**50 Common Front End Developer Interview Questions [2019 Edition]**

*Front-end development involves the building of webpages and user interfaces for web applications. Front-end developers implement the structure, design, behavior, and animation of everything you see on the screen when you open up websites, web applications, or mobile apps.*

**Top of Form**

**Bottom of Form**

**Q1: Explain the CSS “box model” and the layout components that it consists of**

Topic: **CSS**  
Difficulty: ⭐⭐

The CSS box model is a rectangular layout paradigm for HTML elements that consists of the following:

* **Content** - The content of the box, where text and images appear
* **Padding** - A transparent area surrounding the content (i.e., the amount of space between the border and the content)
* **Border** - A border surrounding the padding (if any) and content
* **Margin** - A transparent area surrounding the border (i.e., the amount of space between the border and any neighboring elements)

**Q2: What is a CSS rule?**

Topic: **CSS**  
Difficulty: ⭐⭐

Web browsers apply **CSS rules** to a document to affect how they are displayed. A CSS rule is formed from:

* A **set of properties**, which have values set to update how the HTML content is displayed,
* A **selector**, which selects the element(s) you want to apply the updated property values to.

A set of CSS rules contained within a stylesheet determines how a webpage should look.

**Q3: What is Sass?**

Topic: **CSS**  
Difficulty: ⭐⭐

**Sass** or **Syntactically Awesome StyleSheets** is a *CSS* preprocessor that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly.

A *CSS preprocessor* is a scripting language that extends CSS by allowing developers to write code in one language and then compile it into CSS.

**Q4: What is a Mixin and how to use on?**

Topic: **CSS**  
Difficulty: ⭐⭐⭐

A **Mixin** is a block of code that lets us group CSS declarations we may reuse throughout our site.

To define mixin:

@mixin grid($flex: true /\*default argument\*/) {

@if $flex {

@include flex;

} @else {

display: block;

}

}

To use a Mixin, we simply use @include followed by the name of the Mixin and a semi-colon.

/\*scss\*/

.row {

@include grid(true);

}

/\*css\*/

.row {

display: -webkit-flex;

display: flex;

}

**Q5: What’s the difference between “resetting” and “normalizing” CSS? Which would you choose, and why?**

Topic: **CSS**  
Difficulty: ⭐⭐⭐

* **Resetting** — is meant to strip all default browser styling on elements. For e.g. margins, paddings, font-sizes of all elements are reset to be the same. You will have to redeclare styling for common typographic elements.
* **Normalizing** — preserves useful default styles rather than “unstyling” everything. It also corrects bugs for common browser dependencies.

It's a good idea to choose resetting when you have very a customized or unconventional site design such that I need to do a lot of my own styling do not need any default styling to be preserved.

**Q6: What is a Grid System in CSS?**

Topic: **CSS**  
Difficulty: ⭐⭐⭐

A grid system is a structure that allows for content to be stacked both vertically and horizontally in a consistent and easily manageable fashion. Grid systems include two key components: rows and columns.

Some Grid Systems:

* Simple Grid
* Pure
* Flexbox Grid
* Bootstrap
* Foundation

**Q7: Explain meta tags in HTML**

Topic: **HTML5**  
Difficulty: ⭐

* **Meta tags** always go inside **head tag** of the HTML page
* **Meta tags** is always passed as name/value pairs
* **Meta tags** are not displayed on the page but intended for the browser
* **Meta tags** can contain information about **character encoding**, **description**, **title** of the document etc,

**Example**:

<!DOCTYPE html>

<html>

<head>

<meta name="description" content="I am a web page with description">

<title>Home Page</title>

</head>

<body>

</body>

</html>

**Q8: What is the difference between span and div?**

Topic: **HTML5**  
Difficulty: ⭐⭐

* div is a block element
* span is inline element

For bonus points, you could point out that it’s illegal to place a block element inside an inline element, and that while div can have a p tag, and a p tag can have a span, it is not possible for span to have a div or p tag inside.

**Q9: What are defer and async attributes on a <script> tag?**

Topic: **HTML5**  
Difficulty: ⭐⭐⭐

If neither attribute is present, the script is downloaded and executed synchronously, and will halt parsing of the document until it has finished executing (default behavior). Scripts are downloaded and executed in the order they are encountered.

The defer attribute downloads the script while the document is still parsing but waits until the document has finished parsing before executing it, equivalent to executing inside a DOMContentLoaded event listener. defer scripts will execute in order.

The async attribute downloads the script during parsing the document but will pause the parser to execute the script before it has fully finished parsing. async scripts will not necessarily execute in order.

Note: both attributes must only be used if the script has a src attribute (i.e. not an inline script).

<script src="myscript.js"></script>

<script src="myscript.js" defer></script>

<script src="myscript.js" async></script>

**Q10: What's new in HTML 5?**

Topic: **HTML5**  
Difficulty: ⭐⭐⭐

HTML 5 adds a lot of new features to the HTML specification

**New Doctype**

Still using that pesky, impossible-to-memorize XHTML doctype?

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

If so, why? Switch to the new HTML5 doctype. You'll live longer -- as Douglas Quaid might say.

<!DOCTYPE html>

**New Structure**

* <section> - to define sections of pages
* <header> - defines the header of a page
* <footer> - defines the footer of a page
* <nav> - defines the navigation on a page
* <article> - defines the article or primary content on a page
* <aside> - defines extra content like a sidebar on a page
* <figure> - defines images that annotate an article

**New Inline Elements**

These inline elements define some basic concepts and keep them semantically marked up, mostly to do with time:

* <mark> - to indicate content that is marked in some fashion
* <time> - to indicate content that is a time or date
* <meter> - to indicate content that is a fraction of a known range - such as disk usage
* <progress> - to indicate the progress of a task towards completion

**New Form Types**

* <input type="datetime">
* <input type="datetime-local">
* <input type="date">
* <input type="month">
* <input type="week">
* <input type="time">
* <input type="number">
* <input type="range">
* <input type="email">
* <input type="url">

**New Elements**

There are a few exciting new elements in HTML 5:

* <canvas> - an element to give you a drawing space in JavaScript on your Web pages. It can let you add images or graphs to tool tips or just create dynamic graphs on your Web pages, built on the fly.
* <video> - add video to your Web pages with this simple tag.
* <audio> - add sound to your Web pages with this simple tag.

**No More Types for Scripts and Links**

You possibly still add the type attribute to your link and script tags.

<link rel="stylesheet" href="path/to/stylesheet.css" type="text/css" />

<script type="text/javascript" src="path/to/script.js"></script>

This is no longer necessary. It's implied that both of these tags refer to stylesheets and scripts, respectively. As such, we can remove the type attribute all together.

<link rel="stylesheet" href="path/to/stylesheet.css" />

<script src="path/to/script.js"></script>

**Make your content editable**

The new browsers have a nifty new attribute that can be applied to elements, called contenteditable. As the name implies, this allows the user to edit any of the text contained within the element, including its children. There are a variety of uses for something like this, including an app as simple as a to-do list, which also takes advantage of local storage.

<h2> To-Do List </h2>

<ul contenteditable="true">

<li> Break mechanical cab driver. </li>

<li> Drive to abandoned factory

<li> Watch video of self </li>

</ul>

**Attributes**

* require to mention the form field is required
* autofocus puts the cursor on the input field

**Q11: What is Coercion in JavaScript?**

Topic: **JavaScript**  
Difficulty:

In JavaScript conversion between different two build-in types called coercion. Coercion comes in two forms in JavaScript: *explicit* and *implicit*.

Here's an example of explicit coercion:

var a = "42";

var b = Number( a );

a; // "42"

b; // 42 -- the number!

And here's an example of implicit coercion:

var a = "42";

var b = a \* 1; // "42" implicitly coerced to 42 here

a; // "42"

b; // 42 -- the number!

**Q12: What is Scope in JavaScript?**

Topic: **JavaScript**  
Difficulty: ⭐

In JavaScript, each function gets its own *scope*. Scope is basically a collection of variables as well as the rules for how those variables are accessed by name. Only code inside that function can access that function's scoped variables.

A variable name has to be unique within the same scope. A scope can be nested inside another scope. If one scope is nested inside another, code inside the innermost scope can access variables from either scope.

**Q13: What is strict mode?**

Topic: **JavaScript**  
Difficulty: ⭐⭐

*Strict Mode* is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This strict context prevents certain actions from being taken and throws more exceptions.

// Non-strict code...

(function(){

"use strict";

// Define your library strictly...

})();

// Non-strict code...

**Q14: What is IIFEs (Immediately Invoked Function Expressions)?**

Topic: **JavaScript**  
Difficulty: ⭐⭐⭐

It’s an Immediately-Invoked Function Expression, or IIFE for short. It executes immediately after it’s created:

(function IIFE(){

console.log( "Hello!" );

})();

// "Hello!"

This pattern is often used when trying to avoid polluting the global namespace, because all the variables used inside the IIFE (like in any other normal function) are not visible outside its scope.

**Q15: What is “closure” in javascript? Provide an example?**

Topic: **JavaScript**  
Difficulty: ⭐⭐⭐⭐

A *closure* is a function defined inside another function (called parent function) and has access to the variable which is declared and defined in parent function scope.

The closure has access to variable in three scopes:

* Variable declared in his own scope
* Variable declared in parent function scope
* Variable declared in global namespace

var globalVar = "abc";

// Parent self invoking function

(function outerFunction (outerArg) { // begin of scope outerFunction

// Variable declared in outerFunction function scope

var outerFuncVar = 'x';

// Closure self-invoking function

(function innerFunction (innerArg) { // begin of scope innerFunction

// variable declared in innerFunction function scope

var innerFuncVar = "y";

console.log(

"outerArg = " + outerArg + "\n" +

"outerFuncVar = " + outerFuncVar + "\n" +

"innerArg = " + innerArg + "\n" +

"innerFuncVar = " + innerFuncVar + "\n" +

"globalVar = " + globalVar);

// end of scope innerFunction

})(5); // Pass 5 as parameter

// end of scope outerFunction

})(7); // Pass 7 as parameter

innerFunction is closure which is defined inside outerFunction and has access to all variable which is declared and defined in outerFunction scope. In addition to this function defined inside function as closure has access to variable which is declared in global namespace.

Output of above code would be:

outerArg = 7

outerFuncVar = x

innerArg = 5

innerFuncVar = y

globalVar = abc

**Q16: Explain the Prototype Design Pattern**

Topic: **JavaScript**  
Difficulty: ⭐⭐⭐⭐

*The Prototype Pattern* creates new objects, but rather than creating non-initialized objects it returns objects that are initialized with values it *copied* from a prototype - or sample - object. The Prototype pattern is also referred to as the Properties pattern.

An example of where the Prototype pattern is useful is the initialization of business objects with values that match the default values in the database. The prototype object holds the default values that are copied over into a newly created business object.

Classical languages rarely use the Prototype pattern, but JavaScript being a prototypal language uses this pattern in the construction of new objects and their prototypes.

**Q17: Compare SQL databases and MongoDB at a high level.**

Topic: **MongoDB**  
Difficulty: ⭐⭐

SQL databases store data in form of tables, rows, columns and records. This data is stored in a pre-defined data model which is not very much flexible for today's real-world highly growing applications. MongoDB in contrast uses a flexible structure which can be easily modified and extended.

**Q18: What is Node.js?**

Topic: **Node.js**  
Difficulty: ⭐

Node.js is a web application framework built on Google Chrome's JavaScript Engine (V8 Engine).

Node.js comes with runtime environment on which a Javascript based script can be interpreted and executed (It is analogus to JVM to JAVA byte code). This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well.

*Node.js = Runtime Environment + JavaScript Library*

**Q19: What is npm?**

Topic: **Node.js**  
Difficulty: ⭐

npm stands for Node Package Manager. npm provides following two main functionalities:

* Online repositories for node.js packages/modules which are searchable on [search.nodejs.org](http://search.nodejs.org/)
* Command line utility to install packages, do version management and dependency management of Node.js packages.

**Q20: If Node.js is single threaded then how it handles concurrency?**

Topic: **Node.js**  
Difficulty: ⭐⭐

Node provides a single thread to programmers so that code can be written easily and without bottleneck. Node internally uses multiple POSIX threads for various I/O operations such as File, DNS, Network calls etc.

When Node gets I/O request it creates or uses a thread to perform that I/O operation and once the operation is done, it pushes the result to the event queue. On each such event, event loop runs and checks the queue and if the execution stack of Node is empty then it adds the queue result to execution stack.

This is how Node manages concurrency.

**Q21: What is Callback Hell?**

Topic: **Node.js**  
Difficulty: ⭐⭐

The asynchronous function requires callbacks as a return parameter. When multiple asynchronous functions are chained together then callback hell situation comes up.

**Q22: How can you avoid callback hells?**

Topic: **Node.js**  
Difficulty: ⭐⭐⭐

To do so you have more options:

* **modularization**: break callbacks into independent functions
* use *Promises*
* use yield with *Generators* and/or *Promises*

**Q23: How would you scale Node application?**

Topic: **Node.js**  
Difficulty: ⭐⭐⭐⭐⭐

We can scale Node application in following ways:

* cloning using *Cluster* module.
* decomposing the application into smaller services – i.e micro services.

**Q24: What is encapsulation?**

Topic: **OOP**  
Difficulty: ⭐⭐

**Encapsulation** is defined *as the process of enclosing one or more items within a physical or logical package*. Encapsulation, in object oriented programming methodology, prevents access to implementation details.

**Q25: What is polymorphism?**

Topic: **OOP**  
Difficulty: ⭐⭐

The word polymorphism means having many forms. In object-oriented programming paradigm, polymorphism is often expressed as *one interface, multiple functions*.

**Q26: What is Stress Testing?**

Topic: **Software Testing**  
Difficulty: ⭐⭐⭐

**Stress testing** — also known as fatigue testing — is meant to measure system performance outside of the parameters of normal working conditions. The software is given more users or transactions that can be handled. The goal of stress testing is to measure the software stability. At what point does software fail, and how does the software recover from failure?

**Q27: Name some Performance Testing best practices**

Topic: **Software Testing**  
Difficulty: ⭐⭐⭐

* Test as early as possible in development.
* Conduct multiple performance tests to ensure consistent findings and determine metrics averages.
* Test the individual software units separately as well as together
* Baseline measurements provide a starting point for determining success or failure
* Performance tests are best conducted in test environments that are as close to the production systems as possible
* Isolate the performance test environment from the environment used for quality assurance testing
* Keep the test environment as consistent as possible
* Calculating averages will deliver actionable metrics. There is value in tracking outliers also. Those extreme measurements could reveal possible failures.

**Q28: What is Spike Testing?**

Topic: **Software Testing**  
Difficulty: ⭐⭐⭐⭐

**Spike testing** is a type of stress testing that evaluates software performance when workloads are substantially increased quickly and repeatedly. The workload is beyond normal expectations for short amounts of time.

**Q29: What is ReactJS?**

Topic: **React**  
Difficulty: ⭐

ReactJS is an **open-source frontend JavaScript library** which is used for building user interfaces especifically for single page applications. It is used for handling view layer for web and mobile apps. React was created by Jordan Walke, a software engineer working for Facebook. ReactJS was first deployed on Facebook’s newsfeed in 2011 and on Instagram.com in 2012.

**Q30: What is the point of Redux?**

Topic: **React**  
Difficulty: ⭐⭐

Application state management that is easy to reason about, maintain and manage in an asynchronous web application environment.

**Q31: What is Flux?**

Topic: **React**  
Difficulty: ⭐⭐

Unidrectional application flow paradigm popular a few years back in React; mostly superceded by Redux these days.

**Q32: What are advantages of REST web services?**

Topic: **SOA & REST API**  
Difficulty: ⭐⭐

Some of the advantages of REST web services are:

* Learning curve is easy since it works on HTTP protocol
* Supports multiple technologies for data transfer such as text, xml, json, image etc.
* No contract defined between server and client, so loosely coupled implementation.
* REST is a lightweight protocol
* REST methods can be tested easily over browser.

**Q33: Mention what is the difference between PUT and POST?**

Topic: **SOA & REST API**  
Difficulty: ⭐⭐⭐

*PUT* puts a file or resource at a particular URI and exactly at that URI. If there is already a file or resource at that URI, PUT changes that file or resource. If there is no resource or file there, PUT makes one

*POST* sends data to a particular URI and expects the resource at that URI to deal with the request. The web server at this point can decide what to do with the data in the context of specified resource

*PUT is idempotent* meaning, invoking it any number of times will not have an impact on resources.

However, *POST is not idempotent*, meaning if you invoke POST multiple times it keeps creating more resources

**Q34: Name some best practices for better RESTful API design**

Topic: **SOA & REST API**  
Difficulty: ⭐⭐⭐⭐⭐

* Use nouns and HTTP methods but no verbs
* GET /cars
* POST /cars
* DELETE /cars/:id
* instead
* GET /getAllCars
* POST /createNewCar
* GET /deleteAllRedCars
* GET method and query parameters should not alter the state
* Use plural nouns
* /cars instead of /car
* /users instead of /user
* /products instead of /product
* /settings instead of /setting
* Use sub-resources for relations
* GET /cars/711/drivers/ Returns a list of drivers for car 711
* GET /cars/711/drivers/4 Returns driver #4 for car 711
* Use HTTP headers for serialisation formats
* Content-Type defines the request format.
* Accept defines a list of acceptable response formats.
* Use **HATEOAS** - Hypermedia as the Engine of Application State is a principle that hypertext links should be used to create a better navigation through the API.
* {
* "id": 711,
* "manufacturer": "bmw",
* "model": "X5",
* "seats": 5,
* "drivers": [{
* "id": "23",
* "name": "Stefan Jauker",
* "links": [{
* "rel": "self",
* "href": "/api/v1/drivers/23"
* }]
* }]
* }
* Use appropriate HTTP response status codes
  + **2xx** (Success category)
  + **3xx** (Redirection Category)
  + **4xx** (Client Error Category)
  + **5xx** (Server Error Category)
* Provide filtering, sorting, field selection and paging for collections
* GET /cars?color=red Returns a list of red cars
* GET /cars?seats<=2 Returns a list of cars with a maximum of 2 seats
* Version your API
* /blog/api/v1
* Use error payloads - All exceptions should be mapped in an error payload.
* {
* "errors": [{
* "userMessage": "Sorry, the requested resource does not exist",
* "internalMessage": "No car found in the database",
* "code": 34,
* "more info": "http://dev.mwaysolutions.com/blog/api/v1/errors/12345"
* }]
* }
* Allow overriding HTTP method

In certain situations (for example, when the service or its consumers are behind an overzealous corporate firewall, or if the main consumer is a web page), only the GET and POST HTTP methods might be available. In such a case, it is possible to emulate the missing verbs by passing a custom header in the requests. To support a RESTful API with these limitations, the API needs a way to override the HTTP method and use the custom HTTP Header X-HTTP-Method-Override to map the request to an appropriate API method.

**Q35: What is meant by the KISS principle?**

Topic: **Software Architecture**  
Difficulty: ⭐⭐

**KISS**, a backronym for "keep it simple, stupid", is a design principle noted by the U.S. Navy in 1960. The KISS principle states that most systems work best if they are kept simple rather than made complicated; therefore simplicity should be a key goal in design, and that unnecessary complexity should be avoided.

**Q36: What Is Load Balancing?**

Topic: **Software Architecture**  
Difficulty: ⭐⭐⭐

**Load balancing** is simple technique for distributing workloads across multiple machines or clusters. The most common and simple load balancing algorithm is Round Robin. In this type of load balancing the request is divided in circular order ensuring all machines get equal number of requests and no single machine is overloaded or underloaded.

**The Purpose of load balancing is to**

* Optimize resource usage (avoid overload and under-load of any machines)
* Achieve Maximum Throughput
* Minimize response time

**Most common load balancing techniques in web based applications are**

1. Round robin
2. Session affinity or sticky session
3. IP Address affinity

**Q37: What does SOLID stand for? What are its principles?**

Topic: **Software Architecture**  
Difficulty: ⭐⭐⭐

**S.O.L.I.D** is an acronym for the first five object-oriented design (OOD) principles by Robert C. Martin.

* **S** - *Single-responsiblity principle*. A class should have one and only one reason to change, meaning that a class should have only one job.
* **O** - *Open-closed principle*. Objects or entities should be open for extension, but closed for modification.
* **L** - *Liskov substitution principle*. Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.
* **I** - *Interface segregation principle*. A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.
* **D** - *Dependency Inversion Principle*. Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions.

**Q38: Are you familiar with The Twelve-Factor App principles?**

Topic: **Software Architecture**  
Difficulty: ⭐⭐⭐⭐⭐

The **Twelve-Factor App** methodology is a methodology for building software as a service applications. These best practices are designed to enable applications to be built with portability and resilience when deployed to the web.

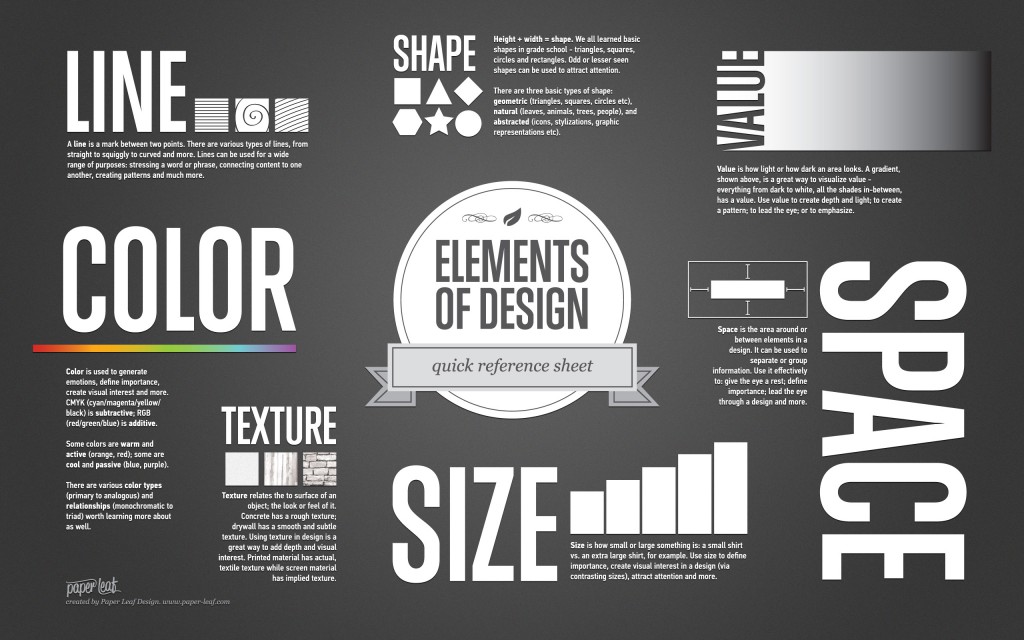
* **Codebase** - There should be exactly one codebase for a deployed service with the codebase being used for many deployments.
* **Dependencies** - All dependencies should be declared, with no implicit reliance on system tools or libraries.
* **Config** - Configuration that varies between deployments should be stored in the environment.
* **Backing services** All backing services are treated as attached resources and attached and detached by the execution environment.
* **Build, release, run** - The delivery pipeline should strictly consist of build, release, run.
* **Processes** - Applications should be deployed as one or more stateless processes with persisted data stored on a backing service.
* **Port binding** - Self-contained services should make themselves available to other services by specified ports.
* **Concurrency** - Concurrency is advocated by scaling individual processes.
* **Disposability** - Fast startup and shutdown are advocated for a more robust and resilient system.
* **Dev/Prod parity** - All environments should be as similar as possible.
* **Logs** - Applications should produce logs as event streams and leave the execution environment to aggregate.
* **Admin Processes** - Any needed admin tasks should be kept in source control and packaged with the application.

**Q39: Name some basic design elements**

Topic: **UX Design**  
Difficulty: ⭐

The elements of design are:

* **LINE** – The linear marks made with a pen or brush or the edge created when two shapes meet.
* **SHAPE** – A shape is a self contained defined area of geometric (squares and circles), or organic (free formed shapes or natural shapes). A positive shape automatically creates a negative shape.
* **DIRECTION** – All lines have direction – Horizontal, Vertical or Oblique. Horizontal suggests calmness, stability and tranquillity. Vertical gives a feeling of balance, formality and alertness. Oblique suggests movement and action
* **SIZE** – Size is simply the relationship of the area occupied by one shape to that of another.
* **TEXTURE** – Texture is the surface quality of a shape – rough, smooth, soft hard glossy etc.
* **COLOUR** – Colour is light reflected off objects. Color has three main characteristics: hue or its name (red, green, blue, etc.), value (how light or dark it is), and intensity (how bright or dull it is).



**Q40: What is User Centered Design?**

Topic: **UX Design**  
Difficulty: ⭐⭐

**User-centered design** is an iterative design process in which designers focus on the users and their needs in each phase of the design process. UCD calls for involving users throughout the design process via a variety of research and design techniques so as to create highly usable and accessible products for them.

User-centered design demands that designers employ a mixture of *investigative* (e.g., surveys and interviews) and *generative* (e.g., brainstorming) methods and tools to develop an understanding of user needs.

**Q41: Name fundamental principles of design**

Topic: **UX Design**  
Difficulty: ⭐⭐⭐

The fundamental principles of design are:

* **BALANCE** — Balance in design is similar to balance in physics. A large shape close to the center can be balanced by a small shape close to the edge. Balance provides stability and structure to a design. It’s the weight distributed in the design by the placement of your elements.
* **PROXIMITY** — Proximity creates a relationship between elements. It provides a focal point. Proximity doesn’t mean that elements have to be placed together, it means they should be visually connected in some way.
* **ALIGNMENT** — Allows us to create order and organization. Aligning elements allows them to create a visual connection with each other.
* **REPETITION** — Repetition strengthens a design by tying together individual elements. It helps to create association and consistency. Repetition can create rhythm (a feeling of organized movement).
* **CONTRAST** — Contrast is the juxtaposition of opposing elements (opposite colors on the color wheel, or value light/dark, or direction — horizontal/vertical). Contrast allows us to emphasize or highlight key elements in your design.
* **SPACE** — Space in art refers to the distance or area between, around, above, below, or within elements. Both positive and negative space are important factors to be considered in every design.

**Q42: What is SQL injection?**

Topic: **Web Security**  
Difficulty: ⭐

Injection attacks stem from a lack of strict separation between program instructions (i.e., code) and user-provided (or external) input. This allows an attacker to inject malicious code into a data snippet.

*SQL injection* is one of the most common types of injection attack. To carry it out, an attacker provides malicious SQL statements through the application.

How to prevent:

* **Prepared statements with parameterized queries**
* **Stored procedures**
* **Input validation** - blacklist validation and whitelist validation
* **Principle of least privilege** - Application accounts shouldn’t assign DBA or admin type access onto the database server. This ensures that if an application is compromised, an attacker won’t have the rights to the database through the compromised application.

**Q43: What is Cross-Site Scripting (XSS)?**

Topic: **Web Security**  
Difficulty: ⭐⭐

Cross-Site Scripting (XSS) is an attack that occurs when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

The page provided by the server when someone requests it is unaltered. Instead, an XSS attack exploits a weakness in a page that include a variable submitted in a request to show up in raw form in the response. The page is only reflecting back what was submitted in that request.

**Q44: What is Content Security Policy?**

Topic: **Web Security**  
Difficulty: ⭐⭐

**Content Security Policy (CSP)** is an HTTP header that allows site operators fine-grained control over where resources on their site can be loaded from. The use of this header is the best method to prevent cross-site scripting (XSS) vulnerabilities. Due to the difficulty in retrofitting CSP into existing websites, CSP is mandatory for all new websites and is strongly recommended for all existing high-risk sites.

The primary benefit of CSP comes from disabling the use of unsafe inline JavaScript. Inline JavaScript – either reflected or stored – means that improperly escaped user-inputs can generate code that is interpreted by the web browser as JavaScript. By using CSP to disable inline JavaScript, you can effectively eliminate almost all XSS attacks against your site.

**Q45: What is ClickJacking?**

Topic: **Web Security**  
Difficulty: ⭐⭐⭐

**ClickJacking** is an attack that fools users into thinking they are clicking on one thing when they are actually clicking on another. The attack is possible thanks to HTML frames (iframes).

Its other name, user interface (UI) redressing, better describes what is going on. Users think they are using a web page’s normal UI, but in fact there is a hidden UI in control; in other words, the UI has been redressed. When users click something they think is safe, the hidden UI performs a different action.

**Q46: What is webpack?**

Topic: **Webpack**  
Difficulty: ⭐

**Webpack** is a build tool that puts all of your assets, including Javascript, images, fonts, and CSS, in a dependency graph. Webpack lets you use require() in your source code to point to local files, like images, and decide how they're processed in your final Javascript bundle, like replacing the path with a URL pointing to a CDN.

**Q47: Why and when should I Use Webpack?**

Topic: **Webpack**  
Difficulty: ⭐

If you're building a complex Front End application with many **non-code static assets** such as CSS, images, fonts, etc, then yes, Webpack will give you great benefits.

If your application is fairly small, and you don't have many static assets and you only need to build one Javascript file to serve to the client, then Webpack might be more overhead than you need.

**Q48: Why do we use jQuery?**

Topic: **jQuery**  
Difficulty: ⭐⭐

Due to following advantages.

* Easy to use and learn.
* Easily expandable.
* Cross-browser support (IE 6.0+, FF 1.5+, Safari 2.0+, Opera 9.0+)
* Easy to use for DOM manipulation and traversal.
* Large pool of built in methods.
* AJAX Capabilities.
* Methods for changing or applying CSS, creating animations.
* Event detection and handling.
* Tons of plug-ins for all kind of needs.

**Q49: How JavaScript and jQuery are different?**

Topic: **jQuery**  
Difficulty: ⭐⭐

JavaScript is a language While jQuery is a library built in the JavaScript language that helps to use the JavaScript language.

**Q50: When would you use AngularJS vs jQuery?**

Topic: **jQuery**  
Difficulty: ⭐⭐⭐

* **jQuery** - is a library used for DOM Manipulations - Has nothing to do with models - don't have two-way binding feature - becomes complex and difficult to maintain when size of project increases - Sometimes you have to write more code to achieve the same functionality as in Angular
* **Angular** - is a MVVM Framework - Used for creating SPA (Single Page Applications) - Has key features like routing, directives, two way data binding, models, dependency injection, unit tests etc - is modular - Maintainable, when project size increases - is Fast and many more.

Basically jQuery is a single tool (solves one specific problem: dom manipulation) where AngularJS is a whole toolbox with all kind of tools for different problems (routing, modelbindings, dom manipulation, etc.). Actually jqLite (subset of jQuery) is part of the AngularJS and you use it to solve the dom-manipulation thing.