gatsby-remark-prismjs

Adds syntax highlighting to code blocks in markdown files using PrismJS.

Install

```
npm install gatsby-transformer-remark gatsby-remark-prismjs
prismjs
```

How to use

```
// In your gatsby-config.js
plugins: [
  {
    resolve: `gatsby-transformer-remark`,
    options: {
      plugins: [
        {
          resolve: `gatsby-remark-prismjs`,
          options: {
            // Class prefix for <pre> tags containing syntax highlighting;
            // defaults to 'language-' (e.g. <pre class="language-js">).
            // If your site loads Prism into the browser at runtime,
            // (e.g. for use with libraries like react-live),
            // you may use this to prevent Prism from re-processing syntax.
            // This is an uncommon use-case though;
            // If you're unsure, it's best to use the default value.
            classPrefix: "language-",
            // This is used to allow setting a language for inline code
            // (i.e. single backticks) by creating a separator.
            // This separator is a string and will do no white-space
            // stripping.
            // A suggested value for English speakers is the non-ascii
            // character '›'.
            inlineCodeMarker: null,
            // This lets you set up language aliases.  For example,
            // setting this to '{ sh: "bash" }' will let you use
            // the language "sh" which will highlight using the
            // bash highlighter.
            aliases: {},
            // This toggles the display of line numbers globally alongside the code.
            // To use it, add the following line in gatsby-browser.js
            // right after importing the prism color scheme:
            //  require("prismjs/plugins/line-numbers/prism-line-numbers.css")
            // Defaults to false.
            // If you wish to only show line numbers on certain code blocks,
```

```
            // leave false and use the {numberLines: true} syntax below
            showLineNumbers: false,
            // If setting this to true, the parser won't handle and highlight inline
            // code used in markdown i.e. single backtick code like `this`.
            noInlineHighlight: false,
            // This adds a new language definition to Prism or extend an already
            // existing language definition. More details on this option can be
            // found under the header "Add new language definition or extend an
            // existing language" below.
            languageExtensions: [
              {
                language: "superscript",
                extend: "javascript",
                definition: {
                  superscript_types: /(SuperType)/,
                },
                insertBefore: {
                  function: {
                    superscript_keywords: /(superif|superelse)/,
                  },
                },
              },
            ],
            // Customize the prompt used in shell output
            // Values below are default
            prompt: {
              user: "root",
              host: "localhost",
              global: false,
            },
            // By default the HTML entities <>&'" are escaped.
            // Add additional HTML escapes by providing a mapping
            // of HTML entities and their escape value IE: { '}': '&#123;' }
            escapeEntities: {},
          },
        },
      ],
    },
  },
]
```

Include CSS

Required: Pick a PrismJS theme or create your own PrismJS ships with a number of themes (previewable on the PrismJS website) that you can easily include in your Gatsby site, or you can build your own by copying and

modifying an example (which is what we've done for gatsbyjs.com).

To load a theme, just require its CSS file in your `gatsby-browser.js` file, e.g.

```
// gatsby-browser.js
require("prismjs/themes/prism-solarizedlight.css")
```

Optional: Add line highlighting styles If you want to highlight lines of code, you also need to add some additional CSS that targets our *custom line highlighting implementation* (which slightly differs from PrismJS's own plugin for that – more on that later).

For line highlights similar to PrismJS's, try:

```
.gatsby-highlight-code-line {
  background-color: #feb;
  display: block;
  margin-right: -1em;
  margin-left: -1em;
  padding-right: 1em;
  padding-left: 0.75em;
  border-left: 0.25em solid #f99;
}
```

This should work out quite nicely for the "Solarized Light" PrismJS theme we just added in the previous part. However, you will notice that when a highlighted line runs wider than the surrounding code block container (causing a horizontal scrollbar), its background won't be drawn for the initially hidden, overflowing part. :(

We saw others fix that problem and decided to do so, too. Just add the following CSS along your PrismJS theme and the styles for `.gatsby-highlight-code-line`:

```
/**
 * Add back the container background-color, border-radius, padding, margin
 * and overflow that we removed from <pre>.
 */
.gatsby-highlight {
  background-color: #fdf6e3;
  border-radius: 0.3em;
  margin: 0.5em 0;
  padding: 1em;
  overflow: auto;
}

/**
 * Remove the default PrismJS theme background-color, border-radius, margin,
 * padding and overflow.
```

```
 * 1. Make the element just wide enough to fit its content.
 * 2. Always fill the visible space in .gatsby-highlight.
 * 3. Adjust the position of the line numbers
 */
.gatsby-highlight pre[class*="language-"] {
  background-color: transparent;
  margin: 0;
  padding: 0;
  overflow: initial;
  float: left; /* 1 */
  min-width: 100%; /* 2 */
}
```

Optional: Add line numbering If you want to add line numbering alongside your code, you need to import the corresponding CSS file from PrismJS, right after importing your colorscheme in `gatsby-browser.js`:

```
// gatsby-browser.js
require("prismjs/plugins/line-numbers/prism-line-numbers.css")
```

Then add in the corresponding CSS:

```
/**
 * If you already use line highlighting
 */

/* Adjust the position of the line numbers */
.gatsby-highlight pre[class*="language-"].line-numbers {
  padding-left: 2.8em;
}

/**
 * If you only want to use line numbering
 */

.gatsby-highlight {
  background-color: #fdf6e3;
  border-radius: 0.3em;
  margin: 0.5em 0;
  padding: 1em;
  overflow: auto;
}

.gatsby-highlight pre[class*="language-"].line-numbers {
  padding: 0;
  padding-left: 2.8em;
  overflow: initial;
```

```
}
```

Optional: Add shell prompt If you want a fancy prompt on anything with `shell` or `bash`, you need to import the following CSS file in `gatsby-browser.js`:

```
// gatsby-browser.js
require("prismjs/plugins/command-line/prism-command-line.css")
```

If you want to change the resulting prompt, use the following CSS:

```
.command-line-prompt > span:before {
  color: #999;
  content: " ";
  display: block;
  padding-right: 0.8em;
}

/* Prompt for all users */
.command-line-prompt > span[data-user]:before {
  content: "[" attr(data-user) "@" attr(data-host) "] $";
}

/* Prompt for root */
.command-line-prompt > span[data-user="root"]:before {
  content: "[" attr(data-user) "@" attr(data-host) "] #";
}

.command-line-prompt > span[data-prompt]:before {
  content: attr(data-prompt);
}
```

Usage in Markdown

This is some beautiful code:

````
```javascript
// In your gatsby-config.js
plugins: [
  {
    resolve: `gatsby-transformer-remark`,
    options: {
      plugins: [
        `gatsby-remark-prismjs`,
      ]
    }
  }
]
```
````

Line numbering

To see the line numbers alongside your code, you can use the `numberLines` option:

````
```javascript{numberLines: true}
// In your gatsby-config.js
plugins: [
  {
    resolve: `gatsby-transformer-remark`,
    options: {
      plugins: [
        `gatsby-remark-prismjs`,
      ]
    }
  }
]
```
````

You can also start numbering at any index you wish (here, numbering will start at index 5):

````
```javascript{numberLines: 5}
// In your gatsby-config.js
plugins: [
  {
    resolve: `gatsby-transformer-remark`,
    options: {
      plugins: [
        `gatsby-remark-prismjs`,
      ]
    }
  }
]
```
````

Line highlighting

You can also add line highlighting. It adds a span around lines of code with a special class `.gatsby-highlight-code-line` that you can target with styles. See this README for more info.

To highlight lines, you can use one of the following directives as comments in your code:

- `highlight-line` highlights the current line;
- `highlight-next-line` highlights the next line;
- `highlight-start` highlights the lines until the matching `hightlight-end`;

- `highlight-range{1, 4-6}` will highlight the next line, and the fourth, fifth and sixth lines.

````
```jsx
class FlavorForm extends React.Component { // highlight-line
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    // highlight-next-line
    this.setState({value: event.target.value});
  }

  // highlight-start
  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }
  // highlight-end

  render() {
    return (
      { /* highlight-range{1,4-9,12} */ }
      <form onSubmit={this.handleSubmit}>
        <label>
          Pick your favorite flavor:
          <select value={this.state.value} onChange={this.handleChange}>
            <option value="grapefruit">Grapefruit</option>
            <option value="lime">Lime</option>
            <option value="coconut">Coconut</option>
            <option value="mango">Mango</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```
````

You can also specify the highlighted lines outside of the code block. In the following code snippet, lines 1 and 4 through 6 will get the line highlighting.

The line range parsing is done with https://www.npmjs.com/package/parse-numeric-range.

```javascript{1,4-6}
// In your gatsby-config.js
plugins: [
  {
    resolve: `gatsby-transformer-remark`,
    options: {
      plugins: [
        `gatsby-remark-prismjs`,
      ]
    }
  }
]
```

Shell prompt

To show fancy prompts next to shell commands (only triggers on `bash` and `shell`), either set `prompt.global` to `true` in `gatsby-config.js`, or pass at least one of `{outputLines: <range>}`, `{promptUser: <user>}`, or `{promptHost: <host>}` to a snippet

By default, every line gets a prompt appended to the start, this behaviour can be changed by specifying `{outputLines: <range>}` to the language.

```shell{outputLines: 2-10,12}
```

The user and host used in the appended prompt is pulled from the `prompt.user` and `prompt.host` values, unless explicitly overridden by the `promptUser` and `promptHost` options in the snippet, e.g.:

```shell{promptUser: alice}{promptHost: dev.localhost}
```

Diff code blocks

You can specify language for `diff` code blocks by using `diff-[language]` to enable syntax highlighting in diffs:

```diff-javascript
```

Line hiding

As well as highlighting lines, it's possible to *hide* lines from the rendered output. Often this is handy when using `gatsby-remark-prismjs` along with `gatsby-remark-embed-snippet`.

As with highlighting lines, you can control which lines to hide by adding directives as comments in your source code.

The available directives are:

- `hide-line` hides the current line;
- `hide-next-line` hides the next line;
- `hide-start` hides the lines until the matching `hide-end`;
- `hide-range{1, 4-6}` will hide the next line, and the fourth, fifth and sixth lines.

The hide-line directives will always be hidden too. Check out the using-remark example site to see how this looks on a live site.

Inline code blocks

In addition to fenced code blocks, inline code blocks will be passed through PrismJS as well.

If you set the `inlineCodeMarker`, then you can also specify a format style.

Here's an example of how to use this if the `inlineCodeMarker` was set to `±`:

```
I can highlight `css±.some-class { background-color: red }` with CSS syntax.
```

This will be rendered in a `<code class=language-css>` with just the (syntax highlighted) text of `.some-class { background-color: red }`

Disabling syntax highlighting

If you need to prevent any escaping or highlighting, you can use the `none` language; the inner contents will not be changed at all.

Add new language definition or extend an existing language

You can provide a language extension by giving a single object or an array of language extension objects as the `languageExtensions` option.

A language extension object looks like this:

```
languageExtensions: [
  {
    language: "superscript",
    extend: "javascript",
    definition: {
      superscript_types: /(SuperType)/,
    },
    insertBefore: {
      function: {
        superscript_keywords: /(superif|superelse)/,
      },
    },
  },
]
```

used options:

- `language` (optional) The name of the new language.
- `extend` (optional) The language you wish to extend.
- `definition` (optional) This is the Prism language definition.
- `insertBefore` (optional) Is used to define where in the language definition we want to insert our extension.

More information of the format can be found here: https://prismjs.com/extending.html

Note:

- One of the parameters `language` and `extend` is needed.
- If only `language` is given, a new language will be defined from scratch.
- If only `extend` is given, an extension will be made to the given language.
- If both `language` and `extend` is given, a new language that extends the `extend` language will be defined.

In case a language is extended, note that the definitions will not be merged. If the extended language definition and the given definition contains the same token, the original pattern will be overwritten.

One of the parameters `definition` and `insertBefore` needs to be defined. `insertBefore` needs to be combined with `definition` or `extend` (otherwise there will not be any language definition tokens to insert before).

In addition to this extension parameters the css also needs to be updated to get a style for the new tokens. Prism will wrap the matched tokens with a `span` element and give it the classes `token` and the token name you defined. In the example above we would match `superif` and `superelse`. In the html it would result in the following when a match is found:

```
<span class="token superscript_keywords">superif</span>
```

Implementation notes

Line highlighting

Please note that we do *not* use PrismJS's line highlighting plugin. Here's why:

- PrismJS plugins assume you're running things client side, but we are *build-time folks*.
- PrismJS's line highlighting plugin implementation does not allow for solid background colors or 100% wide backgrounds that are drawn beyond the *visible part* of the container when content is overflowing.

Our approach follows the Pygments-based implementation of the React Tutorial/Documentation for line highlights:

- It uses a wrapper element `<div class="gatsby-highlight">` around the PrismJS-formatted `<pre><code>`-blocks.
- Highlighted lines are wrapped in `<span class="gatsby-highlight-code-line">`.

- We insert a linebreak before the closing tag of `.gatsby-highlight-code-line` so it ends up at the start of the following line.

With all of this in place, we can apply `float:left; min-width:100%` to `<pre>`, throw our overflow and background on `.gatsby-highlight`, and use `display:block` on `.gatsby-highlight-code-line` – all of this coming together to facilitate the desired line highlight behavior.

Line numbering

Because the line numbering PrismJS plugin runs client-side, a few adaptations were required to make it work:

- A class `.line-numbers` is dynamically added to the `<pre>` element.
- A new node `<span class="line-numbers-rows">` is added right before the closing `</pre>` containing as many empty `<span>`s as there are lines.

See the client-side PrismJS implementation for reference.