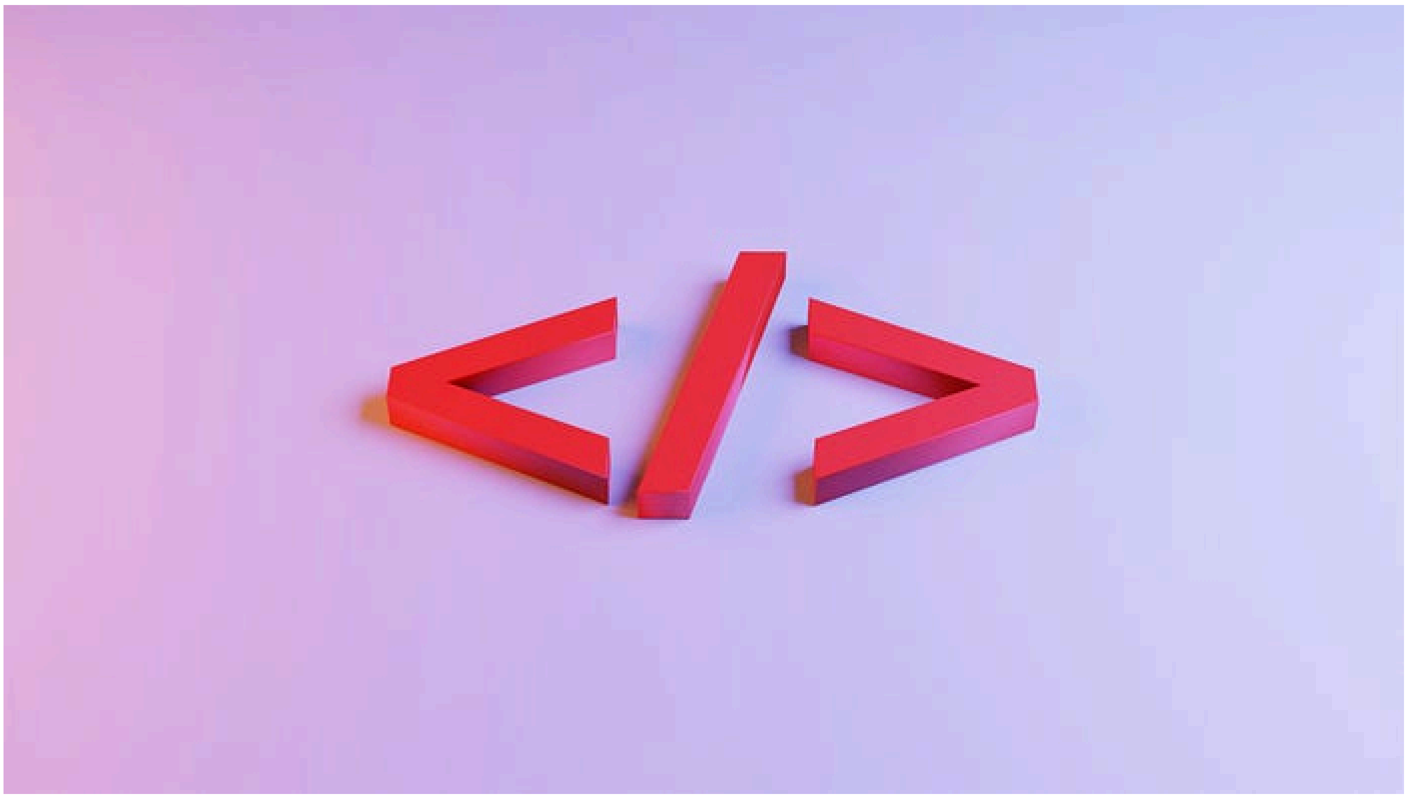
< Go to the original



Accessible form validation from scratch— the form's basic structure

Part 2: The Markup

Daniel Berryhill

Follow

UX Collective a11y-light ~12 min read · May 5, 2024 (Updated: May 7, 2024) · Free: No

This is Part 2 in my series on creating accessible form validation from scratch. In this article, we'll create the form in HTML.

The previous article: Accessible form validation from scratch — Part 1: Requirements

Table of Contents

Introduction

I'll go through these one at a time, explain the markup, and test them with assistive technology (AT). For brevity, I'll be using JAWS and NVDA — though keep in mind, screen readers are not the only form of AT.

A quick note on styles — this is the only styling I'll put in the examples for this article:

```
* {
/* Labels (by default) will be above
their corresponding controls */
  display: block;
}

/* Labels of checkboxes and radio buttons
will be inline with their
corresponding controls */
input[type="checkbox"],
input[type="radio"]
{
  display: inline;
}
```

I'm only doing this to keep the rendered controls from being bunched up and difficult to read. I'll implement further styling in a future article.

Alright, let's give this form some bones.

The <form> element

Let's start with the form:

Copy

```
<form id="formNewUser"
  action="#" method="post">

</form>
```

Nothing special, really. I'm just using these values for `action` and `method` because I'm not going to be communicating with a server for the final solution. That will be somewhat simulated when we get to the JavaScript step in a later article.

I use an `id` attribute because a form is an interactive element — and you should always give interactive elements an `id` attribute. This isn't an accessibility thing, just a web development thing.

Adding text boxes

Let's add a few text boxes. Keep in mind that this won't be the final markup for the end product. We're moving through this incrementally. To keep the code sections smaller, I'll only include the new markup in the examples. I'll bring it all together in the end.

The markup for text boxes

Forgive the use of so many new lines. Reading codes snippets for these articles on mobile devices often results in horizontal scrolling. This is my attempt to mitigate that.

```
</label>
<input type="text" id="textFirstName"
  name="given-name" required >

<label for="textEmail">Email</label>
<input type="email" id="textEmail"
  name="email" required>

<label for="textPassword">Password</label>
<input type="password" id="textPassword"
  name="newPassword" autocomplete="off"
  required>

<label for="textConfirmPassword">
  Confirm Password</label>
<input type="password"
  id="textConfirmPassword"
  name="newConfirmPassword"
  autocomplete="off" required>
```

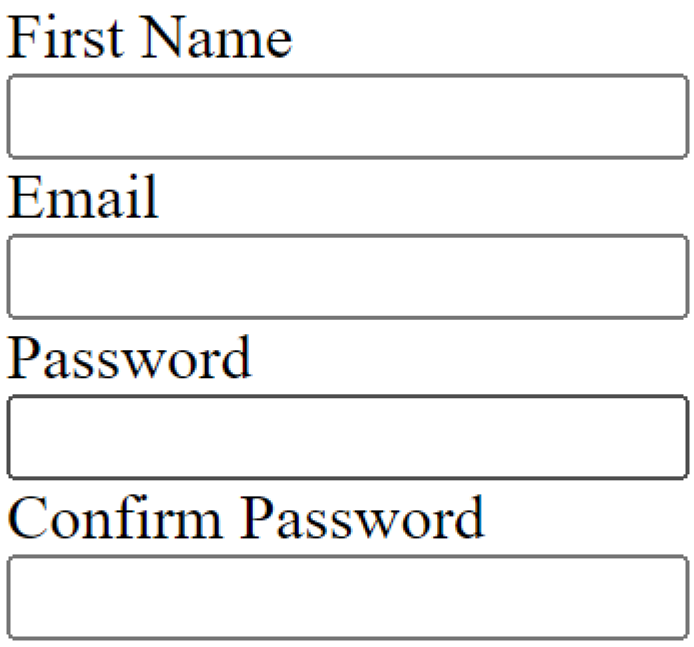
A few things to note:

- Each `<label>` has a `for` attribute that programmatically links it to the text box. Alternatively, you can wrap the text box in the `<label>` element and skip the `for` attribute altogether.
- The `name` attributes for the "Email" and "First Name" fields facilitate the browsers "autofill" functionality. The `autocomplete` behavior (a browser providing suggestions for fields when focused) is on by default on all browsers. So, you don't have to have an `autocomplete="on"` attribute. Just use the `name` attribute and store the applicable autofill value. Look at the following link to determine the value: HTML spec on autofill.
- I set `autocomplete` to "off" so that the browser would not offer suggestions for passwords, since we're using a new password and not entering an existing one.
- The `name` attributes for all fields are for the server, which we won't be getting into much in this series — I'm just implementing a good practice. Generally speaking, think of the `id` attribute as the key for the client-side and the `name` attribute as the key for the server-side.

you're grouping controls, as you'll read later when we get to checkboxes and radio buttons.

How the text boxes render

Here's how it renders:



Again, the only relevant style here is that each element has the style `display: block`. Otherwise, they'd all appear from left to right, only going to the next line when it reaches the edge of the window.

Assistive Technology check for text boxes

Here's what JAWS outputs when the "First Name" field receives focus (the text in square brackets, such as "[Tab]", indicates the user input — in this case, the Tab key):

> First Name edit required type in text [Tab] Email edit required type in text [Tab] Password password edit required type in text [Tab] Confirm Password password edit required type in text

And here's what NVDA outputs:

> edit protected required blank

JAWS (since last I saw) doesn't announce the autocomplete (also called "autofill") functionality.

You may notice that NVDA didn't announce "autocomplete" for the "First Name" field. That's because my browser doesn't have that value stored under `given-name`. If I were to change `given-name` to `name`, it would announce autocomplete and the browser would display my first and last name as an option to populate the text box. Your experience may be different.

Also notice that for the password fields, NVDA states the password fields are "protected". This is due to the `type="password"` attribute that masks the user's input. JAWS uses "password" to communicate the same thing.

Adding a <select> control

Now, we'll add a `<select>` control. We use a `<select>` control for a list of mutually exclusive choices. If there are only 2 to 4 choices, you should probably opt for radio buttons. The `<select>` control is better for a list of choices that are 5 or greater (though I'm sure there's debate about that number).

But, of course, if you don't know the number of choices ahead of time (i.e., you're populating it with values from a data source), go with a `<select>` control.

The markup for the select control

Copy

```
<label for="selectFavoriteColor">
  Favorite Color</label>
<select id="selectFavoriteColor"
  name="selectFavoriteColor" required>
  <option value="optionNotSelected">
    [Not Selected]</option>
  <option value="optionRed">Red</option>
```

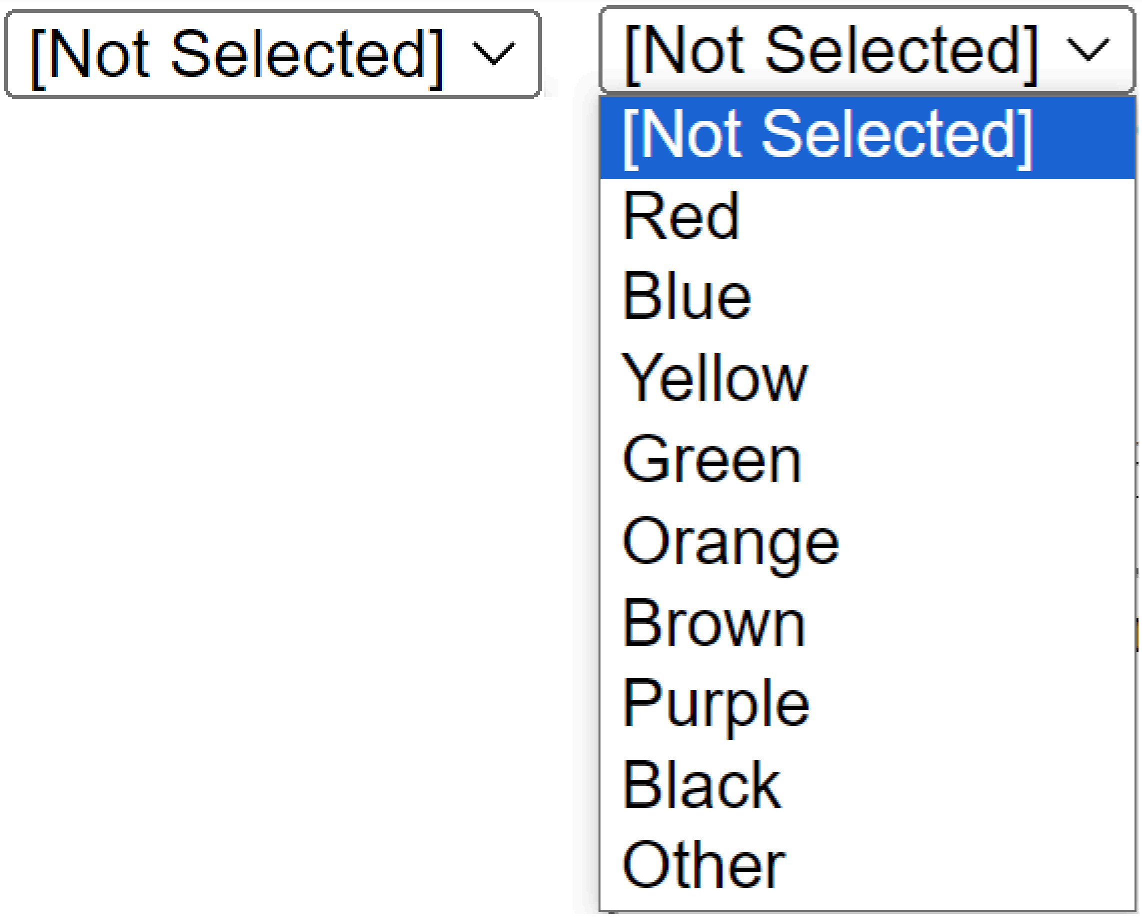
```
    <option value="optionOrange">Orange</option>
    <option value="optionBrown">Brown</option>
    <option value="optionPurple">Purple</option>
    <option value="optionBlack">Black</option>
    <option value="optionOther">Other</option>
</select>
```

Let's go through it:

- As with the text boxes, we use the `for` attribute to programmatically assign the label to the `<select>` control.
- It has a unique `id` attribute value for assigning the control to the label and for JavaScript selectors.
- It has a unique `name` attribute value for the server-side.
- Most will leave the `value` for the "[Not Selected]" `<option>` element as blank (""), but I prefer to be more explicit. Besides, we're not going to be using HTML5's native validation for this control.
- I'm using the verbiage "Not Selected" instead of an empty string or "NA" or "- -" because, again, I want to be explicit. Remember, not everyone uses their eyes to consume web content.

How the <select> control renders

Here's how it renders with the collapsed `<select>` control on the left and the expanded one on the right:



Assistive Technology check for the <select> control

We'll focus, expand, and choose "Green".

For JAWS:

> Favorite Color combo box required left bracket Not Selected right bracket To change the selection use the arrow keys [Space Bar] Space expanded required list required with 10 items [Down Arrow key] Red [Down Arrow key] Blue [Down Arrow key] Yellow [Down Arrow key] Green [Enter key] Enter Favorite Color combo box required Green

For NVDA:

> Favorite Color combo box Not Selected collapsed required [Space Bar] expanded list Not Selected 1 of 10 [Down Arrow key] Red 2 of 10 [Down Arrow key] Blue 3 of 10 [Down Arrow key] Yellow 4 of 10 [Down Arrow key] Green 5 of 10 [Enter key] Favorite Color combo box Green collapsed required

Adding checkboxes and radio buttons

When your controls are related, they should be programmatically grouped. Otherwise, an assistive technology user may not know that the controls are related to one another.

Whenever possible, you should use the `<fieldset>` element to group controls. The first child is the `<legend>` element, which serves as the label for the grouped elements.

We use checkboxes when selecting more than one option is possible/permitted, and radio buttons when only one is.

The markup for checkboxes and radio buttons

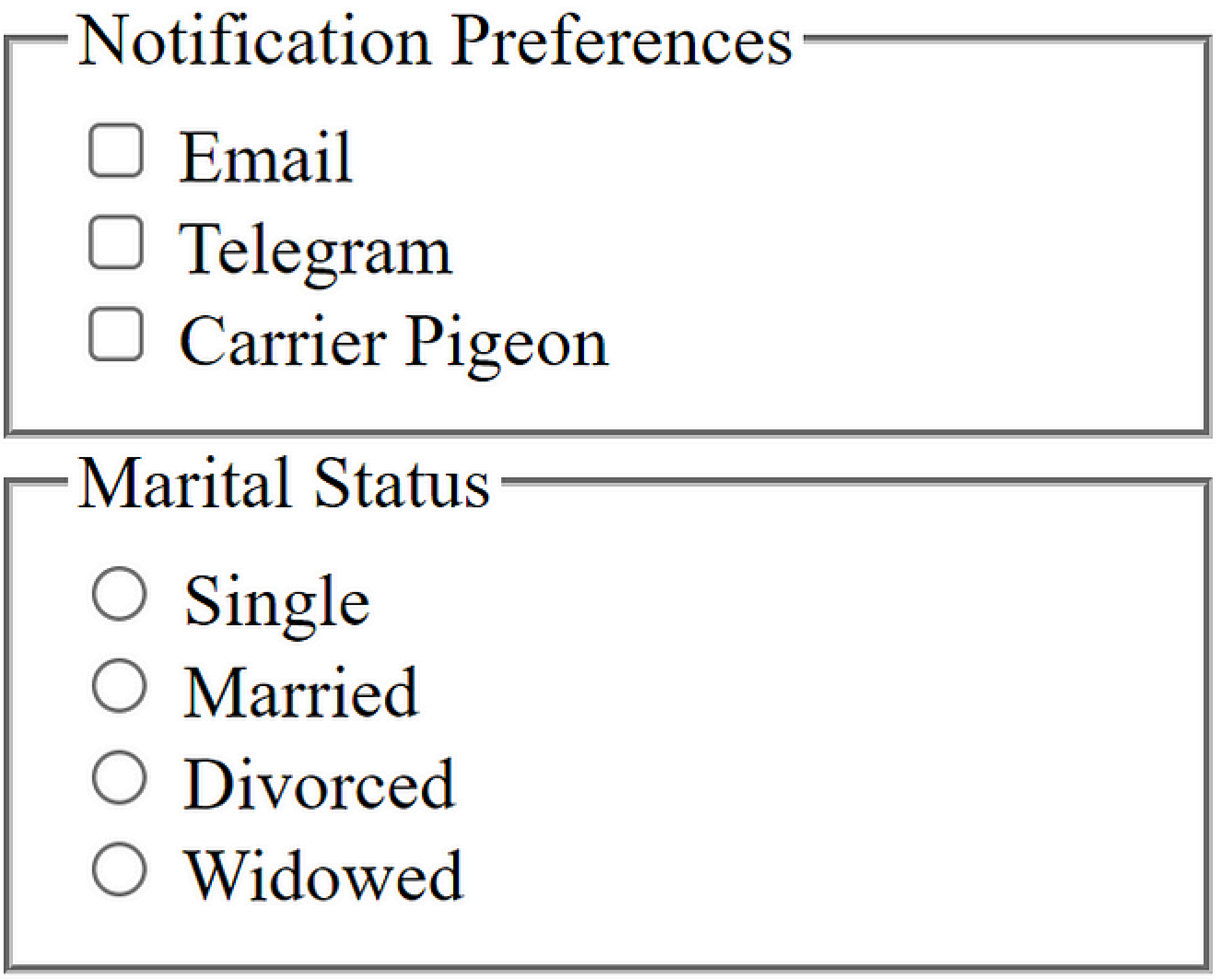
Let's add some checkboxes and radio buttons.

Copy

```
<fieldset>
    <legend>Notification Preferences</legend>
    <label>
      <input type="checkbox" id="checkEmail"
        name="notification[]" value="email"
        required> Email
    </label>
    <label>
      <input type="checkbox"
        id="checkTelegram"
        name="notification[]"
        value="telegram"> Telegram
    </label>
    <label>
      <input type="checkbox"
        id="checkCarrierPigeon"
        name="notification[]"
        value="pigeon"> Carrier Pigeon
    </label>
</fieldset>

<fieldset>
    <legend>Marital Status</legend>
    <label>
    <label>
      <input type="radio" name="color"
        value="married"> Married
    </label>
    <label>
      <input type="radio" name="color"
        value="divorced"> Divorced
    </label>
    <label>
      <input type="radio" name="color"
        value="widowed"> Widowed
    </label>
</fieldset>
```

Notes on these elements:

- As stated earlier, the `<legend>` should be the first child of the `<fieldset>`.
- The `required` attribute is included in the first checkbox and radio button. Because all share the same `name` attribute value, that signifies that the user has to select one of the checkboxes/radio buttons — not necessarily the one that has the `required` attribute.
- The `name` attribute is used for the checkboxes so that the inputs from the user can be easily read on the server-side or in JavaScript. In other words, I don't have to get the value of `"checkEmail"` and its sister checkboxes individually. I can just get an array of the checked checkboxes.
- Having square brackets (`[]`) at the end of the `name` attribute for the checkboxes facilitates sending the checked values to the server or to JavaScript. If you use jQuery, you may have to do some character escape magic for some of your selectors.
- For the radio buttons, the `name` attribute groups them to each other to ensure they are mutually exclusive (e.g., you cannot select "Red" and "Blue" as your favorite color).
- In this example, I wrap the radio buttons and checkboxes in the `<label>` element. Alternatively, you can be explicit (as I was with the text boxes) and use the `for` attribute.



Assistive Technology check for checkboxes and radio buttons

For this example, we'll check "Carrier Pigeon" and "Divorced".

For JAWS:

> Notifications Preferences group Email checkbox not checked required invalid entry to check press Space Bar [Tab] Telegram checkbox not checked [Tab] Carrier Pigeon checkbox not checked [Space Bar] Space checked [Tab]

> Marital Status group Single radio button not checked required invalid entry [Down Arrrow key] Married radio button not checked [Down Arrrow key] Divorced radio button not checked [Down Arrrow key] Widowed radio button not checked [Up Arrow key] Divorced radio button not checked [Space Bar] Space Divorced radio button checked

> Notification Preferences grouping Email check box not checked [Tab] Telegram check box not checked [Tab] Carrier Pigeon check box not checked [Space Bar] checked [Tab]

> Marital Status grouping Single radio button not checked required invalid entry 1 of 4 [Down Arrrow key] Married radio button checked 2 of 4 [Down Arrrow key] Divorced radio button checked 3 of 4 [Down Arrrow key] Widowed radio button checked 4 of 4 [Up Arrow key] Divorced radio button checked 3 of 4

Notice that when the radio buttons first receive focus, the first radio button is selected but not checked. For NVDA, when you press the down arrow key, it gives focus to the second radio button and "checks" it. It will also "check" any radio button in that grouping that receives focus while you're still on the page.

JAWS doesn't do this. Instead it moves from one item to the next without checking.

As far as I know, unless you're using a screen reader or some other AT, there's no keyboard shortcut that allows you to prevent selection as you navigate radio buttons. If you're using JAWS, that key is the "A" key; if you're using NVDA, it's the "R" key. (JAWS Source, NVDA Source)

Also, notice that both read "invalid entry" for the first checkbox/radio button. That's because the first item is marked as `required`, and since none of the items in that set (those that share the same `name` value) are checked, the AT is giving us native HTML validation.

Adding a <button>

We have a form — so we need a submit button. Let's add it.

Copy

What? Were you expecting more?

AT and default behavior

The `<button>` element is `type="submit"` by default; so it doesn't need that attribute. I gave it an `id` because all interactive elements should have one.

As far as AT, NVDA and JAWS both output "Create Account button" (JAWS adds "to activate press Enter").

Later, we'll likely prevent the default behavior in JavaScript, but this is enough for now.

Button verbiage

Notice that the button's label ("Create Account") says exactly what the button is intended to do. I don't use "Submit" or "Done" or "Go". Start with a verb and state what the button does.

Also, remember that the submit button should be inside the `<form>`, not outside.

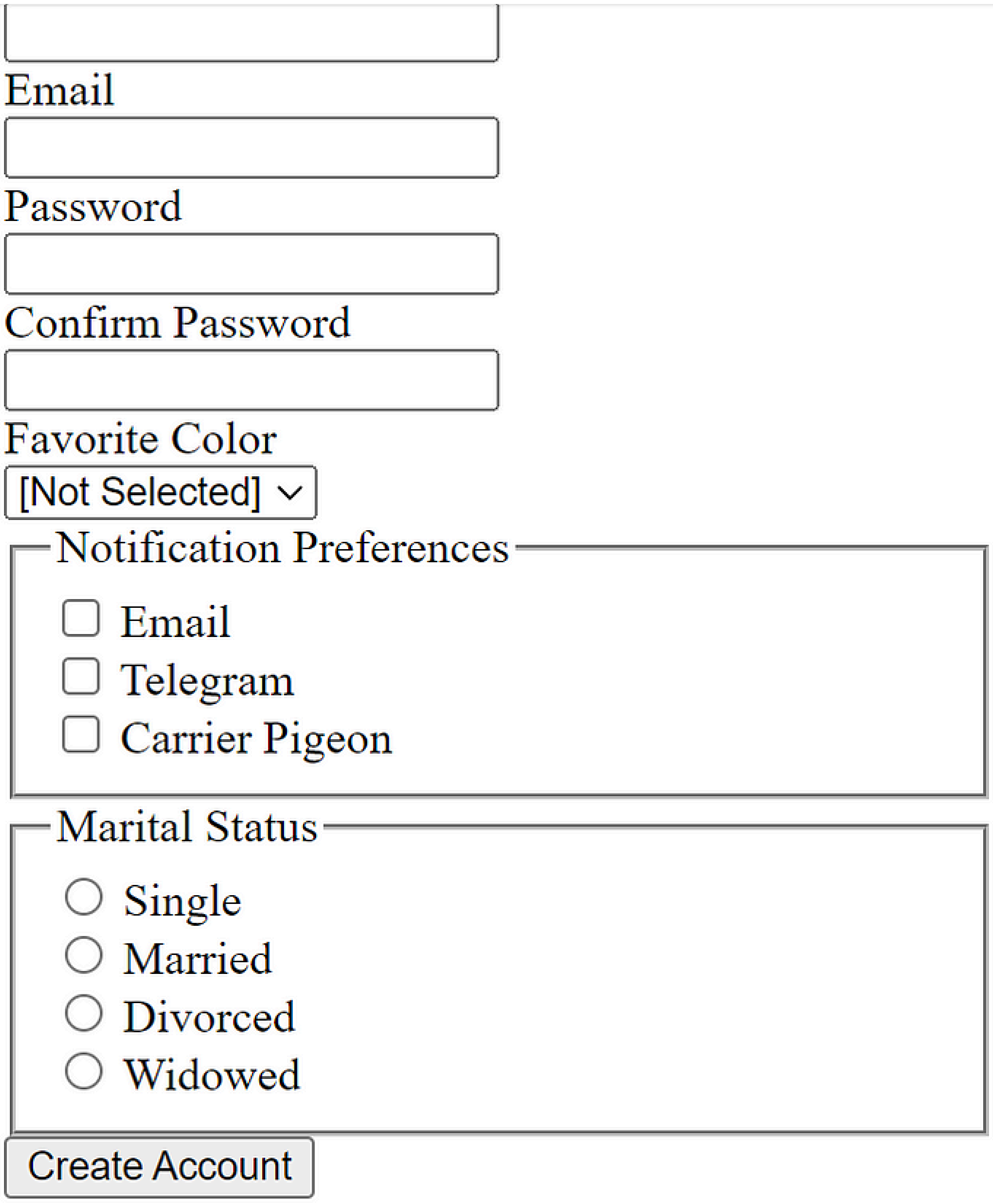
Photo by Josh Olalde on Unsplash

Let's put it all together

Copy

```
<form action="#" method="post">
  <label for="textFirstName">First Name</label>
  <input type="text" id="textFirstName" name="given-name"
    required >

  <label for="textEmail">Email</label>
  <input type="email" id="textEmail" name="email" required>

  <label for="textPassword">Password</label>
  <input type="password" id="textPassword" name="newPassword"
    autocomplete="off" required>

  <label for="textConfirmPassword">Confirm Password</label>
  <input type="password" id="textConfirmPassword" name="newConfirmPassword"
    autocomplete="off" required>

  <label for="selectFavoriteColor">Favorite Color</label>
  <select id="selectFavoriteColor" name="selectFavoriteColor" required>
    <option value="optionNotSelected">[Not Selected]</option>
    <option value="optionRed">Red</option>
    <option value="optionGreen">Green</option>
    <option value="optionOrange">Orange</option>
    <option value="optionBrown">Brown</option>
    <option value="optionPurple">Purple</option>
    <option value="optionBlack">Black</option>
    <option value="optionOther">Other</option>
  </select>

  <fieldset>
    <legend>Notification Preferences</legend>
    <label><input type="checkbox" id="checkEmail"
 name="notification[]" value="email" required> Email</label>
    <label><input type="checkbox" id="checkTelegram"
 name="notification[]" value="telegram"> Telegram</label>
    <label><input type="checkbox" id="checkCarrierPigeon"
 name="notification[]" value="pigeon"> Carrier Pigeon</label>
  </fieldset>

  <fieldset>
    <legend>Marital Status</legend>
    <label><input type="radio" name="color" value="single" required> Single</
    <label><input type="radio" name="color" value="married"> Married</label>
    <label><input type="radio" name="color" value="divorced"> Divorced</labe
    <label><input type="radio" name="color" value="widowed"> Widowed</label>
  </fieldset>

  <button id="buttonCreateAccount">Create Account</button>

</form>
```

And, let's see how it renders:



Why start with the markup?

Perhaps you're cringing because of the lack of styling.

I purposely omitted styles (outside of making most of the controls display as block) because thinking about the markup in isolation can help you remember that HTML is for structure, not function, nor presentation.

menus, no logos. I didn't even include a visual indication that any of the fields are required.

Imagine fitting all of this on a single page.

NOTE: There should be a heading for the form. I didn't include it to make that point. There are other changes I'd make, but I had to stop somewhere.

Starting with the HTML can help you come to decisions early in the process. For example, one look at this form makes me ask, "Which of these fields do we actually need in order to create an account?"

Questions like this will pop up even more as we add more markup, like the help text.

But, for now, we have our skeleton.

Conclusion

This isn't the end of the markup, obviously. In subsequent articles, I'll be marking fields as required, adding help text, including elements to house inline validation errors, adding a summary validation alert element, and incorporating various other elements to facilitate styling, validation, and focus management.

But this is a good place to stop.

In the interest of brevity, I refrained from using date controls, `<textarea>` elements, or any other control type.

Hopefully, this article was helpful to you — and perhaps you learned something new.

Again, I'll welcome any feedback, and I'm happy to correct anything I get wrong.

Links

References

- HTML Spec (The HTML Living Standard)
- MDN Web Docs — <input>: The Input (Form Input) element
- MDN Web Docs — <select>: The HTML Select element
- MDN Web Docs — <button>: The Button element
- MDN Web Docs — HTML attribute: autocomplete

Further reading

- The A11y Project: Create Accessible by Hamsa Harcourt
- Carnegie Museums: Forms, Form Validation, and CAPTCHAs

My related articles

- Accessible form validation from scratch — Part 1: Requirements
- 5 Ways to Make Labels Accessible for Input Controls — and 3 Ways Not to

#design #html #accessibility #ux #web-development