Session 4

Accessible form notifications

Slide instructions

`SPACEBAR` to **move forward** through slides.

`SHIFT` & `SPACEBAR` to **move backwards** through slides.

`LEFT ARROW` & `RIGHT ARROW` to **move through sections**.

`ESC` to **see overview** and `ESC` again to exit.

`F` to **enter presentation mode** and `ESC` to exit.

Introduction

Part of what makes forms complex for accessibility is the **variety of feedback we need to give our users**.

We can give **live updates** on how well they are meeting the instructions and requirements.

When the form is submitted, we must **let the user know if there were any errors** so they can be fixed before we send the data to the server.

For accessibility we need to make sure that **all of this feedback is available to everyone**.

Live region states and properties

Elements that are dynamically inserted after the initial page load can **cause two potential problems for screen readers**.

Problem 1:

Screen readers “buffer” pages as they are loaded. Any content that is added after this time many not be picked up by the screen reader.

Problem 2:

Screen readers can only focus on one part of the page at a time. If something changes on another area of the page, screen readers will not announce this change unless the user moves to that area.

The live region attributes allow authors to **notify screen readers when content is updated**. The possible live region attributes are:

aria-live
aria-atomic
aria-relevant
aria-busy (state)

aria-live

There are **three possible values**:

- `off`
- `polite`
- `assertive`

Off

Assistive Technologies will be aware that changes have occurred within the element. However, these changes should not be announced unless the region is currently in focus.

The `off` value should be used for information that is not critical, **where changes do not need to be announced**.

```
<div aria-live="off">
</div>
```

Polite

Assistive Technologies should announce updates at the next graceful opportunity.

The `polite` value can be used for warning notifications that users may need to know.

```
<div aria-live="polite">
</div>
```

Assertive

Assistive Technologies should announce updates immediately.

The `assertive` value should only be used if the interruption is imperative for users to know immediately.

```
<div aria-live="assertive">
</div>
```

aria-atomic

The `aria-atomic` attribute indicates whether assistive technologies will present **all or only parts of the changed region**.

The `false` attribute will present **only the changed regions**. This is the default.

```
<div
  aria-live="polite"
  aria-atomic="false"
>
</div>
```

The `true` attribute will present **the region as a whole** when changes are detected.

```
<div
  aria-live="polite"
  aria-atomic="true"
>
</div>
```

This attribute becomes important **depending on the amount of content that needs to be announced**.

```
<div
  aria-live="polite"
  aria-atomic="false"
>
  There are 3 items available.
  Use the arrow keys to navigate.
</div>
```

```
<div
  aria-live="polite"
  aria-atomic="true"
>
  There are 3 items available.
  Use the arrow keys to navigate.
</div>
```

The `aria-live` attribute **should always be present** when using the `aria-atomic` attribute.

```
<div
  aria-live="[ off | polite | assertive ]"
  aria-atomic=" [ false | true ]"
>
</div>
```

aria-relevant

The `aria-relevant` attribute defines **what sorts of changes will be presented to screen readers**.

The `additions` value informs Assistive Technologies of **nodes that are dynamically inserted** within the live region.

```
<div
  aria-live="polite"
  aria-relevant="additions"
>
</div>
```

The `removals` value informs Assistive Technologies of **nodes that are dynamically deleted** within the live region.

```
<div
  aria-live="polite"
  aria-relevant="removals"
>
</div>
```

The `text` value informs Assistive Technologies of **dynamic changes to the textual content within existing nodes** within the live region.

```
<div
  aria-live="polite"
  aria-relevant="text"
>
</div>
```

The `all` value informs Assistive Technologies of **dynamic insertion or deletion of nodes** within the live region.

```
<div
  aria-live="polite"
  aria-relevant="all"
>
</div>
```

The `aria-live` attribute **should always be present** when using the `aria-relevant` attribute.

```
<div
  aria-live="[ off | polite | assertive ]"
  aria-relevant=" [ additions | removals | text |
>
</div>
```

aria-busy

The `aria-busy` attribute indicates **whether an element, and its subtree, are currently being updated**.

The `aria-busy` attribute is a state, which means that it **needs to be changed dynamically using JavaScript** to determine the current state.

The value can be set to `true` **when content is being loaded** within the live region.

```
<div
  aria-live="polite"
  aria-busy="true"
>
</div>
```

The value can be set to `false` **when the content loading has completed** within the live region.

```
<div
  aria-live="polite"
  aria-busy="false"
>
</div>
```

The `aria-live` attribute **should always be present** when using the `aria-busy` attribute.

```
<div
  aria-live="[ off | polite | assertive ]"
  aria-busy=" [ true | false ]"
>
</div>
```

Any questions or comments?

Exercise: Using `aria-live`

Accessing exercise 1:

DEVELOPER EXERCISE: `aria-live="assertive"`

Accessing exercise 2:

DEVELOPER EXERCISE: `aria-live="polite"`

Accessing exercise 3:

DEVELOPER EXERCISE: `aria-live="off"`

Online demos:

- `aria-live="assertive"`
- `aria-live="polite"`
- `aria-live="off"`
- `role="alert"`
- `role="status"`
- Support chart

Password strength

We want to let the user know if their password is secure enough, so we **present a strength meter**.

It should dynmically update **as soon as the user changes what they're typing**.

This is a **good opportunity** to an `assertive` live region.

Any questions or comments?

Exercise: Creating a password strength indicator

Accessing the exercise:

DEVELOPER EXERCISE: Creating a password strength indicator

Character count

If we need a maximum or minimum number of characters in a text area, **we can show a character count**.

A `polite` live region is **more user-friendly than an assertive one** in this situation.

Any questions or comments?

Exercise: Creating a character count indicator

Accessing the exercise:

DEVELOPER EXERCISE: Creating a character count indicator

Client-side form validation

Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This is called **client-side form validation**.

Client-side form validation **helps ensure data submitted matches the requirements** in our instructions.

One key aspect of client side form validation is **notifying users of errors in the form**.

There are **two fundamental approaches** to error notification.

on blur

The first approach involves validating and notifying users of errors as focus leaves each individual form field, often referred to as **onBlur**.

As focus leaves each form field, **users should be notified** if the field is in a state of error.

On submit

The second approach involves validation and notifying users of errors as they attempt to submit the form, often referred to as **onSubmit**.

After the submit button is activated, **users should be notified** of all fields that are in a state of error.

In reality, forms that use onBlur for form field validation **must still use onSubmit validation as well**.

The reason is that users **may not resolve individual form field errors** before attempting to submit the form.

Why?

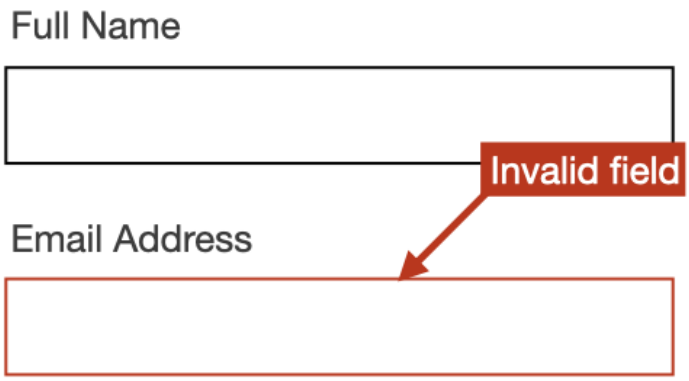
- The user may have been focusing on specific fields and failed to notice the errors above.
- Screen reader software may not have notified users of error messages.
- Screen magnifier software may have magnified the page so that errors are no longer visible.

Any questions or comments?

onBlur form field errors

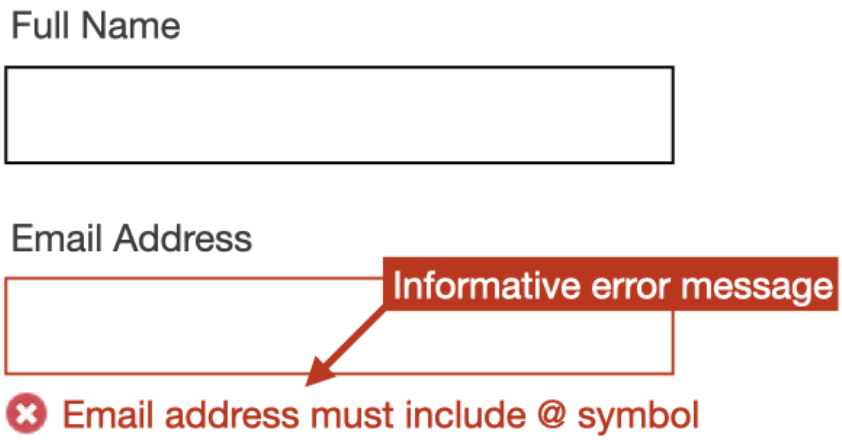
If a form field is in a state of error, **the following should occur**:

1. Each form field that has errors should be **visually "flagged"** to show that they are invalid.



2. An error message **should be provided in the direct vicinity** of the invalid form field.

3. The error message **should be informative** - it should provide information that will help users fill in the field correctly.



4. The error message **must be programmatically associated** with the invalid form field.

This can be done with **matching** `aria-describedby` and `ID` values.

```
<label for="name">Name</label>
<input
  type="text"
  id="name"
  aria-describedby="aaa"
>
<span id="aaa">Error message</span>
```

5. The invalid form field **should be set with** `aria-invalid="true"`.

This will **inform assistive technologies** that the value entered into the input field does not conform to the format expected by the application.

```
<label for="name">Name</label>
<input
  type="text"
  id="name"
  aria-describedby="aaa"
  aria-invalid="true"
>
<span id="aaa">Error message</span>
```

6. The form field and error message can be **wrapped inside a container that is set with** `aria-live`.

This means that the dynamic error state **will be announced as users leave the field**.

```
<div aria-live="polite">
  <label for="name">Name</label>
  <input
    type="text"
    id="name"
    aria-describedby="aaa"
    aria-invalid="true"
  >
  <span id="aaa">Error message</span>
</div>
```

Any questions or comments?

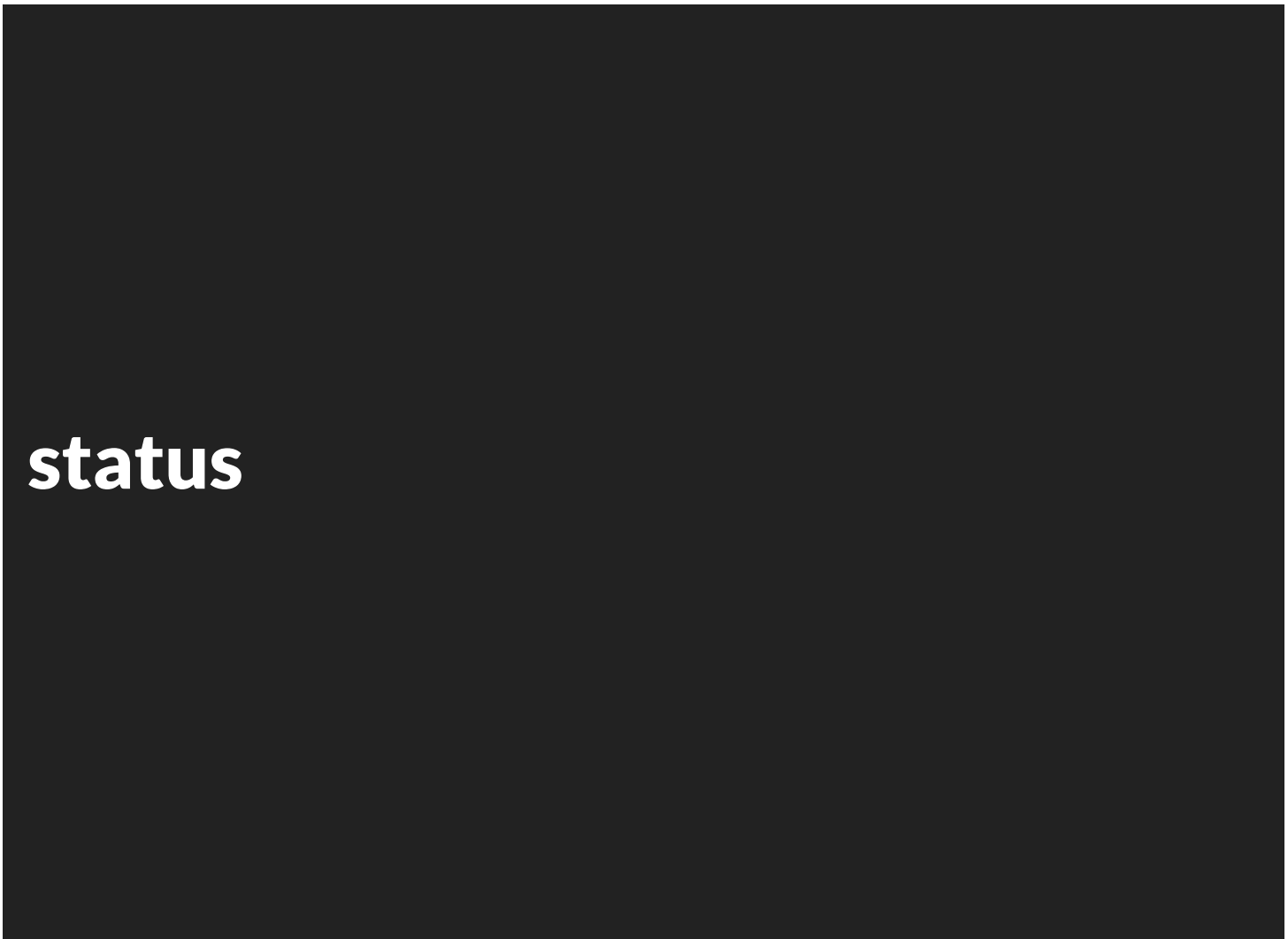
Exercise: Setting up a form field error

Accessing the exercise:

DEVELOPER EXERCISE: Setting up a form field error

Live region roles

Live region **roles** can be used to define live regions of a document, and may be modified by live region attributes.



The `role="status"` is used for content that is **advisory information**, but is not important enough to justify an alert.

```
<div
  role="status"
>
</div>
```

Elements set with `role="status"` have **two implicit values**.

1. Implicit `aria-live="polite"`

This means assistive technologies should announce updates at the next graceful opportunity.

2. Implicit `aria-atomic="true"`

This means the entire region should be announced when changes are detected.

For this reason, these two attributes **do not be need to be defined** when using `role="status"`.

```
<div
  role="status"
  aria-live="polite"
  aria-atomic="true"
>
</div>
```

alert

The `role="alert"` is used for content that is **important, and usually time-sensitive** such as error messages.

```
<div
  role="alert"
>
</div>
```

Elements set with `role="alert"` have **two implicit values**.

1. Implicit `aria-live="assertive"`

This means assistive technologies should announce updates immediately.

2. Implicit `aria-atomic="true"`

This means the entire region should be announced when changes are detected.

For this reason, these two attributes **do not be need to be defined** when using `role="alert"`.

```
<div
  role="alert"
  aria-live="assertive"
  aria-atomic="true"
>
</div>
```

Any questions or comments?

Exercise: Adding live region roles

Accessing exercise 1:

DEVELOPER EXERCISE: `role="status"`

Accessing exercise 2:

DEVELOPER EXERCISE: `role="alert"`

onSubmit errors

If there any errors within the form when the user attempts to submit the form, users must be notified of **all fields that are in an error state**.

Focus could be **sent to either**:

- the first form field in a state of error, or
- an overall error message at the top of the form.

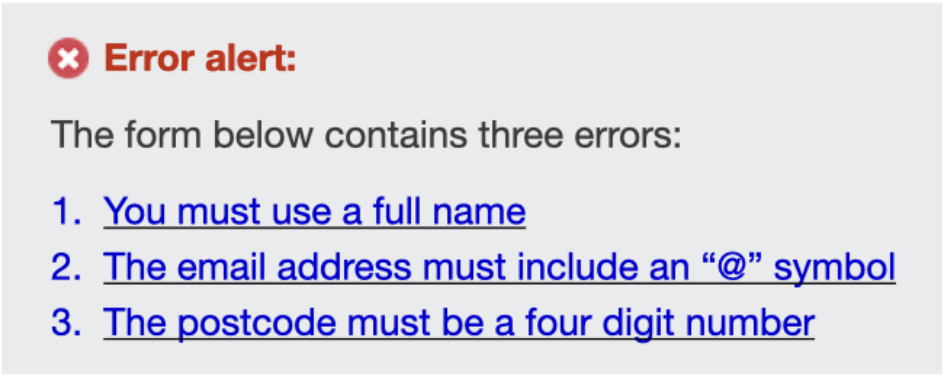
If an overall error message is used, **the following must occur**:

1. An error message must be presented to users, and it should be placed **above the form**.

2. Focus must **shift to the error message**.

3. The error message **should list all errors**.

4. Ideally, **each listed error should be a link** that takes the user to the relevant form error.



5. The error message should be set with `role="alert"` as this will **announce the error to screen reader users** as soon as the error is triggered.

Even though shifting focus will already allow the error message to be announced, using `role="alert"` **changes the semantic nature of the element**.

```
<div role="alert">
  <h3>Error alert</h3>
  <p>The form below contains three errors:</p>
  <ul>
    <li><a href="#">Form field error 1</a></li>
    <li><a href="#">Form field error 2</a></li>
    <li><a href="#">Form field error 3</a></li>
  </ul>
</div>
```

6. The error message must also be set with `tabindex="0"` so that the element **can receive focus** as soon as it is triggered.

```
<div role="alert" tabindex="0">
  <h3>Error alert</h3>
  <p>The form below contains three errors:</p>
  <ul>
    <li><a href="#">Form field error 1</a></li>
    <li><a href="#">Form field error 2</a></li>
    <li><a href="#">Form field error 3</a></li>
  </ul>
</div>
```

Any questions or comments?

Exercise: Creating an onSubmit error message

Accessing the exercise:

DEVELOPER EXERCISE: Creating an on-submit error message

Success messages

In some cases, you may need to inform users that the form has been **submitted sucessfully**.

Ideally, form-based success messages should follow **many of the same principles** as form-based error messages.

1. The success message must be presented to users.

The success message is normally **positioned above the form or as a replacement for the entire form**. If you're unsure, check with your UX team.

2. Focus must **shift to the success message**.

3. The success message must be set with `role="status"` as it is **not an alert or error**.

```
<div role="status">
  <h3>Sucess message heading alert</h3>
  <p>Success message content.</p>
</div>
```

4. The success message must also be set with `tabindex="0"` **so that the element can receive focus** as soon as it is triggered.

```
<div role="status" tabindex="0">
  <h3>Sucess message heading alert</h3>
  <p>Success message content.</p>
</div>
```

Any questions or comments?

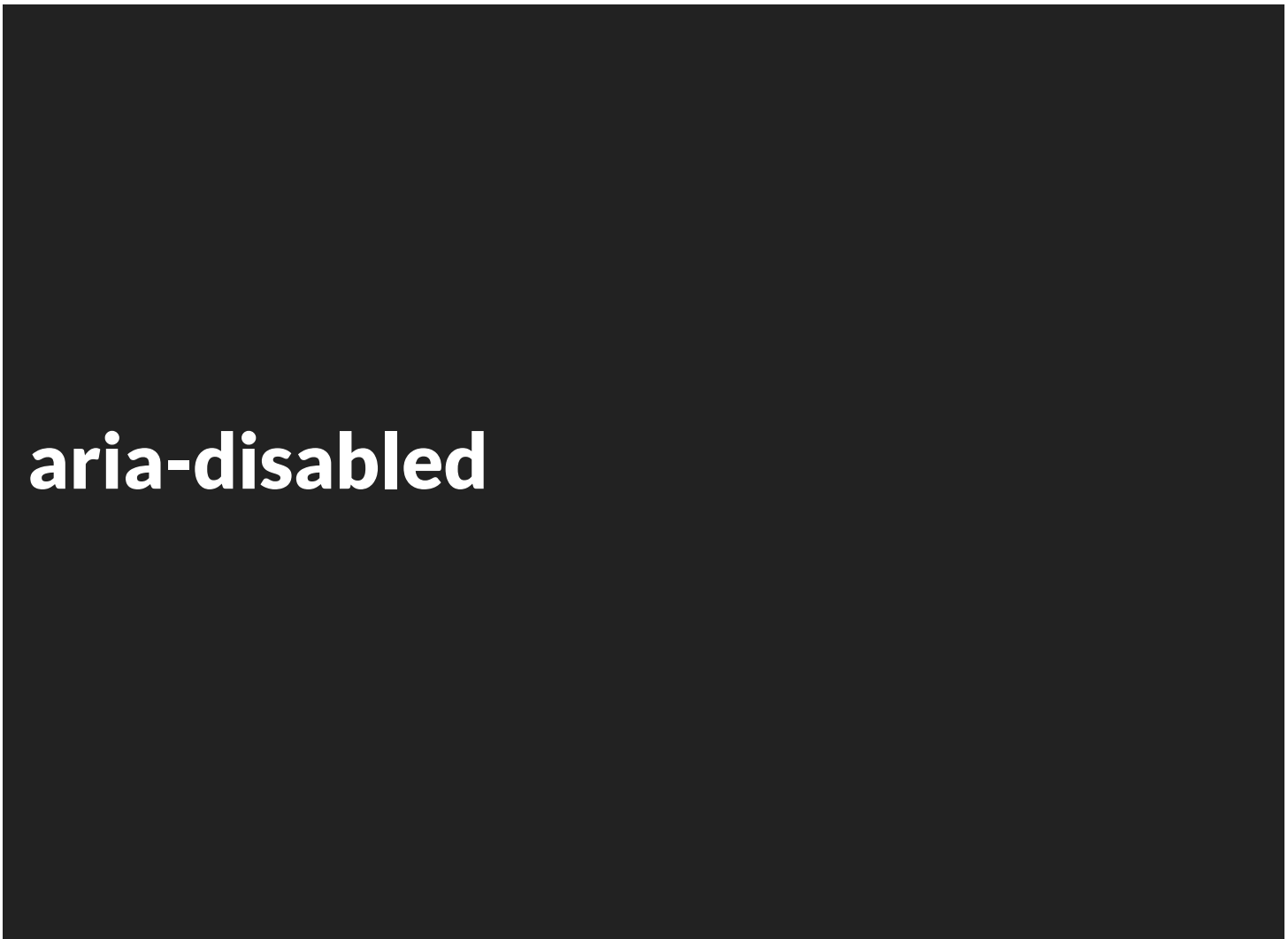
Exercise: Creating a success message

Accessing the exercise:

DEVELOPER EXERCISE: Creating a success message

ARIA attributes that are no longer needed

The following aria-attributes are **no longer needed for native form controls**.



The `aria-disabled` attribute **sets the disabled state on an element** in the accessibility tree only.

```
<button aria-disabled="true"></button>
<button aria-disabled="false"></button>
```

However, as the HTML `disabled` attribute is so well supported across assistive technologies, **there is no longer any need to use** the `aria-disabled` attribute.

```
<button disabled></button>
```

aria-required

The `aria-required` attribute **sets the required state on an element** in the accessibility tree only.

```
<input type="text" aria-required="true">
<input type="text" aria-required="false">
```

However, as the HTML `required` attribute is so well supported across assistive technologies, **there is no longer any need to use** the `aria-required` attribute.

```
<input type="text" required>
```

Any questions or comments?

Recap

When we give our form notifications their **correct roles and relationships**, it's easy for the accessibility API to adapt our messages to the user's needs.

Make it clear what type of information you're giving, and how it relates to the form as a whole or to a specific input.