Create Amazing Forms

By Pete LePage
Pete is a Developer Advocate

Forms are hard to fill out on mobile. The best forms are the ones with the fewest inputs. Good forms provide semantic input types. Keys should change to match the user's input type; users pick a date in a calendar. Keep your user informed. Validation tools should tell the user what they need to do before submitting the form.

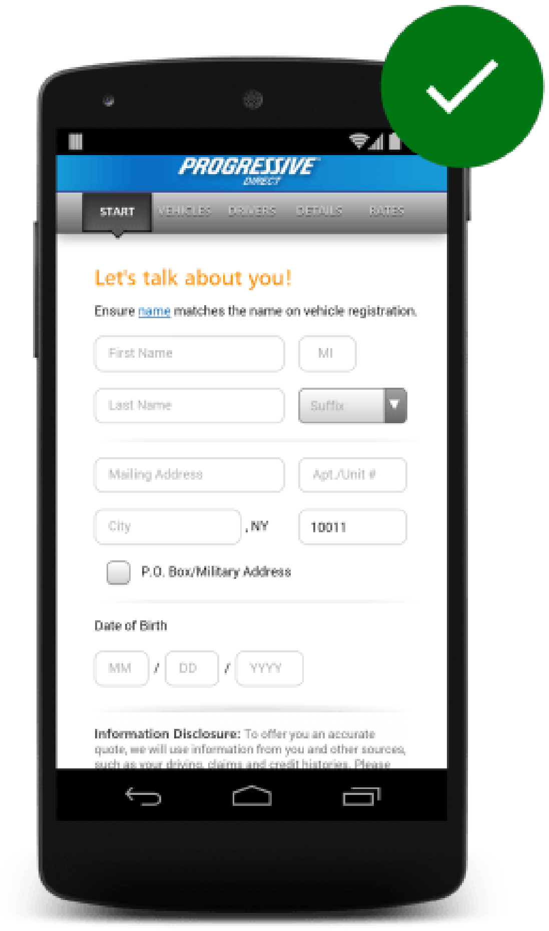
Design efficient forms

Design efficient forms by avoiding repeated actions, asking for only the necessary information and guide users by showing them how far along they are in multi-part forms.

TL;DR

- Use existing data to pre-populate fields and be sure to enable autofill.
- Use clearly-labeled progress bars to help users get through multi-part forms.
- Provide visual calendar so users don't have to leave your site and jump to the calendar app on their smartphones.

Minimize repeated actions and fields



Let's talk about you!

Ensure name matches the name on vehicle registration.

First Name | MI
Last Name | Suffix
Mailing Address | Apt./Unit #
City , NY | 10011

P.O. Box/Military Address

Date of Birth

MM / DD / YYYY

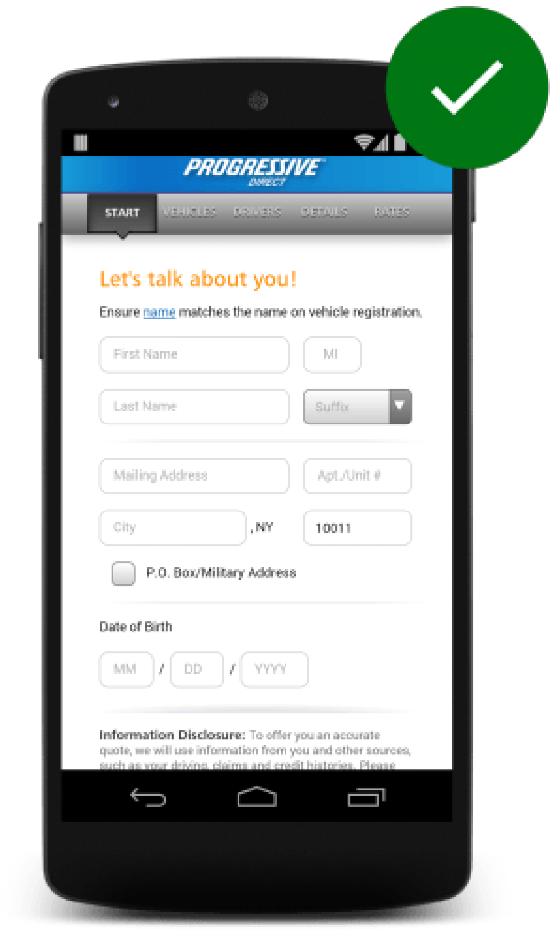
Information Disclosure: To offer you an accurate quote, we will use information from you and other sources, such as your driving, claims and credit histories. Please

On the Progressive.com website, users are asked first for their ZIP code, which is then pre-populated into the next part of the form.

Make sure your forms have no repeated actions, only as many fields as necessary, and take advantage of autofill, so that users can easily complete forms with pre-populated data.

Look for opportunities to pre-fill information you already know, or may anticipated to save the user from having to provide it. For example, pre-populate the shipping address with the last shipping address supplied by the user.

Show users how far along they are



Let's talk about you!

Ensure name matches the name on vehicle registration.

, NY 10011

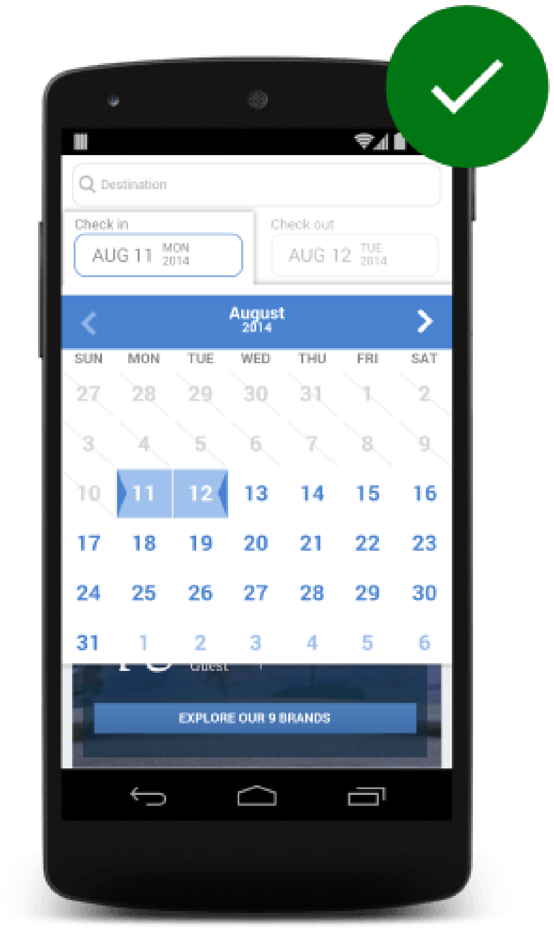
Information Disclosure: To offer you an accurate quote, we will use information from you and other sources, such as your driving, claims and credit histories. Please

Use clearly-labeled progress bars to help users get through multi-part forms.

Progress bars and menus should accurately convey overall progress through multi-step forms and processes.

If you place a disproportionately complex form in an earlier step, users are more likely to abandon your site before they go through the entire process.

Provide visual calendars when selecting dates



Hotel booking website with easy to use
calendar widget for picking dates.

Users often need more context when scheduling appointments and travel dates, to make things easier and prevent them from leaving your site to check their calendar app, provide a visual calendar with clear labeling for selecting start and end dates.

Choose the best input type

Streamline information entry by using the right input type. Users appreciate websites that automatically present number pads for entering phone numbers, or automatically advance fields as they entered them. Look for opportunities to eliminate wasted taps in your forms.

TL;DR

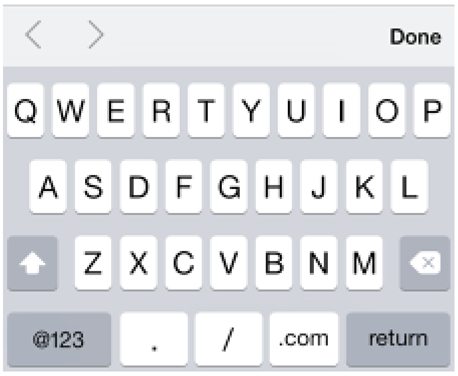
- Choose the most appropriate input type for your data to simplify input.
- Offer suggestions as the user types with the `datalist` element.

HTML5 input types

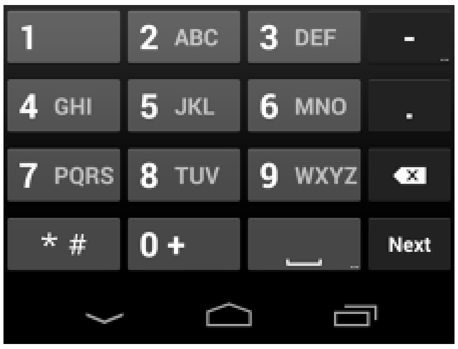
HTML5 introduced a number of new input types. These new input types give hints to the browser about what type of keyboard layout to display for on-screen keyboards. Users are more easily able to enter the required information without having to change their keyboard and only see the appropriate keys for that input type.

Input `type`

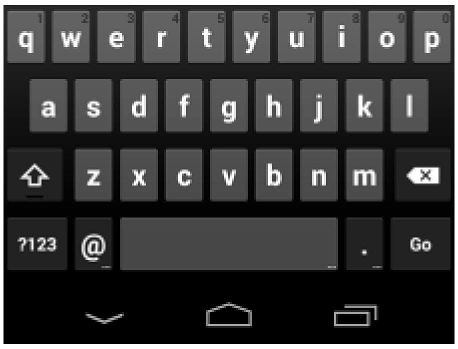
`url`
For entering a URL. It must start with a valid URI scheme, for example `http://`, `ftp://` or `mailto:`.

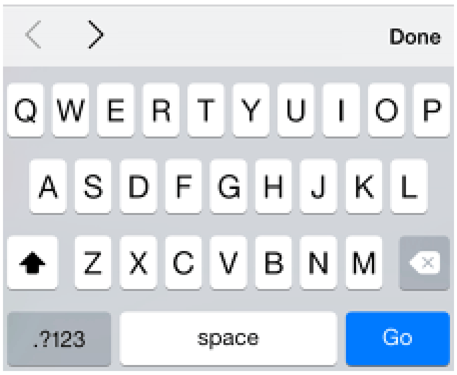
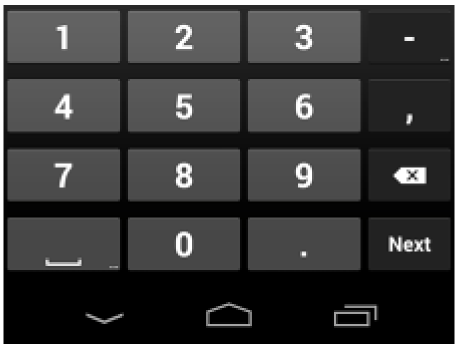
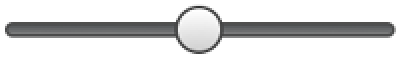
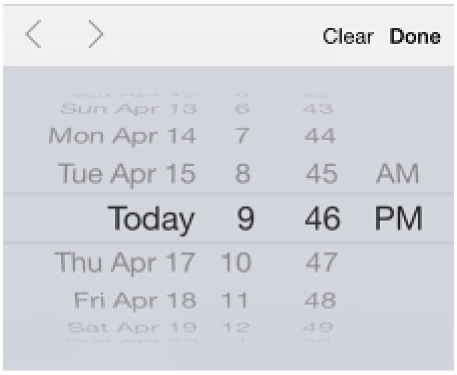


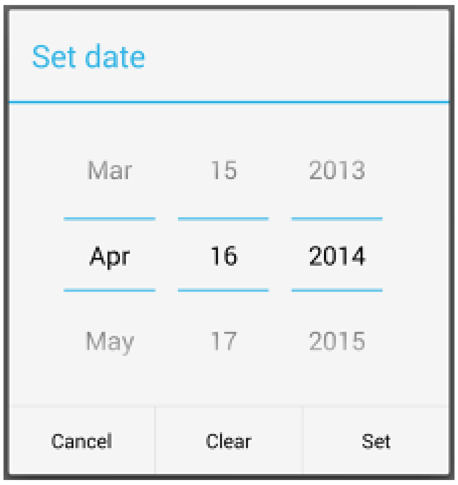
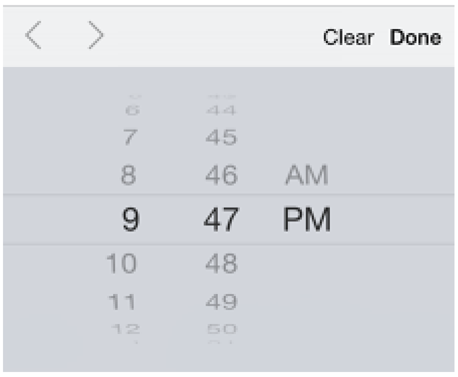
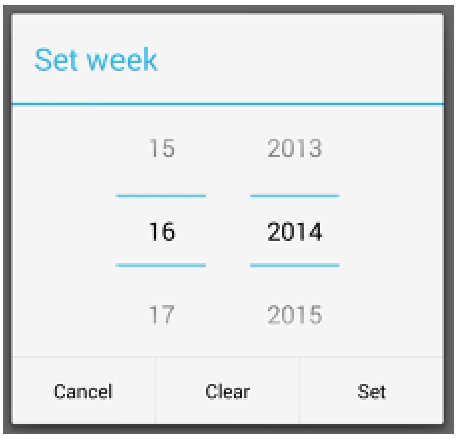
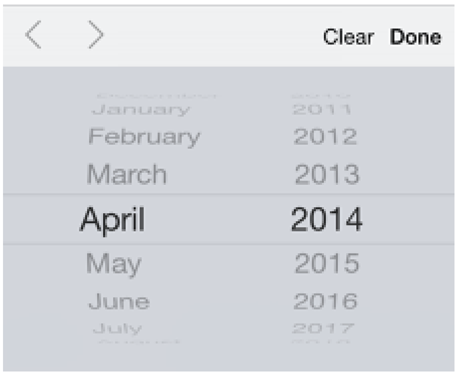
`tel`
For entering phone numbers. It does **not** enforce a particular syntax for validation, so if you want to ensure a particular format, you can use pattern.

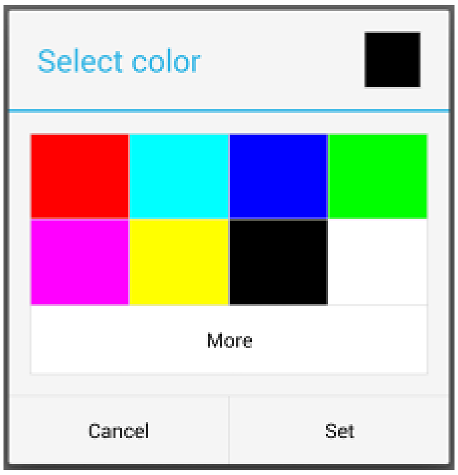


`email`
For entering email addresses, and hints that the @ should be shown on the keyboard by default. You can add the multiple attribute if more than one email address will be provided.



Input `type`	
`search` A text input field styled in a way that is consistent with the platform's search field.	 Done Q W E R T Y U I O P A S D F G H J K L Z X C V B N M .?123 space Go
`number` For numeric input, can be any rational integer. Additionally, iOS requires using `pattern="\d*"` to show the numeric keyboard.	
`range` For number input, but unlike the number input type, the value is less important. It is displayed to the user as a slider control.	
`datetime-local` For entering a date and time value where the time zone provided is the local time zone.	 Clear Done Sun Apr 13 6 43 Mon Apr 14 7 44 Tue Apr 15 8 45 AM Today 9 46 PM Thu Apr 17 10 47 Fri Apr 18 11 48 Sat Apr 19 12 49

Input `type`	
`date` For entering a date (only) with no time zone provided.	
`time` For entering a time (only) with no time zone provided.	
`week` For entering a week (only) with no time zone provided.	
`month` For entering a month (only) with no time zone provided.	

Input `type`	
`color` For picking a color.	

Caution: Remember to keep localization in mind when choosing an input type, some locales use a dot (.) as a separator instead of a comma (,)

Offer suggestions during input with datalist

The `datalist` element isn't an input type, but a list of suggested input values to associated with a form field. It lets the browser suggest autocomplete options as the user types. Unlike select elements where users must scan long lists to find the value they're looking for, and limiting them only to those lists, `datalist` element provides hints as the user types.

```
<label for="frmFavChocolate">Favorite Type of Chocolate</label>
<input type="text" name="fav-choc" id="frmFavChocolate" list="chocType">
<datalist id="chocType">
  <option value="white">
  <option value="milk">
  <option value="dark">
</datalist>
```

Try it

Note: The `datalist` values are provided as suggestions, and users are not restricted to the suggestions provided.

Label and name inputs properly

Forms are hard to fill out on mobile. The best forms are the ones with the fewest inputs. Good forms provide semantic input types. Keys should change to match the user's input type; users pick a date in a calendar. Keep your user informed. Validation tools should tell the user what they need to do before submitting the form.

TL;DR

- Always use `label`s on form inputs, and ensure they're visible when the field is in focus.
- Use `placeholder`s to provide guidance about what you expect.
- To help the browser auto-complete the form, use established `name`'s for elements and include the `autocomplete` attribute.

The importance of labels

The `label` element provides direction to the user, telling them what information is needed in a form element. Each `label` is associated with an input element by placing it inside the `label` element, or by using the "`for`" attribute. Applying labels to form elements also helps to improve the touch target size: the user can touch either the label or the input in order to place focus on the input element.

```
<label for="frmAddressS">Address</label>
<input type="text" name="ship-address" required id="frmAddressS"
  placeholder="123 Any Street" autocomplete="shipping street-address">
```

Try it

Label sizing and placement

Labels and inputs should be large enough to be easy to press. In portrait viewports, field labels should be above input elements, and beside them in landscape. Ensure field labels and the corresponding input boxes are visible at the same time. Be careful with custom scroll handlers that may scroll input elements to the top of the page hiding the label, or labels placed below input elements may be covered by the virtual keyboard.

Use placeholders

The placeholder attribute provides a hint to the user about what's expected in the input, typically by displaying the value as light text until the the user starts typing in the element.

MM-YYYY

```
<input type="text" placeholder="MM-YYYY" ...>
```

Caution: Placeholders disappear as soon as the user starts typing in an element, thus they are not a replacement for labels. They should be used as an aid to help guide users on the required format and content.

Use metadata to enable auto-complete

Users appreciate when websites save them time by automatically filling common fields like names, email addresses and other frequently used fields, plus it helps to reduce potential input errors – especially on virtual keyboards and small devices.

Browsers use many heuristics to determine which fields they can auto-populate based on previously specified data by the user, and you can give hints to the browser by providing both the `name` attribute and the `autocomplete` attribute on each input element.

Note: Chrome requires `input` elements to be wrapped in a `<form>` tag to enable auto-complete. If they're not wrapped in a `form` tag, Chrome will offer suggestions, but will **not** complete the form.

For example, to hint to the browser that it should auto-complete the form with the users name, email address and phone number, you should use:

```
<label for="frmNameA">Name</label>
<input type="text" name="name" id="frmNameA"
  placeholder="Full name" required autocomplete="name">

<label for="frmEmailA">Email</label>
<input type="email" name="email" id="frmEmailA"
  placeholder="name@example.com" required autocomplete="email">

<label for="frmEmailC">Confirm Email</label>
<input type="email" name="emailC" id="frmEmailC"
  placeholder="name@example.com" required autocomplete="email">

<label for="frmPhoneNumA">Phone</label>
<input type="tel" name="phone" id="frmPhoneNumA"
  placeholder="+1-555-555-1212" required autocomplete="tel">
```

Try it

Recommended input `name` and `autocomplete` attribute values

`autocomplete` attribute values are part of the current WHATWG HTML Standard. The most commonly used `autocomplete` attributes are shown below.

The `autocomplete` attributes can be accompanied with a section name, such as **`shipping`**`given-name` or **`billing`**`street-address`. The browser will auto-complete different sections separately, and not as a continuous form.

Content type	`name` attribute	`autocomplete` attribute
Name	`name` `fname` `mname` `lname`	• `name` (full name) • `given-name` (first name) • `additional-name` (middle name) • `family-name` (last name)
Email	`email`	`email`
Address	`address` `city` `region` `province` `state` `zip` `zip2` `postal` `country`	• For one address input: • `street-address` • For two address inputs: • `address-line1` • `address-line2` • `address-level1` (state or province) • `address-level2` (city) • `postal-code` (zip code) • `country`
Phone	`phone` `mobile` `country-code` `area-code` `exchange` `suffix` `ext`	`tel`
Credit Card	`ccname` `cardnumber` `cvc` `ccmonth` `ccyear` `exp-date` `card-type`	• `cc-name` • `cc-number` • `cc-csc` • `cc-exp-month` • `cc-exp-year` • `cc-exp` • `cc-type`
Usernames	`username`	• `username`

Content type	`name` attribute	`autocomplete` attribute
Password	`password`	• `current-password` (for sign-in forms) • `new-password` (for sign-up and password-change forms)

Note: Use either only `street-address` or both `address-line1` and `address-line2`. `address-level1` and `address-level2` are only necessary if they're required for your address format.

The `autofocus` attribute

On some forms, for example the Google home page where the only thing you want the user to do is fill out a particular field, you can add the `autofocus` attribute. When set, desktop browsers immediately move the focus to the input field, making it easy for users to quickly begin using the form. Mobile browsers ignore the `autofocus` attribute, to prevent the keyboard from randomly appearing.

Be careful using the autofocus attribute because it will steal keyboard focus and potentially preventing the backspace character from being used for navigation.

```
<input type="text" autofocus ...>
```

Avoid common patterns that break Chrome Autofill

Chrome Autofill makes filling out forms easier by automatically entering information they've saved to their Google account, Chrome browser, or mobile device. As a developer, you want to ensure that Autofill works well on your website so you can create a better experience for your users. This is especially important for checkout forms; users who successfully use Chrome Autofill to enter their information go through checkout an average of 30% faster than those who don't.

If you haven't already, make sure you have read the previous sections on developing good forms and using autocomplete attributes (part of the WHATWG HTML standard) on your site. This section covers some of the common mistakes developers make when building forms. Avoiding these pitfalls helps ensure that your users can effectively use Autofill, and could help increase conversions.

Field validation pitfalls

Some developers use client-side validation, which triggers input change or key events. For example, a site might truncate fields with Javascript instead of using the fields' "maxlength" attribute. Because Autofill does not recognize client-side validation, this truncation may cause the data to become invalid.

This often happens with phone fields when the maximum length is enforced using Javascript. Without the use of autocomplete attributes, Autofill may infer that it needs to fill a full phone number including the country code (e.g., in the US, eleven digits, such as "15552125555"). If the website truncates the value to ten digits using Javascript, the field value incorrectly becomes "1555212555". The correct way to support Autofill is to include `autocomplete="tel-national"` on the field, as pointed out in the WHATWG HTML standard.

While client-side validation may provide some benefits to users typing in their data, it usually ends up removing values that are pasted or autofilled.

Use standard input fields

Don't create your own form controls, especially custom dropdowns that replace `<select>` elements. This works poorly with accessibility frameworks as well as with Chrome Autofill. Instead, use standard dropdowns and other elements that can be easily modified through modern CSS.

Don't use fake placeholders in input fields

Some websites use "fake placeholders" in input fields instead of using the placeholder attribute. This is done by setting the placeholder text as the value of the field (e.g., `value="First Name"`) and using JavaScript to remove the value when the field gains focus. Autofill interprets such values as user-entered and doesn't replace the placeholder text with actual values, resulting in a poor Autofill experience. Instead, use floating field labels or `placeholder="First Name"` to guide users.

Don't copy the shipping address into the billing address section

Another common pitfall is when a user wants to use a billing address that differs from the shipping address. Often, the site automatically copies the shipping address values into the billing address section. This potentially creates additional work for the user, because Autofill has to be conservative about replacing the contents of pre-populated fields and is thus unable to assist in clearing the form and filling in the desired address.

Ensure that autocomplete attributes are correct

Autocomplete attributes as defined in the WHATWG HTML standard help your website tell Chrome Autofill explicitly what the fields are supposed to be, removing guesswork. However, these attributes are often misspelled or otherwise incorrect. When this happens, Autofill won't recognize the attribute and the unknown field type will not be autofilled.

For example, the correct attribute for the Credit Card CVC is "cc-csc". Many sites mistakenly use "cc-cvc", and because Autofill does not recognize this attribute, this field won't get autofilled.

The best practice for these attributes is to use this format: `autocomplete="<section> <fieldtype>"`, for example: `autocomplete="shipping address-line1"`. For a complete list of all the accepted values, please see the WHATWG HTML Living Standard.

Provide real-time validation

Real-time data validation doesn't just help to keep your data clean, but it also helps improve the user experience. Modern browsers have several built-in tools to help provide real-time data validation and may prevent the user from submitting an invalid form. Visual cues should be used to indicate whether a form has been completed properly.

TL;DR

- Leverage the browser's built-in validation attributes like `pattern`, `required`, `min`, `max`, etc.
- Use JavaScript and the Constraints Validation API for more complex validation requirements.
- Show validation errors in real time, and if the user tries to submit an invalid form, show all fields they need to fix.

Use these attributes to validate input

The `pattern` attribute

The `pattern` attribute specifies a regular expression used to validate an input field. For example, to validate a US Zip code (5 digits, sometimes followed by a dash and an additional 4 digits), we would set the `pattern` like this:

```
<input type="text" pattern="^\d{5,6}(?:[-\s]\d{4})?$" ...>
```

Common regular expression patterns

Regular expression

Postal address	`[a-zA-Z\d\s\-\,\#\.\+]+`
Zip Code (US)	`^\d{5,6}(?:[-\s]\d{4})?$`
IP Address (IPv4)	`^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$`
IP Address (IPv6)	`^(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|:((:[0-9a-fA-F]{1,4}){1,7}|:)|fe80:(:[0-9a-fA-F]{0,4}){0,4}%[0-9a-zA-Z]{1,}|::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])|([0-9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]))$`
IP Address (both)	`^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)|(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|:((:[0-9a-fA-F]{1,4}){1,7}|:)|fe80:(:[0-9a-fA-F]{0,4}){0,4}%[0-9a-zA-Z]{1,}|::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])|([0-9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]).){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]))$`
Credit Card Number	`^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|3[47][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|6(?:011|5[0-9]{2})[0-9]{12}|(?:2131|1800|35\d{3})\d{11})$`

Regular expression	
Social Security Number	`^\d{3}-\d{2}-\d{4}$`
North American Phone Number	`^(?:(?:\+?1\s*(?:[.-]\s*)?)?(?:\(\s*([2-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9])\s*\)|([2-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9]))\s*(?:[.-]\s*)?)?([2-9]1[02-9]|[2-9][02-9]1|[2-9][02-9]{2})\s*(?:[.-]\s*)?([0-9]{4})(?:\s*(?:#|x\.?|ext\.?|extension)\s*(\d+))?$`

The `required` attribute

If the **`required`** attribute is present, then the field must contain a value before the form can be submitted. For example, to make the zip code required, we'd simply add the required attribute:

```
<input type="text" required pattern="^\d{5,6}(?:[-\s]\d{4})?$" ...>
```

The `min`, `max` and `step` attributes

For numeric input types like number or range as well as date/time inputs, you can specify the minimum and maximum values, as well as how much they should each increment/decrement when adjusted by the slider or spinners. For example, a shoe size input would set a minimum size of 1 and a maximum size 13, with a step of 0.5

```
<input type="number" min="1" max="13" step="0.5" ...>
```

The `maxlength` attribute

The **`maxlength`** attribute can be used to specify the maximum length of an input or textbox and is useful when you want to limit the length of information that the user can provide. For example, if you want to limit a filename to 12 characters, you can use the following.

```
<input type="text" id="83filename" maxlength="12" ...>
```

The `minlength` attribute

The **`minlength`** attribute can be used to specify the minimum length of an input or textbox and is useful when you want to specify a minimum length the user must provide. For example, if you want to specify that a file name requires at least 8 characters, you can use the following.

```
<input type="text" id="83filename" minlength="8" ...>
```

The `novalidate` attribute

In some cases, you may want to allow the user to submit the form even if it contains invalid input. To do this, add the **`novalidate`** attribute to the form element, or individual input fields. In this case, all pseudo classes and JavaScript APIs will still allow you to check if the form validates.

```
<form role="form" novalidate>
  <label for="inpEmail">Email address</label>
  <input type="email" ...>
</form>
```

Success: Even with client-side input validation, it is always important to validate data on the server to ensure consistency and security in your data.

Use JavaScript for more complex real-time validation

When the built-in validation plus regular expressions aren't enough, you can use the Constraint Validation API, a powerful tool for handling custom validation. The API allows you to do things like set a custom error, check whether an element is valid, and determine the reason that an element is invalid:

Constraint Validation

`setCustomValidity()`	Sets a custom validation message and the **`customError`** property of the **`ValidityState`** object to **`true`**.
`validationMessage`	Returns a string with the reason the input failed the validation test.
`checkValidity()`	Returns **`true`** if the element satisfies all of its constraints, and **`false`** otherwise. Deciding how the page responds when the check returns **`false`** is left up to the developer.
`reportValidity()`	Returns **`true`** if the element satisfies all of its constraints, and **`false`** otherwise. When the page responds **`false`**, constraint problems are reported to the user.

Constraint Validation

`validity`	Returns a `ValidityState` object representing the validity states of the element.

Set custom validation messages

If a field fails validation, use `setCustomValidity()` to mark the field invalid and explain why the field didn't validate. For example, a sign up form might ask the user to confirm their email address by entering it twice. Use the blur event on the second input to validate the two inputs and set the appropriate response. For example:

```
if (input.value != primaryEmail) {
  // the provided value doesn't match the primary email address
  input.setCustomValidity('The two email addresses must match.');
  console.log("E-mail addresses do not match", primaryEmail, input.value);
} else {
  // input is valid -- reset the error message
  input.setCustomValidity('');
}
```

Try it

Prevent form submission on invalid forms

Because not all browsers will prevent the user from submitting the form if there is invalid data, you should catch the submit event, and use the `checkValidity()` on the form element to determine if the form is valid. For example:

```
form.addEventListener("submit", function(evt) {
  if (form.checkValidity() === false) {
    evt.preventDefault();
    alert("Form is invalid - submission prevented!");
    return false;
  } else {
    // To prevent data from being sent, we've prevented submission
    // here, but normally this code block would not exist.
    evt.preventDefault();
    alert("Form is valid - submission prevented to protect privacy.");
    return false;
  }
});
```

Try it

Show feedback in real-time

It's helpful to provide a visual indication on each field that indicates whether the user has completed the form properly before they've submitted the form. HTML5 also introduces a number of new pseudo-classes that can be used to style inputs based on their value or attributes.

Real-time Feedback

`:valid`	Explicitly sets the style for an input to be used when the value meets all of the validation requirements.
`:invalid`	Explicitly sets the style for an input to be used when the value does not meet all of the validation requirements.
`:required`	Explicitly sets the style for an input element that has the required attribute set.
`:optional`	Explicitly sets the style for an input element that does not have the required attribute set.
`:in-range`	Explicitly sets the style for a number input element where the value is in range.
`:out-of-range`	Explicitly sets the style for a number input element where the value is out of range.

Validation happens immediately which means that when the page is loaded, fields may be marked as invalid, even though the user hasn't had a chance to fill them in yet. It also means that as the user types, and it's possible they'll see the invalid style while typing. To prevent this, you can combine the CSS with JavaScript to only show invalid styling when the user has visited the field.

```
input.dirty:not(:focus):invalid {
  background-color: #FFD9D9;
}
input.dirty:not(:focus):valid {
  background-color: #D9FFD9;
}
```

```
var inputs = document.getElementsByTagName("input");
var inputs_len = inputs.length;
var addDirtyClass = function(evt) {
  sampleCompleted("Forms-order-dirty");
  evt.srcElement.classList.toggle("dirty", true);
};
for (var i = 0; i < inputs_len; i++) {
  var input = inputs[i];
  input.addEventListener("blur", addDirtyClass);
  input.addEventListener("invalid", addDirtyClass);
  input.addEventListener("valid", addDirtyClass);
}
```

Try it

Success: You should show the user all of the issues on the form at once, rather than showing them one at a time.