**Q. Write a Javascript function which will emulate the functionality of a blink tag.**

**Ans.**

$(document).ready(function() { var f = document.getElementById('Foo'); setInterval(function() { f.style.display = (f.style.display == 'none' ? '' : 'none'); }, 1000); });

Or using Java Script.

<html>

<title></title>

<head>

<script>

function blink() {

var f = document.getElementById('Foo');

setInterval(function() {

f.style.display = (f.style.display == 'none' ? '' : 'none');

}, 1000);

}

</script>

</head>

<body onload="blink();">

<div id="Foo">Blink</div>

</body>

</html>

**Q. What is the difference between the GET and POST methods?**

**Ans.**

|  |  |  |
| --- | --- | --- |
| **GET** | **POST** |  |
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL  Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

**Q. Difference between complete and success method of jquery?**

**Ans.**

.success() only gets called if your webserver responds with a 200 OK HTTP header - basically when everything is fine.

However, .complete() will *always* get called no matter if the ajax call was successful or not - maybe it outputted errors and returned an error - .complete() will still get called.

It's worth mentioning that .complete() will get called *after* .success() gets called - if it matters to you.

**Q. How can you load 2 different versions of jquery on a page and work with both?**

**Ans.**

<!-- load jQuery 1.1.3 -->

<script type="text/javascript" src="http://example.com/jquery-1.1.3.js"></script> <script type="text/javascript"> var jQuery\_1\_1\_3 = $.noConflict(true); </script>

<!-- load jQuery 1.3.2 -->

<script type="text/javascript" src="http://example.com/jquery-1.3.2.js"></script> <script type="text/javascript"> var jQuery\_1\_3\_2 = $.noConflict(true); </script>

**Q. Implement an Image zoomer plugin in JavaScript? For exmp: - Dp page of junglee.com**

**Ans.**

<html>

<head>

<script type="text/javascript">

function resizeImg (img)

{

var resize = 150; // resize amount in percentage

var origH = 250; // original image height

var origW = 250; // original image width

var mouseX = event.x;

var mouseY = event.y;

var newH = origH \* (resize / 100);

var newW = origW \* (resize / 100);

// Set the new width and height

img.style.height = newH;

img.style.width = newW;

var c = img.parentNode;

// Work out the new center

c.scrollLeft = (mouseX \* (resize / 100)) - (newW / 2) / 2;

c.scrollTop = (mouseY \* (resize / 100)) - (newH / 2) / 2;

}

</script>

<style type="text/css">

#Container {

position:relative;

width:400px;

height:400px;

overflow:hidden;

}

</style>

</head>

<body>

<div id="Container">

<img alt="Click to zoom" onclick="resizeImg(this)"

src="logo.png" />

</div>

</body>

</html>

**Q. HOW DO YOU DEFINE A PRIVATE METHOD IN JS? HOW WILL YOU EXPLAIN IT TO A COLLEGE STUDENT WHO ONLY KNOWS JAVA?**

**Ans.**

Private

*Private* members are made by the constructor. Ordinary vars and parameters of the constructor becomes the private members.

function Container(param) {  
 this.member = param;  
 var secret = 3;  
 var that = this;  
}

This constructor makes three private instance variables: param, secret, and that. They are attached to the object, but they are not accessible to the outside, nor are they accessible to the object's own public methods. They are accessible to private methods. Private methods are inner functions of the constructor.

function Container(param) {  
  
 function dec() {  
 if (secret > 0) {  
 secret -= 1;  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 this.member = param;  
 var secret = 3;  
 var that = this;  
}

The private method dec examines the secret instance variable. If it is greater than zero, it decrements secret and returns true. Otherwise it returns false. It can be used to make this object limited to three uses.

By convention, we make a private that variable. This is used to make the object available to the private methods. This is a workaround for an error in the ECMAScript Language Specification which causes this to be set incorrectly for inner functions.

Private methods cannot be called by public methods. To make private methods useful, we need to introduce a privileged method.

**CSS and HTML**

**Q. Doc type of HTML5?**

**Ans.** The HTML5 doctype

<!DOCTYPE html>

The HTML 4.01 strict doctype

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">

**Q. What does Doctype do?**

**Ans.**

The DOCTYPE declaration, which should be the first tag in the source markup of any web page, is utilized by the web browser to identify the version of the markup language in which the page is written. It may contain a link to a Document Type Definition, or DTD for short. The DTD sets out the rules and grammar for that flavor of markup, thus enabling the browser to render the content accordingly.

**Q. What happens when Doctype missing?**

**Ans.** If the web page coding does not include a DOCTYPE Declaration (DTD or Document Type Declaration) or it is done incorrectly:

1. You will not be able to use a HTML (HyperText Markup Language) Validator to check the page coding. HTML validation requires the DOCTYPE declaration.
2. The browser rending the web page will process the coding in Quirks Mode.
3. The stylesheet may not be implemented as planned.

**Q. Semantic HTML, CSS Reset?**

**Ans.** Semantic's default theme includes the latest [Normalize CSS](http://necolas.github.io/normalize.css/) to provide a base line HTML reset. In addition, Semantic UI requires a Box-sizing reset, to make sure that elements handle width definitions in the same way.

Reset Options

The basic themes only includes the required **box-sizing** reset and nothing else. The resetcss theme provides a version of [Reset CSS](http://meyerweb.com/eric/tools/css/reset/), a less opinionated CSS reset.

A browser’s default styles can interfere with the styles we actually want to apply. That’s why a CSS reset has been devised to provide a consistent base across all browsers.

It’s particularly useful for normalizing the text size and removing all margins.

You can find (and download) the [HTML 5 reset.css](http://marksheet.io/css/reset.css) of this website. Just include it in your <head> before your own stylesheet.

<head>  
 <link rel="stylesheet" type="text/css" href="reset.css">  
 <link rel="stylesheet" type="text/css" href="styles.css">  
</head>

**Q. Explain about Box Model?**

**Ans.**

**CSS Box Model**

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

**Q. Different types of CSS positioning properties**

**Ans.** **There are four different position values:**

* **static**. HTML elements are positioned static by default.An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

div.static {

position: static;

border: 3px solid #73AD21;

}

* **relative**. An element with position: relative; is positioned relative to its normal position.

div.relative {

position: relative;

left: 30px;

border: 3px solid #73AD21;

}

* **fixed**. An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

div.fixed {

position: fixed;

bottom: 0;

right: 0;

width: 300px;

border: 3px solid #73AD21;

}

* **absolute**. An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

**Note:** A "positioned" element is one whose position is anything except static.

div.relative {

position: relative;

width: 400px;

height: 200px;

border: 3px solid #73AD21;

}

div.absolute {

position: absolute;

top: 80px;

right: 0;

width: 200px;

height: 100px;

border: 3px solid #73AD21;

}

**Q. Center align a box at the center of the page and some text center aligned inside that box. Assume box dimensions are 300X200.**

**Ans.**

<html>

<title></title>

<head>

<style>

div {

width: 300px;

height: 200px;

background-color: red;

position: absolute;

top:0;

bottom: 0;

left: 0;

right: 0;

margin: auto;

}

</style>

</head>

<body>

<div></div>

</body>

</html>

**Security Related:**

Q. **Security - Given a form that collects name/mobile number/company name on Amazon.in, what kind of attacks is it vulnerable to?**

Browser Compatibility

**Q. Explain Lifecycle of a webpage and also who how the rendering happens in a browser?**

**Ans.** <http://taligarsiel.com/Projects/howbrowserswork1.htm>

**Rendering a web page – step by step**

Have you ever thought about what happens when you surf the web? It’s not as simple as it seems:

1. You type an URL into address bar in your preferred browser.
2. The browser parses the URL to find the protocol, host, port, and path.
3. It forms a HTTP request (that was most likely the protocol)
4. To reach the host, it first needs to translate the human readable host into an IP number, and it does this by doing a DNS lookup on the host
5. Then a socket needs to be opened from the user’s computer to that IP number, on the port specified (most often port 80)
6. When a connection is open, the HTTP request is sent to the host
7. The host forwards the request to the server software (most often Apache) configured to listen on the specified port
8. The server inspects the request (most often only the path), and launches the server plugin needed to handle the request (corresponding to the server language you use, PHP, Java, .NET, Python?)
9. The plugin gets access to the full request, and starts to prepare a HTTP response.
10. To construct the response a database is (most likely) accessed. A database search is made, based on parameters in the path (or data) of the request
11. Data from the database, together with other information the plugin decides to add, is combined into a long string of text (probably HTML).
12. The plugin combines that data with some meta data (in the form of HTTP headers), and sends the HTTP response back to the browser.
13. The browser receives the response, and parses the HTML (which with 95% probability is broken) in the response
14. A DOM tree is built out of the broken HTML
15. New requests are made to the server for each new resource that is found in the HTML source (typically images, style sheets, and JavaScript files). Go back to step 3 and repeat for each resource.
16. Stylesheets are parsed, and the rendering information in each gets attached to the matching node in the DOM tree
17. Javascript is parsed and executed, and DOM nodes are moved and style information is updated accordingly
18. The browser renders the page on the screen according to the DOM tree and the style information for each node
19. You see the page on the screen
20. You get annoyed the whole process was too slow.

**Q. How do you optimise a web page on client side?**

**Ans:**

5 steps for optimizing the load time of web pages

**1. Get clear on how you want your site to work.**

This requires a closer investigation of your target audience, their needs, and how well your site meets those needs.

For example, if most of your pages load in two thirds of a second or less, and your target audience is a typical retail consumer, declare victory. You get to turn your attention and resources to other tasks such as adding new functionality. If, on the other hand, a desktop or mobile browser churns for ten seconds before a customer sees your page, roll up your sleeves and keep reading.

**2. Stop throwing money at the wrong problem.**

The good news: Accurate, low-cost monitors for page loading are readily available online. First, check with your server hosting provider; you might already be paying for this service. Once you land a monitor you like, measure the bottlenecks that matter for your site.

Three broad domains deal specifically with a website’s load-time delays. When it comes to how quickly a website renders and how well it delivers web applications, these include:

Server computation.

Network transmission.

Browser rendering.

**3. Be mindful of mobile.**

Check how your site renders on mobile devices. Folklore abounds with websites that are lightning fast when viewed with top-end equipment in the developer’s office, but fall apart in a customer’s hands when trying to access the same site with a smartphone or tablet. Paying attention to mobile performance means more than taking into account unpredictable transmission systems. You must also consider slow mobile browsers whose performance varies from device to device.

**4. Manage your image sizes.**

Take images, for example. If image transmission is your site’s biggest problem, you have a slew of solutions to choose from. Let’s say the servers deliver content snappily, and pages are easy to load, but measurements reveal that it takes many seconds to pass large or complex images. This not only slows down the data transmission, but also bogs down the browser’s rendering engine. Here’s what you can do to remedy the problem:

Translate images from formats like PNG or JPEG into Scalable Vector Graphics (SVG) to shrink their size dramatically, as well as make them more portable across different display devices and screen sizes.

Convert large images into a lower-quality format, such as using a more-compressed JPEG encoding scheme.

“Lazy-load” images so that readers begin to see your page immediately, with its graphic content arriving in the background.

Redesign the page to require only a fraction as much graphics content.

Implement a client-side caching scheme; while the initial page loading might remain pokey, navigation around the site is snappy.

Host images separately from HTML and JavaScript text to accelerate browser responsiveness in common situations.

Improve page-loading speed significantly with browser re-load, pre-fetch, and pre-render hints on occasion.

Recode JavaScript in specific ways to help browser pre-loaders see images and load them earlier.

Use a Content Delivery Network to stage the images at network sites near end-users. This only works well for the highest-volume websites. Otherwise, it’s an expensive and impractical option.

Always be guided by current measurements of your pages’ actual performance, and make sure any optimizations you undertake aim directly at true bottlenecks. That way you increase your chances of significantly improving what your users see today.

Lastly, always keep in mind there might be more than one big problem to solve. For example, sometimes a browser hint or two will nearly double page-load speeds. Or other times, one has to break through four or five bottlenecks in succession to reach adequate performance. When trying to optimize the load time of your site, you must consider the bigger picture. No pun intended.

Excellent books such as Peter Smith’s Professional Website Performance: Optimizing the Front-End and Back-End, are devoted to those sorts of coding tips, tricks and techniques.

**5. Examine JavaScript coding.**

Nearly all modern web pages process thousands of lines of JavaScript for advertisement management, Web analytics, client-side interactions, multimedia controls, and more. Web frameworks “churn” so fast that few have matured enough for true best practices to be well known.

For instance, think about jQuery — perhaps the most widely used JavaScript library. In jQuery, use of “… innerHTML is massively faster …” than the corresponding jquery.html(), in the words of Google Chrome Tooling specialist Paul Irish. That said, millions of web pages use the latter, despite the performance hit. To wring out such inefficiencies, you’ll need to do a performance audit, like the one a Google team did for Wikipedia in the write-up quoted immediately above.

Auditing at this level — whether of JavaScript, downloaded elements, or server-side computations — demands considerable experience and breadth of knowledge. Unfortunately, overnight success in not guaranteed. However, depending on the level of your expertise, try to cycle through the following steps:

**Run a good page-speed profiler.**

Compare the results to your organization’s business requirements for the pages in question.

Use the profiler’s automated hints and suggestions as starting points for alternative codings.

Measure your experiments and ruled by measurements, not subjective impressions.

Working with a budget that won’t budge

Even if you’re on a tight budget, you still have options for improving site speed. In fact, it’s been my experience that every — and I mean every — team that uses a page-speed profiler, at all, systematically discovers at least one surprise with the potential to improve speed by at least 20 percent. And that’s without any fees or loss of end-user functionality.

Of course, if you have the means to hire help, you can still cut costs by getting a clear picture of what you have and what you want. The first step is deciding by how much you want to enhance user experience. Got questions on how to optimize load time of your web pages? Connect with GoDaddy’s community of developers just like you in the comments section below.

**Q. HOW WOULD YOU IMPLEMENT A PAGE SIMILAR TO AMAZON.IN "DEALS" PAGE, WITH 60 ROWS AND 4 PRODUCT DEALS PER COLUMNS? HOW WOULD YOU IMPLEMENT THE "ADD TO CART" BUTTON FUNCTIONALITY?**

**Q. IMPLEMENT A SIMPLE WEB APPLICATION CALLED "HTTP REDIRECT CHECKER" THAT CHECKS WHETHER A GIVEN URL REDIRECTS TO ANOTHER PAGE OR NOT ?**

**Q. IMPLEMENT AN EXCEL-LIKE WORKSHEET IN JAVASCRIPT THAT SUPPORTS THE "SUM" FUNCTION**