

HTML & CSS Handout

English edition

Contents

1 HTML basics

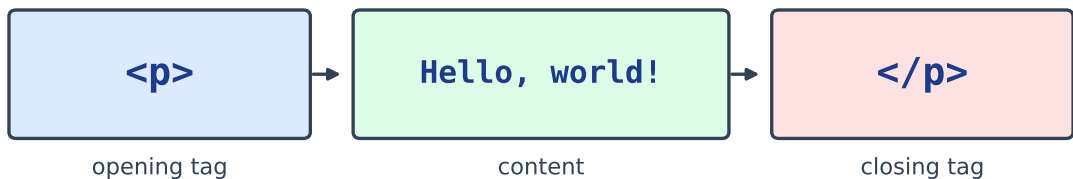
1.1 Your first page

HTML describes a web page with tags . A tag is a word in angle brackets, like `<p>`. Most come in pairs — an opening tag `<p>` and a closing tag `</p>` wrap some content. A tag plus its content is an element .

```
<p>Hello, world!</p>
<p>This is my first web page.</p>
```

- The browser draws whatever is between the tags.
- A tag you forget to close, or spell wrong, may not show correctly.

one element = opening + content + closing



Exam tip: most tags come in pairs — forgetting `</p>` breaks layout below

One element = opening tag + content + closing tag

1.2 Headings & paragraphs

Headings run from `<h1>` (biggest) down to `<h6>` (smallest). A paragraph is `<p>`. Use exactly one `<h1>` for the page's main title.

```
<h1>My Blog</h1>
<h2>About me</h2>
<p>I am learning to build web pages.</p>
```

1.3 Text formatting

Make text bold with `<strong>` and italic with `<em>`. A line break is `<br>` — it has no closing tag.

```
<p><strong>Warning:</strong> read this carefully.</p>
<p>First line<br>second line</p>
<p>You can <em>emphasise</em> any word.</p>
```

Common mistakes

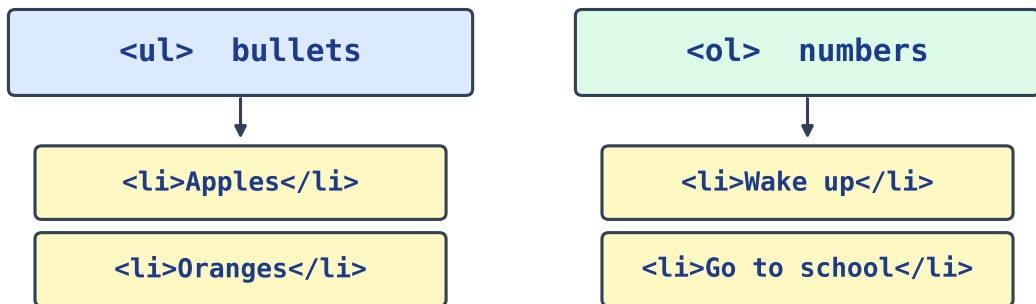
- Most tags come in pairs — `<p>...</p>`. Forgetting the closing tag breaks the layout below it.
- The tag goes in angle brackets; the text you see goes between the opening and closing tags.
- Headings run `<h1>` (biggest) to `<h6>` — do not pick a heading just for its size.

2 Lists, links & images

2.1 Lists

An unordered list `<ul>` shows bullets; an ordered list `<ol>` shows numbers. Each list item is an `<li>`.

```
<ul>
  <li>Apples</li>
  <li>Oranges</li>
</ul>
<ol>
  <li>Wake up</li>
  <li>Go to school</li>
</ol>
```



Exam tip: every item is `<li>`; wrap them in `<ul>` (bullets) or `<ol>` (numbers)

ul for bullets, ol for numbers; each item is li

2.2 Links

A link is an anchor `<a>`. Its `href` attribute holds the address it points to.

```
<a href="https://example.com">Visit Example</a>
<br>
<a href="page2.html">Go to the next page</a>
```

- Add `target="_blank"` to open the link in a new tab.

2.3 Images

An image is `<img>` — it has no closing tag. `src` is the picture's address; `alt` is text shown if the image cannot load (and read aloud to blind users).

```
<img src=|  
↪ "https://upload.wikimedia.org/wikipedia/commons/thumb/6/6b/Bitmap_VS_SVG.svg/  
    alt="bitmap versus vector" width="120">  
<p>If the picture is missing, the alt text shows instead.</p>
```

Common mistakes

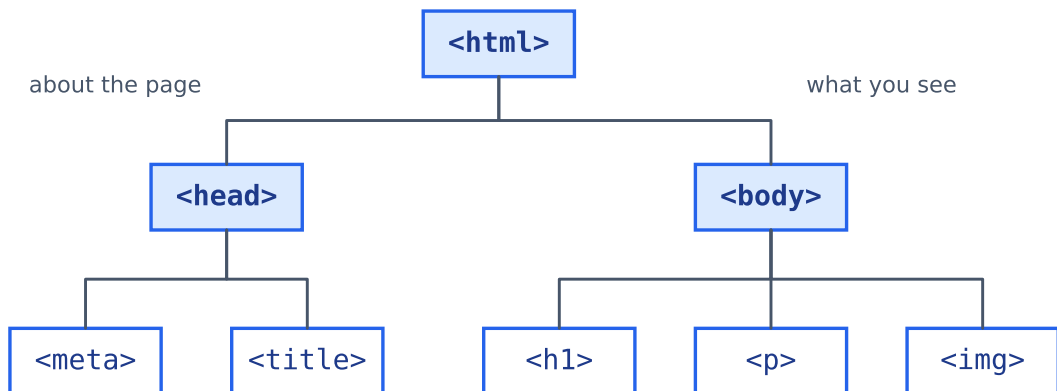
- An image needs `src` (the file) and `alt` (a description); a link needs `href`.
- `<img>` has no separate closing tag.
- A broken `src` path shows nothing — check the file name and folder.

3 Page structure

3.1 A complete page

A real page has a fixed skeleton . `<!DOCTYPE html>` says it is HTML5. Everything sits inside `<html>`. The `<head>` holds information about the page; the `<body>` holds what you see.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My Page</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>This is a complete page.</p>
  </body>
</html>
```



Elbow line from the bottom of node a to the top of node b

A page is a tree: the head describes the page, the body holds what you see

- The `<title>` shows in the browser tab.
- `<meta charset="utf-8">` lets the page show any language.

3.2 Comments & nesting

A comment `<!-- ... -->` is a note the browser ignores. Tags nest inside one another; indent each level so the structure stays clear. Close tags in the reverse order you opened them.

```
<!-- a simple navigation bar -->
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
  </ul>
</nav>
```

Common mistakes

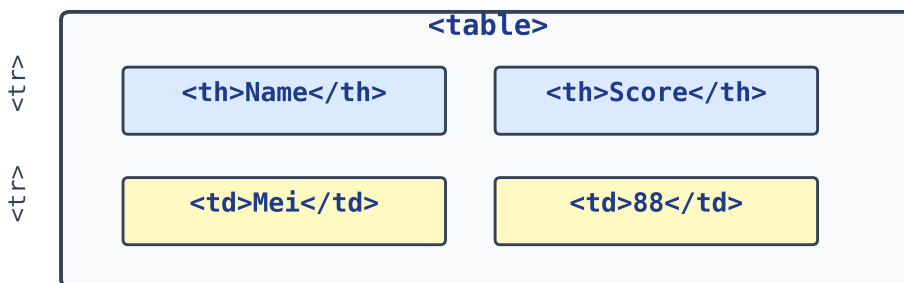
- Tags must nest, not overlap: `<b><i>text</i></b>`, never `<b><i>text</b></i>`.
- A page needs `<html>`, a `<head>` (for the title and links) and a `<body>` (for what you see).
- A comment is `<!-- ... -->`; the browser ignores it.

4 Tables & forms

4.1 Tables

A table `<table>` is built from rows `<tr>`. A header cell is `<th>` (bold, centred); a normal data cell is `<td>`.

```
<table border="1" cellpadding="6">
  <tr><th>Name</th><th>Score</th></tr>
  <tr><td>Mei</td><td>88</td></tr>
  <tr><td>Sam</td><td>71</td></tr>
</table>
```



Exam tip: each `<tr>` is a row; use `<th>` for headers and `<td>` for data cells

table → *tr* rows → *th/td* cells

4.2 Forms

A form `<form>` collects input from the user. An `<input>` is a box; a `<label>` names it; a `<button>` submits it.

```
<form>
  <label>Name: <input type="text"></label>
  <br><br>
  <label>Age: <input type="number"></label>
  <br><br>
  <button type="submit">Send</button>
</form>
```

- Common `<input>` type values: `text`, `number`, `email`, `password`, `checkbox`.

Common mistakes

- Table cells go inside rows: a `<tr>` holds the `<td>` cells.
- Every form input should have a matching `<label>` so it is clear and accessible.
- A button inside a form submits it by default — set `type="button"` if you do not want that.

5 Boxes & sections

5.1 div, span & semantic sections

A `<div>` is a generic block box that groups content on its own lines. A `<span>` is a generic inline box inside a line of text. You use them to group things so you can style or position them.

```
<div>
  <span>Price:</span> <span>$9.99</span>
</div>
<div>
  <span>Stock:</span> <span>in stock</span>
</div>
```

Semantic tags describe their role, which helps screen readers and search engines. Prefer them over plain `<div>` where they fit: `<header>`, `<nav>`, `<main>`, `<section>`, `<article>`, `<footer>`.

```
<header><h1>My Site</h1></header>
<main>
  <section>
    <h2>News</h2>
    <p>Today we launched the site.</p>
  </section>
</main>
<footer>© 2026 My Site</footer>
```

Common mistakes

- `<div>` is a block (starts a new line); `<span>` is inline (stays in the text).
- Use semantic tags like `<header>`, `<nav>` and `<footer>` where they fit, not `<div>` for everything.
- Do not use a tag just for its default look — style it with CSS instead.

div — block

```
<div>  
  block box  
  full width</div>
```

span — inline

```
text <span>inline</span> text
```

Exam tip: div groups sections; span styles a word inside a line; prefer semantic tags when possible

div is a block box; span wraps inline text

6 CSS basics

6.1 Adding CSS

CSS styles HTML. The simplest way is a `<style>` block in the `<head>` (a bigger site uses a separate `.css` file with `<link>`). A CSS rule has a selector, then `{ property: value; }` pairs called declarations.

```
<style>
  p { color: blue; font-size: 18px; }
  h1 { color: darkred; }
</style>
<h1>Styled heading</h1>
<p>This paragraph is blue.</p>
```

Separate structure from look



Exam tip: put CSS in a `.css` file and link it in `<head>`; class/id target elements

Link a stylesheet so CSS rules style HTML elements

6.2 Selectors: class & id

A class styles many elements: add `class="..."` in HTML and write `.name` in CSS. An id styles ONE element: `id="..."` and `#name`.

```
<style>
  .note { color: green; }
  #title { font-size: 24px; }
</style>
<h1 id="title">Welcome</h1>
<p class="note">First note.</p>
<p class="note">Second note.</p>
```

The selectors you will meet most, in one place:

Selector	Chooses
p	every <p> element
.note	everything with class="note"
#title	the one element with id="title"
h1, h2	several selectors at once
nav a	every <a> inside a <nav>
.btn:hover	one state of an element (see topic 10)

Common mistakes

- A class selector starts with . and an id selector with #.
- Link the stylesheet in the <head>, or put rules inside a <style> block.
- A class can be reused on many elements; an id should appear once per page.

7 Colours & text

7.1 Colours & backgrounds

Set the text colour with `color` and the background with `background-color`. A colour can be a name (`red`), a hex code (`#ff0000`), or `rgb(255, 0, 0)`.

```
<style>
  body { background-color: #f0f4ff; }
  .card { background-color: white; color: #1e3a8a; padding: 12px; }
</style>
<div class="card">A card with its own colours.</div>
```



Exam tip: colour can be a name, #hex, or `rgb()`; `background-color` paints the box

CSS colours: named, hex, and `rgb()`

7.2 Styling text

Common text properties: `font-size`, `font-weight` (e.g. `bold`), `font-family`, `text-align` (`left` / `center` / `right`), and `line-height` for spacing.

```
<style>
  h1 { font-family: Arial, sans-serif; text-align: center; }
  p { font-size: 18px; line-height: 1.6; }
</style>
<h1>Centered title</h1>
<p>Readable paragraph text with comfortable line spacing.</p>
```

Common mistakes

- `color` sets the text colour; `background-color` sets the background.
- A colour can be a name, a `#rrggbb` hex code, or `rgb(...)`.

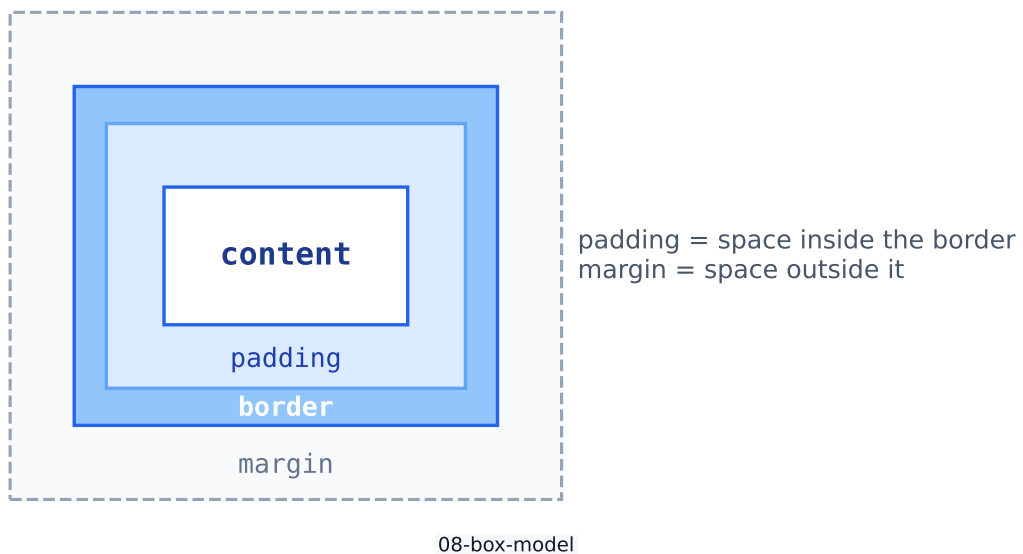
- End each declaration inside the braces with a semicolon ; — it separates the **property: value** pairs (the one after the last declaration is optional). The rule itself ends with the closing brace }.

8 The box model

8.1 The box model

Every element is a box with four layers, from inside out: the content , the padding (space inside the border), the border , and the margin (space outside the border).

```
<style>
  .box { background: #dbeafe; border: 3px solid #2563eb;
        padding: 16px; margin: 20px; }
</style>
<div class="box">content + padding + border + margin</div>
```



The box model: content, then padding, then border, then margin

8.2 Margin & padding

padding adds space **inside** the border, pushing the content inwards. **margin** adds space **outside**, pushing other boxes away. Give one value for all sides, or four (top right bottom left).

```
<style>
  .a { background: #fde68a; padding: 4px; }
  .b { background: #bbf7d0; padding: 20px; margin-top: 16px; }
</style>
<div class="a">tight padding</div>
<div class="b">roomy padding</div>
```

8.3 Display

The **display** property sets how a box flows. **block** takes a whole line; **inline** sits within a line of text; **none** removes the box completely; **flex** lays the children in a row (next topic).

```
<style>
  .hidden { display: none; }
  .chip { display: inline; background: #e0e7ff; padding: 2px 8px;
    ↪ border-radius: 6px; }
</style>
<p>Visible text. <span class="hidden">(secret)</span> <span
  ↪ class="chip">a chip</span></p>
```

Common mistakes

- Padding is inside the border; margin is the space outside it.
- **box-sizing**: **border-box** makes the set width include padding and border — often what you want.
- **display: none** hides an element completely; **visibility: hidden** leaves its space.

9 Layout

9.1 Width & centering

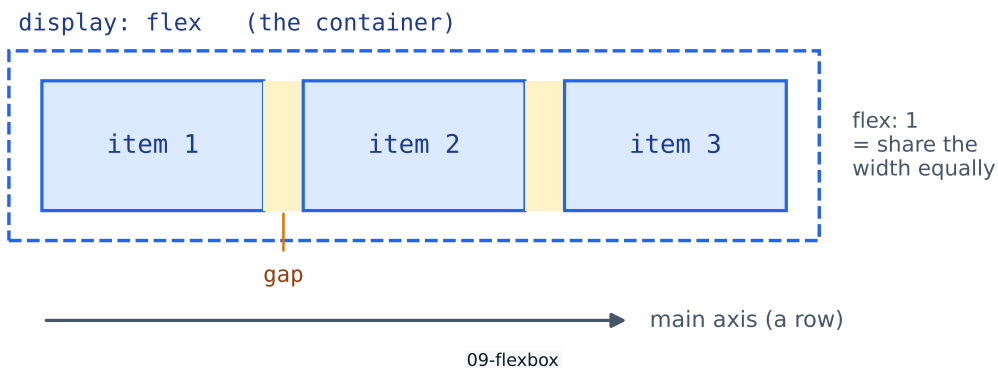
Set a box's width. To centre a block, give it a width and `margin: 0 auto` — `auto` shares the leftover space equally on the left and right.

```
<style>
  .page { width: 300px; margin: 0 auto; background: #e0e7ff;
          padding: 16px; text-align: center; }
</style>
<div class="page">A centered, fixed-width page.</div>
```

9.2 Flexbox

Flexbox lays children out in a row (or a column). Set `display: flex` on the parent — the container. `gap` adds space between children; `flex: 1` makes them share the width.

```
<style>
  .row { display: flex; gap: 10px; }
  .row div { background: #93c5fd; padding: 12px; flex: 1; text-align:
  ↪ center; }
</style>
<div class="row">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
</div>
```



A flex container lays its items along the main axis; gap adds space between them

9.3 Mini-project: a profile card

Combine boxes, colours, text and spacing into one small component .

```
<style>
  .card { width: 240px; margin: 0 auto; border: 1px solid #cbd5e1;
          border-radius: 12px; padding: 16px; font-family: sans-serif; }
  .card h2 { margin: 0; color: #1e3a8a; }
  .card .role { color: #64748b; margin: 4px 0 8px; }
</style>
<div class="card">
  <h2>Mei Chen</h2>
  <p class="role">Student . Grade 11</p>
  <p>Loves maths and building web pages.</p>
</div>
```

Common mistakes

- Centre a block with a set width using `margin: 0 auto`.
- Flexbox needs `display: flex` on the CONTAINER, not on the items.
- A width in % is relative to the parent, so check the parent's width too.

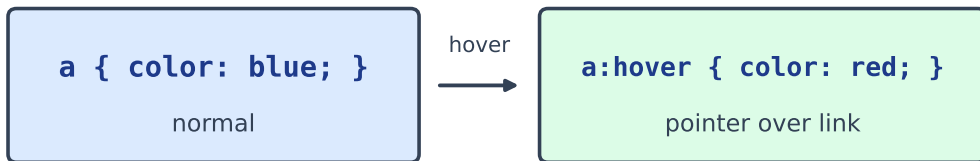
10 Going further

10.1 Hover & pseudo-classes

A pseudo-class styles an element in one particular state. `:hover` applies while the mouse is over the element. Add `transition` so the change happens smoothly instead of jumping.

```
<style>
  .btn { background: #2563eb; color: white; padding: 10px 18px;
        border: none; border-radius: 8px; font-size: 16px;
        transition: background 0.3s; }
  .btn:hover { background: #16a34a; }
</style>
<button class="btn">Hover me</button>
```

- Other useful pseudo-classes: `:focus` (an input the user clicked into) and `:first-child`.



Exam tip: `:hover`, `:focus`, `:active` are pseudo-classes — write `selector:pseudo`

`:hover` styles an element while the pointer is over it

10.2 Positioning

Positioning moves a box away from its normal place. `relative` nudges the box (and makes it the anchor for its children); `absolute` places a box exactly, measured from the nearest positioned parent; `fixed` pins it to the screen even when the page scrolls.

```
<style>
  .card { position: relative; width: 260px; height: 90px;
    background: #dbeafe; border-radius: 10px; padding: 12px; }
  .badge { position: absolute; top: -10px; right: -10px;
    background: #dc2626; color: white; padding: 4px 10px;
    border-radius: 999px; font-size: 13px; }
</style>
<div class="card">
  A product card
  <span class="badge">NEW</span>
</div>
```

- `top`, `right`, `bottom`, `left` say how far to move — they only work on positioned boxes.

10.3 Media queries & responsive design

A responsive page adapts to the screen size. A media query applies its rules only while a condition about the screen holds, such as `max-width`.

```
<style>
  .banner { background: #2563eb; color: white; padding: 16px; }
  @media (max-width: 600px) {
    .banner { background: #16a34a; } /* narrow screens turn green */
  }
</style>
<div class="banner">Resize the window: blue when wide, green when
↪ narrow.</div>
```

- Design mobile-first : write the phone styles first, then add `@media (min-width: ...)` rules for bigger screens.
- Percent widths and `max-width` let boxes shrink with the screen.

Common mistakes

- `:hover` sticks to its selector with no space: `.btn:hover`, not `.btn :hover`.
- `position: absolute` is measured from the nearest **positioned** parent — give that parent `position: relative`.

- A media query wraps whole rules: `@media (max-width: 600px) { .a { ... } }`.