------------------------------------------------------------------------------

HTML & CSS

-------------------------------------------------------------------------------

What is an example of a block element and an inline element? What is the difference between the two? In what situation would you use one over the other?

Ans. Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

The <div> element is a block-level element.

Examples of block-level elements:

* <div>
* <h1> - <h6>
* <p>
* <form>

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary.

This is an inline <span> element inside a paragraph.

Examples of inline elements:

* <span>
* <a>
* <img>

Why use CSS? Why can't I simply use a center tag or a font tag to style my page?

Ans. The <center> element was deprecated because it defines the presentation of its contents -- it doesn't describe its contents.

One method of centering is to set the margin-left and margin-right properties of the element to auto, and then set the parent element's text-align property to center. This guarantees that the element will be centered in all modern browsers.

What are the different ways (or locations) you can define styles on an element?

I can think of 3: External, Internal, In-line and Bootstrap.

What are the different values for the following properties? What are the effects of each of those values?

- display: block, inline, inline-block

- position: absolute, relative, fixed, static

- overflow: auto, hidden, scroll, visible

- float: left, right

What is the box model?

## Ans. The 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.

What are the two different box-sizing values? How do they work?

Ans. The CSS3 box-sizing property allows us to include the padding and border in an element's total width and height.

If you set box-sizing: border-box; on an element padding and border are included in the width and height:

What is z-index? How does it work? In what situation would I use it?

Ans.

The z-index property specifies the stack order of an element.

An element with greater stack order is always in front of an element with a lower stack order.

Note: z-index only works on positioned elements (position:absolute, position:relative, or position:fixed).

z-index: auto|*number*|initial|inherit;

What is the difference between visibility: hidden and display: none?

Ans. display: none will not be available in the page and does not occupy any space. visibility: hidden hides an element, but it will still take up the same space as before. The element will be hidden, but still affect the layout. visibility: hidden preserve the space, whereas display: none doesn't preserve the space.

What is specificity? How do you calculate it?

## Ans

## The concept:

Specificity is the means by which browsers decide which CSS property values are the most relevant to an element and, therefore, will be applied. Specificity is based on the matching rules which are composed of [CSS selectors](https://developer.mozilla.org/en/CSS/CSS_Reference#Selectors) of different sorts.

## How is it calculated?

Specificity is a weight that is applied to a given CSS declaration, determined by the number of each [selector type](https://developer.mozilla.org/en/docs/Web/CSS/Specificity#Selector_Types) in the matching selector. When specificity is equal to any of the multiple declarations, the last declaration found in the CSS is applied to the element. Specificity only applies when the same element is targeted by multiple declarations. As per CSS rules [directly targeted element](https://developer.mozilla.org/en/docs/Web/CSS/Specificity#directly-targeted-elements) will always take precedence over rules that an element inherits from an ancestor.

Note: [Proximity of elements](https://developer.mozilla.org/en/docs/Web/CSS/Specificity#tree-proximity-ignorance) in the document tree has no effect on the specificity.

### Selector Types

The following list of selector types is by increasing specificity:

1. Type selectors (e.g., h1) and pseudo-elements (e.g., :before).
2. Class selectors (e.g., .example), attributes selectors (e.g., [type="radio"]) and pseudo-classes (e.g., :hover).
3. ID selectors (e.g., #example).

Universal selector (\*), combinators (+, >, ~, ' ') and negation pseudo-class (:not()) have no effect on specificity. (The selectors declared inside :not() do, however.)

Inline styles added to an element (e.g., style="font-weight:bold") always overwrite any styles in external stylesheets and thus can be thought of as having the highest specificity.

What does !important do? Why is it considered a bad practice?

Ans. When an important rule is used on a style declaration, this declaration overrides any other declarations. Although technically !important has nothing to do with specificity, it interacts directly with it. Using !important is **bad practice** and should be avoided because it makes debugging more difficult by breaking the natural [cascading](https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade) in your stylesheets. When two conflicting declarations with the !important rule are applied to the same element, the declaration with greater specificity will be applied.

How would you go about building a 3 column layout?

Ans.

Flexbox consists of flex containers and flex items.

A flex container is declared by setting the display property of an element to either flex (rendered as a block) or inline-flex (rendered as inline).

Inside a flex container there is one or more flex items.

**Note:** Everything outside a flex container and inside a flex item is rendered as usual. Flexbox defines how flex items are laid out inside a flex container.

Flex items are positioned inside a flex container along a flex line. By default there is only one flex line per flex container.

<style>   
.flex-container {  
    display: -webkit-flex;  
    display: flex;  
    width: 400px;  
    height: 250px;  
    background-color: lightgrey;  
}  
  
.flex-item {  
    background-color: cornflowerblue;  
    width: 100px;  
    height: 100px;  
    margin: 10px;  
}  
</style>  
</head>  
<body>  
  
<div class="flex-container">  
  <div class="flex-item">flex item 1</div>  
  <div class="flex-item">flex item 2</div>  
  <div class="flex-item">flex item 3</div>   
</div>

If you have a div on a page that matches on all of the rules below, what would be the computed (finally applied) style on the element?

.myclass1 {

color: black;

border: 2px solid black;

overflow: scroll;

}

div.myclass2 {

color: red;

float: right;

}

div#myclass1 {

color: yellow;

border: 2px solid red;

}

Ans: div#myclass1 {

color: yellow;

border: 2px solid red;

}

Specificity rule

-------------------------------------------------------------------------------

SASS

-------------------------------------------------------------------------------

What are the benefits of using SASS over CSS?

Ans.

**Features**

* Fully CSS-compatible
* Language extensions such as variables, nesting, and mixins
* Many [useful functions](http://sass-lang.com/documentation/Sass/Script/Functions.html) for manipulating colors and other values
* Advanced features like [control directives](http://sass-lang.com/documentation/file.SASS_REFERENCE.html#control_directives__expressions) for libraries
* Well-formatted, customizable output

-------------------------------------------------------------------------------

JavaScript

-------------------------------------------------------------------------------

How would you create an array? Get its length? Access one of its elements?

Add and remove elements?

Ans.

var arr = [‘neha’ , ‘ipsha’ ];

arr.length();

arr[0];

arr.push(‘krunal’);

arr.pop(); //removes last element

arr.shift(); //removes first element

arr.unshift(‘mansi’);

How would you create a simple object? Get the number of keys? Access one of its values? Add and remove keys?

Ans.

var obj1 = {name:’neha’, age: 22 }; //literal method

var person = new Object(); //new keyword  
person.firstName = "John";  
person.lastName = "Doe";

//constructor method

function person(first, last, age, eye) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eye;  
}  
var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

//find length of properties in the object

var x = Object.keys(obj1).length;

//access obj property

obj1.name;

//add new property

obj1.roll\_no = 22;

//delete property

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
delete person.age;

How would you decide when to use an array or when to use an object?

What is the global object?

What kind of scoping do you have in JavaScript? And related to that, what is hoisting?

Ans.

Local

Global

//variable hoisting

x = 5;

Var x;

What are closures? How are they useful?

Ans.

**Closures are functions that refer to independent (free) variables (variables that are used locally, but defined in an enclosing scope). In other words, these functions 'remember' the environment in which they were created.**

function init() {

var name = "Mozilla"; // name is a local variable created by init

function displayName() { // displayName() is the inner function, a closure

alert(name); // use variable declared in the parent function

}

displayName();

}

init();

init() creates a local variable name and then a function called displayName(). displayName() is an inner function that is defined inside init() and is only available within the body of that function. displayName() has no local variables of its own, however it has access to the variables of outer functions and so can use the variable namedeclared in the parent function.

When defining objects -

- what is the constructor pattern, it's drawback?

- what is the prototype patter, it's drawback?

- give the above drawbacks, what approach would you go with?

- how would you implement inheritance between classes?

What are the different patterns? When would you use each one?

Ans.

* [Constructor Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#constructorpatternjavascript)
* [Module Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript)
* [Revealing Module Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#revealingmodulepatternjavascript)
* [Singleton Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#singletonpatternjavascript)
* [Observer Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#observerpatternjavascript)
* [Mediator Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#mediatorpatternjavascript)
* [Prototype Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#prototypepatternjavascript)
* [Command Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#commandpatternjavascript)
* [Facade Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#facadepatternjavascript)
* [Factory Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#factorypatternjavascript)
* [Mixin Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#mixinpatternjavascript)
* [Decorator Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#decoratorpatternjavascript)
* [Flyweight Pattern](https://addyosmani.com/resources/essentialjsdesignpatterns/book/#detailflyweight)

In simple terms, what is a DOM?

Ans.

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

How would you pick elements from a page? For each of these standard methods what is the input passed in and what would be the output? And what object can these methods be used on?

Ans.

Document.getElementById();

And so on…..

How would one update the styles on element?

Ans.

function display(){

var x=document.getElementById("demo").value;

document.getElementById("main").style.backgroundColor=x;

}

How can I create an element, add it to a page, and then remove it?

How do I attach event listeners on an element?

Ans.

var x = document.getElementById("myBtn");

x.addEventListener("mouseover", myFunction);

function myFunction() {

document.getElementById("demo").innerHTML += "Moused over!<br>";

}

What are the two different event models?

-------------------------------------------------------------------------------

jQuery

-------------------------------------------------------------------------------

Why use jQuery?

There are some standard native lookups in JavaScript, what are they corresponding versions in jQuery?

What is the advantage of using these standard lookups in jQuery as well?

What is the difference between .find() and .children()? When would you use one over the other?

How would one update the styles on element?

How can I create an element, add it to a page, and then remove it?

How can I figure out the dimensions of an element?

How do I attach event listeners on an element?

How would you go about implementing a div based pop-up that is triggered by clicking on a link? It needs to be centered across the screen as well.

How would go about implementing a feature where I need a fixed header and a fixed footer (i.e., these sections should be visible at all times and the header should appear at the top of the screen while the footer at the bottom of the screen). And the portion in the middle of the page (i.e., the content) is the only one that should be scrollable.