TECHNICAL QUESTIONS: *HTML*

What does a DOCTYPE do?

A DOCTYPE is a declaration/definition for a document. All HTML documents must start with a <!DOCTYPE> declaration. The declaration is not an HTML tag. It is "information" to the browser about what type of information to expect (what version of HTML the page is written in).

In HTML 5, the declaration is simple:

```
<!DOCTYPE html>
```

*If you don't declare a doctype, the HTML will be invalid and the browser will read your page in "Quirks" mode, which is a mode browsers use to interpret CSS. Quirk mode is the default compatibility mode and may be different from browser to browser, which may result in a lack of consistency in appearance. HTML emails generally use a "Strict" or standards mode, not your usual doctype.

What is Quirks mode?
Back in the wild west days of the Web, the two different browsers at the time had different rules. If you wrote a webpage in netscape vs explorer, your webpage would look different. Browsers needed to display webpages depending on what mode was declared: "Standards" mode (traditional, valid CSS) vs "Quirks" mode. Quirks mode is becoming less relevant because most webpages are using the valid layout.

What is the difference between HTML and XHTML?

XHTML is based upon the rules of a markup language called XML. HTML (the language to define the structure of the webpage) is not very strict. While browser will render the page and everything looks fine, the rules are still not being followed correctly.

XML has a very strict structure (certain syntax rules that must be followed). If these rules aren't followed, the webpage will be invalid. Since it is so structured, XHTML will allow you to identify problems/bugs within your layout. Whereas, in HTML, it is much more flexible, which may seem like a benefit but it doesn't allow you to find the bugs as well.

If you have a good editor, you won't really need to write in XHTML. If you use a validator for your HTML code, your HTML code will be fine.

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

You should always include the "lang" attribute inside the <html> tag, to declare the language of the webpage. This is meant to assist browsers and search engines. Also, you can let the browser know the language of each element by using the "lang" attribute within it. This is done because the "lang" attribute can only take in one value so if you have multiple languages inside of your page, you would have separate those elements and make sure they have their own "lang" attribute assigned to them, like so:

```
<!DOCTYPE html>
<html lang="fr">

<body>
    <p>
        We can use different languages in the HTML
        document simply by defining the
        "lang" property
    </p>

    <p>French " <span lang="fr">Bonjour</span> "</p>
    <p>Spanish " <span lang="es">Hola</span> "</p>
    <p>Hindi " <span lang="hi">नमस्ते</span> "</p>

</body>

</html>
```

One way is to store multiple version of the site's content on a database. Based upon a user interaction or location, you would serve up that location's spoken language.

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

Here are some pointers on designing our multilingual website:

- Make sure to use the "lang" attribute. This will allow the browser to determine the original language of the webpage, thereby allowing it to translate the page effectively.
- Make the language switcher really easy to find. This will allow the user to switch to their native language as soon as they enter the webpage, without hassle.
- Don't use texts inside of your images, as they can't be translated.
- Make sure to design your page for right-to-left languages, such as Farsi, Arabic, Hebrew. A good example of this is the facebook's homepage where the design for Arabic is mirrored version of the English counterpart. This can be done with flexbox.
- You want to be wary of the images you are using on your website. For example, if you want to include an image of the globe on your multilingual website, it would be wise to use one that doesn't feature a specific country or region on the front.
- Make sure to use culturally appropriate colours. In western culture, red is seen as a symbol of love and passion, in Asian culture, red is used for good luck, and in African culture, red is associated with death and aggression. When choosing colours for your website, you can google symbolism associated with any colours you plan to use.
- Not every country uses the same date format, even across the English language. Same goes for units of measure.
- Another thing to consider when serving multilingual webpages is the captchas. You can't expect a Japanese user to enter the correct captcha if it is in English.
- Also, make sure you are covering the characters (meta tag with UTF-8 will serve up the Latin characters required).

What are data- attributes good for?

data- attributes can be useful because they can be added to an element, to which you can make changes to via the DOM and CSS. For example:

Let's say you have a website for upcoming concert events. Currently, on your page, you have two divs with a class of events that have information about the concert within them, both having a data-date attribute, e.g., data-date = "1/1/2022" and data-date = "1/8/2020".

What if the date of the event has passed and you want that div to be either hidden, faded away, or whatever? You can go on your JS file:

```
const events = document.querySelectorAll('.event')

events.forEach( e => {
const date = new Date(e.dataset.date);
const todaysDate = new Date();

if (date < todaysDate){
    e.classList.add('disabled');
}});
```

This will add a 'disabled' class to the div that has a date < todaysDate. You can then apply CSS properties, such as low opacity and so on. On your website, the div for the past concert will now be faded.

What are the building blocks of HTML5?

- Semantic Tags: Allows you to describe more precisely what your content is. Instead of filling your entire page with tags, you can use tags that provide more information, such as: article, aside, section, header, nav, and so on. They help with searchability (helps search engines), accessibility (helpful for users who use screen readers), and it is another way of self-documenting code (someone can figure out the layout/intentions of your code just by looking at the tags).
- Audio/Video: HTML5 allows you to embed audio and video into your page.
- Web Storage API: sessionStorage is a temporary storage mechanism within the browser, which ends when you close the browser. localStorage persists even when you shut the browser (a good example for it is if a user has selected a theme for the website that they want to use. You can store that in their local browser rather than your server). These enable you to store information locally.
- 2D/3D graphics and effects: allows for much more diverse range of presentation options.

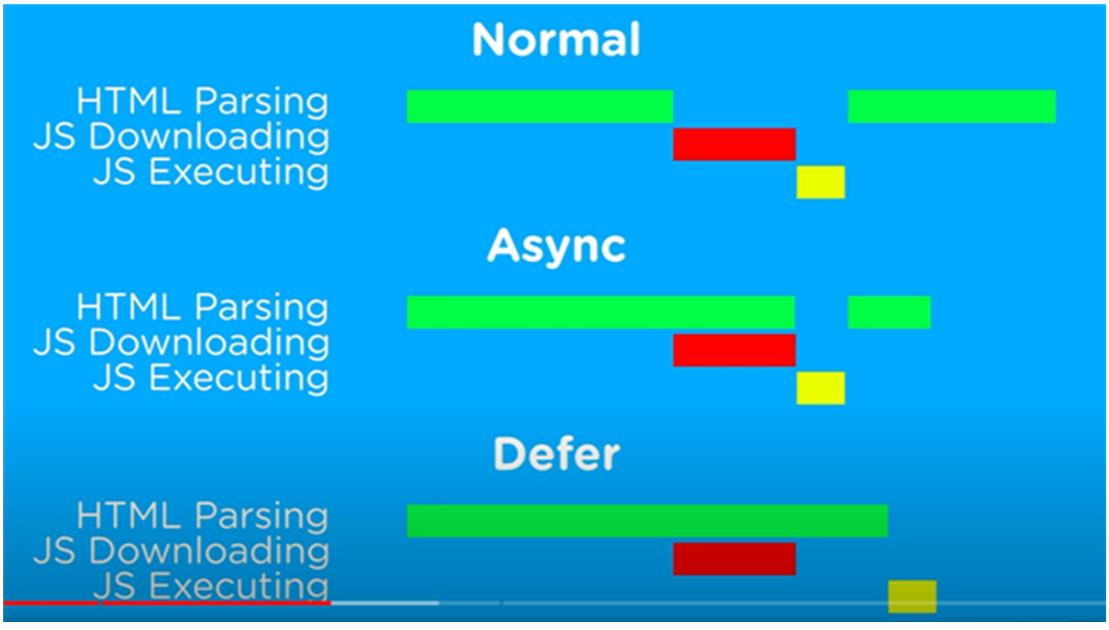
Describe the difference between a cookie, sessionStorage, and localStorage.

Cookies are a way to store data on client's side. They're used for authentication (keep users logged via a session).

sessionStorage and localStorage are almost exactly the same. The main difference is that sessionStorage only sticks around as long the browser is open and localStorage is persistent. It is not recommended to store sensitive information into session or localStorage. It can be done on cookies via encryption.

	cookies	local	session
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

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. This tag is "blocking" meaning it's blocking the parsing of the HTML page, which is not a good user experience.

<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. This is a little bit better because it isn't blocking the parsing of the HTML completely.

<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 from putting a normal <script> at the end of <body>.

Where do <link> and <script> tags go in HTML? Are there exceptions?

Placing <link>s in the <head>— Putting <link>s in the head is part of the specification. Besides that, placing at the top allows the page to render progressively which improves user experience. The problem with putting stylesheets near the bottom of the document is that it prohibits progressive rendering in many browsers, including Internet Explorer. Some browsers block rendering to avoid having to repaint elements of the page if their styles change. The user is stuck viewing a blank white page. It prevents the flash of unstyled contents.

Placing <scripts>s just before </body> — <script>s block HTML parsing while they are being downloaded and executed. Downloading the scripts at the bottom will allow the HTML to be parsed and displayed to the user first.

An exception for positioning of <script>s at the bottom is when your script contains document.write(), but these days it's not a good practice to use document.write(). Also, placing <script>s at the bottom means that the browser cannot start downloading the scripts until the entire document is parsed. One possible workaround is to put <script>in the <head> and use the defer attribute.

What is progressive rendering?

When we are talking about rendering, we are talking about what a browser does when it downloads a webpage. It goes through a variety of steps:

1. First step is the DOM, which takes the HTML page and translates that into an object representation.
2. Then the browser does the same for the styles (creating another model: CSS object model).
3. Once those two steps are done. It renders the render tree (DOM + CSS), which is another object.

When the browser is doing all this, it is called rendering. Progressive rendering asks the question 'what is the most useful thing for users to see ASAP?'. The user will get a feeling that the page is loading. We have to decide what is the most critical part of the webpage and how it can be shown to them ASAP.

It used to be much more prevalent in the days before broadband internet but it is still used in modern development as mobile data connections are becoming increasingly popular (and

unreliable)!

Examples of such techniques:

- Lazy loading of images - Images on the page are not loaded all at once. JavaScript will be used to load an image when the user scrolls into the part of the page that displays the image.
- Prioritizing visible content (or above-the-fold rendering) - Include only the minimum CSS/content/scripts necessary for the amount of page that would be rendered in the users browser first to display as quickly as possible, you can then use deferred scripts or listen for the DOMContentLoaded/load event to load in other resources and content.
- Async HTML fragments - Flushing parts of the HTML to the browser as the page is constructed on the back end.

Why would you use a srcset attribute in an image tag?

You would use the srcset attribute when you want to serve different images to users depending on their device display width - serve higher quality images to devices with retina display enhances the user experience while serving lower resolution images to low-end devices increase performance and decrease data wastage (because serving a larger image will not have any visible difference). For example: <img srcset="small.jpg 500w, medium.jpg 1000w, large.jpg 2000w" src="..." alt=""> tells the browser to display the small, medium or large .jpg graphic depending on the client's resolution. The first value is the image name and the second is the width of the image in pixels. For a device width of 320px, the following calculations are made:

- 500 / 320 = 1.5625
- 1000 / 320 = 3.125
- 2000 / 320 = 6.25

If the client's resolution is 1x, 1.5625 is the closest, and 500w corresponding to small.jpg will be selected by the browser.

If the resolution is retina (2x), the browser will use the closest resolution above the minimum. Meaning it will not choose the 500w (1.5625) because it is greater than 1 and the image might look bad. The browser would then choose the image with a resulting ratio closer to 2 which is 1000w (3.125).

srcsets solve the problem whereby you want to serve smaller image files to narrow screen devices, as they don't need huge images like desktop displays do — and also optionally that you want to serve different resolution images to high density/low-density screens.

Have you used different HTML templating languages before?

Yes, Pug (formerly Jade), ERB, Slim, Handlebars, Jinja, Liquid, and EJS just to name a few. In my opinion, they are more or less the same and provide similar functionality of escaping content and helpful filters for manipulating the data to be displayed. Most templating engines will also allow you to inject your own filters in the event you need custom processing before display.