CSS
INTERVIEW QUESTION

PW SKILLS

Basics

Q1. What is the difference between “display:none” and “visibility:hidden”?

Answer:
1. Display: none:
When you set display:none on an element, it is completely removed from the document's render tree. This means that the element, and its child elements, are not rendered on the page at all, and no space is allocated for it. It's as if the element doesn't exist in the document flow. This property is useful when you want to completely hide an element and reclaim the space when it is needed.

```
<div>
  <p>This paragraph is visible.</p>
  <p style="display:none">This paragraph is hidden and takes no
space.</p>
</div>
```

Output:

This paragraph is visible.

2. visibility:hidden
When you set visibility:hidden on an element, the element is still rendered in the document's render tree and takes up space, but it's not visible on the page. The element, and its child elements, are invisible, but the space they would have占ied remains allocated. This property is useful when you want to hide an element temporarily but maintain its space in the document flow.

```
<div>
  <p>This paragraph is visible.</p>
  <p style="visibility:hidden">This paragraph is hidden but takes
up space.</p>
</div>
```

Output:

This paragraph is visible.

Key Differences:

- display:none removes the element from the document's render tree, while visibility:hidden keeps the element in the render tree but makes it invisible.
- display:none does not allocate any space for the element, while visibility:hidden maintains the space the element would have占ied.
- Elements with display:none do not respond to mouse events or keyboard events, while elements with visibility:hidden can still receive events but cannot be seen.
- display:none is better for performance because the browser doesn't have to render the element or its child elements.

Q2. What is a CSS framework? Name some CSS frameworks.

Answer:
A CSS framework is like a toolbox full of pre-built styles and components that web developers can use to create websites and web applications more quickly and consistently. Instead of building everything from scratch, a CSS framework provides a set of ready-to-use styles for common UI elements like buttons, menus, forms, and typography.
Similarly, CSS frameworks offer a collection of pre-designed styles and components that developers can use to build their websites faster while maintaining a consistent look and feel across different pages and elements.

Some popular CSS frameworks include:
- BootStrap
- Tailwind CSS
- Chakra UI
- Bulma
- Emotion

Q3. Explain any 3 ways to center a div.

Answer:
1. Using Margin: Auto

This method works for horizontally centering a div with a fixed width. By setting the left and right margins to auto, the browser will automatically calculate and distribute the remaining space equally on both sides, effectively centering the div.

```
<div class="centered">This div is horizontally centred.</div>
```

```
.centered {
  width: 300px; /* Set a fixed width */
  margin: 0 auto; /* 0 for top-bottom margin, auto for left-right
*/
}
```

Output:

2. Using Flexbox

Flexbox is a modern layout module in CSS that provides an efficient way to center elements both horizontally and vertically. By setting the display property of the parent element to flex, and using the justify-content and align-items properties, you can center the child div easily.

```
<div class="flex-container">
  <div class="flex-child">This div is centered both horizontally
and vertically.</div>
</div>
```

```
.flex-container {
  display: flex;
  justify-content: center; /* Horizontal centering */
  align-items: center; /* Vertical centering */
  height: 100vh; /* Set the height to fill the viewport */
}
```

Output:

3. Using Transform and Position: Absolute

This method involves positioning the div absolutely and using the transform property to translate it by 50% of its own dimensions. This approach works for both horizontal and vertical centering, and it doesn't require the div to have a fixed width or height.

```
<div class="outer">
  <div class="centered">This div is centered both horizontally and
vertically.</div>
</div>
```

```
.outer {
  position: relative; /* Required for positioning the child div */
}

.centered {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

In the above examples, the first method is useful when you need to center a div with a fixed width horizontally. The second method, using flexbox, is a modern and versatile approach that allows you to center elements both horizontally and vertically with ease. The third method, using transform and position: absolute, is particularly helpful when you don't know the exact dimensions of the div you're centering.

Q4. What is responsive design?

Answer:

Responsive design is an approach to web development that ensures websites and applications provide an optimal viewing and interaction experience across a wide range of devices and screen sizes. In other words, it's a way to make your website look good and function properly on different devices, from desktop computers to tablets and smartphones.

Responsive design allows websites to adjust their layout, content, and even functionality based on the device's screen size, resolution, and orientation. This ensures that users can easily access and interact with the website, regardless of whether they're using a large desktop monitor or a small mobile phone screen.

Q5. What are the ways that we can add CSS to our HTML file?

Answer:

There are three main ways to add CSS styles to an HTML file: inline styles, internal style sheets, and external style sheet.

1. Inline Styles

Inline styles are applied directly to an HTML element using the “Style” attribute. This method is useful for quickly testing styles or applying unique styles to a specific element. However, it's generally not recommended for larger projects as it can make the code harder to maintain and update.

```
<h1 style="color: blue; font-size: 24px;">Heading with Inline
Styles</h1>
```

Output:

2. Internal Style Sheet

Internal style sheets are defined within the <head> section of an HTML document using the <style> element. This method allows you to write CSS rules that apply to the entire page or specific elements within that page.

```
<!DOCTYPE html>
<html>
<head>
    <title>Internal Style Sheet</title>
    <style>
        h1 {
            color: green;
            font-size: 28px;
        }
        p {
            font-family: Arial, sans-serif;
            line-height: 1.5;
        }
    </style>
</head>
<body>
    <h1>Heading with Internal Style Sheet</h1>
    <p>This paragraph is styled using the internal style sheet.</
p>
</body>
</html>
```

Output:

3. External Style Sheet

An external style sheet is a separate CSS file that is linked to the HTML document using the <link> element in the <head> section. This method is preferred for larger projects as it separates the content (HTML) from the presentation (CSS), making the code more organized and maintainable.

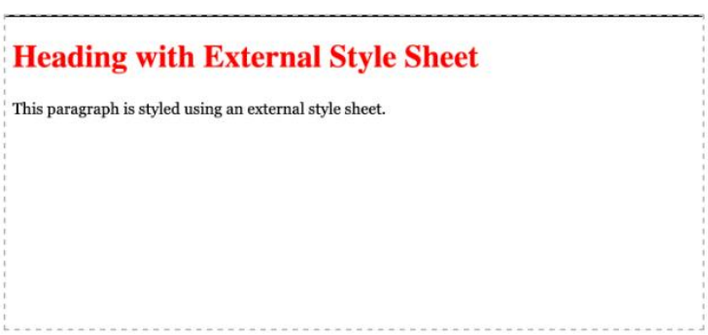
Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>External Style Sheet</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Heading with External Style Sheet</h1>
    <p>This paragraph is styled using an external style sheet.</p>
</body>
</html>
```

In your styles.css fi

```
h1 {
    color: red;
    font-size: 32px;
}
p {
    font-family: Georgia, serif;
    line-height: 1.6;
}
```

Output:



Q6. What is the purpose of media queries in CSS?

Answer:

Media queries are a powerful feature in CSS that allows you to apply different styles based on the characteristics of the device or viewport. They provide a way to create responsive designs that adapt to various screen sizes, resolutions, and orientations, ensuring an optimal viewing experience for users across different devices.

The primary purpose of media queries is to define conditional rules that apply styles based on specific media features, such as the width and height of the viewport, device orientation, resolution, and more. This is particularly useful for creating responsive websites and applications that adjust their layout, typography, and other visual elements to fit the user's device.

Here's an example of how media queries work:

```
/* Default styles for all devices */
body {
  font-size: 16px;
}

/* Styles for screens with a maximum width of 600px (e.g., mobile
devices) */
@media (max-width: 600px) {
  body {
    font-size: 14px;
  }

  .sidebar {
    display: none;
  }
}

/* Styles for screens with a minimum width of 900px (e.g., tablets
and desktops) */
@media (min-width: 900px) {
  body {
    font-size: 18px;
  }

  .main-content {
    width: 70%;
    float: left;
  }

  .sidebar {
    width: 30%;
    float: right;
  }
}
```

By using media queries, you can create responsive designs that adapt to different screen sizes and device capabilities, providing an optimal user experience across a wide range of devices.

Intermediate

Q7. Explain the different types of Selectors in CSS.

Answer:
In CSS, selectors are used to target and apply styles to specific HTML elements on a web page. There are several types of selectors available, each with its own purpose and syntax.

1. **Type Selectors:** Type selectors target elements based on their HTML tag name. They are the most basic selectors and are useful for applying styles to all instances of a particular element type.

```
h1 {
  color: navy;
  font-size: 28px;
}

p {
  line-height: 1.5;
}
```

2. **Class Selectors:** Class selectors target elements based on their class attribute value. They are useful for applying styles to specific groups of elements, regardless of their type.

```
<h1 class="heading">Welcome</h1>
<p class="intro">This is an introduction.</p>
```

```
.heading {
  text-align: center;
}

.intro {
  font-style: italic;
}
```

3. **ID Selectors:** ID selectors target elements based on their unique ID attribute value. Since IDs should be unique within a document, these selectors are useful for applying styles to a single, specific element.

```
<div id="main-content"> <!— content goes here —> </div>
```

```
#main-content {
  max-width: 800px;
  margin: 0 auto;
}
```

4. **Universal Selector:** The universal selector (*) selects all elements on the page. It is typically used as part of a more specific selector or for resetting default styles.

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;}
```

5. **Attribute Selectors:** Attribute selectors target elements based on their attribute values. They are useful for applying styles to elements that match specific attribute conditions.

```
<a href="https://example.com" target="_blank">Link</a>
<input type="text" disabled>
```

```
a[target="_blank"] {
  font-weight: bold;
}

input[type="text"][disabled] {
  background-color: #f0f0f0;
}
```

6. **Combinators:** Combinators are used to combine multiple selectors to target specific elements based on their relationship with other elements in the document tree.

- Descendant Combinator (space): Targets elements that are descendants of the preceding element.
- Child Combinator (>): Targets direct child elements of the preceding element.
- Adjacent Sibling Combinator (+): Targets elements that are immediately preceded by the specified element.
- General Sibling Combinator (~): Targets elements that are preceded by the specified element, regardless of their position.

```
<div>
  <h2>Heading</h2>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <span>Span element</span>
</div>
```

```
div h2 {
  color: green; /* Descendant Combinator */
}

div > p {
  font-weight: bold; /* Child Combinator */
}
```

```
p + span {
  text-decoration: underline; /* Adjacent Sibling Combinator */
}

p ~ span {
  font-style: italic; /* General Sibling Combinator */
}
```

Q8. What are Pseudo elements and Pseudo classes?

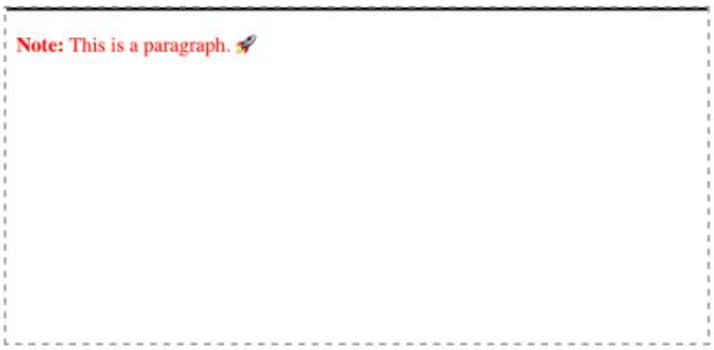
Answer:
Pseudo-elements and pseudo-classes are powerful features in CSS that allow you to target and style specific parts or states of HTML elements. They are essential tools for creating complex and interactive designs. Let's explore each of them:

Pseudo-elements:
Pseudo-elements are used to style specific parts of an element that are not directly accessible through HTML. They create a virtual element that can be targeted and styled using CSS. There are several pseudo-elements available, such as ::before, ::after, ::first-line, and ::selection.

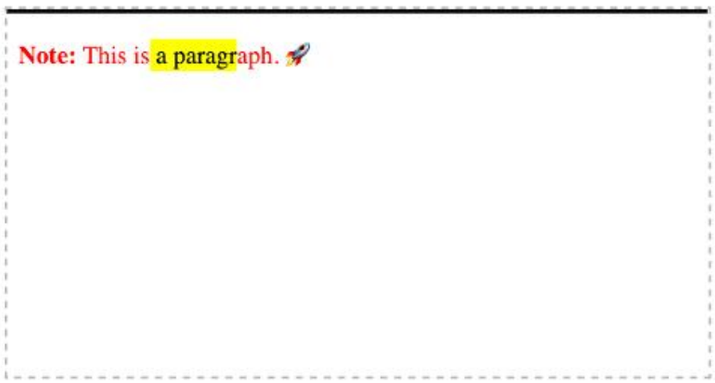
```
<p>This is a paragraph.</p>
```

```
p::before {
  content: "Note: ";
  font-weight: bold;
}

p::after {
  content: " 🚀";
}

p::first-line {
  color: red;
}

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

Output:



when selecting the text:



In this example, the::before and ::after pseudo-elements add content before and after the paragraph text. The ::first-line pseudo-element styles the first line of the paragraph, and the ::selection pseudo-element styles the text that the user selects.

Pseudo-classes:
Pseudo-classes are used to target and style elements based on their state or position within the document. They allow you to apply styles to elements when they meet certain conditions, such as being hovered over, visited, or the first child of a parent element. Some common pseudo-classes include :hover, :active, :visited, :first-child, and :nth-child.

```
<a href="#">Link</a>
<ul>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ul>
```

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

a:visited {
  color: purple;
}

li:first-child {
  font-weight: bold;
}

li:nth-child(even) {
  background-color: #f0f0f0;
}
```

Output:

In this example, the :hover pseudo-class styles the link when the user hovers over it, and the :visited pseudo-class styles visited links. The :first-child pseudo-class applies bold styling to the first list item, and the :nth-child(even) pseudo-class applies a background color to every even-numbered list item.

1. **Grouping Selectors:** You can group multiple selectors together, separated by commas, and apply the same styles to all of them.

Example :

```
h1, h2, h3 {
  color: navy;
  font-family: Arial, sans-serif;
}
```

2. **Multiple Class Selectors:** If the elements share a common class, you can apply styles to that class selector.

Example :

```
<h1 class="styled-element">Heading</h1>
<p class="styled-element">Paragraph</p>
```

```
.styled-element {
  color: green;
  font-weight: bold;
}
```

3. **Multiple ID Selectors:** While not recommended due to the uniqueness constraint of IDs, you can target multiple elements with the same ID using the universal selector (*).

Example :

```
<h1 id="styled-element">Heading</h1>
<p id="styled-element">Paragraph</p>
```

```
*#styled-element {
  text-decoration: underline;
}
```

Q10. What does "!important" mean in CSS?

Answer:
The !important declaration in CSS is used to increase the specificity and importance of a particular CSS rule. It overrides other conflicting rules that have lower specificity, ensuring that the styles declared with !important take precedence.

Q11. What is the use of "z-index"? Explain its function.

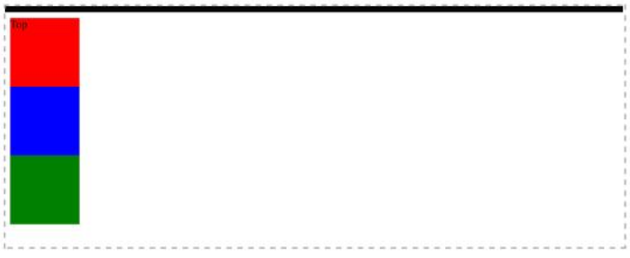
Answer:
The z-index property in CSS is used to control the stacking order of positioned elements on a web page. It determines which element should appear on top or below others when they overlap.

Imagine you have a stack of paper on a table. The z-index property allows you to rearrange the order of those papers, bringing some to the front and pushing others to the back.

Elements with a higher z-index value will appear on top of those with lower values. Elements with the same z-index value are stacked based on their order in the HTML source.

Here's a basic example:

```
<div style="position:relative; z-index:1;">
  <div style="position:absolute; z-index:3; background:red;
height: 100px">Top</div>
  <div style="position:absolute; z-index:2; background:blue;
height: 200px">Middle</div>
  <div style="position:absolute; z-index:1; background:green;
height: 300px">Bottom</div>
</div>
```



In this example, the red div will appear on top, followed by the blue div, and then the green div at the bottom of the stacking order.

z-index only works on positioned elements (position: relative, position: absolute, or position: fixed). Non-positioned elements are not affected by z-index.

Q12. Mention some of the media features and scenarios where media query might be used.

Answer:
Media queries in CSS allow you to apply different styles based on various media features and conditions, enabling responsive web design. Here are some common media features and scenarios where media queries might be used:

- **Screen Width and Height:** Adjust layout, font sizes, and element visibility based on the viewport width or height. For example, switching to a single-column layout on smaller screens.
- **Device Orientation:** Adapt the design based on whether the device is in portrait or landscape mode, such as adjusting the position of navigation menus or rearranging content.
- **Resolution:** Serve optimized images or adjust typography based on the device's resolution to ensure crisp visuals and legible text.
- **Aspect Ratio:** Apply specific styles based on the aspect ratio of the device, which can be useful for adapting layouts to different screen shapes and sizes.
- **Pointer:** Detect the presence of a pointer device (e.g., mouse, touchscreen) and adjust styles accordingly, such as increasing touch target sizes for better usability on touch devices.
- **Hover Capability:** Apply hover effects or tooltips only on devices that support hover interactions, such as desktop computers with a mouse.
- **Scripting Support:** Conditionally load or unload certain styles or functionality based on whether the device supports scripting (e.g., JavaScript).

Advanced

Q13. Explain the concept of CSS specificity.

Answer:
CSS specificity is a way to determine which CSS styles will be applied to an element when there are conflicting rules. It helps the browser decide which rule takes precedence over others. This concept is essential for ensuring that your styles are applied correctly and consistently across your web pages.

CSS specificity follows a set of rules, with each rule having a specificity value. The higher the specificity value, the more weight that rule carries, and the more likely it is to be applied. The specificity rules are as follows:

Inline styles: Styles applied directly to an element using the style attribute have the highest specificity.

1. **ID selectors:** Selectors that target an element's unique ID (e.g., #my-element) have a high specificity.
2. **Class selectors, attribute selectors, and pseudo-classes:** Selectors that target an element's class (e.g., .my-class), attributes (e.g., [type="text"]), or pseudo-classes (e.g., :hover) have a lower specificity than IDs.
3. **Element selectors and pseudo-elements:** Selectors that target an element's tag name (e.g., p, div) or use pseudo-elements (e.g., ::before, ::after) have the lowest specificity.

When multiple rules have the same specificity, the last rule defined in the CSS file takes precedence. Additionally, the !important declaration can be used to override all other specificity rules, but this should be used sparingly as it can make your CSS harder to maintain.

Q14. Explain the CSS position property.

Answer:
The CSS position property is used to specify the positioning method for an element. It determines how the element is positioned relative to its normal position, the viewport, or another positioned element. The most commonly used values for the position property are:

- **static (default):** The element is positioned according to the normal flow of the document.
- **relative:** The element is positioned relative to its normal position. You can use the top, right, bottom, and left properties to offset it from its normal position.

```
.box {
  position: relative;
  top: 20px;
  left: 30px;
}
```

- **absolute:** The element is positioned relative to its nearest positioned ancestor (an element with position: relative, absolute, fixed, or sticky). If there is no positioned ancestor, it is positioned relative to the document body.

```
.container {
  position: relative;
}

.box {
  position: absolute;
  top: 50px;
  right: 30px;
}
```

- **fixed:** The element is positioned relative to the viewport and will stay in the same place even if the page is scrolled.

```
.navbar {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
}
```

- **sticky:** The element is positioned based on the user's scroll position. It behaves like a relatively positioned element until a certain scroll position is met, then it behaves like a fixed element.

```
.sidebar {
  position: sticky;
  top: 20px;
}
```

Q15. Briefly explain the attribute selectors.

Answer:
Attribute selectors in CSS allow you to target HTML elements based on their attributes or attribute values. They provide a powerful way to style elements based on specific attribute characteristics.

1. **[attr]:** Selects elements that have the specified attribute, regardless of its value.
2. **[attr=value]:** Selects elements that have the specified attribute with the exact given value.
3. **[attr~=value]:** Selects elements that have the specified attribute with a value that is a whitespace-separated list and contains the given value.
4. **[attr|=value]:** Selects elements that have the specified attribute with a value that either matches the given value or starts with the given value followed by a hyphen (-).
5. **[attr^=value]:** Selects elements that have the specified attribute with a value that starts with the given value.
6. **[attr$=value]:** Selects elements that have the specified attribute with a value that ends with the given value.
7. ***[attr=value]**:** Selects elements that have the specified attribute with a value that contains the given substring.

Attribute selectors provide a concise way to target elements based on their attributes, enabling precise styling control for specific elements or groups of elements.

Q16. Which is the latest version of CSS? What are the features introduced?

Answer:
The latest version of CSS is CSS3, which was a major revision of the CSS language. CSS3 was developed in modules, with each module introducing new features and improvements. Here are some of the key features introduced in CSS3:

1. Selectors
2. Box Model
3. Text Effects like text-shadow, word-wrap
4. 2D/3D Transformations
5. Transitions and Animations
6. Flexible Box Layout (Flexbox)
7. Grid Layout
8. Media Queries
9. Multiple Column Layout

Q17. What is a CSS Preprocessor? Why do people use them?

Answer:
A CSS preprocessor is a scripting language that extends the functionality of regular CSS by introducing additional features and capabilities. These preprocessors are designed to make CSS more maintainable, modular, and easier to work with, especially in larger projects.
People use CSS preprocessors for several reasons:

1. **Variables:** Preprocessors allow you to define and use variables for things like colours, font sizes, and other values. This makes it easier to maintain consistency across your stylesheets and update values in one place.
2. **Nesting:** Preprocessors support the nesting of selectors, which helps to organise and structure your CSS in a more logical way, reflecting the HTML hierarchy.
3. **Mixins:** Mixins are reusable blocks of CSS that can be included in other rules, promoting code reuse and preventing duplication.
4. **Functions and Operations:** Preprocessors provide built-in functions and operations for performing calculations, manipulating colors, and other tasks, that would otherwise require complex manual calculations in plain CSS.
5. **Code Organization:** Preprocessors encourage modular code organisation by allowing you to split your stylesheets into multiple files, which can be imported and combined during the compilation process.
6. **Cross-browser Compatibility:** Some preprocessors offer built-in support for automatically adding vendor prefixes, ensuring cross-browser compatibility without manually writing prefixed rules.

Popular CSS preprocessors include Sass, Less, and Stylus. While they introduce a new syntax and require a compilation step, preprocessors can significantly improve the maintainability, scalability, and organization of your CSS codebase, especially for large-scale projects.

Q18. What is the difference between CSS grid and flexbox?

Answer:
Flexbox is for one-dimensional layouts along a single axis (row or column). It's great for component-level alignment and distributing space.
CSS Grid is for two-dimensional layouts with rows and columns. It excels at complex, grid-based page layouts with precise control over positioning items.
Flexbox is ideal for component layouts, while Grid is better suited for overall page structure layouts. They can be combined, but their core purposes differ significantly.

Q19. What are some potential challenges or considerations when designing layouts with floats?

Answer:
When designing layouts with floats, there are several potential challenges and considerations to keep in mind:

- **Collapsed Parent Element:** When an element is floated, it is taken out of the normal document flow, which can cause the parent element to collapse and lose its height. This issue requires clearing the floats or using techniques like the clearfix hack.
- **Double Margin Bug:** In some cases, when two adjacent elements are floated, the margins between them can double up, causing unwanted spacing. This bug can be resolved by setting the display property of the parent element to inline-block or using negative margins.
- **Source Order Issues:** Floated elements can disrupt the natural source order of elements on the page, which can cause accessibility and usability problems, especially for screen readers and keyboard navigation.
- **Layout Shifts:** Adding or removing floated elements can cause layout shifts and reflows, which can negatively impact the user experience, especially on slower devices or connections.
- **Vertical Centering:** Vertically centering floated elements can be challenging and may require additional techniques or workarounds.
- **Responsive Design:** Floats can make it difficult to create responsive layouts that adapt well to different screen sizes and orientations, often requiring additional media queries or techniques like clear fix.

Q20. What is the mobile-first approach in responsive design, and how does it relate to media queries?

Answer:
The mobile-first approach in responsive design involves designing and developing websites with the mobile experience in mind first, and then progressively enhancing the layout for larger screens using media queries. Media queries allow you to apply different CSS styles based on device characteristics like screen size, orientation, or resolution. By starting with a mobile-first approach and using media queries to adapt the design for larger screens, you ensure an optimized experience across all devices.

Q21. What are viewport units, and how can they be used in conjunction with media queries to create responsive layouts?

Answer:
Viewport units in CSS (vw, vh, vmin, and vmax) are relative units that represent a percentage of the browser's viewport dimensions. They can be used in conjunction with media queries to create responsive layouts that adapt to different screen sizes and orientations.
For example, setting an element's width to 50vw (50% of the viewport width) ensures it will scale proportionally regardless of the viewport size. Similarly, vh (viewport height) can be used for vertical dimensions.
By combining viewport units with media queries that target specific viewport ranges, you can apply styles tailored to different device sizes and orientations, ensuring an optimised layout and user experience across devices.

Here's an example:

```
/* Mobile styles */
.container {
  width: 90vw;
}

/* Tablet styles */
@media (min-width: 768px) {
  .container {
    width: 70vw;
  }
}

/* Desktop styles */
@media (min-width: 1024px) {
  .container {
    width: 50vw;
  }
}
```

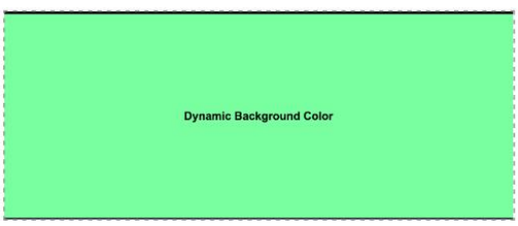
In this way, viewport units and media queries work together to create responsive, fluid layouts that adapt seamlessly to different viewports.

Q22. Create a basic web page that dynamically changes its background color based on the type of screen using media queries. The background color should vary for mobile, tablet, and desktop screens.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Dynamic Background Color</title>
  <style>
```

```
    body {
      margin: 0;
      padding: 0;
      height: 100vh;
      display: flex;
      justify-content: center;
      align-items: center;
      font-family: Arial, sans-serif;
    }
/* Default background color */
    body {
      background-color: #ffa07a;
    }
    /* Media queries */
    @media (min-width: 768px) {
      /* Tablet screens */
      body {
        background-color: #ffd07a;
      }
    }
    @media (min-width: 1024px) {
      /* Desktop screens */
      body {
        background-color: #7affa0;
      }
    }
  </style>
</head>
<body>
  <h1>Dynamic Background Color</h1>
</body>
</html>
```

Output:
Desktop view:



Tablet View:

Default View: