After installing `gatsby-plugin-mdx`, MDX files located in `src/pages` will turn into pages.

Pages are rendered at a URL that is constructed from the filesystem path inside `src/pages`. An MDX file at `src/pages/awesome.mdx` will result in a page being rendered at `mysite.com/awesome`.

> *`gatsby-plugin-mdx` looks for MDX files and automatically transpiles them so that Gatsby internals can render them.*

Using frontmatter in MDX

By default, `gatsby-plugin-mdx` supports frontmatter so you can define things like titles and paths to use in your GraphQL queries. You can declare frontmatter at the beginning of your MDX document:

```
---
title: Hello, world!
path: /hello-world
date: 2019-01-29
---

# Hello, world!
```

Which can then be queried with GraphQL:

```
query {
  allMdx {
    edges {
      node {
        frontmatter {
          title
          path
          date(formatString: "MMMM DD, YYYY")
        }
      }
    }
  }
}
```

> ***Note:** To query `MDX` content, it must be included in the node system using a source like the `gatsby-source-filesystem` plugin first. Instructions for sourcing content from somewhere like your `/src/pages` directory can be found on the plugin's README.*

Frontmatter is also available in `props.pageContext.frontmatter` and can be accessed in blocks of JSX in your MDX document:

```
---
title: Building with Gatsby
author: Jay Gatsby
---

<h1>{props.pageContext.frontmatter.title}</h1>
```

```
<span>{props.pageContext.frontmatter.author}</span>

(Blog post content, components, etc.)
```

Importing JSX components and MDX documents

Similarly to what you'd do in plain React, you can import and render JSX components directly in MDX files. You can also import other MDX documents.

```
import { Chart } from "../components/chart"
import FAQ from "../components/faq.mdx"

# Here’s a chart

The chart is rendered inside our MDX document.

<Chart />
<FAQ />
```

The `<Chart />` component coming from a `.js` file would be written like any other React component, while the `<FAQ />` component coming from an `.mdx` file might look something like this:

```
## Frequently Asked Questions

### Why Gatsby?

Gatsby delivers faster, more secure sites and apps from a variety of data
sources

### Where do I start?

The documentation offers guides for all different skill levels, you can
find more info at the Gatsby's [Quick Start page](/docs/quick-start)

<!-- This default export overrides the default layout ensuring -->
<!--  that the FAQ component isn't wrapped by other elements -->
export default function Layout({ children }) {
  return (
    <>
      {children}
    </>
  );
}
```

> ***Note**: the default export concept used in this code block is explained in more detail in the docs below on defining layouts*

You can read more about using React components from other libraries in the Importing and Using components in MDX guide.

Combining frontmatter and imports

If you would like to include frontmatter metadata *and* import components, the frontmatter needs to appear at the top of the file and then imports can follow:

```
---
title: Building with Gatsby
---

import { Chart } from "../components/chart"

Markdown and more content...
```

Using JavaScript exports

MDX supports `export` syntax as well, which enables specific use cases like providing data for queries and rendering or overriding the default layout on MDX documents. You don't need to export MDX documents to import them in other files.

Exporting page metadata

You can provide additional data about a given document by exporting. `gatsby-plugin-mdx` will automatically add it to the GraphQL schema so you can use the exported data in your queries and in rendering.

Data exported in MDX documents in this manner is also made available on the variable name you've assigned it.

You can export variables, objects, or other data structures:

```
export const metadata = {
  name: "World",
  path: "/world",
};

# Hello, <span children={metadata.name} />

The span above will read: "Hello, World".

<!-- you can also use other variables or data structures -->
export const names = ["Abdullah", "Adam", "Alice", "Aida"]

<ul>{names.map(name => <li>{name}</li>)}</ul>
```

The fields `name` and `path` defined on `metadata` could now alternatively be accessed on MDX nodes in other areas of your Gatsby project by a GraphQL query like this (this query fetches all MDX nodes and the data exports associated with them):

```
query MdxExports {
  allMdx {
    nodes {
      exports {
        metadata {
          name
          path
        }
      }
    }
  }
}
```

Defining a layout

If you have provided a default layout in your `gatsby-config.js` through the `gatsby-plugin-mdx` plugin options, the exported component you define from this file will replace the default.

```
import PurpleBorder from "../components/purple-border"

# This will have a purple border

export default PurpleBorder
```

The `<PurpleBorder />` component might look something like this, wrapping the MDX document in a `<div>` with a 1px purple border:

```
import React from "react"

const PurpleBorder = ({ children }) => (
  <div style={{ border: "1px solid rebeccapurple" }}>{children}</div>
)

export default PurpleBorder
```

GraphQL queries

You can fetch data to use in your MDX file by exporting a `pageQuery` in the same way you would for a `.js` page. The queried data is passed as a prop, and can be accessed inside any JSX block when writing in MDX:

```
import { graphql } from "gatsby"

# My Awesome Page

Here's a paragraph, followed by a paragraph with data!

<p>{props.data.site.siteMetadata.description}</p>

export const pageQuery = graphql`
  query {
    site {
      siteMetadata {
        description
        title
      }
    }
  }
`
```

> *Note: For now, this only works if the `.mdx` file exporting the query is placed in `src/pages`. Exporting GraphQL queries from `.mdx` files that are used for programmatic page creation in `gatsby-node.js` via `actions.createPage` is not currently supported.*