University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Web Application Development

February 6, 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester: S4

Plan

Introduction

Web Application Development refers to the process of creating and maintaining software applications that are accessed over the internet through web browsers.

Types of web developers

- **Frontend developers**: Frontend developers implement web page designs using HTML and CSS. They make sure the website looks pretty on different devices, and that the forms and buttons work.

Types of web developers

- **Frontend developers**: Frontend developers implement web page designs using HTML and CSS. They make sure the website looks pretty on different devices, and that the forms and buttons work.
- **Backend developers**: Backend developers create the backbone of the web application. They write code logic that handles a user's input (for example, what should happen when you click the signup button after filling in a form).

Types of web developers

- **Frontend developers**: Frontend developers implement web page designs using HTML and CSS. They make sure the website looks pretty on different devices, and that the forms and buttons work.
- **Backend developers**: Backend developers create the backbone of the web application. They write code logic that handles a user's input (for example, what should happen when you click the signup button after filling in a form).
- **Full stack developers**: Full stack developers do bits of both backend and frontend.

Internet vs. WWW

Most people use the two terms interchangeably but they are in fact different.

- **The Internet** is a vast, international network, made up of computers and the physical connections (wires, routers, etc.) allowing them to communicate.
- **The World Wide Web** (WWW or just the Web) is a collection of software that spans the Internet and enables the interlinking of documents and resources.
  Provides a way of accessing information on the Internet.

Web Apps Compared to Desktop Apps

Advantages of web apps :

- Accessible from any internet-enabled computer.
- Usable with different operating systems and browser platforms.
- Easier to roll out program updates since only need to update software on server and not on every desktop in organization.
- Centralized storage on the server means fewer concerns about local storage (which is important for sensitive information such as health care data).

Web Apps Compared to Desktop Apps

Disadvantages of web apps :

- Internet is not always available everywhere at all time).
- Security concerns about sensitive private data being transmitted over the internet.
- Concerns over the storage, licensing and use of uploaded data.
- Problems with certain websites on certain browsers not looking quite right.
- Limited access to the operating system can prevent software and hardware from being installed or accessed (like Adobe Flash on iOS).

Internet

- Global public network.
- Accessible by anyone with an internet connection.
- Emphasis on external security.
- Open for public information sharing.
- Users: Worldwide public.

Intranet

- Private network within an organization.
- Access limited to organization members.
- Emphasis on internal security.
- Primarily for internal collaboration.
- Users: Employees or members.

Extranet

- Extends to external parties.
- Controlled access for trusted stakeholders.
- Emphasis on both internal and external security.
- Facilitates external collaboration.
- Users: External parties (e.g., clients, partners).

Static Websites

- Fixed Content
- HTML, CSS, and Limited JavaScript
- Manual Updates
- Limited Interactivity
- Examples: Personal Blogs, Brochure Sites

Dynamic Web Sites

- Dynamic Content
- Server-Side Scripting (PHP, Python, etc.)
- Easy Updates (Content Management Systems)
- High Interactivity (User Logins, E-commerce)
- Examples: Social Media, E-commerce, Web Applications

Server-side and client-side

- Server-side processes are executed on the web server.
- Client-side processes are executed on the user's device.

Server-Side

- **Execution Location**: Code runs on the web server.
- **Languages**: Commonly uses server-side languages like PHP, Python, Ruby, Node.js, etc.
- **Responsibilities**: Handles server operations, database interactions, and business logic.
- **Data Processing**: Data processing occurs on the server.
- **Security**: Secure for sensitive data and logic.
- **Examples**: Content management systems (CMS), e-commerce platforms, web applications.

Client-Side

- **Execution Location**: Code runs in the user's web browser.
- **Languages**: Primarily JavaScript.
- **Responsibilities**: Enhances user interface, interactivity, and user experience.
- **Data Processing**: Limited data processing; relies on server for critical operations.
- **Security**: Limited security for sensitive data and logic.
- **Examples**: Interactive websites, single-page applications (SPAs), browser games.

Server-side and client-side : Key take-aways

- Server-side and client-side refer to the location where certain tasks or processes are carried out in a web application.
- Server-side processes are executed on the web server before the web application is delivered to the user's device.
- Client-side processes are executed on the user's device after the web application is delivered.
- Server-side processes have more access to resources and are more secure, while client-side processes have less access to resources and are potentially less secure.

Internet

- Internet is a world-wide global system of interconnected computer networks.
- It uses the standard Internet Protocol (TCP/IP).
- Every computer is identified by a unique IP address. IP Address is a unique set of numbers (such as 110.22.33.114) which identifies a computer location.
- DNS (Domain Name Server) is used to give name to the IP Address so that user can locate a computer by a name.

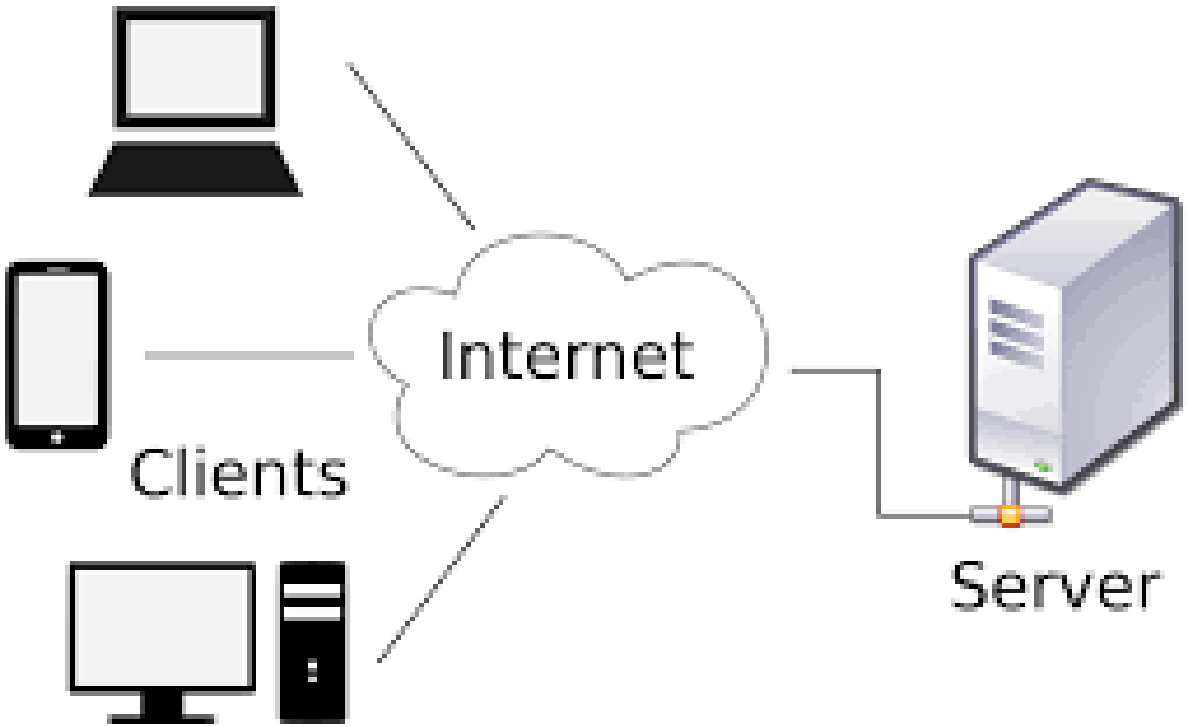
History

- Network of networks, a networking infrastructure
- Created in 1969 as ARPANET by the United States Department of Defense
- On October 29, 1969, the first internet message was sent
- The first message to be distributed was "LO", which was an attempt at "LOGIN" by Charley S. Kline to log into the computer from UCLA (University of California, Los Angeles)
- In the 1980s this networking infrastructure was made available to the general public

World Wide Web

- Tim Berners-Lee introduced WWW in 1989 and became available for everyone August of 1991.
- A distributed document delivery system.
- The documents are formatted in a markup language called HTML (HyperText Markup Language).
- Web browsers make it easy to access the World Wide Web.
- The World Wide Web uses a client-server model.

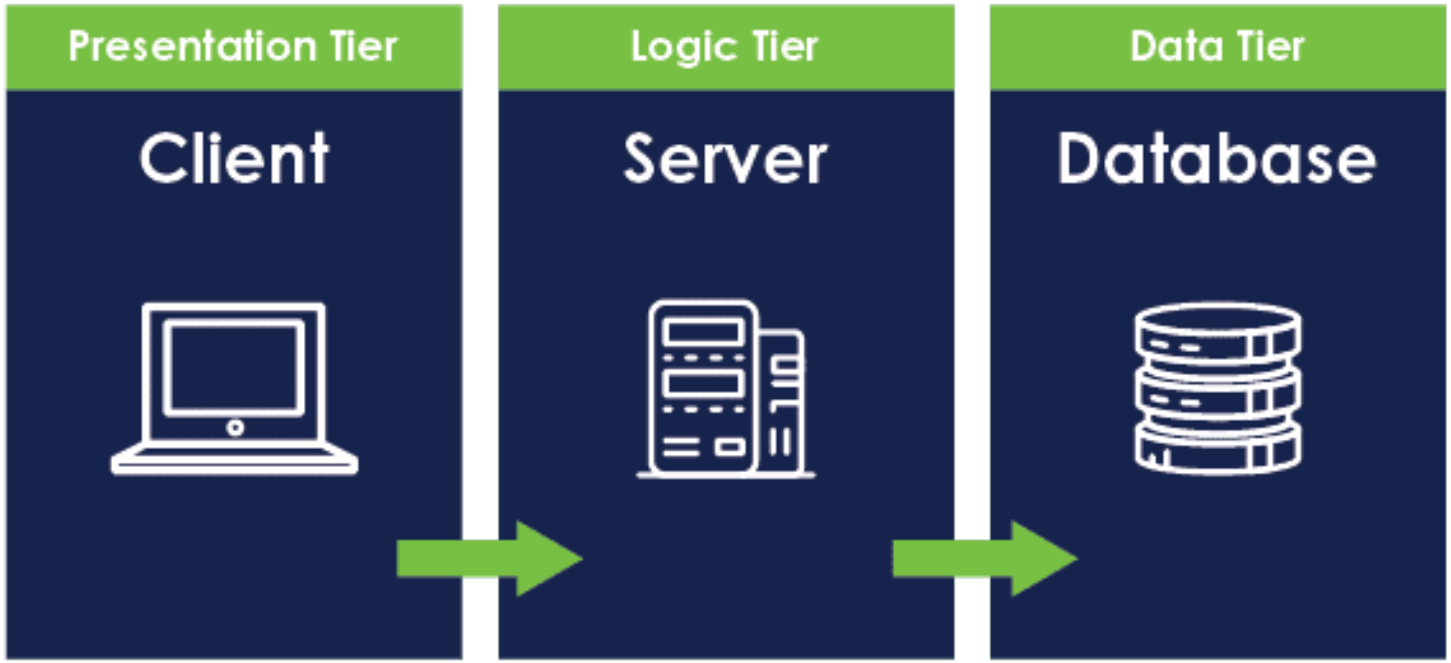
Client-Server Architecture



Three-Tier Client/Server Architecture

- To overcome the limitations of two-tier architecture, a middle tier server (also called gateway) is added between the user machine (typically a thin client) and the backend servers (database servers).
- The middle tier is where the application/business logic of the system resides and it performs a number of different functions like mapping different database queries, integrating results from different data sources and interfacing with backend legacy applications

Three-Tier Architecture



Protocol

In diplomatic circles, a protocol is the set of rules governing a conversation between people

Client and server carry on a machine-to-machine conversation

A network protocol is the set of rules governing a conversation between a client and a server

There are many protocols, HTTP is just one

URI

- A Uniform Resource Identifier (URI) is a string of characters that provides a compact and standardized way to identify or locate resources on the internet or in other contexts.
- can be further categorized into two main types:
  - Uniform Resource Locators (URLs)
  - Uniform Resource Names (URNs)

URL

Address used to locate resources on the internet.

https://www.example.com:8080/path/to/resource?param1=value1&param2=value2#section2

- Protocol: "https://"
- Domain: "www.example.com"
- Port: ":8080" (optional, specifying a custom port)
- Path: "/path/to/resource"
- Query: "?param1=value1&param2=value2"
- Fragment: "#section2"

URN

- URNs are another type of URI that serves as a persistent and location-independent identifier for resources.
- Unlike URLs, URNs do not specify how to access the resource but rather provide a unique name or identifier for it.
- URNs are intended to remain stable and unchanged over time, even if the resource's location or access methods change.
- Examples of URNs
  - "urn:isbn:0451450523" for identifying books by ISBN and
  - "urn:doi:10.1007/978-3-642-04898-2_1" for identifying digital objects by DOI.

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol)

- Set of rules governing the format and content of the conversation between a Web client and server
- Uses TCP as its underlying transport protocol
- Uses port 80
- It's a text-based communication protocol
- Stateless Protocol (doesn't require a server to retain the information of a session or the status of every communicating partner in multiple requests)

Type of data transfered

Protocol for transfer of various data formats between server and client

- Plaintext
- Hypertext
- Images
- Video
- Sound

Meta-information also can be transferred

Message structure

HTTP requests and responses consist of headers and an optional message body.

Headers contain metadata about the message, such as content type and length.

Message Structure

HTTP attaches a header, which contains information such as:

- Name and location of the page being requested,
- Name and IP address of the remote server that contains the Web page,
- IP address of the local client,
- HTTP version number,
- URL of the referring page.

Example Request

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<html>
  <body>
  ...
  </body>
</html>
```

Static Pages

- Content does not change based on user interactions or other factors.
- Simple to create and maintain, as they don't require server-side scripting or database interactions.
- Static pages Load quickly because their content is pre-generated and stored on a server, meaning they don't need to process complex requests or retrieve real-time data.
- Often used to display general information, such as company about pages, contact pages, or simple homepages.
- Typically written in HTML and can include images, text, and links.

Dynamic Pages (1)

- Real-time Updates: Dynamic pages can update their content without requiring the user to reload the entire page.
- User Interactions: They can respond to user actions, such as form submissions, button clicks, or menu selections, by processing user input and displaying relevant information or performing specific actions.
- Data Retrieval: Dynamic pages often involve interactions with databases or external APIs to retrieve and display data in real-time. (ex. product listings, search results, user profiles, ..)

Dynamic Pages (2)

- Personalization: *DP* can provide personalized content to users based on their preferences, behavior, or account information. For example, an e-commerce website may display personalized product recommendations.
- Server-side Processing: *DP* are typically generated on the server-side using programming languages like PHP, Python, Ruby, or JavaScript (Node.js).

What is SEO

- SEO stands for Search Engine Optimization.
- It's the practice of optimizing web content to rank higher in search engine results pages (SERPs).
- The process of improving the visibility of a website or a web page in search engines via the "natural," or un-paid ("organic" or "algorithmic"), search results

How do organic search listings work?

- A ***spider*** or ***crawler*** which is a component of a SE gathers listings by automatically "crawling" the web
- The spider follows links to web pages, makes copies of the pages and stores them in the SE's index
- Based on this data, the SE then indexes the pages and ranks the websites
- Major SEs that index pages using spiders: Google, Yahoo, AltaVista, MSN, AOL, Lycos

Why SEO Matters ?

- **Increased Visibility**: Higher search rankings lead to more visibility for your website.
- **Organic Traffic**: SEO can drive organic (non-paid) traffic to your site.
- **User Trust**: High-ranking sites are often perceived as more trustworthy.
- **Competitive Advantage**: SEO can give you an edge over competitors
- Accessible websites are more SEO-friendly.

SEO Elements in Web Development

- **Content**: High-quality, relevant content is essential for SEO.
- **Keywords**: Research and use keywords strategically in your content.
- **HTML Tags**: Proper use of headings, meta tags, and alt attributes.
- **Site Speed**: Fast-loading websites rank better.
- **Mobile Optimization**: Mobile-friendly sites are favored by search engines.
- **Backlinks**: Quality inbound links from other sites improve SEO.

Popular SEO Tools (1)

- Google Analytics: A comprehensive web analytics tool that tracks website traffic, user behavior, and more.
- Google Search Console: Offers insights into how Googlebot views your website, monitors search performance, and helps fix issues.
- Keyword Research Tools: Tools like Google Keyword Planner, SEMrush, and Ahrefs assist in finding relevant keywords.
- On-Page SEO Tools: Tools like Moz and Yoast SEO provide on-page optimization recommendations.

Popular SEO Tools (2)

- Backlink Analysis Tools: Tools like Majestic and Moz help analyze and manage backlinks.
- SEO Auditing Tools: Tools like Screaming Frog and Site-bulb perform in-depth site audits.
- Rank Tracking Tools: Monitor your website's search engine rankings using tools like Rank Tracker.

Questions ?

University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development
HTML

25 février 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

Introduction

Web Application Development refers to the process of creating and maintaining software applications that are accessed over the internet through web browsers.

Types of web developers

- **Frontend developers** : Frontend developers implement web page designs using HTML and CSS. They make sure the website looks pretty on different devices, and that the forms and buttons work.

Types of web developers

- **Frontend developers** : Frontend developers implement web page designs using HTML and CSS. They make sure the website looks pretty on different devices, and that the forms and buttons work.
- **Backend developers** : Backend developers create the backbone of the web application. They write code logic that handles a user's input (for example, what should happen when you click the signup button after filling in a form).

Types of web developers

- **Frontend developers** : Frontend developers implement web page designs using HTML and CSS. They make sure the website looks pretty on different devices, and that the forms and buttons work.
- **Backend developers** : Backend developers create the backbone of the web application. They write code logic that handles a user's input (for example, what should happen when you click the signup button after filling in a form).
- **Full stack developers** : Full stack developers do bits of both backend and frontend.

Internet vs. WWW

Most people use the two terms interchangeably but they are in fact different.

- **The Internet** is a vast, international network, made up of computers and the physical connections (wires, routers, etc.) allowing them to communicate.
- **The World Wide Web** (WWW or just the Web) is a collection of software that spans the Internet and enables the interlinking of documents and resources.

Provides a way of accessing information on the Internet.

Web Apps Compared to Desktop Apps

Advantages of web apps :

- Accessible from any internet-enabled computer.
- Usable with different operating systems and browser platforms.
- Easier to roll out program updates since only need to update software on server and not on every desktop in organization.
- Centralized storage on the server means fewer concerns about local storage (which is important for sensitive information such as health care data).

Web Apps Compared to Desktop Apps

Disadvantages of web apps :

- Internet is not always available everywhere at all time).
- Security concerns about sensitive private data being transmitted over the internet.
- Concerns over the storage, licensing and use of uploaded data.
- Problems with certain websites on certain browsers not looking quite right.
- Limited access to the operating system can prevent software and hardware from being installed or accessed (like Adobe Flash on iOS).

Internet

- Global public network.
- Accessible by anyone with an internet connection.
- Emphasis on external security.
- Open for public information sharing.
- Users : Worldwide public.

Intranet

- Private network within an organization.
- Access limited to organization members.
- Emphasis on internal security.
- Primarily for internal collaboration.
- Users : Employees or members.

Extranet

- Extends to external parties.
- Controlled access for trusted stakeholders.
- Emphasis on both internal and external security.
- Facilitates external collaboration.
- Users : External parties (e.g., clients, partners).

Static Websites

- Fixed Content
- HTML, CSS, and Limited JavaScript
- Manual Updates
- Limited Interactivity
- Examples : Personal Blogs, Brochure Sites

Dynamic Web Sites

- Dynamic Content
- Server-Side Scripting (PHP, Python, etc.)
- Easy Updates (Content Management Systems)
- High Interactivity (User Logins, E-commerce)
- Examples : Social Media, E-commerce, Web Applications

Server-side and client-side

- Server-side processes are executed on the web server.
- Client-side processes are executed on the user's device.

Server-Side

- **Execution Location** : Code runs on the web server.
- **Languages** : Commonly uses server-side languages like PHP, Python, Ruby, Node.js, etc.
- **Responsibilities** : Handles server operations, database interactions, and business logic.
- **Data Processing** : Data processing occurs on the server.
- **Security** : Secure for sensitive data and logic.
- **Examples** : Content management systems (CMS), e-commerce platforms, web applications.

Client-Side

- **Execution Location** : Code runs in the user's web browser.
- **Languages** : Primarily JavaScript.
- **Responsibilities** : Enhances user interface, interactivity, and user experience.
- **Data Processing** : Limited data processing ; relies on server for critical operations.
- **Security** : Limited security for sensitive data and logic.
- **Examples** : Interactive websites, single-page applications (SPAs), browser games.

Server-side and client-side : Key take-aways

- Server-side and client-side refer to the location where certain tasks or processes are carried out in a web application.
- Server-side processes are executed on the web server before the web application is delivered to the user's device.
- Client-side processes are executed on the user's device after the web application is delivered.
- Server-side processes have more access to resources and are more secure, while client-side processes have less access to resources and are potentially less secure.

What is HTML ?

- HTML(Hypertext Markup Language), is the standard markup language used to create and structure content on the World Wide Web.
- It is the foundation of web pages and is used to define the structure and layout of web documents.
- HTML documents are interpreted by web browsers to render text, images, links, forms, and other elements on a web page.

HTML Skeleton

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,
      initial-scale=1.0">
<title>Document</title>
</head>
<body>

</body>
</html>
```

Basic structure

- `<!DOCTYPE html>` : document type and version of HTML. "HTML5" is the recommended standard.
- `<html>` : The root element that encapsulates the entire HTML document.
- `<head>` : Contains metadata about the document, such as character encoding, document title, and links to external resources like CSS and JavaScript files.
- `<meta charset="UTF-8">` : Declares the character encoding for the document. UTF-8 is a widely used encoding that supports a wide range of characters from various languages.
- `<body>` : Contains the visible content of the web page, including text, images, links, and other elements.

TITLE

Best Practice

The text in your **TITLE** should be as descriptive as possible because this is what many search engines, on the internet, use for indexing your site.

Tags

- Enclosed in angle brackets (< and >)
- Usually paired
- The opening tag indicates the beginning of an element, while the closing tag is used to mark the end of that element
- Not case sensitive

```
<TITLE> My Web Page </TITLE>
```

Attributes

- Attributes are additional information or properties provided within the opening tag of an HTML element
- Used to specify various properties, behaviors, or settings for the element.

```
<img src="image.jpg" alt="An example image" />

<input type="text" name="username" disabled />

<a href="https://www.example.com">Visit Example.com</a>
```

HTML elements

- HTML documents consist of a series of elements that define the structure and content of a web page.
- Each HTML element has a specific purpose and meaning, and they can be combined to create the visual and interactive components of a webpage.
- These elements are represented by tags, and each tag has a specific purpose and meaning.

Common Tags

- `<h1>, <h2>, <h3>, ...` : Headings of various levels.
- `<p>`: Defines a paragraph of text.
- `<a>`: Creates hyperlinks to other web pages or resources.
- `<img>`: Embeds images in the document.
- `<ul>`: Defines an unordered (bulleted) list.
- `<ol>`: Defines an ordered (numbered) list.
- `<li>`: Represents individual items within a list.
- `<table>`: Defines a table.
- `<tr>, <td>`: to define the structure and content of tabular data.
- `<div>`: A generic container element used to group and structure content for styling or scripting purposes.

Semantic Markup

- Semantic markup involves using HTML tags to reinforce the meaning and structure of web content.
- HTML5 introduces a set of semantic elements designed to describe the content's purpose.

Semantic Elements in HTML5

- <**header**> : Defines the header of a section or page.
- <**footer**> : Specifies the footer of a section or page.
- <**nav**> : Represents a navigation menu.
- <**article**> : Defines independent, self-contained content.
- <**section**> : Represents a generic section of a document.

Advantages of Semantic Markup

- Accessibility : Semantic elements improve accessibility for users of assistive technologies by providing clearer structure.
- SEO (Search Engine Optimization) : Search engines can better understand the content and context of a webpage, leading to improved search rankings.
- Consistency : Semantic markup promotes consistency in web development practices and encourages better organization of content.

Example of Semantic Markup

New Semantic Elements

- <**header**> : Defines a header for a document or section.
- <**footer**> : Specifies a footer for a document or section.
- <**article**> : Defines independent, self-contained content.
- <**section**> : Represents a generic document or application section.
- <**nav**> : Defines navigation links.

These elements help structure web pages more semantically and improve SEO and accessibility.

Form Enhancements

- New form types for improved user input : email, date, time, url, search, etc.
- New attributes like placeholder, autocomplete, required, and pattern for better form validation and user experience.

Multimedia Support

- <**video**> and <**audio**> elements for embedding video and audio content natively without requiring third-party plugins.
- Support for multiple source files to ensure compatibility across different browsers.

Graphics and Animation

- Canvas API : Allows for dynamic, scriptable rendering of 2D shapes and bitmap images.
- SVG (Scalable Vector Graphics) : Supports vector graphics embedding directly in HTML documents.
- CSS3 animations and transitions : Enhance web pages with visual effects.

Enhanced Connectivity

- New technologies for communication such as WebSockets for real-time bidirectional communication between client and server.
- Offline storage capabilities with Application Cache, Web Storage, and IndexedDB for creating web applications that work offline.

Accessibility Improvements

- HTML5 places a strong emphasis on making content accessible to all users, including those with disabilities.
- ARIA (Accessible Rich Internet Applications) roles and properties can be used with HTML5 to make web applications more accessible to people with disabilities.

Headings

- Used to define the hierarchical structure and titles of sections or content on a web page.
- Improve readability, accessibility, and SEO.
- Represented by the <h1>, <h2>, <h3>, <h4>, <h5>, and <h6> elements, each indicating a different level of importance and hierarchy.
- You should not skip heading levels : e.g., an H3 should not appear after an H1, unless there is an H2 between them.

Heading Levels

- <h1> : Represents the highest level of importance and is typically used for the main heading or title of the page. There should be only one <h1> per page.
- <h2> : Represents a second-level heading, often used to subdivide the content under the main heading.
- <h3> to <h6> : Represent subsequent lower levels of headings, with h3 being less important than h2 , and so on. They are used to further structure the content within sections.

SEO Benefits (1)

- Hierarchy and Content Organization : search engines use this hierarchy to determine the importance and relationship of different sections of your page
- Keyword Usage : Search engines consider the text within headings as important clues about the page's topic.
- User Experience : Visitors can quickly skim through the headings to get an idea of the page's content.

SEO Benefits (2)

- Semantic Markup : using semantic elements like <header>, <nav>, and <section> alongside appropriate headings helps search engines understand the meaning and relationships between different parts of your page.
- Accessibility : Well-structured headings also improve web accessibility, which is a crucial factor for SEO.
- Featured Snippets : Headings are often used as the basis for featured snippets in search results.

Example Usage

```
<body>
<h1>Main Heading</h1>
<p>This is some introductory content.</p>

<h2>Section 1</h2>
<p>Content for section 1 goes here.</p>

<h3>Subsection 1.1</h3>
<p>Content for subsection 1.1 goes here.</p>

<h2>Section 2</h2>
<p>Content for section 2 goes here.</p>
</body>
```

Common Errors in Heading Usage

- Skipping Levels : Using heading levels non-sequentially (e.g., jumping from h2 to h4) can confuse both users and search engines.
- Styling for Appearance : Applying heading tags solely for styling (e.g., making text larger) rather than for semantic meaning.
- Overuse of <h1> : Using <h1> excessively throughout a page, which can lead to a lack of content hierarchy.
- Empty Headings : Creating headings with no content or using them solely as decorative elements.
- Using Headings for Links : Assigning headings to links (e.g., <a> with <h2>), which can disrupt the page's structure.

Random text generator

Text Generator

1. Microsoft Word :
   - =rand(), or
   - =lorem(4,5) : for 4 paragraphs of 5 sentences
2. VSCode : **lorem***Number_of_words*
   Example : lorem10
3. https://www.lipsum.com/

Paragraphs, <P> </P>

- used to format and present textual content on web pages, such as articles, blog posts, and informational content.
- Defined using the and tags

```
<p>
This is a paragraph of text.
It can contain multiple sentences and line breaks.
</p>

<p>This is the first paragraph.</p>
<p>This is the second paragraph.</p>
<p>This is the third paragraph.</p>
```


-
 : is used to insert a line break or line break element within the content
-
 element does not have a closing tag

```
<p>
   This is some text. <br> This text is on a new line.
</p>
```

<pre> Preformatted Text

- The tag will be displayed exactly as it is written in the HTML source code, including spaces, line breaks, and indentation.

```
<pre>
  This is an example of preformatted text.
  Here are some spaces:     and some tabs:   \t
  This text will be displayed exactly as written,
  including line breaks.
</pre>
```

Text Formatting Tags

```
<B> Bold Face </B>
<I> Italics </I>
<U> Underline </U>
<BR> Next Line
```

Add Space in HTML :

- ** ;** (Non-Breaking Space)
- ** ;**(En Space)
- ** ;**(Em Space)

<HR>

- <HR> : display a horizontal line (rule) within the content
- <HR> does not use a closing tag

```
<p>This is some text.</p>
<hr>
<p>This is more text below the horizontal rule.</p>
```

- <HR> attributes :

```
<hr size="2" width="50%" noshade align="center">
```

Property	Description	Example
`color`	Sets the text color	`color: #333;`
`font-family`	Specifies the font	`font-family: Arial, sans-serif;`
`font-size`	Sets the font size	`font-size: 16px;`
`font-weight`	Controls the font weight	`font-weight: bold;`
`font-style`	Applies font style (italic, etc.)	`font-style: italic;`
`text-align`	Aligns text horizontally	`text-align: center;`
`text-decoration`	Adds text decoration	`text-decoration: underline;`
`text-transform`	Controls text casing	`text-transform: uppercase;`
`line-height`	Sets the line height	`line-height: 1.5;`
`letter-spacing`	Adjusts character spacing	`letter-spacing: 2px;`
`text-shadow`	Applies shadow to text	`text-shadow: 2px 2px #000;`
`text-overflow`	Specifies text overflow behavior	`text-overflow: ellipsis;`
`white-space`	Specifies how white space is handled	`white-space: nowrap;`
`overflow-wrap`	Controls word wrapping	`overflow-wrap: break-word;`

Key CSS Text Formatting Properties

a element

Is used to define hyperlinks, which allow users to jump from one location to another

- **href** : (Hypertext REFerence) : URL of the page the link goes to. This attribute is what makes an <***a***> element a hyperlink.
- **link text** : The clickable text that is displayed to the user.

<a> examples

```
<a href="https://www.example.com">Visit Example.com</a>
```

Open in a New Tab :

```
<a href="https://www.example.com" target="_blank">Visit
      Example.com</a>
```

Download Link :

```
<a href="/path/to/file" download="filename">Download
      File</a>
```

Email Link :

```
<a href="mailto:example@example.com">Send Email</a>
```

<img> tag

- **src** : path to the image file
- **alt** : provides alternative text for the image. It's important for accessibility and is displayed if the image fails to load.

```
<img src="image.jpg" alt="Description of the image">
```

Ordered list

Ordered list (numbered list) :

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ol>
```

Result

1. First item
2. Second item
3. Third item

Unordered list

Unordered list (bulleted list) :

```
<ul>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ul>
```

Result

- First item
- Second item
- Third item

Table

- <**table**> : defines the table.
- <**caption**> : Sets the caption displayed above the table
- <**thead**> : contains the table header row(s).
- <**th**> : defines a header cell in the table.
- <**tbody**> : contains the table body rows.
- <**tr**> : defines a row in the table.
- <**td**> : defines a cell in the table.

```
<table>
  <thead>
    <tr>
      <th>Column 1 Heading</th>
      <th>Column 2 Heading</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Data cell 1, row 1</td>
      <td>Data cell 2, row 1</td>
    </tr>
    <tr>
      <td>Data cell 1, row 2</td>
      <td>Data cell 2, row 2</td>
    </tr>
  </tbody>
</table>
```

Result

Column 1 Heading	Column 2 Heading
Data cell 1, row 1	Data cell 2, row 1
Data cell 1, row 2	Data cell 2, row 2

About FORMs

- Forms are used to collect information from people viewing your web site.
- For example, you can use forms to find out details about your visitors through surveys and feedback, or engage in e-commerce by selling your goods and services to people.
- Forms are defined by the <FORM> </FORM> tags and are made up of different elements to collect data.
- Once the user inputs all of the information, they submit the form by using the "submit" button that you create.
- What happens with the data is a decision you will need to make.
- You can use a script to manage the data, sent the data to database, or even receive data via e-mail.

FORMs content

Forms can contain :

- Text boxes
- Password boxes
- Check boxes
- Radio buttons
- Buttons
- Select lists
- Text areas
- Labels
- Fieldsets
- Legends
- ...

Types of Form elements

Element	Description
<input type="text" >	Allows users to input single-line text.
<input type="password" >	Entered characters, typically used for passwords.
<textarea ></textarea >	Allows users to input multiple lines of text.
<input type="checkbox" >	Allows users to select one or more options from a list.
<input type="radio" >	Allows users to select one option from a list.
<select ></select >with <option ></option >	Presents a dropdown list of options to the user.
<input type="file" >	Allows users to select and upload files from their device.
<input type="submit" >	Submits the form data to the server for processing.
<input type="reset" >	Resets all form fields to their initial values.
<input type="hidden" >	Hidden from the user, used to pass data that should not be visible.
<input type="number" >	Allows users to input numeric values.
<input type="date" >, <input type="time" >, etc.	Allows users to input specific types of data (date, time, etc.).

<form> Tag

- **action** : Specifies where to send the form-data when a form is submitted.
- **method** : Defines the HTTP method for sending data (usually "GET" or "POST").
- **enctype** : Specifies how the form-data should be encoded when submitting it to the server (important for forms with file uploads).
- **autocomplete** : Indicates whether inputs can have their values automatically completed by the browser.
- **novalidate** : Tells the browser not to validate the form before submitting.
- **target** : Defines where to display the response received after submitting the form

METHOD Get

The METHOD attribute specifies the HTTP method to be used when submitting the form data :

GET :

- The default method when submitting form data
- Submitted form data will be visible in the page address field
- The length of a URL is limited (about 3000 characters)
- Never used to send sensitive data ! Better for non-secure data
- Useful for form submissions where a user want to bookmark the result

METHOD POST

- The POST method does not display the submitted form data in the page address field.
- Used for sensitive or personal information.
- Has no size limitations, and can be used to send large amounts of data.

ACTION

- The ACTION attribute defines the action to be performed when the form is submitted.
- Normally, the form data is sent to a web page on the server when the user clicks on the **submit** button.
- In the example below, the form data is sent to a page on the server called "action_page.php". This page contains a server-side script that handles the form data :

```
<form action="action_page.php">
```

Input Elements

- **type** : Specifies the type of input (e.g., text, password, submit).
- **name** : Defines the name of the input.
- **id** : provides a unique identifier for the input element
- **value** : Sets the default value of the input.
- **placeholder** : Provides a hint to the user about what to enter in the input.
- **required** : an input field must be filled out before submitting the form.
- **disabled** : Disables the input field.
- **readonly** : Makes the input field read-only.
- **autocomplete** : Specifies if the browser should autocomplete the form
- **autofocus** : Automatically focuses the input when the page loads.
- **min** and **max** : Define the minimum and maximum values for input types like "number" or "date".
- **maxlength** and **minlength** : maximum and minimum lengths of the input.
- **pattern** : Defines a regular expression against which the input's value will be checked.

Other Attributes

- **multiple** (for <input type="file"> and <select>) : Allows multiple file selections or multiple option selections.
- **selected** (for <option> in <select>) : Specifies that an option should be pre-selected when the page loads.
- **checked** (for <input type="checkbox"> and <input type="radio"> Indicates that a checkbox or radio button is selected by default.

```
<form action="/submit-form" method="post">
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username"
         placeholder="Enter your username" required><br><br>
  <label for="password">Password:</label><br>
  <input type="password" id="password" name="password"
         required><br><br>
  <label for="email">Email:</label><br>
  <input type="email" id="email" name="email" required><br><br>

  <label for="birthdate">Birthdate:</label><br>
  <input type="date" id="birthdate" name="birthdate"
         required><br><br>

  <label for="country">Country:</label><br>
  <select id="country" name="country">
    <option value="algeria">Algeria</option>
    <option value="canada">Canada</option>
    <option value="uk">UK</option>
  </select><br><br>
```

```
<label for="gender">Gender:</label><br>
<input type="radio" id="male" name="gender" value="male">
<label for="male">Male</label>
<input type="radio" id="female" name="gender"
       value="female">
<label for="female">Female</label><br><br>

<label for="color">Favorite Color:</label><br>
<input type="color" id="color" name="color"><br><br>

<label for="avatar">Profile Picture:</label><br>
<input type="file" id="avatar" name="avatar"><br><br>

<label for="bio">Bio:</label><br>
<textarea id="bio" name="bio" rows="4"
       cols="50"></textarea><br><br>

<input type="submit" value="Submit">
</form>
```

legend

- Resides within the <**fieldset**> element
- Acts as a descriptive title for the fieldset
- Improves accessibility for screen readers and other assistive technologies
- Enhances clarity and navigation for users

Fieldsets

- Fieldsets are a powerful tool for structuring and organizing forms in HTML
- They help group related input elements together,
- <**fieldset**> opening tag
- Optional <legend> element for the title
- Content : form controls, labels, and other elements
- </**fieldset**> closing tag

```
<form>
  <fieldset>
   <legend>Personal Information</legend>
      <label for="fname">First Name:</label>
      <input type="text" id="fname" name="fname" /><br>
      <label for="lname">Last Name:</label>
      <input type="text" id="lname" name="lname" /><br>
   </fieldset>
   <fieldset>
     <legend>Contact Information</legend>
       <label for="email">Email:</label>
       <input type="email" id="email" name="email" /><br>
       <label for="phone">Phone:</label>
       <input type="tel" id="phone" name="phone" /><br>
       <label for="address">Address:</label>
       <textarea id="address" name="address"></textarea><br>
     </fieldset>

   <input type="submit" value="Submit" />
</form>
```

div

The element is one of the most fundamental building blocks in HTML, serving as a generic container for any type of content

- Groups related content together semantically, even if it has no inherent meaning itself.
- Creates visual sections on a webpage for styling and layout purposes.
- Acts as a placeholder for applying CSS styles to specific sections.

div : Common Use Cases

- Creating sections like headers, footers, main content, sidebars.
- Grouping related form elements.
- Building layouts using CSS grid or flexbox.
- Highlighting specific content with unique styles

```
<div class="container">
  <h2>This is a heading</h2>
  <p>
     This is some content wrapped in a `<div>` element with
     the class "container".
  </p>
</div>
```

Article and Section

article :

- Represents a self-contained, independent piece of content
- Provides semantic meaning for both users and search engines
- Improves accessibility by helping screen readers identify and announce distinct content units

section :

- Defines a thematic section within a document
- Used to organize and structure content within an article or larger page
- Offers a way to visually and semantically divide content for better understanding

An <article> can contain multiple <section>

```
<article class="blog-post">

 <header>
   <h1>This is an Article Title</h1>
 </header>

 <section class="introduction">
   <p>This is the introduction of the article, providing a brief
           overview.</p>
 </section>

 <section class="main-body">
    <h3>Headline 1</h3>
    <p>This is the main content of the article, with detailed information and
           explanations.</p>
   <h3>Headline 2</h3>
    <p>Here's another section with additional information related to the main
           topic.</p>
  </section>

  <section class="conclusion">
   <p>This is the conclusion of the article, summarizing the key points and
           leaving a final thought.</p>
  </section>

</article>
```

Questions ?

University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development

Cascading Style Sheets (CSS)

12 mars 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

What is CSS ?

CSS : **C**ascading **S**tyle **S**heets

- A style sheet language used to describe the presentation (appearance) of documents written in HTML or XML

  Describes *how* information is to be displayed, not *what* is being displayed

- Can be embedded in HTML document or placed into separate **.css** file

Why CSS ?

- CSS separates content from presentation
- It allows for consistent styling across multiple pages of a website
- CSS simplifies the process of updating the look and feel of a website

Basic CSS rule syntax

```
selector {
  property: value;

  ...

  property: value;
}
```

The selector can either be a grouping of elements, an identifier, class, or single XHTML element (body, div, etc.)

```
p {
  font-family: sans-serif;
  color: red;
}
```

Attaching a CSS file <link>

```
<head>
  ...
<link href="filename" type="text/css" rel="stylesheet" />
  ...
</head>
```

- A page can link to multiple style sheet files
- In case of a conflict (two sheets define a style for the same HTML element), the latter sheet's properties will be used

```
<link href="style.css" type="text/css" rel="stylesheet" />
<link href="http://www.google.com/uds/css/gsearch.css"
rel="stylesheet" type="text/css" />
```

Absolute Units

Unit	Description
px	Pixels, ideal for screen-based design.
pt	Points, equal to 1/72 of an inch, used in print.
pc	Picas, equal to 12 points or 1/6 of an inch.
in	Inches, a physical unit of measurement.
cm	Centimeters, a physical unit of measurement.
mm	Millimeters, a physical unit of measurement.

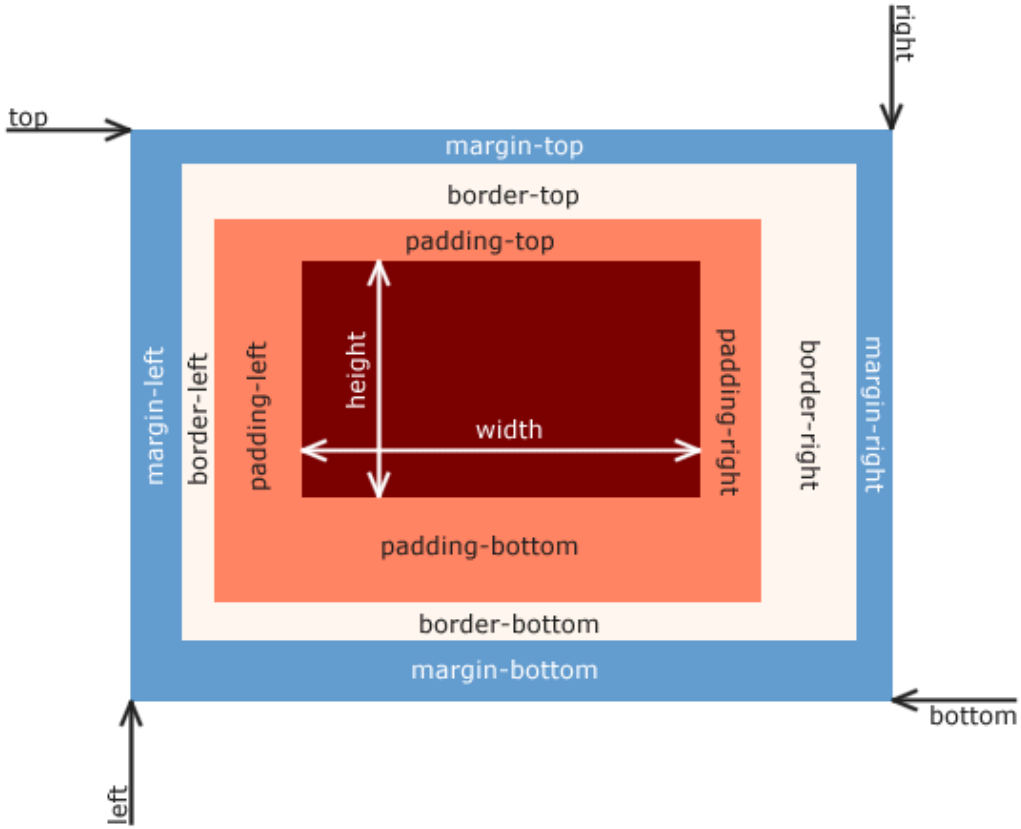
Relative Units

Unit	Description
em	Relative to the font size of the element.
rem	Relative to the font-size of the root element.
%	Percent, relative to the parent element's property.
vw	Viewport width, 1% of the viewport's width.
vh	Viewport height, 1% of the viewport's height.
vmin	1% of the viewport's smaller dimension.
vmax	1% of the viewport's larger dimension.

Experimental/Less Common Units

Unit	Description
ex	Roughly the height of a lowercase letter.
ch	Width of the "0" character in the current font.
q	Quarter-millimeters, mainly used in print contexts.

The Box Model



- Every element in the DOM (Document Object Model) has a conceptual "box" for presentation.
- The box consists of margin, padding, border, content (width, height), and offset (top, left)

Color formats in CSS

- **Color Names** : Predefined names for basic and extended colors (e.g., red, blue, green)
- **Hexadecimal Codes (Hex)** : 6-digit code preceded by # (e.g., #FF0000 for red) RGB and RGBA Values : Specify intensity of red, green, and blue components (0-255) (e.g., rgb(255, 0, 0) for red)
- **HSL and HSLA Values** : Define color based on hue, saturation, lightness (e.g., hsl(0, 100%, 50%) for red

Choosing the right color format (1)

- **Color names** : Good for basic colors or quick prototypes
- **Hex codes** : Ideal for precise color control or matching design palettes
- **RGB/RGBA values** : Useful for programmatic color manipulation or working with design tools
- **HSL/HSLA** : Suitable for users who prefer a more intuitive approach to describing colors

Choosing the right color format (2)

- **Ease of use** : Color names are the simplest, while hex codes and RGB/RGBA values require more technical knowledge.
- **Precision** : Hex codes and RGB/RGBA values offer the most precise color control.
- **Flexibility** : RGB/RGBA values are well-suited for programmatic manipulation.
- **Intuition** : HSL/HSLA can be easier to understand for some users.

Color Names

- A color can be specified by using a predefined color name : red, green, blue, yellow, black, white
- CSS/HTML support 140 standard color names
- Offer a limited range compared to other methods (hex, RGB, ..)

```
body {
  background-color: lightblue;
  color: darkslategray;
}
h1 {
  color: red;
}
p {
  color: blue;
}
```

Color Names

- **RGB(red, green, blue)** : each parameter defines the intensity of the color between 0 and 255
- **RGBA (Red, Green, Blue, Alpha)** is an extension of the RGB color model in CSS. It allows to define colors with both color and transparency alpha : Alpha channel value (0.0 - 1.0) representing transparency :
  1. 0.0 : Fully transparent
  2. 1.0 : Fully opaque (default)

```
p {    /* Applying RGB color to text - Green color */
  color: rgb(0, 128, 0);
}
div { /* Red color with 50% opacity */
    background-color: rgba(255, 0, 0, 0.5);
}
```

Hexadecimal Colors (Hex)

- Offer precise control and are widely used in web development
- Starts with # followed by 6 hexadecimal digits (0-9, A-F)
- **#rrggbb** : rr (red), gg (green) and bb (blue)

```
h1 {
  color: #FF0000; /* Red */
}

p {
  color: #333333; /* Gray */
}

a:link {
  color: #0000FF; /* Blue */
}
```

HSL/HSLA Colors

HSL(hue, saturation, lightness)

- **Hue** : Represents the color itself on a color wheel (0-360 degrees, where 0 is red and 180 is cyan).
- **Saturation** : color intensity (0% is gray, 100% is full saturation).
- **Lightness** : Controls the brightness of the color (0% is black, 100% is white).

HSLA : Alpha : Represents transparency (0.0 - 1.0, where 0.0 is fully transparent and 1.0 is fully opaque)

```
h1 {
  color: hsl(0, 100%, 50%); /* Red */
}
.header { /* Green color with 70% opacity */
  background-color: hsla(120, 100%, 50%, 0.7);
}
```

Property	Description	Example
`color`	Sets the text color	`color: #333;`
`font-family`	Specifies the font	`font-family: Arial, sans-serif;`
`font-size`	Sets the font size	`font-size: 16px;`
`font-weight`	Controls the font weight	`font-weight: bold;`
`font-style`	Applies font style (italic, etc.)	`font-style: italic;`
`text-align`	Aligns text horizontally	`text-align: center;`
`text-decoration`	Adds text decoration	`text-decoration: underline;`
`text-transform`	Controls text casing	`text-transform: uppercase;`
`line-height`	Sets the line height	`line-height: 1.5;`
`letter-spacing`	Adjusts character spacing	`letter-spacing: 2px;`
`text-shadow`	Applies shadow to text	`text-shadow: 2px 2px #000;`
`text-overflow`	Specifies text overflow behavior	`text-overflow: ellipsis;`
`white-space`	Specifies how white space is handled	`white-space: nowrap;`
`overflow-wrap`	Controls word wrapping	`overflow-wrap: break-word;`

Key CSS Text Formatting Properties

Basic Formatting

- **font-family** : Specifies the desired font family for the text.
- **font-size** : Sets the size of the text in various units (pixels, ems, rems, etc.).
- **font-weight** : Controls the boldness of the text (normal, bold, bolder, etc.).
- **color** : Defines the color of the text, using color names, hexadecimal codes, or RGB values.
- **text-decoration** : Adds decorative lines to the text (underline, overline, line-through, none).
- **text-align** : Aligns the text within the element (left, right, center, justify).
- **line-height** : Controls the vertical spacing between lines of text.

Advanced Formatting

- **letter-spacing** : Adjusts the amount of space between individual characters.
- **text-transform** : Transforms the text case (uppercase, lowercase, capitalize, etc.).
- **text-shadow** : Adds a shadow effect to the text.
- **text-indent** : Indents the first line of text.
- **font-style** : Specifies additional styles like italic or oblique.

Selecting Text :

- **: :selection** : Styles the text that is currently selected by the user

Example : CSS Text Formatig

```css
body {
  font-family: Arial,
        sans-serif;
  font-size: 1em;
  line-height: 1.5;
  margin: 0;
  padding: 0;
}

header {
  background-color: #f0f0f0;
  padding: 20px;
  text-align: center;
}

header p {
  font-style: italic;
}
```

```css
main {
  padding: 20px;
}

h1 {
  font-size: 2em;
}

p {
  margin-bottom: 15px;
  text-align: justify;
}

footer {
    text-align: center;
    padding: 10px;
    background-color: #f0f0f0;
}
```

Example : CSS Input Formatig

```
input[type="text"],
input[type="email"] {
  margin-bottom: 10px;
  width: 200px;
  height: 30px;
  border: 1px solid #ccc;
  padding: 5px;
  font-size: 16px;
}
```

```
textarea {
  width: 100%;
    /* width : 100% of its
            container */
  min-height: 100px;
  border: 1px solid #ccc;
  padding: 10px;
  font-size: 16px;
}
```

Before the Flexbox

Before the Flexbox, there were four layout modes :

- Block, for sections in a web page
- Inline, for text
- Table, for two-dimensional table data
- Positioned, for explicit position of an element

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

About Flex

- Flexbox is a one-dimensional layout model
- Designed to provide greater control over alignment and space distribution between items within a container.
- Being one-dimensional, it only deals with layout in a single direction - columns or rows - at a time. This works well for smaller layouts, such as components

Flex container properties

- **display** : Defines a flex container ; set to flex or inline-flex.
- **flex-direction** : Sets the main axis direction (row, row-reverse, column, column-reverse).
- **flex-wrap** : Controls the items' wrapping (nowrap, wrap, wrap-reverse).
- **flex-flow** : Shorthand for flex-direction and flex-wrap.
- **justify-content** : Aligns items along the main axis (f lex-start, flex-end, center, space-between, space-around, space-evenly).
- **align-items** : Aligns items along the cross axis (stretch, flex-start, flex-end, center, baseline).
- **align-content** : Distributes space between rows (stretch, flex-start, flex-end, center, space-between, space-around).

Flex Items properties

The direct child elements of a flex container automatically becomes flexible (flex) items.

- **order** : control order of items (override source order).
- **flex-grow** : how much an item can grow to fill extra space.
- **flex-shrink** : how much an item can shrink if there's not enough space.
- **flex-basis** : initial size of the item before considering grow/shrink.
- **flex** : shorthand for flex-grow, flex-shrink, and flex-basis.
- **align-self** : override default alignment for individual items.

Flex container properties

The use media of queries to create different layouts for different screen sizes and devices

Example : create a two-column layout for most screen sizes, and a one-column layout for small screen sizes (such as phones and tablets)

```
.flex-container {
    display: flex;
    flex-direction: row;
  }

  /* Responsive layout - makes a one column layout instead of a two-column
          layout */
  @media (max-width: 800px) {
    .flex-container {
      flex-direction: column;
    }
  }
```

Example

```
.flex-container {
  display: flex; /*Establishes this container as a flex container */
  justify-content: space-around; /* Distributes space around items */
  align-items: center; /* Vertically centers items in the container */
  flex-wrap: wrap; /* Allows items to wrap onto multiple lines as needed */
  padding: 20px;
  background: lightgrey;
}
.flex-item {
  background: navy;
  color: white;
  padding: 20px;
  margin: 10px;
  flex: 1; /* Allows items to grow to fill available space */
  text-align: center;
}
  /* Responsive behavior */
@media (max-width: 600px) {
  .flex-container {
      flex-direction: column; /* Stack items vertically on small screens */
  }
}
```

TP CSS

Create three web pages using **flexbox**, **text formatting**, and **input elements** :

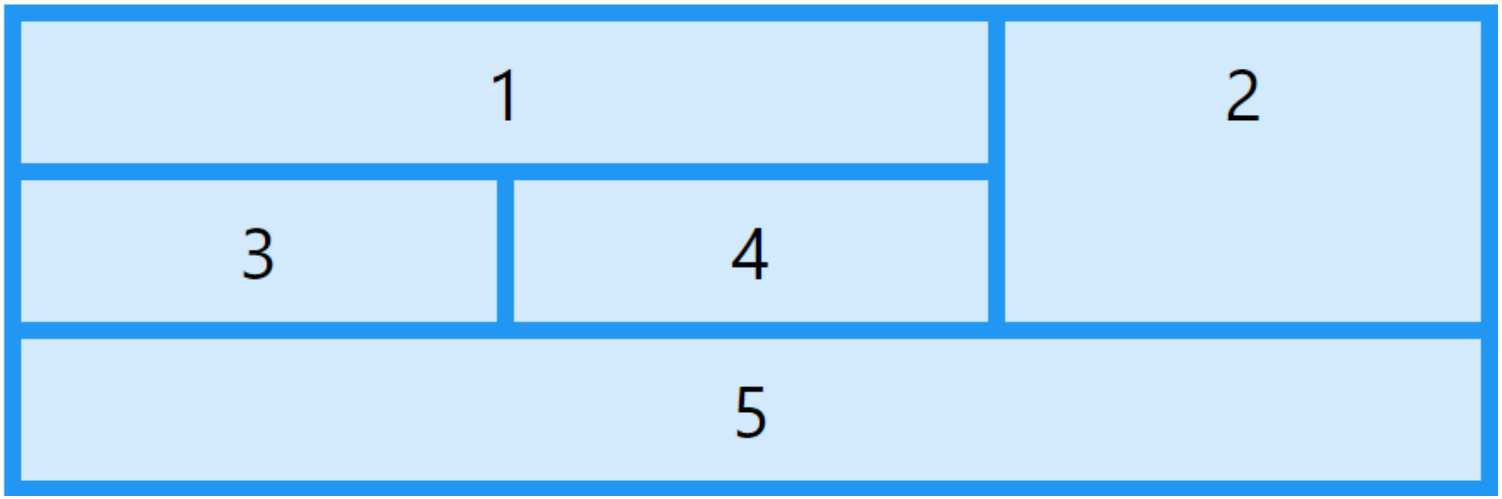
- Login Page
- Profile Page
- University Information Page

Showcase your skills in design and implementation while focusing on usability and creativity.

Example : https://github.com/hamdani2023/Flex_ISIL_2024

About Grid

Offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning



Grid example

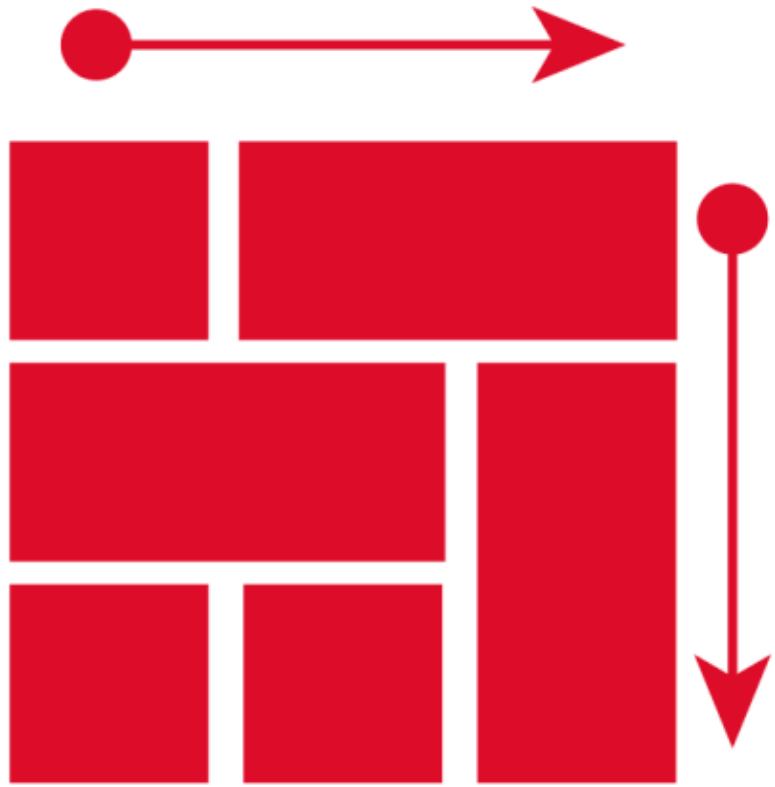
What Grid is ?

- Two-dimensional layout system
- Control of larger layouts, such as whole page layouts
- Similar to tables, it allows for items to be aligned in columns and rows.
- Easier to control and provides more layout options than old table-based layouts.

Grid vs Flexbox

Flexbox

ONE DIMENSION



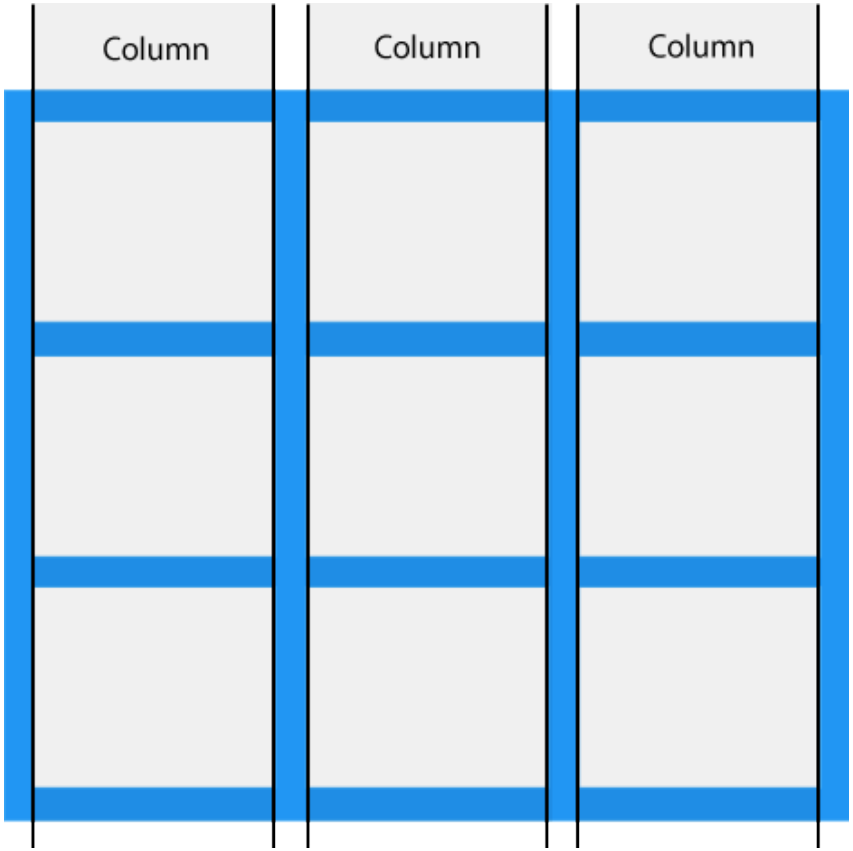
CSS Grids

TWO DIMENSIONS

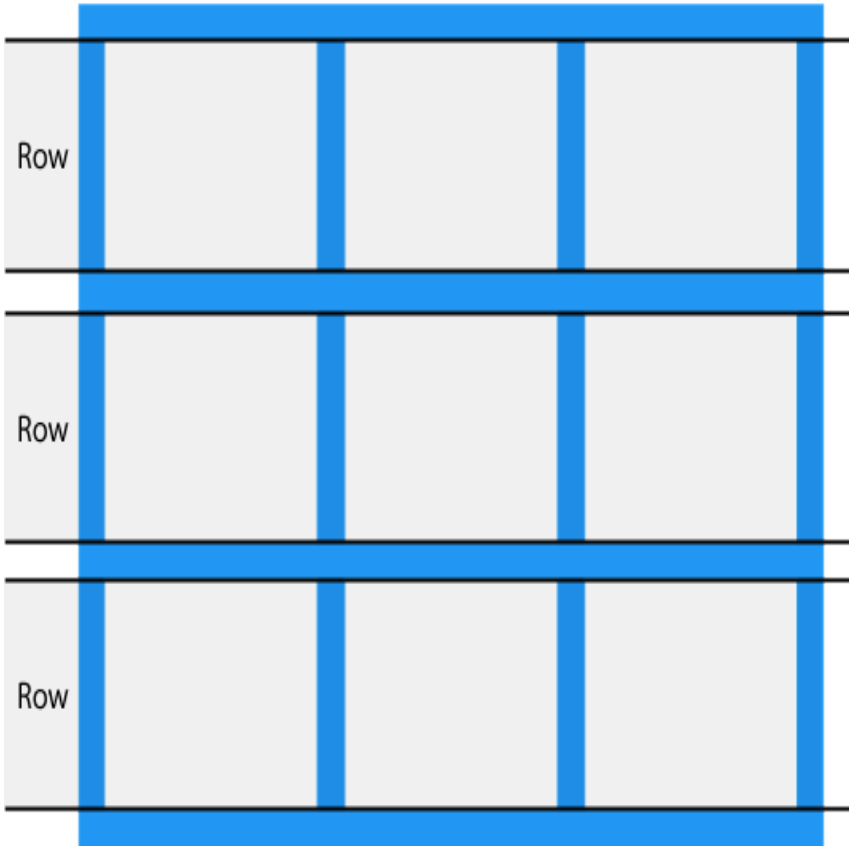
Difference between CSS Grid and Flexbox

- Flexbox is one-dimensional and CSS Grid is two-dimensional
- Flexbox is content-first and CSS Grid is layout-first :
  1. Flexbox is more content-first, adapting to the size of its content. It's useful for distributing space and aligning items in a container when their size is dynamic or unknown.
  2. Grid is more layout-first, meaning you define the grid structure and then place items into it, which can be more aligned with a designer's approach to layout planning.

Grid : Columns and Rows

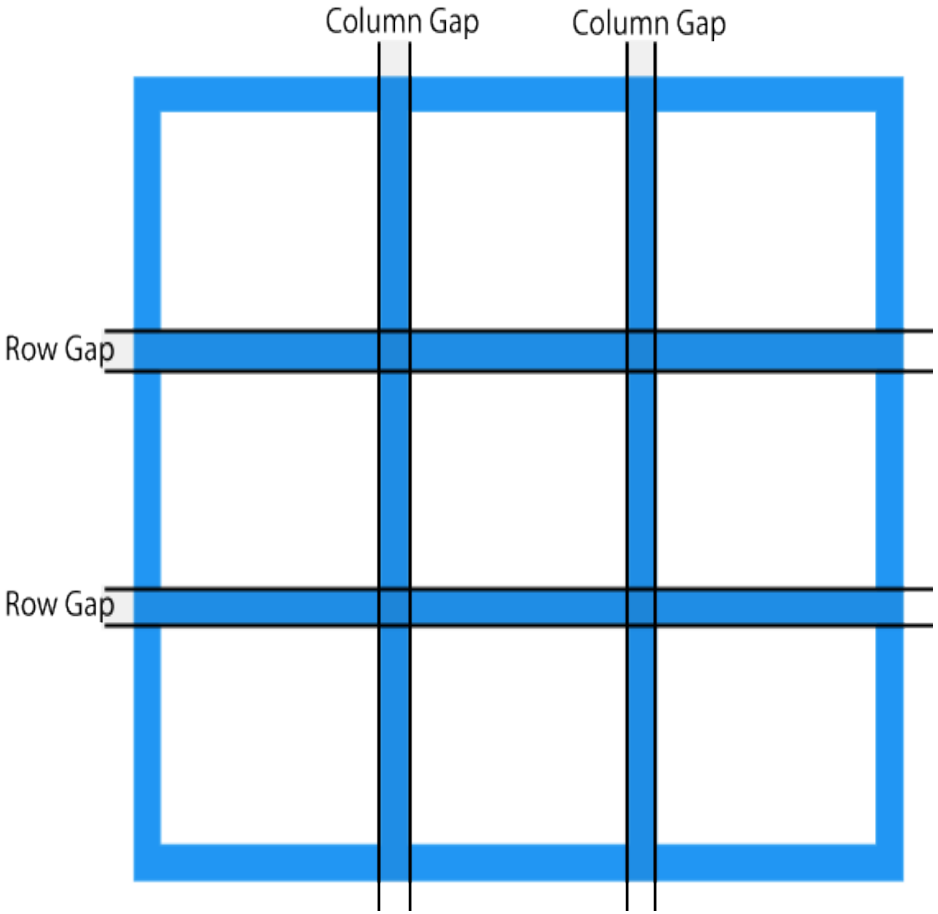


Grid Columns

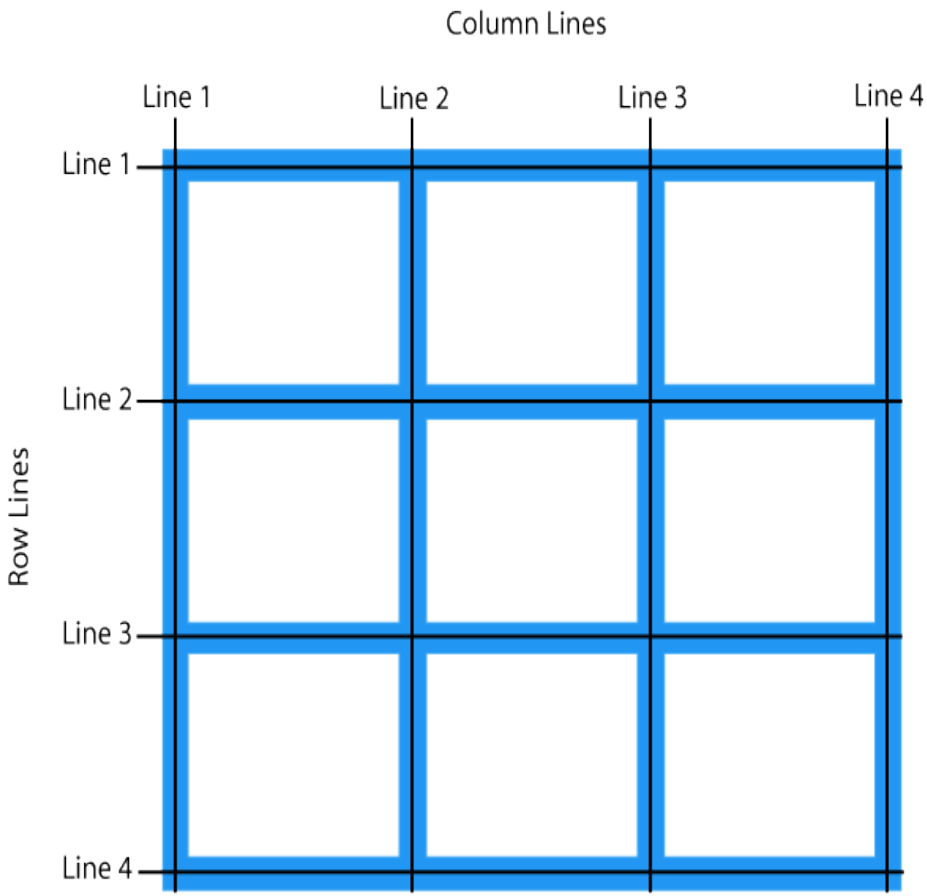


Grid Rows

Grid : Gaps and Lines



Grid Gaps



Grid Lines

Grid container properties

- **display** : Activates grid layout ; grid or inline-grid
- **grid-template-columns, grid-template-rows** : Define sizes of columns and rows.
- **grid-template-areas** : Assigns names to parts of the grid layout.
- **gap (grid-gap)** : Sets space between rows and columns.
- **grid-auto-columns, grid-auto-rows** : Sizes for implicitly created grid tracks.
- **grid-auto-flow** :Directs auto-placement of grid items ; row, column, dense.
- **justify-items** : Aligns items horizontally within their grid area.
- **align-items** : Aligns items vertically within their grid area.
- **justify-content** : Aligns the grid within the container horizontally.
- **align-content** : Aligns the grid within the container vertically.
- **grid-template** : Shorthand for grid-template-rows, grid-template-columns, and grid-template-areas.

Example

```
.boxes {
  display: grid;
     /*  grid-template-columns: 1fr 2fr 1fr;
      can be written :
      grid-template-columns: repeat(3, 1fr)

      grid-template-columns: repeat(auto-fit,
            minmax(100px, 1fr));
      */
  grid-template-columns: repeat(3, 1fr);
       /*1fr: one fraction*/
  gap: 1em;
  grid-auto-rows: minmax(100px, auto);
    /* define the size of rows */
}
```

Grid Item Properties

- **grid-column-start** : Item's start line in grid columns.
- **grid-column-end** : Item's end line in grid columns.
- **grid-row-start** : Item's start line in grid rows.
- **grid-row-end** : Item's end line in grid rows.
- **grid-column** : Shorthand for column start/end (e.g., 1 / 3).
- **grid-row** : Shorthand for row start/end (e.g., 2 / 4).
- **grid-area** : Shorthand for row/column start/end or named area.
- **justify-self** : Aligns item in cell along row axis.
- **align-self** : Aligns item in cell along column axis.
- **order** : Defines the order in which an item appears in the grid

Naming Grid Items

Use the grid-template-areas property on the grid container to define named areas. Each name corresponds to a specific area of the grid. Unnamed areas can be marked with a period (.).

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: auto;
  grid-template-areas:
      "header header header"
      "sidebar content content"
     "footer footer footer";
}
.header  {  grid-area: header;  }
.sidebar {  grid-area: sidebar; }
.content {  grid-area: content; }
.footer  {  grid-area: footer;  }
```

Questions ?

University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development

Cascading Style Sheets (CSS)

12 mars 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

JavaScript

- JavaScript is a versatile programming language used for creating dynamic web content.
- Control inputs in JavaScript allow users to interact with web pages, enhancing user experience and functionality.

Manipulating Control Inputs

- JavaScript enables dynamic manipulation of control inputs.
- Using JavaScript, you can :
  - Retrieve input values.
  - Validate input data.
  - Dynamically update input options.
  - Trigger actions based on user input.

Accessing the Input Arena

- **Accessing elements by ID** : document.getElementById('inputId')
- **Targeting by name** : document.getElementsByName('nameAttribute')
- **Selecting by tag** : document.getElementsByTagName('input')
- **Using CSS Selectors** : document.querySelector('#inputId')

Modifying Input Values

Changing input content : element.value = 'newValue'

```
document.getElementById("username").value = "isil";

document.getElementById("password").value = "isil@2024";
```

Example : Form Validation

```
<script>
  function validateForm() {
  var username = document.getElementById("username").value;
  var password = document.getElementById("password").value;

  if (username.trim() === "isil" && password.trim()==="isil"){
    // Redirect to the Profile page
    window.location.href =
          "http://127.0.0.1:3000/Profile.html";
    return false; // Prevent form submission
  } else {
    alert("Invalid username or password.");
    return false; // Prevent form submission
  }
}
</script>
```

```
<script>

    // Function to clear form inputs
    function clearForm() {
      document.getElementById("username").value = "";
      document.getElementById("password").value = "";
    }
</script>
```

```
 <form action="post" onsubmit="return validateForm()">

        ...

   <button onclick="return clearForm()">Cancel</button>

   <button type="submit">Enter</button>

          ...
```

Example : Dynamic Dropdown Menu

```
<script>
 function populateDropdown() {
 var select = document.getElementById("country");
 var countries = ["Algeria", "Canada", "UK", "Australia"];
 var defaultOption = document.createElement("option");
 defaultOption.text = "Select a country";
 defaultOption.disabled = true;
 defaultOption.selected = true;
 select.add(defaultOption);

 countries.forEach(function(country) {
   var option = document.createElement("option");
   option.text = country;
   select.add(option);
 });
}
 window.onload = function() {
     populateDropdown();
  };
</script>

<select id="country" name="country"></select>
```

Questions ?

University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development

eXtensible Markup Language (XML)

10 avril 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

What is XML

- XML stands for e**X**tensible **M**arkup **L**anguage.
- XML was designed to store and transport data
- It's a text-based markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- Designed to be self-descriptive and easy to understand

Why XML ?

- **Human-readable** : XML documents readability makes it simpler for programmers to comprehend the data structure and content
- **Platform-Independent** : XML documents can be processed and interpreted by different platforms, operating systems, and programming languages.
- **Extensible & Flexible** : creation of custom tags and structures tailored to specific data needs
- **Well-Established Standard** : XML is a widely established standard for encoding data.
- **Structured Data Representation** :Data is represented in a hierarchical tree-like model, making it easy to navigate, search, and manipulate

Applications of XML

- **Web Development and Web Services** : XML is extensively used in web services for exchanging data across different systems and platforms via the internet
- **Configuration Files and Settings** : XML is commonly used for storing configuration settings and preferences in applications and systems
- **Data Exchange and Integration** : XML serves as a common medium for exchanging data between different information systems
- **Other Applications :** Scientific Data, E-Books, Document Management (CMS, ..)

XML vs HTML

- XML is not a replacement for HTML but is one level up

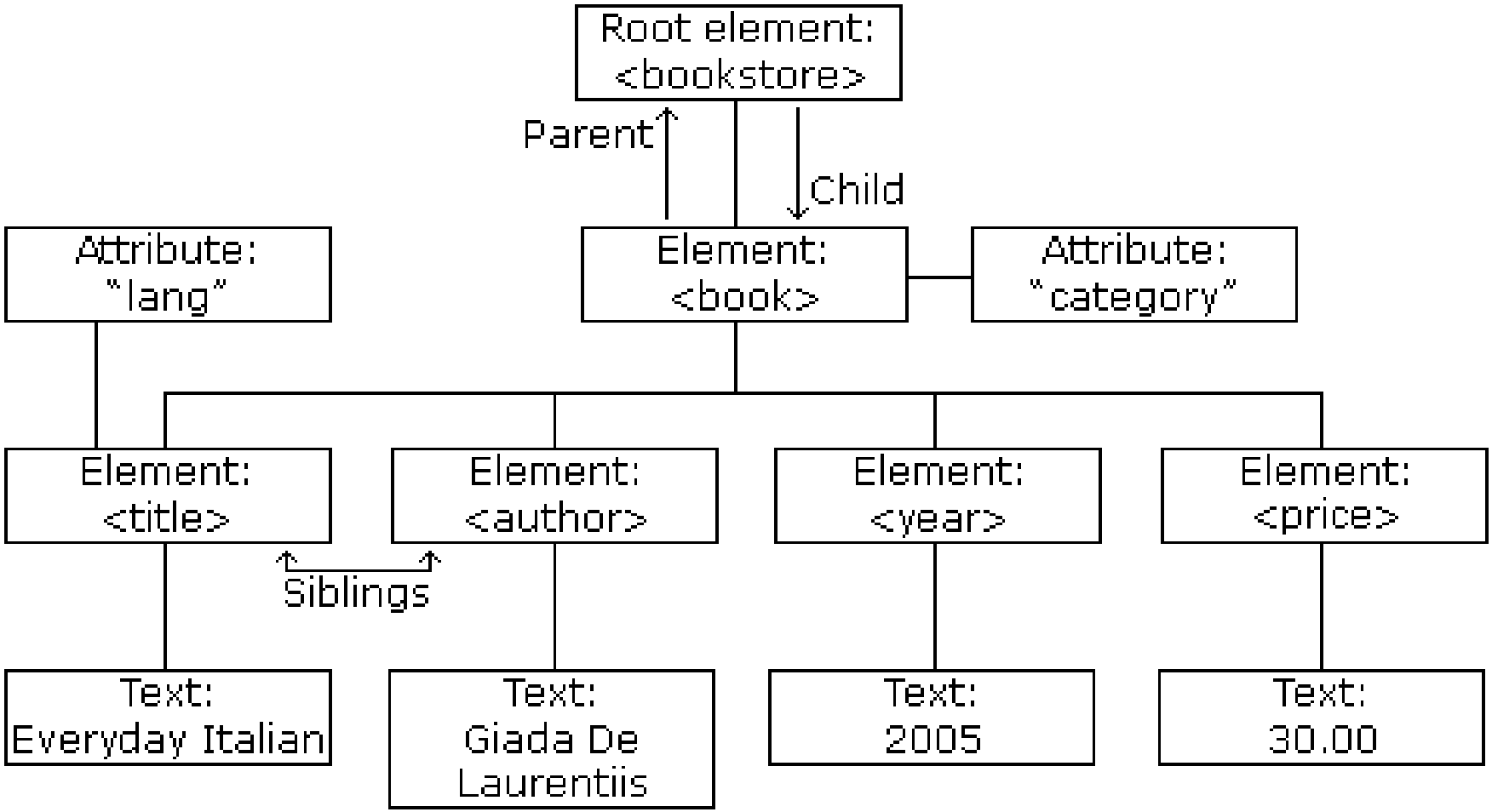
Feature	**XML**	**HTML**
Purpose	Data exchange and representation	Displaying content and structure of web pages
Structure	Hierarchical tree-like (nested elements)	Flexible, some self-closing elements
Data Definition	Extensible - define new elements/attributes	Predefined set of elements and attributes
Tags	Opening and closing tags required for each element (except some empty elements)	Some elements self-closing (e.g., ‘ ‘)
Validation	Often uses DTDs or XSDs for validation	Limited validation, relies on browsers
Focus	Structured data representation	Visual presentation and user interaction

The Difference Between XML and HTML

XML and HTML were designed with different goals :

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Tree Structure (1)



The terms parent, child, and sibling are used to describe the relationships between elements.

XML Tree Structure (2)

- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- All elements can have sub elements (child elements)

```
<root>
   <child>
     <subchild>.....</subchild>
   </child>
</root>
```

Components of an XML File

- Declaration : it is an optional component at the start of an XML file. It contains information about the current XML specification version
- Elements : it is the fundamental units of an XML document. They are contained in tags. Tags can comprise a start tag, an end tag, and the text inside
- Attributes : the term attributes refers to characteristics that an element can have. They are composed of a name and a value. These are contained within the start tag of a component

```xml
<?xml version="1.0" encoding="UTF-8"?> <!-- Prolog -->
<library> <!-- Root element -->
  <!-- Book entry with attributes -->
    <book id="101" genre="fiction">
      <title>Lost in Time</title> <!-- Title element -->
      <author>Emily Carter</author> <!-- Author element -->
      <price currency="USD">15.99</price> <!-- Price element
           with attribute -->
    </book>
</library>
```

XML Syntax Rules

- The XML Prolog (optional) : if it exists, it must come first in the document

```
<?xml version="1.0" encoding="UTF-8"?>
```

- XML documents must have a root element
- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must Always be Quoted
- XML Entity References : (< ; &lt ; less than, ...)
- Comments in XML : *<!- - This is a comment - ->*

Well Formed XML Documents

An XML document with correct syntax is called "Well Formed" :

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

DTD

- DTD stands for Document Type Definition
- A DTD defines the structure and the legal elements and attributes of an XML document
- DTD is a formal definition of the structure and elements in an XML document.
- DTDs are declared inside the XML document or in an external file.
- They provide a way to validate the correctness of an XML document.

Benefits of DTD

- Ensures the consistency and integrity of XML documents.
- Facilitates interoperability between different systems by enforcing a common structure.
- Helps in validating XML documents against predefined rules.
- Allows for automatic generation of parsers and validators.
- Provides documentation for understanding the structure of XML documents.

DTD Components

- **Elements** : the main building blocks
  `<!ELEMENT book (title, author+, chapter+)>`
- **Attributes** : provide extra information about elements
  `<!ATTLIST book id CDATA #REQUIRED>`
- **Entities** : Some characters have a special meaning in XML, like the less than sign (<)–> &lt
  `<!ENTITY copy "©">`
- **PCDATA** (Parsed Character DATA) : text found between the start tag and the end tag of an XML element
  `<!ELEMENT title (#PCDATA)>`
- **CDATA** (Character DATA) : can contain any text data
  `<!ATTLIST book id CDATA #REQUIRED>`

DTD Syntax Example

```
<!ELEMENT book (title, author+, (summary?, chapter*))>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (name)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT summary (#PCDATA)>
<!ELEMENT chapter (title, content)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT #PCDATA ANY>
<!ATTLIST book ISBN CDATA #REQUIRED>
```

XML DTD Declarations (1)

- `<!ELEMENT book (title, author+, (summary?, chapter*))>` : This line specifies that the "book" element must contain child elements for "title", one or more "author" elements, and optionally a "summary" element followed by zero or more "chapter" elements.
- `<!ELEMENT title (#PCDATA)>` : This line defines the structure of the "title" element. It specifies that the content of the "title" element is parsed character data (#PCDATA), meaning it can contain text.
- `<!ELEMENT author (name)>` : This line specifies that the "author" element must contain a single "name" element.
- `<!ELEMENT name (#PCDATA)>` : This line defines the structure of the "name" element, similar to the "title" element.

XML DTD Declarations (2)

- `<!ELEMENT summary (#PCDATA)>` : This line defines the structure of the "summary" element, specifying it can contain parsed character data.
- `<!ELEMENT chapter (title, content)>` : This line specifies that each "chapter" element must contain child elements for "title" and "content".
- `<!ELEMENT content (#PCDATA)>` : This line defines the structure of the "content" element, specifying it can contain parsed character data.

XML DTD Declarations (3)

- `<!ELEMENT #PCDATA ANY>` : This line defines that parsed character data (#PCDATA) can appear in any context, allowing text content in various places within the XML document.
- `<!ATTLIST book ISBN CDATA #REQUIRED>` : This line specifies an attribute named "ISBN" for the "book" element. The attribute type is CDATA, meaning it can contain any text data. The attribute is declared as #REQUIRED, meaning it must be present in each "book" element.

XML Schema

- XML Schema is a way to define the structure, content, and data types of XML documents.
- It provides a means to specify rules and constraints for elements and attributes.
- XML Schema is written in XML format and is used for validation and documentation purposes.
- It offers more features and flexibility compared to Document Type Definition (DTD).

Advantages of XML Schema over DTD

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

Benefits of XML Schema

- **Validation** : Ensures XML documents conform to defined rules and constraints.
- **Interoperability** : Facilitates data exchange between different systems.
- **Documentation** : Provides a clear and structured way to document XML structures.
- **Extensibility** : Supports the definition of complex data structures and types.
- **Namespace Support** : Allows for the definition of XML namespaces, avoiding naming conflicts.

Example of XML Schema - file : student.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="birthDate" type="xs:date"/>
        <xs:element name="speciality" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

File : students.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <students>
      <student>
          <firstName>Amine</firstName>
          <birthDate>1990-01-01</birthDate>
          <speciality>Computer Science</speciality>
          <email>john.doe@example.com</email>
      </student>
      <student>
          <firstName>Ahmed</firstName>
          <birthDate>1995-07-15</birthDate>
          <speciality>Mathematics</speciality>
          <email>jane.smith@example.com</email>
      </student>
  </students>
```

Validation : the process of checking an XML document against a set of rules to ensure it's well-formed and adheres to a specific structure

Validation (Python Example)

```python
from xmlschema import validate

schema_file = "student.xsd"
xml_file = "students.xml"
try:
    validate(schema_file)  # Validate the schema
    print("XML document is valid!")

except Exception as e:
    print("Validation Error:", e)
```

Introduction

- XSLT is a language for transforming XML documents.
- XPath is a language for navigating in XML documents.
- XQuery is a language for querying XML documents

XSL Components

- **XSLT** - a language for transforming XML documents
- **XPath** - a language for navigating in XML documents
- **XSL-FO** - a language for formatting XML documents (discontinued in 2013)
- **XQuery** - a language for querying XML documents

What is XSLT ?

- XSL (eXtensible Stylesheet Language) is a styling language for XML.
- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- With XSLT you can transform an XML document into various formats including HTML, PDF, ..etc
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

Displaying XML with XSLT

- XSLT is far more sophisticated than CSS
- With XSLT we can add/remove elements and attributes to or from the output file
- It allows for the rearrangement and sorting of elements
- It allows to perform tests and make decisions about which elements to hide and display
- Provides extensive capabilities for dynamic and conditional content manipulation

XSLT Transformation Process

1. Loading the Input XML Document and XSLT Stylesheet
2. Matching Templates
3. Applying Transformation Rules
4. Generating the Output

Matching Templates

- The XSLT processor navigates through the source tree starting from the root, and for each node, it finds a matching template in the XSLT stylesheet.
- If a match is found, the template is applied and part of the output tree (result tree) is generated.
- If no match is found, default template rules are applied. These default rules generally copy elements from the source document to the result document or recursively process child nodes

1. XML Input

```
<books>
  <book>
    <title>The Lord of the Rings</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book>
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
  </book>
</books>
```

This XSLT stylesheet will transform the book data into a simple HTML list

2. XSLT Stylesheet

```
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h1>My Book List</h1>
        <ul>
          <xsl:apply-templates select="books/book" />
        </ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="book">
    <li>
      <b><xsl:value-of select="title" /></b> by
            <i><xsl:value-of select="author" /></i>
    </li>
  </xsl:template>
</xsl:stylesheet>
```

3. Processing

The XSLT processor starts by reading both the XML and XSLT files

4. Transformation

- It uses the first template (<xsl :template match="/">) which matches the root element of the XML (books).
- Inside this template, it creates the basic HTML structure for the output document (including an <h1> and a <ul> for the list).
- It then uses xsl :apply-templates to process each child element of books (which are the book elements) using a separate template.
- The second template (<xsl :template match="book">) matches each individual book element.
- Inside this template, it creates an li element for each book in the list.
- It uses xsl :value-of to extract the values of the title and author elements and inserts them into the HTML output with formatting (bold for title, italic for author).

The final result will be an HTML document like this :

5. Output

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Book List</h1>
    <ul>
      <li>
        <b>The Lord of the Rings</b> by <i>J.R.R. Tolkien</i>
      </li>
      <li>
        <b>The Hitchhiker's Guide to the Galaxy</b> by
              <i>Douglas Adams</i>
      </li>
    </ul>
  </body>
</html>
```

XSLT : Commun Elements and Purposes

XSLT Element	Purpose
`<xsl:value-of>`	Extracts and outputs the value of a selected node or an XPath expression.
`<xsl:for-each>`	Iterates over a selected node-set and performs actions for each node in the set.
`<xsl:sort>`	Sorts the nodes selected by a `<xsl:for-each>` or `<xsl:apply-templates>` instruction based on specified criteria.
`<xsl:if>`	Conditionally applies a template or performs actions based on a specified condition.
`<xsl:choose>`	Provides multiple conditional branches to specify alternative actions based on different conditions. Contains `<xsl:when>` and `<xsl:otherwise>` elements.
`<xsl:apply-templates>`	Applies templates to nodes selected by an XPath expression, triggering template matching and processing.

Questions ?

University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development

Introduction to PHP

2024-04-08

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester: S4

Plan

Resources

Resources for individuals interested in mastering PHP programming:

- https://www.w3schools.com/php/php_syntax.asp
- https://www.php.net/manual/en/

What is PHP

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.n
- PHP is easy to learn and runs efficiently on the server side

PHP Installation

- Visual Studio Code (VS Code)
  - **Using XAMP**:
    https://www.youtube.com/watch?v=3xHrMwy-Y5A
  - **Using PHP Server**
    https://www.youtube.com/watch?v=-VcnwkR6QuQ
- Netbeans :
  https://www.youtube.com/watch?v=HvVRfHORHSg

Recommendation

XAMPP provides a convenient platform for running web servers locally, facilitating the setup and testing of web-based projects like WordPress sites.

PHP Formatter - VSCODE

- Install ***intelephense*** extension,
- Open **settings.json**
- add the value :

```
"[php]": {
    "editor.defaultFormatter":
            "bmewburn.vscode-intelephense-client"
}
```

Opening and Closing Tags

PHP code is enclosed within **<?php** and **?>** tags.

```php
<?php

// PHP code goes here

?>
```

The default file extension for PHP files is "**.php**"

Example

A simple **.php** file with both HTML code and PHP code:

```
<!DOCTYPE html>
<html>
<body>
  <h1>My first PHP page</h1>
```

```
  <?php
    echo "Hello World!";
  ?>
```

```
</body>
</html>
```

Comments

- Single-line comments: //
- Multi-line comments: /* */

```php
<?php
    // Single-line comment

    /*
      Multi-line
      comment
    */
?>
```

Variables

- Start with $ followed by the variable name
- Case-sensitive

```php
<?php
  $name = "Ahmed";
  $age = 12;
?>
```

Echoing Output

Print:

- Can only take one argument at a time
- Has a return value of 1 so it can be used in expressions

Echo:

- Can take multiple arguments (although such usage is rare)
- Generally faster than print
- Has no return value

```php
<?php
   echo "Hello, world!";   // no parentheses required
   print("Hello, world!"); // parentheses required
?>
```

Data Types

Strings, integers, floats, booleans, arrays, objects

```php
<?php
    $string = "Hello, world!";
    $integer = 42;
    $float = 3.14;
    $boolean = true;
    $array = array("apple", "banana", "cherry");
```

To verify the type of any object in PHP, use the **var_dump()**

Concatenation

Concatenate strings using the dot (.) operator

```php
<?php
  $greeting = "Hello";
  $name = "Ahmed";
  echo $greeting . " " . $name;
  // Outputs: Hello Ahmed
?>
```

Double or Single Quotes?

Double Quotes ("):

- Variables within double quotes are parsed and their values are inserted into the string.
- Escape sequences like \n, \t, etc., are interpreted as newline, tab, etc

Single Quotes ('):

- Returns the string like it was written (literally)

```php
<?php
  $name = "Ahmed";
  echo "Hello, $name!"; // Output: Hello, Ahmed!

  $name = "Ahmed";
  echo 'Hello, $name!'; // Output: Hello, $name!
```

String Functions

- **strlen()**: Returns the length of a string
- **str_word_count()**: counts the number of words in a string
- **strpos()**: Finds the position of the first occurrence of a substring
- **substr()**: Returns a part of a string
- **str_replace()**: Replaces all occurrences of a substring with another substring
- **strtolower()** / **strtoupper()**: Converts a string to lowercase / uppercase.
- **trim()**: Removes whitespace from the beginning and end of a string
- **explode()**: Splits a string into an array by a specified delimiter.
- **implode()** / **join()**: Joins array elements with a string.

PHP Numbers

- **Integers**: Whole numbers (e.g., 42).
  - PHP_INT_MAX - The largest integer supported
  - PHP_INT_MIN - The smallest integer supported
  - PHP_INT_SIZE - The size of an integer in bytes
  - is_int($var): Checks for true integer (int data type).
- **Floats**: Decimals (e.g., 3.14) or scientific notation.
  - PHP_FLOAT_MAX - The largest floating number
  - PHP_FLOAT_MIN - The smallest floating number
  - PHP_FLOAT_DIG - The number of decimal digits that can be rounded into a float and back without precision loss
  - PHP_FLOAT_EPSILON - The smallest representable positive number x, so that x + 1.0 != 1.0
- **is_numeric($var)**: Checks for any numeric value (number or numeric string).

Change Data Type

- **(string)** - Converts to data type String
- **(int)** - Converts to data type Integer
- **(float)** - Converts to data type Float
- **(bool)** - Converts to data type Boolean
- **(array)** - Converts to data type Array
- **(object)** - Converts to data type Object
- **(unset)** - Converts to data type NULL

Example - Casting

```php
<?php
  $floatNum = 3.14;
  $stringNum = "42";
  $boolValue = true;
  $castToInt = (int)$floatNum; // Cast float to integer
  $castToIntFromString = (int)$stringNum; //string to integer
  $castToFloat = (float)$stringNum; // Cast string to float
  $castToString = (string)$boolValue; // Cast boolean to string
  echo "Original Float: $floatNum, Cast to Integer:
        $castToInt<br>";
  echo "Original String: $stringNum, Cast to Integer:
        $castToIntFromString<br>";
  echo "Original String: $stringNum, Cast to Float:
        $castToFloat<br>";
  echo "Original Boolean: $boolValue, Cast to String:
        $castToString<br>";
```

PHP Operators

PHP offers a rich set of operators, categorized based on their function:

- Arithmetic operators: Perform mathematical calculations (+, -, *, /, %)
- Assignment operators: Assign values to variables (=, +=, -=, *=, /=).
- Comparison operators: Compare values (==, !=, <, >, <=, >=)
- Logical operators: Perform logical operations (&&, ||, !)
- Increment/decrement operators: Increase/decrease variable values(++, - -).
- String operators: PHP has two operators(Concatenation, Concatenation assignment)
- Array operators : are used to compare arrays
- Conditional assignment operators: set a value depending on conditions

Arithmetic Operators in PHP

Op.	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

Op.	Name	Example	Description
=	Assignment	$x = $y	Assigns the value of $y to $x.
+=	Addition ass.	$x += $y	Adds $y to $x and stores the result in $x.
-=	Subtraction ass.	$x -= $y	Subtracts $y from $x and stores the result in $x.
*=	Multiplication ass.	$x *= $y	Multiplies $x by $y and stores the result in $x.
/=	Division ass.	$x /= $y	Divides $x by $y and stores the result in $x.
%=	Modulus ass.	$x %= $y	Computes the modulus of $x divided by $y and stores the result in $x.
**=	Exponentiation ass.	$x **= $y	Raises $x to the power of $y and stores the result in $x.
.=	Concatenation ass.	$x .= $y	Concatenates $y to $x and stores the result in $x.

PHP Comparison Operators

Op.	Name	Example	Description
==	Equal	$x == $y	Returns true if $x is equal to $y.
!=	Not equal	$x != $y	Returns true if $x is not equal to $y.
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type.
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type.
>	Greater than	$x > $y	Returns true if $x is greater than $y.
<	Less than	$x < $y	Returns true if $x is less than $y.
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y.
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y.
<=>	Spaceship	$x <=> $y	Returns -1 if $x is less than $y, 0 if they are equal, and 1 if $x is greater than $y. Introduced in PHP 7.

PHP Increment/Decrement Operators

Op.	Same as...	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

Op.	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

Op.	Name	Example	Description
.	Concatenation	$x . $y	Concatenates $x and $y
.=	Concatenation assignment	$x .= $y	Appends $y to $x

PHP Array Operators

Op.	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Ret. true if $x is not equal to $y
<>	Inequality	$x <> $y	Ret. true if $x is not equal to $y
!==	Non-identity	$x !== $y	Ret. true if $x is not identical to $y

PHP Conditional Assignment Operators

Op.	Name	Example	Result
?:	Ternary	$x = expr1 ? expr2 : expr3	Returns the value of $x. The value of $x is expr2 if expr1 is TRUE. The value of $x is expr3 if expr1 is FALSE.
??	Null coalescing	$x = expr1 ?? expr2	Returns the value of $x. The value of $x is expr1 if expr1 exists and is not NULL. If expr1 does not exist or is NULL, the value of $x is expr2. Introduced in PHP 7.

Conditional Statements - if

```php
<?php
  if (condition) {
    // Code to execute if condition is true
  } elseif (another_condition) {
    // Code to execute if another_condition is true
  } else {
    // Code to execute if none of the above conditions are
          true
  }
?>
```

Conditional Statements - Switch

```php
<?php
  switch (expression) {
    case value1:
        // Code to execute if expression equals value1
       break;
    case value2:
       // Code to execute if expression equals value2
       break;
    default:
     // Code to execute if expression doesn't match any case
  }
?>
```

While Loop

While Loop :

```php
<?php
    while (condition) {
      // Code to execute while condition is true
    }
?>
```

Do-While Loop

```php
<?php
   do {
  //Code to execute at least once, then while condition is true
   } while (condition);
?>
```

For Loop

For Loop

```php
<?php
  for ($i = 0; $i < count($array); $i++) {
    // Code to execute for each iteration
  }
?>
```

Foreach Loop

```php
<?php
  foreach ($array as $key => $value) {
    // Code to execute for each element in the array
  }
```

example - foreach

```php
<?php
  // Define an array of student names
  $students = array("John", "Alice", "Bob", "Emily");

  // Iterate over the array using foreach loop
  echo "<h2>List of Students:</h2>";
  echo "<ul>";
      foreach ($students as $student) {
        echo "<li>$student</li>";
      }
  echo "</ul>";
```

Break Statement

- Used to exit a loop prematurely.
- Terminates the current loop iteration and resumes execution at the next statement after the loop.
- Useful for exiting a loop when a certain condition is met.

```php
<?php
 for ($i = 1; $i <= 10; $i++) {
   if ($i == 5) {
     break; // Exit the loop when $i equals 5
   }
   echo $i . " ";
 }
 //output : 1 2 3 4
?>
```

Continue Statement

- Used to skip the current iteration of a loop.
- Skips the remaining code inside the loop for the current iteration and continues with the next iteration.
- Useful for skipping certain iterations based on a condition.

```php
<?php
   for ($i = 1; $i <= 10; $i++) {
     if ($i % 2 == 0) {
       continue; // Skip even numbers
     }
     echo $i . " ";
   }
   //Output : 1 3 5 7 9
```

HTML Form Structure

- **HTML <form> tag**: Defines the form to collect user input.
- **Action attribute**: Specifies the URL of the PHP script that will handle the form data.
- **Method attribute**: Defines how form data should be transmitted to the server (GET or POST)

```
<form action="process.php" method="post">
            ....
</form>
```

Method: Get and Post

- GET method: Sends form data in the URL query string.
- POST method: Sends form data in the HTTP request body.
- Use POST for sensitive or large data, as it's more secure and has no size limitations.

GET vs. POST Methods

Feature	GET	POST
Data Placement	URL (after ?)	HTTP request body
Security	Less secure (visible in URL)	More secure (hidden)
Data Size Limit	Smaller (limited by URL length)	Larger
Example Use Cases	• Search queries • Pagination • Sorting data	• Form submissions • Login credentials • File uploads

Collect form data

- **$_GET** is an array of variables passed to the current script via the URL parameters.
- **$_POST** is an array of variables passed to the current script via the HTTP POST method.

```
<form action="process.php" method="post">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  <button type="submit">Submit</button>
</form>
```

```
<?php    // file : process.php
    $username = $_POST['username'];
     // Process $username...
```

Example

```
<form action="process.php" method="post">
   <label for="username">Username:</label>
   <input type="text" id="username" name="username" required>
   <label for="password">Password:</label>
   <input type="password" id="password" name="password" required>
   <button type="submit">Login</button>
</form>
```

```
<?php     // file : process.php
  if ($_SERVER["REQUEST_METHOD"] == "GET") {
     $username = $_POST['username'];
     $password = $_POST['password'];
     if ($username === "isil" && $password === "isil") {
       header("Location: /profile.html");// Redirect to profile.html
       exit; // Make sure to exit after redirection
     } else
       { echo "Invalid username or password. Please try again.";}
  }
```

Common $_SERVER Variables

- **$_SERVER["SERVER_ADDR"]**: IP address of the server hosting the script.
- **$_SERVER["SERVER_SOFTWARE"]**: Server software name and version (e.g., Apache, Nginx).
- **$_SERVER["SERVER_NAME"]**: Hostname of the server
- **$_SERVER["DOCUMENT_ROOT"]**: Root directory of the document web server.
- **$_SERVER["REQUEST_METHOD"]**: HTTP method used to submit the request (e.g., GET, POST).
- **$_SERVER["REQUEST_URI"]**: URI path of the requested resource.
- **$_SERVER["REMOTE_ADDR"]**: IP address of the user's machine.
- **$_SERVER["HTTP_REFERER"]**: URL of the referring page (if the user came from another page).
- **$_SERVER["SCRIPT_FILENAME"]**: Absolute path to the currently executing script.

$_SERVER["PHP_SELF"] variable

- **$_SERVER["PHP_SELF"]**: a super global variable that returns the filename of the currently executing script.
- **Warning** : The $_SERVER["PHP_SELF"] variable can be used by hackers!
- **Solution** : $_SERVER["PHP_SELF"] exploits can be avoided by using the ***htmlspecialchars()*** function :

```
htmlspecialchars($_SERVER["PHP_SELF"])
```

- **For more details**: https://www.w3schools.com/php/php_form_validation.asp

Information

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications.

XSS enables attackers to inject client-side script into Web pages viewed by other users

Example of hacking a page

- Create file : test_form.php

```
 <form action="<?php echo $_SERVER["PHP_SELF"]; ?>" method="post">
  <label for="name">Name:</label>
  <input type="text" name="name" id="name">
  <button type="submit">Submit</button>
</form>
```

```
<?php
 if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $name = $_POST["name"];
   echo "<p>Hello, " . $name . "!</p>";
 }
?>
```

- Click on submit, then add to the URL:
  /"><script>alert("XSS Attack!");</script>
- Modify test_form.php, and try to hack the page:
  htmlspecialchars($_SERVER["PHP_SELF"]);

Validate Form Data With PHP

- Pass all variables through PHP's htmlspecialchars()
- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
- Remove backslashes \ from the user input data (with the PHP stripslashes() function)
- Create a function that will do all the checking :

```php
<?php
    function test_input($data) {
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
    }
?>
```

Example - Validate Form Data

```php
<?php
  // define variables and set to empty values
  $name = $email = $gender = $comment = $website = "";
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
  }
  function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
  }
?>
```

Common PHP Form Vulnerabilities (1)

- **XSS (Cross-Site Scripting)**: Unsanitized user input leading to execution of malicious scripts.
- **SQL Injection**: Improperly sanitized form inputs allowing unauthorized SQL commands.
- **CSRF (Cross-Site Request Forgery)**: Lack of CSRF tokens enabling actions on behalf of users.
- **File Upload Vulnerabilities**: Inadequate validation of uploaded files permitting execution of malicious scripts.

Common PHP Form Vulnerabilities (2)

- **Form Spoofing**: Deceptive creation of fake forms to capture sensitive information.
- **Parameter Tampering**: Manipulation of form parameters leading to unauthorized access or data manipulation.
- **Session Fixation**: Poor session management enabling hijacking of user sessions.
- **Denial of Service (DoS)**: Lack of input validation or rate limiting making the server vulnerable to flooding attacks.

Homework Assignment: PHP Form Validation

Objective:

- Make input fields required in your PHP form.
- Create error messages if required fields are left empty.

Instructions:

- Use the `required` attribute in HTML input fields
- Implement PHP validation to check for empty fields
- Display error messages next to the input fields if they are left empty
- Ensure to implement validation and sanitization to avoid vulnerabilities
- Use CSS to style the error messages for better visibility.

Helpful Resource:

- https://www.w3schools.com/php/php_form_required.asp

Classes and Objects

- Classes and objects are the two main aspects of object-oriented programming
- A class is a template for objects, and an object is an instance of a class
- When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties

Define a Class

```php
<?php
  class ClassName {
    // Properties (attributes)
    // Methods (functions)
  }
?>
```

Example Class Definition

```php
<?php
  class Person {
    private $name;
    private $age;

    public function __construct($name, $age) { // Constructor
      $this->name = $name;
      $this->age = $age;
    }
    public function getName() {// Getter for name
      return $this->name;
    }
    public function setName($name) {// Setter for name
      $this->name = $name;
    }
    public function getAge() {// Getter for age
      return $this->age;
    }
    public function setAge($age) { // Setter for age
      $this->age = $age;
    }
  }
//....
```

Constructors

- A constructor is a special method in a PHP class.
- It's automatically called when you create a new object
- Its purpose is to initialize the object's state (properties)
- It can accept arguments to customize the initial state.
- The constructor name must match the class name.

```php
<?php
  class Fruit {
   public $name;
   public $color;
   function __construct($name, $color) {
     $this->name = $name;
     $this->color = $color;
   }
  }
  $apple = new Fruit("Apple", "red");
```

Destructors

- A destructor is a special method named `__destruct` in a PHP class.
- It's automatically called when an object is destroyed (goes out of scope).
- Its purpose is to perform any cleanup tasks before the object is removed from memory.
- This might involve closing database connections, releasing resources, or logging information
- If you create a `__destruct()` function, PHP will automatically call this function at the end of the script

Destructor Example

```php
<?php
 class DatabaseConnection {
   private $connection;
   function __construct($host, $username, $password, $database) {
     $this->connection = new mysqli($host, $username, $password, $database);
     if ($this->connection->connect_error)
       {die("Connection failed: " . $this->connection->connect_error);}
   }
   function __destruct() {
     if (isset($this->connection)) {
       $this->connection->close();
       echo "Database connection closed.\n";
     }
   }
 }
 $db = new DatabaseConnection("localhost", "myuser", "mypassword",
         "mydb");
 // When $db goes out of scope or is explicitly destroyed, the destructor
         closes the connection
 //isset() :  check if a variable is set and is not null
 //die() :  immediately terminate script execution and display an optional
         message
```

Access Modifiers in PHP

- **public**: Accessible everywhere.
- **protected**: Accessible within class and subclasses.
- **private**: Accessible only within the class.

Creating Objects in PHP

Syntax for creating objects in PHP

```php
<?php
    $objectName = new ClassName();
 ?>
```

Example : Create an object of Person

```php
<?php
    $person1 = new Person("Alice", 30);
    echo $person1->getAge();
?>
```

Inheritance

- Inheritance is a way to create a new class from an existing class
- The new class inherits properties and methods from the existing class
- It is a way to reuse code and establish a hierarchical relationship

Why Use Inheritance?

- Code Reusability
- Reduced Redundancy
- Improved Maintainability
- Hierarchical Relationships

```php
<?php
class ClassName {
    // Class properties and methods go here
}

class ChildClass extends ParentClass {
  // Additional or overridden properties and methods
}
```

Example - Class Person

```php
<?php
  class Person {
   public $name;
   public $age;

   public function __construct($name, $age) {
     $this->name = $name;
     $this->age = $age;
   }

   public function displayInfo() {
     echo "Name: " . $this->name . ", Age: " . $this->age;
   }
  }
```

Example - Class Employee

```php
<?php
class Employee extends Person {
   public $salary;
     public $department;

  public function __construct($name, $age, $salary, $dprtmnt){
    parent::__construct($name, $age);
    $this->salary = $salary;
    $this->department = $dprtmnt;
  }
  public function displayInfo($showSalary = false) {
    parent::displayInfo();
    if ($showSalary) {
      echo ", Salary: " . $this->salary;
    }
    echo ", Department: " . $this->dprtmnt;
  }
 }
```

Example - Create Object

```php
<?php
$person = new Person("John Doe", 30);

$person->displayInfo();
// Output: Name: John Doe, Age: 30

$employee = new Employee("Jane Smith", 35, 50000, "Marketing");

$employee->displayInfo();
/* Output: Name: Jane Smith, Age: 35, Department: Marketing*/

$employee->displayInfo(true);
/* Output: Name: Jane Smith, Age: 35, Salary: 50000,
      Department: Marketing  */
```

Common Error Types in PHP

- **Syntax Errors:** Typos, missing semicolons, mismatched parentheses (e.g., `$variable = 5` instead of `$variable = 5;`)
- **Runtime Errors:** Occur during execution (e.g., division by zero, file not found)
- **Logic Errors:** Flaws in program logic (e.g., incorrect calculation, infinite loop)
- **Database Errors:** Issues with database connections, queries (e.g., invalid SQL syntax)

Basic Error Handling: Using the die() function

- The die() function displays a message and immediately terminates script execution.
- Use die() for critical errors where continued execution is pointless.

```php
<?php
  $connection = mysqli_connect("host", "user", "password",
        "database");
  if (!$connection) {
    die("Connection failed: " . mysqli_connect_error());
  }
  echo "Connected successfully";
?>
```

However, simply stopping the script is not always the right way to go

Creating a Custom Error Handler

- Create a special function that can be called when an error occurs

```
error_function(error_level,error_message,
               error_file,error_line,error_context)
```

Parameters :

- **error_level**: Required. The level of the error raised.
- **error_message**: Required. The error message.
- **error_file**: Optional. The filename in which the error was raised.
- **error_line**: Optional. The line number in which the error was raised.
- **error_context**: Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Example - Custom Error Handler

```php
<?php
function myErrorHandler($severity, $message, $file, $line) {
  throw new ErrorException($message, 0, $severity, $file,
        $line);
}

set_error_handler("myErrorHandler");
```

Essentials of Exception Handling in PHP

Proper exception code should include:

- **try** - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
- **throw** - This is how you trigger an exception. Each "throw" must have at least one "catch"
- **catch** - A "catch" block retrieves an exception and creates an object containing the exception information

```php
<?php
try {
  // Code that may throw an error or exception
} catch (Throwable $t) {
  // Handle error or exception
}
```

Example - Exception

```php
<?php

try {
  // Risky code (might throw an error)
  $result = 10 / 0; //  DivisionByZeroError
} catch (Throwable $t) {
  echo "Something went wrong: " . $t->getMessage();
}

echo "Script continues...";
?>
```

File management in PHP

```php
<?php
$file = fopen("example.txt", "r"); // Opening a File

//Reading a File
$content = fread($file, filesize("example.txt"));

fwrite($file, "Hello, World!");//Writing to a File

fclose($file); //Closing a File
```

PHP also provides functions to copy (**copy()**), move (**rename()**), delete (**unlink()**) files, and check if a file exists (**file_exists()**).

The opendir() function can be used to open a directory, and readdir() to read its content

Remote Files

Ensure the **allow_url_fopen** directive is set to **On** in your php.ini file:

php.ini

```
allow_url_fopen = On
```

Reading Remote Files

using **file_get_contents()**:

```
    <?php
$content =
      file_get_contents('http://www.example.com/somefile.txt');
echo $content;
```

Or with **fopen()**

```
    <?php
    $file = fopen("http://www.example.com/somefile.txt", "r");
    if ($file) {
      while (!feof($file)) {
        $line = fgets($file);
        echo $line;
      }
      fclose($file);
    }
```

Writing to Remote Files

Using **cURL** to Send Data

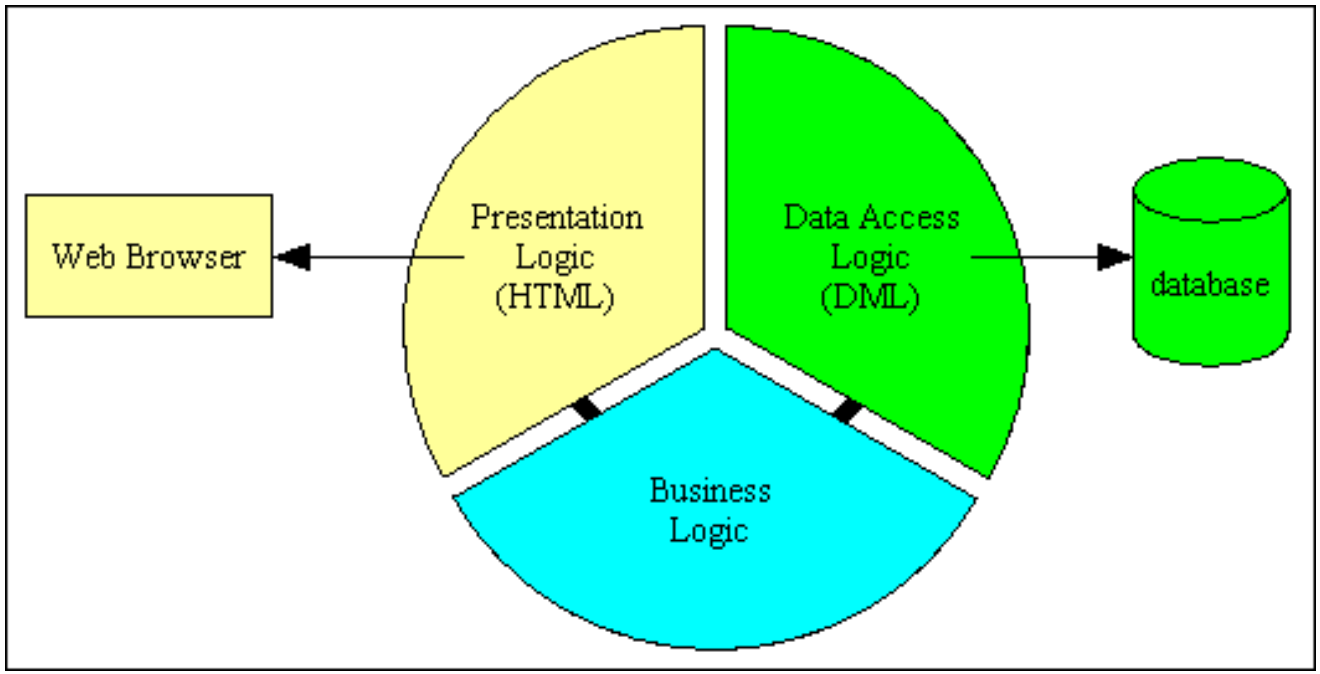
```php
    <?php
$ch = curl_init();

$data = array('key' => 'value');
curl_setopt($ch, CURLOPT_URL,
      "http://www.example.com/receive.php");
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($data));

$response = curl_exec($ch);
curl_close($ch);
```

Security: When enabling **allow_url_fopen** and handling remote files:

Web 3-tier Architecture in PHP

Web 3-tier architecture is a design pattern for developing scalable and maintainable web applications. It divides the application into three layers:

- Presentation Layer
- Business Logic Layer
- Data Access Layer



Presentation Layer

The front-end part of the application. Concerned with how the application looks and interacts with users

- Receives HTTP requests
- Handles user interactions
- Implemented using HTML, CSS, JavaScript, PHP
- Components: web pages, forms, UI

Business Logic Layer

The heart of the application

- Classes and methods that handle the processing of data, such as user authentication, data validation, and business rules.
- Communicates with the presentation layer to send data to be displayed and interacts with the data layer to retrieve, manipulate, and store data

Data Access Layer

This bottom layer manages the storage and retrieval of data

- Interacts with data storage
- Handles database operations
- Implemented using PDO, MySQLi
- Components: DAOs(Data Access Objects), repositories

Questions ?

University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development
PHP & MySQL Database

2024-05-12

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester: S4

Plan

PHP and MySQL

PHP can work with a MySQL database using:

- MySQLi extension (the "i" stands for improved) :only work with MySQL databases
  - MySQLi Procedural: In the procedural approach, you use functions to execute queries and manage connections
  - MySQLi Object-oriented (OO): offers better code organization and reusability,
- PDO (PHP Data Objects) : work on 12 different database systems

Choose MySQLi for dedicated MySQL projects if performance is crucial, or PDO for portability across different databases.

Create a new database

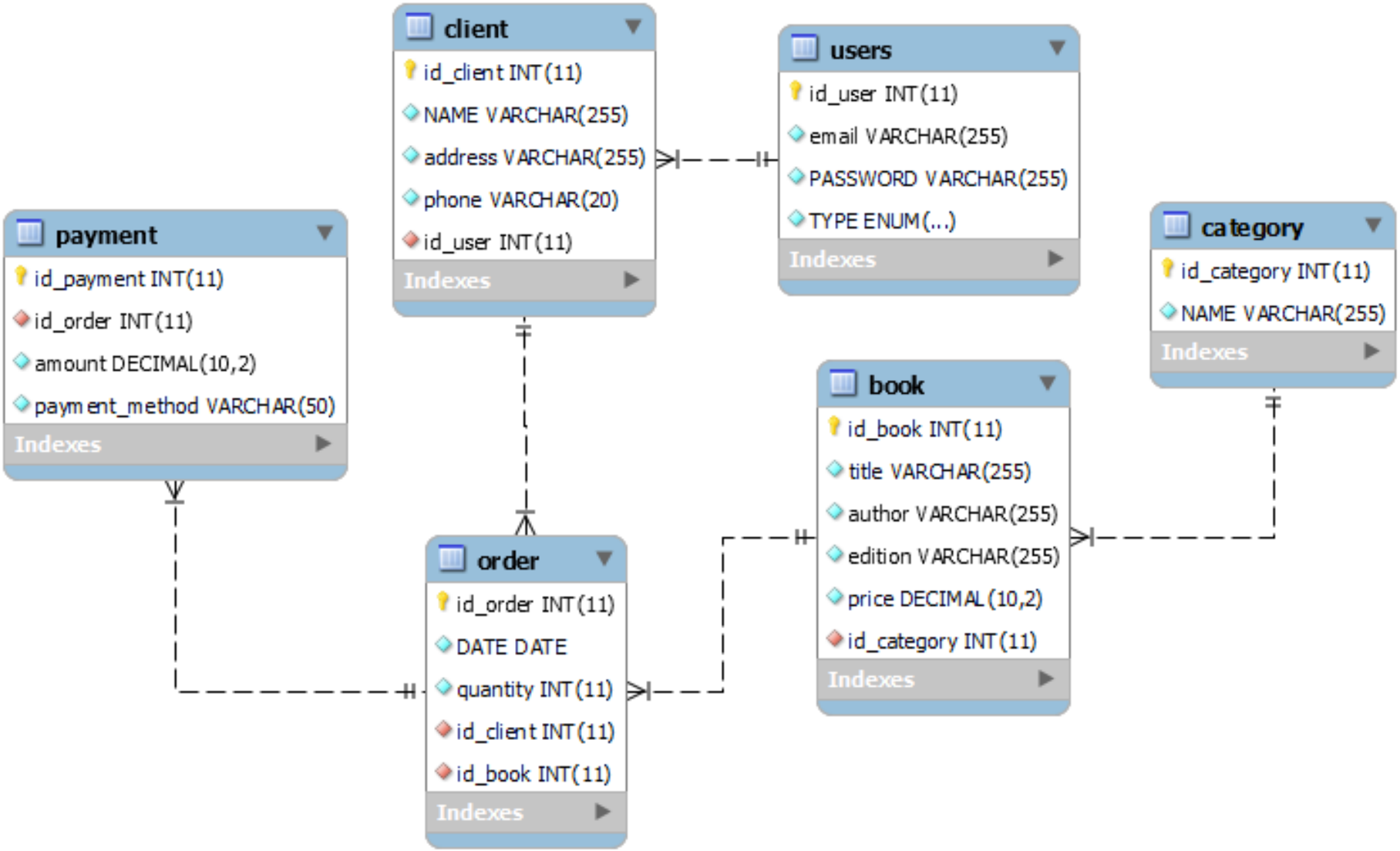
- Start XAMPP: Locate the XAMPP control panel. This is usually an icon in your system tray or a shortcut on your desktop.

  Click the "Start" buttons for both "Apache" and "MySQL" modules. The buttons will turn green when the modules are running.

- Access phpMyAdmin: Open your web browser and navigate to the following URL: http://localhost/phpmyadmin/

- Create a new database (you can create a Database /table using SQL)

Conceptual Treatment Model



```sql
-- Create the database if it doesn't exist
CREATE DATABASE IF NOT EXISTS bookdb;
-- Use the bookdb database
USE bookdb;

CREATE TABLE IF NOT EXISTS Users (
    id_user INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    type ENUM('client', 'vendor') NOT NULL
);

CREATE TABLE IF NOT EXISTS Client (
    id_client INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    id_user INT NOT NULL
);
```

```sql
CREATE TABLE IF NOT EXISTS Book (
     id_book INT AUTO_INCREMENT PRIMARY KEY,
     title VARCHAR(255) NOT NULL,
     author VARCHAR(255) NOT NULL,
     edition VARCHAR(255) NOT NULL,
     price DECIMAL(10,2) NOT NULL,
     id_category INT NOT NULL
 );
CREATE TABLE IF NOT EXISTS Category (
     id_category INT AUTO_INCREMENT PRIMARY KEY,
     name VARCHAR(255) NOT NULL
 );
CREATE TABLE IF NOT EXISTS `Order` (
     id_order INT AUTO_INCREMENT PRIMARY KEY,
     date DATE NOT NULL,
     quantity INT NOT NULL,
     id_client INT NOT NULL,
     id_book INT NOT NULL,
     id_vendor INT NOT NULL
 );
```

```
  CREATE TABLE IF NOT EXISTS Payment (
         id_payment INT AUTO_INCREMENT PRIMARY KEY,
         id_order INT NOT NULL,
         amount DECIMAL(10,2) NOT NULL,
         payment_method VARCHAR(50) NOT NULL
  );
 ALTER TABLE Client
  ADD FOREIGN KEY (id_user) REFERENCES Users(id_user);

 ALTER TABLE Book
  ADD FOREIGN KEY (id_category) REFERENCES
        Category(id_category);

ALTER TABLE `Order`
  ADD FOREIGN KEY (id_client) REFERENCES Client(id_client),
  ADD FOREIGN KEY (id_book) REFERENCES Book(id_book),
  ADD FOREIGN KEY (id_vendor) REFERENCES Users(id_user);

ALTER TABLE Payment
  ADD FOREIGN KEY (id_order) REFERENCES `Order`(id_order);
```

Connect to MySQL (MySQLi Object-Oriented)

```
    <?php
<?php
  $servername = "localhost";
  $username = "username";
  $password = "password";

  // Create connection
  $conn = new mysqli($servername, $username, $password);

  // Check connection
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
  }
  echo "Connected successfully";
?>
```

Connection using Class

```php
<?php     //File : config.php
class Database
{   private $servername = "localhost";   private $username = "username";
    private  $password = "password";     private   $dbname = "dbname";
    private $conn = null;
  public function connect()  // Connect to the database
  {    if ($this->conn == null) {
       $this->conn = new mysqli($this->servername, $this->username,
                $this->password, $this->dbname);

       if ($this->conn->connect_error) // Check connection
         { die("Connection failed: " . $this->conn->connect_error); }
       echo "Connected successfully <br><br>";
    }
    return $this->conn;
  }
  public function disconnect() // Disconnect from the database
  {   if ($this->conn != null) {
       $this->conn->close();
       $this->conn = null;
       echo "<br><br>Disconnected successfully";
    }
  }
}
```

Connect to MySQL : Calling the Class

```php
<?php
require_once 'config.php';

$db = new Database();
$conn = $db->connect(); // Connect to the database

$sql = "SELECT userName, password FROM user";
$result = $conn->query($sql);

//...

$db->disconnect(); // Close the database connection
```

Persistent Database Connections(1)

- Persistent database connections maintain an open connection between the application and the database even after executing a query or a series of queries.
- Unlike regular connections, persistent connections remain open for a defined period or until explicitly closed by the application.
- Benefits include reduced server overhead by avoiding repeated connection establishment and reusability of connections for multiple successive queries.

Persistent Database Connections(2)

- Careful resource management is crucial to prevent exhausting server resources.
- Server-side configuration is necessary to enable and optimize persistent connections.
- Persistent connections are suitable for high-traffic applications where connection time is critical and numerous queries need to be executed.
- However, their usage should be judiciously evaluated based on specific application needs and overall performance considerations.

Example

```php
<?php
$server = 'localhost';
$username = 'root';
$password = '';
$database = 'test';

// Establishing a persistent database connection with MySQLi
$conn = new mysqli("p:$server", $username, $password,
       $database);

if ($conn->connect_error) {  // Checking the connection
  die("Connection failed: " . $conn->connect_error);
}

  // Queries and processing with the persistent connection

$conn->close(); // Closing the connection
```

Questions ?