HTML Form `<input>` Element

The `<input>` HTML element is used to create iteractive 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 combination of input types and attributes.

HTML `<input>` Element types

How an `<input>` works varies considerably depending on the value of its `type` attribute, If this attribute is not specified, the default type adopted is text.

Type	Description	Basic Examples
input	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 seleccted/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.	dd - mm - yyyy
datetime-local	A control for entering a date and time, with noe time zone, Opens a date and time, with no time zone, Opens a date picker or numeric wheels for date-and time-components when acive in supporting browsers.	dd - mm - yyyy -- : -- --
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.	Email Address
file	A control that lets the user select a file. Use the `accept` attribute to define the types of the files that the control can select.	Choose File No file chosen

hidden	A control that is not displayed but hwose 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.	---------, ----
number	A Control for entering a numbrer. Displays a spinner and adds default validation when supported. Displays a numeric keypad in some device with dynamic keyboard.	
password	a Single-line text field whose value is obscured/hidden. 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 reset the content of the form to default values.	reset
search	A single-line text field for entering search strings. Line-breaks are automaticallly 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 kepads.	search
submit	A button that submit the form	submit
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.	input text
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.	url
week	A control for entering a date consisting of a week-year number and a week number with no time zone.	Week --, ----

HTML `<input>` Element 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 spcecific 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.

.

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.
autocomplete	`checkbox`, `radio`, and buttons.	Hint for form autofill feature.
file	`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 directionally in from 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` , `range`	Maximum value
maxlength	`text` , `search` , `url` , `tel` , `email` , `password`	Maximum length (number of characters) of `value` .
min	`date` , `month` , `week` , `time` , `datetime-local` , `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.
readonly	all except `hidden` , `range` , `color` , `checkbox` , `radio` , and buttons.	Bollean. 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` , `range`	Incremental values that are valid
type	all	Type of form control
value	all	The initial vlue of the control

width	image	Same as `width` attribute for `<img>`

Individual Attributes

accept

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

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.

autocomplete

The `autocomplete` attribute takes as its value as 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.

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.

autofocus

A boolean attribute which, if present, indicate 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 many gain focus before the `DOMContentLoaded` event fired.

No more than one element in the document may have the `autofocus` attribute. if pur on more than one element, the first on 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 coan confuse visually-impaired people using screen-reading technology and people with congnitive impairments. When `autofocus` is assigned, screen-readers "teleport" their user to the form control without warning them beforehand.

We should 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 not announce anything before the label, and the sighted user on a smell device will equally miss the context created by the preceding content.

capture

It is 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 supproting scenarios.

checked

valid for both `radio` and `checkbox` types, `checked` is a boolean attribute. if present on a `radio`, it indicates that the radio button is the currentyly 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

> **Note:** Unlike other input controls a checkboxes and radio buttons value are only included in the submitted data if they are cureently `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 button 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 seet by the browser.

disabled

A Boolean attribute which, if present, indicate that the user should not be able to interact 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.

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.

formenctype

Valid for the `image` and `input types only.`

formmethod

Valid for the `image` and `input types only.`

fromnovalidate

Valid for the `image` and `input types only.`

formtarget

Valid for the `image` and `input types only.`

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.

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 vlaue of the `<label's>` `for` attribute to link the label with the form control.

inputmode

Global value valid for all elements, it provides a hint to browsers as to the types of virtual keyboard configuration to use when 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 `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 differnt value.

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

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

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 numbrer, then the element has no maximum value.

maxlength

Valid for `text`, `search`, `url`, `tel`, `email`, and `password`, it defines the maximum number of characters (as UTF-16 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 units long. By default, browsers prevent users from entering more characters than allowed by the `maxlength` attribute.

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 constranit validation. if the value of the `min` attribute isn't a number, then the element has no minimuim 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 non-empty value is less than the minimum allowed by the min attribute, constraint validation will prevent from submission.

There is a special case: if the data type is periodic (such as for date or times.) the value of `max` may be lower than the value of `min`, which indicates that the range may wrap around.

minlength

Valid for `text`, `search`, `url`, `tel`, `email`, and `password`, it defines the minimum number of characters (as UTF-16 code units) the useer can enter into the entry field. this must be an non-negative integer value samller 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 fill constraint validation if the length of the text ent3ered into the field is fewer than `minlength` UTF-16 code units long, preventing form submission.

multiple

The Boolean `multiple` attribute, if set, means the user can enter comma separated email address in the email widget or can choose more than on file with the `file` input.

name

A string specifying a name for the input conrol. This name is submitted along with the contro;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.)

Ther are two special case:

- `_charset_` : if used as the name of an `<input>` element of type `hidden` , the input's `value` is automaticallly set by the user agent to the character encoding being used to submit the form.

- `isindex` : For historical reasons, the name `isindex` is not allowed.

The `name` attribute creates a nuique 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 on e 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 thorugh 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's is given an `name` , that name becomes a property of the owning form element's HTMLFormElement.elements property. if we 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.element.guest ;

let hatsize = form.elements [ "hat-size" ];
```

When this code runs, `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 we would then ouveride the predefined property or method with this reference to the corresponding input.

`pattern`

Valid for `text`, `search`, `url`, `tel`, `email`, `password`, and `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. 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 non-empty value does not match the pattern, `constraint validation` will prevent form subission.

> **Note:** if using the `pattern` attribute, iform the suer about the expected format by including explanatory text nearby. we can also include a `title` attribute to explain what the requirments are to match the pattern; most browsers will display this title as a tooltip. The visible explanation is required for accessibiltiy. The tolltip is an enhancement.

`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 *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 lable is "First Name", a suitale 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 our content.

`readonly`

A Boolean attribute which, if present, indicates that the user should not be able to edit the value of the input. The `attribute` is supported by the `text`, `search`, `url`, `tel`,

`email`, `date`, `month`, `week`, `time`, `datetime-local`, `number`, and `password` input types.

required

`required` is a boolean which, if present, indicate that the user must specify a value for the input before the owining 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.

size

Valid for `email`, `password`, `tel`, `url`, and `text`, the `size`, attribute specifies how much of the input is shown. Basically creates sam result as setting CSS `width` property with a few specialities. The actual unit of the value dependes on the input type. For `password` and `text`, it is a number of character (of `em` units) with a default value of 20, and for other, it is pixels (of `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.

step

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

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

> **For example:** if we have `<input type="number" min="10" step="2">`, then any even integer, 10 or greater, is valid. if ommited, `<input type="number">`, any integer iis valid, but floats (like `4.2`) are not valid, because `step` defaults to `1`. for `4.2` to be valid, `step` would be have 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">`

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.

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.

value

The input control's value. Whe 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 fiel to display to represent the graphical submit button.

Method on HTML `<input>` Element

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` at the element.

reprotValidity()

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

Select 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 `<input>` Element

Inputs, being replaced elements, have a few feature not applicable to non form elements. There are css selectros that can specifically target form control 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 usefual as well.

UI pseudo-classes

Pseudo-classes	Description
`:enabled`	Any currently enabled element that can be activated (selected, clicked on, typed into, etc.) or accept focus and also ahs a disabled state, in which it can't be activated or accept focus.
`:disabled`	Any currently disabled element that ahs an enabled state, meaning it other wise could be activated (selected, clicked on, typed into, etc.) or accept focus were it not disabled.
`:read-only`	Element no 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 of 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 thesame 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-emtpy 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 lements that can be required. The attribute included on a non-requirable elements 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 ser interaction, such as by focousing on the control, leaving the control, or attrmpting 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.

☐ Toothe the checkbox on and off.
☐ Toothe the checkbox on and off.
☐ Toothe 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 a attribute or the value of a given attribute.

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

/* 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 gary. The `::placeholder` pseudo-element is the input's `placeholder text`. it can be styled with a limited subset of CSS properties.

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` proprty 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.

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

Setting `appearance: non` removes platform native borders, but not functionality.

caret-color

A proprty specific to text entry-related elements in the css `caret-color` proprty, which lets us set the color used to draw the text input 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 adjust using the css `object-position` and `object-fit` 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 lyout 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 lables

The semantic pairing of `<input>` and `<label>` elements is useful for assistive technologies such as screen reader. by pairing them using the `<label>` 's `for` attribute, we bond the label to the input in a way that lets screen reader describe inputs to users 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>
    </p>


    <p>
        <input id="name" type="text" size="30">
    </p>
```

→ The first example is inaccessible: no relationship exist 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 we use plain text to ('label') our input, this won't happen.

Having the prompt part of the acitvation are for the input is helpful for people with motor control conditions.

> **Note:** → 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 our wibsite-- practically guarantees that some of our site's visitors will have some variation in thought processes and/or circumstances that leads them to interpret our forms very differently from us without clear and properly-presented labels.

Placeholders are not accessible

The `placeholder` attribute lets us specify text that apperas within the `<input>` element's content area itself when it is empty. The placeholder should never be required to understand our form. 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 reader, 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 featurs may skip over attribute when translating, meaning the `placeholder` may not translated.

> **Note:** Don't use the `placeholder` attribute if we can avoid it. if we 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 responce` 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 us.

→ Some input tyupes 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. Aeveral errors could occur, including a `rangeUnderflow` if the value is a number between 2 and 10, but not an even integer (does not match the requirments of the `step` attribute) or `typeMismatch` if the value is not a number.

Periodic Inputs

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 tahn 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 us want to allow the range to be from 8 AM to 8 PM:

```
<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 character 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.
`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, os only integers are

		valie 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 a 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 attribute are present, with the exception of required, and 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 additon to the errors described in the table above, the `validityState` interface contains the `badInput`, `valid`, and `customError` boolean readonly property. 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` indicate that the specified reason validation may have faied 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 emtpy string or `null`), the form will be prvented from beinng 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.

Accessibility Concerns

Labels

when inluding inputs, it si an accessibility requirement to add labels along side. 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, increase the area a user can click or touch to activate the form control. This is especially userful (and even needed) for radio buttons and checkboxes, which are tiny.

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 issue and people using non-precise form of input such as stylus or fingers. A minimum interactive size of 44x44 css pixles is recommended.