

Participant P07

00:04:13

Chat (window: 0)

> Agent

github.copilot.default

> Context

> Message

how do I change colour?

> Response

Could you clarify what you mean by "change colour"? Are you referring to:

1. Changing the color of text or elements in a web application (e.g., CSS or JavaScript)?
2. Changing the color scheme of your Visual Studio Code editor?
3. Changing the color of terminal text in Linux?
4. Something else?

Let me know so I can assist you better!

00:04:13

- 00:04:14

> Response rendering

00:04:28

Chat (window: 0)

> Agent

github.copilot.vscode

> Context

codebase

> Message

#codebase

> Response

The provided codebase contains a React-based TodoMVC application with features like adding, updating, removing, and toggling todos. It uses Webpack for bundling, Babel for transpiling, and React Router for routing.

00:04:37

- 00:04:37

> Response rendering

00:05:21

> Viewing file (src/index.js)

00:05:22

Participant

So.

00:05:23

> Viewing file (webpack.dev.js)

00:05:29

Participant

OK.

00:05:33

Uh.

00:05:41

Researcher

So what are you thinking right now?

00:05:43

Participant

Oh right, thinking out loud.

00:05:46

So I think JavaScript is a, well, scripting language,

00:05:51

so I don't have to run a compiler.

00:05:56

I guess index dot

00:05:58

> Viewing file (src/index.js)

00:05:58

Participant

JS is the main function or something,

00:06:02

so maybe I can run this?

00:06:07

I guess not.

00:06:08

How do I, let me ask this thing.

00:06:12

I run a JavaScript script.

00:06:13

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/index.js)

> Message

How do I run a javascript script?

> Response

To run a JavaScript script, you can follow these steps depending on the environment:

1. In a Browser

- Include your JavaScript file in an HTML file using a `<script>` tag:

```
<script src="index.js"></script>
```

- Open the HTML file in a browser to execute the script.

2. Using Node.js

- Install Node.js if you haven't already: [Node.js Download](#).
- Run the script in the terminal:

```
node src/index.js
```

3. In a React Application (like your codebase)

- Your code is part of a React app. To run it:

1. Ensure dependencies are installed:

```
npm install
```

2. Start the development server:

```
npm start
```

3. Open the app in your browser at the URL provided by the terminal (usually `http://localhost:3000`).

Let me know if you need further clarification!

00:06:13
- 00:06:16

> Response rendering

00:06:23

Participant

So let's see if this node JS thing is installed, no?

00:06:31

Maybe the IDE can do this?

00:06:39

Java in

00:06:41

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/index.js)

> Message

How to run javascript in vscode?

> Response

To run the JavaScript code in Visual Studio Code, follow these steps:

1. For a React Application (like your codebase):

Since your code is part of a React app, you need to run it using a development server:

1. **Install Dependencies** (if not already done):

```
npm install
```

2. Start the Development Server:

```
npm start
```

3. Open the app in your browser at the URL provided by the terminal (usually `http://localhost:3000`).
-

2. For a Standalone JavaScript File:

If you want to run a standalone JavaScript file (not part of React), you can use Node.js:

1. Open the terminal in VS Code.
2. Run the script using:

```
node src/index.js
```

Let me know if you need further assistance!

00:06:42
- 00:06:44

00:06:42

> Response rendering

Participant

VS Code.

00:06:48

Or I can check buttons. Oh, run and debug.

00:07:03

This takes a really long time.

00:07:10

Uh.

00:07:22

To customize, OK we don't need to customize it.

00:07:29

OK.

00:07:29

I can ask him.

00:07:33

Chat (window: 0)

> Agent

```
github.copilot.default
```

> Context

```
viewport (src/index.js)
```

> Message

Why does nothing happen if I click run and debug?

> Response

The reason nothing happens when you click "Run and Debug" in Visual Studio Code is that your project is a React application, which requires a development server to run. The "Run and Debug" button is typically used for standalone scripts or applications with a configured debugger, but React apps are not executed directly like standalone JavaScript files.

How to Fix This:

1. Start the React Development Server:

- Open the terminal in VS Code.
- Run the following command to start the development server:

```
npm start
```

- This will launch your React app in the browser (usually at `http://localhost:3000`).

2. Debugging React in VS Code: If you want to debug your React app in VS Code:

- Install the "Debugger for Chrome" or "Debugger for Edge" extension in VS Code.
- Add a `launch.json` file to configure debugging:
 1. Open the Command Palette (`Ctrl+Shift+P` or `Cmd+Shift+P` on macOS).
 2. Search for "Debug: Add Configuration" and select it.
 3. Choose "Chrome" or "Edge" as the environment.
- Start the development server (`npm start`) and then attach the debugger.

Let me know if you need help setting up debugging!

00:07:34
- 00:07:38

> Response rendering

00:07:36

Participant

I can also look in the code to see what is actually going on, so I see a render function.

00:07:39

That probably prints something.

00:07:42

OK, I can click on the JSON.

00:07:59

Yeah.

00:08:03

Oh, there is a.

00:08:11

Npm is installed, so I can try it again on the index scripts but nothing.

00:08:14

Happens.

00:08:22

Also, no manual installed.

00:08:29

So I'm reading what it says on the left hand side now.

00:08:47

You can use the debug terminal.

00:08:50

Oh, there's also a debug.

00:08:54

Oh there's also an output thing here.

00:08:55

I have to select ah, there we go.

00:09:03

I had to select something.

00:09:04

Which is, I guess the interpreter for JavaScript.

00:09:08

It now says an error.

00:09:12

It cannot find.

00:09:14

Blah blah blah tasks.

00:09:16

Is that part of the?

00:09:18

Is this part of the exercise now?

00:09:21

Researcher

00:09:25

No, it's not.

00:09:27

Participant

Yeah.

00:09:27

Researcher

00:09:32 And I can tell you that only because this is not actually one of the practice tasks.
Participant
OK.

00:09:35 Uhm.

00:09:40 So.

00:09:45 This is I think.

00:09:45 **Researcher**
All right, so so 5 minutes have passed.
You're not the only one who didn't figure this out in 5 minutes, by the way.
Since this is not an actual task itself, I can tell you how to do it now.
Or would you like to figure it out yourself?

00:10:02 **Participant**
No, I would like to know.

00:10:09 Mm hmm.

00:10:10 **Researcher**
So in your, uh, if you go back to the terminal instead of the debug console.
You can type npm install.
To install the dependencies.

00:10:19 **Participant**
Yeah.

00:10:23 **Researcher**
And after that you can write npm run dev.

00:10:32 **Participant**
Is this OK?

00:10:34 **Researcher**
Yep, that's OK.

00:10:38 Yes. Now if you change your files, you only need to refresh this window to.

00:10:43 Restart to to see your changes.

00:10:52 **Participant**
OK. Thanks.

00:10:54 **Researcher**
Great, so feel free to get started on the
practice tasks.

00:11:08 **Participant**
OK.
So I want to have another terminal so I can grep.
Uh.
OK, maybe I will just close this for now.
So I need to change the color from red to blue.
It would be nice to know what the code is, but I guess that's too hard to look up.
I can first look in the code base for red.
Way too much stuff.

00:11:53 **> Viewing file (src/todo/constants.js)**

00:11:53 **Participant**
There's a constant file here, maybe that has color.

00:11:57 No.

00:12:03 Then there is.

00:12:04 **> Viewing file (src/todo/app.css)**

00:12:05 **Participant**
An app dot CSS.
That has color.
So need to find red here somewhere.
It is a button so maybe this one.
But that only has a background color.
I mean, maybe just yeah, I guess the background color is red.
I can change this to something and see what happens.
No, I guess.
OK.

00:13:06

I guess because I killed that process, it doesn't work.

00:13:15

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/index.js)

> Message

how do I open another terminal in vscode

> Response

To open another terminal in Visual Studio Code, follow these steps:

1. Open the Terminal Panel:

- If the terminal panel is not already open, press `Ctrl+``` (backtick) or go to the menu: `View > Terminal``.

2. Create a New Terminal:

- In the terminal panel, click the "+" button in the top-right corner of the terminal tab.
- Alternatively, use the keyboard shortcut `Ctrl+Shift+``` (backtick).

This will open a new terminal tab, and you can switch between terminals using the dropdown menu in the terminal panel.

00:13:16

- 00:13:17

> Response rendering

00:13:23

Participant

I can open another terminal here.

00:13:40

Let me start this again.

00:13:47

Hmm, doesn't look different.

00:13:53

OK.

00:13:54

I'll.

00:13:57

Type in this thing as I am probably supposed to do.

00:14:00

> Viewing file (src/todo/constants.js)

00:14:00

> Viewing file (src/todo/app.jsx)

00:14:01

Participant

Uh, can look at the app.

00:14:06

> Viewing file (src/todo/constants.js)

00:14:07

Participant

Yeah, but I think it's just it should be somewhere in the GSS, in the CSS file.

00:14:08

00:14:10

> Viewing file (src/todo/app.css)

00:14:13

Participant

And.

00:14:19

Oh, that's nice.

00:14:20

It shows the color. Ah, here that's red.

00:14:24

And it says message. OK, so that's must be it then.

00:14:31

And the task was to change this to.

00:14:35

This value.

00:14:39

> Editing file (src/todo/app.css)

00:14:45

Participant

Unable.

00:14:53

OK.

00:14:56

So that is.

00:14:59

Yeah, I think that's task A.

00:15:03

So then we can go to task B, I would like another.

00:15:08

Terminal so I can look for.

00:15:12

For the placeholder.

00:15:22

And that is in main dot jsx.

00:15:22 > **Viewing file** (src/todo/components/main.jsx) through file search

00:15:26 **Participant**
I can control F here, which is nice.

00:15:31 Yeah.

00:15:34 Maybe it's case sensitive?

00:15:40 So that must be this thing.

00:15:42 And then we can change it, it as of to the.

00:15:46 > **Editing file** (src/todo/components/main.jsx)
- 00:15:55

00:15:49 **Participant**
The.

00:15:50 To the new message, enter a task.

00:15:59 That's annoying.

00:16:01 **Researcher**
Sorry if in the I'm not sure how to do this in Firefox.

00:16:05 > **Editing file** (src/todo/components/main.jsx)

00:16:07 **Researcher**
Can you try to paste something again, then press retry on this pop up?

00:16:12 > **Editing file** (src/todo/components/main.jsx)

00:16:13 **Participant**
Uh.

00:16:13 **Researcher**
In the the error pop up in VS Code.

00:16:18 **Participant**
Retry.

00:16:20 **Researcher**
It should ask your permission for reading your clipboard.

00:16:25 **Participant**
Yeah. Also, I don't know if like Linux has a weird
clipboard situation. I think there are multiple clipboards,
but it does work right? It did copy.

00:16:33 **Researcher**
OK.

00:16:35 > **Editing file** (src/todo/components/main.jsx)

00:16:35 **Researcher**
Did work OK.

00:16:37 **Participant**
It's just that when I typed it started to suggest something and not actually type it,
type what I wanted.

00:16:43 **Researcher**
OK.

00:16:46 So if you feel this, this distracts you from your regular.

00:16:55 Strategy of solving please please let me know and we can find a solution for this.

00:16:57 **Participant**
Mm hmm.

00:17:00 Yeah, maybe I could just do it in [unintelligible] that that
is installed here as well.

00:17:03 But it looks like that did.

00:17:06 That task B.

00:17:09 **Researcher**
OK.

00:17:11 **Participant**
And now?

00:17:12 **Researcher**
Actually, I'll, I'll.

00:17:14 Stop you there as well, since you actually changed a different text

00:17:16 **Participant**
Mm hmm.

00:17:19 **Researcher**
than we asked.

00:17:25 **Participant**
Oh, I see.
> **Editing file** (src/todo/components/main.jsx)

00:17:35 **Participant**
I was supposed to change what needs to be done.
And that lives in header dot JSX.
> **Viewing file** (src/todo/components/header.jsx) through file search

00:17:52 **Participant**
OK.
I'm going to do that in vi as well.
Components, header.
> **Editing file** (src/todo/components/header.jsx)

00:18:01 **Participant**
And then change it to.
Task here.
Like this?

00:18:20 **Researcher**
That's perfect. Thanks.

00:18:21 **Participant**
Thank you.
Then C, I have to sort some stuff.
Uhm.
OK, so I can.
Add some things here.
The quick 123 brown fox.
And then it has to be sorted alphabetically.
So.
> **Viewing file** (src/todo/reducer.js)

00:18:59 **Participant**
Let's first look for the word sort. If that occurs anywhere in the code base.
No.
> **Viewing file** (src/todo/components/item.jsx)

00:19:09 **Participant**
I can't look to where I just was.
I also don't know what JSX is. Maybe I'll start.

00:19:13 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
viewport (src/todo/components/item.jsx)
> **Message**
What is jsx?
> **Response**
JSX (JavaScript XML) is a syntax extension for JavaScript commonly used with React to describe what the UI should look like. It allows you to write HTML-like code directly within JavaScript, making it easier to define the structure and appearance of components.

Key Features of JSX:

1. HTML-like Syntax:

- JSX looks similar to HTML but is written within JavaScript files.
- Example:

```
const element = <h1>Hello, world!</h1>;
```

2. Embedding JavaScript:

- You can embed JavaScript expressions inside JSX using curly braces {}.
- Example:

```
const name = "John";
const element = <h1>Hello, {name}!</h1>;
```

3. Transpilation:

- JSX is not valid JavaScript, so it needs to be transpiled (e.g., by Babel) into standard JavaScript before it can run in the browser.
- Example Transpilation:

```
const element = <h1>Hello, world!</h1>;
// Transpiles to:
const element = React.createElement("h1", null, "Hello,
world!");
```

4. Component Rendering:

- JSX is used to define React components and their structure.
- Example:

```
function App() {
  return <div>Hello, React!</div>;
}
```

In your code, JSX is used to define the structure of the [Item](#) ([src/todo/components/item.jsx](#)) component, including conditional rendering ([isWritable ? ... : ... \(src/todo/components/item.jsx\)](#)) and event handling (e.g., [onDoubleClick \(node_modules/@types/react/index.d.ts\)](#), [onChange \(node_modules/@types/react/index.d.ts\)](#)).

00:19:15
- 00:19:25

> Response rendering

00:19:29

Participant

So it's a kind of HTML apparently.

00:19:41

> **Viewing file** (src/todo/components/input.jsx)

00:19:42

Participant

OK, I see input file.

00:19:44

That is probably responsible for putting the input here.

00:19:53

If you press enter.

00:19:56

It.

00:19:58

Does something here.

00:20:01

And then is that that minimum 2 characters maybe?

00:20:08

Yeah. If I just have one character, nothing happens. So.

00:20:12

Probably that is that.

00:20:16

00:20:23

00:20:26

And then where does it go?

So it says callback. I don't know what that means.

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

viewport (src/todo/components/input.jsx)

> **Message**

What is usecallback?

> **Response**

[useCallback\(node_modules/@types/react/index.d.ts\)](#) is a React Hook that is used to memoize a function. It ensures that the same function instance is returned across renders unless its dependencies change. This is particularly useful for optimizing performance in React components by preventing unnecessary re-creations of functions, which can help avoid unnecessary re-renders of child components.

Syntax:

```
const memoizedCallback = useCallback(  
  () => {  
    // Your function logic here  
  },  
  [dependencies] // Array of dependencies  
);
```

Key Points:

1. Purpose:

- [useCallback\(node_modules/@types/react/index.d.ts\)](#) is used to memoize a function so that it doesn't get recreated on every render unless its dependencies change.

2. Dependencies:

- The second argument is an array of dependencies. The memoized function will only be recreated if one of these dependencies changes.

3. Performance Optimization:

- It is often used when passing functions as props to child components to prevent unnecessary re-renders.

Example:

```
import React, { useState, useCallback } from "react";  
  
function Counter() {  
  const [count, setCount] = useState(0);  
  
  const increment = useCallback(() => {  
    setCount((prevCount) => prevCount + 1);  
  }, []); // No dependencies, so the function is memoized forever  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
      <button onClick={increment}>Increment</button>  
    </div>  
  );  
}
```

In Your Code:

In [src/todo/components/input.jsx](#), [useCallback](#) ([node_modules/@types/react/index.d.ts](#)) is used to memoize the [handleBlur](#) ([src/todo/components/input.jsx](#)) and [handleKeyDown](#) ([src/todo/components/input.jsx](#)) functions: - [handleBlur](#) ([src/todo/components/input.jsx](#)):

```
const handleBlur = useCallback(() => {  
  if (onBlur) onBlur();  
}, [onBlur]);
```

- This ensures that [handleBlur](#) ([src/todo/components/input.jsx](#)) is not recreated unless the [onBlur](#) ([src/todo/components/input.jsx](#)) prop changes.

- [handleKeyDown](#) ([src/todo/components/input.jsx](#)):

```
const handleKeyDown = useCallback(  
  (e) => {  
    if (e.key === "Enter") {  
      const value = e.target.value.trim();  
      if (!hasValidMin(value, 2)) return;  
      onSubmit(value);  
      e.target.value = "";  
    }  
  },  
  [onSubmit]  
);
```

- This ensures that [handleKeyDown](#) ([src/todo/components/input.jsx](#)) is not recreated unless the [onSubmit](#) ([src/todo/components/input.jsx](#)) prop changes. This helps optimize the performance of the [Input](#) ([src/todo/components/input.jsx](#)) component by avoiding unnecessary re-renders or re-creation of these functions.

00:20:30
00:20:31
- 00:20:41
00:20:34

Participant

Maybe a function pointer or something?

> Response rendering

Participant

I have no idea what any of this syntax means.

OK.

So performance stuff.

> Viewing file (src/todo/components/item.jsx)

Participant

Item dot jsx.

So there should be an array of input somewhere.

> Viewing file (src/todo/components/input.jsx)

> Viewing file (src/todo/components/footer.jsx)

> Viewing file (src/todo/reducer.js)

Participant

OK, there's add item here.

That this just looks like it depatches.

Functions maybe?

So there's a switch case on the type of the action which can be updating items and removing items, etcetera.

And then it makes a new state.

> Viewing file (src/todo/constants.js)

Participant

OK.

So I can maybe look for.

Add item to see.

Where that is used.

Hmm.

> Viewing file (src/todo/reducer.js)

Participant

Now there's some.

A random string.

I have no idea why it would do that, generate a random.

> Viewing file (src/todo/components/input.jsx)

Participant

Thing.

Input. It doesn't seem to be doing anything.

> Viewing file (src/todo/components/footer.jsx)

Participant

Yeah, I want to look up where I can find an

00:20:49
00:20:49
00:20:54
00:20:56
00:21:01
00:21:06
00:21:07
00:21:15
00:21:29
00:22:03
00:22:08
00:22:10
00:22:15
00:22:18
00:22:23
00:22:25
00:22:29
00:22:32
00:22:34
00:22:41
00:22:44
00:22:52
00:22:56
00:22:58
00:23:02
00:23:04
00:23:05
00:23:06
00:23:16

00:23:19 array type.
00:23:22 > **Viewing file** (src/todo/components/input.jsx)
00:23:22 **Participant**
But I think JavaScript doesn't have types.
00:23:29 Has valid min.
00:23:34 Seems to check if.
00:23:37 The length of a value is at least min.
00:23:40 Oh, that is to check that we have at least 2 characters.
00:23:42 > **Viewing file** (src/todo/components/footer.jsx)
00:23:49 **Participant**
No idea what that is.
00:23:53 > **Viewing file** (src/index.js)
00:23:57 > **Viewing file** (src/todo/reducer.js)
00:23:59 **Participant**
OK, I can maybe look for the button.
00:24:03 And.
00:24:11 So in header.
00:24:14 Now if the key is enter, that is in input JSX.
00:24:16 > **Viewing file** (src/todo/components/input.jsx) through file search
00:24:21 **Participant**
Then it does something.
00:24:25 Yeah. So where does this value go is the
00:24:27 question.
00:24:32 On submit. OK, so where is that function?
00:24:45 Hmm.
00:24:56 This is, how do you define
00:24:59 a function.
00:25:02 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
selection (src/todo/components/input.jsx)
> **Message**
How do you define a function in javascript?
> **Response**
In JavaScript, you can define a function in several ways. Here are the most common methods:

1. Function Declaration

- A named function defined using the function keyword.

```
function greet(name) {  
  return Hello, </span><span class="si">${</span><span  
class="nx">name</span><span class="si">}</span><span class="sb">!;  
}
```

2. Function Expression

- A function assigned to a variable.

```
const greet = function(name) {  
  return Hello, </span><span class="si">${</span><span  
class="nx">name</span><span class="si">}</span><span class="sb">!;  
};
```

3. Arrow Function

- A concise syntax introduced in ES6.

```
const greet = (name) => {
  return Hello, </span><span class="si">${</span><span
class="nx">name</span><span class="si">}</span><span class="sb">!;
};
// For single expressions, you can omit the braces and return:
const greet = (name) => Hello, </span><span class="si">${</span><span
class="nx">name</span><span class="si">}</span><span class="sb">!;
```

4. Anonymous Function

- A function without a name, often used as a callback.

```
setTimeout(function() {
  console.log("This is an anonymous function.");
}, 1000);
```

5. Immediately Invoked Function Expression (IIFE)

- A function that runs immediately after being defined.

```
(function() {
  console.log("IIFE executed!");
})();
```

Example in Your Code:

In your [src/todo/components/input.jsx](#) file, the [onSubmit](#) ([src/todo/components/input.jsx](#)) prop is expected to be a function. It is passed as an argument to the [Input](#) ([src/todo/components/input.jsx](#)) component and invoked in the [handleKeyDown](#) ([src/todo/components/input.jsx](#)) callback:

```
onSubmit(value);
```

This function is likely defined in a parent component and passed down as a prop.

00:25:03
- 00:25:08

in the JavaScript.

> Response rendering

00:25:11

Participant

OK.

00:25:12

That's, yeah, but there's no function on submit

00:25:15

anywhere in the code base.

00:25:18

You can assign a function to a variable.

00:25:23

Oh, these arrow things are functions.

00:25:27

So.

00:25:31

Now I still don't see where it's defined.

00:25:36

Is.

00:25:38

In the standard library of JavaScript?

00:25:40

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/components/input.jsx)

> Message

is onSubmit in the standard library of javascript?

> Response

No, [onSubmit \(src/todo/components/input.jsx\)](#) is not part of the standard JavaScript library. It is a custom prop passed to the [Input \(src/todo/components/input.jsx\)](#) component in your React application.

In the context of React, [onSubmit \(src/todo/components/input.jsx\)](#) is typically a function provided by a parent component. It is used as a callback to handle the submission of data, such as when the user presses "Enter" in an input field. The parent component defines the behavior of [onSubmit \(src/todo/components/input.jsx\)](#) and passes it down to the [Input \(src/todo/components/input.jsx\)](#) component as a prop.

For example:

```
function ParentComponent() {  
  const handleSubmit = (value) => {  
    console.log("Submitted value:", value);  
  };  
  
  return <Input onSubmit={handleSubmit} placeholder="Enter a task" />;  
}
```

In this case, `handleSubmit` is passed as the [onSubmit \(src/todo/components/input.jsx\)](#) prop to the [Input \(src/todo/components/input.jsx\)](#) component, and it gets called when the user presses "Enter" in the input field.

00:25:42
- 00:25:45

> Response rendering

00:25:52

Participant

This custom prop.

00:25:55

OK.

00:25:57

Oh, it can actually look at.

00:26:00

What the code base does, cool.

00:26:05

So.

00:26:09

Researcher

Yeah, about that.

00:26:09

Participant

OK so.

00:26:12

Researcher

At the top of this file, there's the instruction on on Copilot, and it also tells you about what code it can and can't see.

00:26:16

00:26:22

Participant

00:26:24

Ah, this thing.

Researcher

Yes.

00:26:24

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/components/input.jsx)

> Message

Where is onSubmit defined?

> Response

The [onSubmit \(src/todo/components/input.jsx\)](#) function is not defined in the [src/todo/components/input.jsx](#) file itself. It is passed as a prop to the [Input \(src/todo/components/input.jsx\)](#) component from a parent component. To locate its definition, you need to check where the [Input \(src/todo/components/input.jsx\)](#) component is used in your codebase.

Steps to Find [onSubmit \(src/todo/components/input.jsx\)](#):

1. **Search for [Input \(src/todo/components/input.jsx\)](#) Usage:** Look for where the [Input \(src/todo/components/input.jsx\)](#) component is imported and used in your project. For example:

```
import { Input } from "../input";
```

2. **Check the Parent Component:** In the parent component where [Input \(src/todo/components/input.jsx\)](#) is used, the [onSubmit \(src/todo/components/input.jsx\)](#) prop will be passed. For example:

```
<Input onSubmit={handleSubmit} placeholder="Enter a task" />
```

3. **Locate the handleSubmit Function:** The function passed to [onSubmit \(src/todo/components/input.jsx\)](#) (e.g., handleSubmit) will be defined in the parent component. It will contain the logic for handling the submitted value.

Example:

If [Input \(src/todo/components/input.jsx\)](#) is used in a parent component like this:

```
<Input onSubmit={(value) => console.log("Submitted:", value)} />
```

Then the [onSubmit \(src/todo/components/input.jsx\)](#) function is defined inline as an arrow function.

Alternatively, it might look like this:

```
function ParentComponent() {
  const handleSubmit = (value) => {
    console.log("Submitted:", value);
  };

  return <Input onSubmit={handleSubmit} />;
}
```

Here, handleSubmit is defined in the parent component and passed as the [onSubmit \(src/todo/components/input.jsx\)](#) prop.

You can search for [Input \(src/todo/components/input.jsx\)](#) in your codebase to find where it is used and locate the [onSubmit \(src/todo/components/input.jsx\)](#) definition.

00:26:27

Participant

I think I think I.

00:26:28
- 00:26:37

> Response rendering

00:26:30

Participant

Turned this on.

00:26:32

Researcher

So that's something you include in every command.

00:26:35

Participant

Oh, OK.

00:26:35

Researcher

If you want it to do that.

00:26:41

Chat (window: 0)

> Agent

github.copilot.default

> Context

codebase

> Message

#codebase where is onSubmit defined?

> Response

In your codebase, the [onSubmit\(src/todo/components/input.jsx\)](#) prop is defined in two places where the [Input\(src/todo/components/input.jsx\)](#) component is used:

1. In the [Header\(src/todo/components/header.jsx\)](#) Component (header.jsx):

```
<Input onSubmit={addItem} label="New Todo Input" placeholder="Enter a task here" />
```

- Here, the [onSubmit\(src/todo/components/input.jsx\)](#) prop is assigned the [addItem\(src/todo/components/header.jsx\)](#) function.
- The [addItem\(src/todo/components/header.jsx\)](#) function is defined as:

```
const addItem = useCallback((title) => dispatch({ type: ADD_ITEM, payload: { title } })), [dispatch]);
```

- This function dispatches an action to add a new todo item.

2. In the [Item\(src/todo/components/item.jsx\)](#) Component (item.jsx):

```
<Input onSubmit={handleUpdate} label="Edit Todo Input" defaultValue={title} onBlur={handleBlur} />
```

- Here, the [onSubmit\(src/todo/components/input.jsx\)](#) prop is assigned the [handleUpdate\(src/todo/components/item.jsx\)](#) function.
- The [handleUpdate\(src/todo/components/item.jsx\)](#) function is defined as:

```
const handleUpdate = useCallback(
  (title) => {
    if (title.length === 0)
      removeItem(id);
    else
      updateItem(id, title);

    setIsWritable(false);
  },
  [id, removeItem, updateItem]
);
```

- This function updates or removes a todo item based on the input value.

These are the two places where [onSubmit\(src/todo/components/input.jsx\)](#) is defined and passed to the [Input\(src/todo/components/input.jsx\)](#) component.

00:26:44
- 00:26:49

> Response rendering

00:26:54

> Viewing file (src/todo/components/header.jsx)

00:26:54

Participant

OK.

00:26:54 So it says it's defines in header dot JSX.
00:26:59 Maybe it is literally just.
00:27:02 This.
00:27:05 Equal sign.
00:27:10 And then add item is.
00:27:13 A callback to.
00:27:15 OK, so on submit.
00:27:19 Has like a bunch of indirection.
00:27:27 And then it uses this.
00:27:33 And where did I see this? Uh.
00:27:36 Add item or something, and add item function.
00:27:44 That this looks like it is actually doing something instead of just.
00:27:49 Adding layers of indirection. And this handle, can I click on this? Ah, very nice.
00:27:51 > **Viewing file** (src/todo/components/item.jsx) through Copilot link

00:27:56 **Participant**

Yes, so remove item and update item.

00:27:58 That I guess.

00:28:01 Adds things to an array.

00:28:16 Title is.

00:28:26 It says set is writable false, but there's no actual set being passed.

00:28:32 So maybe this is a global variable?

00:28:39 No.

00:29:03 OK.

00:29:03 Maybe I can ask Copilot where this array lives.

00:29:11 In what.

00:29:13 Are the items stored.

00:29:15 **Chat** (window: 0)

> **Agent**

github.copilot.default

> **Context**

codebase

> **Message**

#codebase In what variable are the items stored?

> **Response**

In your codebase, the items (todos) are stored in the [todos \(src/todo/app.jsx\)](#) variable, which is managed by the [useReducer \(node_modules/@types/react/index.d.ts\)](#) hook in the App component (src/todo/app.jsx):

```
const [todos, dispatch] = useReducer(todoReducer, []);
```

- [todos \(src/todo/app.jsx\)](#): This is the state variable that holds the list of todo items.
- [todoReducer \(src/todo/reducer.js\)](#): This is the reducer function that updates the [todos \(src/todo/app.jsx\)](#) state based on dispatched actions (e.g., [ADDITEM \(src/todo/constants.js\)](#), [REMOVEITEM \(src/todo/constants.js\)](#), etc.).
- [dispatch \(src/todo/app.jsx\)](#): This is the function used to send actions to the reducer to modify the [todos \(src/todo/app.jsx\)](#) state.

The [todos \(src/todo/app.jsx\)](#) variable is then passed as a prop to various components, such as: - Header: For adding new items. - Main: For displaying and managing the list of items. - Footer: For filtering and clearing items.

00:29:17 **Participant**

Play.

00:29:19 - 00:29:21 > **Response rendering**

00:29:29 **Participant**

OK so.

00:29:30 > **Viewing file** (src/todo/app.jsx) through Copilot link

00:29:32 **Participant**

This variable.

00:29:37 And then.
00:29:39 I don't know what a hook is.
00:29:40 > **Viewing file** (node_modules/%40types/react/index.d.ts) through Copilot link
00:29:43 **Participant**
But I guess I can click on this.
00:30:25 OK.
00:30:29 So.
00:30:36 I kind of would like to know if I can generate a call.
00:30:42 A step trace so I can see at what point I need to insert the.
00:30:50 Insert the sort function.
00:30:57 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
viewport
(/home/codespace/.cache/typescript/5.8/node_modules/@types/react/index.d.ts)
> **Message**
How do I generate a stacktrace for javascript application?
> **Response**
To generate a stack trace for a JavaScript application, you can use the following methods:

1. Using `console.trace()`

- This method prints a stack trace to the console.
- Example:

```
function exampleFunction() {  
    console.trace("Stack trace:");  
}  
  
exampleFunction();
```

- Output will show the function call hierarchy in the browser's or Node.js console.

2. Using [Error \(node_modules/typescript/lib/lib.es2022.error.d.ts\)](#) Object

- Create an [Error \(node_modules/typescript/lib/lib.es2022.error.d.ts\)](#) object and access its stack property.
- Example:

```
function exampleFunction() {  
    const error = new Error("Stack trace:");  
    console.log(error.stack);  
}  
  
exampleFunction();
```

- This provides a detailed stack trace.

3. Using Debugging Tools

- In **browsers**:
 - Open Developer Tools (F12 or Ctrl+Shift+I).

- Set breakpoints in the "Sources" tab to inspect the call stack.
 - In **Node.js**:
 - Use the `--inspect` flag to debug:

```
node --inspect app.js
```
 - Open `chrome://inspect` in Chrome to debug.
-

4. Using a Logging Library

- Use libraries like [Winston](#) or [Log4js](#) to log stack traces programmatically.
-

5. Manually Throwing an Error

- Throw an error to capture the stack trace:

```
try {  
  throw new Error("Manual stack trace");  
} catch (error) {  
  console.log(error.stack);  
}
```

Let me know if you need help implementing this!

00:31:00
00:31:02
- 00:31:08
00:31:15
00:31:20
00:31:23
00:31:27
00:31:27
00:31:30
00:31:35
00:31:43
00:31:52
00:32:01
00:32:04
00:32:08
00:32:14
00:32:16
00:32:17
00:32:18
00:32:21
00:32:28
00:32:37
00:32:42
00:32:44
00:32:48
00:32:57
00:32:57
00:33:01

Participant

Or maybe you actually cannot do this because it's not compiled.

> Response rendering

Participant

OK, so you have to manually add this.

That is too much work.

Umm.

OK.

So this file.

Is probably part of some kind of library.

Because it's really large.

What else is?

You can pass true or false as a boolean or string.

OK, that's.

An interesting choice.

Oh, they do have types.

Hmm.

So this seems, OK.

Well, I guess it's not really relevant.

> Viewing file (src/todo/components/main.jsx)

Participant

Uhm.

So where were we this.

Uhm.

> Viewing file (src/todo/components/item.jsx)

Participant

So the item.

Where's the array?

OK.

Let me ask again.

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/components/item.jsx)

> Message

Where are the items stored?

> Response

The items (todos) are stored in the todos state, which is managed by the useReducer hook in the App component (src/todo/app.jsx). The todos state is passed down to the [Item \(src/todo/components/item.jsx\)](#) component via props.

Key Details:

1. todos State:

- The todos array holds all the todo items.

- It is managed by the `todoReducer` function, which updates the state based on dispatched actions (e.g., [TOGGLEITEM\(src/todo/constants.js\)](#), [REMOVEITEM\(src/todo/constants.js\)](#), [UPDATE_ITEM\(src/todo/constants.js\)](#)).

Example from App:

```
const [todos, dispatch] = useReducer(todoReducer, []);
```

2. Passing todos to [Item\(src/todo/components/item.jsx\)](#):

- Each todo item is passed as a prop (`todo(src/todo/components/item.jsx)`) to the [Item\(src/todo/components/item.jsx\)](#) component.
- The [dispatch\(src/todo/components/item.jsx\)](#) function is also passed to allow the [Item\(src/todo/components/item.jsx\)](#) component to update the state.

Example:

```
todos.map((todo) => <Item key={todo.id} todo={todo} dispatch={dispatch} />);
```

3. Structure of a Todo Item:

- Each todo item is an object with properties like [id\(src/todo/components/item.jsx\)](#), [title\(src/todo/components/item.jsx\)](#), and [completed\(src/todo/components/item.jsx\)](#).

Example:

```
{
  id: 1,
  title: "Learn React",
  completed: false
}
```

In summary, the items are stored in the `todos` state in the `App` component and passed down to the [Item\(src/todo/components/item.jsx\)](#) component for rendering and interaction.

00:33:02
- 00:33:06

> Response rendering

00:33:13

Participant

And now it says something else. A todo state.

00:33:30

OK.

00:33:30

So I'm reading this paragraph.

00:33:33

But I don't know what a state is.

00:33:35

I don't know what a hook is.

00:33:36

I don't know what I think a component is.

00:33:38

Then probably a file?

00:33:41

I don't know what a prop is.

00:33:46

I don't really want to know what all of this is.

00:33:48

I just want the array.

00:34:15

If I comment this out.

00:34:19

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/components/item.jsx)

> Message

Javascript comment

> Response

```
// The `Item` component represents a single todo item in the list.
// It uses React's `useState` and `useCallback` hooks for managing state
and memoizing functions.

// `isWritable` determines whether the item is in edit mode.
// `setIsWritable` is used to toggle between view and edit modes.

// The `todo` object contains the item's data: `title`, `completed`, and
`id`.
// The `dispatch` function is used to update the global state via actions
like TOGGLE_ITEM, REMOVE_ITEM, and UPDATE_ITEM.

// `toggleItem`: Dispatches an action to toggle the completed status of
the item.
// `removeItem`: Dispatches an action to remove the item from the list.
// `updateItem`: Dispatches an action to update the item's title.

// `handleDoubleClick`: Enables edit mode when the user double-clicks the
item.
// `handleBlur`: Exits edit mode when the input loses focus.
// `handleUpdate`: Updates the item's title or removes it if the title is
empty.

// The component conditionally renders an input field for editing or a
view mode with a checkbox, label, and delete button.
```

00:34:20
- 00:34:24

> Response rendering

00:34:26

Participant

OK, just like C.

00:34:30

So let's just double check if this is

00:34:33

actually the part where.

00:34:38

Where items are headed.

00:34:48

I guess not because.

00:34:51

I have to save first maybe.

00:34:57

Yes. Now, there's nothing happening anymore.

00:34:59

So this is where we update things.

00:35:03

Hmm.

00:35:15

And then.

00:35:21

So I guess we.

00:35:24

Could just immediately here insert it in the correct position.

00:35:31

It's only a few things, so it doesn't have to be efficient.

00:35:39

So.

00:35:43

Then we have to know what update item does.

00:35:46

Chat (window: 0)

> Agent

github.copilot.default

> Context

codebase

> Message

#codebase updateItem

> Response

The [updateItem \(src/todo/components/item.jsx\)](#) function in the [Item \(src/todo/components/item.jsx\)](#) component dispatches an action of type [UPDATE_ITEM \(src/todo/constants.js\)](#) to update the title of a todo item. This action is handled in the [todoReducer \(src/todo/reducer.js\)](#) function in reducer.js.

[updateItem \(src/todo/components/item.jsx\)](#) in [Item \(src/todo/components/item.jsx\)](#) Component:

```
const updateItem = useCallback((id, title) => dispatch({ type: UPDATE_ITEM, payload: { id, title } })), [dispatch]);
```

- **Purpose:** Dispatches an action to update the title of a todo item.
 - **Payload:** Contains the [id \(node_modules/@types/react/index.d.ts\)](#) of the item to update and the new [title \(src/todo/components/item.jsx\)](#).
-

[UPDATE_ITEM \(src/todo/constants.js\)](#) in [todoReducer \(src/todo/reducer.js\)](#):

In `reducer.js`, the [UPDATE_ITEM \(src/todo/constants.js\)](#) action updates the title of the specified todo item:

```
case UPDATE_ITEM:
  newState = state.map((todo) =>
    todo.id === action.payload.id ? { ...todo, title:
action.payload.title } : todo
  );
  break;
```

- **Logic:**
 - Iterates over the [state \(src/todo/reducer.js\)](#) (array of todos).
 - If the [id \(node_modules/@types/react/index.d.ts\)](#) matches the [payload.id \(src/todo/components/item.jsx\)](#), it updates the [title \(src/todo/components/item.jsx\)](#) with [payload.title \(src/todo/components/item.jsx\)](#).
 - Otherwise, the todo remains unchanged.
-

Example Flow:

1. **User Interaction:**
 - The user edits a todo item's title in the [Input \(src/todo/components/input.jsx\)](#) field and submits it.
2. **Dispatch Action:**
 - The [updateItem \(src/todo/components/item.jsx\)](#) function is called, dispatching the [UPDATE_ITEM \(src/todo/constants.js\)](#) action.
3. **Reducer Update:**
 - The [todoReducer \(src/todo/reducer.js\)](#) updates the [todos \(src/todo/app.jsx\)](#) state with the new title for the specified item.
4. **Re-render:**
 - The updated [todos \(src/todo/app.jsx\)](#) state triggers a re-render of the UI.

Let me know if you need further clarification!

> Response rendering

Participant

Can I click on this?

Which is also a callback which.

OK.

So item that should be.

Defined here actually.

Oh it's here, here so it calls

another function.

Update.

Item I guess. Why?

I don't understand why this is so complicated.

00:35:49
- 00:35:58

00:35:52

00:36:00

00:36:08

00:36:08

00:36:11

00:36:13

00:36:15

00:36:18

00:36:23

00:36:32

00:36:34 Why not just define the function here.
00:36:45 OK.
00:36:45 So I guess we have to know.
00:36:49 Where this update item is being used?
00:36:59 So it's just the string update item.
00:37:07 In reducer dot JS.
00:37:11 There is some.
00:37:13 > **Viewing file** (src/todo/reducer.js) through file search
00:37:14 **Participant**
Check.
00:37:18 State map.
00:38:01 OK.
00:38:02 So I guess I have to look up some of these functions and learn what they
00:38:06 actually mean.
00:38:07 So what is a state in JavaScript?
00:38:10 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
selection (src/todo/reducer.js)
> **Message**
What is a state in javascript?
> **Response**
In JavaScript, particularly in the context of frameworks like React, **state** refers to an object or variable that holds data or information about a component. This data can change over time, usually as a result of user interactions, API calls, or other events, and it determines how the component renders and behaves.

Key Characteristics of State:

1. Dynamic Data:

- State represents dynamic data that can change during the lifecycle of a component.
- For example, a counter's value, form input values, or the list of todos in your app.

2. Local to a Component:

- State is typically local to a component, meaning it is managed and used within that component.
- However, in larger applications, state can also be shared globally using tools like `useReducer`, `Context` API, or state management libraries (e.g., `Redux`).

3. Triggers Re-renders:

- When the state changes, React automatically re-renders the component to reflect the updated data in the UI.

Example of State in React:

Using `useState` to manage state in a functional component:

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

function Counter() {
  const [count, setCount] = useState(0); // `count` is the state,
  `setCount` updates it

  const increment = () => setCount(count + 1);
```

```

    return (
      <div>
        <p>Count: {count}</p>
        <button onClick={increment}>Increment</button>
      </div>
    );
  }
}

```

- **count**: The current state value.
 - **setCount**: A function to update the state.
 - **Re-render**: When setCount is called, the component re-renders with the new count value.
-

State in Your Codebase:

In your [todoReducer \(src/todo/reducer.js\)](#): - The **state** represents the list of todo items ([state \(src/todo/reducer.js\)](#) is an array of todo objects). - Actions like [ADDITEM \(src/todo/constants.js\)](#), [UPDATEITEM \(src/todo/constants.js\)](#), and [REMOVEITEM \(src/todo/constants.js\)](#) modify the state. - *Example*:

```

case UPDATEITEM:
  newState = state.map((todo) =>
    todo.id === action.payload.id ? { ...todo, title:
action.payload.title } : todo
  );
  break;

```

- This updates the title of a specific todo item in the state.

In summary, **state** is a way to manage and track dynamic data in your application, ensuring the UI stays in sync with the underlying data.

00:38:11
- 00:38:19

00:38:46

00:38:52

> Response rendering

Participant

A state is an object or a variable that holds data about components. OK.

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/reducer.js)

> Message

javascript component

> Response

The [UPDATE_ITEM \(src/todo/constants.js\)](#) action in the [todoReducer \(src/todo/reducer.js\)](#) is responsible for updating the title of a specific todo item in the state. Here's how it works in the context of your [todoReducer \(src/todo/reducer.js\)](#):

[UPDATE_ITEM \(src/todo/constants.js\)](#) in [todoReducer \(src/todo/reducer.js\)](#):

```
case UPDATE_ITEM:
  newState = state.map((todo) =>
    todo.id === action.payload.id ? { ...todo, title:
action.payload.title } : todo
  );
  break;
```

Explanation:

1. Action Type:

- The [UPDATE_ITEM \(src/todo/constants.js\)](#) action is dispatched when a todo item's title needs to be updated.

2. Logic:

- The [state \(src/todo/reducer.js\)](#) (an array of todos) is iterated using [map\(\) \(src/todo/reducer.js\)](#).
- For each [todo \(src/todo/reducer.js\)](#):
 - If the [todo.id \(src/todo/reducer.js\)](#) matches the [action.payload.id \(src/todo/reducer.js\)](#), a new object is created with the updated [title \(src/todo/reducer.js\)](#) ([action.payload.title \(src/todo/reducer.js\)](#)).
 - Otherwise, the [todo \(src/todo/reducer.js\)](#) remains unchanged.

3. Result:

- A new array ([newState \(src/todo/reducer.js\)](#)) is created with the updated todo item.

4. State Persistence:

- The updated state is saved to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) using the [saveState \(src/todo/reducer.js\)](#) function:

```
saveState(newState);
```

Example of Dispatching [UPDATE_ITEM \(src/todo/constants.js\)](#):

In the Item component:

```
const updateItem = useCallback((id, title) => dispatch({ type:
UPDATE_ITEM, payload: { id, title } })), [dispatch]);
```

- This function dispatches the [UPDATE_ITEM\(src/todo/constants.js\)](#) action with the [id\(src/todo/reducer.js\)](#) of the todo and the new [title\(src/todo/reducer.js\)](#).
-

Example Flow:

1. User Interaction:

- The user edits a todo item's title and submits it.

2. Dispatch Action:

- The updateItem function is called, dispatching the [UPDATE_ITEM\(src/todo/constants.js\)](#) action.

3. Reducer Update:

- The [todoReducer\(src/todo/reducer.js\)](#) updates the [state\(src/todo/reducer.js\)](#) with the new title for the specified item.

4. State Persistence:

- The updated state is saved to [localStorage\(node_modules/@types/node/globals.d.ts\)](#).

5. Re-render:

- The updated state triggers a re-render of the UI.

This ensures that the todo item's title is updated both in the app's state and in persistent storage.

00:38:54

Participant

Well, let's say component .

00:38:55
- 00:39:04

> Response rendering

00:39:07

Participant

OK.

00:39:11

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/reducer.js)

> Message

javascript what is a component?

> Response

In JavaScript, particularly in frameworks like React, a **component** is a reusable, self-contained piece of code that defines part of the user interface (UI). Components allow you to break down the UI into smaller, manageable, and reusable pieces.

Key Features of a Component:

1. Encapsulation:

- A component encapsulates its structure (HTML/JSX), behavior (JavaScript), and styling (CSS) into a single unit.

2. Reusability:

- Components can be reused across different parts of an application, reducing code duplication.

3. Composability:

- Components can be composed together to build complex UIs. For example, a Header component and a Footer component can be combined into a Page component.

4. State and Props:

- Components can manage their own state (using `useState` or `useReducer`) and receive data from parent components via props.

Types of Components in React:

1. Functional Components:

- These are JavaScript functions that return JSX.
- Example:

```
function Greeting({ name }) {  
  return <h1>Hello, {name}</h1>;  
}
```

2. Class Components (Legacy):

- These are ES6 classes that extend `React.Component`.
- Example:

```
class Greeting extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

```
}  
}
```

Example of a Component:

Here's an example of a functional React component:

```
function Button({ label, onClick }) {  
  return <button onClick={onClick}>{label}</button>;  
}
```

- **Props:** The `label` and `onClick` are passed as props to the `Button` component.
 - **Reusable:** This `Button` component can be reused with different labels and click handlers.
-

Components in Your Codebase:

In your codebase, components like `Item`, `Input`, and `App` are React components. For example: - **Item Component:** - Represents a single todo item. - Manages its own state (`isWritable`) and receives props ([todo\(src/todo/reducer.js\)](#), `dispatch`) to interact with the global state.

```
export const Item = memo(function Item({ todo, dispatch }) {  
  // Component logic and JSX here  
});
```

Components are the building blocks of your React application, allowing you to create a modular and maintainable UI.

00:39:12
- 00:39:19

00:39:21

> Response rendering

Participant

So a component is just.

00:39:25

A module or library or something as you would call it in C.

00:39:29

And then a state.

00:39:35

I'm allowed, allowed to use Google, right?

00:39:39

OK.

00:39:40

So I would probably use Google instead of

00:39:43

Copilot to just Google these general terms, but I guess this works too.

00:39:50

So it holds information about.

00:39:55

A.

00:39:58

Library which is kind of maybe about how you initialize the library then or something.

00:40:03

Because a library should just be code, right?

00:40:06

I guess it's the global variables inside that.

00:40:15

Inside that module.

00:40:19

Yeah. OK. So like.

00:40:21

And you can render things, so that's the, OK.

00:40:25

And here the state. OK, so the todo list is some global variable

00:40:39

and that is the state.

00:40:43

And then you can modify state.

00:40:45

OK, so it kind of tells me here where it adds items.

00:40:48

So what does state dot map mean?

00:40:53

And this is specific to the codebase, so I should have that here.

00:40:58

Chat (window: 0)

00:41:01

> Agent

github.copilot.default

> Context

codebase

> Message

#codebase what is state.map

> Response

In your codebase, [state.map\(src/todo/reducer.js\)](#) is used in the [todoReducer\(src/todo/reducer.js\)](#) function to iterate over the [state\(src/todo/reducer.js\)](#) array (which represents the list of todo items) and create a new array based on the logic provided in the callback function.

What is [map\(src/todo/components/main.jsx\)](#) in JavaScript?

- [map\(src/todo/components/main.jsx\)](#) is a method available on arrays in JavaScript.
- It creates a new array by applying a callback function to each element of the original array.
- The original array remains unchanged.

Syntax:

```
array.map((element, index, array) => {  
  // Return a new value for each element  
});
```

Example:

```
const numbers = [1, 2, 3];  
const doubled = numbers.map((num) => num * 2);  
console.log(doubled); // Output: [2, 4, 6]
```

How [state.map\(src/todo/reducer.js\)](#) is Used in Your Codebase:

In the [todoReducer\(src/todo/reducer.js\)](#), [state.map\(src/todo/reducer.js\)](#) is used to update specific todo items while leaving the rest unchanged.

Example: [UPDATE_ITEM\(src/todo/constants.js\)](#) Case

```
case UPDATE_ITEM:  
  newState = state.map((todo) =>  
    todo.id === action.payload.id ? { ...todo, title:  
action.payload.title } : todo  
  );  
  break;
```

Explanation:

1. [state\(src/todo/reducer.js\)](#):
 - Represents the current array of todo items.
 2. [state.map\(src/todo/reducer.js\)](#):
 - Iterates over each [todo\(src/todo/components/item.jsx\)](#) in the [state\(src/todo/reducer.js\)](#) array.
 3. **Callback Logic:**
 - If the [todo.id\(src/todo/components/item.jsx\)](#) matches [action.payload.id\(src/todo/reducer.js\)](#), a new object is created with the updated [title\(src/todo/components/item.jsx\)](#) ([action.payload.title\(src/todo/reducer.js\)](#)).
 - If the [id\(src/todo/components/item.jsx\)](#) does not match, the original [todo\(src/todo/components/item.jsx\)](#) is returned unchanged.
 4. **Result:**
 - A new array ([newState\(src/todo/reducer.js\)](#)) is created where only the matching todo item is updated.
-

Why Use [map\(src/todo/components/main.jsx\)](#)?

- [map\(src/todo/components/main.jsx\)](#) is used here because it creates a new array, which is essential for maintaining immutability in React's state management.
 - React relies on immutability to detect changes and trigger re-renders efficiently.
-

Example in Context:

If the current [state\(src/todo/reducer.js\)](#) is:

```
[
  { id: 1, title: "Learn React", completed: false },
  { id: 2, title: "Build a Todo App", completed: true }
]
```

And the [UPDATE_ITEM\(src/todo/constants.js\)](#) action is dispatched with:

```
{
  type: "UPDATE_ITEM",
  payload: { id: 1, title: "Learn React Hooks" }
}
```

The resulting [newState\(src/todo/reducer.js\)](#) will be:

```
[
  { id: 1, title: "Learn React Hooks", completed: false },
  { id: 2, title: "Build a Todo App", completed: true }
]
```

This ensures only the relevant todo item is updated while the rest remain unchanged.

> Response rendering

Participant

OK.

So it's calls this function.

On all elements.

And then it takes. Well, I guess the todo is an item.

It checks if it is.

Whatever this is.

Chat (window: 0)

> Agent

github.copilot.default

> Context

codebase

> Message

00:41:04
- 00:41:11

00:41:26

00:41:27

00:41:29

00:41:32

00:41:36

00:41:39

00:41:44

#codebase what is ction.payload.id ?

> Response

The term `ction.payload.id` in your code appears to be a **typo**. It is likely intended to be [action.payload.id \(src/todo/reducer.js\)](#).

Correct Code:

```
case UPDATE_ITEM:
  newState = state.map((todo) => (todo.id === action.payload.id ? {
    ...todo, title: action.payload.title } : todo));
  break;
```

Explanation:

1. [action \(src/todo/reducer.js\)](#):

- Represents the object dispatched to the [todoReducer \(src/todo/reducer.js\)](#) function.
- It contains two main properties:
 - [type \(node_modules/@types/react/index.d.ts\)](#): The type of action (e.g., [UPDATE_ITEM \(src/todo/constants.js\)](#)).
 - [payload \(src/todo/components/item.jsx\)](#): The data associated with the action.

2. [action.payload \(src/todo/reducer.js\)](#):

- The [payload \(src/todo/components/item.jsx\)](#) property contains the data needed to perform the action. For [UPDATE_ITEM \(src/todo/constants.js\)](#), it includes:
 - [id \(node_modules/@types/react/index.d.ts\)](#): The unique identifier of the todo item to update.
 - [title \(src/todo/components/item.jsx\)](#): The new title for the todo item.

3. [action.payload.id \(src/todo/reducer.js\)](#):

- Refers to the [id \(node_modules/@types/react/index.d.ts\)](#) of the todo item being updated.
- It is used to find the specific todo in the [state \(src/todo/reducer.js\)](#) array that matches this [id \(node_modules/@types/react/index.d.ts\)](#).

Fix:

Replace `ction.payload.id` with [action.payload.id \(src/todo/reducer.js\)](#) in your code:

```
case UPDATE_ITEM:
  newState = state.map((todo) => (todo.id === action.payload.id ? {
    ...todo, title: action.payload.title } : todo));
  break;
```

This ensures the reducer correctly identifies the todo item to update. Let me know if you need further clarification!

00:41:48

Participant

And then.

00:41:49
- 00:41:53

> Response rendering

00:41:51

Participant

Oh this is a ternary operator, so it either does nothing.

00:41:55

Or.

00:41:58

It appends.

00:42:02

The.

00:42:03

Title.

00:42:10
00:42:20
00:42:24
00:42:28
00:42:31
00:42:41
00:42:47
00:42:54
00:42:57
00:43:02

Ah, it actually explained this here so.

The payload.

OK, so an identifier for the item.

Thay may, it's just in the index I think.

The title is the string.

Or maybe it's not actually an array as in not just contiguous memory.

But more like a hash map and the ID is then the.

The key.

How is an array stored in memory in JavaScript?

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/reducer.js)

> Message

How is an array stored in memory in javascript?

> Response

In JavaScript, arrays are stored in memory as **objects** with special behavior. Unlike low-level languages like C or C++, JavaScript arrays are not contiguous blocks of memory. Instead, they are dynamic and flexible, allowing for non-contiguous storage and mixed data types.

Key Points About Array Storage in JavaScript:

1. Arrays Are Objects:

- Internally, arrays are a type of object with numeric keys (indices) and a length property.
- Example:

```
const arr = [10, 20, 30];  
console.log(typeof arr); // "object"
```

2. Dynamic and Sparse:

- JavaScript arrays can grow or shrink dynamically.
- They can also have "holes" (undefined values) if elements are skipped or deleted.
- Example:

```
const arr = [];  
arr[5] = 50; // Creates a sparse array  
console.log(arr); // [ <5 empty items>, 50 ]
```

3. Non-Contiguous Memory:

- Unlike arrays in low-level languages, JavaScript arrays do not guarantee contiguous memory allocation.
- The JavaScript engine optimizes array storage based on usage patterns.

4. Mixed Data Types:

- JavaScript arrays can store elements of different types (numbers, strings, objects, etc.).
- Example:

```
const arr = [1, "hello", { key: "value" }];
```

5. Indexed Properties:

- Array elements are stored as key-value pairs where the keys are numeric indices.
- Example:

```
const arr = [10, 20, 30];  
console.log(arr[0]); // 10
```

6. Length Property:

- The length property is automatically updated to reflect the highest numeric index + 1.
- Example:

```
const arr = [10, 20];  
arr[5] = 50;  
console.log(arr.length); // 6
```

Memory Representation:

- **Dense Arrays:**

- If the array is small and has no "holes," the JavaScript engine may store it as a dense array for performance.
- Example:

```
const arr = [1, 2, 3]; // Stored densely
```

- **Sparse Arrays:**

- If the array has "holes" or non-sequential indices, it is stored as a sparse array (like a hash map).
- Example:

```
const arr = [];  
arr[100] = 42; // Sparse array
```

Practical Implications:

- Arrays are flexible but may have performance trade-offs if used inefficiently (e.g., creating sparse arrays).
- For performance-critical applications, avoid creating sparse arrays or mixing data types unnecessarily.

Let me know if you'd like further clarification!

00:43:04
- 00:43:20

00:43:19

> Response rendering

Participant

OK.

00:43:20

So it's not.

00:43:21

It's not like an array as

00:43:22

I'm used to it in like Fortran or C.

00:43:30

So.

00:43:39

No, and you don't know how it is stored, so you have to use some kind of method to find on this.

00:43:44

But at least it looks like OK, so.

00:43:45

Can you sort arrays?

00:43:49

Uhm.

00:43:52

Lexicographical of strings because I could just call the sort every time we add an

00:43:57

Chat (window: 0)

00:43:57

> Agent

github.copilot.default

> Context

viewport (src/todo/reducer.js)

> Message

Javascript lexicographical sort of strings

> Response

In JavaScript, you can perform a lexicographical (alphabetical) sort of strings using the `Array.prototype.sort()` method. By default, `sort()` arranges strings in lexicographical order based on Unicode values.

Example of Lexicographical Sort:

```
const strings = ["banana", "apple", "cherry", "date"];
strings.sort();
console.log(strings); // Output: ["apple", "banana", "cherry", "date"]
```

Explanation:

1. Default Behavior:

- `sort()` converts elements to strings (if they aren't already) and compares their Unicode values.
- This works well for simple alphabetical sorting.

2. Case Sensitivity:

- By default, uppercase letters come before lowercase letters because of their Unicode values.
- Example:

```
const strings = ["Banana", "apple", "Cherry"];
strings.sort();
console.log(strings); // Output: ["Banana", "Cherry", "apple"]
```

3. Custom Comparator for Case-Insensitive Sort:

- To sort strings in a case-insensitive manner, use a custom comparator:

```
const strings = ["Banana", "apple", "Cherry"];
strings.sort((a, b) => a.localeCompare(b));
console.log(strings); // Output: ["apple", "Banana", "Cherry"]
```

4. Reverse Lexicographical Sort:

- To sort in descending order, reverse the comparison:

```
const strings = ["banana", "apple", "cherry", "date"];
strings.sort((a, b) => b.localeCompare(a));
console.log(strings); // Output: ["date", "cherry", "banana", "apple"]
```

Notes:

- **localeCompare:**

- This method compares two strings in a locale-sensitive manner, making it useful for sorting strings with special characters or different languages.
- Example:

```
const strings = ["ä", "z", "a"];
strings.sort((a, b) => a.localeCompare(b, "de")); // German
locale
console.log(strings); // Output: ["a", "ä", "z"]
```

Let me know if you need further clarification!

00:44:06 Or update an item which is not very efficient but.
00:44:10 If you want efficient code, I guess you shouldn't use a scripting
00:44:13 language anyway.
00:44:17 OK, so you can just call sort.
00:44:18 > **Editing file** (src/todo/reducer.js)
- 00:44:24

00:44:21 **Participant**
I'm just going to say new state is sort new state.
00:44:30 Let's see if this works.

00:44:35 Uh.
00:44:49 No, this doesn't work, and there's also.
00:44:54 Apparently he doesn't like what this means.
00:44:55 > **Editing file** (src/todo/reducer.js)
- 00:44:56

00:44:58 **Participant**
Maybe state dot sort?
00:45:06 The quick.
00:45:11 No, this is not sorted.
00:45:14 Hmm.
00:45:23 So new, is new state an array?
00:45:29 **Chat** (window: 0)

> **Agent**

github.copilot.default

> **Context**

viewport (src/todo/reducer.js)

> **Message**

is newstate an array?

> **Response**

Yes, [newState \(src/todo/reducer.js\)](#) is an array in your [todoReducer \(src/todo/reducer.js\)](#) function. It is derived from the [state \(src/todo/reducer.js\)](#) array, which represents the current list of todo items. Each case in the switch statement modifies the [state \(src/todo/reducer.js\)](#) array and assigns the result to [newState \(src/todo/reducer.js\)](#).

Why [newState \(src/todo/reducer.js\)](#) is an Array:

1. Initial State:

- The [state \(src/todo/reducer.js\)](#) parameter in the [todoReducer \(src/todo/reducer.js\)](#) is an array, as it is initialized with [initialState \(src/todo/reducer.js\)](#), which is loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) or defaults to an empty array:

```
export const initialState = loadState(); // Returns an array
```

2. Array Operations:

- In each case of the switch statement, array methods like [concat \(src/todo/reducer.js\)](#), [map \(src/todo/reducer.js\)](#), and [filter \(src/todo/reducer.js\)](#) are used to create [newState \(src/todo/reducer.js\)](#). These methods always return a new array:
 - [concat \(src/todo/reducer.js\)](#): Adds a new todo item to the array.
 - [map \(src/todo/reducer.js\)](#): Iterates over the array and returns a new array with modified elements.
 - [filter \(src/todo/reducer.js\)](#): Returns a new array with elements that pass a condition.

3. Examples in the Reducer:

- [ADD ITEM \(src/todo/constants.js\)](#):

```
newState = state.concat({ id: generateUniqueID(), title:
action.payload.title, completed: false });
```

- [state.concat\(src/todo/reducer.js\)](#) returns a new array with the new todo item added.
- [UPDATE_ITEM\(src/todo/constants.js\)](#):

```
newState = state.map((todo) => (todo.id === action.payload.id ?
{ ...todo, title: action.payload.title } : todo));
```

- [state.map\(src/todo/reducer.js\)](#) returns a new array with the updated todo item.
- [REMOVE_ITEM\(src/todo/constants.js\)](#):

```
newState = state.filter((todo) => todo.id !==
action.payload.id);
```

- [state.filter\(src/todo/reducer.js\)](#) returns a new array excluding the removed item.

4. Immutability:

- React's state management relies on immutability. Instead of modifying the original [state\(src/todo/reducer.js\)](#) array, a new array ([newState\(src/todo/reducer.js\)](#)) is created and returned.

5. Saving State:

- After modifying the array, [newState\(src/todo/reducer.js\)](#) is saved to [localStorage\(node_modules/@types/node/globals.d.ts\)](#):

```
saveState(newState);
```

Conclusion:

Yes, [newState\(src/todo/reducer.js\)](#) is always an array in your [todoReducer\(src/todo/reducer.js\)](#). It is a modified version of the [state\(src/todo/reducer.js\)](#) array, created using immutable operations.

00:45:46

00:45:49

00:46:04

And is sort lexicographical? Or does it do something else?

Yeah, it is an array.

OK.

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

viewport (src/todo/reducer.js)

> **Message**

Javascript, how do I sort an array?

> **Response**

In JavaScript, you can sort an array using the `Array.prototype.sort()` method. This method sorts the elements of an array **in place** and returns the sorted array.

Syntax:

```
array.sort([compareFunction]);
```

- **compareFunction** (optional): A function that defines the sort order. If omitted, the array is sorted lexicographically (alphabetically) based on the Unicode values of the elements.
-

Examples:

1. Sorting Strings (Lexicographical Order):

```
const fruits = ["banana", "apple", "cherry"];
fruits.sort();
console.log(fruits); // Output: ["apple", "banana", "cherry"]
```

2. Sorting Numbers:

By default, `sort()` ([src/todo/reducer.js](#)) converts elements to strings and compares them lexicographically, which can lead to incorrect results for numbers. Use a compare function to sort numbers correctly:

```
const numbers = [10, 5, 20, 1];
numbers.sort((a, b) => a - b); // Ascending order
console.log(numbers); // Output: [1, 5, 10, 20]

numbers.sort((a, b) => b - a); // Descending order
console.log(numbers); // Output: [20, 10, 5, 1]
```

3. Sorting Objects by a Property:

```
const todos = [
  { id: 1, title: "Learn React", completed: false },
  { id: 2, title: "Build a Todo App", completed: true },
  { id: 3, title: "Learn JavaScript", completed: false }
];

// Sort by title (alphabetically)
todos.sort((a, b) => a.title.localeCompare(b.title));
console.log(todos);

// Sort by `id` (numerically)
todos.sort((a, b) => a.id - b.id);
console.log(todos);
```

4. Case-Insensitive String Sort:

```
const words = ["Banana", "apple", "Cherry"];
words.sort((a, b) => a.localeCompare(b, undefined, { sensitivity: "base"
}));
console.log(words); // Output: ["apple", "Banana", "Cherry"]
```

5. Reverse Order:

To sort in descending order, reverse the comparison:

```
const fruits = ["banana", "apple", "cherry"];
fruits.sort().reverse();
console.log(fruits); // Output: ["cherry", "banana", "apple"]
```

Notes:

- **Mutates the Array:** The [sort\(\) \(src/todo/reducer.js\)](#) method modifies the original array.
- **Immutability:** If you need to keep the original array intact, create a copy before sorting:

```
const original = [3, 1, 4];
const sorted = [...original].sort((a, b) => a - b);
console.log(original); // [3, 1, 4]
console.log(sorted); // [1, 3, 4]
```

Let me know if you need further clarification!

00:46:07
- 00:46:20

> Response rendering

00:46:09

Participant

That's also a very long answer for just yes.

00:46:17

OK.

00:46:18

Oh, in place. That's nice. So.

00:46:23

I have to call.

00:46:23
- 00:46:25

> **Editing file** (src/todo/reducer.js)

00:46:25

Participant

Array dot sort.

00:46:36
- 00:46:43

> **Editing file** (src/todo/reducer.js)

00:46:36

Participant

Oh I guess array the name of the thing, so.

00:46:39

New state dot sort and then the comparison function which is a lexicographical sort.

00:46:44

All right.

00:46:47

It's also so here.

00:46:48

But there's no function.

00:46:54

Maybe that's the default?

00:46:59

Oh yes, it's the default. If it's submitted, then it starts sorting strings.

00:47:02

> **Editing file** (src/todo/reducer.js)

00:47:04

00:47:05

Participant

OK, start again.

00:47:07

That still doesn't work.

00:47:17

Uh.

00:47:20

So, but it does render after every callback to this to this thing. I can edit the others and see.

00:47:30

00:47:35

Maybe if I was mistaken and it should be in a different case.

00:47:37

> **Editing file** (src/todo/reducer.js)

00:47:39

00:47:51

Participant

No.

00:47:55

Oh, but it's definitely not a remove item.

00:48:05

Researcher

All right, since we're at the time limit, I'll ask you to stop here. Thanks a lot.

00:48:08

00:51:16

Participant

I mean, I think the length was too long.

00:51:20

Uhm.

00:51:22 Because there was also a lot of fluff you to go through. It had good details.
00:51:27 And.
00:51:30 Yeah, technical terms and correctness.
00:51:32 I just don't know enough of JavaScript to judge. Instructions was kind of OK.
00:51:40 So.
00:51:42 Was fine I guess.
00:51:45 Formatting as well.
00:51:51 I find this hard to judge because.
00:51:54 Normally I use.
00:51:57 Language I know.
00:52:00 And I have to change.
00:52:03 Like a few lines in a codebase of 500 thousand lines and it's different than this.
00:52:11 I don't think so to be honest, because I think it's like I don't have to
00:52:15 type or look up that much.
00:52:16 It's really like thinking about the task and the algorithm that is the most time.
00:52:22 So I also don't think this would really improve anything here.
00:52:33 Well, it's easy to use.
00:52:38 Uhm.
00:52:41 I don't know.
00:52:42 I would be very surprised if it.
00:52:46 Could actually solve algorithmic problems because LLMs just guess the next word and
00:52:51 don't word and don't actually or next token I guess instead of actually having
00:52:57 any reasoning.
00:53:00 It probably would be clear and understandable.
00:53:05 Because it.
00:53:07 Yeah, the text itself is pretty fluent.
00:53:11 Skillful.
00:53:15 I guess it was pretty easy to use.
00:53:19 Hmm.
00:53:24 Well it was definitely not physically demanding 'cause I was just typing.
00:53:29 Umm.
00:53:33 But I couldn't solve the even the practice exercises,
00:53:37 but I think that's more to never having touched JavaScript before.
00:53:42 I'll put it like here then.
00:53:45 And.
00:53:49 I never really cared about finishing.
00:53:53 The exercises.
00:53:53 So I'll put that here.
00:53:57 Well, I did accomplish like two so,
00:54:00 I suppose I can put it.
00:54:01 I'll put it here.
00:54:04 Uh.
00:54:07 I did try my best.
00:54:10 Hmm.
00:54:11 I've also also pretty annoyed because the task is so trivial and there was just so
00:54:19 much.
00:54:21 Stuff, I think.
00:54:21 It's not that, I think it's unnecessary in such a simple application.
00:54:27 And like there's no compiler, there's no types.
00:54:31 To much constructs that annoyed me.
00:54:33 Quite a bit.
00:54:39 This was not hard because you could just look for it.
00:54:43 **Researcher**
Sorry, these are for the for the other tasks.
00:54:46 **Participant**
Oh, OK.
00:54:52 Right.
00:54:54 **Researcher**

00:54:57 Again, no worries that you didn't get to these.
00:55:00 You're not the only one there.
00:55:05 Great. So I have a few questions for you.
00:55:09 To start off with, how did you manage to find your way
00:55:12 through the codebase?
Participant
Grep.
00:55:16 **Researcher**
And did Copilot also help you with this?
00:55:23 **Participant**
So I did try to use this at like the last exercise to look where things are added,
00:55:28 but it looks like, I don't know if it was accurate.
00:55:31 I guess if it was accurate, it helped, but it looked like it didn't do anything.
00:55:37 But it may also be because like my sorting call was wrong.
00:55:40 Yeah. So I guess Copilot also helped there.
00:55:44 To like find the array and things like that.
00:55:47 **Researcher**
OK. And what do you mean it's you didn't know
00:55:50 if it did anything. Is this regarding the navigation?
00:55:53 **Participant**
Well, like I.
00:55:55 Yeah, yeah.
00:55:56 So it it pointed me to a place in the code.
00:55:59 **Researcher**
Mm hmm.
00:56:00 **Participant**
But I'm not sure if it was the actual correct place.
00:56:04 **Researcher**
That makes sense, so it's hard to judge.
00:56:06 **Participant**
Yeah.
00:56:09 **Researcher**
Do you feel like you understood the codebase?
00:56:12 **Participant**
No.
00:56:16 **Researcher**
That's very fair.
00:56:19 And and obviously you did spend some effort on trying to understand the
00:56:24 codebase. Did you feel like you at least somewhat
00:56:25 **Participant**
Mm hmm.
00:56:27 **Researcher**
succeeded in this?
00:56:30 **Participant**
Umm.
00:56:32 No, I think if I actually wanted to
00:56:34 understand, I would just have to follow a basic
00:56:37 JavaScript tutorial first. Like that would be more efficient and then look at the
00:56:40 **Researcher**
Mm.
00:56:42 **Participant**
code base again.
00:56:45 **Researcher**
OK.
00:56:45 **Participant**
Because I just didn't have enough handles to like look at things.
00:56:49 **Researcher**
That makes sense.

00:56:51 And did Copilot also impact your understanding of the code?
00:56:55 **Participant**
Umm.
00:56:58 Yeah, well, at least it pointed me into some
00:57:00 directions, but I mean, I know from students in [omitted]
00:57:04 that as well, if you very authoritative answer,
00:57:08 that may be completely incorrect.
00:57:10 So again, hard to judge, but like if accurate then it it did help.
00:57:15 But I still think it is less efficient than just learning the basics of the
00:57:21 language and then.
00:57:23 Reading it as you do normally.
00:57:27 **Researcher**
Yeah, that makes sense.
00:57:29 So.
00:57:32 Without having access to Copilot, what would your strategy for solving
00:57:35 these tasks have looked like?
00:57:40 **Participant**
So, I would for the first two.
00:57:43 How much time would I have?
00:57:47 **Researcher**
How much time?
00:57:47 **Participant**
Could I take?
00:57:48 Could I like take an hour to follow a basic JavaScript tutorial and then do it?
00:57:52 Or do I really have to finish it in one hour?
00:57:53 **Researcher**
Yeah.
00:57:54 So.
00:57:56 So this would have been your preferred strategy if you had the time.
00:57:59 **Participant**
If I had the time, I would look up a tutorial first.
00:58:03 **Researcher**
Yeah.
00:58:04 **Participant**
And then I would do the first two exercises the same,
00:58:04 **Researcher**
At.
00:58:07 **Participant**
and then the third I would know what to what terms to look for and then grab it.
00:58:07 **Researcher**
Mm hmm.
00:58:14 **Participant**
To find out where it is in which file it is, and then change it there.
00:58:19 **Researcher**
OK. And hypothetically say you had to solve
00:58:22 this task as quickly as possible. How would you have done it?
00:58:27 **Participant**
I would have approached it this way and then probably well, I might have.
00:58:35 I probably would have followed, solved the first two this way,
00:58:39 then try to third one again like this for 10 minutes and then after 10 minutes I
00:58:44 could already see OK, this is not going to.
00:58:47 I'm not going to solve it this way, so that's also why I looked up all these
00:58:49 **Researcher**
Yeah.
00:58:51 **Participant**
terms. So at that point I would have.
00:58:54 Went to a tutorial.
00:58:56 **Researcher**

00:58:56 Fair enough.

00:59:01 That makes sense.

00:59:06 So you indicated what you thought about some aspects of Copilot responses.

00:59:11 Did any of these stand out to you specifically?

Participant

00:59:14 Yeah, the one where I asked is state an array

00:59:19 and then it gave me like 200 words instead of yes.

00:59:22 But other than that, not really.

Researcher

00:59:24 So it's mainly about the about the length there.

Participant

00:59:27 Yeah.

Researcher

00:59:29 Yeah.

00:59:33 Yeah, I saw quite a few times you were looking

00:59:39 at answers and then quickly glanced over them.

00:59:44 So why is it that you sort of would not want to read an answer

00:59:47 that's as long as thi? Is it the time?

00:59:49 Or is it the effort?

00:59:51 Or is it the cognitive load?

00:59:52 Do you have any idea?

Participant

00:59:55 Yeah, so after like I read then the first two

00:59:58 sentences and I.

01:00:01 Didn't know so many I already didn't know half of the terms.

01:00:05 So at that point I knew that I would not be able to understand it.

01:00:11 And then I just scanned for like bullet points and some familiar variable names

01:00:14 and things like that because I thought I would.

01:00:19 I'm not able to take more useful things from this answer anyway.

Researcher

01:00:21 Yeah.

01:00:23 About those technical terms.

01:00:27 Would you have preferred if the answer was the same length,

01:00:32 just in simpler terms that you would understand or terms that are more

01:00:36 relevant to your programming experience?

Participant

01:00:41 Yeah, maybe I should have asked what is a state

01:00:44 in C terminology instead?

01:00:48 So just give me, because I probably know these concepts.

01:00:53 I'm just not familiar with the language.

01:00:57 And then, I think the concepts behind it are not

01:01:02 difficult to understand or at least I already know them.

01:01:05 But I just want to know what it is.

Researcher

01:01:07 Fair enough.

01:01:10 So would would that have been?

01:01:14 A nice solution for you if the answers were presented in terms that that were

01:01:16 familiar to you?

Participant

01:01:18 Yes, I think that would have helped.

Researcher

01:01:21 Yeah.

01:01:23 OK.

01:01:26 Let me see.

01:01:32 So you used Copilot over time throughout the tasks.

Participant

01:01:34 Mm.

Researcher

01:01:39 And maybe you observed some things that Copilot was good at or not so good at.
01:01:42 Did you change the way you interacted with Copilot in response to what you
01:01:45 observed?
Participant
No, not really.

01:01:48 **Researcher**
No.
01:01:49 So you feel the way you.
01:01:52 The questions you asked and the way you interacted stayed consistent throughout
01:01:55 the task?
01:01:57 **Participant**
Yeah, kind of like how it would Google things.

01:02:03 **Researcher**
Is this sort of influenced, do you think, by what you expect from
01:02:08 Copilot?
01:02:09 Of what it can and can't do.
01:02:13 **Participant**
Yes, I do think so because I don't expect it
01:02:18 to.
01:02:19 Actually reason to like be a compiler and be able to for example do type inference
01:02:26 or show me the the stack trace of of a thing.
01:02:31 So I didn't even bother to ask.
01:02:33 **Researcher**
Alright.
01:02:36 What if you were able to copy and paste the text of the task into Copilot?
01:02:43 **Participant**
Hmm.
01:02:43 **Researcher**
And it will be able to fully generate all the code changes that you needed to make.
01:02:50 Would you do this or would you want this?
01:02:52 **Participant**
Yeah, if it works.
01:02:55 Does it actually?
01:02:58 **Researcher**
I think for the the tasks that we have here for most of them it it it does work,
01:03:04 yes.
01:03:04 **Participant**
Ah, cool.
01:03:07 **Researcher**
Um.
01:03:08 Do you see any advantages or disadvantages to this?
01:03:14 **Participant**
Yeah, I mean the advantage is that just that you can quickly get things done
01:03:18 without understanding what's going on and also the disadvantage is that you can
01:03:22 **Researcher**
Mm hmm.
01:03:23 **Participant**
quickly get things done without understanding what is going on.
01:03:27 Because if you have like a very big project and a bunch of people who don't
01:03:31 know.
01:03:33 What code they commit. Just put it in and then it doesn't
01:03:37 immediately crash.
01:03:38 Then it's going to cause a lot of problems, like a month, a year,
01:03:42 10 years down the line.
01:03:47 **Researcher**
I think now you took a quite step by step approach trying to understand things
01:03:52 quite carefully. If I interpreted that correctly.
01:03:57 So obviously using Copilot to solve a task immediately.

01:04:03 Is a completely different approach.
01:04:06 Um.
01:04:09 What way do you think?
01:04:12 You might best understand.
01:04:15 The code and the codebase.
01:04:19 **Participant**
So I think.
01:04:22 Having some kind of control flow graph, or at least have that in your head. So the
01:04:27 code that starts executing at this function and then.
01:04:32 If click on this button and this function will execute and things like that.
01:04:37 So kind of like a stack trace thing.
01:04:42 Maybe a short overview for each file is for.
01:04:49 Simplified because I saw there was a lot of indirection like a function.
01:04:53 It's just a call back for another function which just thus yet another
01:04:57 function to kind of write down on piece of paper and simplified it a bit,
01:05:01 or even simplified it into code if possible.
01:05:07 Yeah, I think that is how I would approach it.
01:05:10 **Researcher**
OK.
01:05:12 Also, you asked some general questions. For example in JavaScript,
01:05:16 how do you do this and this?
01:05:19 **Participant**
Mm hmm.
01:05:22 **Researcher**
Would you have wanted?
01:05:25 Copilot to also relate this to the current file you were working on or to
01:05:29 the current task.
01:05:33 **Participant**
No, because it was so general.
01:05:35 Like what is an array that I don't think there is a file specific answer to that.
01:05:40 **Researcher**
OK.
01:05:42 So.
01:05:45 For you, when you ask these kinds of questions,
01:05:48 you just want the general answer.
01:05:52 In order to learn how JavaScript works, for example.
01:05:56 **Participant**
Yes.
01:05:57 **Researcher**
OK.
01:05:57 **Participant**
At least in these cases where I ask about state and components and those things.
01:06:02 **Researcher**
OK. And for example, when you asked how do you sort an array?
01:06:09 This sort of relates more to the the current task.
01:06:14 **Participant**
But it's still a general function.
01:06:17 I just want to know the name of the function basically.
01:06:20 **Researcher**
That makes sense.
01:06:21 **Participant**
But where else?
01:06:22 Where is the list stored?
01:06:25 Or like what variable is it in like that was specific to the code base so.
01:06:31 That's nice.
01:06:33 **Researcher**
And did you feel it?
01:06:34 It managed to in that case relat it to the current codebase well enough?

01:06:39

Participant

Yeah, I think it did

01:06:40

look this, this when I found out when they told me about this map thing

01:06:45

did. Then that is, that looks at least to me like where the list was.

01:06:50

I'm just really confused why the sort didn't work.

01:06:53

Researcher

Yeah.

01:06:53

Participant

At that place.

01:06:56

Researcher

Makes sense.