UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

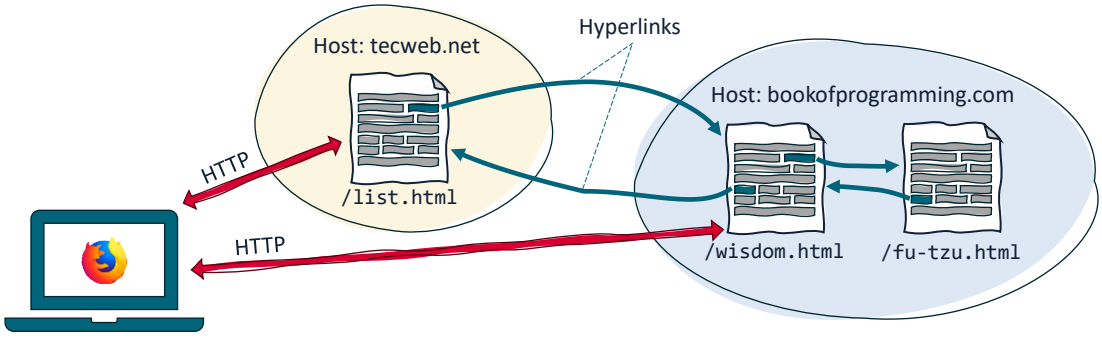
WEB TECHNOLOGIES — LECTURE 02

HTML: HYPERTEXT MARKUP LANGUAGE

Luigi Libero Lucio Starace, PhD
luigiliberolucio.starace@unina.it
https://luistar.github.io
https://www.docenti.unina.it/luigiliberolucio.starace

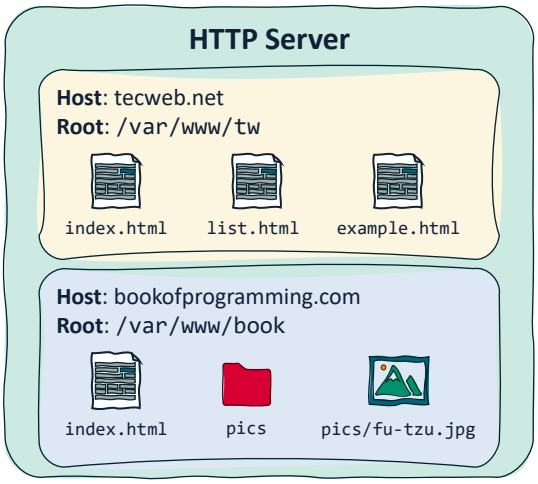
PREVIOUSLY, ON WEB TECHNOLOGIES

- We've seen **the web** is a system of interconnected **hypertext documents**, which are linked to each other through **hyperlinks**.
- Clients (typically web browsers) use HTTP to fetch these hypertexts



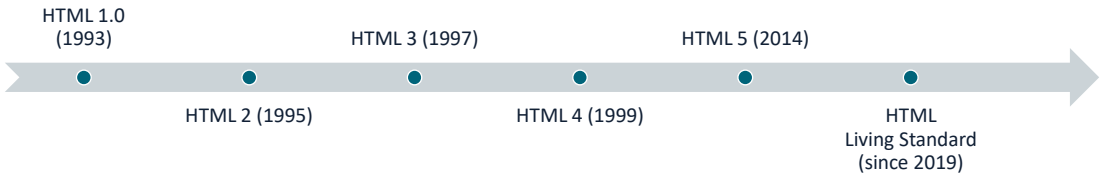
INSIDE AN HTTP SERVER

- An HTTP Server is a **software**
- **Listens** for HTTP requests on a certain port (e.g.: 80)
- Might manage multiple hosts
  - That's why theres a **Host** header in HTTP requests!
- Handles connections by serving files from the **Document root** in its filesystem
  - We do not want all our files to be accessible via HTTP, right?



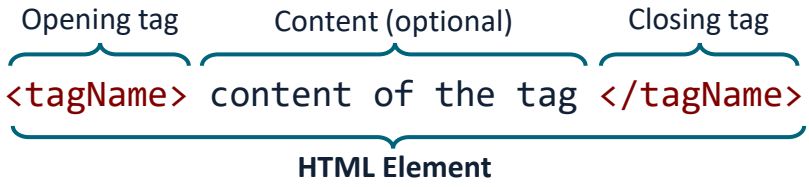
HTML: HYPERTEXT MARKUP LANGUAGE

- Web Browsers display **Documents** described using **HTML**
- A **Web App** consists of one or more documents (a.k.a. web pages)
- Key concept: **Markup Language**
- Documents are enriched with a set of annotations to control their **structure**, **formatting**, or the **relationship between their parts**.
- Several versions since 1993. We'll focus on **HTML Living Standard**



HTML: HYPERTEXT MARKUP LANGUAGE

- In HTML, the annotations are called **tags**
- Tags are denoted using angle brackets



- Opening tags may also contain key-value **attributes** (value optional)

```
<tagName attribute1="value" attribute2>
```


HTML: DOCUMENT STRUCTURE

- HTML documents must start with a `<!DOCTYPE HTML>` declaration
- Not a tag, it just tells the client what document type to expect
- The `<html>` tag represents the entire document
- It contains a `<head>` and a `<body>`

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <title>Hello World!</title>
</head>
<body> <!-- a comment -->
  <p>Hello World!</p>
</body>
</html>
```

`lang` attribute is encouraged for **accessibility**

`<head>` contains document **metadata**

`<body>` contains the actual document contents

THE HEAD ELEMENT

- Is a container for **metadata**
- Metadata are data about data, i.e.: data about the current document
- Often not shown to users, but useful for browsers and search engines
- It is required to contain a `<title>`

```
<head>
  <title>The Book of Programming</title>
  <meta charset="UTF-8">
  <meta name="description" content="Fragments from the Book of Programming">
  <meta name="keywords" content="Wisdom, programming">
  <meta name="author" content="L. L. L. Starace">
  <meta http-equiv="refresh" content="30">
</head>
```


CORE HTML ELEMENTS: HEADINGS

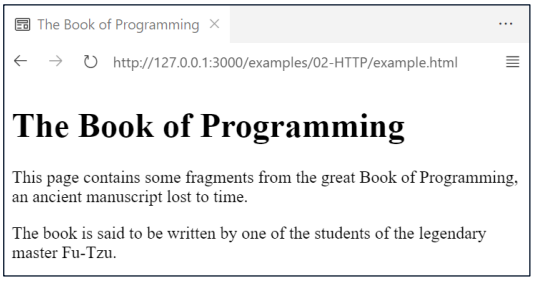
- `<h1>` to `<h6>`: Represent titles or subtitles

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <title>Headings Example</title>
</head>
<body>
  <h1>Top-level heading</h1>
  <h2>Second level heading</h2>
  <h3>Third level heading</h3>
  <h4>Fourth level heading</h4>
  <h5>Fifth level heading</h5>
  <h6>Sixth level heading</h6>
</body>
</html>
```

CORE HTML ELEMENTS: PARAGRAPHS

- `<p>`: Represent paragraphs, typically start on a new line.

```
<h1>The Book of Programming</h1>
<p>
  This page contains some fragments
  from the great Book of Programming,
  an ancient manuscript lost to time.
</p>
<p>
  The book is said to be written by
  one of the students of the legendary
  master Fu-Tzu.
</p>
```



CORE HTML ELEMENTS: COMMENTS

- Are ignored by Browsers, delimited by `<!--` and `-->`
- Can be useful to add notes or temporarily hide content

```
<h1>The two aspects</h1>
<!-- TODO: add more wisdom -->
<p>
  Below the surface of the machine,
  the program moves. Without effort,
  it expands and contracts.
</p>
<!--
<p>
  The forms on the monitor are but
  ripples on the water. The
  essence stays invisibly below.
</p> -->
```

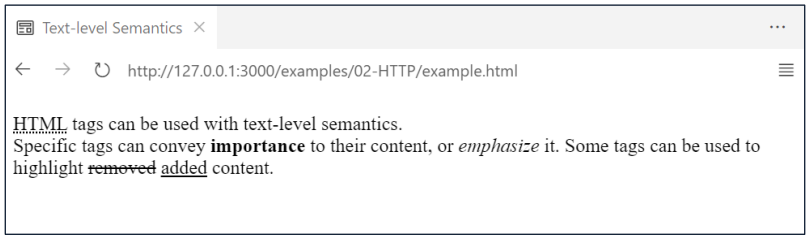
CORE HTML ELEMENTS: TEXT SEMANTICS

Tags can specify text-level semantics:

- `<em>`: *Emphasize* content
- `<strong>`: Represents **Strong** importance
- `<br/>`: Line Break (void tag, can be self-closing)
- `<abbr title="description">`: Define acronyms and abbreviations
- `<del>`: Content that has been deleted from document
- `<ins>`: Content that has been inserted in document

CORE HTML ELEMENTS: TEXT SEMANTICS

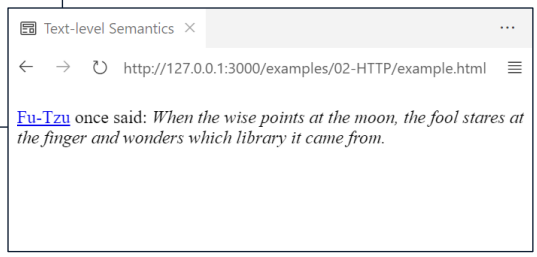
```
<p>
  <abbr title="HyperText Markup Language">HTML</abbr> tags can be used with
  text-level semantics.<br/> Specific tags can convey <strong>importance</strong>
  to their content, or <em>emphasize</em> it.
  Some tags can be used to highlight <del>removed</del> <ins>added</ins> content.
</p>
```



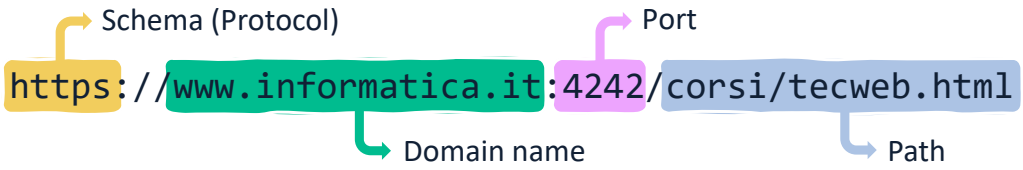
CORE HTML ELEMENTS: ANCHORS

- Hyperlinks can be defined using the anchor tag `<a>`
- The `href` attribute can be used to point to the target URL

```
<p>
  <a href="/fu-tzu.html">Fu-Tzu</a> once said:
  <em>
    When the wise points at the moon, the
    fool stares at the finger and wonders
    which library it came from.
  </em>
</p>
```



A LOOK BACK ON URLs



ANCHORS: RELATIVE VS ABSOLUTE URLS

- URLs specified by `href` can be **absolute** or **relative**
- Absolute URLs include scheme and hostname, and contain all the information necessary to reach the resource
  - e.g.: `<a href="http://tecweb.com/course/intro.html">`
- Relative URLs specify only a path. Scheme and hostname are inferred from the current context
  - e.g.: `<a href="page.html">` or `<a href="/course/react.html">`

ANCHORS: RELATIVE URLS

When a relative URL starts with a "**/**", the **entire path** is replaced

- If the current context is the page at
  http://bookofprogramming.com**/fu-tzu/fu-tzu.html**
- An anchor such as `<a href="/index.html">` points to
  http://bookofprogramming.com**/index.html**

ANCHORS: RELATIVE URLS

When a relative URL does **not** start with a "**/**", only the last part of the path is replaced

- If the current context is the page at

  http://bookofprogramming.com/fu-tzu**/fu-tzu.html**

- An anchor such as `<a href="pic.jpg">` points to

  http://bookofprogramming.com/fu-tzu**/pic.jpg**

ANCHORS: DOT SEGMENTS IN URLs

- Relative URLs can also contain «dot» segments: «.» and «..»
- Dot («`.`») represents the current directory
- Double-dot («`..`») represent the parent directory
- Assume the current path is **`/a/b/c/hello.html`**

HREF	RESULTING PATH
`./index.html`	`/a/b/c/index.html`
`../foo.html`	`/a/b/foo.html`
`../../pic.jpg`	`/a/pic.jpg`
`./../../pic.jpg`	`/a/pic.jpg` (same as above)

ANCHORS: RELATIVE OR ABSOLUTE URLS?

- Should we use **relative** or **absolute** URLs?


```
Back to <a href="/index.html">homepage</a>
Back to <a href="https://www.tw.com/index.html">homepage</a>
```

- Relative URLs should be preferred when linking resources **within** the same web application
  - This way, if the host name changes, there is no need to change the html pages
- When linking **external** web pages or resources, there is no choice but to use absolute URLs.

ANCHORS: TARGET ATTRIBUTE

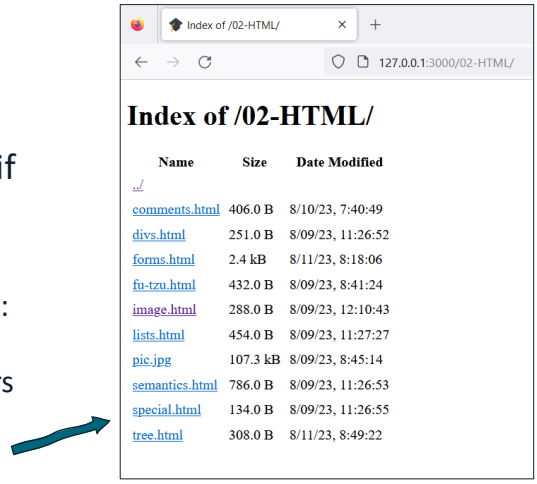
- The `target` attribute can be used to specify **where** to open the linked resource

- `<a target="self" href="pic.jpg">` should be opened in the same browser window/tab than the current page (this is the default behaviour, if you don't specify a target attribute)

- `<a target="_blank" href="pic.jpg">` should be opened in a new browser window/tab

ON URLS AND INDEX.HTML

- What happens when a URL points to a **directory** and not to a file?
- Web servers typically respond with the content of the **index.html** file, if it exists in the requested directory
- This behaviour is configurable:
  - Other default filenames used include: **home.html**, **default.html**
  - If there is no default file, HTTP servers can be configured to automatically generate an index for the directory.



CORE HTML ELEMENTS: TABLES

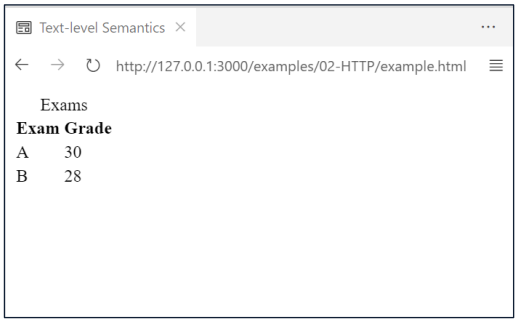
A `<table>` contains a set of `<tr>` (table rows).

- Each `<tr>` can contain one or more:
  - `<td>` (table data cells)
  - `<th>` (table headers)
- `<td>` and `<th>` contain the values to show in the respective cell.

A `<table>` might also contain a `<caption>` that describes it.

CORE HTML ELEMENTS: TABLES

```
<table>
    <caption>Exams</caption>
    <tr>
      <th>Exam</th><th>Grade</th>
    </tr>
    <tr>
      <td>A</td><td>30</td>
    </tr>
    <tr>
      <td>B</td><td>28</td>
    </tr>
</table>
```



CORE HTML ELEMENTS: LISTS

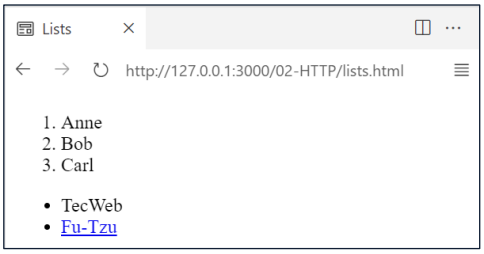
HTML defines three kinds of lists

- Ordered lists `<ol>`, used for enumerations
- Unordered lists `<ul>`, used for bullet lists
- Description lists `<dl>`, consisting of terms and their descriptions. Often used for glossaries.

CORE HTML ELEMENTS: (UN)ORDERED LISTS

- Ordered and unordered lists contain a sequence of list items `<li>`

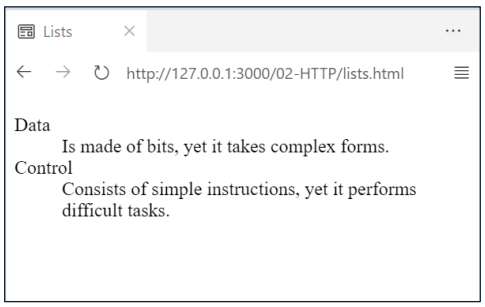
```
<ol>
  <li>Anne</li>
  <li>Bob</li>
  <li>Carl</li>
</ol>

<ul>
  <li>TecWeb</li>
  <li><a href="/">Fu-Tzu</a></li>
</ul>
```



CORE HTML ELEMENTS: DESCRIPTION LISTS

- Description lists contain a sequence of terms `<dt>` and descriptions for the prior terms `<dd>`

```
<dl>
  <dt>Data</dt>
  <dd>
    Is made of bits,
    yet it takes complex forms.
  </dd>
  <dt>Control</dt>
  <dd>
    Consists of simple instructions,
    yet it performs difficult tasks.
  </dd>
</dl>
```



CORE HTML ELEMENTS: ENTITIES

- Some characters are **reserved** in HTML
- What if we want to write in a document: `<p>3<x and y>6</p>`?
- The Browser might mix the symbols with tags
- **Character entities** should be used to display reserved characters
- Character entities look like this:
  - ***&entity_name;*** or ***&#entity_number;***

SOME USEFUL HTML ENTITIES

Result	Description	Entity Name
	Non-breaking space	
<	Less than	&lt;
>	Greater than	&gt;
&	Ampersand	&amp;
"	Double quote	&quot;
'	Single quote (apostrophe)	&apos;
©	Copyright	&copy;

The example in the previous slide should be: `<p>3&lt;x and y&gt;6</p>`

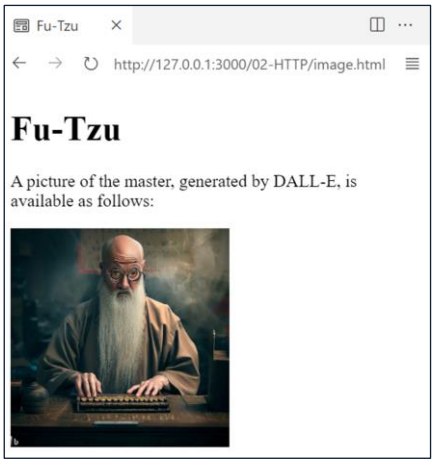
CORE HTML ELEMENTS: IMAGES

`<img>` is used to embed an image in a HTML document

- It's a void element (it shouldn't contain other elements)
- `src` attribute specifies the URL of the image to include
- `alt` attribute specifies an alternate text description for the image
- `width` and `height` attributes can be used to specify the size (in pixels) of the embedded picture in the web page

CORE HTML ELEMENTS: IMAGES

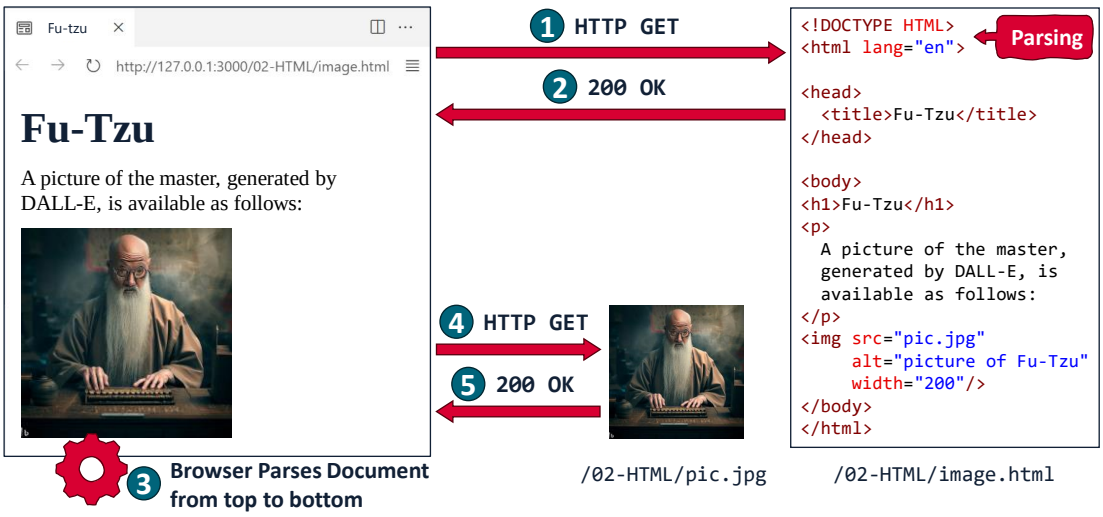
```
<h1>Fu-Tzu</h1>
<p>
  A picture of the master,
  generated by DALL-E, is
  available as follows:
</p>
<img src="pic.jpg"
     alt="picture of Fu-Tzu"
     width="200"/>
```



IMAGES: BEHIND THE SCENES

- Something new (and quite interesting) is going on!
- Until now, HTML documents were entirely **self-contained**
  - All the data in the document was **within** the document
  - We had links, but we were free not navigate them
- The last example web page included some **external content** (the image) by providing only its URL
- The image **itself** is **not included** in the HTML document
- To visualize the document, Browsers need to **fetch** additional resources!

IMAGES: BEHIND THE SCENES

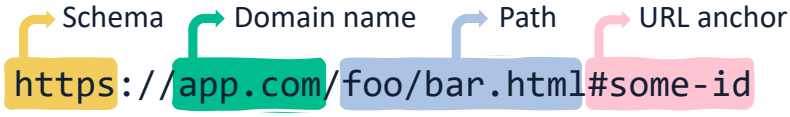


CORE HTML ELEMENTS: GLOBAL ATTRIBUTES

- We've seen some attributes (e.g.: `href`, `target`, `src`, `alt`)
- These attributes are meaningful only for some elements
  - `<strong src="pic.jpg">Hi!</strong>` would make no sense!
- Some attributes are **global**, i.e.: can be used with any HTML element
- Some examples of global attributes in HTML are:

Global Attr.	Description
`id`	Specifies a **unique** (in the document) identifier for an element
`lang`	Specifies the language for the element's content
`style`, `class`	Used for styling (we'll see in the next lecture)

URLS: ANCHORS



- The `id` attribute in html element can be used also in **URL anchors**
- An anchor represents a sort of "bookmark" inside the resource, used to tell browsers to show the content located at that bookmarked spot.
- In the example, `#some-id` is an anchor, pointing to a specific part of the resource itself, namely the element with id "`some-id`"
  - E.g.: https://luistar.github.io/publications/#conference

CORE HTML ELEMENTS: FORMS

- `<form>` elements are used to **collect user inputs**
- The input is typically sent to a server for processing (we'll see that!)
- Forms contain form elements such as `<input>`, `<label>`, `<textarea>`, `<select>`, `<form>`

```
<form>
  Message:
  <input type="text"><br><br>
  <input type="submit">
</form>
```

FORMS: INPUTS

- Different kinds of input elements are available via the `type` attribute
- Some supported types include:

Type	Description
`<input type="text">`	Displays a single-line text input field
`<input type="password">`	Displays an input field for passwords (input is hidden with *****)
`<input type="number">`	Displays an input for numbers
`<input type="radio">`	Displays a radio button (for selecting one of many choices)
`<input type="checkbox">`	Displays a checkbox (for selecting zero or more of many choices)
`<input type="button">`	Displays a clickable button

FORMS: LABELS

- `<label>` can be used to label each input element
- Using labels is a good **usability** and **accessibility** practice
- The `for` attribute of the label should be equal to the `id` of the corresponding input

```
<form>
  <label for="id_usr">Username: </label>
  <input type="text" name="usr" id="id_usr">
  <br/>
  <label for="id_pwd">Password: </label>
  <input type="password" name="pwd" id="id_pwd">
  <br/>
  <input type="submit">
</form>
```

FORMS: SUBMISSION

- Forms can be **submitted** to send collected data to some form-handler
- Upon submission, a new HTTP request is typically performed
- The URL of the form-handler, to which such request is sent, is specified by the `action` attribute on the form (defaults to the same page the form is on)
- It is also possible to specify the HTTP method to use, leveraging the `method` attribute (default is `GET`)

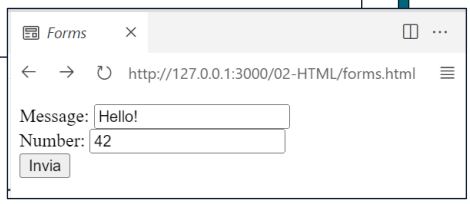
```
<form action="/handler.html" method="GET">
  <!-- form elements here -->
</form>
```

FORMS: SUBMISSION

- Upon submission, the collected user input is represented a series of name/value pairs of the form: **name1=value1**&…&**nameN=valueN**
- Each name is the name of an input element
- The corresponding value is its value at the moment of submission

```
<form action="/handler.html" method="GET">
  Message: <input type="text" name="msg"><br>
  Number: <input type="number" name="num"><br>
  <input type="submit">
</form>
```

msg=Hello!&num=42



FORMS: SUBMISSION WITH GET

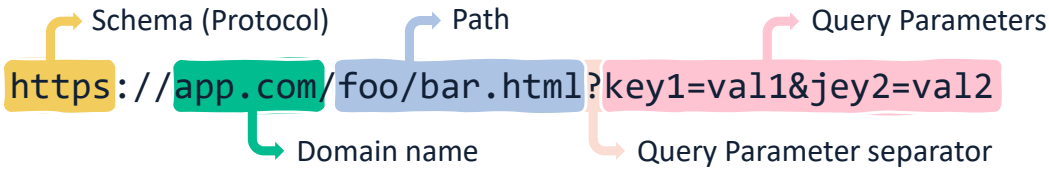
- If the method is GET, the inputs are appended to the handler's URL
- URL of the request is: `/handler.html?msg=Hello!&num=42`
- The part in bold of the URL is also called **query string**

```
<form action="/handler.html" method="GET">
  Message: <input type="text" name="msg"><br>
  Number: <input type="number" name="num"><br>
  <input type="submit">
</form>
```

msg and **num** are called **query parameters**


```
GET /handler.html?msg=Hello!&num=42 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: text/plain
Accept-Language: en-us
Connection: keep-alive
```

URLS: QUERY PARAMETERS



- Query Params are extra parameters provided to the server.
- Those parameters are a list of key/value pairs separated with "**&**"
- The Web server can use those parameters to do extra stuff before returning the resource.
  - **E.g.:** https://search-engine.com/search?q=web+technologies&lang=en

FORMS: SUBMISSION WITH POST

- If the method is POST, the inputs are sent in the request's body
- URL of the request is: `/handler.html`

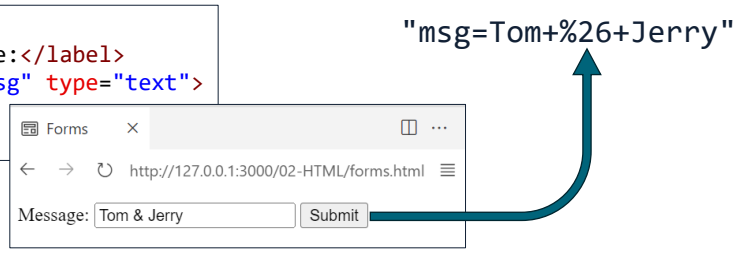
```
<form action="/handler.html" method="POST">
  Message: <input type="text" name="msg"><br>
  Number: <input type="number" name="num"><br>
  <input type="submit">
</form>
```


```
GET /handler.html HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: text/plain
Accept-Language: en-us
Connection: keep-alive

msg=Hello!&num=42
```

URL ENCODING

- Form input is encoded as a string of key-values, separated by '**&**'
- What happens if a user inputs special characters, such as '**&**'?
- These characters are replaced by character triples of the form **%XX**
  - **XX** are two hexadecimal digits representing the replaced character in ASCII
  - Spaces can be replaced by **%20** or by the **+** symbol

```
<form>
  <label for="msg">Message:</label>
  <input id="msg" name="msg" type="text">
  <input type="submit">
</form>
```



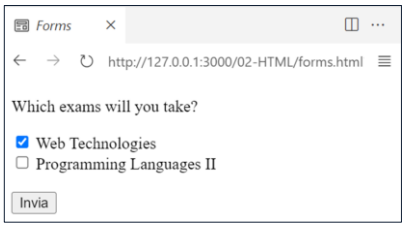
URL ENCODING: ASCII TABLES

Dec	Hex	Char	Description
32	20	space	Space
33	21	!	Exclamation mark
34	22	"	Double quote
35	23	#	Number
36	24	$	Dollar sign
37	25	%	Percent
38	26	&	Ampersand
39	27	'	Single quote
40	28	(	Left parenthesis
41	29	)	Right parenthesis
42	2A	*	Asterisk

Dec	Hex	Char	Description
43	2B	+	Plus
44	2C	,	Comma
45	2D	-	Minus
46	2E	.	Period
47	2F	/	Slash
91	5B	[	Left square bracket
92	5C	\	Backslash
93	5D	]	Right square brack.
94	5E	^	Caret / circumflex
95	5F	_	Underscore
96	60	`	Grave / accent

MORE INPUTS: CHECKBOX

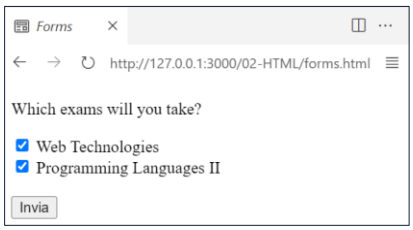
```
<form>
  <p>Which exams will you take?</p>
  <input type="checkbox" name="exams" value="web" id="web_tech">
  <label for="web_tech">Web Technologies</label><br>
  <input type="checkbox" name="exams" value="pl2" id="pl2">
  <label for="pl2">Programming Languages II</label><br><br>
  <input type="submit">
</form>
```



On submit → `"exams=web"`

MORE INPUTS: CHECKBOX

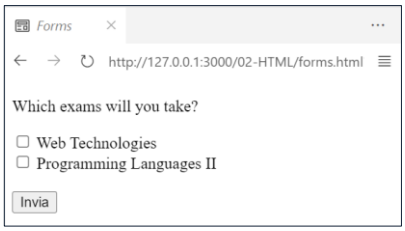
```
<form>
  <p>Which exams will you take?</p>
  <input type="checkbox" name="exams" value="web" id="web_tech">
  <label for="web_tech">Web Technologies</label><br>
  <input type="checkbox" name="exams" value="pl2" id="pl2">
  <label for="pl2">Programming Languages II</label><br><br>
  <input type="submit">
</form>
```



On submit → `"exams=web&exams=pl2"`

MORE INPUTS: CHECKBOX

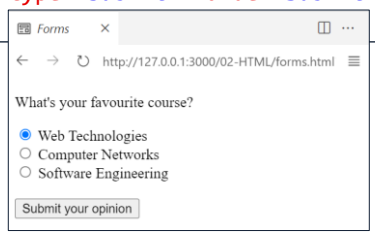
```
<form>
  <p>Which exams will you take?</p>
  <input type="checkbox" name="exams" value="web" id="web_tech">
  <label for="web_tech">Web Technologies</label><br>
  <input type="checkbox" name="exams" value="pl2" id="pl2">
  <label for="pl2">Programming Languages II</label><br><br>
  <input type="submit">
</form>
```



On submit → "" (empty string)

MORE INPUTS: RADIO BUTTONS

```
<form>
  <p>What's your favourite course?</p>
  <input type="radio" name="fav" value="web" id="web_tech">
  <label for="web_tech">Web Technologies</label><br>
  <input type="radio" name="fav" value="net" id="net">
  <label for="net">Computer Networks</label><br>
  <input type="radio" name="fav" value="se" id="se">
  <label for="se">Software Engineering</label><br><br>
  <input type="submit" value="Submit your opinion">
</form>
```

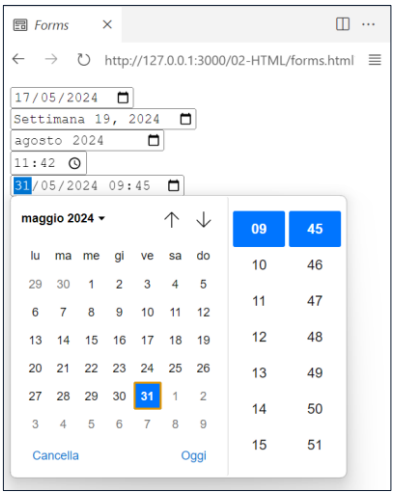


On submit → "fav=web"

MORE INPUTS: DATES

- Dedicated input types exist for **dates** and **times**:

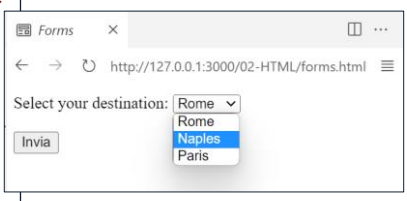
```
<form>
  <input type="date"><br>
  <input type="week"><br>
  <input type="month"><br>
  <input type="time"><br>
  <input type="datetime-local"><br>
</form>
```



MORE INPUTS: SELECT

- `<select>` can be used to define dropdowns
- The `multiple` attribute can be used to allow selecting more than one option

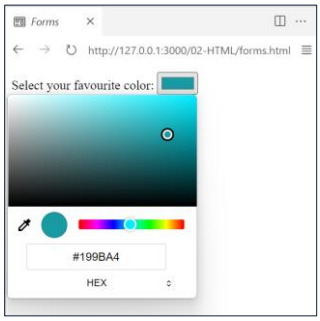
```
<form>
  <label for="dest">Select your destination:</label>
  <select name="destination" id="dest">
    <option value="Rome">Rome</option>
    <option value="Naples">Naples</option>
    <option value="Paris">Paris</option>
  </select><br><br>
  <input type="submit">
</form>
```



THERE'S MORE TO INPUTS

- There's more to inputs! (e.g.: color picker, file picker, datalists…)

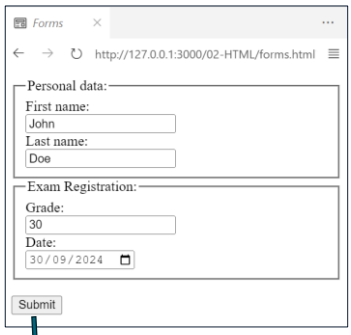
```
<form method="POST">
  <label for="col">Select your favourite color:</label>
  <input name="color" id="col" type="color"><br><br>
  <input type="submit">
</form>
```



- Check out MDN web docs for a complete reference

FORMS: GROUPING INPUTS

```
<form>
  <fieldset>
    <legend>Personal data:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname">
  </fieldset>
  <fieldset>
    <legend>Exam Registration:</legend>
    <label for="grade">Grade:</label><br>
    <input type="number" id="grade" name="grade"><br>
    <label for="date">Date:</label><br>
    <input type="date" id="date" name="date">
  </fieldset><br>
  <input type="submit" value="Submit">
</form>
```




```
fname=John&lname=Doe&grade=30&date=2024-09-30
```

GROUPING AND ORGANIZING CONTENT

- The content of a web page can be organized (**grouped**) in parts
- This can be done by using **divisions** `<div>`…
- … or **semantic tags** such as `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<aside>`, `<footer>` , and others

DIVISIONS

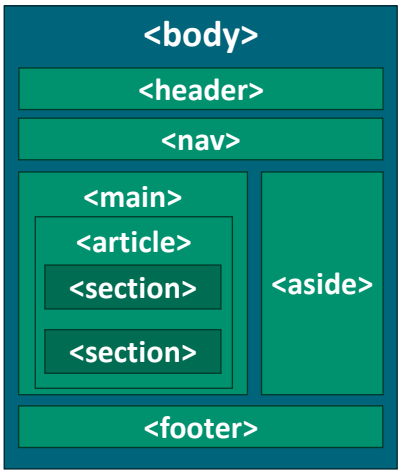
- Divisions `<div>` were the main ways of grouping content in older versions of HTML, before semantic tags were introduced.
- They bear no specific semantics, other than grouping contents that are somewhat related to each other.

```
<div>
    <h1>Section 1</h1>
    <p>Content of Section 1</p>
</div>
<div>
    <h1>Section 2</h1>
    <p>Content of Section 2</p>
</div>
```

SEMANTIC TAGS

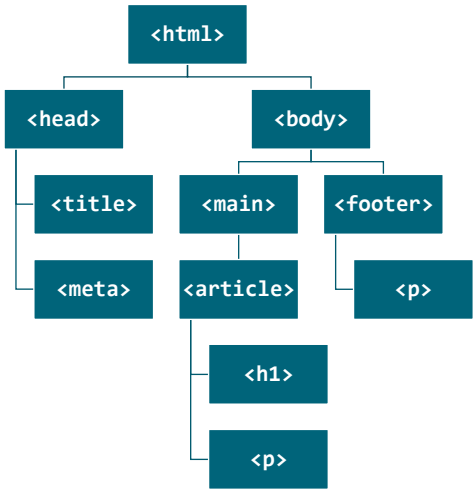
Describe the **meaning** of their content to browsers, developers, and software

- `<nav>` contains navigation links
- `<main>` indicates the main content
- `<article>` used for independent and self-contained content
- `<aside>` for tangentially-related content
- `<header>`, `<footer>`, `<section>` are self-explanatory



HTML DOCUMENTS AS TREES

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>The Book of Programming</title>
  <meta charset="UTF-8">
</head>
<body>
  <main>
    <article>
      <h1>Title</h1>
      <p>Body</p>
    </article>
  </main>
  <footer>
    <p>&copy; Web Technologies 2024</p>
  </footer>
</body>
</html>
```



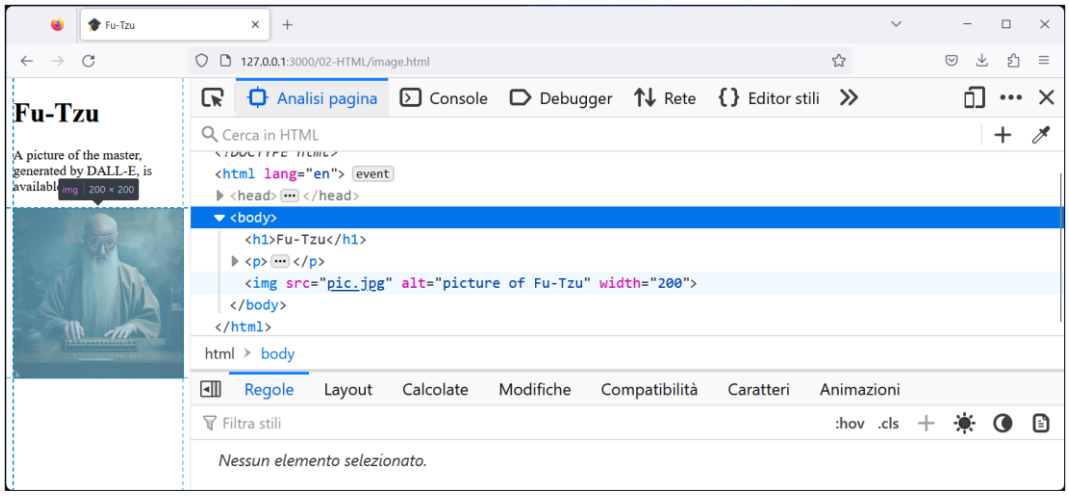
```
<header>
    <h1>Web Technologies!</h1>
    <p>This page contains some contents on Web Technologies.</p>
</header>
<main>
    <article>
        <h2>Semantic elements are good</h2>
        <p>Let's discuss semantic elements.</p>
        <section>
            <h3>Pros</h3>
            <p>They convey more information.</p>
        </section>
        <section>
            <h3>Cons</h3>
            <p>Literally none.</p>
        </section>
    </article>
    <article>
        <h2>HTML is nice</h2>
        <p>And you ain't seen styling and scripting yet!</p>
    </article>
</main>
<footer> &copy; Web Technologies course, 2024. </footer>
```

BROWSER DEV TOOLS

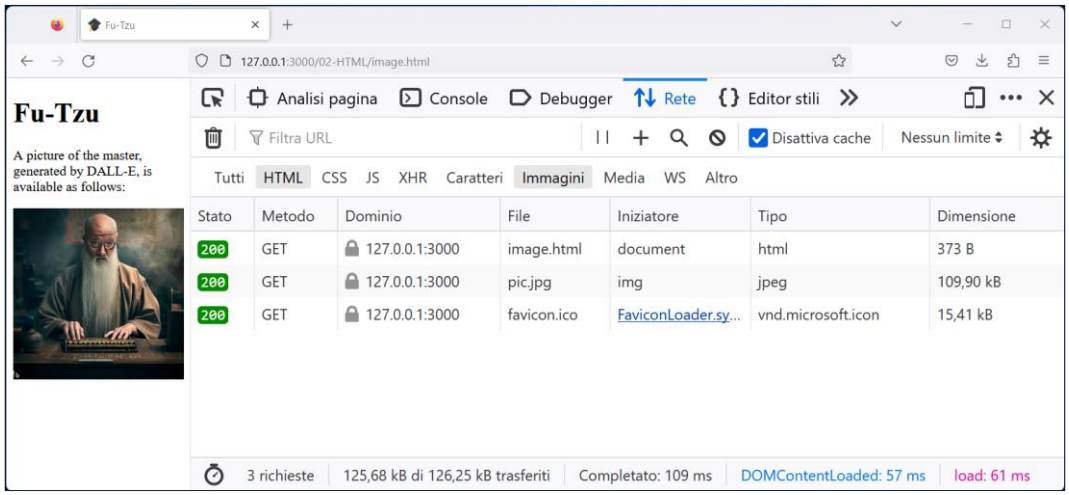
BROWSER DEV TOOLS

- Modern browsers include many features to support web developers
- These **dev tools** can be accessed by pressing the **F12** key
- Features include:
  - Possibility of **inspecting** HTML documents
  - **Network analysis** (detail of HTTP requests/responses involved)
  - **Profiling** (measuring performance and load times)
  - **Debugging** both styling elements and scripting components
- Dev tools will be your best friend as a web dev
- You'll keep them open most of the time!

BROWSER DEV TOOLS: INSPECT PAGE



BROWSER DEV TOOLS: NETWORK ANALYSIS



ASSIGNMENT

Today's lecture comes with the very first course assignment!

- The assignment will guide you in setting up a development environment with **VS Code,** including a development HTTP server
- You will build a static website by authoring and linking toghether HTML documents
- You will work with HTML Forms
- You will learn to deploy a production-grade HTTP server

REFERENCES (1/2)

- **Learn HTML**
  web.dev
  https://web.dev/learn/html
  Sections: 1 to 10 and 12 to 14
- **Learn Forms**
  web.dev
  https://web.dev/learn/forms
  Sections: 1 to 3
- **<input>: The Input (Form Input) element**
  MDN web docs
  https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

REFERENCES (2/2)

- **HTML Forms (overview)**
  W3Schools
  https://www.w3schools.com/html/html_forms.asp
- **HTML Living Standard**
  WHATWG
  https://html.spec.whatwg.org/
  ⚠️ You are **not** required to learn the entire HTML specification! Just be aware it exists and get a rough idea of how it is structured.
- **Browser DevTools**
  Mozilla Firefox DevTools User Docs: https://firefox-source-docs.mozilla.org/devtools-user/
  Google Chrome DevTools: https://developer.chrome.com/docs/devtools
  ℹ️ Get familiar with the DevTools in your web browser of choice!