How Website Works?

A **website works** by using the client-server model: when you type a website address into your browser (the client), your browser finds the server that hosts the website, requests the site's files, and displays them on your screen.

Here's how the process works step-by-step:

- You enter a website address (URL) in your browser.
- The browser asks a DNS server to translate the website name into an IP address (the server's numeric address).
- The browser sends an HTTP request to the web server at that IP address, asking for the website's files.
- The server responds, sending the website's files (HTML, CSS, JavaScript, images) back to the browser.
- Your browser assembles these files and renders the website on your screen for you to interact with.

HTML, **CSS**, and **JavaScript** are the main languages for structuring, styling, and adding interactivity to websites. Static sites show fixed content, while dynamic sites can change based on user actions or server data.

In summary: a website is a collection of files stored on a server, and it appears in your browser by requesting, transferring, and displaying those files through web technologies like HTTP and HTML.

History of HTML -

HTML (**HyperText Markup Language**) is the foundational language of the World Wide Web, enabling browsers to display web pages and structure information for internet users. The evolution of HTML is deeply connected to both technological advances and web standards.

Origins and Early Development

- **1980:** Tim Berners-Lee, a physicist at CERN, started working on systems to help researchers share documents. The concepts ultimately led to hypertext, influenced by earlier ideas of non-linear text navigation.
- **1991:** Berners-Lee created the first version of HTML and published the initial description of HTML tags online, consisting of only **18 elements**. These tags allowed basic structuring—headings, paragraphs, lists, links, and images.
- **1993:** HTML 1.0 was formally released, making it possible to organize a web page in a user-friendly manner. The first public specification documented the markup language.

Version Timeline and Milestones

Year	Version	Highlights
1991	HTML 1.0	Very basic, only 18 tags, used for simple document sharing.
1995	HTML 2.0	Added more features, standardized by the IETF, gave a firmer foundation for web development.
1997	HTML 3.2	Introduced new tags and improved formatting. Extension attempts with HTML 3.0 were abandoned.
1999	HTML 4.01	Became widely used, standardized by the W3C, offering significant improvements like support for CSS and scripting.
2000	XHTML 1.0	XML-based version for stricter syntax and interoperability across systems.
2012-2014	HTML5	Extensive update: multimedia (audio, video), semantic elements, canvas, responsive design.

Modern HTML

HTML5 is the current standard and introduced a host of features:

- Native multimedia support (audio, video).
- Graphics and game development with <canvas>.

- Rich semantic elements (<header>, <footer>, <article>, <section>).
- Enhanced form controls and APIs for web apps.
- Continues to be extended collaboratively by groups like W3C and WHATWG, with a focus on a **“Living Standard”** that evolves over time.

Key Contributors and Standardization

- **Tim Berners-Lee** (CERN): Creator of HTML and the World Wide Web.
- **Dave Raggett**: Promoted richer HTML features and standards.
- **W3C** (World Wide Web Consortium): Oversees standards and maintains development since 1994.
- **WHATWG**: Drives ongoing development and the “Living Standard” approach to HTML.

Summary

HTML's history is a story of collaboration, innovation, and the growing needs of the web. From a handful of tags to hundreds, from static documents to interactive web applications, HTML remains the backbone of the internet.

How browsers determine the language of an HTML document?

Browsers determine the **language of an HTML document primarily using the lang attribute of the <html> element**. When you specify <html lang="en">, for instance, the browser recognizes English as the document’s main language. This helps browsers, screen readers, and translation tools render content, pronounce words correctly, and adapt UI components.

- The lang attribute can also be used on *inline or block elements* inside the document to declare sections written in a different language, improving accessibility and accurate pronunciation for assistive technologies.
- If the lang attribute is omitted, browsers and screen readers may default to the user's browser or operating system language settings, but this can lead to incorrect rendering or pronunciation, especially for multilingual content.
- Browsers may also refer to additional sources, such as HTTP headers (Content-Language), or use new client-side APIs (like Chrome’s Language Detector) to analyze content text and infer its probable language, but these are less common for simple HTML documents.
- For users, browsers expose their own *preferred interface languages* via the navigator.language and navigator.languages JavaScript properties, which web applications can query to adapt content or UI but are separate from document language.

In summary: **the standard way for a browser to recognize the language of an HTML page is via the lang attribute on the <html> tag; lacking it, browsers may guess from user/device settings or use AI APIs for deeper content analysis**.

<html lang=”hi”> --- for hindi

<html lang=”ko”> --- for korean

<html lang=”fr”> --- for france

head and body tags defference -

The **HTML <head> and <body> tags** serve separate and distinctive roles in structuring a web page:

- The **<head> tag** contains *metadata* and instructions for the browser, such as the page title, links to CSS files, character encoding, meta descriptions, scripts, and other resources. Content inside <head> is largely *not visible* to users and is used behind the scenes for page setup, SEO, styling, and browser configuration.
- The **<body> tag** holds all *visible content*—everything users interact with on the page, like headings, paragraphs, images, tables, forms, and links. Whatever is placed inside <body> renders physically on the browser window.

Key Differences

Feature	<head>	<body>
Purpose	Metadata, configuration, resources	Visible web page content
Visibility	Not shown to users (except title/favicon)	Displayed to users (all content/sub-elements)
Typical Contents	<title>, <meta>, <link>, <script>	<h1>, <p>, <img>, <div>, <ul>, etc.
Browser Role	Setup, SEO, style/script links, background	Renders actual layout/content
Number Allowed	One per HTML doc, at page top	One per HTML doc, follows <head>
Display Location	In browser tab, bookmarks, search results	In the web page itself

Summary

- Use **<head>** for everything that configures or describes your document.
- Use **<body>** for everything that you want displayed as content to the visitor.
- Both are required (by convention and standards) for properly structured HTML and predictable browser rendering.

default size of html tags -

The default font sizes of HTML heading tags and paragraph tag are set by browsers with typical values expressed in relative units (em) or absolute pixels, based on a default browser font size of 16px.

Default Sizes of HTML Heading Tags

- <h1>: 2em (approximately 32px) — the largest heading size
- <h2>: 1.5em (approximately 24px)
- <h3>: 1.17em (approximately 18.7px)
- <h4>: 1em (approximately 16px)

- <h5>: 0.83em (approximately 13.3px)
- <h6>: 0.67em (approximately 10.7px)

These headings generally appear bold and block-level by default, with top and bottom margins that provide spacing between them and other content.

Default Size of the Paragraph (<p>) Tag

- Paragraph text typically defaults to 1em, which is 16px if the browser's base font size is 16px.
- Paragraphs are block-level elements with default top and bottom margins to separate them visually from other elements.

Summary Table

HTML Tag	Default Font Size (em)	Approximate Pixel Size (px)
<h1>	2	32
<h2>	1.5	24
<h3>	1.17	18.7
<h4>	1	16
<h5>	0.83	13.3
<h6>	0.67	10.7
<p>	1	16

Notes:

- These values come from typical browser default styles, but can be overridden using CSS.
- The em unit is relative to the parent element’s font size, so these sizes may scale depending on where the element is placed.
- The paragraph tag usually inherits the default body font size of 16px if not otherwise specified.

This default sizing helps establish a typographic hierarchy to visually differentiate headings from paragraph text, guiding readers through the content structure clearly and consistently.

Case Sensitive vs Case Insensitive -

- **Case Sensitive** means that uppercase and lowercase letters are treated as **different**. When something is case sensitive, the exact capitalization of letters matters. For example, the strings "Hello" and "hello" are considered **different** because the first letter's case differs.
- **Case Insensitive** means that uppercase and lowercase letters are treated as **equivalent**. In this case, "Hello" and "hello" are considered the **same**, ignoring any difference in capitalization.

Examples:

- Case Sensitive:
  - "Apple" ≠ "apple"
  - "Test123" ≠ "test123"
- Case Insensitive:
  - "Apple" = "apple"
  - "Test123" = "test123"

Where it matters:

- **Usernames** are often case insensitive so users don’t get confused by capitalization.
- **Passwords** are usually case sensitive for stronger security.
- **Programming languages**: Some (like Java, C, Python) are case sensitive for variable and function names, while others (like SQL) can be case insensitive in some contexts.
- **File names**: Some operating systems treat file names as case sensitive (Unix/Linux), others treat them as case insensitive but case preserving (Windows).
- **Search engines**: Typically case insensitive for user convenience.
- **URLs**: The domain part is case insensitive, but the path can be case sensitive depending on the server.

Summary table:

Aspect	Case Sensitive	Case Insensitive
Definition	Differentiates uppercase vs lowercase letters	Treats uppercase and lowercase letters as the same
Example strings	"Hello" ≠ "hello"	"Hello" = "hello"
Programming use	Variable/function names must match exact case	Variable/function names are not case sensitive
Usernames	Usually case insensitive	Usually case insensitive
Passwords	Usually case sensitive	Rarely case insensitive
File systems	Linux is case sensitive	Windows is case insensitive (but case preserving)
Search queries	Differentiates 'Dog' and 'dog' in some cases	Usually treats 'Dog' and 'dog' as the same

In summary: Case sensitivity strictly considers letter casing important for comparisons, while case insensitivity ignores letter casing differences when comparing or matching text.

html text formating elements -

HTML provides a variety of **text formatting elements** that allow you to modify the appearance and semantic meaning of text on a webpage. These elements help you make text bold, italic, emphasized, highlighted, smaller, subscripted, superscripted, inserted, deleted, and more.

Here’s a concise list of common HTML text formatting elements and their functions:

Tag	Description	Notes
<b>	Makes text **bold** visually without extra importance.	Physical style only
<strong>	Indicates **important** or strong emphasis, displayed bold.	Semantic meaning for importance
<i>	Italicizes text without implying emphasis.	Physical style only
<em>	Emphasizes text, usually rendered in italics.	Semantic emphasis for accessibility
<mark>	Highlights text with a yellow background (like a marker).	Useful for highlighting keywords
<small>	Renders text in a smaller font size than surrounding text.	
<del>	Shows deleted or removed text with a strikethrough.	Often used to indicate edits
<ins>	Displays inserted text, usually underlined.	Indicates additions
<sub>	Formats subscript text, shown slightly below the baseline.	Used in formulas (e.g., H_2O)
<sup>	Formats superscript text, shown slightly above the baseline.	Used for exponents, footnotes
<u>	Underlines text (used less in modern semantic HTML).	Purely visual, not recommended for emphasis

<code>	Displays text in a fixed-width font, typically for code.	Used to show programming code or commands

Semantic vs Visual Tags:

- Tags like <strong> and <em> carry **semantic meaning** about the importance or emphasis of text, improving accessibility and SEO.
- Tags like <b> and <i> affect only the visual appearance without conveying extra meaning.

Example usage in HTML:

xml

```
<p>This is <b>bold</b> and this is <strong>strongly emphasized</strong>.</p>

<p>This text is <i>italic</i> and this is <em>emphasized</em> for importance.</p>

<p>Water is H<sub>2</sub>O and Einstein’s formula is E = mc<sup>2</sup>.</p>

<p><mark>Highlighted</mark> text and <del>deleted</del> text with <ins>inserted</ins>
words.</p>
```

These text formatting tags are essential for structuring readable, accessible, and semantically meaningful content on webpages while providing default styles that browsers render appropriately.

Relative and Absolute path -

In web development, **relative paths** and **absolute paths** are ways to specify the location of files (like images, CSS, or HTML files) in your project or on the web.

Absolute Path

- Specifies the full URL or complete path to a resource, starting from the root of the website or including the full domain.
- It is independent of the current location of the file that references it.
- Used when you want to link to a resource at a fixed location, no matter where the referencing file is.
- Example (full URL):
- text
- `https://www.example.com/images/logo.png`
- Example (root-relative path):

- text
- /images/logo.png
- Here, the / at the start means it begins from the root directory of the website.

Relative Path

- Specifies the location of a file relative to the current file or directory.
- Depends on the location of the file where the path is written.
- Useful for linking local files within the same project or website without needing to specify the full URL.
- Examples:
  - images/logo.png — Looks for the images folder inside the current directory.
  - ../images/logo.png — Goes one directory up and then into the images folder.
  - ./logo.png — Refers to the current directory.

Key Differences:

Feature	Absolute Path	Relative Path
Definition	Full path starting from root or domain	Path relative to the current file's location
Usage	Links to a fixed location	Links to files relative to current file
Flexibility	Less flexible when moving files or folders	More flexible within a project structure
Example	/css/style.css or https://site.com/css/style.css	css/style.css, ../css/style.css

When to Use

- Use **absolute paths** for linking to external sites or resources hosted in a fixed location.
- Use **relative paths** for internal links within your website or web project because they adapt when files/folders change location relative to each other.

Understanding these paths helps you organize and reference files correctly in HTML and CSS to ensure your website's resources load properly.

Map element in html -

The HTML <map> element is used to define an **image map**, which is an image with clickable areas called "hotspots." These hotspots are defined within the <map> element using one or more <area> tags, each specifying a shape and coordinates for a clickable region that links to different URLs or performs specific actions.

How the <map> element works:

- The <map> element gets a unique name attribute, which is then referenced by the usemap attribute on the related <img> tag.
- This association connects the image to its map of clickable areas.
- Inside the <map>, <area> elements define the clickable regions.

Key parts:

- <img src="image.jpg" usemap="#mapname" alt="description"> — The image referencing the map.
- <map name="mapname"> — The container defining clickable areas.
- <area shape="rect|circle|poly" coords="..." href="..." alt="..."> — Defines the shape, coordinates, link, and alternative text of each clickable area.

Common attributes of <area>:

- shape: Defines the clickable area's shape; can be:
  - rect (rectangle)
  - circle
  - poly (polygon)
- coords: Defines the coordinates of the shape on the image.
- href: URL to open when the area is clicked.
- alt: Alternative text for accessibility.
- target (optional): Specifies where to open the link (e.g., _blank for a new tab).

Example of an image map:

```
<img src="workplace.jpg" alt="Workplace" usemap="#workmap" width="400" height="379">



<map name="workmap">

  <area shape="rect" coords="34,44,270,350" alt="Computer" href="computer.htm">

  <area shape="rect" coords="290,172,333,250" alt="Phone" href="phone.htm">

  <area shape="circle" coords="337,300,44" alt="Cup of coffee" href="coffee.htm">
```

</map>

In this example, different areas of the "workplace.jpg" image are clickable and lead to distinct pages like computer.htm, phone.htm, etc.

Summary:

- The <map> element creates interactive clickable zones on images.
- Use the name attribute in <map> and the usemap attribute in <img> to link them.
- Define clickable shapes with <area> tags specifying shapes, coordinates, and links.
- Useful for navigational images, diagrams, or any graphic needing multiple links.

This structure enhances user interaction by enabling different links on parts of a single image.

Srcset -

The **srcset attribute** in HTML is used with the <img> tag (and <source> in <picture>) to provide a list of image sources for different scenarios, allowing the browser to select the most appropriate image based on the device's screen size, resolution, or other factors. This enables **responsive images** that optimize loading performance and visual quality on various devices.

How srcset works:

- It specifies multiple image URLs along with descriptors indicating either the image's width (w) or pixel density (x).
- The browser evaluates the available options and selects the best image to display, minimizing data usage while maintaining clarity.
- Typically used together with the sizes attribute when width descriptors are used, to tell the browser how much space the image will take in the layout.

Types of descriptors:

- **Width descriptor (w)**: Specifies the intrinsic width of the image in pixels.
  Example:
- xml
- <img srcset="small.jpg 480w, medium.jpg 800w, large.jpg 1200w" sizes="(max-width: 600px) 480px, 800px" src="medium.jpg" alt="Example image" />
- This tells the browser:
  - Use small.jpg if viewport width ≤ 600px (480px wide image).
  - Otherwise use the 800px wide image.
- **Pixel density descriptor (x)**: Specifies the device pixel ratio.
  Example:
- xml
- <img srcset="image-1x.jpg 1x, image-2x.jpg 2x" src="image-1x.jpg" alt="Example image" />

- This tells the browser to use image-2x.jpg for devices with high-resolution (like Retina displays).

Why use srcset?

- Provides **responsive images** that adapt to different screen sizes and resolutions.
- Improves **performance** by loading smaller images on small or low-res devices and larger images only when needed.
- Enhances **visual quality** on high-density displays.

Example:

```
<img
 srcset="
  small.jpg 480w,
  medium.jpg 800w,
  large.jpg 1200w
 "
 sizes="(max-width: 600px) 480px, 800px"
 src="medium.jpg"
 alt="A beautiful scenery"
/>
```

This example provides three image versions, and the browser picks the best one depending on the viewport width.

Summary:

- srcset allows specifying multiple images for different devices.
- Improves loading speed and image quality.
- Used with sizes for width-based selections, or with density descriptors for high-DPI displays.

This attribute is essential for modern, responsive web design to deliver optimized images tailored to device capabilities.

Picture Tag -

The HTML <picture> tag is a container element designed to offer **multiple image sources** so that the browser can choose the most appropriate image based on factors like viewport width, device resolution, or image format support. It is very useful for **responsive images** and **art direction** in modern web design.

How the <picture> tag works:

- It contains one or more <source> elements and exactly one <img> element.
- Each <source> element specifies an image source with attributes like srcset (image URLs) and media (media queries).
- The browser evaluates the <source> elements in order and selects the first one whose media query matches the current device conditions.
- The <img> tag acts as a fallback if none of the <source> elements match or if the browser does not support the <picture> tag.

Basic syntax:

```
<picture>

  <source media="(min-width: 650px)" srcset="large-image.jpg">

  <source media="(min-width: 465px)" srcset="medium-image.jpg">

  <img src="small-image.jpg" alt="Description" style="width:auto;">

</picture>
```

- For screens at least 650px wide, the browser uses large-image.jpg.
- For screens at least 465px but less than 650px, the browser uses medium-image.jpg.
- For smaller screens or unsupported browsers, it falls back to small-image.jpg.

Main uses:

- **Responsive Images**: Serving appropriately sized images based on device viewport to improve loading times and user experience.
- **Art Direction**: Choosing different images to better suit different layouts or screen sizes instead of simply scaling one image.
- **Image Format Support**: Providing multiple formats (e.g., WebP and JPEG) so the browser picks the one it can render.

Important points:

- The <picture> element itself doesn’t display an image but controls which image is displayed.
- The <img> element inside <picture> must have an alt attribute for accessibility.
- The <picture> element supports global HTML attributes and event attributes.

This tag gives developers fine-grained control over image selection based on device and browser context, optimizing both performance and visual design.

Figure Tag -

The HTML <figure> tag is used to mark up **self-contained content** such as illustrations, diagrams, photos, code listings, or other media that is related to the main flow of the document but could stand alone. It semantically groups this content along with an optional caption, which is provided by the <figcaption> element.

Key points about <figure>:

- Represents a unit of content that is referenced in the main text but can be independently moved or removed without affecting the flow of the document.
- Typically contains media content like images <img>, diagrams, or code snippets.
- Can include a <figcaption> element as its first or last child to provide a caption or explanation for the content.
- Helps with semantic structure, accessibility, and SEO by linking media with descriptive captions.
- Browsers usually render <figure> as a block-level element with some default margins.

Basic structure example:

xml

```
<figure>

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

  <figcaption>This is a caption describing the image.</figcaption>

</figure>
```

In this example:

- The <img> displays the image.
- The <figcaption> gives a caption explaining or describing the image.
- Together, they form a semantically meaningful and accessible content block independent of the main paragraph text.

Summary

- The <figure> tag is ideal for grouping visual or media content with a caption.
- The optional <figcaption> provides descriptive text for that media.

- Using <figure> and <figcaption> enhances semantic meaning, accessibility, and helps search engines understand the relationship between content and images or media.

This tag is part of HTML5 semantic elements designed to improve web content organization and meaning.