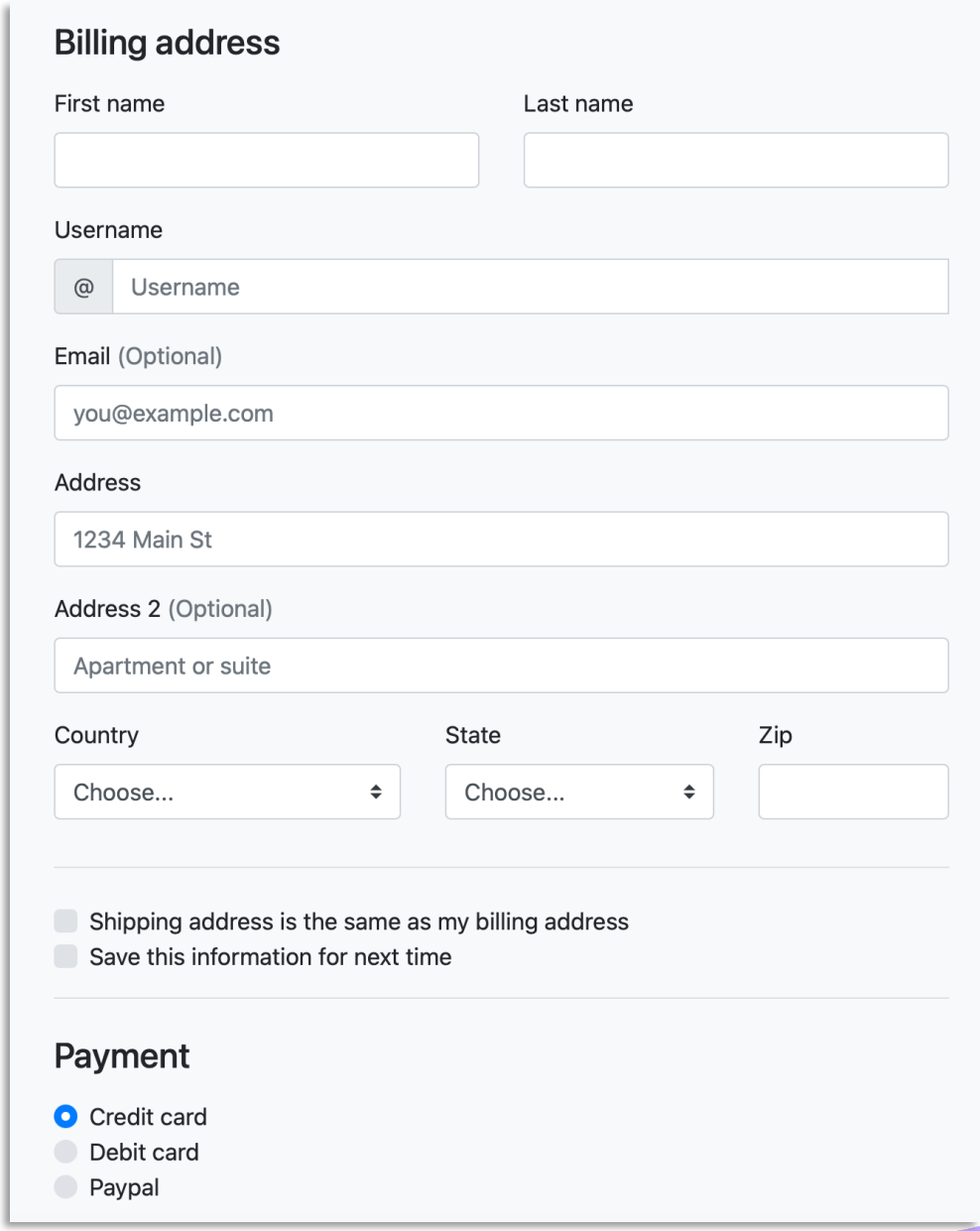
Intro
<AppWeb/>
2022

Forms and Sessions

Enabling User Interaction

Luigi De Russis



Politecnico di Torino

Topics

- Forms for user interaction
  - HTML5 tags for input
  - Validation
  - Handling form in Flask
- Sessions
  - One way to “remember” information

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>
...
```

Your firstname

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 value attribute, empty by default.		
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.		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.	dd/mm/yyyy	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.	dd/mm/yyyy, --:--	HTML5
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.		HTML5
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.	Choose file No file chosen	
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.	--------- ----	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.		HTML5
password	A single-line text field whose value is obscured. Will alert user if site is not secure.		

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 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 htmlattrdefmin and htmlattrdefmax to define the range of acceptable values.		HTML5
reset	A button that resets the contents of the form to default values. Not recommended.	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.		HTML5
submit	A button that submits the form.	Submit	
tel	A control for entering a telephone number. Displays a telephone keypad in some devices with dynamic keypads.		HTML5
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.	--:--	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.		HTML5
week	A control for entering a date consisting of a week-year number and a week number with no time zone.	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.		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

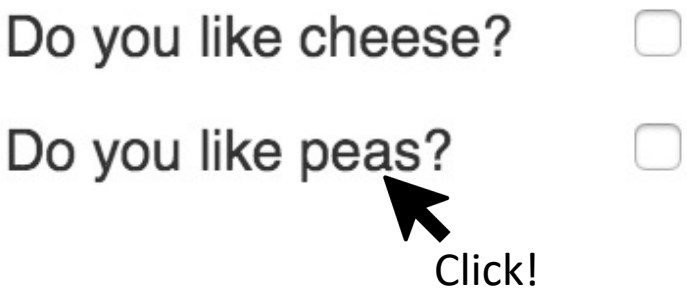
```
<input type="number" name="age" placeholder="Your age" min="18" max="110" />

<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)

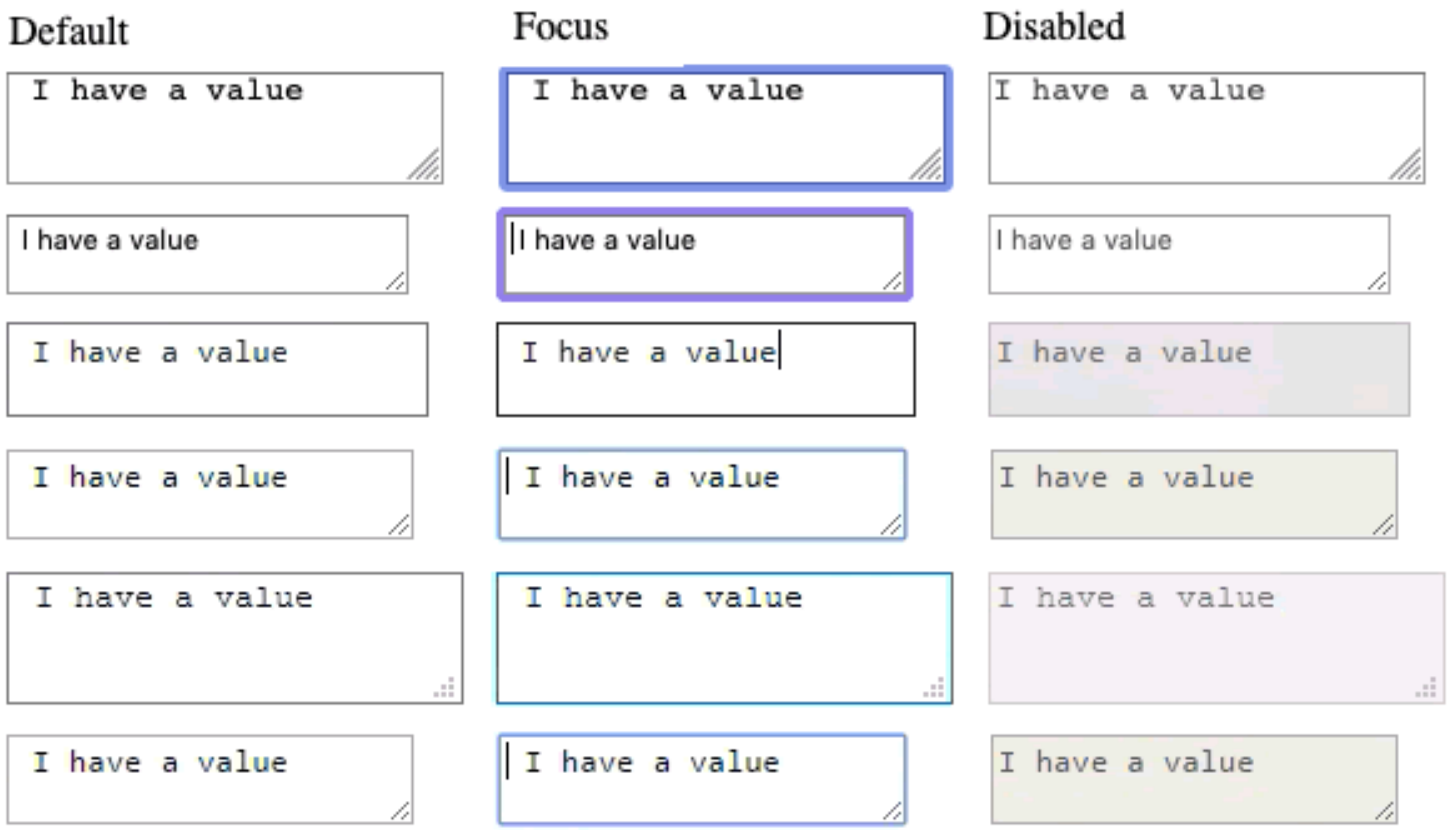
```
<div class="preference">
    <label for="cheese">Do you like cheese?</label>
    <input type="checkbox" name="cheese" id="cheese">
</div>
<div class="preference">
    <label for="peas">Do you like peas?</label>
    <input type="checkbox" name="peas" id="peas">
</div>
```



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

Other Form Controls

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

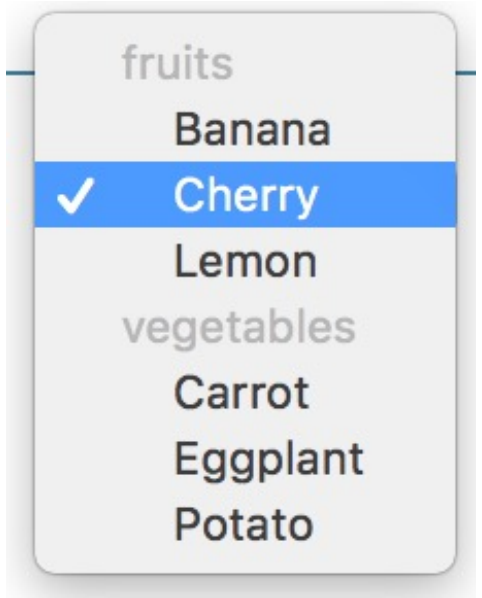


https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls

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>
```



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

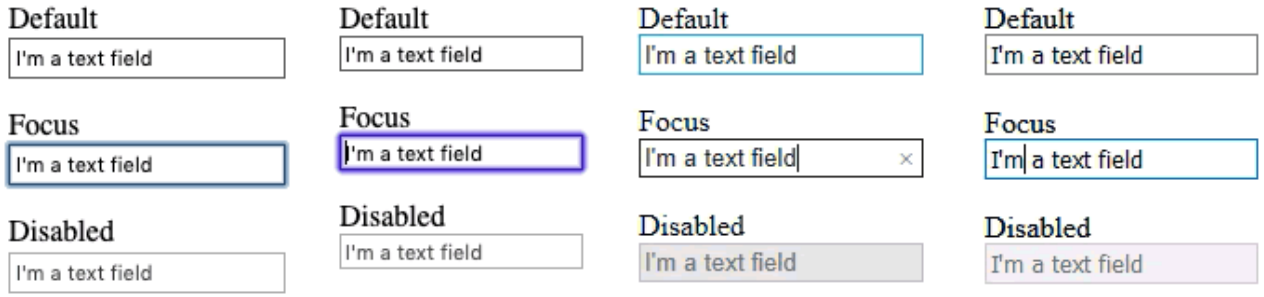
More flexible, can have content (markup, images, etc.)

```
...
<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



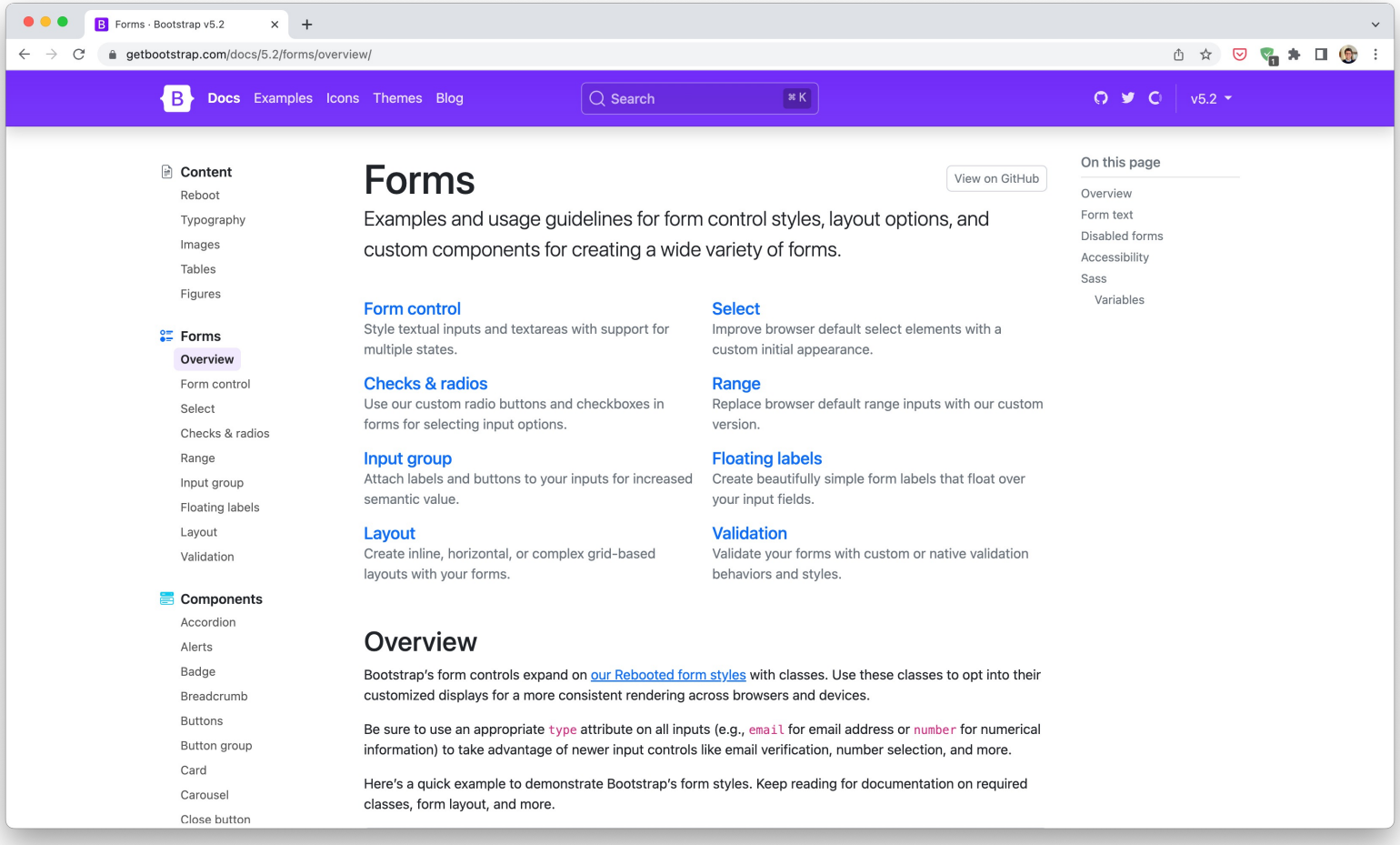
https://developer.mozilla.org/en-US/docs/Learn/Forms/Styling_web_forms

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



https://getbootstrap.com/docs/5.2/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: Enter a valid email addre

Email Address: a@

Email Address: a@p.it

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

Remembering information

SESSIONS

Sessions

- **HTTP is stateless**
  - each request is independent and must be self-contained
- A web application may need to keep some information between different interactions
- For example:
  - in an on-line shop, we put a book in a shopping cart
  - we do not want our book to disappear when we go to another page to buy something else!
  - we want our “state” to be remembered while we navigate through the website

Sessions

- A **session** is temporary and interactive data interchanged between two or more parties (e.g., devices)
- It involves one or more messages in each direction
- Often, one of the parties keeps the state of the application
- It is established at a certain point it time and ended at some later point

Cookie

https://en.wikipedia.org/wiki/HTTP_cookie
https://tools.ietf.org/html/rfc6265

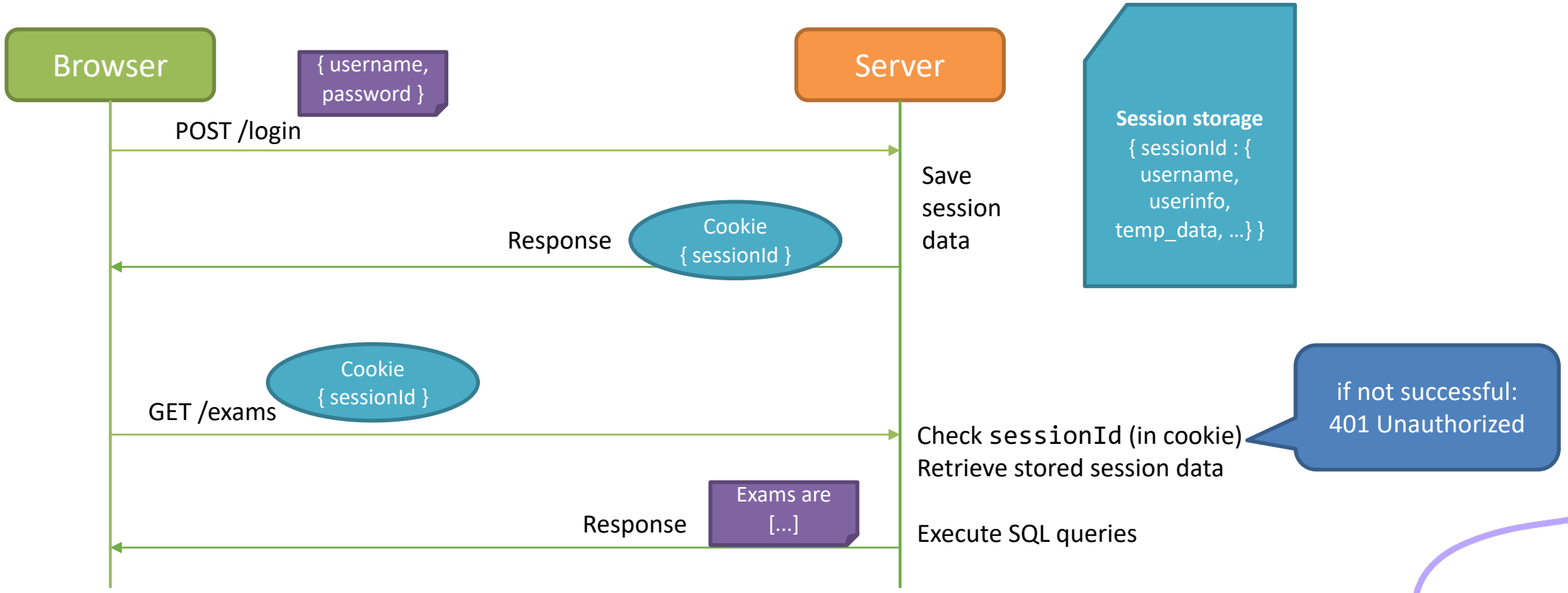
- A small portion of information stored in the browser (in its cookie storage)
- Automatically handled by browsers
- Automatically sent by the browser to servers when performing a request to the same **domain** and **path**
  - options are available to send them in other cases
- Keep in mind that sensitive information should **NEVER** be stored in a cookie!

Cookie

- Some relevant attributes, typically set by the server:
  - **name**, the name of the cookie [mandatory]
    - Example: `ID`
  - **value**, the value contained in the cookie [mandatory]
    - Example: `94$KKDEC3343KCQ1!`
  - **secure**, *if set*, the cookie will be sent to the server over HTTPS, only
  - **httpOnly,** *if set*, the cookie will be inaccessible to JavaScript code running in the browser
  - **expiration date**

Example: Sessions for User Authentication

- The user state is stored on the server
  - in a storage or, for development only, in memory



A Note About Security…

- **Always** use HTTPS and “secure” cookies (at least in production)
  - use “httpOnly” cookies
- **Never** store sensitive information into cookies
  - even if they are “httpOnly”
- Rely on **best practices** and avoid to *re-invent the wheel* for auth
- Web applications can be exposed to several “basic” attacks
  - *CSRF* (Cross-Site Request Forgery), a user is tricked by an attacker into submitting a request that they did not intend
  - *XSS* (Cross-Site Scripting), attackers inject malicious JavaScript code into web pages
  - Most of these can be prevented with a proper usage of frameworks, best practices, and dedicated libraries

Sessions in Flask

- Sessions are automatically initialized and managed by Flask as **client-side sessions**
- Session data is encrypted. You must define a *secret key*
  - app.secret_key = '*whoknowsthissecret*'
  - the user could look at the contents of the cookie but not modify it, unless they know the secret key
- The 'session' object is a global shared dictionary that stores attribute-value pairs in a cookie

```
session['user'] = name
```

```
<p>Welcome {{ session['user'] }}!</p>
```

Client-side vs. Server-side Sessions

Client-side Sessions

- All the data is in a cookie, in the user's browser
  - The cookie can become very big
  - Clients can read all the pieces of information (secrets?)
- The server is entirely stateless
  - It does not need to store any data
  - The server cannot revoke a session

Server-side Sessions

- All the data is on the server
  - Cookies are typically used to store and pass around a SessionID
  - Clients can only read the SessionID
- The server is stateful
  - You can store more data than in a cookie
  - Scalability is more challenging

Server-side Sessions in Flask

- Use the Flask-Sessions extension
  - https://flask-session.readthedocs.io/en/latest/
  - `pip install Flask-Session`
- Implements best practices for cookies (e.g., httponly)
  - Allow the developer to change the other properties
- Support six different session storages (default: *null*)
  - null, redis, memcached, filesystem, mongodb, db (SQLAlchemy)

Automatic Redirects

- In some cases, a user action does not need to generate a response page
  - E.g., the Logout action needs to destroy the session, but will just bring you to the normal ‘index’ page
- You may use a ‘redirect’ method to instruct the browser that the current response is empty, and it must load the new page (HTTP 302)

```
return redirect(url_for('index'))
```

Example – app.py

```
from flask import Flask, url_for, render_template, redirect, request, session
from flask_session import Session

app = Flask(__name__)
app.config['SESSION_TYPE'] = 'filesystem'
Session(app)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/new-user', methods=['POST', 'GET'])
def new-user():
    if request.method == "POST":
        session['name'] = request.form.get('name')
        return redirect('/')
    return render_template('new-user.html')
```

Example – index.html

```
{% extends "base.html" %}

{% block content %}

   {% if session.name %}
      <p>Welcome, {{ session.name }}!</p>
   {% else %}
      <p>Welcome, John Doe!</p>
   {% endif %}

{% endblock %}
```

License