What does a DOCTYPE do?

DOCTYPE is an abbreviation for **DOCument TYPE**. A DOCTYPE is always associated to a **DTD** - for **Document Type Definition**.

A DTD defines how documents of a certain type should be structured (i.e. a `button` can contain a `span` but not a `div`), whereas a DOCTYPE declares what DTD a document *supposedly* respects (i.e. this document respects the HTML DTD).

For webpages, the DOCTYPE declaration is required. It is used to tell user agents what version of the HTML specifications your document respects. Once a user agent has recognized a correct DOCTYPE, it will trigger the **no-quirks mode** matching this DOCTYPE for reading the document. If a user agent doesn't recognize a correct DOCTYPE, it will trigger the **quirks mode**.

The DOCTYPE declaration for the HTML5 standards is `<!DOCTYPE html>`.

References

- https://html.spec.whatwg.org/multipage/syntax.html#the-doctype
- https://html.spec.whatwg.org/multipage/xhtml.html
- https://quirks.spec.whatwg.org/

[↑] Back to top

How do you serve a page with content in multiple languages?

The question is a little vague, I will assume that it is asking about the most common case, which is how to serve a page with content available in multiple languages, but the content within the page should be displayed only in one consistent language.

When an HTTP request is made to a server, the requesting user agent usually sends information about language preferences, such as in the `Accept-Language` header. The server can then use this information to return a version of the document in the appropriate language if such an alternative is available. The returned HTML document should also declare the `lang` attribute in the `<html>` tag, such as `<html lang="en">...</html>`.

In the back end, the HTML markup will contain `i18n` placeholders and content for the specific language stored in YML or JSON formats. The server then dynamically generates the HTML page with content in that particular language, usually with the help of a back end framework.

References

- https://www.w3.org/International/getting-started/language

[↑] Back to top

What kind of things must you be wary of when designing or developing for multilingual sites?

- Use `lang` attribute in your HTML.
- Directing users to their native language - Allow a user to change his country/language easily without hassle.
- Text in raster-based images (e.g. png, gif, jpg, etc.), is not a scalable approach - Placing text in an image is still a popular way to get good-looking, non-system fonts to display on any computer. However, to translate image text, each string of text will need to have a separate image created for each language. Anything more than a handful of replacements like this can quickly get out of control.
- Restrictive words/sentence length - Some content can be longer when written in another language. Be wary of layout or overflow issues in the design. It's best to avoid designing where the amount of text would make or break a design. Character counts come into play with things like headlines, labels, and buttons. They are less of an issue with free-flowing text such as body text or comments.
- Be mindful of how colors are perceived - Colors are perceived differently across languages and cultures. The design should use color appropriately.
- Formatting dates and currencies - Calendar dates are sometimes presented in different ways. Eg. "May 31, 2012" in the U.S. vs. "31 May 2012" in parts of Europe.
- Do not concatenate translated strings - Do not do anything like `"The date today is " + date`. It will break in languages with different word order. Use a template string with parameters substitution for each language instead. For example, look at the following two sentences in English and Chinese respectively: `I will travel on {% date %}` and `{% date %} 我会出发`. Note that the position of the variable is different due to grammar rules of the language.
- Language reading direction - In English, we read from left-to-right, top-to-bottom, in traditional Japanese, text is read up-to-down, right-to-left.

References

- https://www.quora.com/What-kind-of-things-one-should-be-wary-of-when-designing-or-developing-for-multilingual-sites

[↑] Back to top

What are `data-` attributes good for?

Before JavaScript frameworks became popular, front end developers used `data-` attributes to store extra data within the DOM itself, without other hacks such as non-standard attributes, extra properties on the DOM. It is intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements.

These days, using `data-` attributes is generally not encouraged. One reason is that users can modify the data attribute easily by using inspect element in the browser. The data model is better stored within JavaScript itself and stay updated with the DOM via data binding possibly through a library or a framework.

However, one perfectly valid use of data attributes, is to add a hook for *end to end* testing frameworks such as Selenium and Capybara without having to create a meaningless classes or ID attributes. The element needs a way to be found by a particular Selenium spec and something like `data-selector='the-thing'` is a valid way to do so without convoluting the semantic markup otherwise.

References

- http://html5doctor.com/html5-custom-data-attributes/
- https://www.w3.org/TR/html5/dom.html#embedding-custom-non-visible-data-with-the-data-*-attributes

[↑] Back to top

Consider HTML5 as an open web platform. What are the building blocks of HTML5?

- Semantics - Allowing you to describe more precisely what your content is.
- Connectivity - Allowing you to communicate with the server in new and innovative ways.
- Offline and storage - Allowing webpages to store data on the client-side locally and operate offline more efficiently.
- Multimedia - Making video and audio first-class citizens in the Open Web.
- 2D/3D graphics and effects - Allowing a much more diverse range of presentation options.
- Performance and integration - Providing greater speed optimization and better usage of computer hardware.
- Device access - Allowing for the usage of various input and output devices.
- Styling - Letting authors write more sophisticated themes.

References

- https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[↑] Back to top

Describe the difference between a `cookie`, `sessionStorage` and `localStorage`.

All the above-mentioned technologies are key-value storage mechanisms on the client side. They are only able to store values as strings.

	`cookie`	`localStorage`	`sessionStorage`
Initiator	Client or server. Server can use `Set-Cookie` header	Client	Client
Expiry	Manually set	Forever	On tab close
Persistent across browser sessions	Depends on whether expiration is set	Yes	No
Sent to server with every HTTP request	Cookies are automatically being sent via `Cookie` header	No	No
Capacity (per domain)	4kb	5MB	5MB
Accessibility	Any window	Any window	Same tab

Note: If the user decides to clear browsing data via whatever mechanism provided by the browser, this will clear out any cookie, localStorage, or sessionStorage stored. It's important to keep this in mind when designing for local persistance, especially when comparing to alternatives such as server side storing in a database or similar (which of course will persist despite user actions).

References

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
- http://tutorial.techaltum.com/local-and-session-storage.html

[↑] Back to top

Describe the difference between `<script>`, `<script async>` and `<script defer>`.

- `<script>` - HTML parsing is blocked, the script is fetched and executed immediately, HTML parsing resumes after the script is executed.
- `<script async>` - The script will be fetched in parallel to HTML parsing and executed as soon as it is available (potentially before HTML parsing completes). Use `async` when the script is independent of any other scripts on the page, for example, analytics.
- `<script defer>` - The script will be fetched in parallel to HTML parsing and executed when the page has finished parsing. If there are multiple of them, each deferred script is executed in the order they were encountered in the document. If a script relies on a fully-parsed DOM, the `defer` attribute will be useful in ensuring that the HTML is fully parsed before executing. There's not much difference in putting a normal `<script>` at the end of `<body>`. A deferred script must not contain `document.write`.

Note: The `async` and `defer` attributes are ignored for scripts that have no `src` attribute.

References

- http://www.growingwiththeweb.com/2014/02/async-vs-defer-attributes.html
- https://stackoverflow.com/questions/10808109/script-tag-async-defer
- https://bitsofco.de/async-vs-defer/

[↑] Back to top

Why is it generally a good idea to position CSS `<link>`s between `<head></head>` and JS `<script>`s just before `</body>`? Do you know any exceptions?

Placing s in the

Putting `<link>`s in the head is part of proper specification in building an optimized website. When a page first loads, HTML and CSS are being parsed simultaneously; HTML creates the DOM (Document Object Model) and CSS creates the CSSOM (CSS Object Model). Both are needed to create the visuals in a website, allowing for a quick "first meaningful paint" timing. This progressive rendering is a category optimization sites are measured in their performance scores. Putting stylesheets near the bottom of the document is what prohibits progressive rendering in many browsers. Some browsers block rendering to avoid having to repaint elements of the page if their styles change. The user is then stuck viewing a blank white page. Other times there can be flashes of unstyled content (FOUC), which can shows a webpage with no styling applied.

Placing