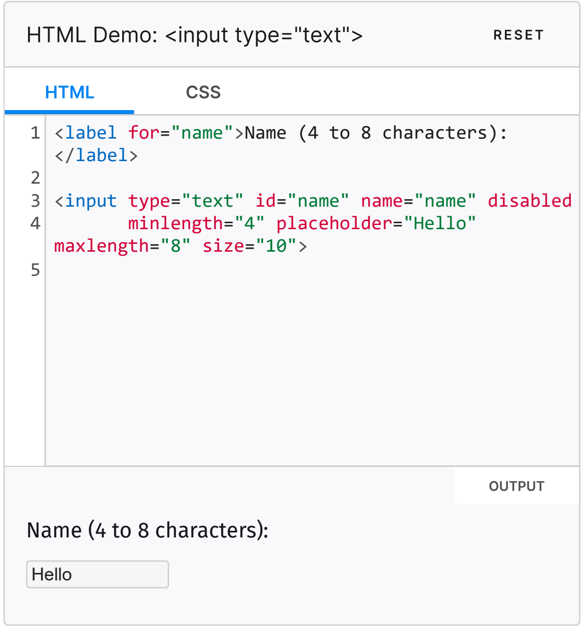
<input>: The Input (Form Input) element

The `<input>` HTML element is used to create interactive controls for web-based forms in order to accept data from the user; a wide variety of types of input data and control widgets are available, depending on the device and user agent. The `<input>` element is one of the most powerful and complex in all of HTML due to the sheer number of combinations of input types and attributes.

Try it



```html
<label for="name">Name (4 to 8 characters):
</label>

<input type="text" id="name" name="name" disabled
       minlength="4" placeholder="Hello"
maxlength="8" size="10">
```

Name (4 to 8 characters):

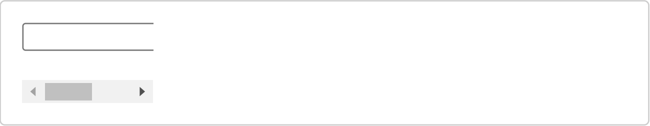
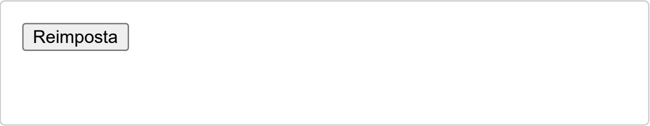
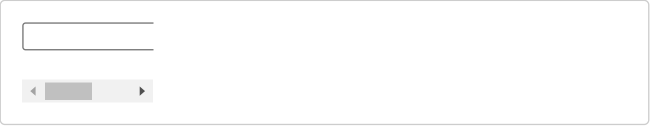
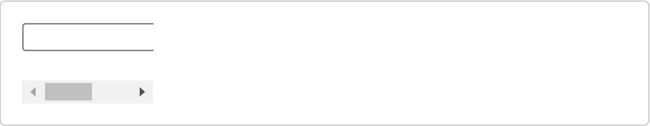
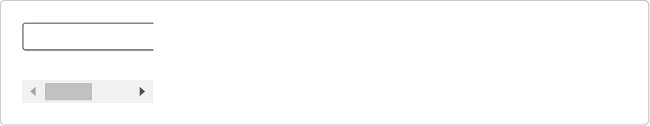
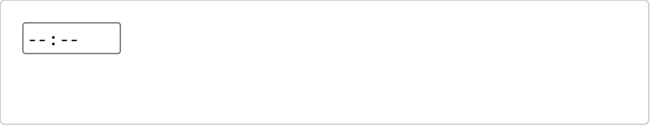
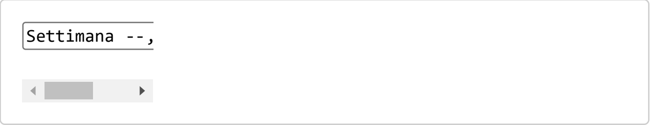
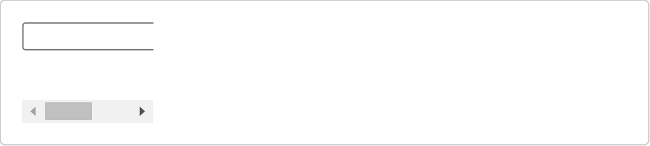
Hello

<input> types

How an `<input>` works varies considerably depending on the value of its `type` attribute, hence the different types are covered in their own separate reference pages. If this attribute is not specified, the default type adopted is `text`.

The available types are as follows:

button	A push button with no default behavior displaying the value of the `value` attribute, empty by default.	Button
checkbox	A check box allowing single values to be selected/deselected.	□
color	A control for specifying a color; opening a color picker when active in supporting browsers.	▭
date	A control for entering a date (year, month, and day, with no time). Opens a date picker or numeric wheels for year, month, day when active in supporting browsers.	gg/mm/aaaa
datetime-local	A control for entering a date and time, with no time zone. Opens a date picker or numeric wheels for date- and time-components when active in supporting browsers.	gg/mm/aaaa --:--
email	A field for editing an email address. Looks like a `text` input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.	▭
file	A control that lets the user select a file. Use the `accept` attribute to define the types of files that the control can select.	Scegli file Nes
hidden	A control that is not displayed but whose value is submitted to the server. There is an example in the next column, but it's hidden!	
image	A graphical `submit` button. Displays an image defined by the `src` attribute. The `alt` attribute displays if the image `src` is missing.	image input
month	A control for entering a month and year, with no time zone.	

Type	Description	Basic Examples
number	A control for entering a number. Displays a spinner and adds default validation. Displays a numeric keypad in some devices with dynamic keypads.	
password	A single-line text field whose value is obscured. Will alert user if site is not secure.	
radio	A radio button, allowing a single value to be selected out of multiple choices with the same `name` value.	
range	A control for entering a number whose exact value is not important. Displays as a range widget defaulting to the middle value. Used in conjunction `min` and `max` to define the range of acceptable values.	
reset	A button that resets the contents of the form to default values. Not recommended.	
search	A single-line text field for entering search strings. Line-breaks are automatically removed from the input value. May include a delete icon in supporting browsers that can be used to clear the field. Displays a search icon instead of enter key on some devices with dynamic keypads.	
submit	A button that submits the form.	
tel	A control for entering a telephone number. Displays a telephone keypad in some devices with dynamic keypads.	
text	The default value. A single-line text field. Line-breaks are automatically removed from the input value.	
time	A control for entering a time value with no time zone.	
url	A field for entering a URL. Looks like a `text` input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.	
week	A control for entering a date consisting of a week-year number and a week number with no time zone.	
Obsolete values		
`datetime` 🗑	A control for entering a date and time (hour, minute, second, and fraction of a second) based on UTC time zone.	

Attributes

The `<input>` element is so powerful because of its attributes; the `type` attribute, described with examples above, being the most important. Since every `<input>` element, regardless of type, is based on the `HTMLInputElement` interface, they technically share the exact same set of attributes. However, in reality, most attributes have an effect on only a specific subset of input types. In addition, the way some attributes impact an input depends on the input type, impacting different input types in different ways.

This section provides a table listing all the attributes with a brief description. This table is followed by a list describing each attribute in greater detail, along with which input types they are associated with. Those that are common to most or all input types are defined in greater detail below. Attributes that are unique to particular input types—or attributes which are common to all input types but have special behaviors when used on a given input type—are instead documented on those types' pages.

Attributes for the `<input>` element include the global HTML attributes and additionally:

Attribute	Type or Types	Description
`accept`	`file`	Hint for expected file type in file upload controls
`alt`	`image`	alt attribute for the image type. Required for accessibility

Attribute	Type or Types	Description
`autocomplete`	all except `checkbox`, `radio`, and buttons	Hint for form autofill feature
`capture`	`file`	Media capture input method in file upload controls
`checked`	`checkbox`, `radio`	Whether the command or control is checked
`dirname`	`search`, `text`	Name of form field to use for sending the element's directionality in form submission
`disabled`	all	Whether the form control is disabled
`form`	all	Associates the control with a form element
`formaction`	`image`, `submit`	URL to use for form submission
`formenctype`	`image`, `submit`	Form data set encoding type to use for form submission
`formmethod`	`image`, `submit`	HTTP method to use for form submission
`formnovalidate`	`image`, `submit`	Bypass form control validation for form submission
`formtarget`	`image`, `submit`	Browsing context for form submission
`height`	`image`	Same as height attribute for `<img>`; vertical dimension
`list`	all except `hidden`, `password`, `checkbox`, `radio`, and buttons	Value of the id attribute of the `<datalist>` of autocomplete options
`max`	`date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`	Maximum value
`maxlength`	`text`, `search`, `url`, `tel`, `email`, `password`	Maximum length (number of characters) of `value`
`min`	`date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`	Minimum value
`minlength`	`text`, `search`, `url`, `tel`, `email`, `password`	Minimum length (number of characters) of `value`
`multiple`	`email`, `file`	Boolean. Whether to allow multiple values
`name`	all	Name of the form control. Submitted with the form as part of a name/value pair
`pattern`	`text`, `search`, `url`, `tel`, `email`, `password`	Pattern the `value` must match to be valid
`placeholder`	`text`, `search`, `url`, `tel`, `email`, `password`, `number`	Text that appears in the form control when it has no value set
`popovertarget`	`button`	Designates an `<input type="button">` as a control for a popover element
`popovertargetaction`	`button`	Specifies the action that a popover control should perform
`readonly`	all except `hidden`, `range`, `color`, `checkbox`, `radio`, and buttons	Boolean. The value is not editable
`required`	all except `hidden`, `range`, `color`, and buttons	Boolean. A value is required or must be check for the form to be submittable
`size`	`text`, `search`, `url`, `tel`, `email`, `password`	Size of the control
`src`	`image`	Same as `src` attribute for `<img>`; address of image resource
`step`	`date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`	Incremental values that are valid
`type`	all	Type of form control
`value`	all except `image`	The initial value of the control
`width`	`image`	Same as `width` attribute for `<img>`

A few additional non-standard attributes are listed following the descriptions of the standard attributes.

Individual attributes

`accept`

Valid for the `file` input type only, the `accept` attribute defines which file types are selectable in a `file` upload control. See the file input type.

`alt`

Valid for the `image` button only, the `alt` attribute provides alternative text for the image, displaying the value of the attribute if the image `src` is missing or otherwise fails to load. See the image input type.

`autocomplete`

(**Not** a Boolean attribute!) The `autocomplete` attribute takes as its value a space-separated string that describes what, if any, type of autocomplete functionality the input should provide. A typical implementation of autocomplete recalls previous values entered in the same input field, but more complex forms of autocomplete can exist. For instance, a browser could integrate with a device's contacts list to autocomplete `email` addresses in an email input field. See `autocomplete` for permitted values.

The `autocomplete` attribute is valid on `hidden`, `text`, `search`, `url`, `tel`, `email`, `date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`, `color`, and `password`. This attribute has no effect on input types that do not return numeric or text data, being valid for all input types except `checkbox`, `radio`, `file`, or any of the button types.

See the `autocomplete` attribute for additional information, including information on password security and how `autocomplete` is slightly different for `hidden` than for other input types.

`autofocus`

A Boolean attribute which, if present, indicates that the input should automatically have focus when the page has finished loading (or when the `<dialog>` containing the element has been displayed).

> **Note:** An element with the `autofocus` attribute may gain focus before the `DOMContentLoaded` event is fired.

No more than one element in the document may have the `autofocus` attribute. If put on more than one element, the first one with the attribute receives focus.

The `autofocus` attribute cannot be used on inputs of type `hidden`, since hidden inputs cannot be focused.

> **Warning:** Automatically focusing a form control can confuse visually-impaired people using screen-reading technology and people with cognitive impairments. When `autofocus` is assigned, screen-readers "teleport" their user to the form control without warning them beforehand.

Use careful consideration for accessibility when applying the `autofocus` attribute. Automatically focusing on a control can cause the page to scroll on load. The focus can also cause dynamic keyboards to display on some touch devices. While a screen reader will announce the label of the form control receiving focus, the screen reader will not announce anything before the label, and the sighted user on a small device will equally miss the context created by the preceding content.

`capture`

Introduced in the HTML Media Capture specification and valid for the `file` input type only, the `capture` attribute defines which media—microphone, video, or camera—should be used to capture a new file for upload with `file` upload control in supporting scenarios. See the file input type.

`checked`

Valid for both `radio` and `checkbox` types, `checked` is a Boolean attribute. If present on a `radio` type, it indicates that the radio button is the currently selected one in the group of same-named radio buttons. If present on a `checkbox` type, it indicates that the checkbox is checked by default (when the page loads). It does *not* indicate whether this checkbox is currently checked: if the checkbox's state is changed, this content attribute does not reflect the change. (Only the `HTMLInputElement`'s `checked` IDL attribute is updated.)

> **Note:** Unlike other input controls, a checkboxes and radio buttons value are only included in the submitted data if they are currently `checked`. If they are, the name and the value(s) of the checked controls are submitted.
>
> For example, if a checkbox whose `name` is `fruit` has a `value` of `cherry`, and the checkbox is checked, the form data submitted will include `fruit=cherry`. If the checkbox isn't active, it isn't listed in the form data at all. The default `value` for checkboxes and radio buttons is `on`.

`dirname`

Valid for `text` and `search` input types only, the `dirname` attribute enables the submission of the directionality of the element. When included, the form control will submit with two name/value pairs: the first being the `name` and `value`, the second being the value of the `dirname` as the name with the value of `ltr` or `rtl` being set by the browser.

```
<form action="page.html" method="post">
  <label
    >Fruit:
    <input type="text" name="fruit" dirname="fruit.dir" value="cherry" />
  </label>
  <input type="submit" />
</form>
<!-- page.html?fruit=cherry&fruit.dir=ltr -->
```

When the form above is submitted, the input cause both the `name` / `value` pair of `fruit=cherry` and the `dirname` / direction pair of `fruit.dir=ltr` to be sent.

`disabled`

A Boolean attribute which, if present, indicates that the user should not be able to interact with the input. Disabled inputs are typically rendered with a dimmer color or using some other form of indication that the field is not available for use.

Specifically, disabled inputs do not receive the `click` event, and disabled inputs are not submitted with the form.

> **Note:** Although not required by the specification, Firefox will by default persist the dynamic disabled state of an `<input>` across page loads. Use the `autocomplete` attribute to control this feature.

`form`

A string specifying the `<form>` element with which the input is associated (that is, its **form owner**). This string's value, if present, must match the `id` of a `<form>` element in the same document. If this attribute isn't specified, the `<input>` element is associated with the nearest containing form, if any.

The `form` attribute lets you place an input anywhere in the document but have it included with a form elsewhere in the document.

> **Note:** An input can only be associated with one form.

`formaction`

Valid for the `image` and `submit` input types only. See the submit input type for more information.

`formenctype`

Valid for the `image` and `submit` input types only. See the submit input type for more information.

`formmethod`

Valid for the `image` and `submit` input types only. See the submit input type for more information.

`formnovalidate`

Valid for the `image` and `submit` input types only. See the submit input type for more information.

`formtarget`

Valid for the `image` and `submit` input types only. See the submit input type for more information.

`height`

Valid for the `image` input button only, the `height` is the height of the image file to display to represent the graphical submit button. See the image input type.

`id`

Global attribute valid for all elements, including all the input types, it defines a unique identifier (ID) which must be unique in the whole document. Its purpose is to identify the element when linking. The value is used as the value of the `<label>`'s `for` attribute to link the label with the form control. See `<label>`.

`inputmode`

Global value valid for all elements, it provides a hint to browsers as to the type of virtual keyboard configuration to use when editing this element or its contents. Values include `none`, `text`, `tel`, `url`, `email`, `numeric`, `decimal`, and `search`.

`list`

The value given to the `list` attribute should be the `id` of a `<datalist>` element located in the same document. The `<datalist>` provides a list of predefined values to suggest to the user for this input. Any values in the list that are not compatible with the `type` are not included in the suggested options. The values provided are suggestions, not requirements: users can select from this predefined list or provide a different value.

It is valid on `text`, `search`, `url`, `tel`, `email`, `date`, `month`, `week`, `time`, `datetime-local`, `number`, `range`, and `color`.

Per the specifications, the `list` attribute is not supported by the `hidden`, `password`, `checkbox`, `radio`, `file`, or any of the button types.

Depending on the browser, the user may see a custom color palette suggested, tic marks along a range, or even an input that opens like a `<select>` but allows for non-listed values. Check out the browser compatibility table for the other input types.

See the `<datalist>` element.

`max`

Valid for `date`, `month`, `week`, `time`, `datetime-local`, `number`, and `range`, it defines the greatest value in the range of permitted values. If the `value` entered into the element exceeds this, the element fails constraint validation. If the value of the `max` attribute isn't a number, then the element has no maximum value.

There is a special case: if the data type is periodic (such as for dates or times), the value of `max` may be lower than the value of `min`, which indicates that the range may wrap around; for example, this allows you to specify a time range from 10 PM to 4 AM.

`maxlength`

Valid for `text`, `search`, `url`, `tel`, `email`, and `password`, it defines the maximum number of characters (as UTF-16 code units) the user can enter into the field. This must be an integer value `0` or higher. If no `maxlength` is specified, or an invalid value is specified, the field has no maximum length. This value must also be greater than or equal to the value of `minlength`.

The input will fail constraint validation if the length of the text entered into the field is greater than `maxlength` UTF-16 code units long. By default, browsers prevent users from entering more characters than allowed by the `maxlength` attribute. See Client-side validation for more information.

`min`

Valid for `date`, `month`, `week`, `time`, `datetime-local`, `number`, and `range`, it defines the most negative value in the range of permitted values. If the `value` entered into the element is less than this, the element fails constraint validation. If the value of the `min` attribute isn't a number, then the element has no minimum value.

This value must be less than or equal to the value of the `max` attribute. If the `min` attribute is present but is not specified or is invalid, no `min` value is applied. If the `min` attribute is valid and a non-empty value is less than the minimum allowed by the `min` attribute, constraint validation will prevent form submission. See Client-side validation for more information.

There is a special case: if the data type is periodic (such as for dates or times), the value of `max` may be lower than the value of `min`, which indicates that the range may wrap around; for example, this allows you to specify a time range from 10 PM to 4 AM.

`minlength`

Valid for `text`, `search`, `url`, `tel`, `email`, and `password`, it defines the minimum number of characters (as UTF-16 code units) the user can enter into the entry field. This must be a non-negative integer value smaller than or equal to the value specified by `maxlength`. If no `minlength` is specified, or an invalid value is specified, the input has no minimum length.

The input will fail constraint validation if the length of the text entered into the field is fewer than `minlength` UTF-16 code units long, preventing form submission. See Client-side validation for more information.

`multiple`

The Boolean `multiple` attribute, if set, means the user can enter comma separated email addresses in the email widget or can choose more than one file with the `file` input. See the email and file input type.

`name`

A string specifying a name for the input control. This name is submitted along with the control's value when the form data is submitted.

Consider the `name` a required attribute (even though it's not). If an input has no `name` specified, or `name` is empty, the input's value is not submitted with the form! (Disabled controls, unchecked radio buttons, unchecked checkboxes, and reset buttons are also not sent.)

There are two special cases:

1. `_charset_` : If used as the name of an `<input>` element of type hidden, the input's `value` is automatically set by the user agent to the character encoding being used to submit the form.
2. `isindex`: For historical reasons, the name `isindex` is not allowed.

The `name` attribute creates a unique behavior for radio buttons.

Only one radio button in a same-named group of radio buttons can be checked at a time. Selecting any radio button in that group automatically deselects any currently-selected radio button in the same group. The value of that one checked radio button is sent along with the name if the form is submitted,

When tabbing into a series of same-named group of radio buttons, if one is checked, that one will receive focus. If they aren't grouped together in source order, if one of the group is checked, tabbing into the group starts when the first one in the group is encountered, skipping all those that aren't checked. In other words, if one is checked, tabbing skips the unchecked radio buttons in the group. If none are checked, the radio button group receives focus when the first button in the same name group is reached.

Once one of the radio buttons in a group has focus, using the arrow keys will navigate through all the radio buttons of the same name, even if the radio buttons are not grouped together in the source order.

When an input element is given a `name`, that name becomes a property of the owning form element's `HTMLFormElement.elements` property. If you have an input whose `name` is set to `guest` and another whose `name` is `hat-size`, the following code can be used:

```
let form = document.querySelector("form");

let guestName = form.elements.guest;
let hatSize = form.elements["hat-size"];
```

When this code has run, `guestName` will be the `HTMLInputElement` for the `guest` field, and `hatSize` the object for the `hat-size` field.

> **Warning:** Avoid giving form elements a `name` that corresponds to a built-in property of the form, since you would then override the predefined property or method with this reference to the corresponding input.

`pattern`

Valid for `text`, `search`, `url`, `tel`, `email`, and `password`, the `pattern` attribute defines a regular expression that the input's `value` must match in order for the value to pass constraint validation. It must be a valid JavaScript regular expression, as used by the `RegExp` type, and as documented in our guide on regular expressions; the `'u'` flag is specified when compiling the regular expression, so that the pattern is treated as a sequence of Unicode code points, instead of as ASCII. No forward slashes should be specified around the pattern text.

If the `pattern` attribute is present but is not specified or is invalid, no regular expression is applied and this attribute is ignored completely. If the pattern attribute is valid and a non-empty value does not match the pattern, constraint validation will prevent form submission.

> **Note:** If using the `pattern` attribute, inform the user about the expected format by including explanatory text nearby. You can also include a `title` attribute to explain what the requirements are to match the pattern; most browsers will display this title as a tooltip. The visible explanation is required for accessibility. The tooltip is an enhancement.

See Client-side validation for more information.

`placeholder`

Valid for `text`, `search`, `url`, `tel`, `email`, `password`, and `number`, the `placeholder` attribute provides a brief hint to the user as to what kind of information is expected in the field. It should be a word or short phrase that provides a hint as to the expected type of data, rather than an explanation or prompt. The text *must not* include carriage returns or line feeds. So for example if a field is expected to capture a user's first name, and its label is "First Name", a suitable placeholder might be "e.g. Mustafa".

> **Note:** The `placeholder` attribute is not as semantically useful as other ways to explain your form, and can cause unexpected technical issues with your content. See Labels for more information.

`popovertarget`

Turns an `<input type="button">` element into a popover control button; takes the ID of the popover element to control as its value. See the Popover API landing page for more details.

`popovertargetaction`

Specifies the the action to be performed on a popover element being controlled by a control `<input type="button">`. Possible values are:

`"hide"`

The button will hide a shown popover. If you try to hide an already hidden popover, no action will be taken.

`"show"`

The button will show a hidden popover. If you try to show an already showing popover, no action will be taken.

`"toggle"`

The button will toggle a popover between showing and hidden. If the popover is hidden, it will be shown; if the popover is showing, it will be hidden. If `popovertargetaction` is omitted, `"toggle"` is the default action that will be performed by the control button.

`readonly`

A Boolean attribute which, if present, indicates that the user should not be able to edit the value of the input. The `readonly` attribute is supported by the `text`, `search`, `url`, `tel`, `email`, `date`, `month`, `week`, `time`, `datetime-local`, `number`, and `password` input types.

See the HTML attribute: `readonly` for more information.

`required`

`required` is a Boolean attribute which, if present, indicates that the user must specify a value for the input before the owning form can be submitted. The `required` attribute is supported by `text`, `search`, `url`, `tel`, `email`, `date`, `month`, `week`, `time`, `datetime-local`, `number`, `password`, `checkbox`, `radio`, and `file` inputs.

See Client-side validation and the HTML attribute: `required` for more information.

`size`

Valid for `email`, `password`, `tel`, `url`, and `text`, the `size` attribute specifies how much of the input is shown. Basically creates same result as setting CSS `width` property with a few specialities. The actual unit of the value depends on the input type. For `password` and `text`, it is a number of characters (or `em` units) with a default value of `20`, and for others, it is pixels (or `px` units). CSS `width` takes precedence over the `size` attribute.

`src`

Valid for the `image` input button only, the `src` is string specifying the URL of the image file to display to represent the graphical submit button. See the image input type.

`step`

Valid for `date`, `month`, `week`, `time`, `datetime-local`, `number`, and `range`, the `step` attribute is a number that specifies the granularity that the value must adhere to.

If not explicitly included:

- `step` defaults to 1 for `number` and `range`.
- Each date/time input type has a default `step` value appropriate for the type; see the individual input pages: `date`, `datetime-local`, `month`, `time`, and `week`.

The value must be a positive number—integer or float—or the special value `any`, which means no stepping is implied, and any value is allowed (barring other constraints, such as `min` and `max`).

If `any` is not explicitly set, valid values for the `number`, date/time input types, and `range` input types are equal to the basis for stepping — the `min` value and increments of the step value, up to the `max` value, if specified.

For example, if you have `<input type="number" min="10" step="2">`, then any even integer, `10` or greater, is valid. If omitted, `<input type="number">`, any integer is valid, but floats (like `4.2`) are not valid, because `step` defaults to `1`. For `4.2` to be valid, `step` would have had to be set to `any`, 0.1, 0.2, or any the `min` value would have had to be a number ending in `.2`, such as `<input type="number" min="-5.2">`

> **Note:** When the data entered by the user doesn't adhere to the stepping configuration, the value is considered invalid in constraint validation and will match the `:invalid` pseudoclass.

See Client-side validation for more information.

`tabindex`

Global attribute valid for all elements, including all the input types, an integer attribute indicating if the element can take input focus (is focusable), if it should participate to sequential keyboard navigation. As all input types except for input of type hidden are focusable, this attribute should not be used on form controls, because doing so would require the management of the focus order for all elements within the document with the risk of harming usability and accessibility if done incorrectly.

`title`

Global attribute valid for all elements, including all input types, containing a text representing advisory information related to the element it belongs to. Such information can typically, but not necessarily, be presented to the user as a tooltip. The title should NOT be used as the primary explanation of the purpose of the form control. Instead, use the `<label>` element with a `for` attribute set to the form control's `id` attribute. See Labels below.

`type`

A string specifying the type of control to render. For example, to create a checkbox, a value of `checkbox` is used. If omitted (or an unknown value is specified), the input type `text` is used, creating a plaintext input field.

Permitted values are listed in Input types above.

`value`

The input control's value. When specified in the HTML, this is the initial value, and from then on it can be altered or retrieved at any time using JavaScript to access the respective `HTMLInputElement` object's `value` property. The `value` attribute is always optional, though should be considered mandatory for `checkbox`, `radio`, and `hidden`.

`width`

Valid for the `image` input button only, the `width` is the width of the image file to display to represent the graphical submit button. See the image input type.

Non-standard attributes

The following non-standard attributes are also available on some browsers. As a general rule, you should avoid using them unless it can't be helped.

Attribute	Description
`autocapitalize`	A string indicating how auto-capitalization should be applied to the content of text elements. **Safari only.**
`autocorrect`	A string indicating whether autocorrect is `on` or `off`. **Safari only.**
`incremental`	Whether or not to send repeated `search` events to allow updating live search results while the user is still editing the value of the field. **WebKit and Blink only (Safari, Chrome, Opera, etc.).**
`mozactionhint`	A string indicating the type of action that will be taken when the user presses the `Enter` or `Return` key while editing the field; this is used to determine an appropriate label for that key on a virtual keyboard. **Deprecated: use `enterkeyhint` instead.**
`orient`	Sets the orientation of the range slider. **Firefox only**.
`results`	The maximum number of items that should be displayed in the drop-down list of previous search queries. **Safari only.**
`webkitdirectory`	A Boolean indicating whether to only allow the user to choose a directory (or directories, if `multiple` is also present)

`autocapitalize` ⚠

(Safari only). A string which indicates how auto-capitalization should be applied while the user is editing this field. Permitted values are:

`none`

Do not automatically capitalize any text

`sentences`

Automatically capitalize the first character of each sentence.

`words`

Automatically capitalize the first character of each word.

`characters`

Automatically capitalize every character.

`autocorrect` ⚠

(Safari only). A string which indicates whether to activate automatic correction while the user is editing this field. Permitted values are:

`on`

Enable automatic correction of typos, as well as processing of text substitutions if any are configured.

`off`

Disable automatic correction and text substitutions.

`incremental` ⚠

The Boolean attribute `incremental` is a WebKit and Blink extension (so supported by Safari, Opera, Chrome, etc.) which, if present, tells the user agent to process the input as a live search. As the user edits the value of the field, the user agent sends `search` events to the `HTMLInputElement` object representing the search box. This allows your code to update the search results in real time as the user edits the search.

If `incremental` is not specified, the `search` event is only sent when the user explicitly initiates a search (such as by pressing the `Enter` or `Return` key while editing the field).

The `search` event is rate-limited so that it is not sent more frequently than an implementation-defined interval.

`orient` ⚠

Similar to the -moz-orient non-standard CSS property impacting the `<progress>` and `<meter>` elements, the `orient` attribute defines the orientation of the range slider. Values include `horizontal`, meaning the range is rendered horizontally, and `vertical`, where the range is rendered vertically.

`results` ⚠

The `results` attribute—supported only by Safari—is a numeric value that lets you override the maximum number of entries to be displayed in the `<input>` element's natively-provided drop-down menu of previous search queries.

The value must be a non-negative decimal number. If not provided, or an invalid value is given, the browser's default maximum number of entries is used.

`webkitdirectory` ⚠

The Boolean `webkitdirectory` attribute, if present, indicates that only directories should be available to be selected by the user in the file picker interface. See `HTMLInputElement.webkitdirectory` for additional details and examples.

Though originally implemented only for WebKit-based browsers, `webkitdirectory` is also usable in Microsoft Edge as well as Firefox 50 and later. However, even though it has relatively broad support, it is still not standard and should not be used unless you have no alternative.

Methods

The following methods are provided by the `HTMLInputElement` interface which represents `<input>` elements in the DOM. Also available are those methods specified by the parent interfaces, `HTMLElement`, `Element`, `Node`, and `EventTarget`.

`checkValidity()`

Returns `true` if the element's value passes validity checks; otherwise, returns `false` and fires an `invalid` event at the element.

`reportValidity()`

Returns `true` if the element's value passes validity checks; otherwise, returns `false`, fires an `invalid` event at the element, and (if the event isn't canceled) reports the problem to the user.

`select()`

Selects the entire content of the `<input>` element, if the element's content is selectable. For elements with no selectable text content (such as a visual color picker or calendar date input), this method does nothing.

`setCustomValidity()`

Sets a custom message to display if the input element's value isn't valid.

`setRangeText()`

Sets the contents of the specified range of characters in the input element to a given string. A `selectMode` parameter is available to allow controlling how the existing content is affected.

`setSelectionRange()`

Selects the specified range of characters within a textual input element. Does nothing for inputs which aren't presented as text input fields.

`stepDown()`

Decrements the value of a numeric input by one, by default, or by the specified number of units.

`stepUp()`

Increments the value of a numeric input by one or by the specified number of units.

CSS

Inputs, being replaced elements, have a few features not applicable to non form elements. There are CSS selectors that can specifically target form controls based on their UI features, also known as UI pseudo-classes. The input element can also be targeted by type with attribute selectors. There are some properties that are especially useful as well.

UI pseudo-classes

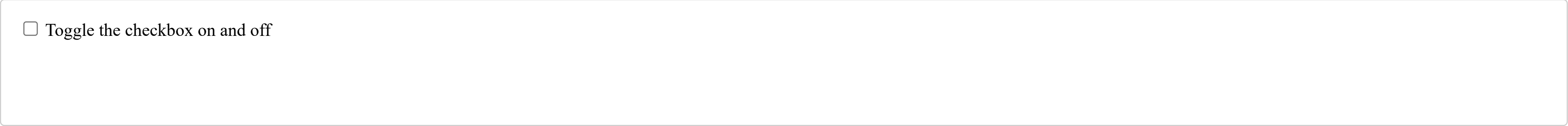
Captions super relevant to the `<input>` element:

Pseudo-class	Description
`:enabled`	Any currently enabled element that can be activated (selected, clicked on, typed into, etc.) or accept focus and also has a disabled state, in which it can't be activated or accept focus.
`:disabled`	Any currently disabled element that has an enabled state, meaning it otherwise could be activated (selected, clicked on, typed into, etc.) or accept focus were it not disabled.
`:read-only`	Element not editable by the user
`:read-write`	Element that is editable by the user.
`:placeholder-shown`	Element that is currently displaying `placeholder` text, including `<input>` and `<textarea>` elements with the `placeholder` attribute present that has, as yet, no value.
`:default`	Form elements that are the default in a group of related elements. Matches checkbox and radio input types that were checked on page load or render.
`:checked`	Matches checkbox and radio input types that are currently checked (and the (`<option>` in a `<select>` that is currently selected).
`:indeterminate`	checkbox elements whose indeterminate property is set to true by JavaScript, radio elements, when all radio buttons with the same name value in the form are unchecked, and `<progress>` elements in an indeterminate state
`:valid`	Form controls that can have constraint validation applied and are currently valid.
`:invalid`	Form controls that have constraint validation applied and are currently not valid. Matches a form control whose value doesn't match the constraints set on it by its attributes, such as `required`, `pattern`, `step` and `max`.
`:in-range`	A non-empty input whose current value is within the range limits specified by the `min` and `max` attributes and the `step`.
`:out-of-range`	A non-empty input whose current value is NOT within the range limits specified by the `min` and `max` attributes or does not adhere to the `step` constraint.
`:required`	`<input>`, `<select>`, or `<textarea>` element that has the `required` attribute set on it. Only matches elements that can be required. The attribute included on a non-requirable element will not make for a match.
`:optional`	`<input>`, `<select>`, or `<textarea>` element that does NOT have the `required` attribute set on it. Does not match elements that can't be required.
`:blank`	`<input>` and `<textarea>` elements that currently have no value.
`:user-invalid`	Similar to `:invalid`, but is activated on blur. Matches invalid input but only after the user interaction, such as by focusing on the control, leaving the control, or attempting to submit the form containing the invalid control.

Pseudo-classes example

We can style a checkbox label based on whether the checkbox is checked or not. In this example, we are styling the `color` and `font-weight` of the `<label>` that comes immediately after a checked input. We haven't applied any styles if the `input` is not checked.

```
input:checked + label {
  color: red;
  font-weight: bold;
}
```



☐ Toggle the checkbox on and off

Attribute selectors

It is possible to target different types of form controls based on their `type` using attribute selectors. CSS attribute selectors match elements based on either just the presence of an attribute or the value of a given attribute.

```
/* matches a password input */
input[type="password"] {
}

/* matches a form control whose valid values are limited to a range of values*/
input[min][max] {
}

/* matches a form control with a pattern attribute */
input[pattern] {
}
```

::placeholder

By default, the appearance of placeholder text is a translucent or light gray. The `::placeholder` pseudo-element is the input's `placeholder` text. It can be styled with a limited subset of CSS properties.

```
::placeholder {
  color: blue;
}
```

Only the subset of CSS properties that apply to the `::first-line` pseudo-element can be used in a rule using `::placeholder` in its selector.

appearance

The `appearance` property enables the displaying of (almost) any element as a platform-native style based on the operating system's theme as well as the removal of any platform-native styling with the `none` value.

You could make a `<div>` look like a radio button with `div {appearance: radio;}` or a radio look like a checkbox with `[type="radio"] {appearance: checkbox;}`, but don't.

Setting `appearance: none` removes platform native borders, but not functionality.

caret-color

A property specific to text entry-related elements is the CSS `caret-color` property, which lets you set the color used to draw the text input caret:

HTML

```
<label for="textInput">Note the red caret:</label>
<input id="textInput" class="custom" size="32" />
```

CSS

```
input.custom {
  caret-color: red;
  font: 16px "Helvetica", "Arial", "sans-serif";
}
```

Result

Note the red caret:

object-position and object-fit

In certain cases (typically involving non-textual inputs and specialized interfaces), the `<input>` element is a replaced element. When it is, the position and size of the element's size and positioning within its frame can be adjusted using the CSS `object-position` and `object-fit` properties

Styling

For more information about adding color to elements in HTML, see:

- Applying color to HTML elements using CSS.

Also see:

- Styling HTML forms
- Advanced styling for HTML forms and
- the compatibility table of CSS properties.

Additional features

Labels

Labels are needed to associate assistive text with an `<input>`. The `<label>` element provides explanatory information about a form field that is *always* appropriate (aside from any layout concerns you have). It's never a bad idea to use a `<label>` to explain what should be entered into an `<input>` or `<textarea>`.

Associated labels

The semantic pairing of `<input>` and `<label>` elements is useful for assistive technologies such as screen readers. By pairing them using the `<label>`'s `for` attribute, you bond the label to the input in a way that lets screen readers describe inputs to users more precisely.

It does not suffice to have plain text adjacent to the `<input>` element. Rather, usability and accessibility requires the inclusion of either implicit or explicit `<label>`:

```
<!-- inaccessible -->
<p>Enter your name: <input id="name" type="text" size="30" /></p>

<!-- implicit label -->
<p>
  <label>Enter your name: <input id="name" type="text" size="30" /></label>
</p>

<!-- explicit label -->
<p>
  <label for="name">Enter your name: </label>
  <input id="name" type="text" size="30" />
</p>
```

The first example is inaccessible: no relationship exists between the prompt and the `<input>` element.

In addition to an accessible name, the label provides a larger 'hit' area for mouse and touch screen users to click on or touch. By pairing a `<label>` with an `<input>`, clicking on either one will focus the `<input>`. If you use plain text to "label" your input, this won't happen. Having the prompt part of the activation area for the input is helpful for people with motor control conditions.

As web developers, it's important that we never assume that people will know all the things that we know. The diversity of people using the web—and by extension your website—practically guarantees that some of your site's visitors will have some variation in thought processes and/or circumstances that leads them to interpret your forms very differently from you without clear and properly-presented labels.

Placeholders are not accessible

The `placeholder` attribute lets you specify text that appears within the `<input>` element's content area itself when it is empty. The placeholder should never be required to understand your forms. It is not a label, and should not be used as a substitute, because it isn't. The placeholder is used to provide a hint as to what an inputted value should look like, not an explanation or prompt.

Not only is the placeholder not accessible to screen readers, but once the user enters any text into the form control, or if the form control already has a value, the placeholder disappears. Browsers with automatic page translation features may skip over attributes when translating, meaning the `placeholder` may not get translated.

Note: Don't use the `placeholder` attribute if you can avoid it. If you need to label an `<input>` element, use the `<label>` element.

Client-side validation

Warning: Client-side validation is useful, but it does *not* guarantee that the server will receive valid data. If the data must be in a specific format, *always* verify it also on the server-side, and return a `400` HTTP response if the format is invalid.

In addition to using CSS to style inputs based on the `:valid` or `:invalid` UI states based on the current state of each input, as noted in the UI pseudo-classes section above, the browser provides for client-side validation on (attempted) form submission. On form submission, if there is a form control that fails constraint validation, supporting browsers will display an error message on the first invalid form control; displaying a default message based on the error type, or a message set by you.

Some input types and other attributes place limits on what values are valid for a given input. For example, `<input type="number" min="2" max="10" step="2">` means only the number 2, 4, 6, 8, or 10 are valid. Several errors could occur, including a `rangeUnderflow` error if the value is less than 2, `rangeOverflow` if greater than 10, `stepMismatch` if the value is a number between 2 and 10, but not an even integer (does not match the requirements of the `step` attribute), or `typeMismatch` if the value is not a number.

For the input types whose domain of possible values is periodic (that is, at the highest possible value, the values wrap back around to the beginning rather than ending), it's possible for the values of the `max` and `min` properties to be reversed, which indicates that the range of permitted values starts at `min`, wraps around to the lowest possible value, then continues on until `max` is reached. This is particularly useful for dates and times, such as when you want to allow the range to be from 8 PM to 8 AM:

```
<input type="time" min="20:00" max="08:00" name="overnight" />
```

Specific attributes and their values can lead to a specific error `ValidityState`:

Validity object errors depend on the `<input>` attributes and their values:

Attribute	Relevant property	Description
`max`	`validityState.rangeOverflow`	Occurs when the value is greater than the maximum value as defined by the `max` attribute
`maxlength`	`validityState.tooLong`	Occurs when the number of characters is greater than the number allowed by the `maxlength` property
`min`	`validityState.rangeUnderflow`	Occurs when the value is less than the minimum value as defined by the `min` attribute
`minlength`	`validityState.tooShort`	Occurs when the number of characters is less than the number required by the `minlength` property
`pattern`	`validityState.patternMismatch`	Occurs when a pattern attribute is included with a valid regular expression and the `value` does not match it.
`required`	`validityState.valueMissing`	Occurs when the `required` attribute is present but the value is `null` or radio or checkbox is not checked.
`step`	`validityState.stepMismatch`	The value doesn't match the step increment. Increment default is `1`, so only integers are valid on `type="number"` is step is not included. `step="any"` will never throw this error.
`type`	`validityState.typeMismatch`	Occurs when the value is not of the correct type, for example an email does not contain an `@` or a url doesn't contain a protocol.

If a form control doesn't have the `required` attribute, no value, or an empty string, is not invalid. Even if the above attributes are present, with the exception of `required`, an empty string will not lead to an error.

We can set limits on what values we accept, and supporting browsers will natively validate these form values and alert the user if there is a mistake when the form is submitted.

In addition to the errors described in the table above, the `validityState` interface contains the `badInput`, `valid`, and `customError` boolean readonly properties. The validity object includes:

- `validityState.valueMissing`
- `validityState.typeMismatch`
- `validityState.patternMismatch`
- `validityState.tooLong`
- `validityState.tooShort`
- `validityState.rangeUnderflow`
- `validityState.rangeOverflow`
- `validityState.stepMismatch`
- `validityState.badInput`
- `validityState.valid`
- `validityState.customError`

For each of these Boolean properties, a value of `true` indicates that the specified reason validation may have failed is true, with the exception of the `valid` property, which is `true` if the element's value obeys all constraints.

If there is an error, supporting browsers will both alert the user and prevent the form from being submitted. A word of caution: if a custom error is set to a truthy value (anything other than the empty string or `null`), the form will be prevented from being submitted. If there is no custom error message, and none of the other properties return true, `valid` will be true, and the form can be submitted.

```
function validate(input) {
  let validityState_object = input.validity;
  if (validityState_object.valueMissing) {
    input.setCustomValidity("A value is required");
  } else if (validityState_object.rangeUnderflow) {
    input.setCustomValidity("Your value is too low");
  } else if (validityState_object.rangeOverflow) {
    input.setCustomValidity("Your value is too high");
  } else {
    input.setCustomValidity("");
  }
}
```

The last line, setting the custom validity message to the empty string is vital. If the user makes an error, and the validity is set, it will fail to submit, even if all the values are valid, until the message is `null`.

Custom validation error example

If you want to present a custom error message when a field fails to validate, you need to use the Constraint Validation API available on `<input>` (and related) elements. Take the following form:

```
<form>
  <label for="name">Enter username (upper and lowercase letters): </label>
  <input type="text" name="name" id="name" required pattern="[A-Za-z]+" />
  <button>Submit</button>
</form>
```

The basic HTML form validation features will cause this to produce a default error message if you try to submit the form with either no valid filled in, or a value that does not match the `pattern`.

If you wanted to instead display custom error messages, you could use JavaScript like the following:

```
const nameInput = document.querySelector("input");

nameInput.addEventListener("input", () => {
  nameInput.setCustomValidity("");
  nameInput.checkValidity();
});

nameInput.addEventListener("invalid", () => {
  if (nameInput.value === "") {
    nameInput.setCustomValidity("Enter your username!");
  } else {
    nameInput.setCustomValidity(
      "Usernames can only contain upper and lowercase letters. Try again!"
    );
  }
});
```

The example renders like so:

Enter username (upper and lowercase letters):

In brief:

- We check the valid state of the input element every time its value is changed by running the `checkValidity()` method via the `input` event handler.
- If the value is invalid, an `invalid` event is raised, and the `invalid` event handler function is run. Inside this function we work out whether the value is invalid because it is empty, or because it doesn't match the pattern, using an `if ()` block, and set a custom validity error message.
- As a result, if the input value is invalid when the submit button is pressed, one of the custom error messages will be shown.
- If it is valid, it will submit as you'd expect. For this to happen, the custom validity has to be cancelled, by invoking `setCustomValidity()` with an empty string value. We therefore do this every time the `input` event is raised. If you don't do this, and a custom validity was previously set, the input will register as invalid, even if it currently contains a valid value on submission.

> **Note:** Always validate input constraints both client side and server side. Constraint validation doesn't remove the need for validation on the *server side*. Invalid values can still be sent by older browsers or by bad actors.

> **Note:** Firefox supported a proprietary error attribute — `x-moz-errormessage` — for many versions, which allowed you set custom error messages in a similar way. This has been removed as of version 66 (see Firefox bug 1513890).

Localization

The allowed inputs for certain `<input>` types depend on the locale. In some locales, 1,000.00 is a valid number, while in other locales the valid way to enter this number is 1.000,00.

Firefox uses the following heuristics to determine the locale to validate the user's input (at least for `type="number"`):

- Try the language specified by a `lang`/`xml:lang` attribute on the element or any of its parents.
- Try the language specified by any `Content-Language` HTTP header. Or,
- If none specified, use the browser's locale.

Technical summary

Content categories	Flow content, listed, submittable, resettable, form-associated element, phrasing content. If the `type` is not `hidden`, then labelable element, palpable content.
Permitted content	None; it is a void element.
Tag omission	Must have a start tag and must not have an end tag.
Permitted parents	Any element that accepts phrasing content.
Implicit ARIA role	• `type=button`: `button` • `type=checkbox`: `checkbox` • `type=email` ◦ with no `list` attribute: `textbox` ◦ with `list` attribute: `combobox` • `type=image`: `button` • `type=number`: `spinbutton` • `type=radio`: `radio` • `type=range`: `slider` • `type=reset`: `button` • `type=search` ◦ with no `list` attribute: `searchbox` ◦ with `list` attribute: `combobox` • `type=submit`: `button` • `type=tel` ◦ with no `list` attribute: `textbox` ◦ with `list` attribute: `combobox` • `type=text` ◦ with no `list` attribute: `textbox` ◦ with `list` attribute: `combobox` • `type=url` ◦ with no `list` attribute: `textbox` ◦ with `list` attribute: `combobox` • `type=color|date|datetime-local|file|hidden|month|password|time|week`: no corresponding role
Permitted ARIA roles	• `type=button`: `checkbox`, `combobox`, `link`, `menuitem`, `menuitemcheckbox`, `menuitemradio`, `option`, `radio`, `switch`, `tab` • `type=checkbox`: `button` when used with `aria-pressed`, `menuitemcheckbox`, `option`, `switch` • `type=image`: `link`, `menuitem`, `menuitemcheckbox`, `menuitemradio`, `radio`, `switch` • `type=radio`: `menuitemradio` • `type=text` with no `list` attribute: `combobox`, `searchbox`, `spinbutton` • `type=color|date|datetime-local|email|file|hidden|` `month|number|password|range|reset|search|submit|tel|url|week` or `text` with `list` attribute: no `role` permitted
DOM interface	`HTMLInputElement`

Accessibility concerns

Labels

When including inputs, it is an accessibility requirement to add labels alongside. This is needed so those who use assistive technologies can tell what the input is for. Also, clicking or touching a label gives focus to the label's associated form control. This improves the accessibility and usability for sighted users, increases the area a user can click or touch to activate the form control. This is especially useful (and even needed) for radio buttons and checkboxes, which are tiny. For more information about labels in general see Labels .

The following is an example of how to associate the `<label>` with an `<input>` element in the above style. You need to give the `<input>` an `id` attribute. The `<label>` then needs a `for` attribute whose value is the same as the input's `id`.

```
<label for="peas">Do you like peas?</label>
<input type="checkbox" name="peas" id="peas" />
```

Size

Interactive elements such as form input should provide an area large enough that it is easy to activate them. This helps a variety of people, including people with motor control issues and people using non-precise forms of input such as a stylus or fingers. A minimum interactive size of 44×44 CSS pixels is recommended.

- Understanding Success Criterion 2.5.5: Target Size | W3C Understanding WCAG 2.1
- Target Size and 2.5.5 | Adrian Roselli
- Quick test: Large touch targets - The A11Y Project

Specifications

Specification
HTML Standard # the-input-element

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS
`input`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1 *	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4 *	✓ Opera Android 12.1	✓ Safari on iOS
`accept`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`align` 🗑	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`alt`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`capture`	✗ Chrome No	✗ Edge No	✗ Firefox No	✗ Opera No	✗ Safari No	✓ Chrome Android 25	✓ Firefox for Android 79	✓ Opera Android 14	✓ Safari on iOS
`checked`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`dirname`	✓ Chrome 17	✓ Edge 79	✗ Firefox No	✓ Opera 12.1	✓ Safari 6	✓ Chrome Android 18	✗ Firefox for Android No	✓ Opera Android 12.1	✓ Safari on iOS
`disabled`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`form`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`formaction`	✓ Chrome 9	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 4
`formenctype`	✓ Chrome 9	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 4
`formmethod`	✓ Chrome 9	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 4
`formnovalidate`	✓ Chrome 4	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`formtarget`	✓ Chrome 9	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 4
`list`	✓ Chrome 20	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 12.1	✓ Chrome Android 25	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 12
`max`	✓ Chrome 4	✓ Edge 12	✓ Firefox 16	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 16	✓ Opera Android 12.1	✓ Safari on iOS
`maxlength`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`min`	✓ Chrome 4	✓ Edge 12	✓ Firefox 16	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 16	✓ Opera Android 12.1	✓ Safari on iOS
`minlength`	✓ Chrome 40	✓ Edge 17	✓ Firefox 51	✓ Opera 27	✓ Safari 10.1	✓ Chrome Android 40	✓ Firefox for Android 51	✓ Opera Android 27	✓ Safari on iOS 10
`multiple`	✓ Chrome 2	✓ Edge 12	✓ Firefox 3.6	✓ Opera 12.1	✓ Safari 4	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 3
`name`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`pattern`	✓ Chrome 4	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`placeholder`	✓ Chrome 3	✓ Edge 12	✓ Firefox 4	✓ Opera 12.1	✓ Safari 4	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS 3
`readonly`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS
`src`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`step`	✓ Chrome 5	✓ Edge 12	✓ Firefox 16	✓ Opera 12.1	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 16	✓ Opera Android 12.1	✓ Safari on iOS
`type="button"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="checkbox"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="color"`	✓ Chrome 20	✓ Edge 14	✓ Firefox 29	✓ Opera 12	✓ Safari 12.1	✓ Chrome Android 25	◆ Firefox for Android 27	✓ Opera Android 12	✓ Safari on iOS 12
`type_color.autocomplete`	✓ Chrome 20	✓ Edge 14	✗ Firefox No *	✓ Opera 15	? Safari ?	✓ Chrome Android 25	✗ Firefox for Android No *	✓ Opera Android 14	? Safari on iOS
`type_color.list`	✓ Chrome 20	✓ Edge 14	◆ Firefox 110	✓ Opera 15	✓ Safari 12.1	✓ Chrome Android 25	✗ Firefox for Android No *	✓ Opera Android 14	✓ Safari on iOS 12
`type="date"`	✓ Chrome 20	✓ Edge 12	✓ Firefox 57	✓ Opera 11	✓ Safari 14.1	✓ Chrome Android 25	✓ Firefox for Android 57	✓ Opera Android 11	✓ Safari on iOS
`type="datetime-local"`	✓ Chrome 20	✓ Edge 12	✓ Firefox 93	✓ Opera 11	✓ Safari 14.1	✓ Chrome Android 25	✓ Firefox for Android 93	✓ Opera Android 11	✓ Safari on iOS
`type="email"`	✓ Chrome 5	✓ Edge 12	✓ Firefox 1	✓ Opera 11	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 11	✓ Safari on iOS 3
`type="file"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1 *	✓ Opera 11	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 11	✓ Safari on iOS
`type="hidden"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 2	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="image"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="month"`	✓ Chrome 20	✓ Edge 12	✗ Firefox No *	✓ Opera 11	✗ Safari No *	✓ Chrome Android 25	✓ Firefox for Android 18	✓ Opera Android 14	✓ Safari on iOS Y
`type="number"`	✓ Chrome 7	✓ Edge 12	✓ Firefox 29	✓ Opera 15	✓ Safari 5.1	✓ Chrome Android 18	✓ Firefox for Android 29	✓ Opera Android 14	✓ Safari on iOS
`type="password"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 2	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
Special handling of password inputs in insecure login pages ⚠	✗ Chrome No	✗ Edge No	✓ Firefox 52	✗ Opera No	✗ Safari No	✗ Chrome Android No	✓ Firefox for Android 52	✗ Opera Android No	✗ Safari on iOS I
`type="radio"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="range"`	✓ Chrome 4	✓ Edge 12	✓ Firefox 23	✓ Opera 11	✓ Safari 3.1	✓ Chrome Android 57	✓ Firefox for Android 52	✓ Opera Android 11	✓ Safari on iOS
Tick mark support	✓ Chrome Yes	✓ Edge 79	✓ Firefox 109	✓ Opera Yes	✓ Safari 12.1	✓ Chrome Android Yes	✓ Firefox for Android 109	✓ Opera Android Yes	✓ Safari on iOS 12
Vertically-oriented slider support	✓ Chrome Yes *	✓ Edge 12 *	✗ Firefox No *	✓ Opera Yes *	✓ Safari Yes *	✓ Chrome Android Yes *	✗ Firefox for Android No *	✓ Opera Android Yes *	✓ Safari on iOS Yes
`type="reset"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1 *	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4 *	✓ Opera Android 14	✓ Safari on iOS
`type="search"`	✓ Chrome 5	✓ Edge 12	✓ Firefox 4	✓ Opera 10.6	✓ Safari 5	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS 4
`type="submit"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1 *	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4 *	✓ Opera Android 14	✓ Safari on iOS
`type="tel"`	✓ Chrome 3 *	✓ Edge 12	✓ Firefox Yes	✓ Opera 11	✓ Safari 4 *	✓ Chrome Android 18	✓ Firefox for Android Yes	✓ Opera Android 11	✓ Safari on iOS
`type="text"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 15	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="time"`	✓ Chrome 20	✓ Edge 12	✓ Firefox 57	✓ Opera 10	✓ Safari 14.1	✓ Chrome Android 25	✓ Firefox for Android 57	✓ Opera Android 10.1	✓ Safari on iOS
`type="url"`	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 11	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 14	✓ Safari on iOS
`type="week"`	✓ Chrome 20	✓ Edge 12	✗ Firefox No *	✓ Opera 11	✗ Safari No *	✓ Chrome Android 25	✓ Firefox for Android 18	✓ Opera Android 14	✗ Safari No on iOS
`usemap` 🗑	✓ Chrome 1	✓ Edge 12	✓ Firefox 1	✓ Opera 12.1	✓ Safari 1	✓ Chrome Android 18	✓ Firefox for Android 4	✓ Opera Android 12.1	✓ Safari on iOS
`x-moz-errormessage` 🗑 ⚠ attribute	✗ Chrome No	✗ Edge No	✗ Firefox No	✗ Opera No	✗ Safari No	✗ Chrome Android No	✗ Firefox for Android No	✗ Opera Android No	✗ Safari on iOS I

Tip: you can click/tap on a cell for more information.

✓ Full support ◈ Partial support ⊗ No support ? Compatibility unknown ⚠ Non-standard. Check cross-browser support before using. 🗑 Deprecated. Not for use in new websites.

✱ See implementation notes. ··· Has more compatibility info.

See also

- Form constraint validation
- Your first HTML form
- How to structure an HTML form
- The native form widgets
- Sending form data
- Form data validation
- How to build custom form widgets
- HTML forms in legacy browsers
- Styling HTML forms
- Advanced styling for HTML forms
- CSS property compatibility table

This page was last modified on May 3, 2023 by MDN contributors.