Part 1: Tables

Data representation in HTML

Tables are for data, not layout

- Use tables when information is naturally tabular (schedule, list, price table).
- Do NOT use tables to position layout (modern layout uses CSS: flex/grid).
- A good rule: if you can describe it as “data”, a table is acceptable.

When to Use Tables

(and when NOT to)

Tables

Use tables for:

- Schedules
- Grade lists
- Product comparisons
- Any real row/column data

Key points

- Start with a header row
- Then add data rows
- Keep columns consistent

Avoid tables for:

- Page layout / positioning
- Spacing and alignment
- Building “columns” for design

Modern layout uses CSS; tables should represent tabular data.

Table Anatomy

<table>, <tr>, <th>, <td>

Tables

Minimal structure

```
<table>
  <tr>
    <th>Name</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Ayşe</td>
    <td>85</td>
  </tr>
</table>
```

- <table> = the table container
- <tr> = a row
- <th> = header cell (meaning)
- <td> = data cell
- Tables are built row-by-row

Think: table → rows → cells.

Header vs Data Cells

<th> vs <td>

Tables

- <th> is for headings (labels) of columns or rows.
- <td> is for the actual data values.
- Headers improve readability and accessibility
- Using <th> correctly improves readability and accessibility. , not for styling

- Rows go left-to-right using cells
- Columns are created by cells aligned under each other
- Every row should have same number of columns (usually)
- Keep structure consistent

Key points

- Use th for headers, not td
- scope helps accessibility
- col = column header

Headers with scope

```
<tr>
  <th scope="col">Course</th>
  <th scope="col">Day</th>
  <th scope="col">Time</th>
</tr>
```

Rule: labels go in <th>, data goes in <td>.

Your First Table

Simple example (3×3)

Tables

3×3 table

```
<table>
  <tr><td>A1</td><td>A2</td><td>A3</td></tr>
  <tr><td>B1</td><td>B2</td><td>B3</td></tr>
  <tr><td>C1</td><td>C2</td><td>C3</td></tr>
</table>
```

Practice tip:

- Start simple.
- Then add headers.
- Then add caption and sections.

Table Headers Done Right: Using <th> Properly

- Use <th> for the top row or first column
- Make headers descriptive: Day, Time, Topic
- Avoid empty headers
- Headers help scanning and understanding

Caption

Adding a title to a table

Tables

Caption goes inside <table>

```
<table>
  <caption>Course Schedule</caption>
  <tr>
    <th>Course</th><th>Day</th><th>Time</th>
  </tr>
  <tr>
    <td>UMI224</td><td>Mon</td><td>10:00</td>
  </tr>
</table>
```

- Caption describes the table.
- <caption> gives the table a visible title
- Placed immediately after <table> start tag
- Helps users understand context quickly.
- Placed immediately after <table> start tag
- Keep it short and clear.

Caption is a “table title”, not a page title.

Row–Column Thinking

How to design a table

Tables

- Step 1: Decide the columns (what categories?).
- Step 2: Each row is one record/item.
- Step 3: Use <th> to label columns (and sometimes rows).
- Step 4: Keep tables consistent—same number of cells per row (unless spanning).

Design first on paper, then write HTML.

Merging Cells

rowspan and colspan

Tables

Spanning example

```
<table>
  <tr>
    <th colspan="2">Student</th>
    <th>Score</th>
  </tr>
  <tr>
    <td>Ayşe</td><td>Yılmaz</td><td>85</td>
  </tr>
  <tr>
    <td rowspan="2">Ali</td><td>Kaya</td><td>90</td>
  </tr>
  <tr>
    <td>Demir</td><td>88</td>
  </tr>
</table>
```

- colspan = merge columns (horizontal)
- rowspan = merge rows (vertical)
- Use for grouped headers or combined cells
- Do not overuse (can confuse structure)
- Use sparingly for clarity
- After merging, adjust remaining cells
- Validate structure by counting columns

If a table becomes confusing, simplify the design.

Table Sections

<thead>, <tbody>, <tfoot>

Tables

Sectioned table

```
<table>
  <thead>
    <tr><th>Course</th><th>Day</th><th>Time</th></tr>
  </thead>
  <tbody>
    <tr><td>UMI224</td><td>Mon</td><td>10:00</td></tr>
    <tr><td>UYG218</td><td>Tue</td><td>13:00</td></tr>
  </tbody>
  <tfoot>
    <tr><td colspan="3">Updated: 2026</td></tr>
  </tfoot>
</table>
```

- <thead> = header group
- <tbody> = main content rows
- <tfoot> = summary/footer rows (optional)

- Improves structure and meaning.
- Helps screen readers and tools.
- Helps large tables and styling later
- Makes styling easier later.

Use sections for readability even in small tables.

Common Table Mistakes

What to avoid

Tables

- Using tables for page layout.
- Missing header cells (<th>) so the table is unclear.
- Inconsistent number of cells per row.
- Too much rowspan/colspan causing confusion.
- No caption for a table that needs context.

Fix = keep structure simple and meaningful.

Accessible Tables (Basics)

Headers & meaning

Tables

scope helps association

```
<tr>
  <th scope="col">Course</th>
  <th scope="col">Day</th>
  <th scope="col">Time</th>
</tr>
<tr>
  <td>UMI224</td><td>Mon</td><td>10:00</td>
</tr>
```

- scope="col" = header for a column
- scope="row" = header for a row
- Used on <th> elements
- Improves screen reader output

Accessibility = clearer structure for everyone.

Mini Lab (Tables)

Build a Course Schedule table

Practice

Requirements:

- A table with a <caption> (e.g., “Course Schedule”).
- Header row using <th> with scope="col".
- 3–5 data rows in <tbody>.
- Optional: add a <tfoot> note row (colspan="3").

- Build a table with 4–6 rows
- Columns: Day | Time | Topic
- Use <caption>
- Use <th> for header row (and scope)
- Save as week5_table.html

Work in pairs for 10 minutes, then we review 2 examples.

Part 2: Forms

Collecting user input in HTML

Intro to Forms

What is a form? What data do we collect?

Forms

- A form collects user input (text, choices, files).
- Examples: login, search, contact, registration, contact.
- Forms send data to a server (later)
- Today: structure + inputs (no backend processing)
- When submitted, the browser sends the data to a URL.

Today: HTML only (no backend).

How Forms Work

User input → request

Forms

- User fills inputs (name, email, message).
- Browser builds key=value pairs using input “name” attributes.
- Browser sends the data to the form’s action URL using the chosen method.

Server receives and processes

Server responds (success/error)

Key idea: name attributes define what gets sent.

The <form> Element

action & method

Forms

Form skeleton

```
<form action="/submit" method="post">
  <!-- inputs go here -->
  <button type="submit">Send</button>
</form>
```

- <form> defines the form area
- action = where data is sent
- method = how data is sent (GET or POST)
- submit triggers sending

If action is missing, data is sent to the current page URL.

GET vs POST

Simple and practical

Forms

GET

- Default method
- Data is appended to the URL(query string)
- Good for search / filters
- Avoid for sensitive data

POST

- Data is sent in the request body (not shown in URL)
- Better for create/update actions
- Use for login/contact forms
- good for passwords, long text, sensitive info

Rule of thumb: GET for “read/search”, POST for “submit/create”.

Input Basics

type, name, value, placeholder

Forms

Key attributes

```
<label for="email">Email</label>
<input id="email" name="email" type="email"
       placeholder="name@example.com">
```

- type = input kind (normal text)
- name = key sent to server
- value = current value
- placeholder = hint text

If name is missing, that value usually won’t be submitted.

Text Inputs

text / email / password / number

Forms

Common types

```
<input type="text" name="fullName">
<input type="email" name="email">
<input type="password" name="password">
<input type="number" name="age" min="0" max="120">
```

- email may check email format.
- password masks text.
- number can use min/max.
- Always include labels.
- submit: sends the form

Keep forms simple: collect only what you need.

Choices

radio vs checkbox

Forms

Radio vs checkbox

```
<!-- ONE choice -->
<input type="radio" name="role" id="student"
value="student">
<label for="student">Student</label>

<input type="radio" name="role" id="teacher"
value="teacher">
<label for="teacher">Teacher</label>

<!-- MANY choices -->
<input type="checkbox" name="topics" id="html"
value="html">
<label for="html">HTML</label>
```

- Radio = choose ONE from many (same name).
- Checkbox = choose zero or multiple options.
- Use radio for exclusive choices (Yes/No)
- Use checkbox for multiple selections (Interests)
- Use clear labels for each choice.

Rule: radio for “one”, checkbox for “many”.

Dropdown & Long Text

<select> / <option> / <textarea>

Forms

Dropdown + multi-line

```
<label for="dept">Department</label>
<select id="dept" name="dept">
  <option value="mis">MIS</option>
  <option value="cs">CS</option>
</select>

<label for="msg">Message</label>
<textarea id="msg" name="message" rows="4"></textarea>
```

- Select = for dropdown menus, pick from a list.
- Textarea = longer message. for multi-line messages
- <option> defines each choice
- Good for limited choices
- Always label controls.

Use select when options are fixed; textarea for free text.

Labels Matter

<label for="..."> + id

Forms

Explicit association

```
<label for="phone">Phone</label>
<input id="phone" name="phone" type="tel">
```

- Labels tell users what to enter
- for="id" connects label to input
- Clicking label focuses input
- Essential for accessibility

- Clicking the label focuses the input.
- Better for screen readers.
- Improves overall UX.

Best practice: every input should have a label.

Buttons

submit vs reset vs button

Forms

Button types

```
<button type="submit">Send</button>
<button type="reset">Clear</button>
<button type="button">Just a Button</button>
```

- submit sends the form.
- reset clears values.
- :general purpose (used with JS later)
- Use clear button text: “Send Message”

In this course: use submit; avoid reset unless requested.

Basic Validation (Very Basic)

required, min/max, minlength/maxlength, pattern

Forms

HTML validation examples

```
<input type="email" name="email" required>

<input type="number" name="age" min="0" max="120">

<input type="text" name="code" minlength="4" maxlength="8">

<input type="text" name="id" pattern="[0-9]{11}">
```

- required = must fill before submit
- min/max for numbers and lengths

- minlength/maxlength for text
- pattern for simple formats
- Validation improves data quality

Validation helps, but server-side validation is still required (later).

Grouping Fields (Optional)

<fieldset> + <legend>

Forms

Grouping example

```
<fieldset>
  <legend>Contact</legend>
  <label for="name">Name</label>
  <input id="name" name="name" type="text">
</fieldset>
```

- Groups related controls.
- Legend gives the group title.
- Improves readability.

Use when a form has multiple sections.

Mini Lab (Forms)

Build a simple Contact Form (no backend)

Practice

Requirements:

- <form action="#" method="get"> (for now).
- Inputs: name (text), email (email, required).
- Message (textarea, required).
- One submit button.
- Every control must have a label (for/id).

We will test by submitting and observing the URL for GET.

Concept

“Submitted Data Table”

Wrap

- In real systems, submitted form data is stored (database) and later displayed.
- Tables are a natural way to display submitted records.
- Today we only build the HTML pieces; server/database comes later.

Think end-to-end: form collects data → system stores it → table displays it.

Week 5 Summary

What you should be able to do now

Summary

- Build tables with correct structure and headers.
- Add captions, span cells, and use thead/tbody/tfoot when useful.
- Create forms with action/method and common inputs.
- Use labels properly and add basic validation attributes.
- Complete a simple table and contact form in HTML.

Next week: deeper forms + embedded media (per syllabus).

Quick Quiz (3 minutes)

Exit ticket

Quiz

- 1) When should we use a table? When should we avoid it?
- 2) What is the difference between <th> and <td>?
- 3) What does scope="col" mean?
- 4) In forms, what do action and method do?
- 5) Give one example where GET is appropriate and one where POST is appropriate.

Collect answers in pairs, then we review quickly.