Forms

- Fields
  - ARIA
  - HTML
- Labels
- Descriptions
- Errors

A form is an element that allows a user to provide data into a field or a group of fields. Forms can be as simple as a single field or as complicated as a multi-step form element with multiple fields per page, input validation, and sometimes a CAPTCHA.

Forms are considered one of the most difficult elements to get right from an accessibility perspective, as they require knowledge of all the elements we have already covered, as well as additional rules specific just to forms. With some understanding and time, you can make an accessible form to suit you and your users.

Forms is the last component-specific module in this course. This module will focus on the form-specific guidelines, but all other guidelines you learned about in the earlier modules, such as content structure, keyboard focus, and color contrast, also apply to form elements.

{% Aside %} Review our Learn Forms course to learn how to create better forms. There is a form accessibility module in that course with additional accessibility-specific form content. {% endAside %}

Fields

The backbone of forms is fields. Fields are small interactive patterns, such as an input or radio button element, that allow users to enter content or make a choice. There is a wide variety of form fields to choose from.

The default recommendation is to use established HTML patterns instead of building something custom with ARIA, as certain features and functionalities—such as field states, properties, and

values—are inherently built into the HTML elements, while you have to add custom ARIA manually.

{% Aside %}

If the situation requires you to build custom form fields, be sure to review the ARIA and HTML, Keyboard focus, and JavaScript modules to understand how to make these custom form fields as accessible as possible.

{% endAside %}

ARIA

```
<div role="form" id="sundae-order-form">
  <!-- form content -->
</div>
```

HTML

```
<form id="sundae-order-form">
  <!-- form content -->
</form>
```

In addition to choosing the most accessible form field patterns, where applicable, you should also add HTML autocomplete attributes to your fields. Adding autocomplete attributes allows a more fine-grained definition or identification of purpose to user agents and assistive technologies (AT).

Autocomplete attributes allow users to personalize visual presentations, such as showing a birthday cake icon in a field with the birthday autocomplete attribute (`bday`) assigned to it. More generally, autocomplete attributes make filling out forms easier and quicker. This is especially helpful for people with cognitive and reading disorders and those using ATs, such as screen readers.

```
<form id="sundae-order-form">
  <p>Name: <input type="name" autocomplete="name"></p>
  <p>Telephone: <input type="tel" autocomplete="tel"></p>
  <p>Email address: <input type="email" autocomplete="email"></p>
</form>
```

Lastly, form fields should not produce contextual changes when they receive focus or user input unless the user has been warned about the behavior before using the component. For example, a form should not be automatically submitted when a field receives focus or once a user adds content to the field.

Labels

Labels inform a user about the purpose of a field, if the field is required, and can also provide information about the field requirements, such as input format. Labels must be visible at all times and accurately describe the form field to users.

One of the foundational tenets of accessible forms is establishing a clear and accurate connection between a field and its label. This is true both visually and programmatically. Without this context, a user might not understand how to fill out the fields in the form.

```
<form id="sundae-order-form">
  <p><label>Name (required): <input type="name" autocomplete="name" required></label></p>
  <p><label>Telephone (required): <input type="tel" autocomplete="tel" required></label></p>
  <p><label>Email address: <input type="email" autocomplete="email"></label></p>
</form>
```

Additionally, related form fields, such as a mailing address, need to be programmatically and visually grouped. One method is to use the fieldset/legend pattern to group elements that are similar.

{% Codepen { user: 'web-dev-codepen-external', id: 'zYLqwwQ', height: 400, theme: 'auto', tab: 'html, result' } %}

Descriptions

Field descriptions are similar to labels in purpose in that they are used to give more context to the field and requirements. Field descriptions are not required for accessibility if the labels or form instructions are descriptive enough.

However, there are situations in which adding additional information is useful to avoid form errors, such as relaying information about the minimum length of input for a password field or telling a user which calendar format to use (such as MM-DD-YYYY).

There are many different methods you can use to add field descriptions to your forms. One of the best methods is to add an aria-describedby attribute to the form element, in addition to a `<label>` element. The screen reader will read both the description and the label.

If you add the aria-labelledby attribute to your element, the attribute value overrides the text within your `<label>`. As always, be sure to test the final code with all of the ATs you plan to support.

{% Codepen { user: 'web-dev-codepen-external', id: 'ExpKmXq', height: 400, theme: 'auto', tab: 'html, result' } %}

Errors

When creating accessible forms, there's a lot you can do to prevent users from making form errors. Earlier in this module, we covered clearly marking-up fields, creating identifying labels, and adding detailed descriptions whenever possible. But no matter how clear you think your form is, eventually, a user will make a mistake.

When a user encounters a form error, the first step is to make the error known. The field where the error occurred must be clearly identified, and the error itself must be described to the user in text.

There are different methods for displaying error messages, such as:

- A pop-up modal, inline near where the error occurred
- A collection of errors grouped in one larger message at the top of the page

Be sure to pay attention to the keyboard focus and ARIA live region options when announcing errors.

Whenever possible, offer the user a detailed suggestion on how to fix the error. There are two attributes available to notify users of errors.

- You can use the HTML required attribute. The browser will supply a generic error message based on the filed validation parameters.
- Or you can use the aria-required attribute to share a customized error message to ATs. Only ATs will receive the message unless you add additional code to make the message visible to all users.

Once a user thinks all of the errors have been resolved, allow them to resubmit the form and provide feedback about the results of their submission. An error message tells a user they have more updates to make, while a success message confirms that they have resolved all errors and successfully submitted the form.

{% Codepen { user: 'web-dev-codepen-external', id: 'abjJYWR', height: 400, theme: 'auto', tab: 'html, result' } %}

{% Aside %}

While WCAG 2.2 is still under development, there are several proposed success criteria that will focus on more accessible form experiences, such as Target Size (Minimum), Consistent Help, Accessible Authentication, and Redundant Entry to be aware of for future projects.

{% endAside %}

{% Assessment 'forms' %}

□□□ #2
□ □□□ □□□ 2026-10-31 22:44:06 UTC
□ □□□ □□□ 2026-10-31 22:44:07 UTC