implement this function to subtract the input value from the result.
- Multiplication (times): Implement this function to multiply the result by the input value.
- Division (divide): Implement this function to divide the result by the input value. Make sure to handle division by zero by showing an alert.
- Reset Input (resetInput): This function will reset the input field when the user clicks the reset button.
- Reset Result (resetResult): This function will reset the displayed result to 0.

Step 3.

- At the top of the lab environment, locate the Terminal menu. Click on it to open a dropdown, then select New Terminal. Use the npm start command to start the development server.

- Ensure that you have the necessary dependencies installed (like React, ReactDOM, and React scripts).
- You can now view the App in your browser by navigating to localhost:3000. To view the output, click on the Browser Preview icon located on the left panel. It is the last icon in the panel.
- In your browser, enter: http://localhost:3000 to see the output.a fully working simple calculator app.

Discussion/Observations to complete App.js

The import of `useState` and `useRef` indicates that we will be managing state data.

useState

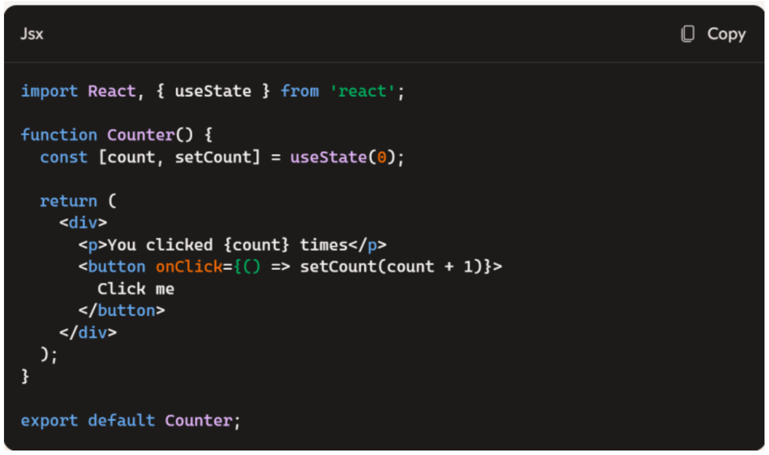
The `useState`[1] hook is fundamental to managing state in React functional components. Here's a brief rundown of how it works:

1. **Initialization**:
   - When you call `useState`, you pass an initial state value.
   - Example: `const [count, setCount] = useState(0);` initializes count to 0.
2. **State Variable**:
   - `count` is a state variable that holds the current value.
   - You can use this variable directly in your component to render state-dependent UI.
3. **Updater Function**:
   - `setCount` is a function that lets you update the state.
   - When you call `setCount(newValue)`, React schedules a re-render of the component with the new state.
4. **Reactivity**:

[1] I used Microsoft co-pilot for all package summaries

- On state change, React automatically re-renders the component, reflecting the updated state in the UI.

Here’s a simple example:



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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```

useState in our App.js

```
  // Holds the current result of a calculation
  const [result, setResult] = useState(0);
```

It appears to hold the current result of the calculation. I can infer this via the `plus()` function:

```

  function plus(e) {
    e.preventDefault();
    setResult((result) => result + Number(inputRef.current.value));
  };
```

When the user presses the “add” button the `plus()` function is invoked. The `plus()` function takes the current value of the result and adds to it the value entered by the user. (see `useRef` section)

```
<button onClick={plus}>add</button>
```

useRef

The `useRef` hook in React is a handy tool for managing ***mutable references*** and accessing DOM elements directly in functional components. Here's a summary of how it works:

1. **Initialization**:
    - When you call `useRef`, you pass an initial value (often null for DOM elements).

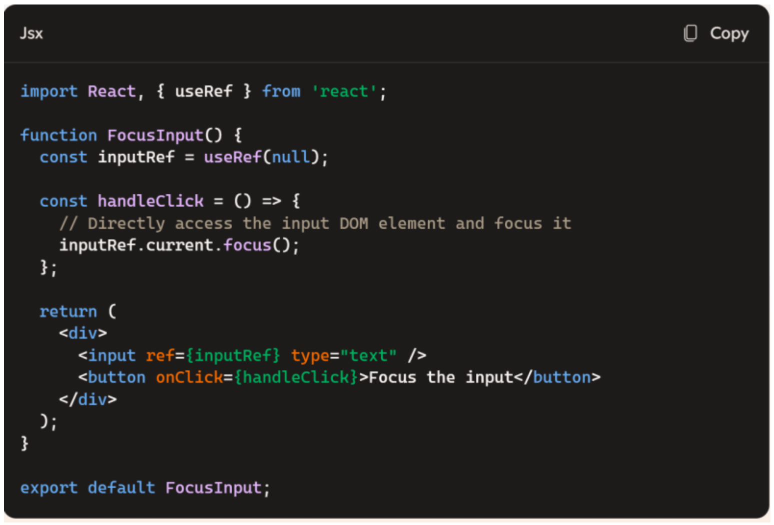
   - Example: `const inputRef = useRef(null);`

2. **Reference Object**:
   - `useRef` returns a mutable ref object with a `.current` property.
   - The current property can hold any value (e.g., a DOM element or a value that you want to persist).

3. **DOM Access**:
   - You can assign the ref object to a DOM element's ref attribute.
   - This provides direct access to the DOM element for reading and modifying its properties or calling methods.

4. **Mutable Value**:
   - Unlike state variables, changes to the ref object don't trigger re-renders.
   - This makes `useRef` useful for storing values that need to persist across renders without causing re-rendering.

Here's a simple example:



```jsx
import React, { useRef } from 'react';

function FocusInput() {
  const inputRef = useRef(null);

  const handleClick = () => {
    // Directly access the input DOM element and focus it
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={handleClick}>Focus the input</button>
    </div>
  );
}

export default FocusInput;
```

In this example:

- The `useRef` hook creates a ref object (`inputRef`).
- The input element's ref attribute is set to `inputRef`, ***giving direct access to the DOM element***.

- Clicking the button triggers the `handleClick` function, which calls `inputRef.current.focus()` to focus the input element.

With `useRef`, you have a powerful way to interact with DOM elements and maintain mutable values without causing unnecessary re-renders.

How is useRef being used in our calculator?

In this calculator app the `useRef` is used to "hook" to DOM elements:

```
  const inputRef = useRef(null);
```

The above `useRef` is associated to changes to the input text:

```
        <input
          pattern="[0-9]"
          ref={inputRef}
          type="number"
          placeholder="Type a number"
        />
```

The `inputRef` above is used when the add button is pressed in the `plus()` function:

```
  function plus(e) {
    e.preventDefault();
    setResult((result) => result + Number(inputRef.current.value));
  };
```

The other `useRef` variable:

```
  const resultRef = useRef(null);
```

The `resultRef` is associated to a "TBD" section showing the result of the calculation.

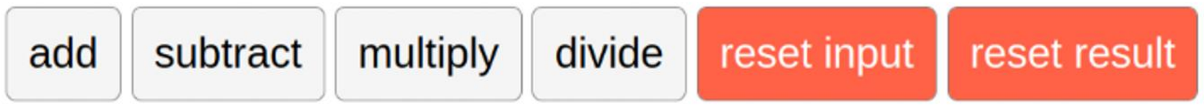
```
       <p ref={resultRef}>
          {/* add the value of the current total */}
        </p>
```

We need to add code to update this section if and when the result changes.

Completing the project

I will only work on the complete implementation of the plus/add functionality. But, since I need to reset the result and clear the input I will implement those two buttons first.

The buttons:



resetInput button

Added:

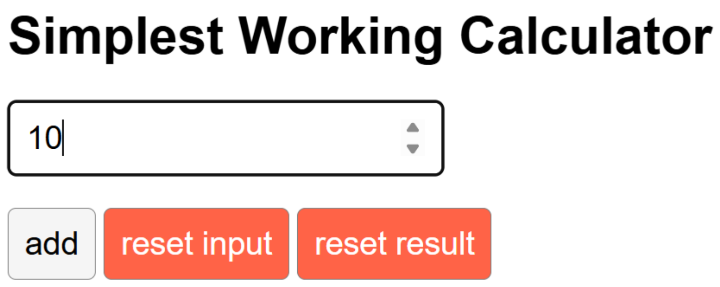
```
 <button onClick={resetInput}>reset input</button>
```

Updated the resetInput() function:

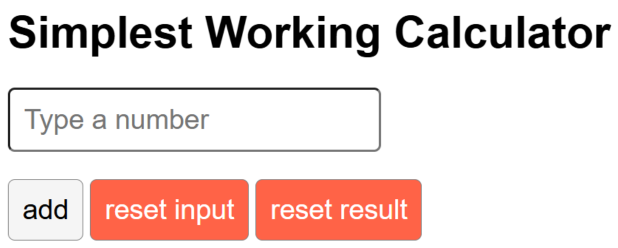
```
  function resetInput(e) {
    // Clear the input field
    inputRef.current.value = "";
  };
```

Test:

1. Enter a number in the input field



2. Click on “reset input”

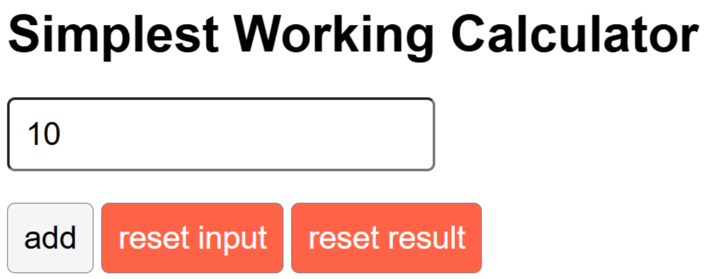


It works as expected

Viewing the results

Let’s do a test by performing the following steps:

1. Enter 10
2. Click on Add
3. Enter 10



Things are a bit awkward! At the end of step 2 the input field should clear so I can enter the next value.

Update the plus() function to clear the input. I will invoke the “reset input” to do the job.

```
function plus(e) {
  e.preventDefault();
  setResult((result) => result + Number(inputRef.current.value));
  resetInput(e);
};
```

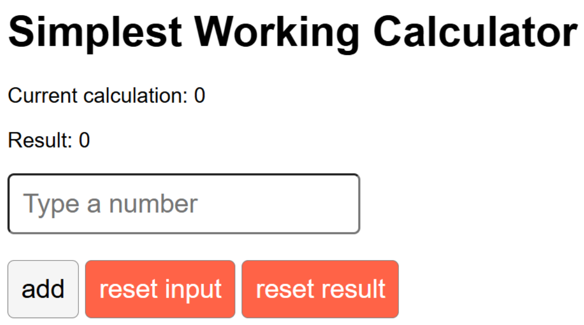
The problem is I don’t see the current calculation or the current result and don’t know how to “signal” – go ahead I am done with the inputting the second operand. I will assume that clicking on an operation is equivalent to hitting the “=” sign. That is we will perform a calculation.

Results:

```
        <p ref={resultRef}>
          Result: {result}
        </p>
```

I will also have a string section that will be used to view the details of the current result, e.g. “0 + 20” will be displayed and the Result section will show 20. If the user enters 30 and clicks on “add” the result field will display 50 and the calc field will show 0 + 20 → 20 + 30

This is how the screen will start:



Simplest Working Calculator

Current calculation: 0

Result: 0

If I type in a number, e.g. 10, I want to see the current calculation as 0 + 10

or <previous result> <operation clicked> <input number>

and the result as 10.

Adding a “current calculation” section:

I added a new <p> element to display the current calculation (e.g. “0 + 10” or “10 + 20”). It will always show the current calculation as <previous result> <operation> < input number>. The result field will show the result of actually calculating the above.

```
      <form>
        <p ref={currentCalculationRef}>
          Current calculation: {currentCalculation}
        </p>
        <p ref={resultRef}>
          Result: {result}
        </p>
        <input
          pattern="[0-9]"
          ref={inputRef}
          type="number"
          placeholder="Type a number"
        />
        <button onClick={plus}>add</button>
        {/* Add the subtract button */}
        {/* Add the multiply button */}
        {/* Add the divide button */}
        <button onClick={resetInput}>reset input</button>
        <button onClick={resetResult}>reset result</button>
        {/* Add the resetResult button */}
      </form>
```

I created a new function:

```
  function doOp(e, operator) {
```

That all the operation buttons shall call. Therefore I updated the plus() function to:

```
  function plus(e) {

    doOp(e, "+");

  };
```

doOp Function

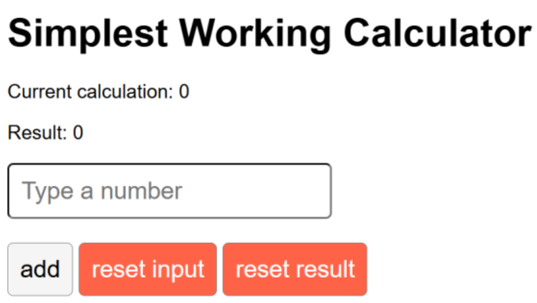
```
function doOp(e, operator) {
    e.preventDefault(); // disable the default behavior of the form

    // recalculate the currentCalculation
    let currentCalculation = result + " " + operator + " " +
inputRef.current.value;

    // set the value in order to be displayed
    setCurrentCalculation(currentCalculation); // update currentCalculation

    let currentNumber = Number(inputRef.current.value); // value entered
    let currentResult = result; // keep track of currentResult
    let newResult = 0;
    switch (operator) {
      case "+":
        newResult = currentResult + currentNumber;
        break;
      case "-":
        newResult = currentResult - currentNumber;
        break;
      case "*":
        newResult = currentResult * currentNumber;
        break;
      case "/":
        newResult = currentResult / currentNumber;
        break;
      default:
        newResult = currentNumber;
    }
    setResult(newResult);
    resetInput(e);
  }
```

Handling only add:

```
  return (
    <div className="App">
      <div>
        <h1>Simplest Working Calculator</h1>
      </div>
      <form>
        <p>
          Current calculation: {currentCalculation}
        </p>
        <p ref={resultRef}>
          Result: {result}
        </p>
        <input
          pattern="[0-9]"
          ref={inputRef}
          type="number"
          placeholder="Type a number"
        />
        <button onClick={plus}>add</button>
        {/* Add the subtract button */}
        {/* Add the multiply button */}
        {/* Add the divide button */}
        <button onClick={resetInput}>reset input</button>
        <button onClick={resetResult}>reset result</button>
        {/* Add the resetResult button */}
      </form>
    </div>
  );
```

TEST

Start Screen:

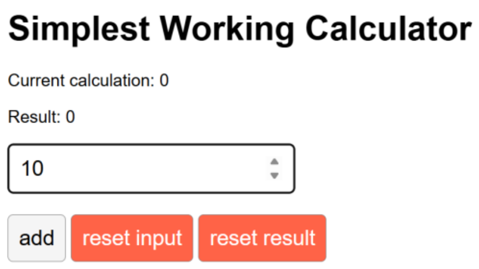


Simplest Working Calculator

Current calculation: 0

Result: 0

1. Enter 10

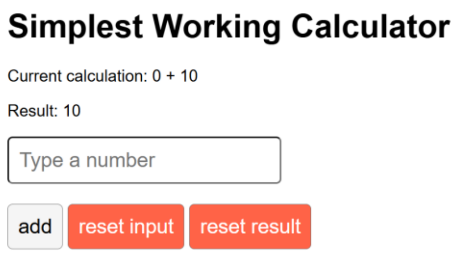


Simplest Working Calculator

Current calculation: 0

Result: 0

2. Press add

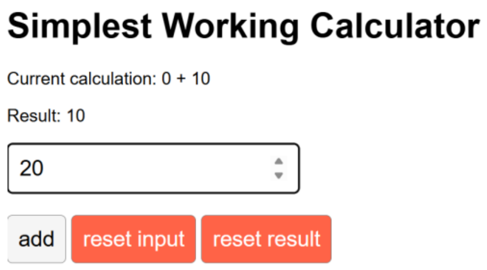


Simplest Working Calculator

Current calculation: 0 + 10

Result: 10

3. Enter 20

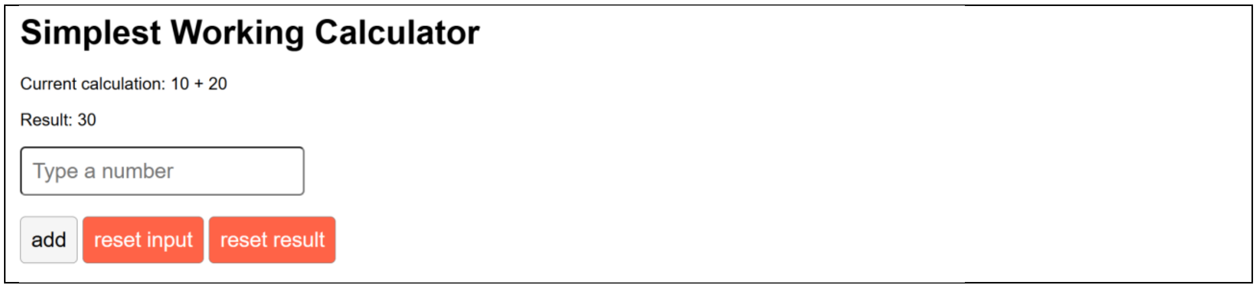


Simplest Working Calculator

Current calculation: 0 + 10

Result: 10

4. Press add



Two problems:

- The one thing I do not like is having to move back to the input field in order to enter the next number. Task: After pressing an operation button move the focus back to the input field.
- If I am in the middle of a calculation, pressing the “result input” clears the current result and calculation

Fixing the focus

I will use the example in the course

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

Updating the function resetInput(e) to:

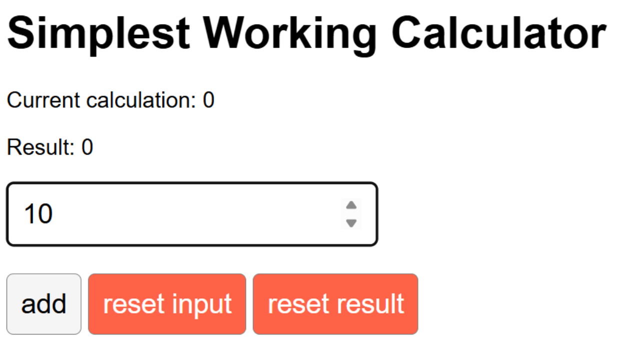
```
  function resetInput(e) {
    // Clear the input field
    inputRef.current.value = "";
    inputRef.current.focus();
  };
```

Fixing reset input

Clicking the resetInput SHOULD not clear the current result or current calculation fields. Why does it?

Here is the issue: Suppose I do the following:

1. Enter: 10

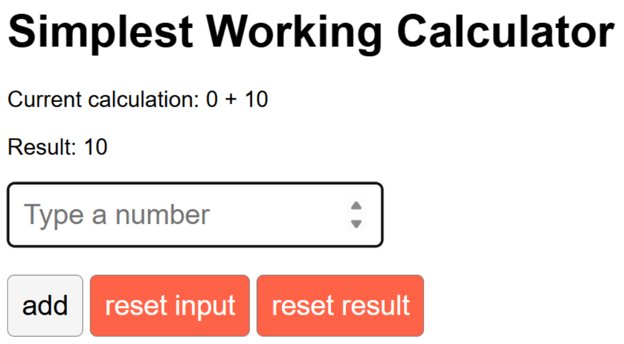


Simplest Working Calculator

Current calculation: 0

Result: 0

2. Click on “add” button (result: 10, current calculation: 0 + 10)

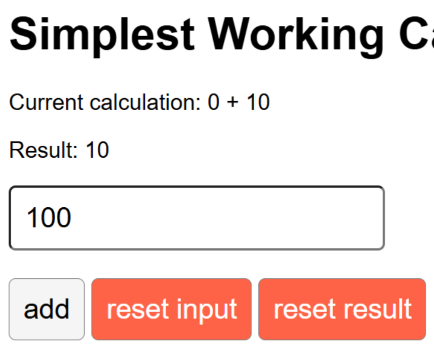


Simplest Working Calculator

Current calculation: 0 + 10

Result: 10

3. Enter 100

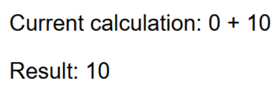


Simplest Working Calculator

Current calculation: 0 + 10

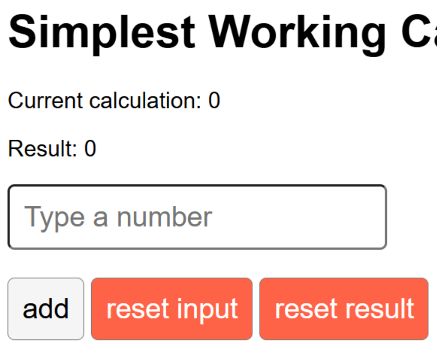
Result: 10

4. I made a mistake I really wanted 10 again, Click on “reset input” . I expect the Current calculation to stay “as is” and the Result to remain 10:



Current calculation: 0 + 10

Result: 10



Simplest Working Calculator

Current calculation: 0

Result: 0

But everything clears as if I clicked on reset result!!!

The solution was added a call to `preventDefault` in the `resetInput` function:

```
function resetInput(e) {
   e.preventDefault();
  // Clear the input field
  inputRef.current.value = "";
  inputRef.current.focus();
};
```

Adding Logic for the other operations:

```
function plus(e) {
   doOp(e, "+");
```

```
  };

  function minus(e) {
    doOp(e, "-");
  };

  function times(e) {
    doOp(e, "*");
  };

  function divide(e) {
    doOp(e, "/");
  };

  function resetInput(e) {
    e.preventDefault();
   // Clear the input field
   inputRef.current.value = "";
   inputRef.current.focus();
 };
```

Adding the buttons for the other operators:

```
        <button onClick={minus}>subtract</button>
        <button onClick={times}>multiply</button>
        <button onClick={divide}>divide</button>
```

Testing…

Grading Criteria

When interacting with the calculator app in the UGL or VS Code:

- Did each button have a function?
- Did each button contain the required mathematical operator?
- Did each calculation provide the correct result as its output?
- Can you suggest any improvements for the calculator app?

You'll also need to give feedback on and grade the assignments of two other learners using the same criteria.

Final App.js

```
import {
  useState,
  useRef
} from "react";
import "./App.css";

function App() {
  // Refs to the input and result elements
  const inputRef = useRef(null); // hooked to input element
  const resultRef = useRef(null); // hooked to p element

  // Holds the current result of a calculation
  const [result, setResult] = useState(0);
  const [currentCalculation, setCurrentCalculation] = useState("0");

  function plus(e) {
    doOp(e, "+");
  };

  function plus(e) {
    doOp(e, "+");
  };

  function minus(e) {
    doOp(e, "-");
  };

  function times(e) {
    doOp(e, "*");
  };

  function divide(e) {
    doOp(e, "/");
  };

  function resetInput(e) {
    e.preventDefault();
   // Clear the input field
   inputRef.current.value = "";
   inputRef.current.focus();
 };

  function resetInput(e) {
    e.preventDefault();
   // Clear the input field
   inputRef.current.value = "";
   inputRef.current.focus();
 };


  function resetResult(e) {
    // Clear the result field
    setResult(0);
  };

  function doOp(e, operator) {
    console.log("doOp", operator);
    e.preventDefault(); // disable the default behavior of the form

    // recalculate the currentCalculation
    let currentCalculation = result + " " + operator + " " +
inputRef.current.value;

    // set the value in order to be displayed
    setCurrentCalculation(currentCalculation); // update currentCalculation

    let currentNumber = Number(inputRef.current.value); // value entered
    let currentResult = result; // keep track of currentResult
    let newResult = 0;
    switch (operator) {
      case "+":
        newResult = currentResult + currentNumber;
        break;
      case "-":
        newResult = currentResult - currentNumber;
        break;
      case "*":
        newResult = currentResult * currentNumber;
        break;
      case "/":
        newResult = currentResult / currentNumber;
        break;
      default:
        newResult = currentNumber;
    }
    setResult(newResult);
    resetInput(e);
  }
  return (
    <div className="App">
      <div>
        <h1>Simplest Working Calculator</h1>
      </div>
      <form>
        <p>
          Current calculation: {currentCalculation}
        </p>
        <p ref={resultRef}>
          Result: {result}
        </p>
        <input
          pattern="[0-9]"
          ref={inputRef}
          type="number"
          placeholder="Type a number"
        />
        <button onClick={plus}>add</button>
        <button onClick={minus}>subtract</button>
        <button onClick={times}>multiply</button>
        <button onClick={divide}>divide</button>
        <button onClick={resetInput}>reset input</button>
        <button onClick={resetResult}>reset result</button>
      </form>
    </div>
  );
}

export default App;
```