Session 3

Accessible forms

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

Forms are important because it's how our site visitors can **interact with our products**.

They have the same access requirements as content and structure: all key form elements must be **identifiable in the Accessibility API**.

But they have **additional requirements** which are there to help with the interactions that are possible in a form.

People need to know if form controls are **required or optional**.

They need to know if there are any **instructions** related to each form control.

And they need to know when form controls are in **either success or error states**.

Today we'll first learn how to create **semantic, valid forms**.

Then we'll learn how to **add instructions and requirements**.

In the following session we'll cover what happens **after the user clicks Submit on the form**.

Exercise: An HTML element quiz

We're going to do a **quick quiz consisting of two questions**. Just type your answers into the chat window.

And remember, there is **no judgement here**. It's just a quick warm-up exercise!

Question 1:

List as many of the HTML `<input>` types as you can.

```html
<!-- Here are 2 examples of all 22 types -->
<input type="text">
<input type="radio">
```

Ready for the **full list**?

```
<input type="button">
<input type="checkbox">
<input type="file">
<input type="hidden">
<input type="image">
<input type="password">
<input type="radio">
<input type="reset">
<input type="submit">
<input type="text">
<input type="color">  (HTML5)
<input type="date">  (HTML5)
```

```
<input type="datetime-local">  (HTML5)
<input type="email">  (HTML5)
<input type="month">  (HTML5)
<input type="number">  (HTML5)
<input type="range">  (HTML5)
<input type="search">  (HTML5)
<input type="tel">  (HTML5)
<input type="time">  (HTML5)
<input type="url">  (HTML5)
<input type="week">  (HTML5)
<input type="datetime">  (Deprecated)
```

Question 2:

List as many form-related HTML elements as you can.

```
<!-- 2 examples of all 14 form elements -->
<input>
<label>
```

Ready for the **full list**?

```
<button>
<fieldset>
<form>
<input>
<label>
<legend>
<optgroup>
<option>
<select>
<textarea>
```

```
<datalist> (HTML5)
<meter> (HTML5)
<output> (HTML5)
<progress> (HTML5)
```

~~<keygen>~~ (Deprecated)

The <label> element

The `<label>` element can be used to provide **a visible text label** for some specific HTML elements.

There are only **five elements** that require a `<label>` element in order to provide a visible text label:

```html
<!-- Some INPUT elements -->
<label for="aaa">Label text</label>
<input id="aaa" type="text">

<input id="bbb" type="radio">
<label for="bbb">Yes</label>

<input id="ccc" type="checkbox">
<label for="ccc">Subscribe</label>
```

```html
<!-- SELECT -->
<label for="bbb">Label text</label>
<select id="bbb"></select>
```

```html
<!-- TEXTAREA -->
<label for="ccc">Label text</label>
<textarea id="ccc"></textarea>
```

```html
<!-- METER -->
<label for="ddd">Label text</label>
<meter id="ddd"></meter>
```

```
<!-- PROGRESS -->
<label for="eee">Label text</label>
<progress id="eee"></progress>
```

Misuse of the `<label>`

The `<label>` element **should never be applied** to other non-form related HTML elements.

```
<!-- Do not do this -->
<label>Choose a state</label>
<div>New South Wales</div>
```

The `<label>` element **should never be used as a replacement** for a `<legend>`.

```
<!-- Do not do this -->
<label>Overall question</label>
<ul>
  <li>
    <input type="radio" id="one" name="yesno">
    <label for="one">Yes</label>
  </li>
  <li>
    <input type="radio" id="two" name="yesno">
    <label for="two">No</label>
  </li>
</ul>
```

The `<label>` element **should never be used as a replacement** for heading element.

```
<!-- Do not do this -->
<label>Heading content</label>
<p>Paragraph content</p>
<p>Paragraph content</p>
```

Any questions or comments?

Programmatically associated `<label>` elements

The `<label>` element can also be used to **provide an accessible name** to form controls.

An accessible name is only generated if the `<label>` and form control are **programmatically associated**.

Step 1:

Apply a `for` attribute and a value to the `<label>`.

```
<label for="name">Full name</label>
<input type="text">
```

Step 2:

Apply a matching `id` value to the form control.

```
<label for="name">Full name</label>
<input type="text" id="name">
```

The form control will now have an **programmatically associated** `<label>`.

This will generate an “accessible name” **in the accessibility tree**.

Wrapped labels

It is acceptable to wrap the `<label>` **around the visible text label and form control**.

However, the `<label>` and form control must still be **programmatically associated** via matching `for` and `id` values.

```
<label for="name">
  Full name
  <input type="text" id="name">
</label>
```

Any questions or comments?

Exercise: Programmatically associating labels and form controls

Accessing the exercise:

DEVELOPER EXERCISE: Programmatically associating labels and form controls.

Visible text label order

Visible text labels for most fields should be positioned **immediately before the form control**, either to the left or above it.

```
<!-- Visible text label before input -->
<label for="name">Full name</label>
<input type="text" id="name">
```

```html
<!-- Visible text label before textarea -->
<label for="comment">Comment</label>
<textarea id="comment"></textarea>
```

```html
<!-- Visible text label before select -->
<label for="fruit">Choose a fruit</label>
<select id="fruit"></select>
```

Visible text labels for radio buttons and checkboxes should be positioned **immediately after the field**.

```
<!-- Visible text label after radio button -->
<input type="radio" id="yes">
<label for="yes">Yes</label>
```

```html
<!-- Visible text label after checkbox -->
<input type="checkbox" id="subscribe">
<label for="subscribe">Subscribe</label>
```

These positions are defined because they are the usual (and therefore most predictable) position for visible text labels in relation to form controls.

What about when the `<label>` is wrapped around the form control?

In these cases, the visible text label content **should still be positioned before** the form control.

```
<label for="name">
  Full name
  <input type="text" id="name">
</label>
```

```
<label for="comment">
  Comment
  <textarea id="comment"></textarea>
</label>
```

```
<label for="fruit">
  Choose a fruit
  <select id="fruit"></select>
</label>
```

For radio buttons and checkboxes, the visible text label **should still be positioned after** the form control.

```html
<label for="yes">
  <input type="radio" id="yes">
  Yes
</label>
```

```html
<label for="subscribe">
  <input type="checkbox" id="subscribe">
  Subscribe
</label>
```

Any questions or comments?

Exercise: Fixing the label order

Accessing the exercise:

DEVELOPER EXERCISE: Fixing the label order.

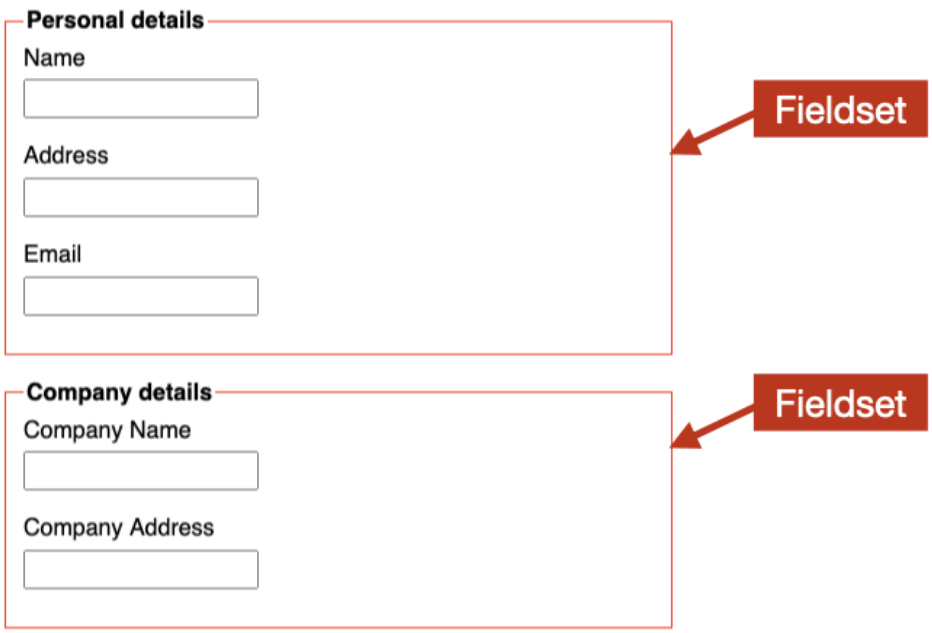
`<fieldset>` and `<legend>`

The `<fieldset>`

The `<fieldset>` element represents a set of form controls **grouped under a common name**.

```
<fieldset>
  <legend>Contact details</legend>
  ...
</fieldset>
```

The `<fieldset>` element allows us to **visually and semantically group related form questions**.



We're allowed to **nest more than one** `<fieldset>` element inside another `<fieldset>` element.

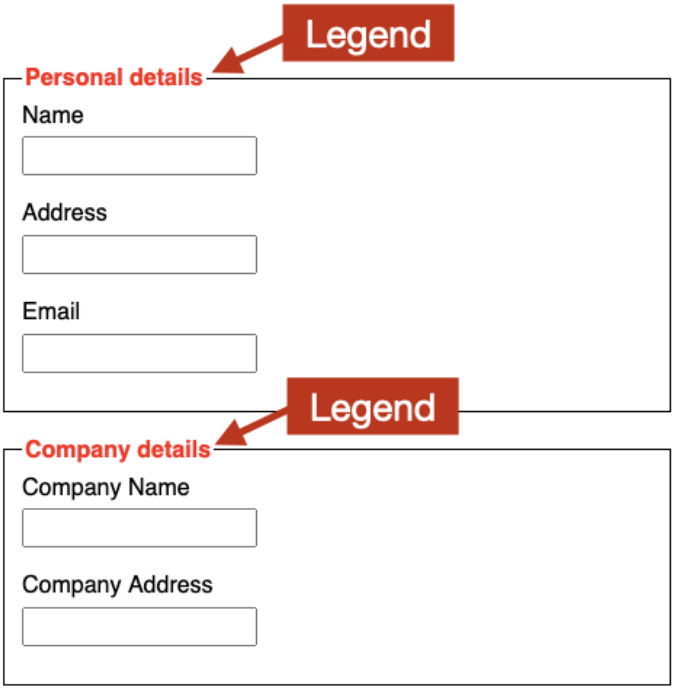
```
<fieldset>
  <legend>Contact details</legend>
  ...
  <fieldset>
    <legend>Additional info</legend>
    ...
  </fieldset>
</fieldset>
```

However, while this is considered valid markup, screen readers **often struggle to describe nested** `<fieldset>` elements effectively.

The `<legend>`

The `<legend>` element **represents a caption** for the `<fieldset>` element.

```
<fieldset>
  <legend>Contact details</legend>
  ...
</fieldset>
```



There **should be** a single `<legend>` inside each `<fieldset>` element.

If present, the `<legend>` element **must be** the first child of any `<fieldset>` element.

```
<!-- LEGEND must be first child -->
<fieldset>
  <p>Some other content</p>
  <legend>Incorrect legend</legend>
</fieldset>
```

The `<legend>` element **must not be** used outside of `<fieldset>` element.

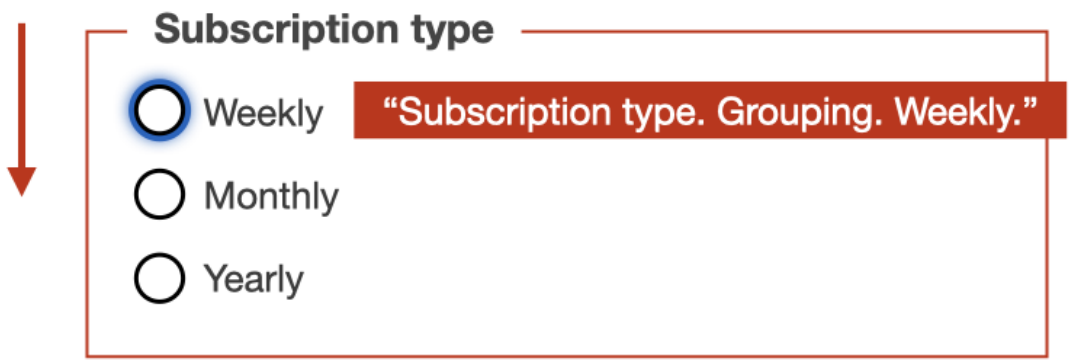
```
<!-- LEGEND should not be used outside FIELDSET --
<p>Some other content</p>
<legend>Legend used outside of FIELDSET</legend>
<p>Some other content</p>
```

As of HTML5, `<legend>` elements are **allowed to contain heading elements**.

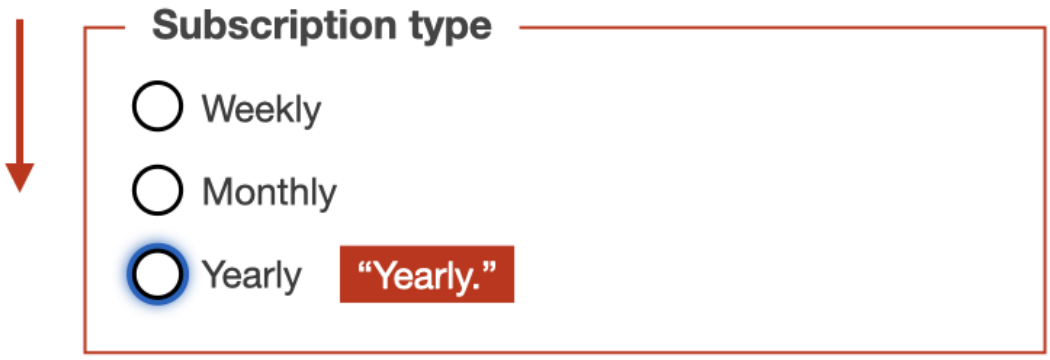
```
<fieldset>
  <legend><h3>Contact details</h3></legend>
  ...
</fieldset>
```

Why are the `<fieldset>` and `<legend>` elements important for accessibility?

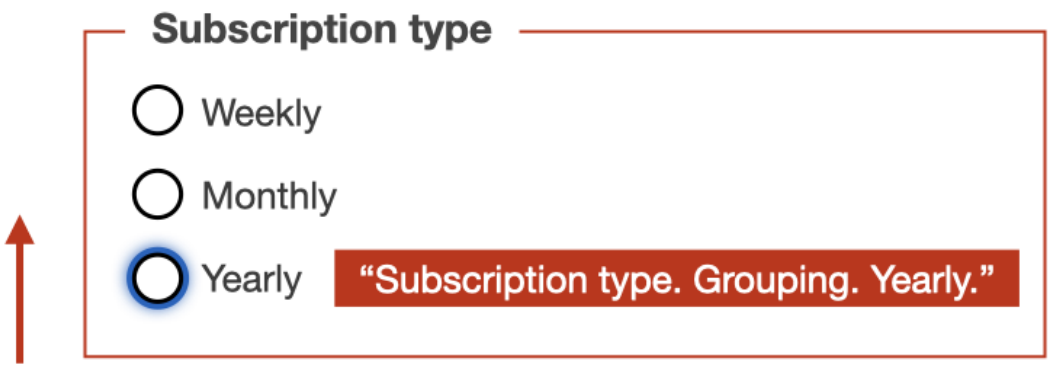
When focus moves to the first form control inside a `<fieldset>`, **screen readers will announce the** `<legend>` content along with the label content.




Subscription type
Weekly
Monthly
"Monthly."
Yearly


Subscription type
Weekly
Monthly
Yearly
"Yearly."

The same happens if the user **moves back up the form**.




Subscription type
Weekly
Monthly
"Monthly."
Yearly


Subscription type
Weekly
"Weekly."
Monthly
Yearly

Let’s look at how the `<legend>` is **announced in various screen reader combinations** when focus is placed on the `<input>` element.

```
<fieldset>
  <legend>Contact details</legend>
  <div>
    <label for="name">Full name</label>
    <input type="text" id="name">
  </div>
</fieldset>
```

OSX / VoiceOver

- **Chrome**: Full name. Edit text. **Contact details**. Group.
- **Firefox**: Full name. Edit text. **Contact details**. Group.
- **Safari**: Full name. Edit text. **Contact details**. Contact details. Group.

Windows / NVDA

- **Chrome**: **Contact details**. Grouping. Full name. Edit. Blank.
- **Firefox**: **Contact details**. Grouping. Full name. Edit. Has autocomplete. Blank.
- **Edge**: **Contact details**. Grouping. Full name. Edit. Blank.

Windows / JAWS

- **Chrome**: **Contact details**. Group. Full name. Edit. Type in text.
- **Firefox**: **Contact details**. Full name. Edit. Type in text.
- **Edge**: **Contact details**. Group. Full name. Edit. Type in text.

Radio and checkbox groups

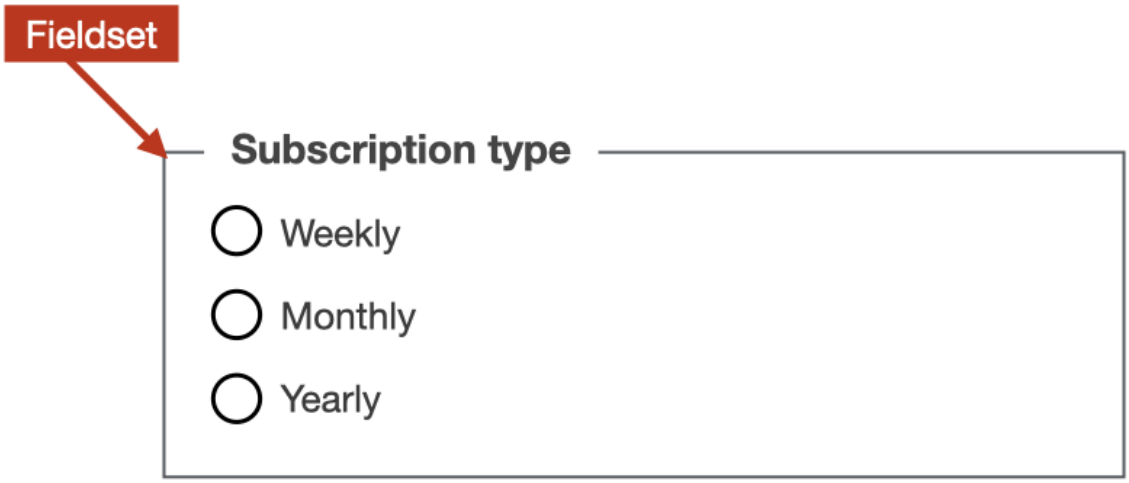
Groups of related radio buttons are called **“radio groups”**.

All radio buttons must have **matching** `name` **values** in order for a single radio button to be chosen at a time.

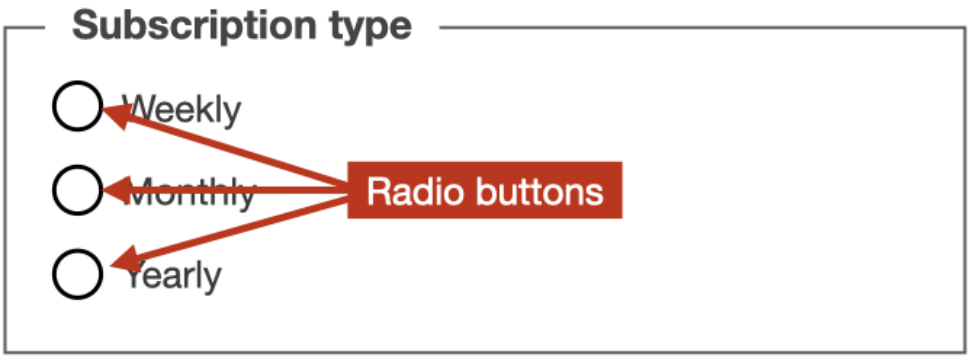
```html
<input id="weekly" type="radio" name="sub">
<label for="weekly">Weekly</label>

<input id="monthly" type="radio" name="sub">
<label for="monthly">Monthly</label>

<input id="yearly" type="radio" name="sub">
<label for="yearly">Yearly</label>
```

The `<fieldset>` and `<legend>` elements **should be used for radio groups** in order to provide an overall name for the group.

```html
<fieldset>
  <legend>Subscription type</legend>
  <input id="weekly" type="radio" name="sub">
  <label for="weekly">Weekly</label>

  <input id="monthly" type="radio" name="sub">
  <label for="monthly">Monthly</label>

  <input id="yearly" type="radio" name="sub">
  <label for="yearly">Yearly</label>
</fieldset>
```

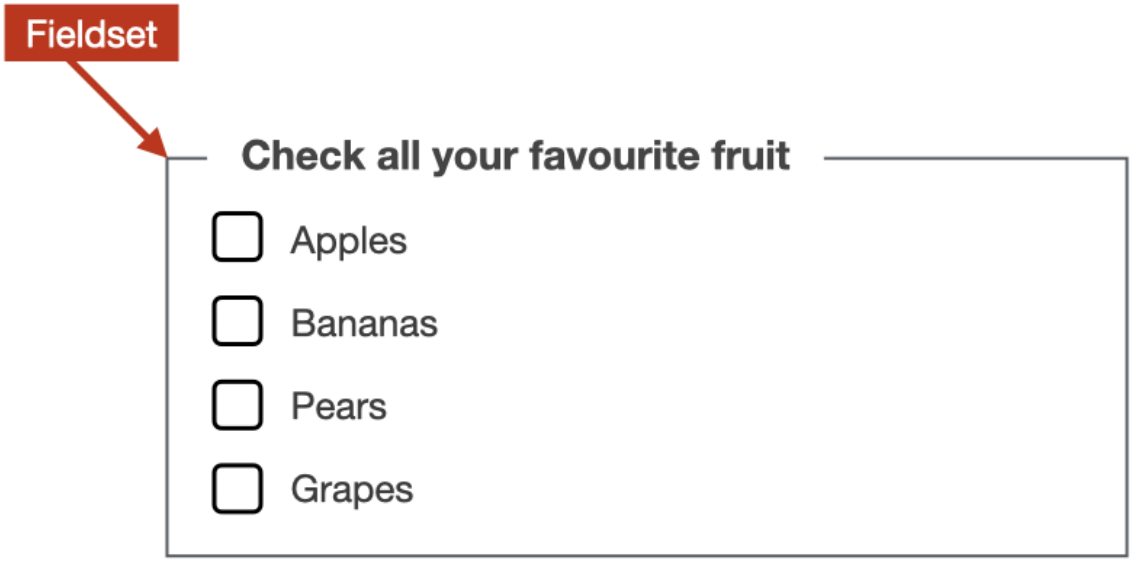



Legend
Subscription type
Weekly
Monthly
Yearly

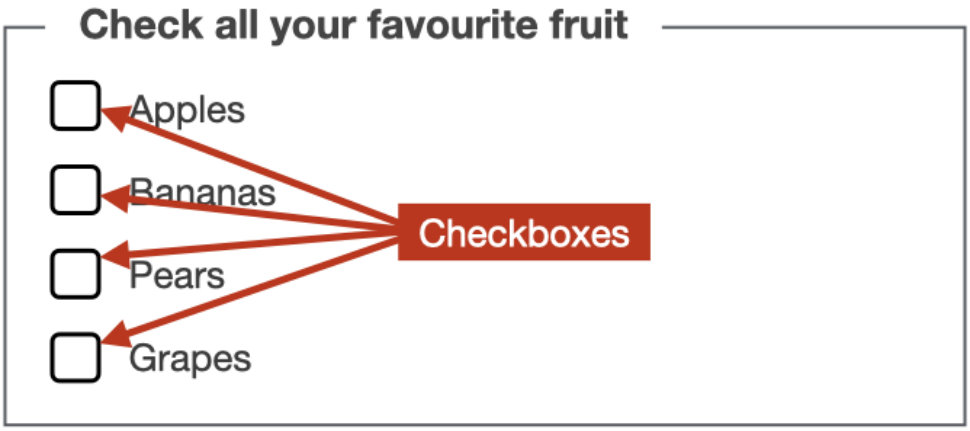

Subscription type
Weekly
Monthly
Yearly
Radio buttons

The `<fieldset>` and `<legend>` elements **should be used for checkbox groups** as well.

```html
<fieldset>
  <legend>Check all your favourite fruit</legend>
  <input id="apples" type="radio">
  <label for="apples">Apples</label>

  <input id="bananas" type="radio">
  <label for="bananas">Bananas</label>

  <input id="pears" type="radio">
  <label for="pears">Pears</label>
</fieldset>
```




Legend
Check all your favourite fruit
Apples
Bananas
Pears
Grapes


Check all your favourite fruit
Apples
Bananas
Pears
Grapes
Checkboxes

Lists for radio and checkbox groups

Individual radio buttons and their associated `<label>` elements can be **wrapped in `<div>` elements**.

```
<fieldset>
  <legend>Do you like boats?</legend>
  <div>
    <input type="radio" id="boats-y" name="boats">
    <label for="boats-y">Yes</label>
  </div>
  <div>
    <input type="radio" id="boats-n" name="boats">
    <label for="boats-n">No</label>
  </div>
</fieldset>
```

They can also be **placed inside a list**.

```
<fieldset>
  <legend>Do you like boats?</legend>
  <ul>
    <li>
      <input type="radio" id="boats-y" name="boats
      <label for="boats-y">Yes</label>
    </li>
    <li>
      <input type="radio" id="boats-n" name="boats
      <label for="boats-n">No</label>
    </li>
  </ul>
</fieldset>
```

Any questions or comments?

Exercise: Making a radio group accessible

Accessing the exercise:

DEVELOPER EXERCISE: Making a radio group accessible.

Form field instructions

Form field instructions are additional information that can be **added to a form field to provide more context for users**.

These instructions can also be used to **specify the exact input or formatting** required for each field.

Form field instructions must be **programatically associated with each form field**, so that this information is announced to assistive technologies.

The best way to **programatically associate instructions** with their form fields is via matching `aria-describedby` and `id` attributes.

```
<div>
  <label for="a1">Name</label>
  <span id="a2">Include full name</span>
  <input aria-describedby="a2" id="a1" type="text"
</div>
```

Instructions associated with fieldsets

In the case of `<fieldset>` elements, there may be occasions where the entire `<fieldset>` group **needs additional instructions**.

Ideally, the instructions would be programmatically associated **directly with the `<fieldset>` element** using `aria-describedby`.

```html
<fieldset aria-describedby="a1">
  <legend>Do you like boats?</legend>
  <p id="a1">Choose at least one option</p>
  <div>
    <input type="radio" id="b1" name="b">
    <label for="b1">Yes</label>
  </div>
  <div>
    <input type="radio" id="b2" name="b">
    <label for="b2">No</label>
  </div>
</fieldset>
```

However, this method has very poor support in VoiceOver.

An alternate, and slightly more robust, method would be to programmatically associate the instructions **with each form control** using `aria-describedby`.

```
<fieldset>
  <legend>Do you like boats?</legend>
  <p id="a1">Choose at least one option</p>
  <div>
    <input aria-describedby="a1" type="radio" id="
    <label for="b1">Yes</label>
  </div>
  <div>
    <input aria-describedby="a1" type="radio" id="
    <label for="b2">No</label>
  </div>
</fieldset>
```

Any questions or comments?

Exercise: Adding form field instructions

Accessing the exercise:

DEVELOPER EXERCISE: Adding form field instructions.

Required fields

All “required” fields within a form **must be programmatically identifiable**. This is achieved by applying the `required` attribute.

```
<label for="name">
  Full name
</label>
<input type="text" id="name" required>
```

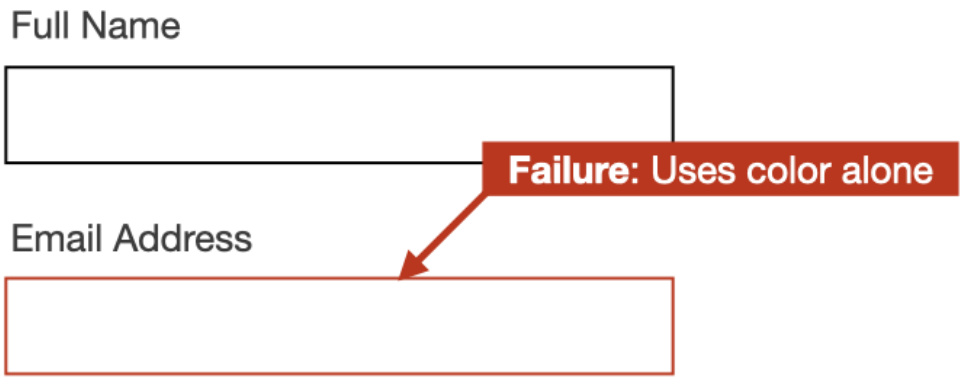
This means that for screen readers, **the “required” state will be announced**.

Visually flagging required fields

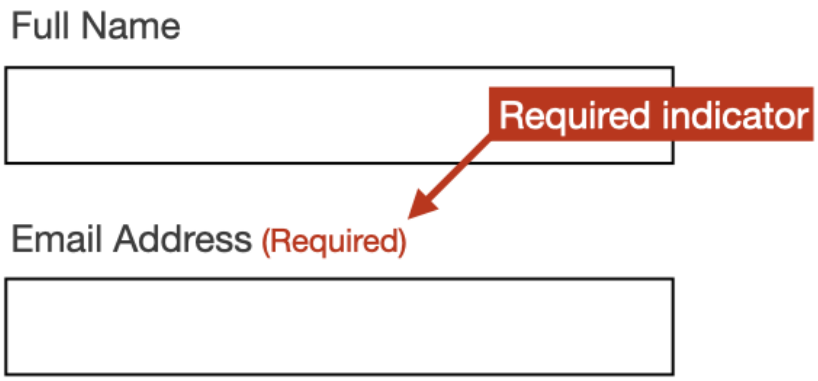
Any required fields **must also have some sort of visual indicator**, so that sighted users know that these fields must be filled in.

The **visual indication method** must be:

- Intuitive for users.
- More than a colour difference.



One simple solution is to add the text “(Required)” to the `<label>` element content. This additional content can also be **styled in a colour to make it stand out**.




```
<label for="name">
  Full name
  <span>(Required)</span>
</label>
<input type="text" id="name" required>
```

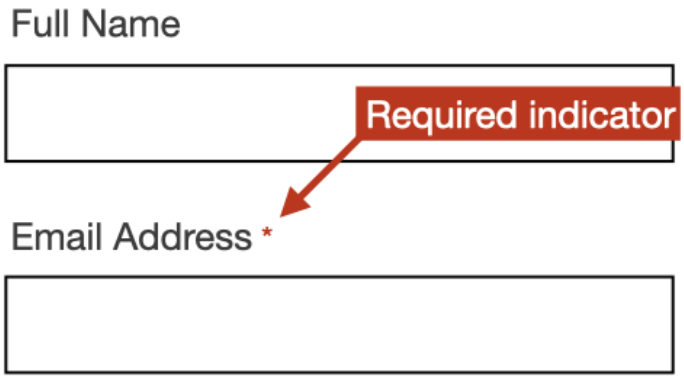
This “(Required)” text should be set with `aria-hidden="true"`, to **hide this additional information** from screen readers.

Screen readers are informed that the field is required via the `required` attribute, **so they don’t need to hear this message** from the `<label>` as well.

```
<label for="name">
  Full name
  <span aria-hidden="true">(Required)</span>
</label>
<input type="text" id="name" required>
```

The asterisk symbol

In the old days, some authors **used the asterisk symbol** to visually define required fields.



However, this method **presents a range of accessibility issues**.

1. Users often **do not understand the meaning of the asterisk**, so a key would need to be presented at the top of the form.

2. The asterisk is also **an extremely small indicator**, and can be missed by users with poor eye-sight.

3. The asterisk symbol is often **ignored or mis-announced by screen readers**.

What about when all fields are required?

If **all fields in the form are required**, users can be presented with a message along the lines of:
"All fields in the form below are required"

All fields in the form below are required.

Full Name

Email Address

This message **must be placed before the form** so that assistive technology users can read this information before entering the form.

Using this method, individual form fields **do not have to be visually identified** as “required”.

However, if this method is used, **all required fields** must still include the required attribute.

```
<label for="name">
  Full name
</label>
<input type="text" id="name" required>
```

What about when there's a mix of required and optional?

Only required fields **need to be flagged to users**. So, there are two options:

Option 1:

Visually flag all required fields, even if most of the fields in the form are required.

Option 2:

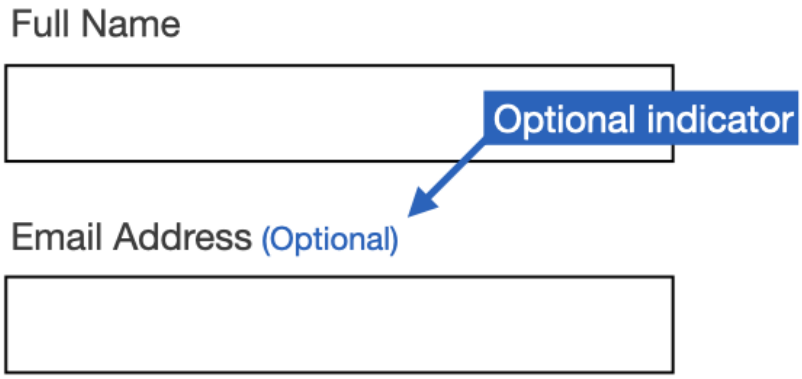
Visually flag just the optional fields. A message would need to be presented at before the form, along the lines of:
"All fields in the form below are required unless marked OPTIONAL".

All fields in the form below are required unless marked OPTIONAL

Full Name

Email Address

Then all optional fields **would need some sort of visual flag**:



If this method is used, **all required fields must still include** the `required` attribute.

```
<label for="name">
  Full name
</label>
<input type="text" id="name" required>
```

Any questions or comments?

Exercise: Adding required features

Accessing the exercise:

DEVELOPER EXERCISE: Adding required features.

The `autocomplete` attribute

The `autocomplete` attribute lets web browsers **provide automated assistance** when people are filling in form fields.

```
<form>
  <div class="form-group">
    <label for="name">Full name</input>
    <input type="text" id="name" autocomplete="on"
  </div>
</form>
```

This attribute **can be applied to**:

- `<input>` elements that take text or numeric values.
- `<textarea>` elements.
- `<select>` elements.
- `<form>` elements.

In order to use the `autocomplete` attribute, **the following need to be true**:

- The element must be inside a `<form>`.
- The form must have a submit button.
- The element must have a `name` and/or an `id`.

The autocomplete attribute can take a wide range of values. The most common values are:

Attribute	Purpose
`off`	The browser is not permitted to automatically provide a value.
`on`	The browser is allowed to automatically complete the input.
`name`	The field expects the value to be a person's full name.
`email`	The field expects the value to be a person's email address.
`username`	A username or account name.
`tel`	A full telephone number, including the country code.

Why is the `autocomplete` attribute important for accessibility?

The autocomplete attribute helps users by reducing the amount of repetitive typing that is needed when filling in forms.

This is especially important for people with mobility issues or who have low vision.

Exercise: Adding the `autocomplete` value

Accessing the exercise:

DEVELOPER EXERCISE: Adding the autocomplete value.

Recap

Accessible forms are **critical for all users**, but especially keyboard and screen reader users.

We need to make sure our form controls have programatically associated **labels, instructions and required information**.