



Forms

Enabling User Interaction

Luigi De Russis

Billing address

First name

Last name

Username

Email (Optional)

Address

Address 2 (Optional)

Country

State

Zip

Shipping address is the same as my billing address

Save this information for next time

Payment

Credit card

Debit card

Paypal



Topics

- Forms for user interaction
 - HTML5 tags for input
 - Validation
 - Handling form in Flask

Handling User Input

HTML5 FORMS

Form Declaration

- `<form>` tag
- Specifies URL to be used for submission (attribute `action`)
- Specifies HTTP method (attribute `method`, default GET)

```
...  
<form action="/new-user" method="POST" id="userdata">  
    ...normal HTML content...  
        and  
    ...FORM Controls...  
</form>  
...
```

Form Controls

- A set of HTML elements allowing different types of user input/interaction. Each element should be uniquely identified by the value of the name attribute
- Several control categories
 - Input
 - Selection
 - Button
- Support elements
 - Label
 - Datalist

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Forms>

Input Control

- `<input>` tag
- Text input example
- The `value` attribute will hold user-provided text

```
...  
<input type="text" name="firstname" placeholder="Your username"></input>  
...
```

Input Control (2)


- type attribute
 - button
 - checkbox
 - color
 - date
 - email
 - file
 - hidden
 - month
 - number
 - password

Type	Description	Basic Examples	Spec
button	A push button with no default behavior displaying the value of the <code>value</code> attribute, empty by default.	<input type="button"/>	
checkbox	A check box allowing single values to be selected/deselected.	<input type="checkbox"/>	
color	A control for specifying a color; opening a color picker when active in supporting browsers.	<input type="color"/>	HTML5
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.	<input type="date"/>	HTML5
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.	<input type="datetime-local"/>	HTML5
email	A field for editing an email address. Looks like a <code>text</code> input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.	<input type="email"/>	HTML5
file	A control that lets the user select a file. Use the <code>accept</code> attribute to define the types of files that the control can select.	<input type="file"/>	
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!	<input type="hidden"/>	
image	A graphical <code>submit</code> button. Displays an image defined by the <code>src</code> attribute. The <code>alt</code> attribute displays if the image <code>src</code> is missing.	<input type="image"/>	
month	A control for entering a month and year, with no time zone.	<input type="month"/>	HTML5
number	A control for entering a number. Displays a spinner and adds default validation when supported. Displays a numeric keypad in some devices with dynamic keypads.	<input type="number"/>	HTML5
password	A single-line text field whose value is obscured. Will alert user if site is not secure.	<input type="password"/>	

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>

Input Control (3)

- type attribute
 - radio (button)
 - range
 - submit/reset (button)
 - search
 - tel
 - text
 - url
 - week

radio	A radio button, allowing a single value to be selected out of multiple choices with the same <code>name</code> value.	<input type="radio"/>	
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 <code>htmlattrdefmin</code> and <code>htmlattrdefmax</code> to define the range of acceptable values.	<input type="range"/>	HTML5
reset	A button that resets the contents of the form to default values. Not recommended.	<input type="reset" value="Reset"/>	
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.	<input type="search"/>	HTML5
submit	A button that submits the form.	<input type="submit" value="Submit"/>	
tel	A control for entering a telephone number. Displays a telephone keypad in some devices with dynamic keypads.	<input type="tel"/>	HTML5
text	The default value. A single-line text field. Line-breaks are automatically removed from the input value.	<input type="text"/>	
time	A control for entering a time value with no time zone.	<input type="time" value="--:--"/>	HTML5
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.	<input type="url"/>	HTML5
week	A control for entering a date consisting of a week-year number and a week number with no time zone.	<input type="week" value="Week --, ----"/>	HTML5
Obsolete values			
datetime	 A control for entering a date and time (hour, minute, second, and fraction of a second) based on UTC time zone.	<input type="datetime"/>	HTML5

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>

Input Control: Commonly Used Attributes

Attribute	Meaning
checked	radio/checkbox is selected
disabled	control is disabled
readonly	value cannot be edited
required	need a valid input to allow form submission
size	the size of the control (pixels or characters)
value	the value inserted by the user
autocomplete	hint for form autofill feature of the browser

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes>

Input Control: Other Attributes

- Depends on the control

```
<input type="number" name="age" placeholder="Your age" min="18" max="110" />
```

```
<input type="text" name="username" pattern="[a-zA-Z]{8}" />
```

```
<input type="file" name="docs" accept=".jpg, .jpeg, .png" />
```

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes>

Label Tag

- The HTML `<label>` element represents a caption for an item in a user interface. Associated with `for` attribute and `id` on input
- Important for accessibility purposes (e.g. screenreader etc.), clicking the label activates the control (larger activation area e.g. in touch screens)

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

Do you like cheese?

Do you like peas?



Click!

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label>

Other Form Controls

`<textarea>`:
a multi-line text field

Default

Focus

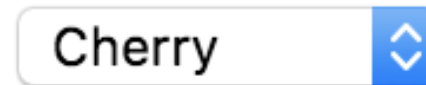
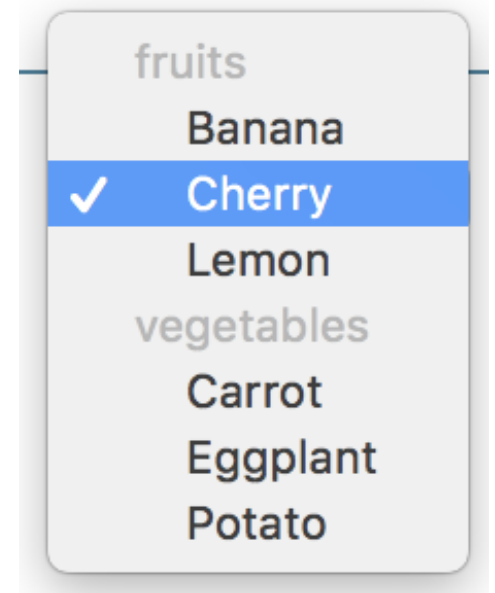
Disabled

[https://developer.mozilla.org/en-US/docs/Learn/Forms/Other form controls](https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls)

Other Form Controls

Drop-down controls

```
<select id="groups" name="groups">
  <optgroup label="fruits">
    <option>Banana</option>
    <option selected>Cherry</option>
    <option>Lemon</option>
  </optgroup>
  <optgroup label="vegetables">
    <option>Carrot</option>
    <option>Eggplant</option>
    <option>Potato</option>
  </optgroup>
</select>
```

A white rounded rectangular button with the text 'Cherry' in black. To the right of the text is a blue square containing two white downward-pointing chevrons, indicating a drop-down menu.A white rounded rectangular menu with a light gray background. The menu is open, showing a list of items. The items are grouped into two sections: 'fruits' and 'vegetables'. The 'fruits' section contains 'Banana', 'Cherry', and 'Lemon'. The 'vegetables' section contains 'Carrot', 'Eggplant', and 'Potato'. The 'Cherry' item is highlighted with a blue background and a white checkmark to its left.

https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls

Button Control

- `<button>` tag
- Three types of buttons
 - `submit`: submits the form to the server
 - `reset`: reset the content of the form to the initial value
 - `button`: just a button, whose behavior needs to be specified by JavaScript

```
...  
<button type="submit" value="Send data" />  
...
```

button vs. input type=button

More flexible, can have content (markup, images, etc.)

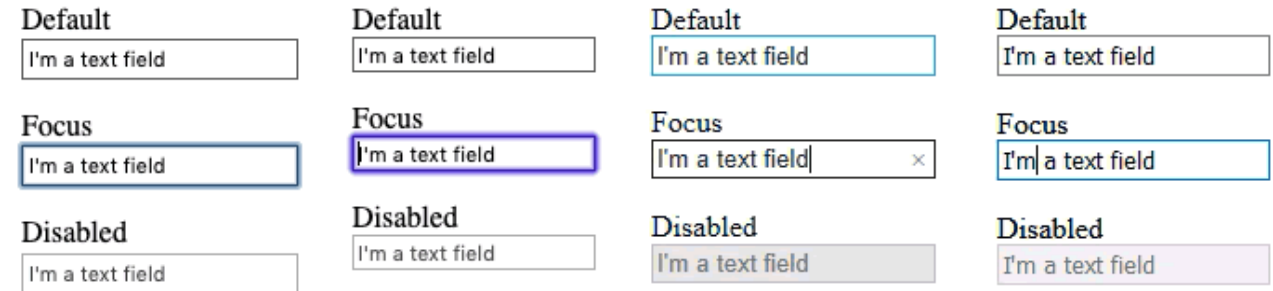
```
...  
<button class="favorite styled"  
        type="button">  
    Add to favorites  
</button>  
...  
<button name="favorite">  
    <svg aria-hidden="true" viewBox="0 0 10 10"><path  
d="M7 9L5 8 3 9V6L1 4h3l1-3 1 3h3L7 6z"/></svg>  
    Add to favorites  
</button>  
...
```



<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button>

Default Appearance May Vary

- Solve with CSS, but
- Some problems still remain
 - See: "Styling web forms" in MDN
 - Examples of controls difficult to manage:
 - Bad: Checkboxes, ...
 - Ugly: Color, Range, File: cannot be styled via CSS



https://developer.mozilla.org/en-US/docs/Learn/Forms/Styling_web_forms

The Road to Nicer Forms

- Useful libraries (frameworks)
 - Especially for controls difficult to handle via CSS
- Suggestions
 - Bootstrap
 - Using libraries may improve accessibility

https://developer.mozilla.org/en-US/docs/Learn/Forms/Advanced_form_styling

Forms in Bootstrap

The screenshot shows the Bootstrap 5.2 documentation page for Forms. The page title is "Forms" and the subtitle is "Examples and usage guidelines for form control styles, layout options, and custom components for creating a wide variety of forms." The page is divided into several sections: "Form control", "Checks & radios", "Input group", "Layout", "Select", "Range", "Floating labels", and "Validation". Each section has a brief description of the component. The page also includes a navigation menu on the left, a search bar at the top, and a "View on GitHub" button.

Forms
Examples and usage guidelines for form control styles, layout options, and custom components for creating a wide variety of forms.

Form control
Style textual inputs and textareas with support for multiple states.

Checks & radios
Use our custom radio buttons and checkboxes in forms for selecting input options.

Input group
Attach labels and buttons to your inputs for increased semantic value.

Layout
Create inline, horizontal, or complex grid-based layouts with your forms.

Select
Improve browser default select elements with a custom initial appearance.

Range
Replace browser default range inputs with our custom version.

Floating labels
Create beautifully simple form labels that float over your input fields.

Validation
Validate your forms with custom or native validation behaviors and styles.

Overview
Bootstrap's form controls expand on [our Rebooted form styles](#) with classes. Use these classes to opt into their customized displays for a more consistent rendering across browsers and devices.

Be sure to use an appropriate `type` attribute on all inputs (e.g., `email` for email address or `number` for numerical information) to take advantage of newer input controls like email verification, number selection, and more.

Here's a quick example to demonstrate Bootstrap's form styles. Keep reading for documentation on required classes, form layout, and more.

<https://getbootstrap.com/docs/5.3/forms/overview/>

Form Validation

- When entering data into a form, the browser will check to see if the data is in the correct format and with the constraints set by the application
 - Client-side validation: via HTML5 and JavaScript
 - Server-side validation: the application server will take care of it
- After client-side validation, data can be submitted to the server
- Why client-side validation?
 - We want to get the right data in the right format before processing the data
 - We want to protect users' data (e.g., enforcing secure passwords)
 - We want to protect the application (however, **NEVER TRUST** client-side validation on server side)

Types Of Client-Side Validation

- Built-in form validation by HTML5 input elements. Examples:
 - Email: check if the inserted value is a valid email (syntax only)
 - URL: check if it is a valid URL
 - Number: check if the text is a number
 - Attribute required: if a value is not present, form cannot be submitted
 - ...
- JavaScript validation: custom code is used to check correctness of values
 - More on this later in the course

Built-In Form Validation

- Mainly relies on element attributes such as:
 - **required**: if a value is not present, form cannot be submitted
 - **minlength** **maxlength** for text
 - **min** **max** for numerical values
 - **type**: type of data (email, url, etc.)
 - **pattern**: regular expression to be matched
- When element is valid, the `:valid` CSS pseudo-class applies, which can be used to style valid elements, otherwise `:invalid` applies

Built-In Form Validation Styling

```
...  
<form>  
  <label for="e_addr">Email Address:<label>  
  <input type="email" id="e_addr" id="email" required  
placeholder="Enter a valid email address">  
</form>  
...
```

```
input:invalid {  
  border: 2px dashed red;  
}  
  
input:valid {  
  border: 2px solid black;  
}
```

Email Address:

Email Address:

Email Address:

When the client interacts with the server

HANDLING FORMS IN FLASK

Forms Data In Flask

- The entire content of a submitted form is sent with an HTTP request (POST or PUT) to the application server (e.g., Flask)
- Flask packs all form's variables in a 'request.form' object
 - A dictionary
- 'request' is a global implicit object that must be imported

```
from flask import request  
  
age = request.form['age'] # or .get(age) - safer
```


Forms Data In Flask

- If the key does not exist in the form attribute, a `KeyError` is raised
- If you do not catch it, a HTTP 400 Bad Request error page is shown
 - For many situations, this is a good behavior

- For URL query parameters, instead, use `request.args`

```
age = request.args.get('age')
```

Server-Side Form Validation

- Fundamental
 - To use and store the “correct” values
- You can do it manually, e.g.,

```
if age and isinstance(age, int):  
    if age > 0:  
        ...
```

- You can also use a Flask extension such as *WTForm*

Logging

- Sometimes you want to log what is going on in the server
 - And notify any errors!
- Flask provides pre-configured logging facilities, ready to use

```
app.logger.debug('A value for debugging')  
app.logger.warning('A warning occurred (%d apples)', 42)  
app.logger.error('An error occurred')
```

Passing Values To a Template

- Pass it with `render_template()`
 - as you did for any other variables
 - form values are independent from template parameters

```
return render_template('welcome.html', name=user_from_form)
```

```
<p>Welcome {{ name }}!</p>
```

File Uploads

- Forms for uploading files needs the `enctype="multipart/form-data"` attribute in the HTML document
 - Sent with POST, PUT, or PATCH
 - Otherwise, the browser will not transfer the files
- In Flask, you can access and save the uploaded files via the request object

```
uploaded_file = request.files['file']  
uploaded_file.save('uploads/uploaded_file.txt')
```

File Uploads

- The original filename (with extension) is available in the `filename` attribute
 - Before using it to save a file on disk, it must be checked and sanitized with `secure_filename()`
 - Again, never trust the information coming from a client!

```
uploaded_file = request.files['file']
filename = secure_filename(uploaded_file.filename)
uploaded_file.save(f'uploads/{filename}')
```

Remembering Values

Problem: values in `request.form` expire immediately

- We may want to “remember” values for a longer time

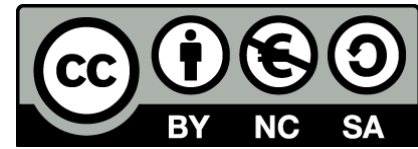
Solutions:

1. Storing them in *session* containers

- Based on HTTP cookies
- Kept in memory (often) in the web server
- Valid for limited time, e.g., until browser disconnection or timeout

2. Storing them in a connected *database*

- Persistent storage
- Kept on disk in the database server
- Requires explicit DB connection



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

