01. Does **localStorage** throw error after reaches maximum limits?

Maybe it depends on the browser. Different browsers do things differently.

02. What are the new form elements in HTML5?

**<datalist>**

You can use the <datalist> element if you want to provide suggestions for the input, and at the same time allow user to enter inputs on his own.

**<meter>**

The <meter> element provides for a gauge, displaying a general value within a range. You provide minimum (min) and maximum (max) values, and the required value that falls between those minimum and maximum values. While many think it’s a form control with attributes similar to some numeric input types, it has no name attribute and won’t be submitted on form submission.

***Note***: The <meter> tag should not be used to indicate progress (as in a progress bar). For progress bars, use the <progress> tag.

**<progress>**

The <progress> tag represents the completion progress of a task.

**<output>**

Perform a calculation and show the result in an <output> element.

<keygen> deprecated… sad..

03. What is the DOM? How does the DOM work?

The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web.

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

04. How does the browser rendering engine work?

When a web page is loaded, the browser first reads the HTML text and constructs DOM Tree from it. Then it processes the CSS whether that is inline, embedded, or external CSS and constructs the CSSOM Tree from it.

After these trees are constructed, then it constructs the Render-Tree from it. Once the Render-Tree is constructed, then the browser starts the printing individual elements on the screen.

05. What does a <!DOCTYPE html> do?

All HTML documents **must start with a <!DOCTYPE>** declaration.

The declaration is not an HTML tag. It is an "information" to the browser about what document type to expect.

06. What happens when DOCTYPE is not given?

The browser enters Quirks mode and tries to deal with your code as if it was written in the late 90’s. This means they will imitate many bugs that existed in the old browsers. It is not consistent on how to do this (so you will get significantly different results, for example, from Firefox to Internet Explorer).

07. What is the difference between standards mode and quirks mode?

Obviously,the **css box model**?

In computing, quirks mode refers to a technique used by some web browsers for the sake of maintaining backward compatibility with web pages designed for old web browsers instead of strictly complying with W3C and IETF standards in standards mode.

In "standards mode" pages are rendered according to the HTML and CSS specifications, while in "quirks mode" attempts are made to emulate the behavior of older browsers.

08. What is difference between HTML and XHTML?

XHTML stands for EXtensible HyperText Markup Language

XHTML is a stricter, more XML-based version of HTML

XHTML is HTML defined as an XML application

XHTML is supported by all major browsers

* <!DOCTYPE> is mandatory
* The xmlns attribute in <html> is mandatory
* <html>, <head>, <title>, and <body> are mandatory
* Elements must always be properly nested
* Elements must always be closed
* Elements must always be in lowercase
* Attribute names must always be in lowercase
* Attribute values must always be quoted
* Attribute minimization is forbidden

09. 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.

* more semantic text markup
* new form elements
* vedio and audio
* new javascript API
* canvas and SVG
* new communication API
* geolocation API
* web worker API
* new data storage

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

**HTML5 web storage** = generic umbrella term for the new client-side data storage options:

Web Storage is more secure and faster. With web storage, web applications can store data locally within the user's browser. Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server. Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

* **Local Storage** = persistant and scoped to **the domain**(store data with **no expiration date**). At the moment two flavors are usually mentioned:
  + 'default' = stores things in name/value pairs
  + Web SQL (aka Web Database) = uses a SQL database
* **Session Storage** = non persistent and scoped only to the current window(stores data **for one session**)

**Cookies** = the old school way of doing all of the above. Stores name/value pairs per domain.

11. What is Critical Rendering Path?

The Critical Rendering Path is the sequence of steps the browser goes through to convert the HTML, CSS, and JavaScript into pixels on the screen. Optimizing the critical render path improves render performance. The critical rendering path includes the Document Object Model (DOM), CSS Object Model (CSSOM), render tree and layout.

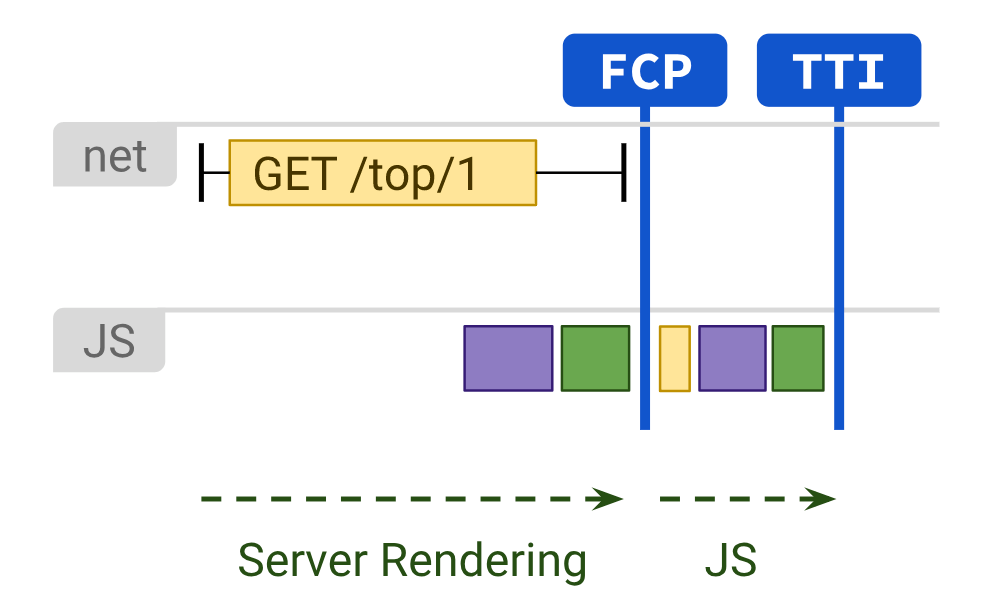
The document object model is created as the HTML is parsed. The HTML may request JavaScript, which may, in turn, alter the DOM. The HTML includes or makes requests for styles, which in turn builds the CSS object model. The browser engine combines the two to create the Render Tree. Layout determines the size and location of everything on the page. Once layout is determined, pixels are painted to the screen.

Optimizing the critical rendering path improves the time to first render. Understanding and optimizing the critical rendering path is important to ensure reflows and repaints can happen at 60 frames per second, to ensure performant user interactions and avoid jank.

12. What are the Benefits of Server Side Rendering (SSR) Over Client Side Rendering (CSR)?

**Server rendering** generates the full HTML for a page on the server in response to navigation. This avoids additional round-trips for data fetching and templating on the client, since it’s handled before the browser gets a response.

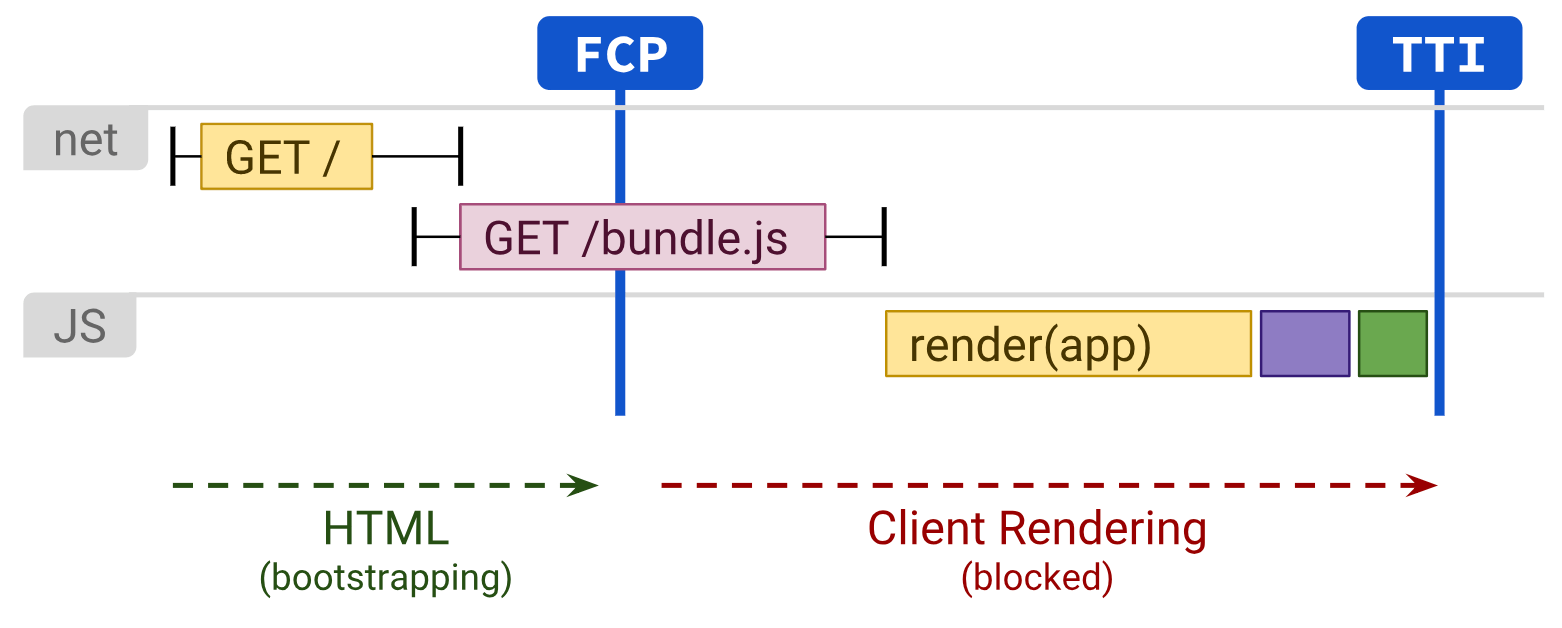
Server rendering generally produces a fast [First Paint](https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics#first_paint_and_first_contentful_paint) (FP) and [First Contentful Paint](https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics#first_paint_and_first_contentful_paint) (FCP). Running page logic and rendering on the server makes it possible to avoid sending lots of JavaScript to the client, which helps achieve a fast [Time to Interactive](https://developers.google.com/web/tools/lighthouse/audits/time-to-interactive) (TTI). This makes sense, since with server rendering you’re really just sending text and links to the user’s browser. This approach can work well for a large spectrum of device and network conditions, and opens up interesting browser optimizations like streaming document parsing.



With server rendering, users are unlikely to be left waiting for CPU-bound JavaScript to process before they can use your site. Even when [third-party JS](https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/loading-third-party-javascript/) can’t be avoided, using server rendering to reduce your own first-party [JS costs](https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4) can give you more "[budget](https://medium.com/@addyosmani/start-performance-budgeting-dabde04cf6a3)" for the rest. However, there is one primary drawback to this approach: generating pages on the server takes time, which can often result in a slower [Time to First Byte](https://en.wikipedia.org/wiki/Time_to_first_byte) (TTFB).

**Client-side rendering** (CSR) means rendering pages directly in the browser using JavaScript. All logic, data fetching, templating and routing are handled on the client rather than the server.

Client-side rendering can be difficult to get and keep fast for mobile. It can approach the performance of pure server-rendering if doing minimal work, keeping a [tight JavaScript budget](https://mobile.twitter.com/HenrikJoreteg/status/1039744716210950144) and delivering value in as few [RTTs](https://en.wikipedia.org/wiki/Round-trip_delay_time) as possible. Critical scripts and data can be delivered sooner using [HTTP/2 Server Push](https://www.smashingmagazine.com/2017/04/guide-http2-server-push/) or <link rel=preload>, which gets the parser working for you sooner. Patterns like [PRPL](https://developers.google.com/web/fundamentals/performance/prpl-pattern/) are worth evaluating in order to ensure initial and subsequent navigations feel instant.



The primary downside to Client-Side Rendering is that the amount of JavaScript required tends to grow as an application grows. This becomes especially difficult with the addition of new JavaScript libraries, polyfills and third-party code, which compete for processing power and must often be processed before a page’s content can be rendered. Experiences built with CSR that rely on large JavaScript bundles should consider [aggressive code-splitting](https://developers.google.com/web/fundamentals/performance/optimizing-javascript/code-splitting/), and be sure to lazy-load JavaScript - "serve only what you need, when you need it". For experiences with little or no interactivity, server rendering can represent a more scalable solution to these issues.

For folks building a Single Page Application, identifying core parts of the User Interface shared by most pages means you can apply the [Application Shell caching](https://developers.google.com/web/updates/2015/11/app-shell) technique. Combined with service workers, this can dramatically improve perceived performance on repeat visits.

13. What is the difference between a <span> and a <div>?

The difference between span and div is that a **span** element is **in-line** and usually used for a small chunk of HTML inside a line (such as inside a paragraph) whereas a **div** (division) element is **block-line** (which is basically equivalent to having a line-break before and after it) and used to group larger chunks of code.

14. Name 5 common block-level and inline HTML elements?

**block level:**

[<address>](https://www.w3schools.com/tags/tag_address.asp) [<article>](https://www.w3schools.com/tags/tag_article.asp) [<aside>](https://www.w3schools.com/tags/tag_aside.asp) [<blockquote>](https://www.w3schools.com/tags/tag_blockquote.asp) [<canvas>](https://www.w3schools.com/tags/tag_canvas.asp) [<dd>](https://www.w3schools.com/tags/tag_dd.asp) [<div>](https://www.w3schools.com/tags/tag_div.asp) [<dl>](https://www.w3schools.com/tags/tag_dl.asp) [<dt>](https://www.w3schools.com/tags/tag_dt.asp) [<fieldset>](https://www.w3schools.com/tags/tag_fieldset.asp) [<figcaption>](https://www.w3schools.com/tags/tag_figcaption.asp) [<figure>](https://www.w3schools.com/tags/tag_figure.asp) [<footer>](https://www.w3schools.com/tags/tag_footer.asp) [<form>](https://www.w3schools.com/tags/tag_form.asp) [<h1>-<h6>](https://www.w3schools.com/tags/tag_hn.asp) [<header>](https://www.w3schools.com/tags/tag_header.asp) [<hr>](https://www.w3schools.com/tags/tag_hr.asp) [<li>](https://www.w3schools.com/tags/tag_li.asp) [<main>](https://www.w3schools.com/tags/tag_main.asp) [<nav>](https://www.w3schools.com/tags/tag_nav.asp) [<noscript>](https://www.w3schools.com/tags/tag_noscript.asp) [<ol>](https://www.w3schools.com/tags/tag_ol.asp) [<p>](https://www.w3schools.com/tags/tag_p.asp) [<pre>](https://www.w3schools.com/tags/tag_pre.asp) [<section>](https://www.w3schools.com/tags/tag_section.asp) [<table>](https://www.w3schools.com/tags/tag_table.asp) [<tfoot>](https://www.w3schools.com/tags/tag_tfoot.asp) [<ul>](https://www.w3schools.com/tags/tag_ul.asp) [<video>](https://www.w3schools.com/tags/tag_video.asp)

**inline:**

[<a>](https://www.w3schools.com/tags/tag_a.asp) [<abbr>](https://www.w3schools.com/tags/tag_abbr.asp) [<acronym>](https://www.w3schools.com/tags/tag_acronym.asp) [<b>](https://www.w3schools.com/tags/tag_b.asp) [<bdo>](https://www.w3schools.com/tags/tag_bdo.asp) [<big>](https://www.w3schools.com/tags/tag_big.asp) [<br>](https://www.w3schools.com/tags/tag_br.asp) [<button>](https://www.w3schools.com/tags/tag_button.asp) [<cite>](https://www.w3schools.com/tags/tag_cite.asp) [<code>](https://www.w3schools.com/tags/tag_code.asp) [<dfn>](https://www.w3schools.com/tags/tag_dfn.asp) [<em>](https://www.w3schools.com/tags/tag_em.asp) [<i>](https://www.w3schools.com/tags/tag_i.asp) [<img>](https://www.w3schools.com/tags/tag_img.asp) [<input>](https://www.w3schools.com/tags/tag_input.asp) [<kbd>](https://www.w3schools.com/tags/tag_kbd.asp) [<label>](https://www.w3schools.com/tags/tag_label.asp) [<map>](https://www.w3schools.com/tags/tag_map.asp) [<object>](https://www.w3schools.com/tags/tag_object.asp) [<output>](https://www.w3schools.com/tags/tag_output.asp) [<q>](https://www.w3schools.com/tags/tag_q.asp) [<samp>](https://www.w3schools.com/tags/tag_samp.asp) [<script>](https://www.w3schools.com/tags/tag_script.asp) [<select>](https://www.w3schools.com/tags/tag_select.asp) [<small>](https://www.w3schools.com/tags/tag_small.asp) [<span>](https://www.w3schools.com/tags/tag_span.asp) [<strong>](https://www.w3schools.com/tags/tag_strong.asp) [<sub>](https://www.w3schools.com/tags/tag_sub.asp) [<sup>](https://www.w3schools.com/tags/tag_sup.asp) [<textarea>](https://www.w3schools.com/tags/tag_textarea.asp) [<time>](https://www.w3schools.com/tags/tag_time.asp) [<tt>](https://www.w3schools.com/tags/tag_tt.asp) [<var>](https://www.w3schools.com/tags/tag_var.asp)

15. What are semantic and non-semantic elements?

These elements simply mean, elements with meaning. The reason being, there definition in the code tells the browser and the developer what they are supposed to do. Framing in simpler words, these elements describe the type of content they are supposed to contain. For example,

* article
* aside
* details
* figcaption
* figure
* footer
* form
* header
* main
* mark
* nav
* table
* section

Non-Semantic elements: Unlike, semantic elements they don’t have any meaning. They don’t tell anything about the content they contain. They can be used with different attributes to mark up semantics common to a group. For example, <div> and <span>.

16. What is the purpose of main element?

The **HTML <main> element** represents the dominant content of the [<body>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/body) of a document. The main content area consists of content that is directly related to or expands upon the central topic of a document, or the central functionality of an application.

17. Define semantic markup. What are the semantic meanings for <section>, <article>, <aside>,

<nav>, <header>, <footer> and when/how should each be used in structuring html markup?

There are two different practices that must be put into place if we are going to write semantic markup.

* Semantic markup requires that HTML elements be used according to their intended purpose.
* Semantic markup requires the separation of content and presentation.

The **<section>** element defines a section in a document.

According to W3C's HTML documentation: "A section is a thematic grouping of content, typically with a heading."

A web page could normally be split into sections for introduction, content, and contact information.

The **<article>** element specifies independent, self-contained content.

An article should make sense on its own, and it should be possible to distribute it independently from the rest of the web site.

Examples of where an <article> element can be used:

* Forum post
* Blog post
* Newspaper article

The **<header>** element represents a container for introductory content or a set of navigational links.

A <header> element typically contains:

one or more heading elements (<h1> - <h6>)

logo or icon

authorship information

Table

Description automatically generatedNote: You can have several <header> elements in one HTML document. However, <header> cannot be placed within a <footer>, <address> or another <header> element.

The **<footer>** element defines a footer for a document or section.

A <footer> element typically contains:

* authorship information
* copyright information
* contact information
* sitemap
* back to top links
* related documents

You can have several <footer> elements in one document.

The **<nav>** element defines a set of navigation links.

Notice that NOT all links of a document should be inside a <nav> element. The <nav> element is intended only for major block of navigation links.

Browsers, such as screen readers for disabled users, can use this element to determine whether to omit the initial rendering of this content.

The **<aside>** element defines some content aside from the content it is placed in (like a sidebar).

The <aside> content should be indirectly related to the surrounding content.

18. When should you use section, div or article?

**div**

The **<div>** element is the most general purpose element. It has no special meaning. It’s purpose is to group content that is not semantically related. Because it is essentially meaningless to screen readers, it should be sparingly used.

*Authors are strongly encouraged to view the div element as an element of last resort, for when no other element is suitable.*

[*W3C Recommendation*](http://www.w3.org/TR/html5/grouping-content.html#the-div-element%E2%80%9D%20target=%E2%80%9C_blank)

The **<div>** element, therefore, is mainly used to group content for targeting with CSS. For example, as a styling container for other elements.

**section**

The **<section>** element is slightly more specific that a **<div>**. It is applied to a generic section of content that *can* be grouped together in a semantically meaningful way.

*A general rule of thumb is that the section element is appropriate only if the element’s contents would be listed explicitly in the document’s outline.*

[*W3C Recommendation*](http://www.w3.org/TR/html5/sections.html#the-section-element%E2%80%9D%20target=%E2%80%9C_blank)

Because the contents of a **<section>** are meaningful when grouped together, they should have a “theme”. A **<section>**’s “theme” should be defined by including a heading element within the element, often immediately after the opening tag.

**article**

The **<article>** element is even more specific than a **<section>**. It is also applied to a section of content that is semantically related, and should also have a heading. However, this section of content should additionally be "self-contained". This means that the contents of an **<article>** should be able to be isolated from the rest of the page and still be meaningful.

One common purpose for an **<article>** is in marking content for syndication. For example, marking a blog post.

**div or section or article?**

So which should you use and when?

If the content within the element is not semantically related, then use a <div>. If the semantically related content is also able to be self-contained, then use an <article>. Otherwise, use a <section>.

19. What is Character Encoding?

A character encoding is a way to convert text data into binary numbers.

The HTML5 specification encourages web developers to use the UTF-8 character set, which covers almost all of the characters and symbols in the world!

20. What is the purpose of meta tags?

The <meta> tag defines **metadata** about an HTML document. Metadata is data (information) about data. <meta> tags always go inside the <head> element, and are typically used to specify character set, page description, keywords, author of the document, and viewport settings.

Metadata will not be displayed on the page, but is machine parsable.

Metadata is used by browsers (how to display content or reload page), search engines (keywords), and other web services.

There is a method to let web designers take control over the viewport (the user's visible area of a web page), through the <meta> tag.

21. What does async and defer refer in script tag? Describe the difference between <script>, <script

async> and <script defer>

The **defer**attribute is a boolean attribute.

When present, it specifies that the script is executed when the page has finished parsing.

**Note:** The defer attribute is only for external scripts (should only be used if the src attribute is present).

**Note:** There are several ways an external script can be executed:

* If **async** is present: The script is executed asynchronously with the rest of the page (the script will be executed while the page continues the parsing)
* If async is not present and **defer** is present: The script is executed when the page has finished parsing
* If neither async or defer is present: The script is fetched and executed immediately, before the browser continues parsing the page

22. Can you describe the difference between progressive enhancement and graceful degradation?

“**Mobilis in mobile**” — moving in a constantly changing environment

Just like Captain Nemo from “20,000 Leagues under the Sea”, web developers find themselves in a constantly changing and fluctuating environment that can be pretty hostile to what we try to achieve.

The web was invented and defined to be used with any display device, in any language, anywhere you want. The only thing expected of end users is that they are using a browsing device that can reach out to the web and understand the protocols used to transmit information — http, https, ftp and so on.

This means that we can’t expect anything of the setup or ability of our end users. We can also be fairly sure that our experience of the web as developers is totally different to the one of the people we want to reach.

So, **graceful degradation** is the practice of building your web functionality so that it provides a certain level of user experience in more modern browsers, but it will also *degrade gracefully* to a lower level of user in experience in older browsers. This lower level is not as nice to use for your site visitors, but it does still provide them with the basic functionality that they came to your site to use; things do not break for them.

**Progressive enhancement** is similar, but it does things the other way round. You start by establishing a basic level of user experience that all browsers will be able to provide when rendering your web site, but you also build in more advanced functionality that will automatically be available to browsers that can use it.

In other words, graceful degradation starts from the status quo of complexity and tries to fix for the lesser experience whereas progressive enhancement starts from a very basic, working example and allows for constant extension for future environments. Degrading gracefully means looking back whereas enhancing progressively means looking forward whilst keeping your feet on firm ground.

23. What is the purpose of cache busting and how can you achieve it?

When a static file gets cached it can be stored for very long periods of time before it ends up expiring. This can be an annoyance in the event that you make an update to a site however, since the cached version of the file is stored in your visitors' browsers, they **may be unable to see the changes made**. This is due to the fact that a visitor's browser will locally store a cached copy of your static assets given that your website is configured to [leverage browser caching](https://www.keycdn.com/support/leverage-browser-caching).

Cache busting solves the browser caching issue by using a unique file version identifier to tell the browser that a new version of the file is available. Therefore the browser doesn't retrieve the old file from cache but rather makes a request to the origin server for the new file.

Cache busting is useful because it allows your visitors to **receive the most recently updated files** without having to perform a hard refresh or clear their browser cache. From a developer's point of view, using cache busting is beneficial so that the latest changes can be pushed out and become available to everyone immediately. There are a few methods you can use to take advantage of cache busting:

* File name versioning (e.g. style.v2.css)
* File path versioning (e.g. /v2/style.css)
* Query strings (e.g. style.css?ver=2) (not recommended)

For whichever method you choose to use, ensure that once the file name or path is modified you also update the HTML which references said file.

24. Name 3 ways to decrease page load?

* Cache Your Web Pages, Optimize the caches
* Minimize image size (scale down w/ Photoshop rather than using width/height attributes inside html)
* [HTTP compression - Wikipedia](https://en.wikipedia.org/wiki/HTTP_compression)
* Minimize HTTP requests (fewer js/css files, using  [CSS sprites](https://www.webfx.com/blog/web-design/css-sprites-site-speed/) to combine images)
* Use CDN (Content Delivery Network)

25. What ARIA and screen readers are, and how to make a website accessible?

Accessible Rich Internet Applications **(ARIA)**is a set of attributes that define ways to make web content and web applications (especially those developed with JavaScript) more accessible to people with disabilities.

It supplements HTML so that interactions and widgets commonly used in applications can be passed to assistive technologies when there is not otherwise a mechanism. For example, ARIA enables accessible navigation landmarks in HTML4, JavaScript widgets, form hints and error messages, live content updates, and more.

Many of these widgets were later incorporated into HTML5, and **developers should prefer using the correct semantic HTML element over using ARIA**, if such an element exists. For instance, native elements have built-in [keyboard accessibility](https://developer.mozilla.org/en-US/docs/Web/Accessibility/Keyboard-navigable_JavaScript_widgets), roles and states. However, if you choose to use ARIA, you are responsible for mimicking (the equivalent) browser behavior in script.

**Note:** ARIA was invented after HTML4, so does not validate in HTML4 or its XHTML variants. However, the accessibility gains it provides far outweigh any technical invalidity. In HTML5, all ARIA attributes validate. The new landmark elements (<main>, <header>, <nav> etc) have built-in ARIA roles, so there is no need to duplicate them.

A screen reader is an assistive technology, primarily used by people with vision impairments. It converts text, buttons, images and other screen elements into speech or braille.

* Add Images with Alt Text
* Allow Users to Enlarge Font Sizes
* Keep Contrast Sensitivity in Mind
* Add Keyboard Navigation
* Make Video and Multimedia Accessible
* Give your links unique and descriptive names.
* Design tables carefully
* Use ARIA roles and landmarks (Remember, the first rule of ARIA is “Don’t use ARIA.” Many functions that used to require ARIA attributes are now implemented in HTML5.)

26. What is the purpose of the alt attribute on images?

The required alt attribute specifies an alternate text for an image, if the image cannot be displayed.

27. Explain some of the pros and cons for CSS animations versus JavaScript animations?

* **Use CSS when you have smaller, self-contained states for UI elements.** CSS transitions and animations are ideal for bringing a navigation menu in from the side, or showing a tooltip. You may end up using JavaScript to control the states, but the animations themselves will be in your CSS. Cannot do rotation/scale/position scale w/ different time.
* **Use JavaScript when you need significant control over your animations.** The Web Animations API is the standards-based approach, available today in most modern browsers. This provides real objects, ideal for complex object-oriented applications. JavaScript is also useful when you need to stop, pause, slow down, or reverse your animations.

28. What does CORS stand for and what issue does it address?

**Cross-Origin Resource Sharing** ([CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS)) is an [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP)-header based mechanism that allows a server to indicate any other [origin](https://developer.mozilla.org/en-US/docs/Glossary/Origin)s (domain, scheme, or port) than its own from which a browser should permit loading of resources. CORS also relies on a mechanism by which browsers make a “preflight” request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

29. Ways to improve website performance

30. Comparison of browsers engines like Chrome, Firefox, Internet explorer, Safari?

31. What does the lang attribute in html do?

32. What is desktop first and mobile first design approach?

33. What are data- attributes good for?

34. Explain the difference between layout, painting and compositing?

35. Explain about HTML Canvas?

36. Explain about HTML Layout Engines used by browsers?

37. What are the semantic tags available in html5?

38. Why you would like to use semantic tag?

39. How to make page responsive?

40. What is difference between span tag and div tag?

41. What are optional closing tag?

42. What is a self closing tag?

43. Does the following trigger http request at the time of page load?

44. How Geo-location API works in html5?

45. What is difference between SVG and Canvas?

46. Explain Drag and Drop in HTML5?

47. Why to use IndexedDB instead of WebSQL in HTML5?

48. Explain Application Cache in HTML5. OR What is a manifest file in HTML?

49. Explain Microdata in HTML5?

50. List the API available in HTML5?

51. What are different new form element types provided by HTML5?

52. What are the HTML tags which deprecated in HTML5?

53. How you can Use Modernizr in HTML5?

54. What is the use of WebSocket API?

55. What does enctype='multipart/form-data' mean?

56. What is progressive rendering?

57. What is difference between Select and Datalist?

58. What are data- attributes good for?

readyState – document

contentLoaded – document

onload – document

defer requires content load

$q built in service – promise

ng-hide

ng-show