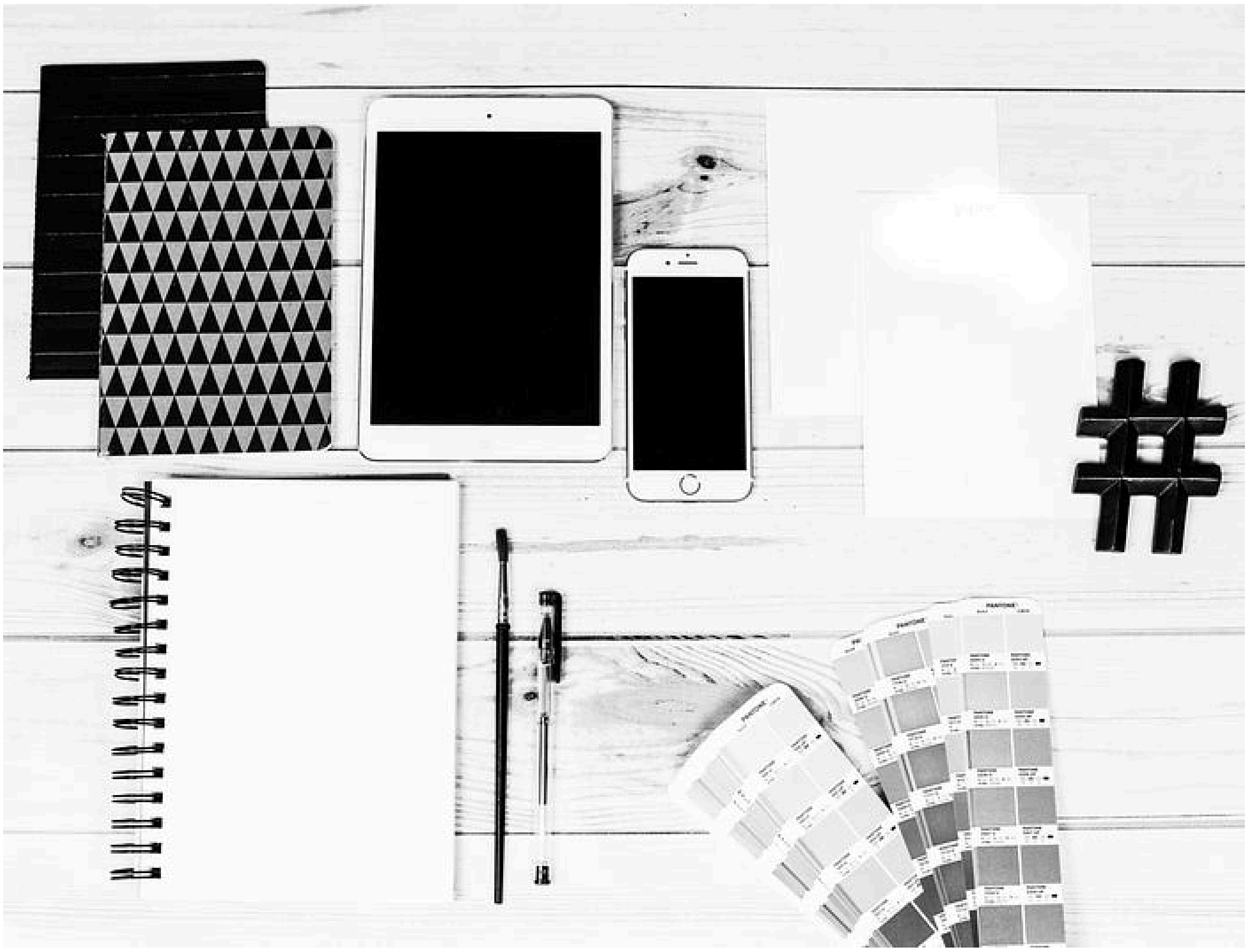
< Go to the original



Designing accessible forms: the 10 foundational rules

How to design forms with the user in mind

Tyler Hawkins

Follow

UX Collective a11y-light ~8 min read · February 25, 2020 (Updated: July 4, 2022) · Free: No

I'm sure at some point in our lives we've all had a painful experience filling out a form on the web. Confusing inputs, unclear expected formats, cryptic error messages, lack of keyboard-accessibility… The list could go on and on.

Forms are needed for all sorts of things online. You use them to sign up for a service, or to log in to your account, or to place an order on an e-commerce site. Forms are everywhere!

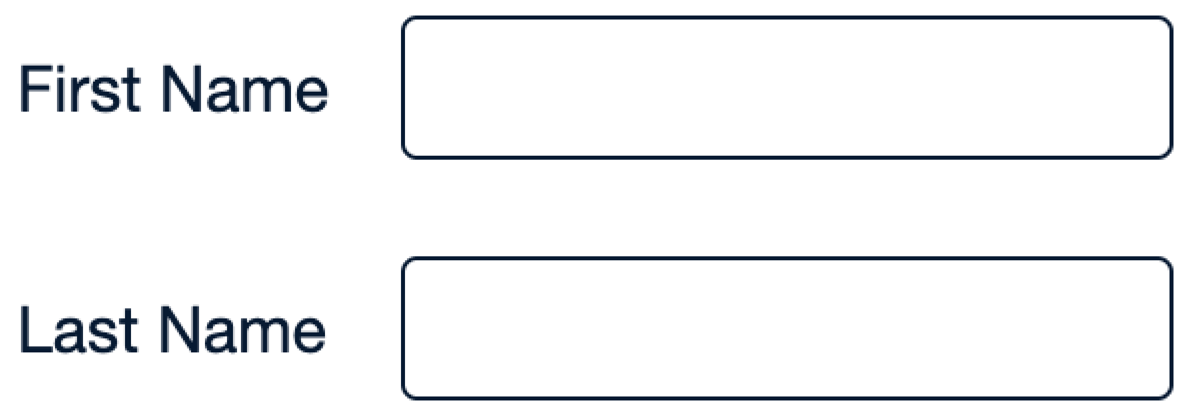
So as designers and developers, let's make them good.

Below are 10 guidelines you can follow to make your forms more accessible and more user-friendly today.

1. All inputs should have associated labels

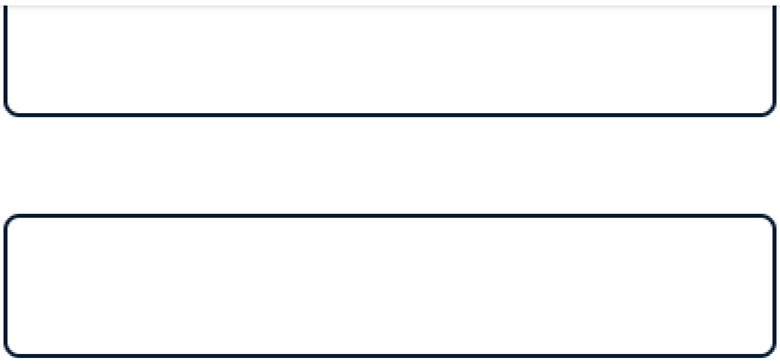
Labels are important for identifying your inputs. They help sighted users as well as blind users know what each input is for. Labels are especially important for screen readers so that they can correctly announce what each input is. You should use the actual `label` HTML element as well and not simply a `span` or `div` element.

Good Example:



Good: Inputs with associated labels

Bad Example:

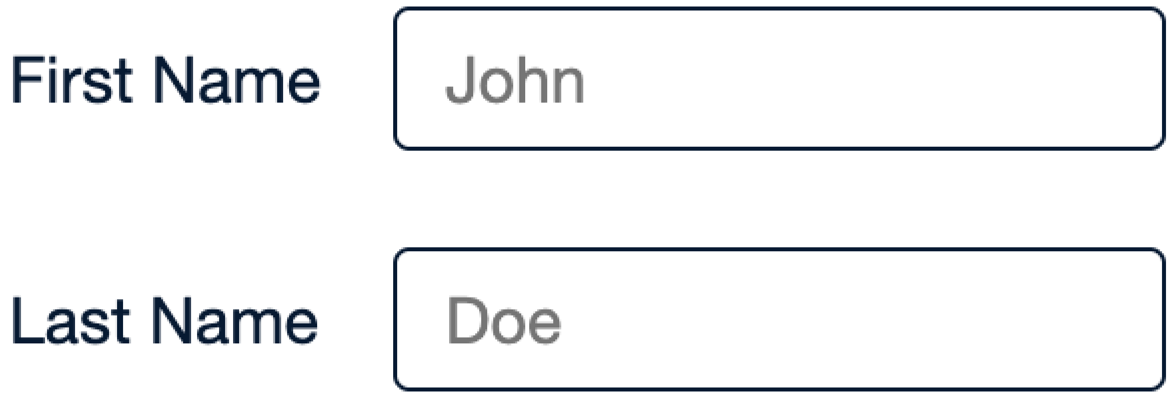


Bad: No labels. What am I filling out here?!

2. Placeholder text should be used for examples

Placeholder text is meant to be helpful as an example response. It should *not* be used as a replacement for a label. The problem with trying to use placeholder text as a label is that the placeholder text disappears once the user has entered some information, so the label is now effectively missing.

Good Example:



Good: Placeholder text as an example response

Bad Example:

First Name
Last Name

Bad: Placeholder text being used as a label

Placeholder text also does not need to be used to repeat what the label says. If your label says "First Name", it's redundant to have placeholder text that also says "First Name".

Bad Example:

First Name
First Name
Last Name
Last Name

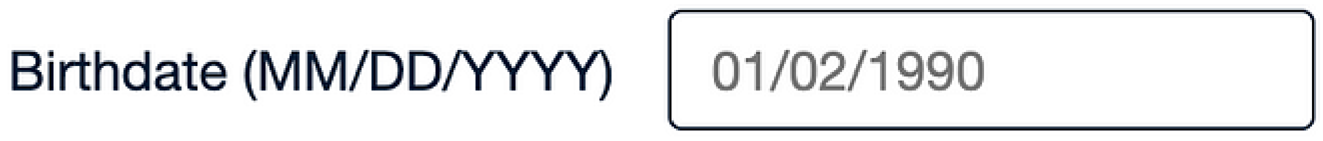
Bad: Redundant placeholder text

3. Formatting expectations should be displayed

If you are expecting the user to enter some information in a particular format, tell them up front! For example, if you need their birthdate to be entered in the `MM/DD/YYYY` format, display that somewhere near the input.

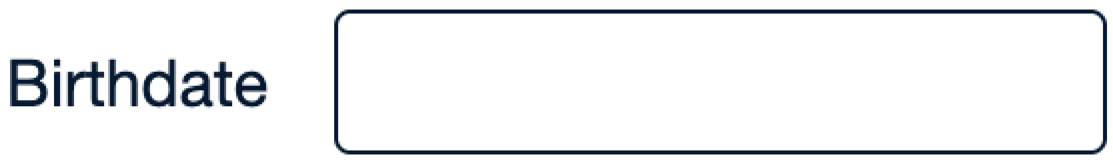
It's incredibly frustrating to enter something that you think is valid input, only to receive an error message when submitting your form that you've provided an

Good Example:



Good: Clear formatting expectations for the birthdate

Bad Example:



Bad: How should I enter my birthdate?!

Or, even better, be flexible in what data you can accept. If you are asking for someone's phone number, you could allow a wide variety of input values, such as:

- 8881234567
- 888 123 4567
- 888–123–4567
- (888) 123–4567

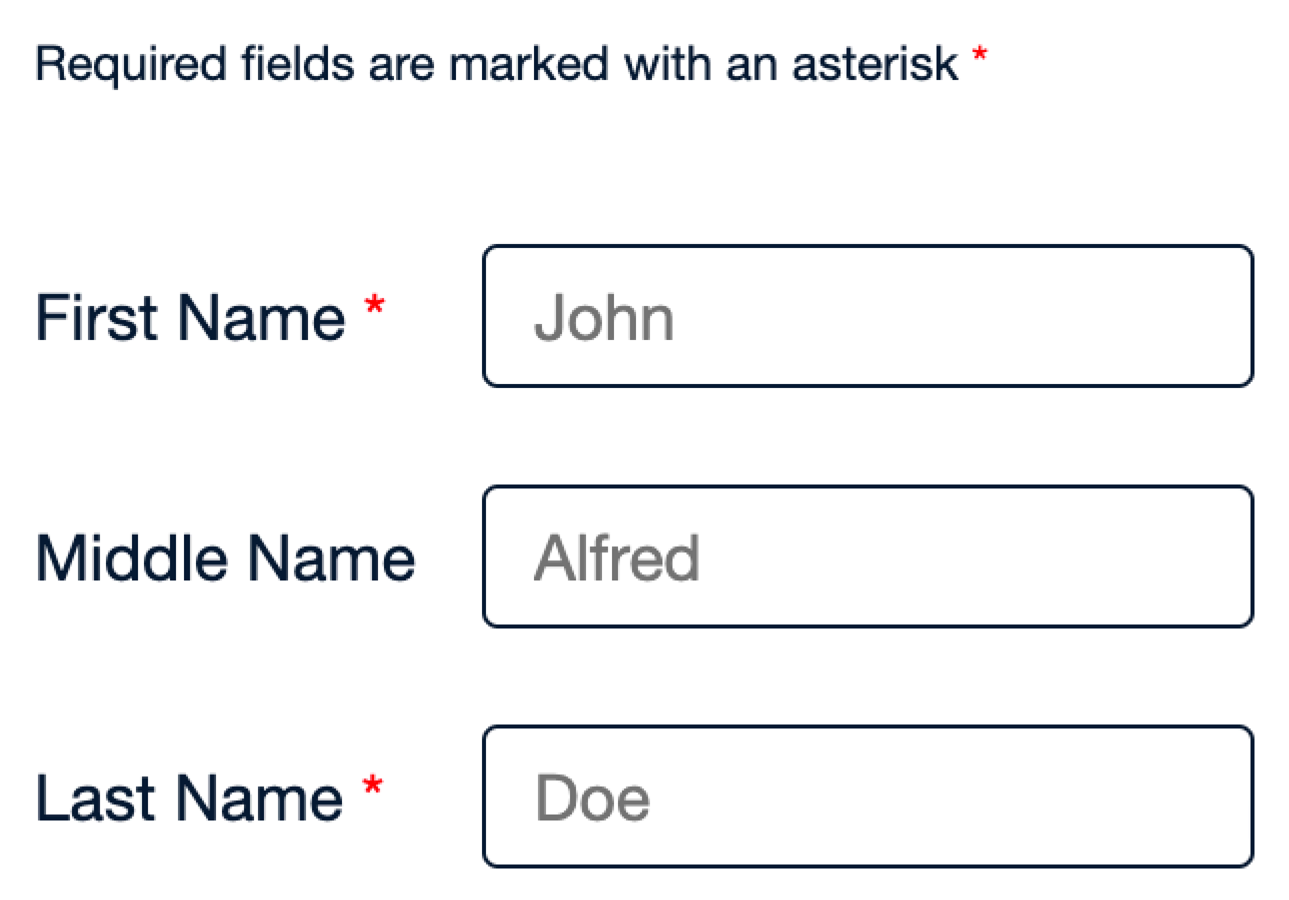
You can then parse the input yourself on the backend and strip away any non-digit characters and extra whitespace.

4. Required fields should be identified

This one is simple enough. If you have some fields that are required and some fields that are optional, make it clear which are which. Some designers prefer to mark the required fields with something like an asterisk or the text `required`

To me, it doesn't seem to matter which of these approaches you choose as long as you are consistent.

Good Example:

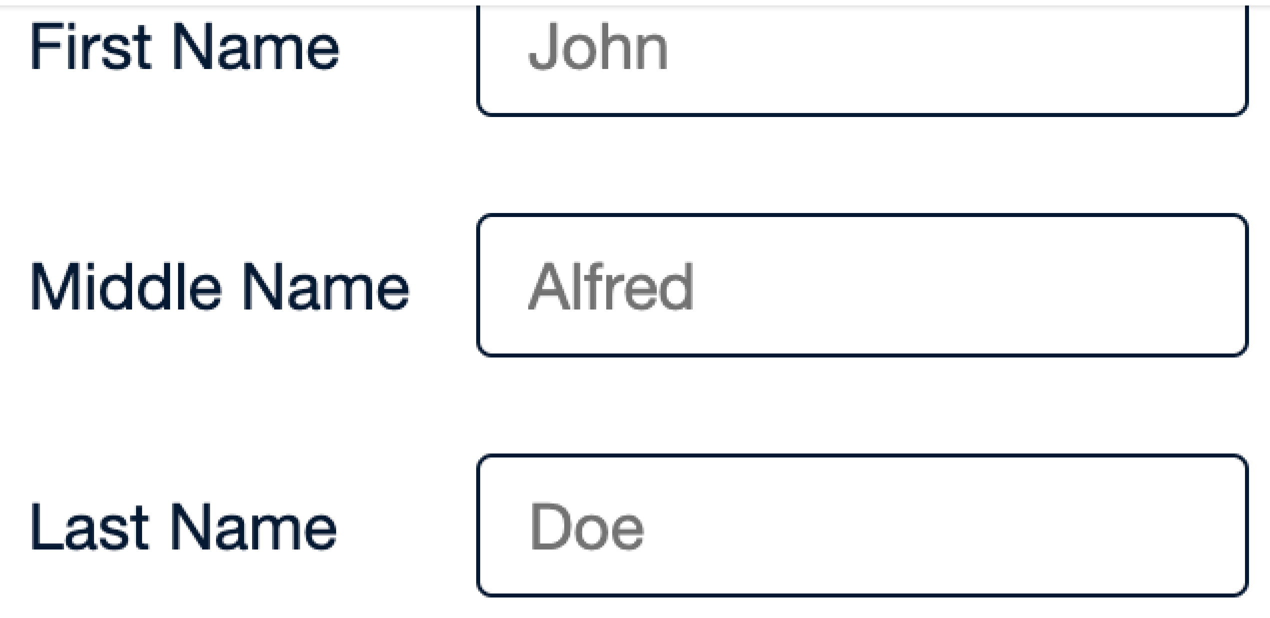


Required fields are marked with an asterisk *

First Name *	John
Middle Name	Alfred
Last Name *	Doe

Good: Required fields are clearly marked

Bad Example:


First Name John

Middle Name Alfred

Last Name Doe

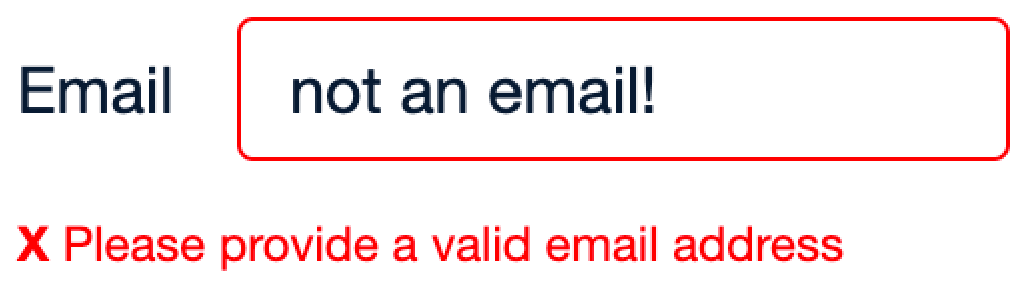
Bad: Assuming "Middle Name" is not required, this is not clearly shown

5. Color should not be the only indicator for feedback

When showing error messages, the color red is often used. For success messages, green is often used. However, color should not be the only indicator for feedback. If all you do is change the border on a text input when its value is valid or invalid, colorblind users may not be able to distinguish that difference. (In fact, red-green colorblindness is the most common form of colorblindness.)

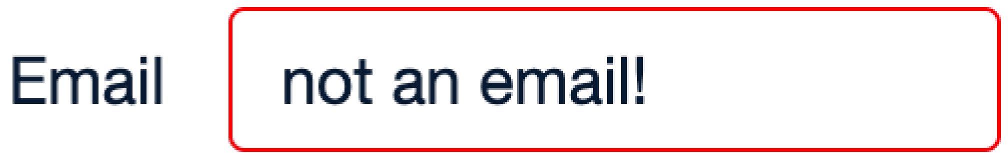
Rather, you should use a combination of color, text, and icons to display feedback such as error messages. A red X and a green checkmark are great to use, because even colorblind users can still note the obvious difference between the X and the checkmark.

Good Example:


Email not an email!

X Please provide a valid email address

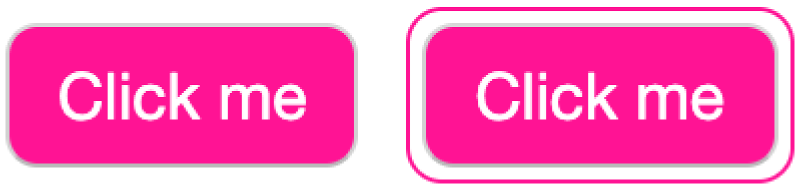
Bad Example:



Bad: Feedback is color-only, the red border

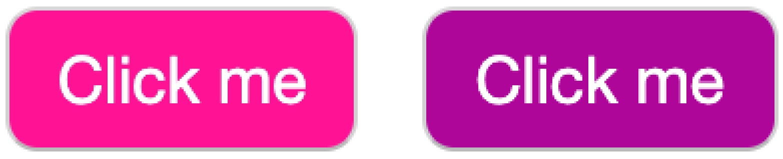
The same goes for buttons. Sometimes, you may have a button change color when it is hovered by the mouse or focused by the keyboard. However, something like an outline or underline indicator is much more effective at indicating focus since even users who can't distinguish between your chosen colors will still be able to see your outlines and underlines.

Good Example:



Good: The focused button has an outline

Bad Example:



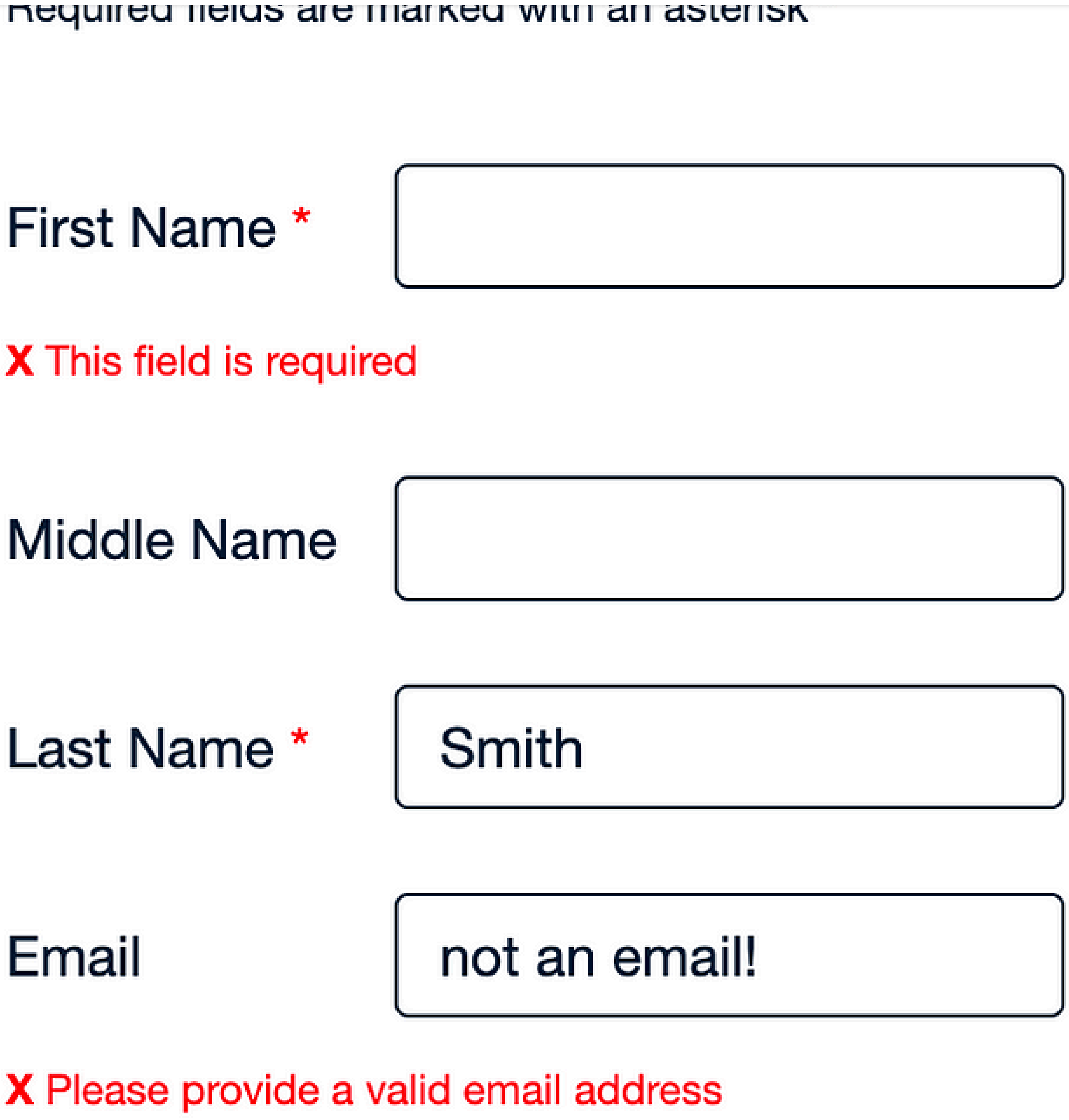
Bad: The focused button only changes color

6. Error messages should be helpful and close to the input

When validating a form, error messages should be displayed as soon as possible, preferably on the client-side rather than waiting until the whole form is submitted.

For example, if you have an email field, and the user enters something that is not an email, a helpful error message should be displayed right next to the form field when the input loses focus. Something like "Please provide a valid email address" should suffice for an error message.

Good Example:



Required fields are marked with an asterisk

First Name *

X This field is required

Middle Name

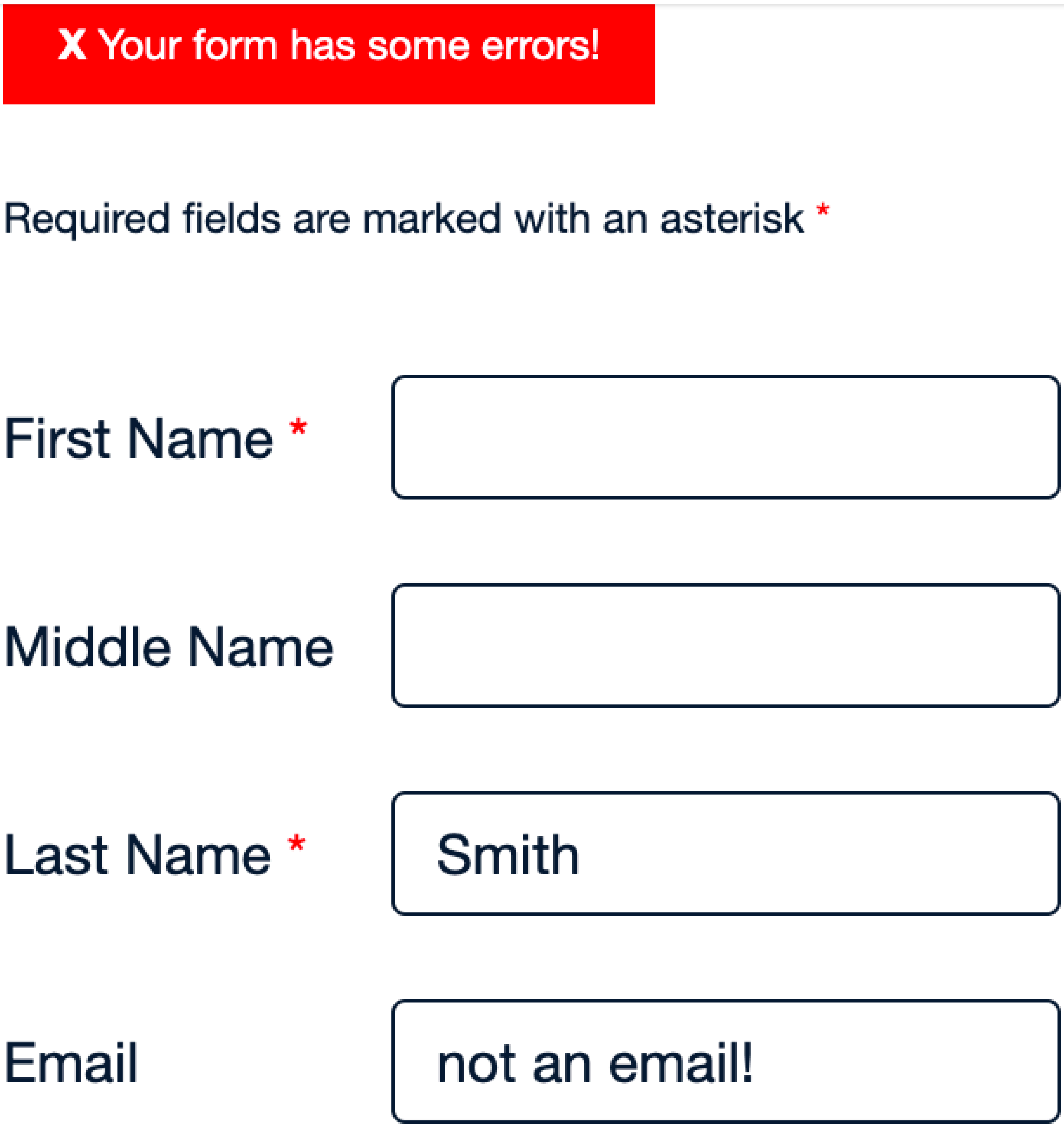
Last Name * Smith

Email not an email!

X Please provide a valid email address

Good: After submitting an invalid form, helpful error messages are shown

Bad Example:


X Your form has some errors!

Required fields are marked with an asterisk *

First Name *	
Middle Name	
Last Name *	Smith
Email	not an email!

Bad: After submitting an invalid form, an unhelpful error message is shown

Error messages should always tell the user how to correct the error. Unhelpful error messages like "Invalid input" or "Error" are not useful to the user. Perhaps the user doesn't understand why what they entered is not valid.

As noted in Guideline 3, if you have an input that requires a specific format for the response, make sure the user knows what the expected format is. In addition to displaying the expected format up front, you could also provide an error message such as "Please provide your birthdate in the following format: MM/DD/YYYY".

7. Every element should be reachable by the keyboard

The keyboard is for more than just power users who want to quickly navigate between form fields. Some motor-impaired users aren't able to use a mouse, so they solely rely on the keyboard. Screen reader users solely use the keyboard as well.

Inputs and buttons are keyboard-navigable by default, so as long as you are using the semantically correct HTML elements throughout your app, you should already get the correct functionality out the box.

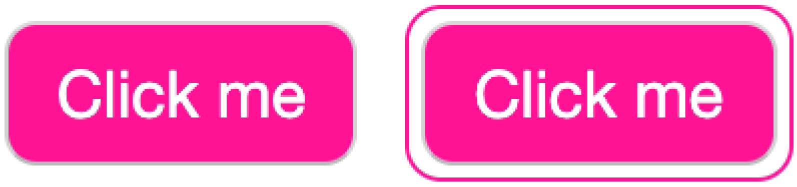
The problem arises when developers do silly things like use a `div` or a `span` in place of a `button`. Please… don't be that person. Using the correct HTML elements provides the correct handling for the keyboard and helps screen readers know what ARIA `role` each element has and how it should be treated.

8. Inputs and buttons should have focus indicators

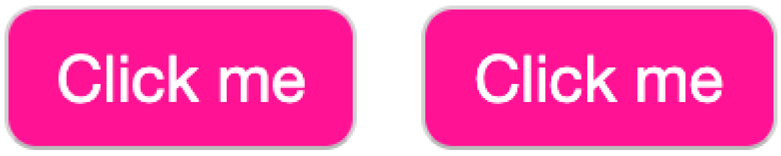
We briefly touched on this back in Guideline 5. To reiterate, you need to clearly indicate to the user what element they are currently focused on. Rather than using color only, you should use something like an outline or an underline.

You should feel free to use CSS to disable the default browser focus indicators if you want a consistent focus indicator across each browser, but don't simply disable these focus indicators without adding your own!

Good Example:



Good: The focused button has an outline



Bad: Even though the second button is focused, there is no indicator to make that apparent

9. Tab order should make sense

When a user is tabbing through your form, the tab order should be logical. For most Western languages that read left to right, your tab order should go left to right, top to bottom.

The tab order can get out of order by messing with the `tabindex` attribute on your HTML elements. Generally you'll only want to use `-1` and `0` as values for `tabindex`.

`-1` removes the element from the natural tab order but still allows the element to be focused programmatically. `0` is good for elements that are inserted dynamically, since `0` places the element in the logical tab order according to its placement in the DOM.

Any other values where you try to specifically control the tab order of your application are probably unnecessary and may be a sign that you are doing something wrong.

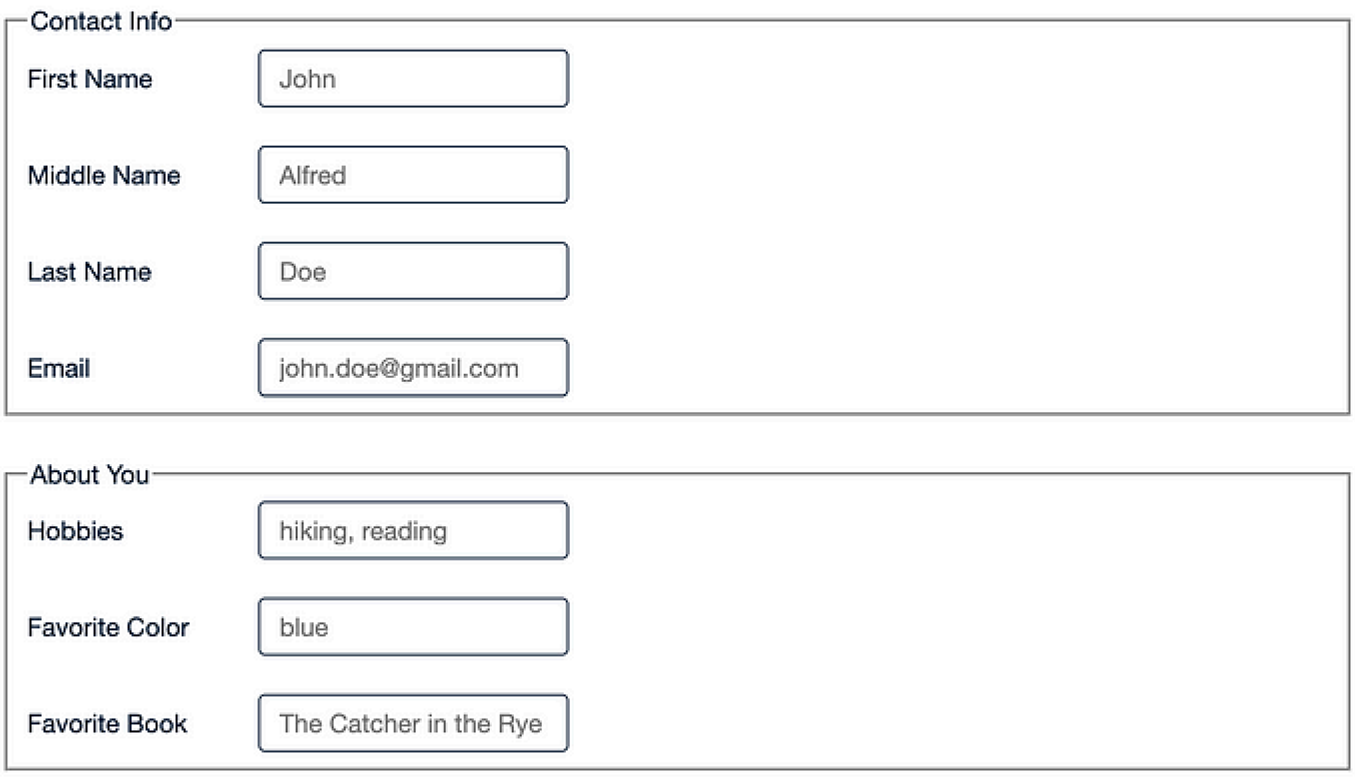
And while we're at it, don't automatically shift the focus from one input to the next after the user has entered some information. For example, if you have a text input that asks for a two-letter state abbreviation ("UT" for Utah), don't move the user to the next input after they've entered two characters.

You may think you're being helpful, but really you're just being annoying. Let the user control what they're focused on.

10. Fieldsets and legends should be used to group inputs

Lengthy forms can often be broken up into logical sections. If your form is displayed on a single page, `fieldset` and `legend` HTML elements can help group your inputs and make your form more readable.

Good Example:



Contact Info

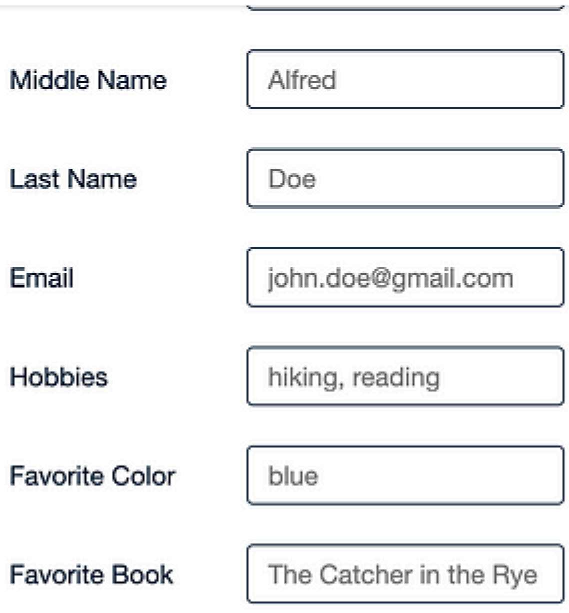
First Name	John
Middle Name	Alfred
Last Name	Doe
Email	john.doe@gmail.com

About You

Hobbies	hiking, reading
Favorite Color	blue
Favorite Book	The Catcher in the Rye

Good: Fieldsets are used to group sections of the form

Bad Example:



Middle Name	Alfred
Last Name	Doe
Email	john.doe@gmail.com
Hobbies	hiking, reading
Favorite Color	blue
Favorite Book	The Catcher in the Rye

Bad: This long form has no groupings to make it more manageable

Conclusion

To recap, the 10 guidelines you should follow to create accessible forms are:

1. All inputs should have associated labels
2. Placeholder text should be used for examples
3. Formatting expectations should be displayed
4. Required fields should be identified
5. Color should not be the only indicator for feedback
6. Error messages should be helpful and close to the input
7. Every element should be reachable by the keyboard
8. Inputs and buttons should have focus indicators
9. Tab order should make sense
10. Fieldsets and legends should be used to group inputs

By following these guidelines, you'll create a much better experience for users with and without disabilities. Thanks for reading!

#design #javascript #software-development #accessibility #ux