

CSS Basics

Q. What is CSS?

CSS stands for **Cascading Style Sheets**. CSS is used to define styles for web pages, including the design, layout and variations in display for different devices and screen sizes. CSS was intended to allow web professionals to separate the content and structure of a website's code from the visual design.

CSS can be used for document text styling — for example changing the color and size of headings and links. It can be used to create layout — for example turning a single column of text into a layout with a main content area and a sidebar for related information. It can even be used for effects such as animation.

Example:

```
h1 {  
  color: red;  
  font-size: 5em;  
}  
  
p {  
  color: black;  
}
```

Q. What is the use of css ruleset?

CSS is a rule or set of rules that describe the formatting (change of appearance) of individual elements on a web page. The rule consists of two parts: the selector and the next declaration block. The image below shows the structure (syntax) of the rule:

```
div {  
  color: blue;  
  text-align: justify;  
}
```

- The first is always the **selector**, it tells the browser which element or elements of the web page will be styled.

- Next is the **declaration block**, which begins with the opening curly brace { and ends with the closing }, between the curly braces are specified formatting commands (declarations), which are used by the browser to stylize the selected selector element.
- Each **declaration** consists of two parts: the property and its value. The declaration must always end with a semicolon (;). You can omit the ; only at the end of the last declaration before the closing curly brace.
- A **property** is a formatting command that defines a specific style effect for an element. Each property has its own predefined set of values. After the property name, a colon is specified, which separates the property name from the valid value.

Q. What are the possible ways to apply CSS styles to a web page?

There are three ways to apply CSS to HTML: Inline, internal, and external.

1. Inline CSS:

Inline CSS is specified directly in the opening tag of the element you want it to apply to. It is entered into the style attribute within HTML. This allows CSS properties on a "per tag" basis.

Example:

```
<p style="font-weight:bold;">Bold Font</p>
```

This CSS type is not really recommended, as each HTML tag needs to be styled individually. However, inline CSS in HTML can be useful in some situations. For example, in cases where you don't have access to CSS files or need to apply styles for a single element only.

2. Internal CSS:

Internal or Embedded, styles are used for the whole page. Inside the head element, the style tags surround all of the styles for the page.

Example:

```
<!DOCTYPE html>  
<html>
```

```

<head>
  <title>Internal CSS Example</title>
<style>
  p {
    color: red;
  }

  a {
    color: blue;
  }
</style>
...

```

This CSS style is an effective method of styling a single page. However, using this style for multiple pages is time-consuming as you need to put CSS rules to every page of your website.

3. External CSS:

In external CSS rules are stored in a separate file. To refer to that file from the HTML page, add the link element (and its closing element within XHTML) to the head element. This CSS type is a more efficient method, especially for styling a large website. By editing one `.css` file, you can change your entire site at once.

style.css:

```

p {
  color: red;
}

a {
  color: blue;
}
<!DOCTYPE html>
<html>
<head>
  <title>External CSS Example</title>
  <link rel="stylesheet" type="text/css" href="style.css">
...

```

The link element in the example has three attributes. The first, `rel`, tells the browser the type of the target of the link. The second, `type`, tells the browser what type of stylesheet it is. And the third, `href`, tells the browser under which URL to find the stylesheet.

Q. What does the cascading portion of CSS mean?

The **cascading** in CSS refers to the fact that styling rules "cascade" down from several sources. This means that CSS has an inherent hierarchy and styles of a higher precedence will overwrite rules of a lower precedence.

Even the simplest HTML document may have three or more style sheets associated with it including:

- The browser's style sheet
- The user's style sheet
- The author's style sheet

1. Browser style sheets:

Browsers apply style sheets to all web documents. Although these style sheets vary from browser to browser, they all have common characteristics such as black text, blue links, purple visited links etc. These are referred to as a "default" browser stylesheet.

As soon as you, the author, apply a style sheet to a document, it will override the browser's style sheet. This is because author style sheets will always override browser style sheets.

2. User style sheets:

Most modern browsers allow users to set their own style sheets within their browser. These style sheets will override the browsers default style sheets - for that user only.

3. Author style sheets:

As soon as you apply a basic style sheet or an inline style to a page, you have added what is referred to as an "author style sheet". Everything you do, from choosing fonts, colours and laying out pages in CSS is done using author style sheets.

Q. Explain new features in CSS3?

1. CSS3 Selectors:

- Matches any element **E** whose attribute **attr** starts with the value **val**. In other words, the val matches the beginning of the attribute value.

```
E[attr^=val]  
/* Example */
```

```
a[href^='http://sales.'){color: teal;}
```

- Matches any element **E** whose attribute **attr** ends in **val**. In other words, the val matches the end of the attribute value.

```
E[attr$=val]
/* Example */
a[href$='.jsp']{color: purple;}
```

- Matches any element **E** whose attribute **attr** matches **val** anywhere within the attribute. It is similar to `E[attr~=val]`, except the val can be part of a word.

```
E[attr*=val]
/* Example */
img[src*='artwork']{
    border-color: #C3B087 #FFF #FFF #C3B087;
}
```

2. Pseudo-classes:

CSS2 supports user interaction pseudo-classes, namely `:link`, `:visited`, `:hover`, `:active`, and `:focus`.

A few more pseudo-class selectors were added in CSS3. One is the `:root` selector, which allows designers to point to the root element of a document.

```
:root{overflow:auto;}
```

As a complement to the `:first-child` selector, the `:last-child` was added. With it one can select the last element named of a parent element.

```
div.article > p:last-child{font-style: italic;}
```

A new user interaction pseudo-class selector was added, the `:target` selector.

```
<style>
  span.notice:target { font-size: 2em; font-style: bold; }
</style>

<a href='#section2'>Section 2</a>
<p id='section2'>...</p>
```

The negation pseudo-class selector, `:not` can be coupled with almost any other selector that has been implemented.

```
img:not([border]){ border: 1; }
```

3. CSS3 Colors:

The color keyword list has been extended in the CSS3 color module to include 147 additional keyword colors (that are generally well supported), CSS3 also provides us with a number of other options: **HSL**, **HSLA**, **RGBA** and **Opacity**.

```
div.halfopaque {
  background-color: rgb(0, 0, 0);
  opacity: 0.5;
  color: #000000;
}
div.halfalpha {
  background-color: rgba(0, 0, 0, 0.5);
  color: #000000;
}
```

4. Rounded Corners: border-radius:

```
border-radius: 25px;
```

5 Drop Shadows:

```
box-shadow: 2px 5px 0 0 rgba(72,72,72,1);
```

6. Text Shadow:

```
text-shadow: topOffset leftOffset blurRadius color;
```

7. Linear Gradients:

```
#grad {
  background: linear-gradient(to right, red, yellow);
}
```

8. Radial Gradients:

```
#grad {
  background: radial-gradient(red, yellow, green);
} //Default
#grad {
  background: radial-gradient(circle, red, yellow, green);
} //Circle
```

9. Multiple Background Images:

In CSS3, there's no need to include an element for every background image; it provides us with the ability to add more than one background image to any element, even to pseudo-elements.

```
background-image:
url(firstImage.jpg),
url(secondImage.gif),
url(thirdImage.png);
```

Q. What are the css selectors?

A CSS selector is the part of a CSS rule set that actually selects the content you want to style.

i) Universal Selector: The universal selector works like a wild card character, selecting all elements on a page. Every HTML page is built on content placed within HTML tags. Each set of tags represents an element on the page.

```
* {
  color: green;
  font-size: 20px;
  line-height: 25px;
}
```

ii) Element Type Selector: This selector match one or more HTML elements of the same name.

```
ul {
  list-style: none;
  border: solid 1px #ccc;
}
<ul>
  <li>Fish</li>
  <li>Apples</li>
  <li>Cheese</li>
</ul>

<div class="example">
  <p>Example paragraph text.</p>
</div>

<ul>
  <li>Water</li>
  <li>Juice</li>
  <li>Maple Syrup</li>
</ul>
```

iii) ID Selector: This selector matches any HTML element that has an ID attribute with the same value as that of the selector.

```
#container {
```

```
width: 960px;
margin: 0 auto;
}
<div id="container"></div>
```

iv) Class Selector: The class selector also matches all elements on the page that have their class attribute set to the same value as the class.

```
.box {
padding: 20px;
margin: 10px;
width: 240px;
}
<div class="box"></div>
```

v) Descendant Combinator: The descendant selector or, more accurately, the descendant combinator lets you combine two or more selectors so you can be more specific in your selection method.

```
#container .box {
float: left;
padding-bottom: 15px;
}
```

This declaration block will apply to all elements that have a class of box that are inside an element with an ID of `container`. It's worth noting that the `.box` element doesn't have to be an immediate child: there could be another element wrapping `.box`, and the styles would still apply.

```
<div id="container">
  <div class="box"></div>

  <div class="box-2"></div>
</div>

<div class="box"></div>
```

vi) Child Combinator: A selector that uses the child combinator is similar to a selector that uses a descendant combinator, except it only targets immediate child elements.

```
#container > .box {
float: left;
padding-bottom: 15px;
}
```

The selector will match all elements that have a class of `box` and that are immediate children of the `#container` element. That means, unlike the descendant combinator, there can't be another element wrapping `.box`—it has to be a direct child element.


```
<div id="container">
  <div class="box"></div>

  <div>
    <div class="box"></div>
  </div>
</div>
```

vii) General Sibling Combinator: A selector that uses a general sibling combinator matches elements based on sibling relationships. The selected elements are beside each other in the HTML.

```
h2 ~ p {
  margin-bottom: 20px;
}
```

In this example, all paragraph elements (`<p>`) will be styled with the specified rules, but only if they are siblings of `<h2>` elements. There could be other elements in between the `<h2>` and `<p>`, and the styles would still apply.

```
<h2>Title</h2>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<div class="box">
  <p>Paragraph example.</p>
</div>
```

viii) Adjacent Sibling Combinator: A selector that uses the adjacent sibling combinator uses the plus symbol (+), and is almost the same as the general sibling selector. The difference is that the targeted element must be an immediate sibling, not just a general sibling.

```
p + p {
  text-indent: 1.5em;
  margin-bottom: 0;
}
```

In this example will apply the specified styles only to paragraph elements that immediately follow other paragraph elements. This means the first paragraph element on a page would not receive these styles. Also, if another element appeared between two paragraphs, the second paragraph of the two wouldn't have the styles applied.

```
<h2>Title</h2>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
```

```
<div class="box">
  <p>Paragraph example.</p>
  <p>Paragraph example.</p>
</div>
```

ix) Attribute Selector: The attribute selector targets elements based on the presence and/or value of HTML attributes, and is declared using square brackets

```
input[type="text"] {
  background-color: #444;
  width: 200px;
}
<input type="text">
```

The attribute selector can also be declared using just the attribute itself, with no value, like this:

```
input[type] {
  background-color: #444;
  width: 200px;
}
```

x) Pseudo-class: A pseudo-class uses a colon character to identify a pseudo-state that an element might be in—for example, the state of being hovered, or the state of being activated.

```
a:hover {
  color: red;
}
```

xi) Pseudo-element: A CSS pseudo-element is used to style specified parts of an element. For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p::first-line {
        color: #ff0000;
        font-variant: small-caps;
      }

      p::first-letter {
        color: #ff0000;
        font-size: xx-large;
      }
    </style>
  </head>
</html>
```

```

h1::before {
  content: url(smiley.gif);
}

h1::after {
  content: url(smiley.gif);
}

::selection {
  color: red;
  background: yellow;
}
</style>
</head>
<body>
  <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry.
  Lorem Ipsum has been the industry\'s standard dummy text ever since the 1500s,
  <h1>when an unknown printer took a galley of type and scrambled it to make a
  type specimen book.<h1></p>
</body>
</html>

```

Live Demo: [CSS Selectors](#)

Q. What is contextual selector?

Contextual selector addresses specific occurrence of an element. It is a string of individual selectors separated by white space (search pattern), where only the last element in the pattern is addressed providing it matches the specified context.

It also check the context of the class in the html tree, assigning the style to the element through a specific route, taking into account the order of depth in the tree.

Example:

```
table p { property: value; }
```

Q. What is the difference between Pseudo-classes and pseudo-elements?

A pseudo-class is a selector that assists in the selection of something that cannot be expressed by a simple selector, for example `:hover`. A pseudo-element however

allows us to create items that do not normally exist in the document tree, for example `::after`.

Pseudo-classes:

Pseudo-classes select regular elements but under certain conditions, like when their position relative to siblings or when they're under a particular state. Here is a list of pseudo-classes in CSS3:

a) Dynamic pseudo-classes:

- `:link`
- `:visited`
- `:hover`
- `:active`
- `:focus`

b) UI element states pseudo-classes:

- `:enabled`
- `:disabled`
- `:checked`

c) Structural pseudo-classes:

- `:first-child`
- `:nth-child(n)`
- `:nth-last-child(n)`
- `:nth-of-type(n)`
- `:nth-last-of-type(n)`
- `:last-child`
- `:first-of-type`
- `:last-of-type`
- `:only-child`
- `:only-of-type`
- `:root`
- `:empty`

d) Other pseudo-classes:

`:not(x)` `:target` `:lang(language)`

Pseudo-elements:

Pseudo-elements effectively create new elements that are not specified in the markup of the document and can be manipulated much like a regular element.

- ::before
- ::after
- ::first-letter
- ::first-line
- ::selection

Q. What is Combinator selector?

A combinator is the character in a selector that connects two selectors together. There are four types of combinators.

a) Descendant Combinator (space): The descendant selector matches all elements that are descendants of a specified element.

The following example selects all `<p>` elements inside `<div>` elements:

```
div p {  
  background-color: yellow;  
}
```

b) Child Combinator (>): The child selector selects all elements that are the children of a specified element.

The following example selects all `<p>` elements that are children of a `<div>` element:

```
div > p {  
  background-color: yellow;  
}
```

c) Adjacent Sibling Combinator (+): The adjacent sibling selector selects all elements that are the adjacent siblings of a specified element.

The following example selects all `<p>` elements that are placed immediately after `<div>` elements:

```
div + p {  
  background-color: yellow;  
}
```

d) General Sibling Combinator (~): The general sibling selector selects all elements that are siblings of a specified element.

The following example selects all `<p>` elements that are siblings of `<div>` elements:

```
div ~ p {  
    background-color: yellow;  
}
```

Q. What is the difference between class selectors and id selectors?

In the CSS, a class selector is a name preceded by a full stop (".") and an ID selector is a name preceded by a hash character ("#"). The difference between an ID and a class is that an ID can be used to identify one element, whereas a class can be used to identify more than one.

```
#top {  
    background-color: #ccc;  
    padding: 20px  
}  
  
.intro {  
    color: red;  
    font-weight: bold;  
}  
<div id="top">  
    <h1>Welcome to the CSS3 Tutorial</h1>  
    <p class="intro">Select element by class</p>  
    <p class="intro">Example for class selector paragraph</p>  
</div>
```

Q. What is the difference between the “nth-child()” and “nth-of-type()” selectors?

The `nth-child()` pseudo-class is used to match an element based on a number, which represents the element's position amongst its siblings. More specifically, the number represents the number of siblings that exist before the element in the document tree (minus 1).

Example:

```
.example :nth-child(4) { background: #ffdb3a; }
<div class="example">
  <p>This is a <em>paragraph</em>.</p>
  <p>This is a <em>paragraph</em>.</p>
  <p>This is a <em>paragraph</em>.</p>
  <div>This is a <em>divider</em>.</div> <!-- Element to select -->
  <div>This is a <em>divider</em>.</div>
</div>
```

The `nth-of-type()` pseudo-class, like `nth-child()`, is used to match an element based on a number. This number, however, represents the element's position within only those of its siblings that are of the same element type.

This number can also be expressed as a function, or using the keywords even or odd.

```
.example p:nth-of-type(odd) { background: #ffdb3a; }
<div class="example">
  <p>This is a <em>paragraph</em>.</p> <!-- Element to select -->
  <p>This is a <em>paragraph</em>.</p>
  <p>This is a <em>paragraph</em>.</p> <!-- Element to select -->
  <div>This is a <em>divider</em>.</div>
  <div>This is a <em>divider</em>.</div> <!-- Element to select -->
</div>
```

Q. Explain CSS grid layout with example?

CSS Grid Layout excels at dividing a page into major regions or defining the relationship in terms of size, position, and layer, between parts of a control built from HTML primitives. Like tables, grid layout enables us to align elements into columns and rows.

Basic Terminology:

- **Grid Item:** The grid container contains the grid items.
- **Grid Line:** The grid line is either a vertical or horizontal grid line that makes up the structure of the grid.
- **Grid Cell:** The smallest unit on a grid is referred to as a Grid cell. It is the space between the two adjacent rows and columns.
- **Rows:** The grid row is the horizontal track of the grid.
- **Columns:** The grid column is the vertical track of the grid.
- **Gutter:** A gutter is a space between the rows and columns in the grid.

CSS Grid Properties:

| Property | Description |
|-------------------|---|
| column-gap | Specifies the gap between the columns |
| gap | A shorthand property for the row-gap and the column-gap properties |
| grid | A shorthand property for the grid-template-rows, grid-template-columns, grid-template-areas, grid-auto-rows, grid-auto-columns, and the grid-auto-flow properties |
| grid-area | Either specifies a name for the grid item, or this property is a shorthand property for the grid-row-start, grid-column-start, grid-row-end, and grid-column-end properties |
| grid-auto-columns | Specifies a default column size |
| grid-auto-flow | Specifies how auto-placed items are inserted in the grid |
| grid-auto-rows | Specifies a default row size |
| grid-column | A shorthand property for the grid-column-start and the grid-column-end properties |
| grid-column-end | Specifies where to end the grid item |
| grid-column-gap | Specifies the size of the gap between columns |
| grid-column-start | Specifies where to start the grid item |

| Property | Description |
|-----------------------|--|
| grid-gap | A shorthand property for the grid-row-gap and grid-column-gap properties |
| grid-row | A shorthand property for the grid-row-start and the grid-row-end properties |
| grid-row-end | Specifies where to end the grid item |
| grid-row-gap | Specifies the size of the gap between rows |
| grid-row-start | Specifies where to start the grid item |
| grid-template | A shorthand property for the grid-template-rows, grid-template-columns and grid-areas properties |
| grid-template-areas | Specifies how to display columns and rows, using named grid items |
| grid-template-columns | Specifies the size of the columns, and how many columns in a grid layout |
| grid-template-rows | Specifies the size of the rows in a grid layout |
| row-gap | Specifies the gap between the grid rows |

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Grid Layout</title>
  <style>
    .item1 {
      grid-area: header;
    }
  </style>
</head>
</html>
```

```

.item2 {
  grid-area: menu;
}

.item3 {
  grid-area: main;
}

.item4 {
  grid-area: right;
}

.item5 {
  grid-area: footer;
}

.grid-container {
  display: grid;
  grid-template-areas:
    'header header header header header header'
    'menu main main main right right'
    'menu footer footer footer footer footer';
  grid-gap: 10px;
  background-color: rgba(155, 249, 249);
  padding: 10px;
}

.grid-container>div {
  background-color: rgba(0, 150, 149);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Grid Layout</h1>
<p>This grid layout contains six columns and three rows:</p>

<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">main</div>
  <div class="item4">Right</div>
  <div class="item5">Footer</div>
</div>
</body>
</html>

```

Live Demo: [Grid Layout](#)

Q. What is CSS flexbox?

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning. Flexbox makes it simple to align items vertically and horizontally using rows and columns. Items will "flex" to different sizes to fill the space.

Before the Flexbox Layout module, there were four layout modes:

- **Block**, for sections in a webpage
- **Inline**, for text
- **Table**, for two-dimensional table data
- **Positioned**, for explicit position of an element

Flex Container:

An area of a document laid out using flexbox is called a **flex container**. To create a flex container, we set the value of the area's container's `display` property to `flex` or `inline-flex`. As soon as we do this the direct children of that container become **flex items**.

Flexbox Terminology:

Q. Write all the properties of the flexbox?

The CSS Flexbox Container Properties:

| Property | Values |
|------------------------------|---|
| <code>display:</code> | <code>flexbox</code> , <code>inline-flex</code> ; |
| <code>flex-direction:</code> | <code>row</code> , <code>row-reverse</code> , <code>column</code> , <code>column-reverse</code> ; |
| <code>flex-wrap:</code> | <code>nowrap</code> , <code>wrap</code> , <code>wrap-reverse</code> ; |
| <code>flex-flow:</code> | <code><'flex-direction'></code> , <code><'flex-wrap'></code> |

| Property | Values |
|------------------|---|
| justify-content: | flex-start, flex-end, center, space-between, space-around; |
| align-items: | flex-start, flex-end, center, baseline, stretch; |
| align-content: | flex-start, flex-end, center, space-between, space-around, stretch; |

The CSS Flexbox Properties:

| Property | Values |
|--------------|--|
| order: | <integer>; |
| flex-grow: | <number>; /* default 0 */ |
| flex-shrink: | <number>; /* default 1 */ |
| flex-basis: | <length>, auto; /* default auto */ |
| flex: | none, [<'flex-grow'> <'flex-shrink'>?] |
| align-self: | auto, flex-start, flex-end, center, baseline, stretch; |

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>The flex-direction Property</title>
  <style>
    .flex-container {
      display: flex;
      flex-direction: column;
      background-color: DodgerBlue;
    }

    .flex-container>div {
      background-color: #f1f1f1;
      width: 100px;
      margin: 10px;
      text-align: center;
      line-height: 75px;
      font-size: 30px;
    }
  </style>
```

```

</head>
<body>
  <h1>The flex-direction Property</h1>
  <p>The "flex-direction: column;" stacks the flex items vertically (from top to
bottom):</p>

  <div class="flex-container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>
</body>
</html>

```

Live Demo: [flex-direction Property](#)

Q. When to use css grid and flexbox?

- CSS Grid Layout is a **two-dimensional** system, meaning it can handle both columns and rows, unlike flexbox which is largely a **one-dimensional** system (either in a column or a row).
- A core difference between CSS Grid and Flexbox is that — CSS Grid's approach is **layout-first** while Flexbox' approach is **content-first**. If you are well aware of your content before making layout, then blindly opt for Flexbox and if not, opt for CSS Grid.
- Flexbox layout is most appropriate to the components of an application (as most of them are fundamentally linear), and **small-scale** layouts, while the Grid layout is intended for **larger-scale** layouts which aren't linear in their design.
- If you only need to define a layout as a row or a column, then you probably need flexbox. If you want to define a grid and fit content into it in two dimensions — you need the grid.

Example:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Grid vs Flexbox Layout</title>
  </head>
  <style>
    /* Grid layout */
    .row {

```



```
        <div>Five</div>
    </div>
</body>
</html>
```

Live Demo: [CSS Grid and flexbox](#)

Q. What is CSS BEM?

The BEM (**Block Element Modifier**) methodology is a naming convention for CSS classes in order to keep CSS more maintainable by defining namespaces to solve scoping issues. A Block is a standalone component that is reusable across projects and acts as a "namespace" for sub components (Elements). Modifiers are used as flags when a Block or Element is in a certain state or is different in structure or style.

```
/* block component */
.block {
}

/* element */
.block__element {
}

/* modifier */
.block__element--modifier {
}
```

Example:

```
.button {
    display: inline-block;
    border-radius: 3px;
    padding: 7px 12px;
    border: 1px solid #D5D5D5;
    background-image: linear-gradient(#EEE, #DDD);
    font: 700 13px/18px Helvetica, arial;
}
.button--state-success {
    color: #FFF;
    background: #569E3D linear-gradient(#79D858, #569E3D) repeat-x;
    border-color: #4A993E;
}
.button--state-danger {
    color: #900;
}
<button class="button">
    Normal button
</button>
<button class="button button--state-success">
    Success button
```

```
</button>
<button class="button button--state-danger">
  Danger button
</button>
```

Benefits:

- **Modularity:** Block styles are never dependent on other elements on a page, so you will never experience problems from cascading.
- **Reusability:** Composing independent blocks in different ways, and reusing them intelligently, reduces the amount of CSS code that you will have to maintain.
- **Structure:** BEM methodology gives your CSS code a solid structure that remains simple and easy to understand.

Q. What are the benefits of using CSS sprites?

CSS sprites combine multiple images into one single larger image. It is a commonly-used technique for icons.

Advantages:

- Reduce the number of HTTP requests for multiple images (only one single request is required per spritesheet). But with HTTP2, loading multiple images is no longer much of an issue.
- Advance downloading of assets that won't be downloaded until needed, such as images that only appear upon :hover pseudo-states. Blinking wouldn't be seen.
- When you have multiple images/ icons, browser makes separate call to the server for each one of them. sprite is a technique to combine all/ some of them (usually similar one in terms of type of image. For example, you will put jpg in one sprite) in one image. To display the icon you set height, width and background position.

Alternatives:

- Data URIs - allow you to embed the image data directly into a stylesheet. This avoids additional HTTP requests for images, making it essentially the same thing as a sprite, without the fancy positioning.
- Icon Fonts
- SVGs

Q. What is tweening in css?

The pose-to-pose option is to create a few keyframes throughout the sequence, and then fill in the gaps later. Filling in these gaps is known as **tweening**. It is the process of generating intermediate frames between two images. It gives the impression that the first image has smoothly evolved into the second one. In CSS3, Transforms (matrix, translate, rotate, scale etc.) module can be used to achieve tweening.

Example:

```
p {
  animation-duration: 3s;
  animation-name: slidein;
}

@keyframes slidein {
  from {
    margin-left: 100%;
    width: 300%;
  }

  to {
    margin-left: 0%;
    width: 100%;
  }
}
```

Q. Explain the difference between **visibility: hidden;** and **display: none;**? What are the pros and cons of using **display:none**?

- **visibility: hidden** simply hides the element but it will occupy space and affect the layout of the document.
- **display: none** removes the element from the normal layout flow (causes DOM reflow). It will not affect the layout of the document nor occupy space.

Q. What is the purpose of the **z-index** and how a stacking context is formed?

The `z-index` helps specify the stack order of positioned elements that may overlap one another. The `z-index` default value is zero, and can take on either a positive or negative number. An element with a higher `z-index` is always stacked above than a lower index.

`z-index` can take the following values:

- **Auto**: Sets the stack order equal to its parents.
- **Number**: Orders the stack order.
- **Initial**: Sets this property to its default value (0).
- **Inherit**: Inherits this property from its parent element.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS z-index Property</title>
    <style>
      img {
        position: absolute;
        left: 0px;
        top: 0px;
        padding: 5px 2px;
        margin: 5px 1px 2px;
        z-index: -1;
      }

      p {
        color: red;
        font-size: 20px;
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>The z-index Property</h1>
    
    <p>Because the image has a z-index of -1, it will be placed behind the
    heading.</p>
  </body>
</html>
```

Live Demo: [Z-Index](#)

Q. Explain CSS position Property?

- **absolute**, place an element exactly where you want to place it. absolute position is actually set relative to the element's parent. if no parent available then relatively place to the page itself (it will default all the way back up to the element).
- **relative**, means "relative to itself". Setting position: relative; on an element and no other positioning attributes, it will no effect on it's positioning. It allows the use of `z-index` on the element and it limits the scope of absolutely positioned child elements. Any child element will be absolutely positioned within that block.
- **fixed**, element is positioned relative to viewport or the browser window itself. viewport doesn't changed if you scroll and hence fixed element will stay right in the same position.
- **static** default for every single page element. The only reason you would ever set an element to position: static is to forcefully-remove some positioning that got applied to an element outside of your control.
- **sticky** - Sticky positioning is a hybrid of relative and fixed positioning. The element is treated as `relative` positioned until it crosses a specified threshold, at which point it is treated as `fixed` positioned.

Live Demo: [CSS Position Property](#)

Q. What are the differences between relative and absolute in CSS?

Relative Position

An element with `position: relative;` is positioned relative to its normal position. Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

Example:

```

<!DOCTYPE html>
<html>
<head>
  <title>Relative Position</title>
  <style>
    div.relative {
      position: relative;
      top: 20px;
      left: 20px;
      border: 3px solid #2321ad;
    }
  </style>
</head>
<body>
  <h2>position: relative;</h2>
  <p>An element with position: relative; is positioned relative to its normal
position:</p>

  <div class="relative">
    This div element has position: relative;
  </div>
</body>
</html>

```

Live Demo: [Relative Position Property](#)

Absolute Position:

An element with `position: absolute;` will cause it to adjust its position with respect to its parent. If no parent is present, then it uses the document body as parent.

Example:

```

<!DOCTYPE html>
<html>
<head>
  <title>Absolute Position</title>
  <style>
    div.relative {
      position: relative;
      width: 400px;
      height: 200px;
      border: 3px solid #2321ad;
    }

    div.absolute {
      position: absolute;
      top: 80px;
      right: 0;
      width: 200px;
      height: 100px;
      border: 3px solid #ee0b0b;
    }
  </style>
</head>
<body>
  <div class="relative">
    <div class="absolute">
      This div element has position: absolute;
    </div>
  </div>
</body>
</html>

```

```
</style>
</head>
<body>
  <h2>position: absolute;</h2>

  <p>An element with position: absolute; is positioned relative to the nearest
  positioned ancestor
  (instead of positioned relative to the viewport, like fixed):</p>

  <div class="relative">This div element has position: relative;
    <div class="absolute">This div element has position: absolute;</div>
  </div>
</body>
</html>
```

Live Demo: [Absolute Position Property](#)

Q. The difference between block, inline and inline-block element?

a) Block Elements

The block elements always start on a new line. They will also take space of an entire row or width. List of block elements are `<p>`, `<h1>`, `<div>`, `<header>`.

Example:

```
<p>
  Lorem ipsum dolor sit amet consectetur adipisicing elit. Unde autem,
  consequatur deleniti nobis beatae quo dolore nemo corporis. Ad delectus
  dignissimos pariatur illum eveniet dolor rem eius laborum sed iure!
</p>

<p>
  Lorem ipsum dolor sit amet consectetur adipisicing elit. Unde autem,
  consequatur deleniti nobis beatae quo dolore nemo corporis. Ad delectus
  dignissimos pariatur illum eveniet dolor rem eius laborum sed iure!
</p>
```

b) Inline Elements

Inline elements don't start on a new line, they appear on the same line as the content and tags beside them. Some examples of inline elements are `<a>`, ``, ``, and `` tags.

When it comes to margins and padding, browsers treat inline elements differently. You can add space to the left and right on an inline element, but you cannot add height to the top or bottom padding or margin of an inline element.

Example:

```
<a href="#">Link</a>

<span>Span</span>
<strong>Strong Player</strong>
```

c) Inline-Block Elements

Inline-block elements are similar to inline elements, except they can have padding and margins added on all four sides. One common use for using inline-block is for creating navigation links horizontally. Some examples of inline-block elements are `<input>`, `<button>`, `<select>`, `<textarea>` etc.

```
input {
  width: 300px;
  height: 50px;
}

button {
  width: 100px;
  height: 50px;
  margin-top: 20px;
}
<input type="text" /> <button>Submit</button>
```

Live Demo: [Display Property](#)

Q. What are counters in CSS3?

CSS counters let you adjust the appearance of content based on its location in a document. To use a CSS counter, it must first be initialized to a value with the `counter-reset` property (0 by default). The same property can also be used to change its value to any specific number. Once initialized, a counter's value can be increased or decreased with counter-increment. The counter's name must not be "none", "inherit", or "initial"; otherwise the declaration is ignored.

```
body {
  counter-reset: section; /* Set a counter named 'section', and its initial
value is 0. */
}

h3::before {
  counter-increment: section; /* Increment the value of section counter by 1 */
  content: "Section " counter(section) ": "; /* Display the word 'Section ', the
value of
section counter, and a colon
before the content
of each h3 */
}
<h3>Introduction</h3>
<h3>Body</h3>
```

<h3>Conclusion</h3>

CSS Counter Properties:

| Property | Description |
|-------------------|---|
| content | Used with the ::before and ::after pseudo-elements, to insert generated content |
| counter-increment | Increments one or more counter values |
| counter-reset | Creates or resets one or more counters |

Example:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      counter-reset: section; /* Set a counter named 'section', and its initial
value is 0. */
    }

    h3::before {
      counter-increment: section; /* Increment the value of section counter by 1
*/
      content: "Section " counter(section) ": "; /* Display the word 'Section ',
the value of
                                                    section counter, and a colon
before the content
                                                    of each h3 */
    }
  </style>
</head>
<body>
  <h3>Introduction</h3>
  <h3>Body</h3>
  <h3>Conclusion</h3>
</body>
</html>
```

Live Demo: [CSS Counters](#)

Q. How do you specify units in the CSS?

There are various units in CSS to express the measurement and length. A CSS unit is used to determine the property size, which we set for an element or its content. The units in CSS are required to define the measurement such as margin: 20px; in which the px (or pixel) is the CSS unit. They are used to set margin, padding, lengths, and so on.

The length unit in CSS is of two types:

- Absolute length units.
- Relative length units.

Absolute length units:

The following are all absolute length units — they are not relative to anything else, and are generally considered to always be the same size.

| Unit | Name | Equivalent to |
|------|---------------------|----------------------|
| cm | Centimeters | 1cm = 38px = 25/64in |
| mm | Millimeters | 1mm = 1/10th of 1cm |
| Q | Quarter-millimeters | 1Q = 1/40th of 1cm |
| in | Inches | 1in = 2.54cm = 96px |
| pc | Picas | 1pc = 1/6th of 1in |
| pt | Points | 1pt = 1/72th of 1in |
| px | Pixels | 1px = 1/96th of 1in |

Relative length units:

Relative length units specify a length relative to another length property. Relative length units scale better between different rendering medium.

| Unit | Relative to |
|------|---|
| em | Font size of the parent, in the case of typographical properties like font-size, and font size of the element itself, in the case of other properties like width. |
| ex | x-height of the element's font. |
| ch | The advance measure (width) of the glyph "0" of the element's font. |
| rem | Font size of the root element. |
| lh | Line height of the element. |
| vw | 1% of the viewport's width. |
| vh | 1% of the viewport's height. |
| vmin | 1% of the viewport's smaller dimension. |
| vmax | 1% of the viewport's larger dimension. |

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS Units</title>
    <style>
      .wrapper {
        font-size: 1em;
      }

      .px {
        width: 200px;
      }

      .vw {
        width: 10vw;
      }

      .em {
        width: 10em;
      }
    </style>
```

```
</head>
<body>
  <div class="wrapper">
    <div class="box px">I am 200px wide</div>
    <div class="box vw">I am 10vw wide</div>
    <div class="box em">I am 10em wide</div>
  </div>
</body>
</html>
```

Live Demo: [CSS Units](#)

Q. Which one would you prefer among px, em % or pt and why?

- **px** gives fine grained control and maintains alignment because 1 px or multiple of 1 px is guaranteed to look sharp. px is not cascade, this means if parent font-size is 20px and child 16px. child would be 16px.
- **em** maintains relative size. you can have responsive fonts. em is the width of the letter 'm' in the selected typeface. However, this concept is tricky. 1em is equal to the current font-size of the element or the browser default. if u sent font-size to 16px then 1em = 16px. The common practice is to set default body font-size to 62.5% (equal to 10px). em is cascade
- **%** sets font-size relative to the font size of the body. Hence, you have to set font-size of the body to a reasonable size. this is easy to use and does cascade. for example, if parent font-size is 20px and child font-size is 50%. child would be 10px.
- **pt** (points) are traditionally used in print. 1pt = 1/72 inch and it is fixed-size unit.

Q. What is pseudo element and pseudo class?

1. Pseudo Element: A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

CSS Pseudo Elements:

| Sl.No | Selector | Example | description |
|-------|----------------|-----------------|--|
| 01. | ::after | p::after | Insert something after the content of each element |
| 02. | ::before | p::before | Insert something before the content of each element |
| 03. | ::first-letter | p::first-letter | Selects the first letter of each element |
| 04. | ::first-line | p::first-line | Selects the first line of each element |
| 05. | ::selection | p::selection | Selects the portion of an element that is selected by a user |

2. Pseudo-classes: A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

CSS Pseudo Classes:

| Sl.No | Selector | Example | description |
|-------|-----------|----------------|---|
| 01. | :active | a:active | Selects the active link |
| 02. | :checked | input:checked | Selects every checked <code><input></code> element |
| 03. | :disabled | input:disabled | Selects every disabled <code><input></code> element |
| 04. | :empty | p:empty | Selects every <code><p></code> element that has no children |

| Sl.No | Selector | Example | description |
|-------|----------------------|-----------------------|---|
| 05. | :enabled | input:enabled | Selects every enabled <code><input></code> element |
| 06. | :first-child | p:first-child | Selects every <code><p></code> elements that is the first child of its parent |
| 07. | :first-of-type | p:first-of-type | Selects every <code><p></code> element that is the first <code><p></code> element of its parent |
| 08. | :focus | input:focus | Selects the <code><input></code> element that has focus |
| 09. | :hover | a:hover | Selects links on mouse over |
| 10. | :in-range | input:in-range | Selects <code><input></code> elements with a value within a specified range |
| 11. | :invalid | input:invalid | Selects all <code><input></code> elements with an invalid value |
| 12. | :lang(language) | p:lang(it) | Selects every <code><p></code> element with a lang attribute value starting with "it" |
| 13. | :last-child | p:last-child | Selects every <code><p></code> elements that is the last child of its parent |
| 14. | :last-of-type | p:last-of-type | Selects every <code><p></code> element that is the last <code><p></code> element of its parent |
| 15. | :link | a:link | Selects all unvisited links |
| 16. | :not(selector) | :not(p) | Selects every element that is not a <code><p></code> element |
| 17. | :nth-child(n) | p:nth-child(2) | Selects every <code><p></code> element that is the second child of its parent |
| 18. | :nth-last-child(n) | p:nth-last-child(2) | Selects every <code><p></code> element that is the second child of its parent, |
| 19. | :nth-last-of-type(n) | p:nth-last-of-type(2) | Selects every <code><p></code> element that is the second <code><p></code> element of its |

| Sl.No | Selector | Example | description |
|-------|-----------------|--------------------|--|
| | | | parent, counting from the last child |
| 20. | :nth-of-type(n) | p:nth-of-type(2) | Selects every <code><p></code> element that is the second <code><p></code> element of its parent |
| 21. | :only-of-type | p:only-of-type | Selects every <code><p></code> element that is the only <code><p></code> element of its parent |
| 22. | :only-child | p:only-child | Selects every <code><p></code> element that is the only child of its parent |
| 23. | :optional | input:optional | Selects <code><input></code> elements with no "required" attribute |
| 24. | :out-of-range | input:out-of-range | Selects <code><input></code> elements with a value outside a specified range |
| 25. | :read-only | input:read-only | Selects <code><input></code> elements with a "readonly" attribute specified |
| 26. | :read-write | input:read-write | Selects <code><input></code> elements with no "readonly" attribute |
| 27. | :required | input:required | Selects <code><input></code> elements with a "required" attribute specified |
| 28. | :root root | | Selects the document's root element |
| 29. | :target | #news:target | Selects the current active #news element (clicked on a URL containing that anchor name) |
| 30. | :valid | input:valid | Selects all <code><input></code> elements with a valid value |
| 31. | :visited | a:visited | Selects all visited links |

Q. Explain the CSS “box model” and the layout components that it consists of?

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)

The size of the box itself is calculated like this:

| Property | Total |
|----------|--|
| Width | width + padding-left + padding-right + border-left + border-right |
| Height | height + padding-top + padding-bottom + border-top + border-bottom |

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Box Model</title>
  <style>
    div {
      background-color: lightgrey;
      width: 300px;
      border: 10px solid rgb(3, 141, 233);
      padding: 50px;
      margin: 20px;
    }
  </style>
</head>
<body>
  <h2>CSS Box Model</h2>

  <p>The CSS box model is essentially a box that wraps around every HTML element.
  It consists of: borders, padding, margins, and the actual content.</p>
```

```
<div>This text is the content of the box. We have added a 50px padding, 20px  
margin and a 10px blue  
border.</div>  
</body>  
</html>
```

Live Demo: [CSS Box Model](#)

Q. How you would tell the browser in CSS to render your layout in different box models?

The **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

Syntax:

```
box-sizing: content-box|border-box|initial|inherit;
```

Property Values:

| Value | Description |
|-------------|--|
| content-box | Default. The width and height properties (and min/max properties) includes only the content. Border and padding are not included |
| border-box | The width and height properties (and min/max properties) includes content, padding and border |
| initial | Sets this property to its default value. Read about initial |
| inherit | Inherits this property from its parent element. Read about inherit |

Example:

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>The box-sizing Property</title>  
  <style>  
    .content-box {
```

```

    box-sizing: content-box;
    width: 300px;
    height: 100px;
    padding: 30px;
    border: 10px solid rgb(0, 89, 255);
}

.border-box {
    box-sizing: border-box;
    width: 300px;
    height: 100px;
    padding: 30px;
    border: 10px solid rgb(255, 102, 0);
}
</style>
</head>
<body>
    <h2>The box-sizing Property</h2>
    <p>Defines how the width and height of an element are calculated: should they
include padding
    and borders, or not.</p>

    <h3>1. box-sizing: content-box (default):</h3>
    <p>Width and height only apply to the content of the element:</p>
    <div class="content-box">This div has a width of 300px. But the full width is
300px + 20px
    (left and right border) + 60px (left and right padding) = 380px!</div>

    <h3>2. box-sizing: border-box:</h3>
    <p>Width and height apply to all parts of the element: content, padding and
borders:</p>
    <div class="border-box">Here, the full width is 300px</div>
</body>
</html>

```

Q. What is the difference between border-box and content-box?

1. content-box:

Default box-sizing property. The width and height properties (and min/max properties) includes only the content. Border and padding are not included

2. border-box:

The width and height properties (and min/max properties) includes content, padding and border.

Q. Explain the meaning of each of these CSS units for expressing length?

- `cm` centimeters
- `em` elements (i.e., relative to the font-size of the element; e.g., 2 em means 2 times the current font size)
- `in` inches
- `mm` millimeters
- `pc` picas (1 pc = 12 pt = 1/6th of an inch)
- `pt` points (1 pt = 1/72nd of an inch)
- `px` pixels (1 px = 1/96th of an inch)

Q. In CSS3, how would you select?

- Every `<a>` element whose href attribute value begins with "https".

```
a[href^="https"]
```

- Every `<a>` element whose href attribute value ends with ".pdf".

```
a[href$=".pdf"]
```

- Every `<a>` element whose href attribute value contains the substring "css".

```
a[href*="css"]
```

Q. What is the purpose of the box-sizing property?

The box-sizing CSS property sets how the total width and height of an element is calculated.

- **content-box**: the default width and height values apply to the element's content only. The padding and border are added to the outside of the box.

- **padding-box**: Width and height values apply to the element's content and its padding. The border is added to the outside of the box. Currently, only Firefox supports the padding-box value.
- **border-box**: Width and height values apply to the content, padding, and border.
- **inherit**: inherits the box sizing of the parent element.

Example:

```
box-sizing: content-box;  
width: 100%;  
border: solid rgb(90,107,204) 10px;  
padding: 5px;
```

Q. What is the difference between RGBa, HEX and HSLa?

- **RGB** (Red/Green/Blue) is a color model.

```
p {  
  color: rgba(37, 84, 127, 1);  
}
```

- **HEX (Hexadecimal color values)**

```
p {  
  color: #25547f;  
}
```

- **HSLa (Hue Saturation Lightness alpha)**

```
p {  
  color: hsla(209, 55%, 32%, 1);  
}
```

Q. What is CSS preprocessor?

Pre-processors extend CSS with variables, operators, interpolations, functions, mixins and many more other usable assets. After development, these specific files are compiled into regular CSS that any browser can understand. Pre-processor help writing reusable, easily maintainable and extensible codes in CSS.

CSS preprocessors:

- SASS (SCSS)
- LESS
- Stylus
- PostCSS

Advantages:

- CSS is made more maintainable.
- Easy to write nested selectors.
- Variables for consistent theming. Can share theme files across different projects.
- Mixins to generate repeated CSS.
- Splitting your code into multiple files. CSS files can be split up too but doing so will require an HTTP request to download each CSS file.

Q. What is the difference between "resetting" and "normalizing" CSS?

1. Resetting: CSS resets aim to remove all built-in browser styling. For example margins, paddings, font-sizes of all elements are reset to be the same. You will have to redeclare styling for common typographic elements.

Example:

```
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p,
blockquote, pre, a, abbr,
acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small,
strike, strong,
sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label,
legend, table,
caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas, details, embed,
figure, figcaption,
footer, header, hgroup, menu, nav, output, ruby, section, summary, time, mark,
audio, video {
    margin: 0;
```

```
padding: 0;
border: 0;
font-size: 100%;
font: inherit;
vertical-align: baseline;
}
```

2. Normalizing: Normalize CSS aims to make built-in browser styling consistent across browsers. It also corrects bugs for common browser dependencies.

Example:

```
/*
  Correct the font size and margin on `h1` elements within `section`
  and `article` contexts in Chrome, Firefox, and Safari.
*/
h1 { font-size: 2em; margin: 0.67em 0;}
```

Q. Explain CSS Block Formatting Context?

Floats, absolutely positioned elements, block containers (such as inline-blocks, table-cells, and table-captions) that are not block boxes, and block boxes with 'overflow' other than 'visible' (except when that value has been propagated to the viewport) establish new block formatting contexts for their contents.

In a block formatting context, each box's left outer edge touches the left edge of the containing block (for right-to-left formatting, right edges touch)

A BFC is an HTML box that satisfies at least one of the following conditions:

- The value of `float` is not `none`.
- The value of `position` is neither `static` nor `relative`.
- The value of `display` is `table-cell`, `table-caption`, `inline-block`, `flex`, or `inline-flex`.
- The value of `overflow` is not `visible`.

In a BFC, each box's left outer edge touches the left edge of the containing block (for right-to-left formatting, right edges touch). Vertical margins between adjacent block-level boxes in a BFC collapse.

Example:

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

```

<head>
  <title>CSS Block Formatting Context</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
<style type="text/css">
  * {
    box-sizing: border-box;
  }

  body {
    margin: 40px;
    background-color: #fff;
    color: #444;
    font: 1.4em Arial, sans-serif;
  }

  .outer {
    background-color: #ccc;
    margin: 0 0 40px 0;
  }

  p {
    padding: 0;
    margin: 20px 0 20px 0;
    background-color: rgb(233, 78, 119);
    color: #fff;
  }

  .overflow {
    overflow: auto;
  }
</style>
<body>
  <h2>no BFC</h2>
  <div class="outer">
    <p>I am paragraph one and I have a margin top and bottom of 20px;</p>
    <p>I am paragraph two and I have a margin top and bottom of 20px;</p>
  </div>

  <h2>With a BFC</h2>

  <div class="outer overflow">
    <p>I am paragraph one and I have a margin top and bottom of 20px;</p>
    <p>I am paragraph two and I have a margin top and bottom of 20px;</p>
  </div>
</body>
</html>

```

Live Demo: [CSS Block Formatting](#)

Q. What is the float property and what float do?

The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it.

Syntax:

```
/* Keyword values */
float: left;
float: right;
float: none;
float: inline-start;
float: inline-end;

/* Global values */
float: inherit;
float: initial;
float: unset;
```

Property Values:

| Sl.No | Value | Description |
|-------|---------|---|
| 01. | none | The element does not float, (will be displayed just where it occurs in the text). |
| 02. | left | The element floats to the left of its container |
| 03. | right | The element floats the right of its container |
| 04. | initial | Sets this property to its default value. |
| 05. | inherit | Inherits this property from its parent element. |

Example::

```
section {
  border: 1px solid blue;
  width: 100%;
  float: left;
}

div {
  margin: 5px;
  width: 50px;
  height: 150px;
}

.left {
```

```

float: left;
background: pink;
}

.right {
float: right;
background: cyan;
}
<section>
  <div class="left">1</div>
  <div class="left">2</div>
  <div class="right">3</div>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Morbi tristique sapien ac erat tincidunt, sit amet dignissim
    lectus vulputate. Donec id iaculis velit. Aliquam vel
    malesuada erat. Praesent non magna ac massa aliquet tincidunt
    vel in massa. Phasellus feugiat est vel leo finibus congue.</p>
</section>

```

Q. Describe clear Property in css?

The clear property specifies what elements can float beside the cleared element and on which side.

| Sl.No | Properties | Description |
|-------|----------------|---|
| 01. | clear: none | Allows floating elements on both sides. This is default |
| 02. | clear: left | No floating elements allowed on the left side |
| 03. | clear: right | No floating elements allowed on the right side |
| 04. | clear: both | No floating elements allowed on either the left or the right side |
| 05. | clear: inherit | The element inherits the clear value of its parent |

Example:

```

div {
  clear: left;
}

```


Q. Explain the purpose of clearing floats in CSS?

The **clear** property is directly related to the float property. It specifies if an element should be next to the floated elements or if it should move below them. This property applies to both floated and non-floated elements.

CSS Syntax:

```
clear: none|left|right|both|inherit|inline-start|inline-end;
```

Property Values:

| Value | Description |
|-------|---|
| none | The element is not moved down to clear past floats. |
| left | The element is moved down to clear past left floats. |
| right | The element is moved down to clear past right floats. |
| Both | The element is moved down to clear past both left and right floats. |

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS clear Property</title>
  <style>
    .div1 {
      float: left;
      width: 100px;
      height: 50px;
      margin: 10px;
      border: 3px solid #73AD21;
    }

    .div2 {
      border: 1px solid red;
      height: 100px;
    }

    .div3 {
      float: left;
```

```

        width: 100px;
        height: 50px;
        margin: 10px;
        border: 3px solid #73AD21;
    }

    .div4 {
        border: 1px solid red;
        height: 100px;
        clear: left;
    }
</style>
</head>
<body>
    <h2>Without clear</h2>
    <div class="div1">div1</div>
    <div class="div2">div2 - Notice that the div2 element is after div1, in the
HTML code.
    However, since div1 is floated to the left, this happens: the text in div2
is floated
    around div1, and div2 surrounds the whole thing.
    </div><br/>

    <h2>Using clear</h2>
    <div class="div3">div3</div>
    <div class="div4">div4 - Using clear moves div4 down below the floated div3.
The value
    "left" clears elements floated to the left. You can also clear "right" and
"both".</div>
</body>
</html>

```

Live Demo: [CSS clear Property](#)

Q. What is a clearfix in CSS?

A **clearfix** is a way for an element to clear its child elements automatically without any additional markup. The clearfix property is generally used in float layouts where elements are floated to be stacked horizontally.

The clearfix property allows a container to wrap its floated children. Without a clearfix, a container will not wrap around its floated children and will collapse, just as if its floated children had been positioned absolutely.

Syntax:

```
.clearfix {
```

```
properties
}
```

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Clearfix property</title>
  <style>
    div {
      border: 3px solid #4CAF50;
      padding: 5px;
    }

    .clearfix {
      overflow: auto;
    }

    .img {
      float: right;
    }
  </style>
</head>
<body>
  <div class="clearfix">
    
    Running Horse
  </div>
</body>
</html>
```

Live Demo: [CSS Clearfix](#)

Q. Does `overflow: hidden` create a new block formatting context?

Yes. overflow property deals with the content if content size exceeds the allocated size for the content. You can make extra content visible, hidden, scroll or auto (viewport default behavior).

Q. How would you approach fixing browser-specific styling issues?

- Use a separate style sheet that only loads when that specific browser is being used. This technique requires server-side rendering though.
- Use `autoprefixer` to automatically add vendor prefixes to your code.
- Use Reset CSS or Normalize.css.

```
/*Example: 01*/
.box-shadow {
  background-color: red;
  background-image: url(gradient-slice.png);
  background-image: -webkit-linear-gradient(top right, #A60000, #FFFFFF); /*Chrome
and Safari*/
  background-image: -moz-linear-gradient(top right, #A60000, #FFFFFF);
/*Firefox*/
  background-image: -ms-linear-gradient(top right, #A60000, #FFFFFF);
/*Internet Explorer*/
  background-image: -o-linear-gradient(top right, #A60000, #FFFFFF);
/*Opera*/
  background-image: linear-gradient(top right, #A60000, #FFFFFF);
}

/*Example: 02*/
.box {
  -moz-border-radius: 15px; /* Firefox */
  -webkit-border-radius: 15px; /* Safari and Chrome */
  border-radius: 15px;
}
```

Q. What are your favorite image replacement techniques and which do you use when?

Technique: 01:

```
h1#technique-one {
  width: 250px;
  height: 25px;
  background-image: url(logo.gif);
}
h1#technique-one span {
  display: none;
}
<h1 id="technique-one">
  <span>CSS-Tricks</span>
</h1>
```

Technique: 02:

```
h1.technique-two {
```

```

width: 2350px;
height: 75px;
background: url("images/header-image.jpg") top right;
margin: 0 0 0 -2000px;
}
<h1 class="technique-two">
  CSS-Tricks
</h1>

```

Technique: 03:

```

h1.technique-three {
  width: 350px;
  height: 75px;
  background: url("images/header-image.jpg");
  text-indent: -9999px;
}
<h1 class="technique-three">
  CSS-Tricks
</h1>

```

Technique: 04:

```

h1.technique-four {
  width: 350px;
  height: 75px;
  background: url("images/header-image.jpg");
  text-indent: -9999px;
}
<h1 class="technique-four">
  <a href="#">
    
  </a>
</h1>

```

Technique: 05:

```

h1.technique-five {
  width: 350px;
  height: 75px;
  background: url("images/header-image.jpg");
}
h1.technique-five span {
  display: none;
}
<h1 class="technique-five">
  
  <span>CSS-Tricks</span>
</h1>

```

Technique: 06:

```

h1.technique-six {

```

```
width: 350px;
padding: 75px 0 0 0;
height: 0;
background: url("images/header-image.jpg") no-repeat;
overflow: hidden;
}
<h1 class="technique-six">
  CSS-Tricks
</h1>
```

Technique: 07:

```
h1.technique-seven {
  width: 350px;
  height: 75px;
  background: url("images/header-image.jpg") no-repeat;
}
h1.technique-seven span {
  display: block;
  width: 0;
  height: 0;
  overflow: hidden;
}
<h1 class="technique-seven">
  <span>CSS-Tricks</span>
</h1>
```

Technique: 08:

```
h1.technique-eight {
  width: 350px;
  height: 75px;
  position: relative;
}
h1.technique-eight span {
  background: url("images/header-image.jpg");
  position: absolute;
  width: 100%;
  height: 100%;
}
<h1 class="technique-eight">
  <span></span>CSS-Tricks
</h1>
```

Technique: 09:

```
h1.technique-nine {
  width: 350px;
  height: 75px;
  background: url("images/header-image.jpg") no-repeat;
  font-size: 1px;
  color: white;
}
<h1 class="technique-nine">
```

Q. What are media queries? How could you apply css rules specific to a media?

Media queries are useful when you want to modify your site or app depending on a device's general type (such as print vs. screen) or specific characteristics and parameters (such as screen resolution or browser viewport width). It uses the @media rule to include a block of CSS properties only if a certain condition is true.

Media Types:

| Sl.No | Value | Description |
|-------|--------|---|
| 01. | all | Default. Used for all media type devices |
| 02. | print | Used for printers |
| 03. | screen | Used for computer screens, tablets, smart-phones etc. |
| 04. | speech | Used for screenreaders that "reads" the page out loud |

Media Features:

| Sl.No | Value | Description |
|-------|--------------|--|
| 01. | any-hover | Does any available input mechanism allow the user to hover over elements? |
| 02. | any-pointer | Is any available input mechanism a pointing device, and if so, how accurate is it? |
| 03. | aspect-ratio | The ratio between the width and the height of the viewport |

| Sl.No | Value | Description |
|-------|------------------|--|
| 04. | color | The number of bits per color component for the output device |
| 05. | color-gamut | The approximate range of colors that are supported by the user agent and output device |
| 06. | color-index | The number of colors the device can display |
| 07. | grid | Whether the device is a grid or bitmap |
| 08. | height | The viewport height |
| 09. | hover | Does the primary input mechanism allow the user to hover over elements? |
| 10. | inverted-colors | Is the browser or underlying OS inverting colors? |
| 11. | light-level | Current ambient light level |
| 12. | max-aspect-ratio | The maximum ratio between the width and the height of the display area |
| 13. | max-color | The maximum number of bits per color component for the output device |
| 14. | max-color-index | The maximum number of colors the device can display |
| 15. | max-height | The maximum height of the display area, such as a browser window |
| 16. | max-monochrome | The maximum number of bits per "color" on a monochrome (greyscale) device |

| Sl.No | Value | Description |
|-------|------------------|--|
| 17. | max-resolution | The maximum resolution of the device, using dpi or dpcm |
| 18. | max-width | The maximum width of the display area, such as a browser window |
| 19. | min-aspect-ratio | The minimum ratio between the width and the height of the display area |
| 20. | min-color | The minimum number of bits per color component for the output device |
| 21. | min-color-index | The minimum number of colors the device can display |
| 22. | min-height | The minimum height of the display area, such as a browser window |
| 23. | min-monochrome | The minimum number of bits per "color" on a monochrome (greyscale) device |
| 24. | min-resolution | The minimum resolution of the device, using dpi or dpcm |
| 25. | min-width | The minimum width of the display area, such as a browser window |
| 26. | monochrome | The number of bits per "color" on a monochrome (greyscale) device |
| 27. | orientation | The orientation of the viewport (landscape or portrait mode) |
| 28. | overflow-block | How does the output device handle content that overflows the viewport along the block axis |

| Sl.No | Value | Description |
|-------|-----------------|--|
| 29. | overflow-inline | Can content that overflows the viewport along the inline axis be scrolled |
| 30. | pointer | Is the primary input mechanism a pointing device, and if so, how accurate is it? |
| 31. | resolution | The resolution of the output device, using dpi or dpcm |
| 32. | scan | The scanning process of the output device |
| 33. | scripting | Is scripting (e.g. JavaScript) available? |
| 34. | update | How quickly can the output device modify the appearance of the content |
| 35. | width | The viewport width |

Example:

```

@media print {
  body { font-size: 10pt; }
}

@media screen {
  body { font-size: 13px; }
}

@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}

/* Nested within another conditional at-rule */
@supports (display: flex) {
  @media screen and (min-width: 900px) {
    article {
      display: flex;
    }
  }
}

/* Extra small devices (phones, 600px and down) */

```

```
@media only screen and (max-width: 600px) { }

/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) { }

/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) { }

/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) { }

/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) { }
```

Q. What is the use of `@media only screen`?

It hide style sheets from older user agents.

Q. Does the `screen` keyword apply to the device's physical screen or the browser's viewport?

Browser's Viewport

Q. How would you implement a web design comp that uses non-standard fonts?

Use `@font-face` and define `font-family` for different `font-weight`s.

Q. How a browser determines what elements match a CSS selector?

Browsers match selectors from rightmost (key selector) to left. Browsers filter out elements in the DOM according to the key selector and traverse up its parent elements to determine matches. The shorter the length of the selector chain, the faster the browser can determine if that element matches the selector.

For example with this selector `p span`, browsers firstly find all the `` elements and traverse up its parent all the way up to the root to find the `<p>` element. For a particular ``, as soon as it finds a `<p>`, it knows that the `` matches and can stop its matching.

Q. How can you load css resources conditionally?

@import: allows to load stylesheet by using a path (uri) representing the location of the file.

```
/* By default, include the "light" color theme for syntax highlighting */
@import "cdn.com/atom-one-light.min.css";
/* And if you're in dark mode, have those rules superseded via a different
stylesheet */
@media (prefers-color-scheme: dark) {
  @import "cdn.com/atom-one-dark.min.css";
}
```

matchMedia(): Using matchMedia lets you execute blocks of JavaScript only when a certain media query condition is met. This means you can just write out the CSS when and if the query is true:

```
if (window.matchMedia('screen and (min-width: 600px)')) {
  document.write('<link rel="stylesheet" href="css/small.css">');
}
```

Q. What does `* { box-sizing: border-box; }` do? What are its advantages?

- Make every element in the document include the padding and border in the element's inner dimensions; making it easier to reason about the layout of elements on the page.
- By default, elements have `box-sizing: content-box` applied, and only the content size is being accounted for.
- `box-sizing: border-box` changes how the `width` and `height` of elements are being calculated, `border` and `padding` are also being included in the calculation.
- The `height` of an element is now calculated by the content's `height` + vertical `padding` + vertical `border` width.
- The `width` of an element is now calculated by the content's `width` + horizontal `padding` + horizontal `border` width.
- Taking into account `padding`s and `border`s as part of our box model resonates better with how designers actually imagine content in grids.

Q. List display property in CSS?

The display property specifies the display behavior (the type of rendering box) of an element.

Example:

```
p.ex1 {display: none;}
p.ex2 {display: inline;}
p.ex3 {display: block;}
p.ex4 {display: inline-block;}
```

Property Values:

| Sl.No | Value | Description |
|-------|--------------|---|
| 01. | inline | Displays an element as an inline element (like <code></code>). Any height and width properties will have no effect |
| 02. | block | Displays an element as a block element (like <code><p></code>). It starts on a new line, and takes up the whole width |
| 03. | contents | Makes the container disappear, making the child elements children of the element the next level up in the DOM |
| 04. | flex | Displays an element as a block-level flex container |
| 05. | grid | Displays an element as a block-level grid container |
| 06. | inline-block | Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values |
| 07. | inline-flex | Displays an element as an inline-level flex container |
| 08. | inline-grid | Displays an element as an inline-level grid container |
| 09. | inline-table | The element is displayed as an inline-level table |
| 10. | list-item | Let the element behave like a <code></code> element |
| 11. | run-in | Displays an element as either block or inline, depending on context |
| 12. | table | Let the element behave like a <code><table></code> element |

| Sl.No | Value | Description |
|-------|--------------------|---|
| 13. | table-caption | Let the element behave like a <code><caption></code> element |
| 14. | table-column-group | Let the element behave like a <code><colgroup></code> element |
| 15. | table-header-group | Let the element behave like a <code><thead></code> element |
| 16. | table-footer-group | Let the element behave like a <code><tfoot></code> element |
| 17. | table-row-group | Let the element behave like a <code><tbody></code> element |
| 18. | table-cell | Let the element behave like a <code><td></code> element |
| 19. | table-column | Let the element behave like a <code><col></code> element |
| 20. | table-row | Let the element behave like a <code><tr></code> element |
| 21. | none | The element is completely removed |
| 22. | initial | Sets this property to its default value. Read about initial |
| 23. | inherit | Inherits this property from its parent element. Read about inherit |

Q. How is responsive design different from adaptive design?

1) Responsive design uses `CSS media queries` to change styles based on the target device such as display type, width, height, etc., and only one of these is necessary for the site to adapt to different screens.

Responsive doesn't offer as much control as adaptive, but takes much less work to both build and maintain. Responsive layouts are also fluid and whilst adaptive can and do use percentages to give a more fluid feel when scaling, these can again cause a jump when a window is resized.

2) Adaptive design uses static layouts based on breakpoints which don't respond once they're initially loaded. Adaptive works to detect the screen size and load the appropriate layout for it. Generally adaptive site uses six common screen widths:

- 320 px
- 480 px
- 760 px
- 960 px
- 1200 px
- 1600 px

Adaptive is useful for retrofitting an existing site in order to make it more mobile-friendly. This allows you to take control of the design and develop for specific, multiple viewports.

Q. What is retina graphics? What kind of techniques do you use to handle images for retina screens?

In order to have crisp, good-looking graphics that make the best of retina displays we need to use high resolution images whenever possible. However using always the highest resolution images will have an impact on performance as more bytes will need to be sent over the wire.

To overcome this problem, we can use responsive images, as specified in HTML5. It requires making available different resolution files of the same image to the browser and let it decide which image is best, using the html attribute `srcset` and optionally `sizes`, for instance:

```
<div responsive-background-image>
  
</div>

<!-- It enable the browser to use a medium and large image based on
      the sizes of the image in the viewport
-->

```

The browsers which does not support HTML5's `srcset` (i.e. IE11) will ignore it and uses `src` instead. If we really need to support IE11 and we want to provide this feature for performance reasons, we can use a JavaScript `polyfill`.

HTML5 picture Element:

```
<picture>
  <source media="(min-width: 1024px)" srcset="foo-large.jpg 1024w, foo-medium.jpg 640w" sizes="50vw" />
  <source srcset="foo@2x.jpg 2x, foo.jpg 1x" />
  
</picture>
```

Retina Display Media Query:

```
/* 1.25 dpr */
@media (-webkit-min-device-pixel-ratio: 1.25), (min-resolution: 120dpi) {
  /* Retina-specific stuff here */
}

/* 1.3 dpr */
@media (-webkit-min-device-pixel-ratio: 1.3), (min-resolution: 124.8dpi) {
  /* Retina-specific stuff here */
}

/* 1.5 dpr */
@media (-webkit-min-device-pixel-ratio: 1.5), (min-resolution: 144dpi) {
  /* Retina-specific stuff here */
}
```

Q. Is there any reason you'd want to use `translate()` instead of absolute positioning, or vice-versa?

`translate()` is a value of CSS `transform`. Changing `transform` or `opacity` does not trigger browser reflow or repaint but does trigger compositions; whereas changing the absolute positioning triggers `reflow`. `transform` causes the browser to create a GPU layer for the element but changing absolute positioning properties uses the CPU. Hence `translate()` is more efficient and will result in shorter paint times for smoother animations.

When using `translate()`, the element still occupies its original space (sort of like `position: relative`), unlike in changing the absolute positioning.

Example:

If we combine `position: relative` with one of the offset properties `top`, `bottom`, `left` or `right` the element will be moved from its original

place in the layout whilst preserving the space in the document it once occupied. The element will be moved on to a new layer and its “layer order” or its stacking order can then be controlled with the `z-index` property.

```
.thing {  
  position: relative;  
  top: 100px;  
  left: 50px;  
}
```

In the above example the element will be moved 100px away from the top and 50px away from the left of its original position.

When using `transform:translate(x,y)` we get a very similar visual result to using relative position. The same result as above could be achieved with the following snippet:

```
.thing {  
  transform: translate(50px, 100px);  
}
```

In this case, we are translating the coordinates of the element by `50px` along the x-axis and `100px` along the y-axis. The end result is visually the same as the previous `position` example.

Q. The `translate()` function can move the position of an element on the z-axis?

- False

Q. Tell what each of these tags do, if there are alternatives, which are preferable, why?

``: The HTML `` tag represents stress emphasis of its contents.

```
<em>Emphasized content...</em>
```

``: The `` tag specifies bold text without any extra importance.

```
<p>This is normal text - <b>and this is bold text</b>.</p>
```

<abbr>: The HTML Abbreviation element (`<abbr>`) represents an abbreviation or acronym; the optional title attribute can provide an expansion or description for the abbreviation.

The `<abbr title="World Health Organization">WHO</abbr>` was founded in 1948.

<nav>: The `<nav>` tag defines a set of navigation links.

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

<i>: The content of the `<i>` tag is usually displayed in italic.

`<p>I looked at it and thought <i>This can't be real!</i></p>`

<link>: The HTML `<link>` tag is used for defining a link to an external resource. It is placed in in the `<head>` section of the HTML document.

```
<head>
  <link rel="stylesheet" type="text/css" href="theme.css">
</head>
```

****: The `` element is used to identify text that is of greater importance than the surrounding text. By default, all browsers render `` text in a bold typeface.

`Strong text`

<article>: The `<article>` tag specifies independent, self-contained content.

```
<article>
  <h1>Google Chrome</h1>
  <p>Google Chrome is a free, open-source web browser developed by Google,
  released in 2008.</p>
</article>
```

Q. What is At-Rule?

At-rules are `CSS statements` that instructs CSS how to behave. They begin with an at sign, `@` followed by an identifier and includes everything up to the next semicolon, `;` or the next CSS block, whichever comes first.

```
/* General structure */
```

```
@IDENTIFIER (RULE);
```

```
/* Example: tells browser to use UTF-8 character set */  
@charset "utf-8";
```

| Sl.No | at-rules | Description |
|-------|------------|---|
| 01. | @charset | Defines the character set used by the style sheet. |
| 02. | @import | Tells the CSS engine to include an external style sheet. |
| 03. | @namespace | Tells the CSS engine that all its content must be considered prefixed with an XML namespace. |
| 04. | @media | A conditional group rule that will apply its content if the device meets the criteria of the condition defined using a media query. |
| 05. | @supports | A conditional group rule that will apply its content if the browser meets the criteria of the given condition. |
| 06. | @page | Describes the aspect of layout changes that will be applied when printing the document. |
| 07. | @font-face | Describes the aspect of an external font to be downloaded. |
| 08. | @keyframes | Describes the aspect of intermediate steps in a CSS animation sequence. |

Q. How can the gap under the image be removed?

As images being inline elements are treated same as texts, so there is a gap left, which can be removed by:

```
<figure>  
    
</figure>
```

Q. What is progressive rendering?

Progressive rendering is the name given to techniques used to improve the performance of a webpage (in particular, improve perceived load time) to render content for display as quickly as possible.

Examples:

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

Q. What is mobile-first? Can you explain the difference between coding a website to be responsive versus using a mobile-first strategy?

Making a website responsive means the some elements will respond by adapting its size or other functionality according to the device's screen size, typically the viewport width, through CSS media queries, for example, making the font size smaller on smaller devices.

```
@media (min-width: 601px) {  
  .my-class {  
    font-size: 24px;  
  }  
}  
@media (max-width: 600px) {  
  .my-class {  
    font-size: 12px;  
  }  
}
```

A mobile-first strategy is also responsive, however it agrees we should default and define all the styles for mobile devices, and only add specific responsive rules to other devices later. Following the previous example:

```
.my-class {  
  font-size: 12px;  
}  
  
@media (min-width: 600px) {  
  .my-class {  
    font-size: 24px;  
  }  
}
```

A mobile-first strategy has 2 main advantages:

- It's more performant on mobile devices, since all the rules applied for them don't have to be validated against any media queries.
- It forces to write cleaner code in respect to responsive CSS rules.

Example:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>CSS Media Query</title>  
    <style>  
      body {  
        background-color: lightgreen;  
      }  
  
      @media only screen and (max-width: 800px) {  
        body {  
          background-color: rgb(233, 50, 18);  
        }  
      }  
    </style>  
  </head>  
  <body>  
    <h1>Resize the browser window</h1>  
    <h2>When the width of this document is 800px or less, the background-color is  
"green",  
    otherwise it is "Red".</h2>  
  </body>  
</html>
```

Live Demo: [Media Query](#)

Q. Which property is used to change the face of a font?

The `font-family` property is used to change the face of a font.

Q. Which property is used to make a font italic or oblique?

The `font-style` property is used to make a font italic or oblique.

Q. Which property is used to create a small-caps effect?

The `font-variant` property is used to create a small-caps effect.

Q. Which property is used to increase or decrease how bold or light a font appears?

The `font-weight` property is used to increase or decrease how bold or light a font appears.

Q. Which property is used to add or subtract space between the letters that make up a word?

The `letter-spacing` property is used to add or subtract space between the letters that make up a word.

Q. Which property is used to add or subtract space between the words of a sentence?

The `word-spacing` property is used to add or subtract space between the words of a sentence.

Q. Which property is used to indent the text of a paragraph?

The `text-indent` property is used to indent the text of a paragraph.

Q. Which property is used to align the text of a document?

The `text-align` property is used to align the text of a document.

Q. Which property is used to underline, overline, and strikethrough text?

The `text-decoration` property is used to underline, overline, and strikethrough text.

Q. Which property is used to capitalize text or convert text to uppercase or lowercase letters?

The `text-transform` property is used to capitalize text or convert text to uppercase or lowercase letters.

Q. Which property allows you to control the shape or appearance of the marker of a list?

The `list-style-type` allows you to control the shape or appearance of the marker.

Q. How do I restore the default value of a property?

The keyword `initial` can be used to reset it to its default value, which is defined in the CSS specification of the given property.

Q. What is specificity?

A process of determining which css rule will be applied to an element. It actually determines which rules will take precedence. Inline style usually wins then ID then class value (or pseudo-class or attribute selector), universal selector (*) has no specificity. ID selectors have a higher specificity than attribute selectors.

Selector Types

The following list of selector types increases by specificity:

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

```
/*wins*/  
a#a-02 { background-image : url(n.gif); }  
a[id="a-02"] { background-image : url(n.png); }
```

Contextual selectors are more specific than a single element selector. The embedded style sheet is closer to the element to be styled. The last rule defined overrides any previous, conflicting rules.

```
p { color: red; background: yellow }  
p { color: green } // wins
```

A class selector beats any number of element selectors.

```
.introduction {} //wins  
html body div div h2 p {}
```

Q. What do you know about CSS Transitions?

CSS Transitions allows to add an effect while changing from one style to another. You can set the which property you want to transition, duration, how you want to transit (linear, ease, ease-in, ease-out, cubic-bezier) and delay when transition will start.

CSS Transition Properties

| Sl.No | Property | Description |
|-------|----------------------------|--|
| 01. | transition | A shorthand property for setting the four transition properties into a single property |
| 02. | transition-delay | Specifies a delay (in seconds) for the transition effect |
| 03. | transition-duration | Specifies how many seconds or milliseconds a transition effect takes to complete |
| 04. | transition-property | Specifies the name of the CSS property the transition effect is for |
| 05. | transition-timing-function | Specifies the speed curve of the transition effect |

Example: 01

```
div {
  width: 100px;
  height: 100px;
  background: red;
  -webkit-transition: width 2s; /* Safari prior 6.1 */
  transition: width 2s;
}
```

Live Demo: [CSS Transitions](#)

Q. What are the different css filters you can use?

The filter CSS property applies graphical effects like blur or color shift to an element. Filters are commonly used to adjust the rendering of images, backgrounds, and borders.

Example:

```
img {
  -webkit-filter: brightness(200%); /* Safari 6.0 - 9.0 */
  filter: brightness(200%);
}
```

}

Filter Functions

| Sl.No | Filter | Description |
|-------|--|---|
| 01. | none | Default value. Specifies no effects |
| 02. | blur(px) | Applies a blur effect to the image. A larger value will create more blur. |
| 03. | brightness(%) | Adjusts the brightness of the image. |
| 04. | contrast(%) | Adjusts the contrast of the image. |
| 05. | drop-shadow(h-shadow v-shadow blur spread color) | Applies a drop shadow effect to the image. |
| 06. | grayscale(%) | Converts the image to grayscale. |
| 07. | hue-rotate(deg) | Applies a hue rotation on the image. The value defines the number of degrees around the color circle |
| 08. | invert(%) | Inverts the samples in the image. |
| 09. | opacity(%) | Sets the opacity level for the image. The opacity-level describes the transparency-level |
| 10. | saturate(%) | Saturates the image. |
| 11. | sepia(%) | Converts the image to sepia. |
| 12. | url() | The url() function takes the location of an XML file that specifies an SVG filter, and may include an anchor to a specific filter element. for example <code>filter: url(svg-url#element-id)</code> |

| Sl.No | Filter | Description |
|-------|---------|---|
| 13. | initial | Sets this property to its default value. |
| 14. | inherit | Inherits this property from its parent element. |

Q. Enlist the various fonts properties?

| Property | Description |
|-----------------------|--|
| font-style | Specifies the font style. |
| font-variant | Specifies the font variant. |
| font-weight | Specifies the font weight. |
| font-size/line-height | Specifies the font size and the line-height. |
| font-family | Specifies the font family. Default value depends on the browser |
| caption | Uses the font that are used by captioned controls (like buttons, drop-downs, etc.) |
| icon | Uses the font that are used by icon labels |
| menu | Uses the fonts that are used by dropdown menus |
| message-box | Uses the fonts that are used by dialog boxes |
| small-caption | A smaller version of the caption font |
| status-bar | Uses the fonts that are used by the status bar |

| Property | Description |
|----------|---|
| initial | Sets this property to its default value. |
| inherit | Inherits this property from its parent element. |

Q. Enlist the various background properties css?

| Property | Description |
|-----------------------|--|
| background-color | Specifies the background color to be used |
| background-image | Specifies ONE or MORE background images to be used |
| background-position | Specifies the position of the background images |
| background-size | Specifies the size of the background images |
| background-repeat | Specifies how to repeat the background images |
| background-origin | Specifies the positioning area of the background images |
| background-clip | Specifies the painting area of the background images |
| background-attachment | Specifies whether the background images are fixed or scrolls with the rest of the page |
| initial | Sets this property to its default value. |
| inherit | Inherits this property from its parent element |

Q. What is the difference between `em` and `rem` units?

Both em and rem units are based on the font-size CSS property. The only difference is where they inherit their values from.

- `em` units inherit their value from the font-size of the parent element

```
.parent {
  font-size: 18px;
}
.child {
  font-size: 1.5em;
}
<div class="parent">
  I'm 15px
  <div class="child">
    I'm 30px, as expected
    <div class="child">
      I'm 60px, trouble starts!
      <div class="child">
        I'm 120px, now we're really in trouble!
      </div>
    </div>
  </div>
</div>
```

- `rem` units inherit their value from the font-size of the root element (html)

```
.html {
  font-size: 16px;
}
.parent {
  font-size: 15px;
}
.child-rem {
  font-size: 2rem;
}
<div class="parent">
  I'm 15px
  <div class="child-rem">
    I'm 32px, as expected
    <div class="child-rem">
      I'm 32px, yep!
      <div class="child-rem">
        I'm 32px, like clockwork!
      </div>
    </div>
  </div>
</div>
```

Note: In most browsers, the font-size of the root element is set to 16px by default.

Q. What does !important mean in CSS?

It overrides the cascade and gives the style rule the highest precedence.

```
p {  
  color: red !important;  
}  
#thing {  
  color: green;  
}  
<p id="thing">Will be RED.</p>
```

Q. Does style2.css have to be downloaded and parsed before Paragraph 1 is rendered on the page?

```
<head>  
  <link href="style1.css" rel="stylesheet">  
</head>  
<body>  
  <p>Paragraph 1</p>  
  <p>Paragraph 2</p>  
  <link href="style2.css" rel="stylesheet">  
</body>
```

- Yes

Q. Does style1.css have to be downloaded and parsed before style2.css can be fetched?

```
<head>  
  <link href="style1.css" rel="stylesheet">  
  <link href="style2.css" rel="stylesheet">  
</head>
```

- No

Q. What are the CSS positioning?

| Keyword | Value | Description |
|----------|------------------------|--|
| position | static | The default mode, block element is positioned in the flow. Top, left etc. are ignored. |
| position | relative | The block element is positioned relative to its position in the flow. |
| position | absolute | Block element is positioned relative to its container. |
| position | fixed | Block element is positioned relative to the window and won't scroll. |
| top | Number [px, cm, in...] | Positions the block down from the reference point at the specified distance. |
| bottom | Number [px, cm, in...] | Positions the block up from the reference point at the specified distance. |
| left | Number [px, cm, in...] | Positions the block right from the reference point at the specified distance. |
| right | Number [px, cm, in...] | Positions the block left from the reference point at the specified distance. |

Q. What is the difference between padding and margin?

1) Margin is applied to the outside of your element hence effecting how far your element is away from other elements.

2) Padding is applied to the inside of your element hence effecting how far your element's content is away from the border.

Also, using margin will not affect your element's dimensions whereas padding will make your elements dimensions (set height + padding) so for example if you have a 100x100px div with a 5 px padding, your div will actually be 105x105px

Note: **Top/Bottom margins are collapsible:** if you have a 20px margin at the bottom of an element and a 30px margin at the top of the next element, the margin between the two elements will be 30px rather than 50px. This does not apply to left/right margin or padding.

Q. What are gradients in CSS?

CSS gradients are represented by the `<gradient>` data type, a special type of `<image>` made of a progressive transition between two or more colors. There are three types of gradients:

- linear (created with the `linear-gradient()` function),
- radial (created with `radial-gradient()`), and
- conic (created with the `conic-gradient()` function).

We can also create repeating gradients with the `repeating-linear-gradient()`, `repeating-radial-gradient()`, and `repeating-conic-gradient()` functions.

```
/* Example - 01: A basic linear gradient */
.simple-linear {
  background: linear-gradient(blue, pink);
}

/* Example - 02: Changing the direction */
.horizontal-gradient {
  background: linear-gradient(to right, blue, pink);
}

/* Example - 03: Diagonal gradients */
.diagonal-gradient {
  background: linear-gradient(to bottom right, blue, pink);
}

/* Example - 04: Using angles */
.angled-gradient {
  background: linear-gradient(70deg, blue, pink);
}

/* Example - 05: Creating hard lines */
.stripped {
```



```
background: linear-gradient(to bottom left, cyan 50%, palegoldenrod 50%);
}
```

Q. What is CSS opacity?

The `opacity` CSS property sets the opacity of an element. Opacity is the degree to which content behind an element is hidden, and is the opposite of transparency.

```
div { background-color: lightblue; }
.light {
  opacity: 30%; /* Barely see the text over the background */
}
.medium {
  opacity: 60%; /* See the text more clearly over the background */
}
.heavy {
  opacity: 100%; /* See the text very clearly over the background */
}
<div class="light">You can barely see this.</div>
<div class="medium">This is easier to see.</div>
<div class="heavy">This is very easy to see.</div>
```

Q. How is the concept of inheritance applied in CSS?

Inheritance is a concept in which the child class will inherit the properties of its parent class. It is used in CSS to define the hierarchy from the top level to the bottom level. Inherited properties can be overridden by the children class if the child uses the same name.

Example:

```
span {
  color: blue;
  border: 1px solid black;
}
.extra span {
  color: inherit;
}
```

Q. How do you handle browser differences in your user base?

The `@supports` query in CSS can be very useful to scan if the user's current browser has a certain feature. The `@supports` CSS at-rule lets you specify declarations that depend on a browser's support for one or more specific CSS features. This is called a feature query. The rule may be placed at the top level of your code or nested inside any other conditional group at-rule.

```
@supports (display: grid) {  
  div {  
    display: grid;  
  }  
}  
  
@supports not (display: grid) {  
  div {  
    float: right;  
  }  
}
```

Q. What is Cascade?

Cascade is a method of defining the weight (importance) of individual styling rules thus allowing conflicting rules to be sorted out should such rules apply to the same selector.

```
P {color: white ! important} /* increased weight */  
P {color: black} /* normal weight */
```

Q. What are CSS vendor prefixes?

Vendor prefixes are extensions to CSS standards that can be added to these features to prevent incompatibilities from arising when the standard is extended. CSS vendor prefixes for some common platforms are listed below.

- `-webkit-`: Android, Chrome, iOS, and Safari
- `-moz-`: Mozilla Firefox
- `-ms-`: Internet Explorer
- `-o-`: Opera

Q. What is DOM (Document Object Model) and how is it linked to CSS?

The Document Object Model (DOM) is a programming interface for HTML and XML(Extensible markup language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated. This document enables Javascript to access and manipulate the elements and styles of a website. The model is built in a tree structure of objects and defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

DOM Document

The DOM Document is the owner of all other objects in your webpage. That means if you want to access any object on your webpage you always have to start with the document. It also contains many important properties and methods that enable us to access and modify our website.

Finding HTML Elements

| Methods | Description |
|--------------------------|---|
| getElementById() | This method is used to get a single element by its id. |
| getElementsByClassName() | This method returns an array of elements. |
| getElementsByTagName() | This method is used to get a single element by its tag name |
| querySelector() | This method returns the first element that matches a specified CSS selector. It can get |

| Methods | Description |
|--------------------|--|
| | elements by id, class, tag and all other valid CSS selectors. |
| querySelectorAll() | This method is completely the same as the querySelector() except that it returns all elements that fit the CSS Selector. |

Changing HTML Elements

The innerHTML property can be used to change the content of an HTML element. In this example we get the element with an id of header and set the inner content to "Hello World!".

```
// Example: Using text
document.getElementById("#header").innerHTML = "Hello World!";

// Example: Using text with tag
document.getElementsByTagName("div").innerHTML = "<h1>Hello World!</h1>"
```

Changing a value of an attribute

We can also change the value of an attribute using the DOM.

```
document.getElementsByTagName("img").src = "image.jpg";
```

Changing the style

To change the style of an HTML element we need to change the style property of our elements. The CSS properties need to be written in camelcase instead of the normal css property name.

```
document.getElementsByTagName("h1").style.borderBottom = "solid 3px #000";
```

Adding and deleting elements

Adding elements

create a div element using the createElement() method which takes a tagname as a parameter and saves it into a variable. After that we just need to give it some content and then insert it into our DOM document.

```
var div = document.createElement("div");
```

```
var newContent = document.createTextNode("Hello World!");
div.appendChild(newContent);
document.body.insertBefore(div, currentDiv);
```

Here we create content using the `createTextNode()` method which takes a String as a parameter and then we insert our new div element before a div that already exists in our document.

Deleting elements

Here we get an element and delete it using the `removeChild()` method.

```
var elem = document.querySelector('#header');
elem.parentNode.removeChild(elem);
```

Replace elements

```
var div = document.querySelector('#div');
var newDiv = document.createElement('div');

newDiv.innerHTML = "Hello World2";
div.parentNode.replaceChild(newDiv, div);
```

Here we replace an element using the `replaceChild()` method. The first argument is the new element and the second argument is the element which we want to replace.

Writing directly into the HTML output stream

We can also write HTML expressions and JavaScript directly into the HTML output stream using the `write()` method. The `write()` method can also take multiple arguments that will be appended to the document in order of their occurrence.

```
// HTML Content
document.write("<h1>Hello World!</h1><p>This is a paragraph!</p>");

// date object
document.write(Date());
```

Event Handlers

The HTML DOM also allows Javascript to react to HTML events. for example, mouse click, page load, mouse move, input field change etc.

Assign Events

You can define events directly in your js code. Here is an example of an onclick event:

```
document.getElementById("btn").onclick = changeText();
```

Assign Events Listeners

Here we just assigned a click event that calls the runEvent method when our btn element is clicked.

```
document.getElementById("btn").addEventListener('click', runEvent);
```

Node Relationships

The nodes in the DOM Document have a hierarchical relationship to each other. This means that the nodes are structured like a tree. We use the terms parent, sibling and child to describe the relationship between nodes.

The top node is called the root and is the only node that has no parent. The root in a normal HTML document is the `<html/>` tag because it has no parent and is the top tag of the document.

Navigating Between Nodes

We can navigate between nodes using these properties:

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling

Example:

```
var parent = document.getElementById("heading").parentNode
```

Q. Explain the usage of "table-layout" property?

The **table-layout** CSS property sets the algorithm used to lay out `<table>` cells, rows, and columns.

Syntax

```
table-layout: auto|fixed|initial|inherit;
```

Property Values

| Value | Description |
|----------|---|
| auto: | It is used to set the automatic table layout on the browser. This property set the column width by unbreakable content in the cells. |
| fixed: | It is used to set a fixed table layout. The table and column widths are set by the widths of table and col or by the width of the first row of cells. Cells in other rows do not affect column widths. If no widths are present on the first row, the column widths are divided equally across the table according to content of table. |
| initial: | It is used to set its default value. |
| inherit: | It is used to inherit the property from its parent. |

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>table-layout Property</title>
    <style>
      table {
        width: 100%;
        table-layout: fixed;
        border-collapse: collapse;
      }

      th, td {
        padding: 8px;
        border: 1px solid #dee2e6;
      }

      th {
        height: 40px;
        text-align: left;
      }
    </style>
  </head>
  <body>
    <h2>table-layout Property</h2>
    <table>
      <thead>
        <tr>
          <th>Row</th>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Email</th>
        </tr>
      </thead>
```

```

<tbody>
  <tr>
    <td>1</td>
    <td>Clark</td>
    <td>Kent</td>
    <td>clarkkent@mail.com</td>
  </tr>
  <tr>
    <td>2</td>
    <td>John</td>
    <td>Carter</td>
    <td>johncarter@mail.com</td>
  </tr>
  <tr>
    <td>3</td>
    <td>Peter</td>
    <td>Parker</td>
    <td>peterparker@mail.com</td>
  </tr>
</tbody>
</table>
</body>
</html>

```

Live Demo: [Table-layout Property](#)

Note: The main benefit of `table-layout: fixed;` is that the table renders much faster. On large tables, users will not see any part of the table until the browser has rendered the whole table. So, if you use `table-layout: fixed`, users will see the top of the table while the browser loads and renders rest of the table. This gives the impression that the page loads a lot quicker!

Q. Does margin-top or margin-bottom have an effect on inline elements

Top and bottom margins do not affect inline elements because inline elements flow with content on the page. You can set left and right margins/padding on an inline element but not top or bottom because it would disrupt the flow of content. You can set margins on block (or inline-block but it will only look right if you set the vertical align right) because block level elements disrupt the flow of content.

Example:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Separator</title>
  </head>

```



```

<style>
  .separator {
    border: 1px solid #333;
    margin: 10px;
  }
</style>
<body>
  <div>
    <span>Hello</span>
    <span class="separator"></span>
    <span>World</span>
  </div>
</body>
</html>

```

Live Demo: [Inline Elements](#)

Q. How does Calc() work?

The `calc()` function can be used to perform addition, subtraction, multiplication, and division calculations with numeric property values. Specifically, it can be used with `<length>`, `<frequency>`, `<angle>`, `<time>`, `<number>`, or `<integer>` data types.

Example:

```

/* Example - 1 */
.main-content {
  width: calc(100vh - 10px); /* Subtract 10px from 100vh */
}

/* Example - 2 */
.container {
  padding: calc(1vw + 1em);
  width: calc(var(--variable-width) + 200px);
  transform: rotate( calc(1turn + 28deg) );
  background: hsl(100, calc(3 * 20%), 40%);
  font-size: calc(50vw / 3);
  border-radius: 15px calc(15px / 3) 4px 2px;
}

```

Q. What do CSS Custom properties variables mean?

Custom properties (sometimes referred to as **CSS variables** or **cascading variables**) are entities defined by CSS authors that contain specific values to be reused throughout a document. They are set using custom property notation (e.g., `--main-`

color: black;) and are accessed using the `var()` function (e.g., `color: var(--main-color);`).

Property names that are prefixed with `--`, like `--example-name`, represent custom properties that contain a value that can be used in other declarations using the `var()` function.

Syntax

```
--somekeyword: left;
--somecolor: #0000ff;
--somecomplexvalue: 3px 6px rgb(20, 32, 54);
```

Example:

```
:root {
  --first-color: #16f;
  --second-color: #ff7;
}

#firstParagraph {
  background-color: var(--first-color);
  color: var(--second-color);
}

#secondParagraph {
  background-color: var(--second-color);
  color: var(--first-color);
}

#container {
  --first-color: #290;
}

#thirdParagraph {
  background-color: var(--first-color);
  color: var(--second-color);
}

<p id="firstParagraph">This paragraph should have a blue background and yellow text.</p>
<p id="secondParagraph">This paragraph should have a yellow background and blue text.</p>
<div id="container">
  <p id="thirdParagraph">This paragraph should have a green background and yellow text.</p>
</div>
```

Live Demo: [Custom Properties Variables](#)

Q. What is the difference between CSS variables and preprocessor(SASS, LESS, Stylus) variables?

SASS variables are replaced with their values as the preprocessor produces its CSS output long before the browser interprets the code, while CSS custom properties are evaluated by the browser at runtime.

Example: Preprocessor Variable

```
$brandColor: #F06D06;

.main-header {
  color: $brandColor;
}
.main-footer {
  background-color: $brandColor;
}
```

The above code would do nothing in a browser. The browser wouldn't understand the declarations and toss them out. Preprocessors need to compile into CSS to be used. This code would compile to:

```
.main-header {
  color: #F06D06;
}
.main-footer {
  background-color: #F06D06;
}
```

This is now valid CSS. The variable was part of the preprocessor language, not CSS itself. Once the code compiles, the variables are gone.

Example: CSS Custom Property

The native CSS has started supporting CSS variables, or "CSS Custom Properties". It allows you to work with variables directly in CSS. There is no compiling.

```
:root {
  --main-color: #F06D06;
}

.main-header {
  color: var(--main-color);
}
.main-footer {
  background-color: var(--main-color);
}
```

Q. What are Vendor-Prefixes?

CSS vendor prefixes, also sometimes known as or CSS browser prefixes, are a way for browser makers to add support for new CSS features before those features are fully supported in all browsers.

Example: `transition` property

```
.myClass {  
  -webkit-transition: all 1s linear;  
  -moz-transition: all 1s linear;  
  -ms-transition: all 1s linear;  
  -o-transition: all 1s linear;  
  transition: all 1s linear;  
}
```

CSS prefixes

The major browsers use the following prefixes:

- `-webkit-` (Chrome, Safari, newer versions of Opera, almost all iOS browsers including Firefox for iOS; basically, any WebKit based browser)
- `-moz-` (Firefox)
- `-o-` (old pre-WebKit versions of Opera)
- `-ms-` (Internet Explorer and Microsoft Edge)

Q. What is the property that is used for controlling image-scroll?

The **background-attachment** property in CSS is used to specify the kind of attachment of the background image with respect to its container. It can be set to scroll or remain fixed. It can be applied to all HTML elements.

Syntax

```
background-attachment: scroll|fixed|local|initial|inherit;
```

Property Values

| Value | Description |
|---------|--|
| scroll | The background image will scroll with the page. This is default |
| fixed | The background image will not scroll with the page |
| local | The background image will scroll with the element's contents |
| initial | Sets this property to its default value. Read about initial |
| inherit | Inherits this property from its parent element. Read about inherit |

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-image: url("../images/img_tree.gif");
        background-repeat: no-repeat;
        background-attachment: fixed;
      }
    </style>
  </head>
<body>
  <h1>The background-attachment Property</h1>

  <p>The background-image is fixed. Try to scroll down the page.</p>
  ...
  <p>If you do not see any scrollbars, try to resize the browser window.</p>
</body>
</html>
```

Live Demo: [CSS background-attachment](#)

Q. What is the overflow property in CSS used for?

The CSS overflow property specifies how to handle the content when it overflows its block level container.

Syntax

overflow: visible|hidden|scroll|auto|initial|inherit;

Property Values

| Value | Description |
|------------|--|
| visible | The overflow is not clipped. It renders outside the element's box. |
| hidden | The overflow is clipped, and the rest of the content will be invisible |
| scroll | The overflow is clipped, but a scroll-bar is added to see the rest of the content |
| auto | If overflow is clipped, a scroll-bar should be added to see the rest of the content |
| initial | Sets this property to its default value. |
| inherit | Inherits this property from its parent element. |
| overflow-x | Specifies what to do with the left/right edges of the content if it overflows the element's content area |
| overflow-y | Specifies what to do with the top/bottom edges of the content if it overflows the element's content area |

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS Overflow Property</title>
    <style>
      div {
        background-color: #eee;
        width: 200px;
        height: 100px;
        border: 1px dotted black;
        overflow: scroll;
        padding: 10px;
      }
    </style>
  </head>
  <body>
```

```
<h2>CSS Overflow Property</h2>
<p>Setting the overflow value to scroll, the overflow is clipped and a scrollbar
  is added to scroll inside the box.
</p>

<div>You can use the overflow property when you want to have better control of
  the layout. The overflow property specifies what happens if content overflows
  an element's box.
</div>
</body>
</html>
```

Q. What is word-wrapping in CSS?

The word-wrap property in CSS is used to break long word and wrap into the next line. It defines whether to break words when the content exceeds the boundaries of its container.

Syntax

```
word-wrap: normal|break-word|initial|inherit;
```

Property Values

| Value | Description |
|------------|---|
| normal | Break words only at allowed break points |
| break-word | Allows unbreakable words to be broken |
| initial | Sets this property to its default value. |
| inherit | Inherits this property from its parent element. |

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>The word-wrap Property</title>
  <style>
    div {
      width: 150px;
      border: 1px solid #333;
```

```

    }

    div.a {
        word-wrap: normal;
    }

    div.b {
        word-wrap: break-word;
    }
</style>
</head>
<body>
    <h1>The word-wrap Property</h1>

    <h2>word-wrap: normal (default):</h2>
    <div class="a"> This div contains a very long word:
thisisaveryveryveryveryveryverylongword.
    The long word will break and wrap to the next line.</div>

    <h2>word-wrap: break-word:</h2>
    <div class="b"> This div contains a very long word:
thisisaveryveryveryveryveryverylongword.
    The long word will break and wrap to the next line.</div>
</body>
</html>

```

Live Demo: [The word-wrap Property](#)

Q. What is Accessibility (a11y) in a web application?

Accessibility refers to how software or hardware combinations are designed to make a system accessible to persons with disabilities, such as:

- Visual impairment
- Hearing loss
- Limited dexterity

For example, a website developed with accessibility in mind might have text-to-speech capabilities or output for special braille hardware geared toward individuals with visual impairments.

Q. What is UI/UX?

1) UI or User Interface: is how a product or website is laid out and how you interact with it: Where the buttons are, how big the fonts are, and how menus are organized are all elements of UI.

2) UX or User Experience: is how you feel about using a product or a website. So, your love for the way the new Apple Watch looks or your excitement that there's finally a tablet-sized iPhone to watch those Corgi videos you're obsessed with are reflections of UX. So the new look of the Facebook news feed involves a change to UI, and the way you navigate that new page is the UX.

Q. How do you serve your pages for feature-constrained browsers? What techniques do you use?

- **Graceful degradation:** The practice of building an application for modern browsers while ensuring it remains functional in older browsers.
- **Progressive enhancement:** The practice of building an application for a base level of user experience, but adding functional enhancements when a browser supports it.
- **caniuse:** to check for feature support.
- **Autoprefixer:** for automatic vendor prefix insertion.
- **Feature detection:** using **Modernizr**.
- **CSS Feature queries:** using **@support**

Q. What are the different ways to visually hide content (and make it available only for screen readers)?

These techniques are related to accessibility (a11y).

- **visibility: hidden:** However, the element is still in the flow of the page, and still takes up space.
- **width: 0; height: 0:** Make the element not take up any space on the screen at all, resulting in not showing it.
- **position: absolute; left: -9999px:** Position it outside of the screen.
- **text-indent: -9999px:** This only works on text within the **block** elements.

- **Metadata:** For example by using Schema.org, RDF, and JSON-LD.
- **WAI-ARIA:** A W3C technical specification that specifies how to increase the accessibility of web pages.

Q. What is file splitting? When is it used?

A monolithic file is fine for solo developers or very small projects. For large projects—sites with multiple layouts and content types, or multiple brands under the same design umbrella—it's smarter to use a modular approach and split your CSS across multiple files.

- **reset.css:** reset and normalization styles; minimal color, border, or font-related declarations
- **typography.css:** font faces, weights, line heights, sizes, and styles for headings and body text
- **layouts.css:** styles that manage page layouts and segments, including grids
- **forms.css:** styles for form controls and labels
- **lists.css:** list-specific styles
- **tables.css:** table-specific styles
- **carousel.css:** styles required for carousel components
- **accordion.css:** styles for accordion components

CSS frameworks such as Foundation and Bootstrap use this approach. Both become quite granular with separate files for progress bars, range inputs, close buttons, and tooltips. This allows developers to include only the components that they need for a project.

Q. What is DOM reflow?

Reflow is the name of the web browser process for re-calculating the positions and geometries of elements in the document, for the purpose of re-rendering part or all of the document.

Reflow occurs when

- insert, remove or update an element in the DOM
- modify content on the page, e.g. the text in an input box

- move a DOM element
- animate a DOM element
- take measurements of an element such as `offsetHeight` or `getComputedStyle`
- change a CSS style
- change the `className` of an element
- add or remove a stylesheet
- resize the window
- scroll

Minimizing browser reflow

- Reduce unnecessary DOM depth. Changes at one level in the DOM tree can cause changes at every level of the tree - all the way up to the root, and all the way down into the children of the modified node. This leads to more time being spent performing reflow.
- Minimize CSS rules, and remove unused CSS rules.
- If you make complex rendering changes such as animations, do so out of the flow. Use `position-absolute` or `position-fixed` to accomplish this.
- Avoid unnecessary complex CSS selectors - descendant selectors in particular - which require more CPU power to do selector matching.

Q. How does CSS work under the hood?

The CSS language is designed to be used alongside a "markup" language like HTML. CSS defines how HTML elements are formatted – controlling their layout, colors, fonts, and so on. When a browser displays a document, it must combine the document's content with its style information. It processes the document in a number of stages, which we've listed below.

1. The browser loads the HTML (e.g. receives it from the network).
2. It converts the HTML into a DOM (Document Object Model).
3. The browser then fetches most of the resources that are linked to by the HTML document, such as embedded images and videos and linked CSS.
4. The browser parses the fetched CSS, and sorts the different rules by their selector types into different "buckets", e.g. element, class, ID, and so on. Based on the selectors it finds, it works out which rules should be applied to which nodes in the DOM, and attaches style to them as required (this intermediate step is called a render tree).

5. The render tree is laid out in the structure it should appear in after the rules have been applied to it.
6. The visual display of the page is shown on the screen (this stage is called painting).

The following diagram also offers a simple view of the process.

The DOM and CSSOM:

A DOM has a tree-like structure. Each element, attribute, and piece of text in the markup language becomes a DOM node in the tree structure. The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings. The browser undergoes a process that includes conversion, tokenization, lexing, and parsing which ultimately constructs the DOM and CSSOM.

- **Conversion:** Reading raw bytes of HTML and CSS off the disk or network.
- **Tokenization:** Breaking input into chunks (ex: start tags, end tags, attribute names, attribute values), stripping irrelevant characters such as whitespace and line breaks.
- **Lexing:** Like the tokenizer, but it also identifies the type of each token (this token is a number, that token is a string literal, this other token is an equality operator).
- **Parsing:** Takes the stream of tokens from the lexer, interprets the tokens using a specific grammar, and turns it into an abstract syntax tree.

Example: DOM Representation

```
<p>
  Let's use:
  <span>Cascading</span>
  <span>Style</span>
  <span>Sheets</span>
</p>
```

In the DOM, the node corresponding to our `<p>` element is a parent. Its children are a text node and the three nodes corresponding to our `` elements. The SPAN nodes are also parents, with text nodes as their children:

```
P
├─ "Let's use:"
├─ SPAN
│  └─ "Cascading"
```

```
├ SPAN
├   └ "Style"
└ SPAN
    └ "Sheets"
```

Applying CSS to the DOM

```
span {
  border: 1px solid black;
  background-color: lime;
}
```

Once both tree structures are created, the rendering engine then attaches the data structures into what's called a render tree as part of the layout process. The render tree is a visual representation of the document which enable painting the contents of the page in their correct order.

Render tree construction follows the following order:

- Starting at the root of the DOM tree, traverse each visible node.
- Omit non visible nodes.
- For each visible node find the appropriate matching CSSOM rules and apply them.
- Emit visible nodes with content and their computed styles.
- Finally, output a render tree that contains both the content and style information of all visible content on the screen.