Traditionally, websites are styled using global CSS files.

Globally-scoped CSS rules are declared in external `.css` stylesheets, and CSS specificity and the Cascade determine how styles are applied.

## Adding global styles with a layout component

The best way to add global styles is with a shared layout component. This layout component is used for things that are shared throughout the site, including styles, header components, and other common items.

> ***NOTE:*** *This pattern is implemented by default in the default starter.*

To create a shared layout with global styles, start by creating a new Gatsby site with the hello world starter.

```
gatsby new global-styles https://github.com/gatsbyjs/gatsby-starter-hello-world
```

Open your new site in your code editor and create a new directory at `/src/components`. Inside, create two new files:

```
  global-styles/
  └──src/
      └──components/
+     │   ├─  layout.js
+     │   └─  layout.css
      │
      └──pages/
          └─  index.js
```

Inside `src/components/layout.css`, add some global styles:

```
div {
  background: red;
  color: white;
}
```

In `src/components/layout.js`, include the stylesheet and export a layout component:

```
import React from "react"
import "./layout.css"

export default function Layout({ children }) {
  return <div>{children}</div>
}
```

Finally, update `src/pages/index.js` to use the new layout component:

```
import React from "react"
import Layout from "../components/layout"

export default function Home() {
  return <Layout>Hello world!</Layout>
}
```

Run `npm run develop` and you’ll see the global styles applied.

Global styles

## Adding global styles without a layout component

In some cases, using a shared layout component is not desirable. In these cases, you can include a global stylesheet using `gatsby-browser.js`.

> ***NOTE:*** *This approach does not work with CSS-in-JS. Use shared components to share styles in CSS-in-JS.*

First, open a new terminal window and run the following commands to create a new default Gatsby site and start the development server:

```
gatsby new global-style-tutorial https://github.com/gatsbyjs/gatsby-starter-default
cd global-style-tutorial
npm run develop
```

Second, create a CSS file and define any styles you wish. An example:

```
html {
  background-color: peachpuff;
}

a {
  color: rebeccapurple;
}
```

Then, include the stylesheet in your site's `gatsby-browser.js` file.

> ***NOTE:*** *This solution works when including CSS as those styles are extracted when building the JavaScript but not for CSS-in-JS. Including styles in a layout component or a global-styles.js is your best bet for that.*

```
import "./src/styles/global.css"

// or:
// require('./src/styles/global.css')
```

> *Note: You can use Node.js require or import syntax. Additionally, the placement of the example CSS file in a `src/styles` folder is arbitrary.*

You should see your global styles taking effect across your site:

Global styles example site

### Importing CSS files into components

It is also possible to break up your CSS styles into separate files so that team members can work independently while still using traditional CSS. You can then import files directly into pages, templates, or components:

```
.menu {
  background-color: black;
  color: #fff;
  display: flex;
}
```

```
import "css/menu.css"
```

This approach can simplify integration of CSS or Sass styles into your Gatsby site by allowing team members to write and consume more traditional, class-based CSS. However, there are trade-offs that must be considered with regards to web performance and the lack of dead code elimination.

### Adding classes to components

Since `class` is a reserved word in JavaScript, you'll have to use the `className` prop instead, which will render as the browser-supported `class` attribute in your HTML output.

```
<button className="primary">Click me</button>
```

```
.primary {
  background: orangered;
}
```

### Limitations

The biggest problem with global CSS files is the risk of name conflicts and side effects like unintended inheritance.

CSS methodologies like BEM can help solve this, but a more modern solution is to write locally-scoped CSS using CSS Modules or CSS-in-JS.