On occasion, you may need to use a function or library that only works client-side. This usually is because the library in question accesses something that isn't available during server-side rendering (SSR), like browser DOM methods.

You'll need to use one of the workarounds outlined below if your project fails to compile with `gatsby develop` or `gatsby build` with an error like:

```
Reference error: window is not defined
```

Workaround 1: Use a different library or approach

Sometimes the simplest approach is to work around the problem. If you can re-implement your component using a plugin which *doesn't* break SSR, that's probably best.

Workaround 2: Add client-side package via CDN

In the component where you need it, load the package via CDN using a `<script />` tag.

To embed your script, you can:

- Include it in a custom component as needed using `react-helmet`.
- Add the script tag directly by using Gatsby's `setHeadComponents` in the `onRenderBody` API in `gatsby-ssr`.

You should then follow React's guidelines for Integrating with DOM Manipulation Plugins, using the methods available in the React Component Lifecycle to interact with the library you're using.

```
import React, { Component } from "react"
import { Helmet } from "react-helmet"

class MyComponent extends Component {
  componentDidMount() {
    // set up and use external package as needed
    window.externalLibrary.method()
  }

  render(props) {
    return (
      <React.Fragment>
        <Helmet>
          <script src="https://cdn.example/path-to-external-library.js" />
        </Helmet>

        <h1>Hello World</h1>
        {/* etc */}
      </React.Fragment>
    )
  }
}
```

Workaround 3: Use React.lazy and Suspense on client-side only

React.lazy and Suspense are not ready for server-side rendering, but they can be used by checking that the code is executed only on the client. While this solution is inferior to `loadable-components`, that works both on server side and client, it still provides an alternative for dealing with client-side only packages, without an added dependency. Remember that the following code could break if executed without the `isSSR` guard.

```
import React from "react"

const ClientSideOnlyLazy = React.lazy(() =>
  import("../components/ClientSideOnly")
)
const MyPage = () => {
  const isSSR = typeof window === "undefined"

  return (
    <>
      {!isSSR && (
        <React.Suspense fallback={<div />}>
          <ClientSideOnlyLazy />
        </React.Suspense>
      )}
    </>
  )
}
```

Workaround 4: Load client-side dependent components with loadable-components

Install loadable-components and use it as a wrapper for a component that wants to use a client-side only package.

```
npm install @loadable/component
```

And in your component:

```
import React, { Component } from "react"
import PropTypes from "prop-types"

import Loadable from "@loadable/component"

// these two libraries are client-side only
import Client from "shopify-buy"
import ShopifyBuy from "@shopify/buy-button-js"

const ShopifyBuyButton = props => {
  // custom component using shopify client-side libraries
  return <div>etc</div>
}

const LoadableBuyButton = Loadable(() => import("./ShopifyBuyButton"))
```

```
export default LoadableBuyButton
```

> ***Note:*** *There are other potential workarounds than those listed here. If you've had success with another method, check out the contributing docs and add yours!*

If all else fails, you may also want to check out the documentation on Debugging HTML Builds.