Sprint 1

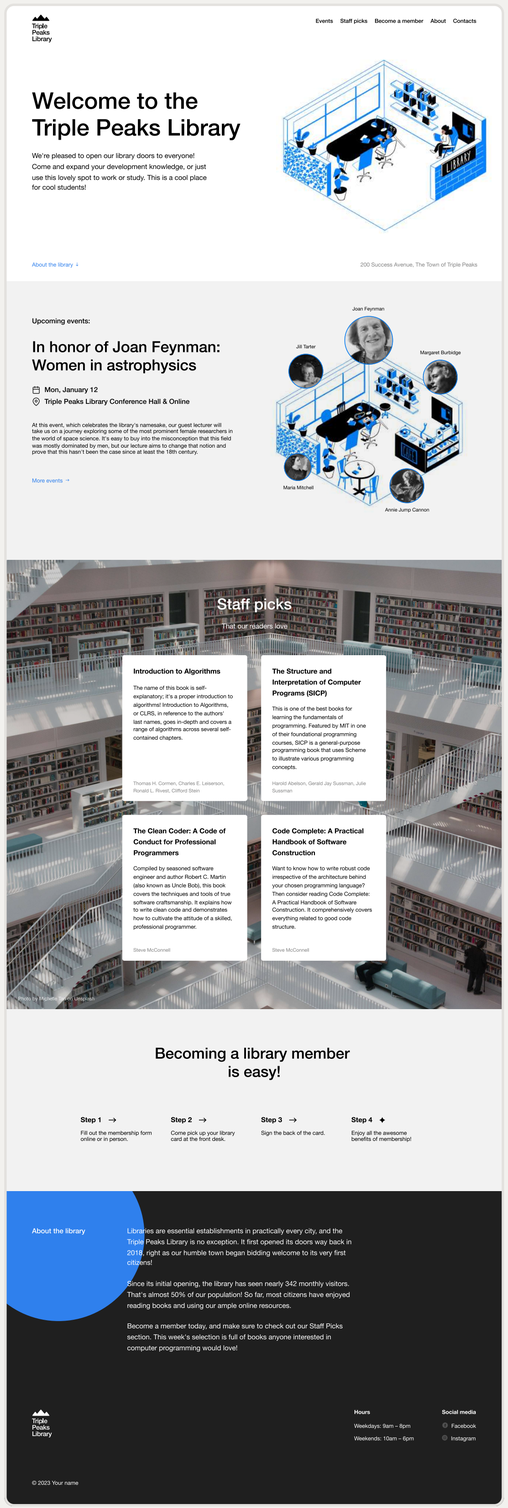
First Project: Advanced HTML and CSS

Final stage

tripleten

Introduction

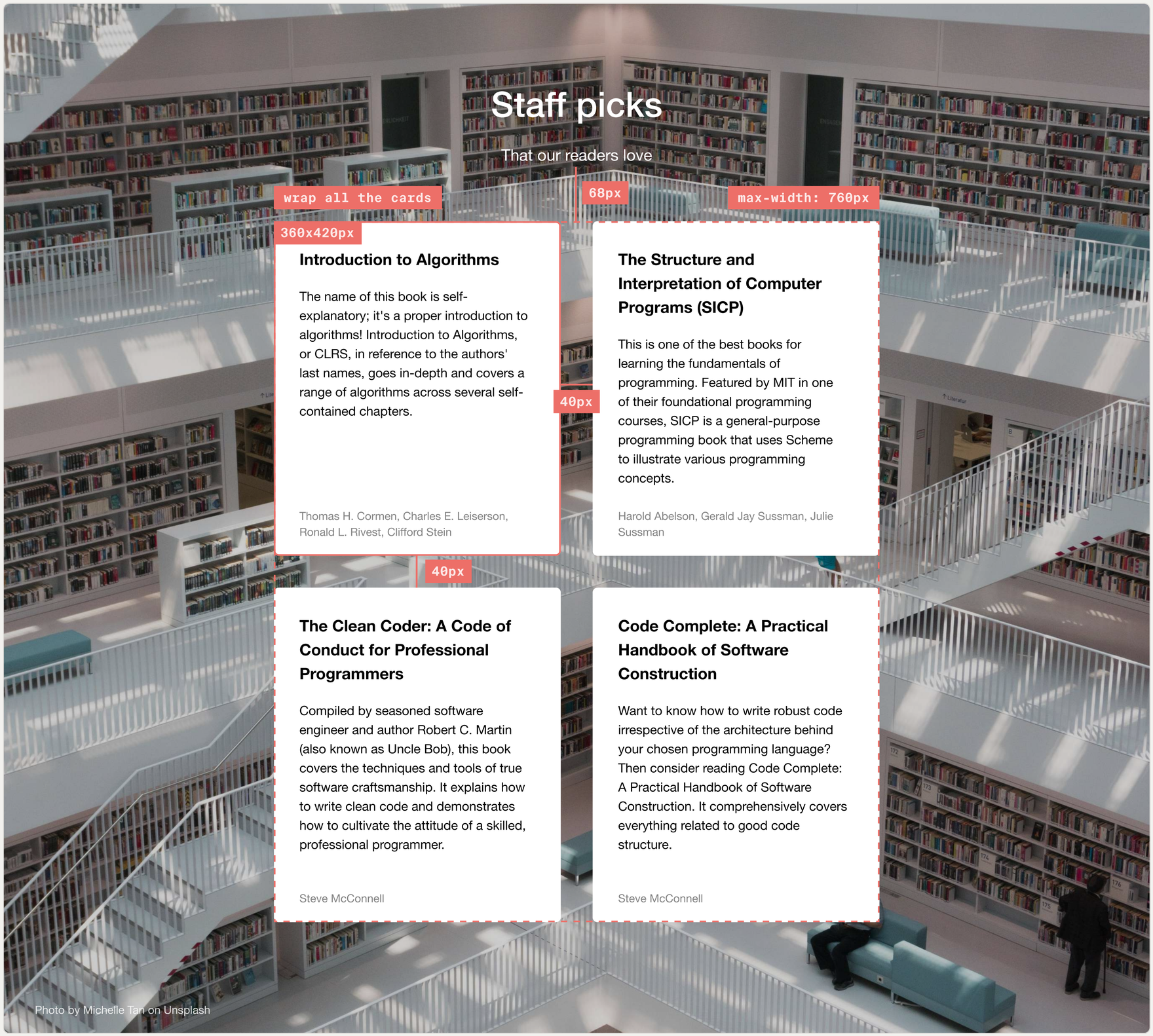
Here's the brief for the final stage of your first project at TripleTen. This time, when you submit the project, it will be personally reviewed by a professional software engineer. By the end of this stage, your project will look like this:



- `#ffffff` (main background, contrast font color)
- `#000000` (main font color)
- `#f2f2f2` (contrast background)
- `#1f1f1f` (footer background)
- `#2f80ed` (links, accents)
- `#1f69c9` (link hover)
- `#c1c1c1` (link hover in footer)

Table of contents

1. Staff picks



See the task on the next page →

1.1. Creating a flexbox container

The cards are currently being rendered one after another, which is the normal behavior of `<ul>` elements with the default `display` value. To arrange the cards as specified in the design, complete the following steps:

- Find the `card` declaration in the provided CSS code, and apply `height` and `width` as shown.
- We've already wrapped the cards in an unordered list tag, or `<ul>`. Give it the `staff__cards` class.
- In the CSS file, before the `card` rule, create a rule for your new class.
- Give the list element the `max-width` and top margin, as shown above, and center it on the page.
- Make the list element a flexbox container.
- Use the flexbox properties `flex-wrap` and `gap` to arrange the cards exactly as they are shown in the design.
- To ensure that the text content is arranged correctly, add `margin-top: auto` to footer of the card.
- Remove the bullet points using the `list-style-type` property.
- Remove the default left padding from the `<ul>` element.
- Add `justify-content: center`, to ensure that the cards will remain centered even if the size of the container changes.

Pro tip:

It can be difficult to see the benefit of using `justify-content` here because we have set it up so that the cards will always fit precisely in their parent container. But if you temporarily remove the `min-width: 1100px` declaration from the page class and reduce the width of the viewport to less than `750px` you can see the issue: when only one column of cards fits on the page, they won't be centered unless you use `justify-content: center`.

In future sprints, you'll learn more techniques for ensuring your project looks good on all screen sizes.

1.2. Set the background-image

Inside the `images` directory you'll find the background image for the staff picks section. Set it as the background for the corresponding `<section>`.

Pro tip:

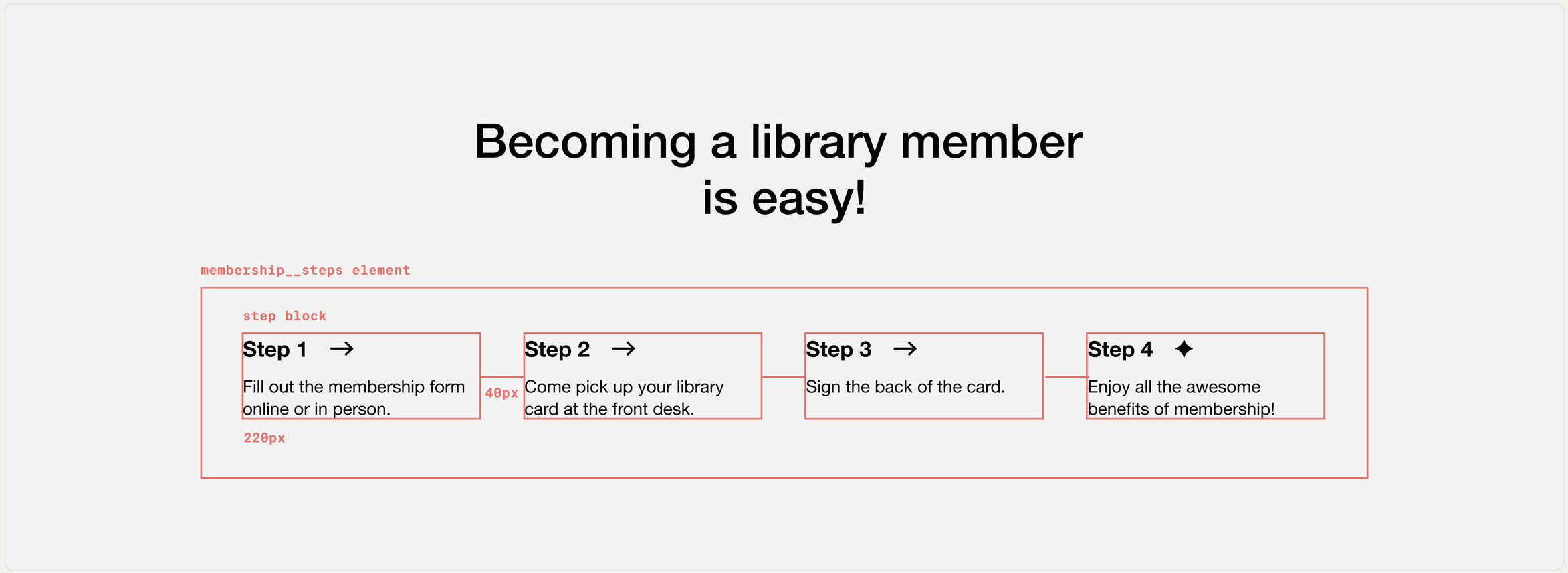
When working with background images, you often need to adjust their presentation with related CSS properties, such as `background-position`, `background-repeat`, and `background-size`. You can read about them on MDN. If you aren't sure which values to use, try them out using the DevTools.

1.3. *Extra task. Adding your books

If you have the time and desire, we suggest changing "Staff picks" to "Reader picks" and adding information about your favorite books! If your text begins to overflow, take this as a bonus challenge. A good search engine query to solve this would be "dealing with text-overflow CSS."

This will give your project a more personal feel, and you'll be able to display more of your personality in your portfolio — your future employer and colleagues are likely to appreciate that!

2. Membership

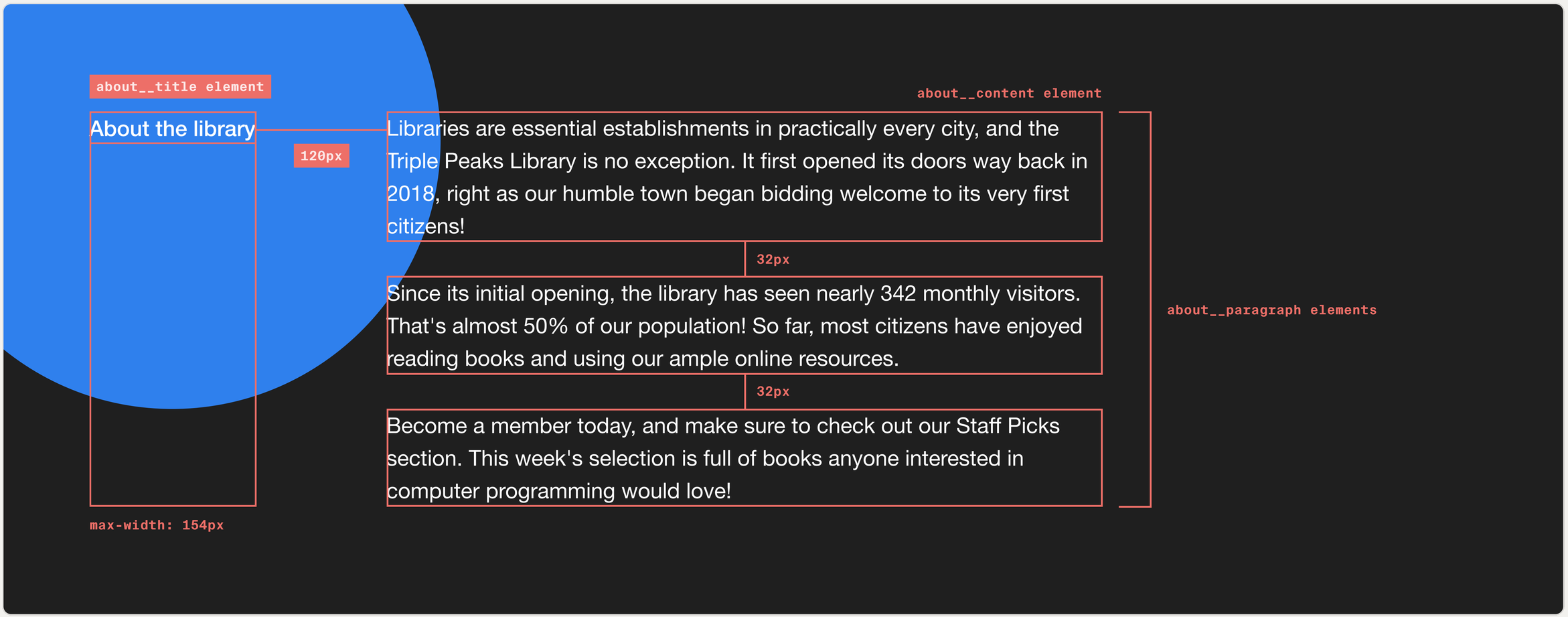


2.1. More flexbox practice!

Update the CSS of the `membership__steps` and `step` classes to arrange the cards as shown in the design. Make sure the gap is correct. Notice that the cards are centered inside the `membership__steps` element.

Use `flex-basis` to set the initial main size of the flex items.

3. About the library

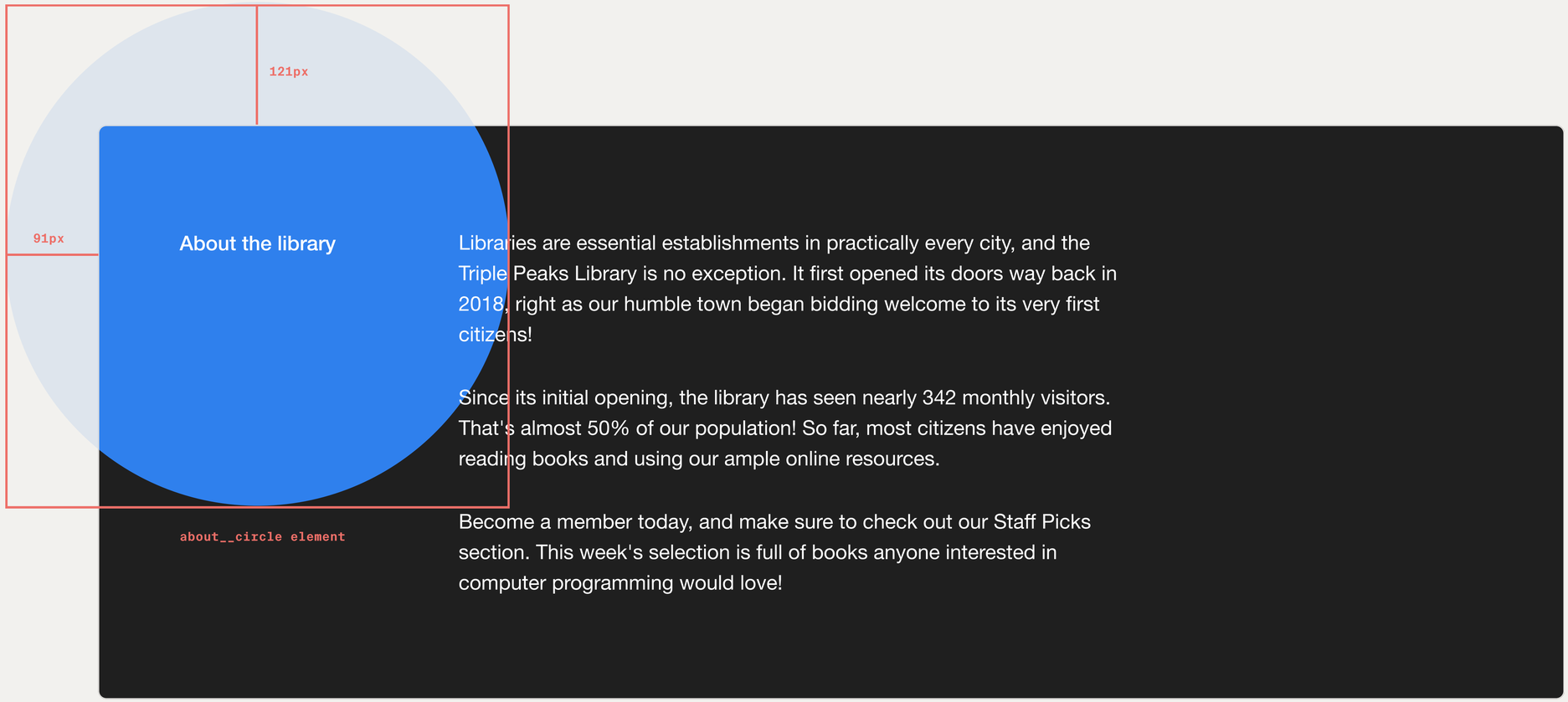


3.1. Arranging text

Arrange the text as shown in the design by adding the necessary styles to `about`, `about__content`, and `about__paragraph`.

Pro tip:

When setting the margins on the `about__paragraph`, keep in mind that there shouldn't be a `32px` margin on the bottom paragraph. To style one of the paragraphs differently, you can make use of a pseudo-class, as described in this lesson.



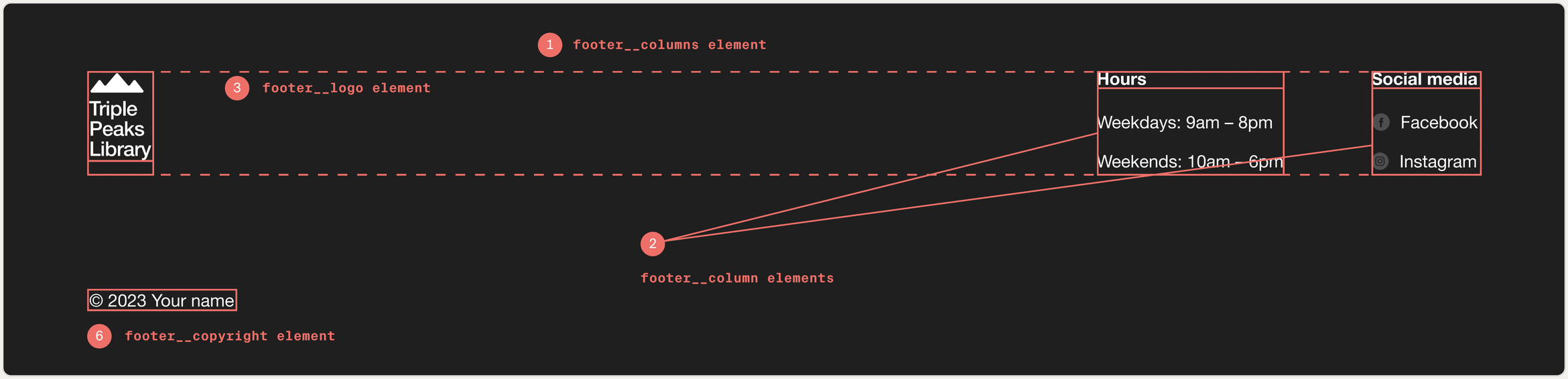
3.2. Adding a circle below the text content

Notice the blue accent circle placed under the text. In the HTML, uncomment the `<div class="about__circle"></div>` element. Finish the styles in `about__circle` to position the circle as shown in the design. Use absolute position and assign `z-index` values to the elements of the sections to arrange them correctly.

Pro tip:

You'll note there's no image corresponding to this element in the `images` directory. In this case, it's simple enough to create the circle using CSS. The trick to making an element look like a circle is to adjust its `border-radius` property. Specifically, if the element is a square (meaning its width and length are equal), then you can make it a circle by giving it `border-radius: 50%`.

4. Footer



4.1. HTML Markup

And so we've come to the last section of the webpage.
Your task is to mark up the footer from scratch based on the details shown in the image above. First, we'll mark up everything in HTML and then add the CSS styles. The appropriate selectors are already in `style.css`. Here's a step-by-step framework to guide you:

1. The logo, hours info, and social media form 3 columns. Let's keep them in one wrapper with the class `footer__columns`.

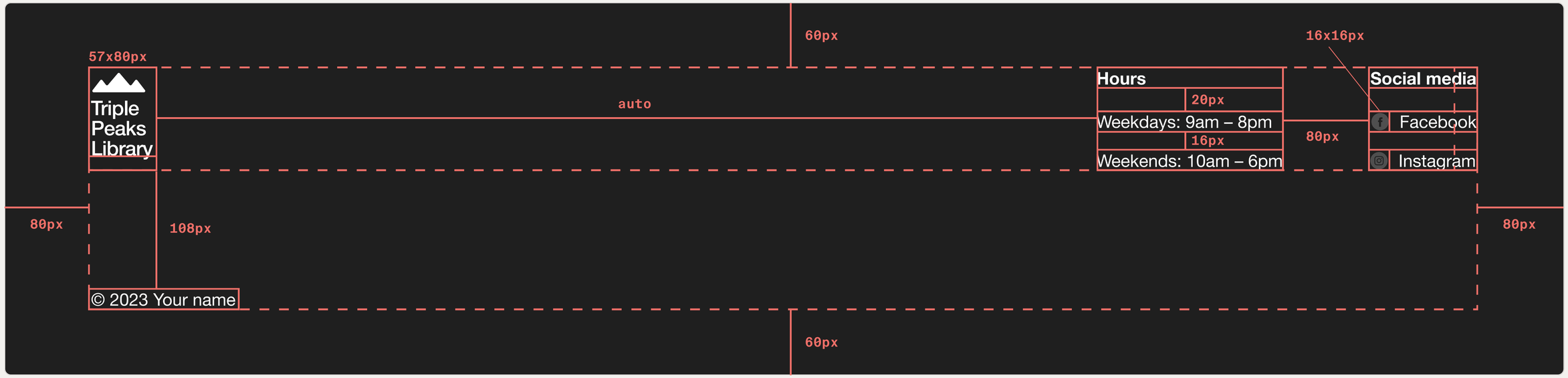
2. Inside the wrapper, create three `<div>` elements, each with the class `footer__column`.

3. Inside the first column, add a logo with the class `footer__logo`.

4. Assign the `footer__column_content_hours` class to the second column. Create a heading of the appropriate level that reads "Hours" and has the class `footer__column-heading`. The info below the column header should be a list with the class `footer__list`. Each line should be a list item with the class `footer__list-item`.

5. Assign the `footer__column_content_social` class to the third column. Create a heading of the appropriate level that reads "Social media" (notice that it's the same level as "Hours") with the class `footer__column-heading`. The info below the column should be a list with the class `footer__list`. Each line should be a list item with the class footer__list-item. Each list item should contain a link with the class `footer__column-link`. Add the text and icons inside the links, so they are both clickable, and assign the class `footer__social-icon` to the icons. The links should lead to `#`.

6. Add an appropriate text element for the copyright. It should say: "© 2023 [Your name]." Assign it the class `footer__copyright`.



4.2 CSS Styles

Now, let's switch to the styles. You can write styles in whatever order is most convenient for you, but you will need to do the following:

1. For the footer: set the background, text color, and padding according to the design.

2. Set the `font-size` and `line-height` of all text elements to `16px` and `20px`, respectively.

> **Pro tip:**
>
> As we noted in our lesson on CSS Inheritance, some CSS properties are inherited. This means that if you apply the property to a parent element, it will also apply to its children. Two examples are `font-size` and `color`.
>
> However, just because a property is inherited, this doesn't mean that the browser will apply it. Inherited property values can be overwritten by the user-agent stylesheet. At least two examples of this can be found in this project.
>
> 1. `color` is an inherited property. However, in the case of `<a>` tags, the user-agent will overwrite the inherited value with the familiar blue color that hyperlinks are often found in.
> 2. `font-size` is inherited too, but for `<h1>`, `<h2>`, and `<h3>` tags, the user-agent will overwrite the inherited value with the default `font-size` that scales with the heading level of the tag.
>
> In both cases, if you want to overwrite the styles provided by the user-agent, you won't be able to rely on inheritance: you'll have to apply the style to the element itself.

3. Use the classes `footer__columns`, `footer__column_content_hours`, and `footer__column_content_social` to arrange the blocks according to the design.

4. Specify the size for the logo and icons: `footer__logo` and `footer__social-icon`.

5. Arrange the content inside the second and third columns according to the design by setting the correct space between elements. Find a way to ensure there is no space after the last item.

6. In the third column, arrange the `footer__column-link` links so that their text and icons are arranged according to the design.

7. Implement the proper hover state for `footer__column-link`. The text color should change to `#c1c1c1`.

8. Specify the margins for `footer__copyright` based on the design. Make sure the copyright text is placed at the bottom of the footer.

5. Finishing touches

At this point, your website should be mostly complete and look close to the design. Soon you'll be able to submit your project for code review by a professional software engineer. But first, we have a few more details to handle.

5.1. Implementing hover states

In the header, we have two types of links: `nav__link` in the navigation bar and `header__link`. According to the design, their colors should change when the user hovers over them with their cursor. To this end, do the following:

- Find the `nav__link` class. Add a new `nav__link` class with the `:hover` pseudo-class directly below these rules. Inside this class, set the color to `#1f69c9`.
- Do the same for `header__link`.
- Do the same for the link in the `events` section too.

5.2. Anchor links

Now, let’s make our menu functional. Four of the links correspond to our `<section>` tags: Staff picks, Events, Become a member, and About. The “Contacts” link should lead to the `<footer>`. Complete the steps as follows:

- Assign an `id` to each of the corresponding target sections and the footer. The name of the `id` is yours to choose, but it's good practice to keep it descriptive. In this case, we suggest making it match the text content of the corresponding anchor tags. Use hyphens `-` as separators instead of space characters, and use lowercase letters.

- Add the `href` attribute for the `nav__link` elements and use their corresponding `id` values. Remember to use `#` for links to the same page.

Also, add one more anchor link for the “About the library” text (the `header__link` element). It should lead to the corresponding section.

Self-review

You are nearing the project finish line, and you’ve already completed the most challenging part. However, there’s still one last crucial task before your project is complete: the self-review. To help, we’ve prepared a checklist:

- [ ] Compare each section of your webpage against the brief (open your project side by side with the brief and go through each point. Consider jotting down notes for yourself with things you want to remember).
- [ ] Check the validity of your code using a validator.
- [ ] Check your project against all the checklist items (you’ll find this in the project description on the platform).
- [ ] Make sure to remove any redundant comment lines. Keep the comments that help you better understand the project structure and facilitate your understanding of complex features.

And with that, you’ve finally finished all the tasks for your very first project at TripleTen. Awesome job!