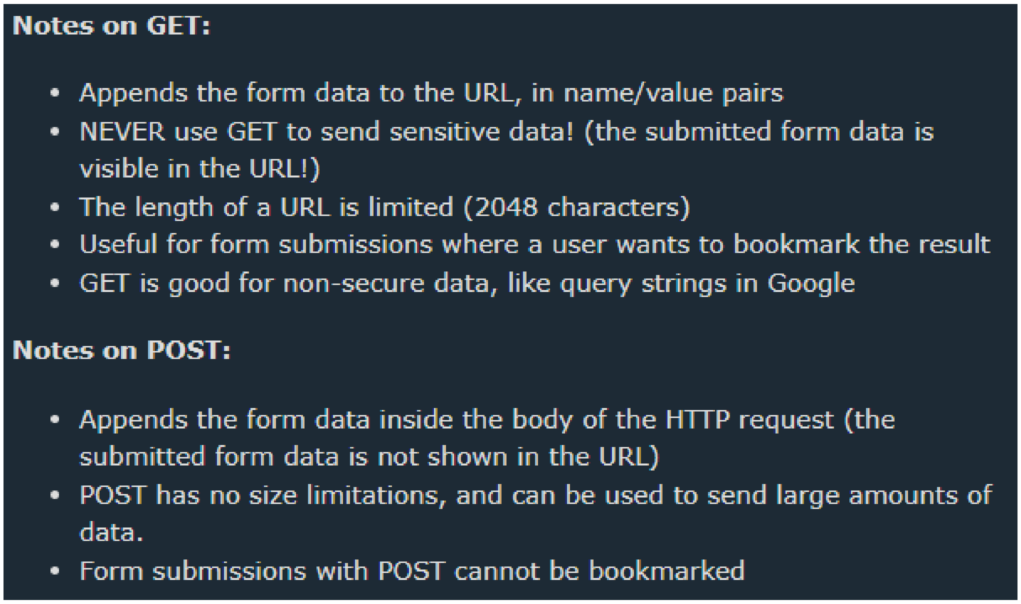
# FORM BASIC

## A. Form element

1) action: tells the form where it should send its data to be processed. If the ‘action’ attribute is omitted, the action is set to the current page.
2) method: tells the browser which HTTP request method it should use to submit the form.
   - GET, when we want to retrieve something from a server. For example, Google uses a GET request when you search as it gets the search results.
   - POST, when we want to change something on the server, for example, when a user makes an account or makes a payment on a website.



**Notes on GET:**

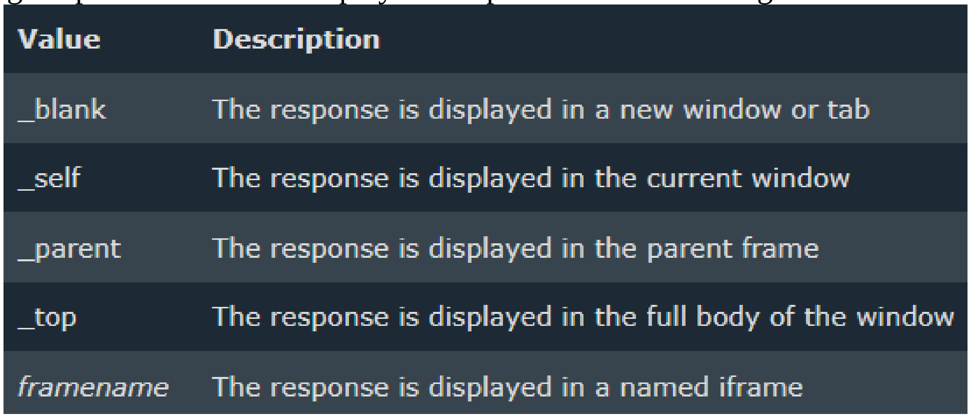
- Appends the form data to the URL, in name/value pairs
- NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
- The length of a URL is limited (2048 characters)
- Useful for form submissions where a user wants to bookmark the result
- GET is good for non-secure data, like query strings in Google

**Notes on POST:**

- Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

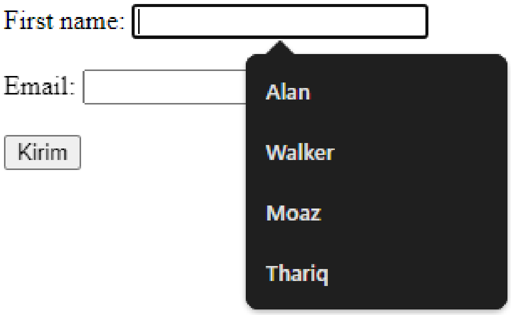
**Tip**: Always use POST if the form data contains sensitive or personal information!

3) target: specifies where to display the response after submitting the form.



| Value | Description |
|---|---|
| _blank | The response is displayed in a new window or tab |
| _self | The response is displayed in the current window |
| _parent | The response is displayed in the parent frame |
| _top | The response is displayed in the full body of the window |
| *framename* | The response is displayed in a named iframe |

4) autocomplete (on/off): When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

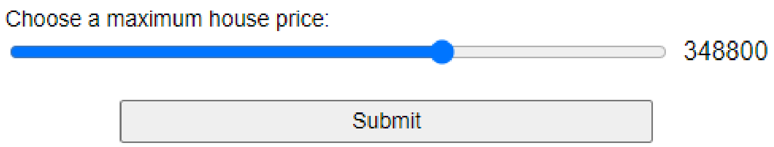


## B. Form controls

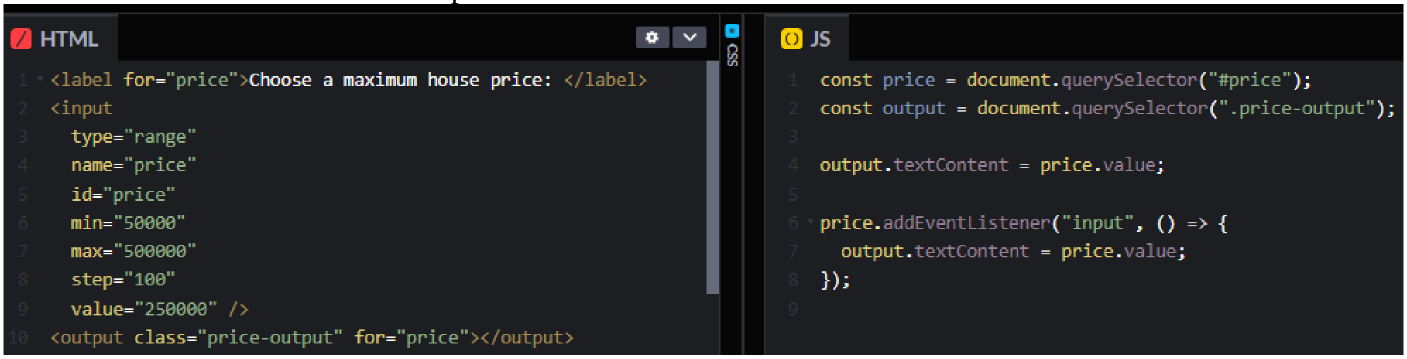
To start collecting user data, we need to use form control elements. These are all the elements users will interact with on the form, such as text boxes, dropdowns, checkboxes and buttons. Most common: input, labels, textarea, select, button, fieldset, legend.

1) Buttons
   - type=”submit” : to submit the form (default)
   - type=”reset” : to reset the form-data that user has intered
   - type=”button” : commonly used with JS for creating interactive UI
2) Organizing form elements
   - <fieldset> : is a container element that allows us to group related form inputs
   - <legend> : give fieldses a heading or caption so the user can see what a grouping of inputs is for.
3) Slider

   Are used to pick a number whose precise value is not necessarily important(e.g on sites like house-buying sites where you want to set a maximum property price to filter by)



Note: slider dont have visual feedback as to what the current value is. This is why we've included an <output> element to contain the current value.

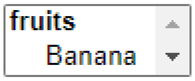


```html
<label for="price">Choose a maximum house price: </label>
<input
  type="range"
  name="price"
  id="price"
  min="50000"
  max="500000"
  step="100"
  value="250000" />
<output class="price-output" for="price"></output>
```

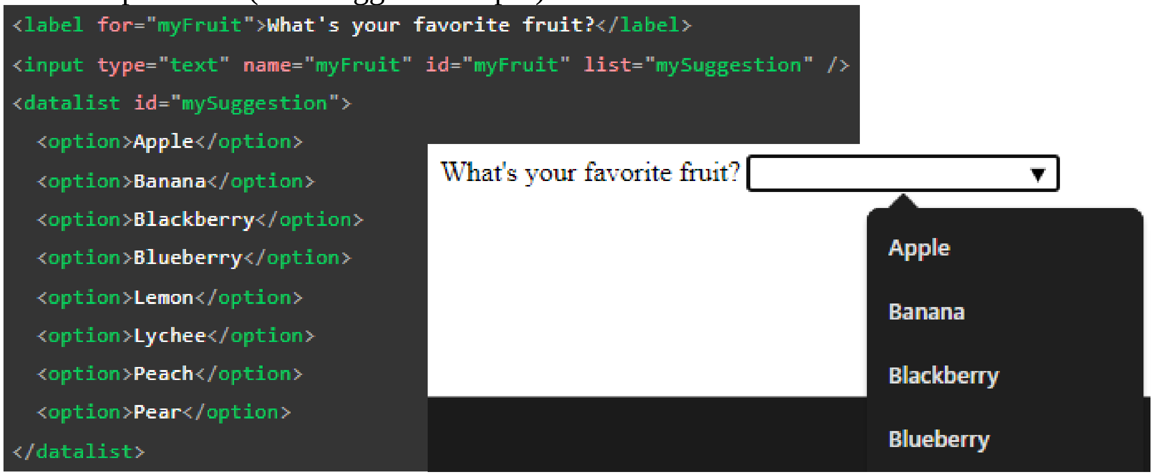
```javascript
const price = document.querySelector("#price");
const output = document.querySelector(".price-output");

output.textContent = price.value;

price.addEventListener("input", () => {
  output.textContent = price.value;
});

```

4) Multiple choice dropdown

   Allow users to select several values <select multiple size="2"> with child <option>, the size attribute define how many choice show in the user display.



5) Autocomplete box (auto-suggestion input)



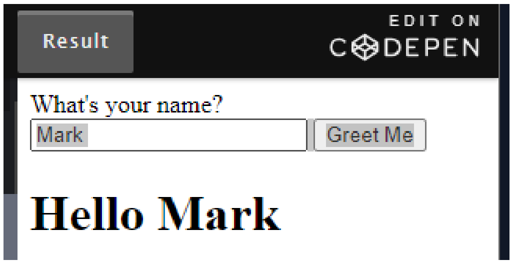
```html
<label for="myFruit">What's your favorite fruit?</label>
<input type="text" name="myFruit" id="myFruit" list="mySuggestion" />
<datalist id="mySuggestion">
  <option>Apple</option>
  <option>Banana</option>
  <option>Blackberry</option>
  <option>Blueberry</option>
  <option>Lemon</option>
  <option>Lychee</option>
  <option>Peach</option>
  <option>Pear</option>
</datalist>
```

Fallback if the browser such as IE below v10 (https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls#datalist_support_and_fallbacks)

6) Range input with tick marks (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/range#adding_tick_marks)
7) <progress> with JS to display the progress of a task.
8) <meter> defines a scalar measurement within a known range, or a fractional value. Example: Disk usage, the relevance of a query result, etc.

## C. Using form controls outside of forms

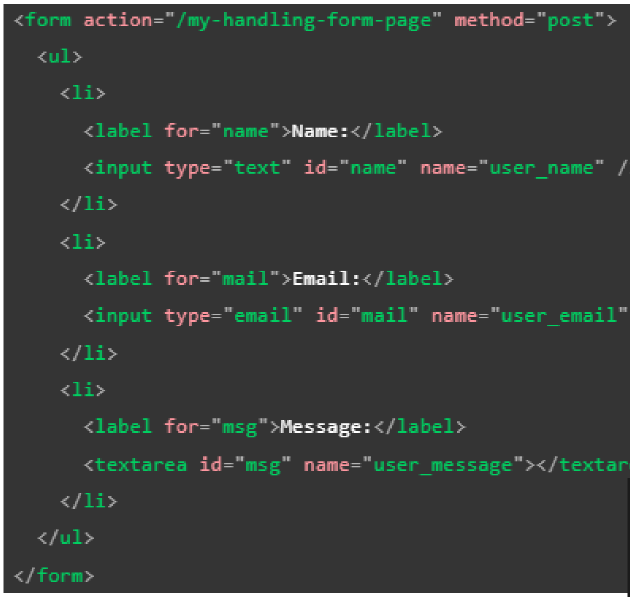
You can use any of the form controls HTML provides outside of the <form> element. For example you want to have an input that gets some data from a user and display that somewhere else on the page with JavaScript:



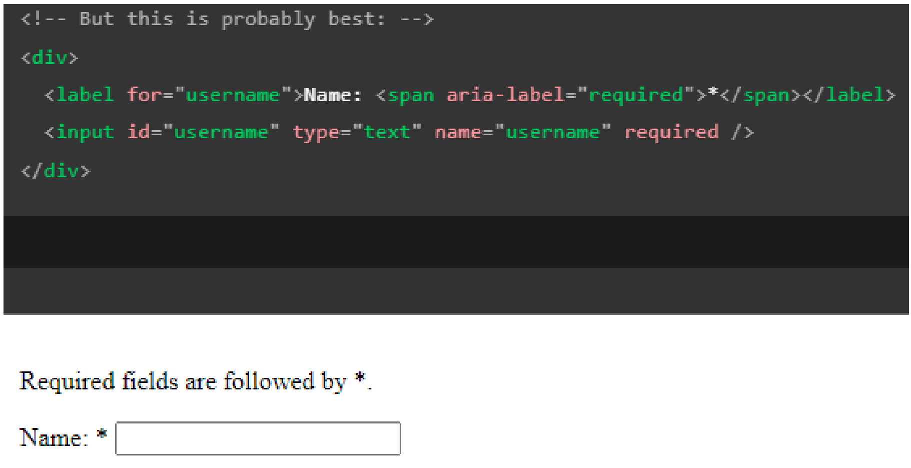
What's your name?
Mark

Hello Mark

## D. Common HTML structure used with forms



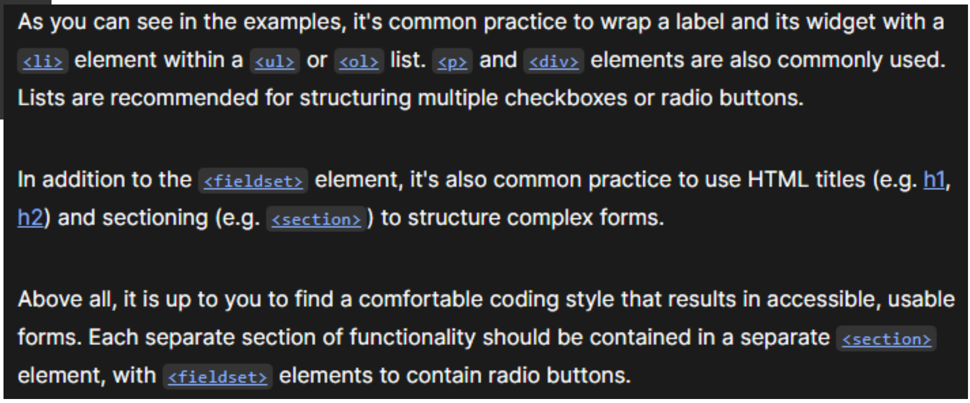
```
<form action="/my-handling-form-page" method="post">
  <ul>
    <li>
      <label for="name">Name:</label>
      <input type="text" id="name" name="user_name" />
    </li>
    <li>
      <label for="mail">Email:</label>
      <input type="email" id="mail" name="user_email" />
    </li>
    <li>
      <label for="msg">Message:</label>
      <textarea id="msg" name="user_message"></textarea>
    </li>
  </ul>
</form>
```



```
<!-- But this is probably best: -->
<div>
  <label for="username">Name: <span aria-label="required">*</span></label>
  <input id="username" type="text" name="username" required />
</div>
```

Required fields are followed by *.

Name: *



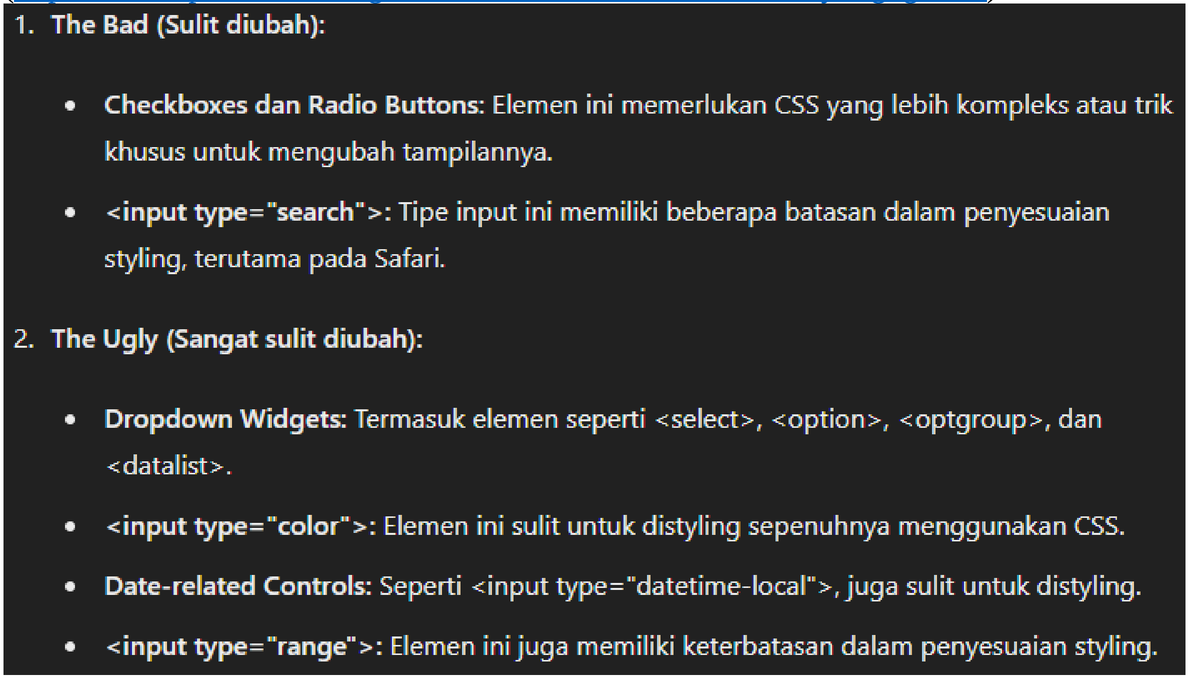
As you can see in the examples, it's common practice to wrap a label and its widget with a <li> element within a <ul> or <ol> list. <p> and <div> elements are also commonly used. Lists are recommended for structuring multiple checkboxes or radio buttons.

In addition to the <fieldset> element, it's also common practice to use HTML titles (e.g. h1, h2) and sectioning (e.g. <section>) to structure complex forms.

Above all, it is up to you to find a comfortable coding style that results in accessible, usable forms. Each separate section of functionality should be contained in a separate <section> element, with <fieldset> elements to contain radio buttons.

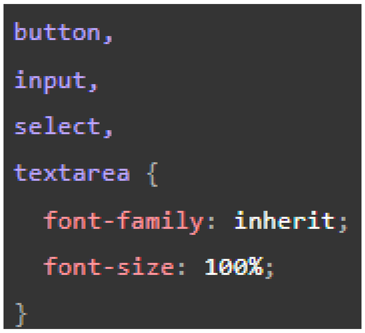
### E. Styling web forms

(https://developer.mozilla.org/enUS/docs/Learn/Forms#form_styling_guides)



1. **The Bad (Sulit diubah):**
    - **Checkboxes dan Radio Buttons:** Elemen ini memerlukan CSS yang lebih kompleks atau trik khusus untuk mengubah tampilannya.
    - **<input type="search">:** Tipe input ini memiliki beberapa batasan dalam penyesuaian styling, terutama pada Safari.
2. **The Ugly (Sangat sulit diubah):**
    - **Dropdown Widgets:** Termasuk elemen seperti <select>, <option>, <optgroup>, dan <datalist>.
    - **<input type="color">:** Elemen ini sulit untuk distyling sepenuhnya menggunakan CSS.
    - **Date-related Controls:** Seperti <input type="datetime-local">, juga sulit untuk distyling.
    - **<input type="range">:** Elemen ini juga memiliki keterbatasan dalam penyesuaian styling.

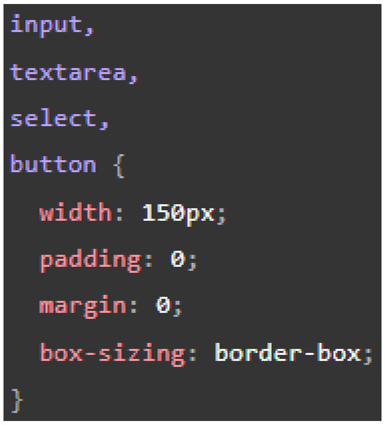
1) Font and text



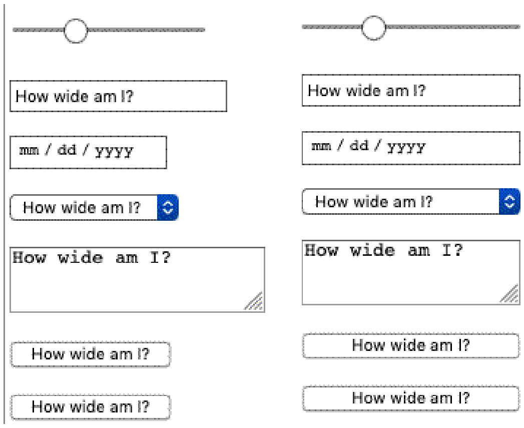
```css
button,
input,
select,
textarea {
  font-family: inherit;
  font-size: 100%;
}
```

Many browsers use the font-family: system-ui; by default. This will make your forms' appearance inconsistent since the font doesn’t same with its parent

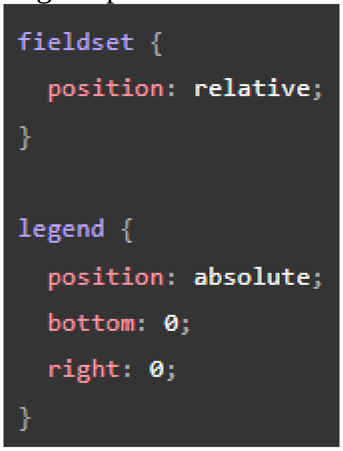
2) Form widget with same width (with box-sizing property)



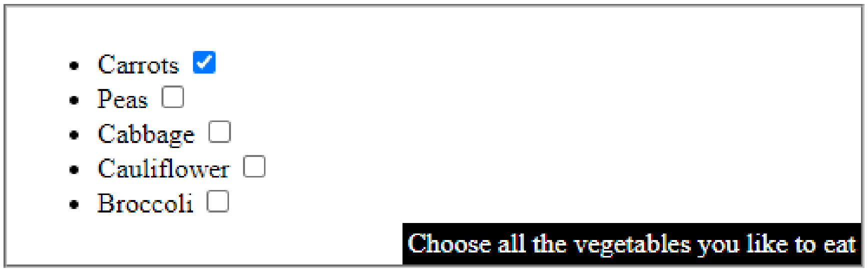
```css
input,
textarea,
select,
button {
  width: 150px;
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
```



3) Legend placement



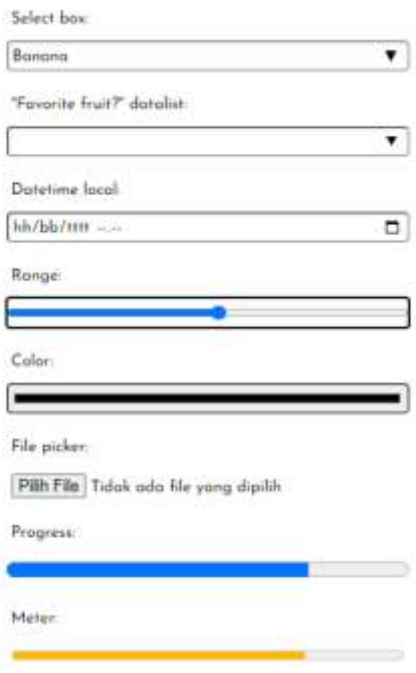
```
fieldset {
  position: relative;
}

legend {
  position: absolute;
  bottom: 0;
  right: 0;
}
```



- Carrots ☑
- Peas □
- Cabbage □
- Cauliflower □
- Broccoli □

Choose all the vegetables you like to eat

4) Avanced form styling
{ appearance: none; } to removes OS-level styling, removes Safari search boxes styling restrictions (height & font-size) & removes checkboxes styling restrictions on browser.



Select box:

Banana ▼

"Favorite fruit?" datalist:

▼

Datetime local:

hh/bb/tttt --:--

Range:

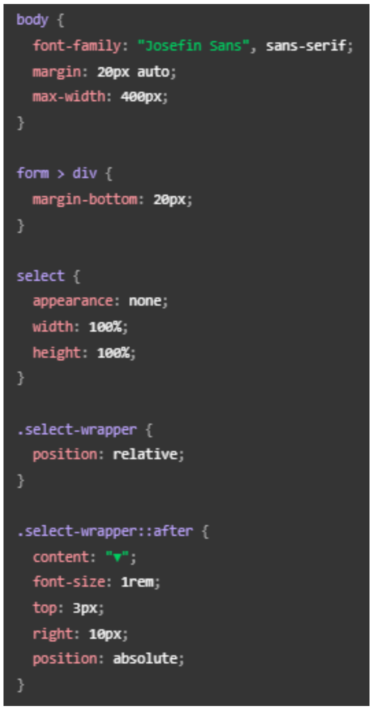
Color:

File picker:

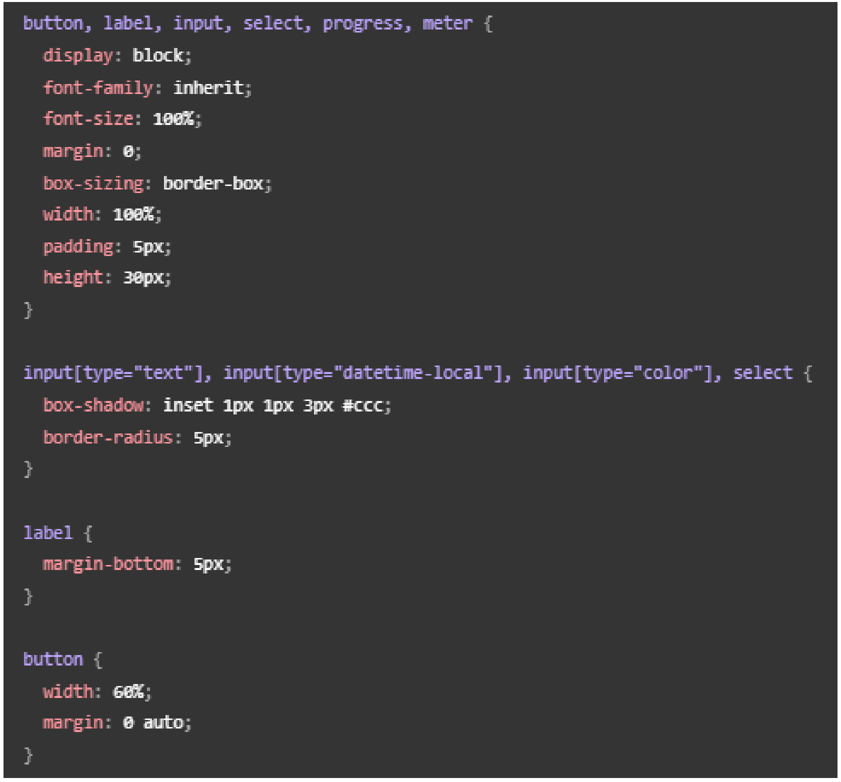
Pilih File Tidak ada file yang dipilih

Progress:

Meter:



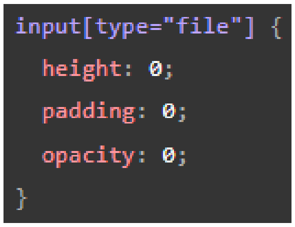
```
body {
  font-family: "Josefin Sans", sans-serif;
  margin: 20px auto;
  max-width: 400px;
}

form > div {
  margin-bottom: 20px;
}

select {
  appearance: none;
  width: 100%;
  height: 100%;
}

.select-wrapper {
  position: relative;
}

.select-wrapper::after {
  content: "▼";
  font-size: 1rem;
  top: 3px;
  right: 10px;
  position: absolute;
}
```



```
button, label, input, select, progress, meter {
  display: block;
  font-family: inherit;
  font-size: 100%;
  margin: 0;
  box-sizing: border-box;
  width: 100%;
  padding: 5px;
  height: 30px;
}

input[type="text"], input[type="datetime-local"], input[type="color"], select {
  box-shadow: inset 1px 1px 3px #ccc;
  border-radius: 5px;
}

label {
  margin-bottom: 5px;
}

button {
  width: 60%;
  margin: 0 auto;
}
```

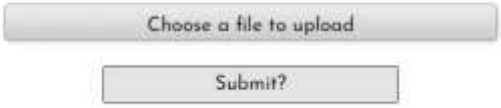
(1) Styling input type="file"

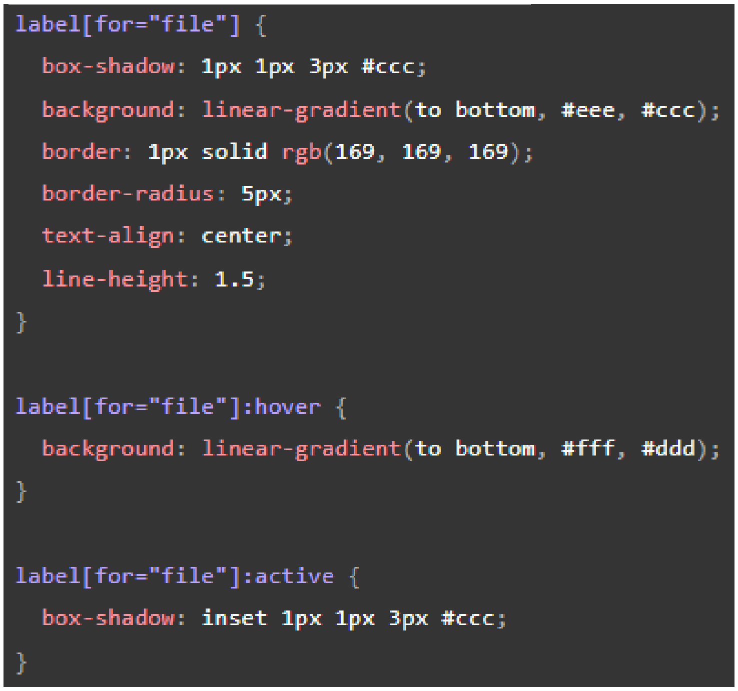
To sized / colorized the color picker you could hide the actual form input using something like this:



```
input[type="file"] {
  height: 0;
  padding: 0;
  opacity: 0;
}
```

clicking the associated label will activate the control And then style the label to act like a button, which when pressed will open the file picker as expected.





```
label[for="file"] {
  box-shadow: 1px 1px 3px #ccc;
  background: linear-gradient(to bottom, #eee, #ccc);
  border: 1px solid rgb(169, 169, 169);
  border-radius: 5px;
  text-align: center;
  line-height: 1.5;
}

label[for="file"]:hover {
  background: linear-gradient(to bottom, #fff, #ddd);
}

label[for="file"]:active {
  box-shadow: inset 1px 1px 3px #ccc;
}
```

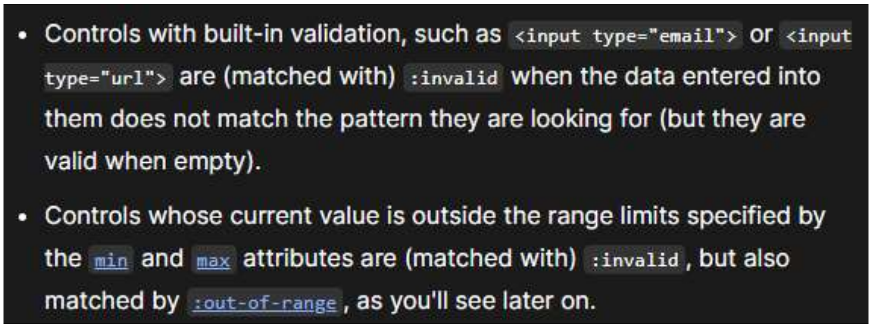
(2) <progress> & <meter>
we can set them to the desired width relatively accurately, you can color the background, but not the foreground bar. Beyond that, It is easier to just create your own custom solution such as progressbar.js(https://kimmobrunfeldt.github.io/progressbar.js/#examples)

5) UI pseudo-classes

(1) Styling inputs based on whether they are required or not
use :required & :optional

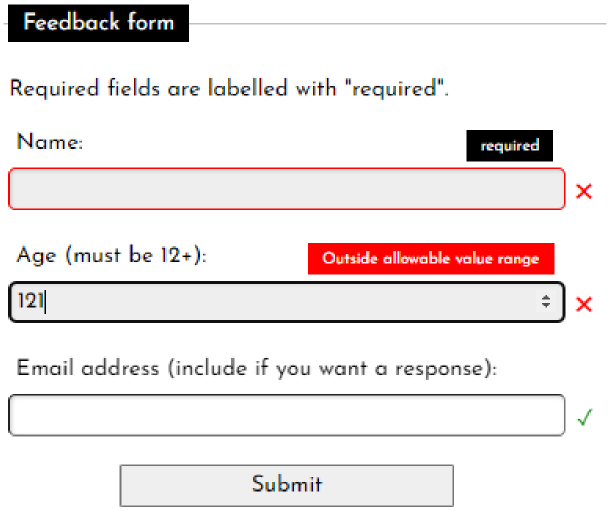
(2) Use generated content & pseud-classes use ::before, ::after, & content=".." (use content at <span> because <input> dont support content)

(3) Based on :valid & :invalid data input



- Controls with built-in validation, such as `<input type="email">` or `<input type="url">` are (matched with) `:invalid` when the data entered into them does not match the pattern they are looking for (but they are valid when empty).
- Controls whose current value is outside the range limits specified by the `min` and `max` attributes are (matched with) `:invalid`, but also matched by `:out-of-range`, as you'll see later on.

(4) based on :in-range & :out-of-range
These match numeric inputs where range limits are specified by the min and max attribute. Example:
Example project: https://mdn.github.io/learning-area/html/forms/pseudo-classes/out-of-range.html
source code: https://github.com/mdn/learning-area/blob/main/html/forms/pseudo-classes/out-of-range.html



Feedback form

Required fields are labelled with "required".

Name: required ✗

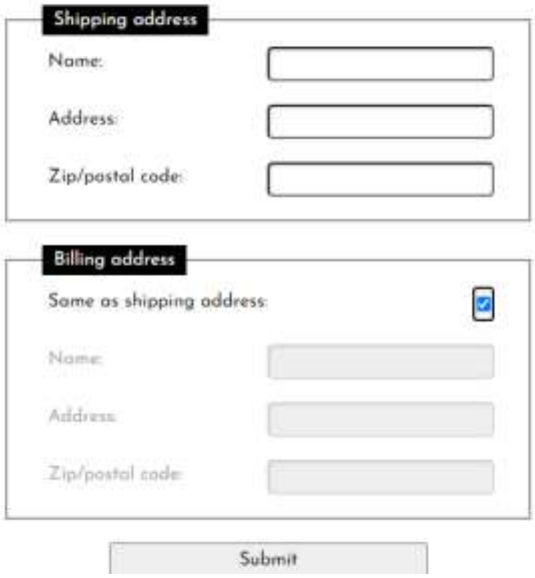
Age (must be 12+): Outside allowable value range

121 ✗

Email address (include if you want a response): ✓

Submit

(5) based on :enabled & :disabled



Shipping address

Name:

Address:

Zip/postal code:

Billing address

Same as shipping address: ☑

Name:

Address:

Zip/postal code:

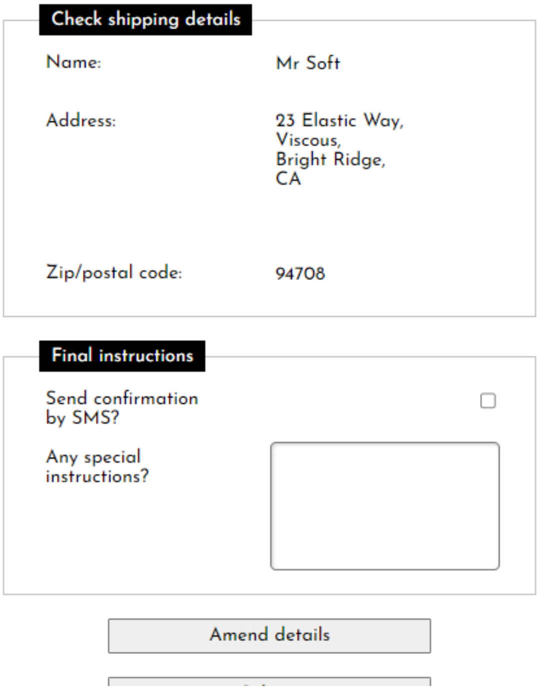
Submit

example project: [https://mdn.github.io/learning-area/html/forms/pseudo-classes/enabled-disabled-shipping.html](https://mdn.github.io/learning-area/html/forms/pseudo-classes/enabled-disabled-shipping.html)

source code: [https://github.com/mdn/learning-area/blob/main/html/forms/pseudo-classes/enabled-disabled-shipping.html](https://github.com/mdn/learning-area/blob/main/html/forms/pseudo-classes/enabled-disabled-shipping.html)

(6) based on :read-only & :read-write

Read-only inputs have their values submitted to the server, but the user can't edit them, whereas read-write means they can be edited. As an (readonly) example, imagine a confirmation page where the developer has sent the details filled in on previous pages over to this page.



Check shipping details

Name: Mr Soft

Address: 23 Elastic Way,
Viscous,
Bright Ridge,
CA

Zip/postal code: 94708

Final instructions

Send confirmation by SMS? □

Any special instructions?

Project:
https://mdn.github.io/learning-area/html/forms/pseudo-classes/readonly-confirmation.html

Source:
https://github.com/mdn/learning-area/blob/main/html/forms/pseudo-classes/readonly-confirmation.html

(7) Radio and checkbox states — checked, default, indeterminate
- :default Matches radios/checkboxes that are checked by default, on page load. These match the :default pseudo-class, even if the user unchecks them.
- :indeterminate when
(1) <input/radio> buttons in a same-named group are unchecked
(2) <input/checkboxes> whose indeterminate property is set to true via JavaScript
(3) <progress> elements that have no value.

(8) More interesting pseudo-classes
- :focus-within
- :focus-visible
- :placeholder-shown

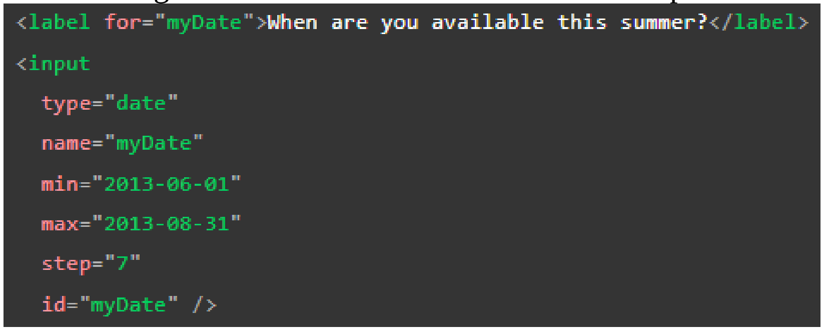
## F. Tricky to style form controls

- Text-based form controls like text, email, password and text areas are easy to style.
- Radio buttons & checkboxes are tricky to style, but there's a guide for checkboxes(https://moderncss.dev/pure-css-custom-checkbox-style/) & any input type(https://www.w3schools.com/cssref/css4_pr_accent-color.php)
- Calendar/date pickers are impossible to style, we will have to build custom form controls with JS or use JS library.

## G. Note

- Dalam ‘ <input type=”radio” /> ‘, semua input bertype radio harus memiliki 1 value name=”..’ yg sama, agar menjadi multiple choice(pilihan ganda yang hanya bisa memilih 1 jawaban dari banyak opsi).
- <label for=”INI HARUS SAMA” > dengan <input id=”INI HARUS SAMA”/> supaya terhubung.
- Use placeholder text to demonstrate how text should be entered and formatted. Example: <input type="text" id="first_name" placeholder="Bob...">
- We need to use name=”value” attribute so that backend understand what the data entered into an input field will represent. SEMUA <Input> HARUS ADA name=”..”
- We use fieldset & legend to organizing form elements.

- Dropdowns are great for list of options. However, when we have a smaller list of 5 or fewer options to choose from, it is often a better UX to use radio buttons.
- <form action="https://httpbin.org/post" method=”post”> to see the backend result
- <input type=”file” accept=”.jpg, .png” multiple> to allow user pick multiple (jpg and png only) files at once { check accept attribute in w3school }
- <ul> <li> & <p> merupakan structural elements yang digunakan untuk mem-wrap label dengan inputnya
  1) radio button di-wrap dengan <ul> <li>
  2) input & dropdown di-wrap dengan <p>
- In input type=”email”, a@b is valid email address so we need to use pattern=”regex” attribute
- In input type=”tel”, alphabet letters is valid so we need to use pattern=”regex” attribute
- Input type=”url” will valid if the input contains ‘ http: ‘
- step=”value” attribute in input type=”number : to define increment value
- ‘number’ input type makes sense when the range of valid values is limited, for example a person's age or height. If the range is too large for incremental increases to make sense (such as USA ZIP codes, which range from 00001 to 99999), the tel type might be a better option.
- Constraining date/time values with min max and step



```
<label for="myDate">When are you available this summer?</label>
<input
  type="date"
  name="myDate"
  min="2013-06-01"
  max="2013-08-31"
  step="7"
  id="myDate" />
```

- Jangan kebalik dalam menggunakan <progress> & <meter>
- <option> jka tidak memiliki value=”..” maka text dalam tag nya yang akan menjadi value(value attribute sebenarnya hanya untuk menyingkat text yg ingin dikirim ke server)

H. Supplement

1. Form UX
   - https://www.smashingmagazine.com/2018/08/ux-html5-mobile-form-part-1/
   - https://www.smashingmagazine.com/2011/11/extensive-guide-web-form-usability/