CSS Fundamentals Review

CSS Basics

- **What is CSS?**: Cascading Style Sheets (CSS) is a markup language used to apply styles to HTML elements. CSS is used for colors, background images, layouts and more.
- **Basic Anatomy of a CSS Rule**: A CSS rule is made up of two main parts: a selector and a declaration block. A selector is a pattern used in CSS to identify and target specific HTML elements for styling. A declaration block applies a set of styles for a given selector or selectors.

Here is the general syntax of a CSS rule:

```css
selector {
    property: value;
}
```

- **`meta name="viewport"` Element**: This `meta` element gives the browser instructions on how to control the page's dimensions and scaling on different devices, particularly on mobile phones and tablets.
- **Default Browser Styles**: Each HTML element will have default browser styles applied to them. This usually includes items like default margins and paddings.

Inline, Internal, and External CSS

- **Inline CSS**: These styles are written directly within an HTML element using the `style` attribute. Most of the time you will not be using inline CSS due to separation of concerns.

Here is an example of inline CSS:

```html
<p style="color: red;">This is a red paragraph.</p>
```

- **Internal CSS**: These styles are written within the `<style>` tags inside the `head` section of an HTML document. This can be useful for creating short code examples, but usually you will not need be using internal CSS.

- **External CSS**: These styles are written in a separate CSS file and linked to the HTML document using the `link` element in the `head` section. For most projects, you will use an external CSS file over internal or inline CSS.

Working With the `width` and `height` Properties

- **`width` Property**: This property specifies the width of an element. If you do not specify a width, then the default is set to `auto`. This lets the browser determine the element's width based on its content, parent, and display type.
- **`min-width` Property**: This property specifies the minimum width for an element.
- **`max-width` Property**: This property specifies the maximum width for an element.
- **`height` Property**: This property specifies the height of an element. Similarly, the height is `auto` by default, which means it will adjust to the content inside.
- **`min-height` Property**: This property specifies the minimum height for an element.
- **`max-height` Property**: This property specifies the maximum height for an element.

Different Types of CSS Combinators

- **Descendant Combinator**: This combinator is used to target elements that are descendants of a specified parent element. The following example will target all `li` items inside `ul` elements.

```
<ul>
    <li>Example item one</li>
    <li>Example item two</li>
    <li>Example item three</li>
</ul>
ul li {
    background-color: yellow;
}
```

- **Child Combinator (`>`)**: This combinator is used to select elements that are direct children of a specified parent element. The following example will target all `p` elements that are direct children of the `container` class.

```
<div class="container">
  <p>This will get styled.</p>

  <div>
    <p>This will not get styled.</p>
  </div>
</div>
.container > p {
  background-color: black;
  color: white;
}
```

- **Next-sibling Combinator (`+`)**: This combinator selects an element that immediately follows a specified sibling element. The following example will select the paragraph element that immediately follows the `h2` element.

```
<h2>I am a sub heading</h2>

<p>This paragraph element will get a red background.</p>
h2 + p {
  background-color: red;
}
```

- **Subsequent-sibling Combinator (`~`)**: This combinator selects all siblings of a specified element that come after it. The following example will style only the second paragraph element because it is the only one that is a sibling of the `ul` element and shares the same parent.

```
<div class="container">
  <p>This will not get styled.</p>
  <ul>
    <li>Example item one</li>
    <li>Example item two</li>
    <li>Example item three</li>
  </ul>
  <p>This will get styled.</p>
</div>
<p>This will not get styled.</p>
ul ~ p {
  background-color: green;
}
```

Inline, Block, and Inline-Block Level Elements

- **Inline Level Elements**: Inline elements only take up as much width as they need and do not start on a new line. These elements flow within the content, allowing text and other inline elements to appear alongside them. Common inline elements are `span`, `anchor`, and `img` elements.
- **Block Level Elements**: Block-level elements start on a new line and take up the full width available to them by default, stretching across the width of their container. Some common block-level elements are `div`, `paragraph`, and `section` elements.
- **Inline-Block Level Elements**: You can set an element to `inline-block` by using the `display` property. These elements behave like inline elements but can have a `width` and `height` set like block-level elements.

Margin and Padding

- **`margin` Property**: This property is used to apply space outside the element, between the element's border and the surrounding elements.
- **`padding` Property**: This property is used to apply space inside the element, between the content and its border.
- **`margin` Shorthand**: You can specify 1–4 values to set the margin sides. One value applies to all four sides; two values set `top` and `bottom`,

then `right` and `left`; three values set `top`, horizontal (`right` and `left`), then `bottom`; four values set `top`, `right`, `bottom`, `left`.

- **`padding` Shorthand**: You can specify 1–4 values to set the padding sides. One value applies to all four sides; two values set `top` and `bottom`, then `right` and `left`; three values set `top`, horizontal (`right` and `left`), then `bottom`; four values set `top`, `right`, `bottom`, `left`.

CSS Specificity

- **Inline CSS Specificity**: Inline CSS has the highest specificity because it is applied directly to the element. It overrides any internal or external CSS. The specificity value for inline styles is (1, 0, 0, 0).
- **Internal CSS Specificity**: Internal CSS is defined within a `style` element in the `head` section of the HTML document. It has lower specificity than inline styles but can override external styles.
- **External CSS Specificity**: External CSS is linked via a `link` element in the `head` section and is written in separate `.css` files. It has the lowest specificity but provides the best maintainability for larger projects.
- **Universal Selector (`*`)**: a special type of CSS selector that matches any element in the document. It is often used to apply a style to all elements on the page, which can be useful for resetting or normalizing styles across different browsers. The universal selector has the lowest specificity value of any selector. It contributes 0 to all parts of the specificity value (0, 0, 0, 0).
- **Type Selectors**: These selectors target elements based on their tag name. Type selectors have a relatively low specificity compared to other selectors. The specificity value for a type selector is (0, 0, 0, 1).
- **Class Selectors**: These selectors are defined by a period (`.`) followed by the class name. The specificity value for a class selector is (0, 0, 1, 0). This means that class selectors can override type selectors, but they can be overridden by ID selectors and inline styles.
- **ID Selectors**: ID selectors are defined by a hash symbol (`#`) followed by the ID name. ID selectors have a very high specificity, higher than type selectors and class selectors, but lower than inline styles. The specificity value for an ID selector is (0, 1, 0, 0).
- **`!important` keyword**: used to give a style rule the highest priority, allowing it to override any other declarations for a property. When used, it forces the browser to apply the specified style, regardless of the specificity of other selectors. You should be cautious when using `!important` because it can make your CSS harder to maintain and debug.

- **Cascade Algorithm**: An algorithm used to decide which CSS rules to apply when there are multiple styles targeting the same element. It ensures that the most appropriate styles are used, based on a set of well-defined rules.
- **CSS Inheritance**: The process by which styles are passed down from parent elements to their children. Inheritance allows you to define styles at a higher level in the document tree and have them apply to multiple elements without explicitly specifying them for each element.

Lists, Links, CSS Background and Borders Review

Styling Lists

- **`line-height` Property**: This property is used to create space between lines of text. The accepted `line-height` values include the keyword `normal`, numbers, percentages and length units like the `em` unit.
- **`list-style-type` Property**: This property is used to specify the marker for a list item. Acceptable values can include a circle, disc, or decimal.
- **`list-style-position` Property**: This property is used to set the position for the list marker. The only two acceptable values are inside and outside.
- **`list-style-image` Property**: This property is used to use an image for the list item marker. A common use case is to use the `url` function with a value set to a valid image location.

Spacing list items using `margin`

- Apart from `line-height`, margins can also be used in CSS to enhance the spacing and readability of list items.
- Margins create space outside each `li` element, allowing control over the gap between list items.

- `margin-bottom` is used to create space below each list item. For example, `margin-bottom: 10px;` will create a 10-pixel gap below each list item.

Styling Links

- **`pseudo-class`**: This is a keyword added to a selector that allows you to select elements based on a particular state. Common states would include the `:hover`, `:visited` and `:focus` states.
- **`:link pseudo-class`**: This pseudo-class is used to style links that have not be visited by the user.
- **`:visited pseudo-class`**: This pseudo-class is used to style links where a user has already visited.
- **`:hover pseudo-class`**: This pseudo-class is used to style an elements where a user is actively hovering over them.
- **`:focus pseudo-class`**: This pseudo-class is used to style an element when it receives focus. Examples would include `input` or `select` elements where the clicks or tabs on the element to focus it.
- **`:active pseudo-class`**: This pseudo-class is used to style an element that was activated by the user. A common example would be when the user clicks on a button.

Working with Backgrounds and Borders

- **`background-size` Property**: This property is used to set the background size for an element. Some common values include `cover` for the background image to cover the entire element and `contain` for the background image to fit within the element.
- **`background-repeat` Property**: This property is used to determine how background images should be repeated along the horizontal and vertical axes. The default value for `background-repeat` is `repeat` meaning the image will repeat both horizontally and vertically. You can also specify that there should be no repeat by using the `no-repeat` property.
- **`background-position` Property**: This property is used to specify the position of the background image. It can be set to a specific length, percentage, or keyword values like `top`, `bottom`, `left`, `right`, and `center`.
- **`background-attachment` Property**: This property is used to specify whether the background image should scroll with the content or remain fixed in place.

The main values are `scroll` (default), where the background image scrolls with the content, and `fixed`, where the background image stays in the same position on the screen.

- **`background-image` Property**: This property is used to set the background image of an element. You can set multiple background images at the same time and use either the `url`, `radial-gradient` or `linear-gradient` functions as values.
- **`background` Property**: This is the shorthand property for setting all background properties in one declaration. Here is an example of setting the background image and setting it to not repeat: `background: no-repeat url("example-url-goes-here");`
- **Good Contrast for Background and Foreground Colors**: It is important to ensure that the background and foreground colors have good contrast to make the text readable. The Web Content Accessibility Guidelines (WCAG) recommend a minimum contrast ratio of 4.5:1 for normal text and 3:1 for large text.

Borders

- **`border-top` Property**: This property is used to set the styles for the top border of an element. `border-top: 3px solid blue;` sets a 3-pixel-wide solid blue border on the top side of the element.
- **`border-right` Property**: This property is used to set the styles for the right border of an element. `border-right: 2px solid red;` sets a 2-pixel-wide solid red border on the right side of the element.
- **`border-bottom` Property**: This property is used to set the styles for the bottom border of an element. `border-bottom: 1px dashed green;` sets a 1-pixel-wide dashed green border on the bottom side of the element.
- **`border-left` Property**: This property is used to set the styles for the left border of an element. `border-left: 4px dotted orange;` sets a 4-pixel-wide dotted orange border on the left side of the element.
- **`border` Property**: This is the shorthand property for setting the width, style, and color of an element's border. `border: 1px solid black;` sets a 1-pixel-wide solid black border.
- **`border-radius` Property**: This property is used to create rounded corners for an element's border.
- **`border-style` Property**: This property is used to set the style of an element's border. Some accepted values include `solid`, `dashed`, `dotted`, and `double`.

Gradients

- `linear-gradient()` **Function**: This CSS function is used to create a transition between multiple colors along a straight line.
- `radial-gradient()` **Function**: This CSS function creates an image that radiates from a particular point, like a circle or an ellipse, and gradually transitions between multiple colors.

Design Fundamentals Review

Design Terminology

- **Layout**: This is how visual elements are arranged on a page or screen to communicate a message. These elements may include text, images, and white space.
- **Alignment**: This is how the elements are placed in relation to one another. Using alignment correctly is helpful for making the design look clean and organized.
- **Composition**: This is the act of arranging elements to create a harmonious design. It determines how elements like images, text, and shapes relate to each other and contribute to the design in an artistic way.
- **Balance**: This is how visual weight is distributed within a composition. Designers aim to create an equilibrium through symmetrical or asymmetrical arrangements.
- **Scale**: This refers to comparing the dimensions or size of one element to that of another.
- **Hierarchy**: This establishes the order of importance of the elements in a design. It's about making sure that the most important information is noticed first.
- **Contrast**: This is the process of creating clear distinctions between the elements. You can do this through variations in color, size, shape, texture, or any other visual characteristic. Strong contrast is also helpful for improving readability.
- **White Space(negative space)**: This is the empty space in a design. It's the area surrounding the elements.
- **UI(User Interface)**: UI includes the visual and interactive elements that users can see on their screens, like icons, images, text, menus, links, and buttons.
- **UX(User Experience)**: UX is about how users feel when using a product or service. An application with a well-design user experience is intuitive, easy to use, efficient, accessible, and enjoyable.
- **Design Brief**: This is a document that outlines the objectives, goals, and requirements of a project. It is a roadmap that guides the design process and ensures that the final product meets the needs of the client.
- **Vector Based Design**: This involves creating digital art using mathematical formulas to define lines, shapes, and colors.
- **Prototyping**: This refers to the process of creating an interactive model of a product or user interface.

UI Design Fundamentals

- **Good Contrast for Background and Foreground Colors**: It is important to ensure that the background and foreground colors have good contrast to make the text readable. The Web Content Accessibility Guidelines (WCAG) recommend a minimum contrast ratio of 4.5:1 for normal text and 3:1 for large text.
- **Good Visual Hierarchy in Design**: A strong visual hierarchy can provide a clear path for the eye to follow, ensuring that the information you convey is consumed in the order that you intend.
- **Responsive Images**: Responsive images are images that scale to fit the size of the screen they are being viewed on. This is important because it ensures that your images look good on all devices, from desktops to mobile phones.
- **Progressive Enhancement**: This is a design approach that ensures all users, regardless of browser or device, can access the essential content and functionality of an application.
- **User-centered Design**: This is an approach that prioritizes the end user, from their needs to their preferences and limitations. The goal of user-centered design is to craft a web page that is intuitive, efficient to use, and pleasing for your users to interact with.
- **User Research**: This is the systematic study of the people who use your product. The goal is to measure user needs, behaviors, and pain points.
- **Exit Interviews**: This is a survey you can give to users when they cancel their accounts. It can help you understand why users are leaving and what you can do to reduce churn.
- **User Testing**: This refers to the practice of capturing data from users as they interface with your application.
- **A/B Testing**: This is the process of shipping a new feature to a randomly selected subset of your user base. You can then leverage analytics data to determine if the feature is beneficial.
- **User Requirements**: This refers to the stories or rubric that your application needs to follow. User requirements might be defined by user research or industry standards. They can even be defined by stakeholder input.
- **Progressive Disclosure**: This is a design pattern used to only show users relevant content based on their current activity and hide the rest. This is done to reduce cognitive load and make the user experience more intuitive.
- **Deferred/Lazy Registration**: This is a UI design pattern that allows users to browse and interact with your application without having to register.

Design Best Practices

- **Dark Mode**: This is a special feature on web applications where you can change the default light color scheme to a dark color scheme. You should use desaturated colors in dark mode. Desaturated colors are colors that are less intense and have a lower saturation level.
- **Breadcrumbs**: This is a navigation aid that shows the user where they are in the site's hierarchy. It is best to place breadcrumbs at the top of the page so users can easily find it. Also, you want to make sure the breadcrumbs are large enough to be easily read, but not so large that they take up too much space on the page.
- **Card Component**: Your card component should be simple in design, not visually cluttered or display too much information. For media, make sure to choose high-quality images and videos to enhance the user experience.
- **Infinite Scroll**: This is a design pattern that loads more content as the user scrolls down the page. You should consider using a load more button because it gives a user control over when they want to see more content. You can also add a back button so users have the ability to go back to the previous page without having to scroll all the way back up.
- **Modal Dialog**: This is a type of pop-up that will display on top of existing page content. Usually the background content will have a dim color overlay in order to help the user better focus on the modal content. Also, it is always a good idea to allow the user to click outside of the modal to close it. When you use the HTML `dialog` element, you will get a lot of the functionality and accessibility benefits built in.
- **Progress Indication for Form Registration**: This is a way to show users how far they are in a process. It can be used in forms, registration, and setup processes. Your design should be simple, easy to find, and make it possible to go back to previous steps.
- **Shopping Cart**: Carts are a place for user to see what item they have already selected on an e-commerce platform. Your carts should always be visible to the user, use a common icon like a cart, bag or basket, and have a clear call-to-action button for users to proceed to checkout.

Common Design Tools

- **Figma**: This cloud-based tool specializes in User Interface and User Experience (UI / UX) design. It enables design and development teams to collaborate from anywhere, offering built-in features including Vector-based design, automatic layout, a commenting and feedback system and more.
- **Sketch**: This is a popular design tool used for its intuitive interface and simplicity, making it ideal for developers who want to quickly create

prototypes. It's also widely used by designers for tasks like creating UIs, icons, and web layouts.

- **Adobe XD**: This is a vector-based design and prototyping tool for UI/UX design, known for its seamless integration with other Adobe apps like Photoshop, Illustrator, and After Effects.
- **Canva**: This tool allows you to create a wide range of visual content, including posters, cover photos, presentations, short videos, and more. Its user-friendly and simple design makes it ideal for beginners.

CSS Relative and Absolute Units Review

Absolute Units

- **`px` (Pixels)**: This absolute unit is a fixed-size unit of measurement in CSS. It is the most common absolute unit and provides precise control over dimensions. `1px` is always equal to 1/96th of an inch.
- **`in` (Inch)**: This absolute unit is equal to 96px.
- **`cm` (Centimeters)**: This absolute unit is equal to 25.2/64 of an inch.
- **`mm` (Millimeters)**: This absolute unit is equal to 1/10th of a centimeter.
- **`q` (Quarter-Millimeters)**: This absolute unit is equal to 1/40th of a centimeter.
- **`pc` (Picas)**: This absolute unit is equal to 1/6th of an inch.
- **`pt` (Points)**: This absolute unit is equal to 1/72th of an inch.

Relative Units

- **Percentages**: These relative units allow you to define sizes, dimensions, and other properties as a proportion of their parent element. For example, if you set `width: 50%;` on an element, it will occupy half the width of its parent container.
- **`em` Unit**: These units are relative to the font size of the element. If you are using `ems` for the `font-size` property, the size of the text will be relative to the font size of the parent element.
- **`rem` Unit**: These units are relative to the font size of the root element, which is the `html` element.
- **`vh` Unit**: `vh` stands for `"viewport height"` and `1vh` is equal to 1% of the viewport's height.
- **`vw` Unit**: `vw` stands for `"viewport width"` and `1vw` is equal to 1% of the viewport's width.

`calc` Function

- **`calc()` Function**: With the `calc()` function, you can perform calculations directly within your stylesheets to determine property values dynamically. This means that you can create flexible and responsive user interfaces by calculating dimensions based on the viewport size or other elements.

Best Practices (VERY IMPORTANT)

1. Add spaces around + and - operators

☐ Wrong
```
calc(100% -30px)
```

✔ Correct
```
calc(100% - 30px)
```

2. Zero (0) must include a unit

☐ `calc(100% - 0)`
✔ `calc(100% - 0px)`

3. For * and /, one side must be unitless

☐ `calc(5px * 50px)`
✔ `calc(5 * 50px)`

☐ `calc(50% / 5%)`
✔ `calc(50% / 5)`

4. You can nest calc()
```
width: calc(50% - calc(2rem + 10px));
```

CSS Pseudo-classes Review

User Action Pseudo-classes

- **Pseudo-classes Definition**: These are special CSS keywords that allow you to select an element based on its specific state or position.
- **User Action Pseudo-classes**: These are special keywords that allow you to change the appearance of elements based on user interactions, improving the overall user experience.
- **`:active` Pseudo-class**: This pseudo-class lets you select the active state of an element, like clicking on a button.
- **`:hover` Pseudo-class**: This pseudo-class defines the hover state of an element.
- **`:focus` Pseudo-class**: This pseudo-class applies styles when an element gains focus, typically through keyboard navigation or when a user clicks into a form input.
- **`:focus-within` Pseudo-class**: This pseudo-class is used to apply styles to an element when it or any of its descendants have focus.

Input Pseudo-classes

- **Input Pseudo-classes**: These pseudo-classes are used to target HTML `input` elements based on the state they are in before and after user interaction.
- **`:enabled` Pseudo-class**: This pseudo-class is used to target form buttons or other elements that are currently enabled.
- **`:disabled` Pseudo-class**: This pseudo-class lets you style an interactive element in disabled mode.
- **`:checked` Pseudo-class**: This pseudo-class is used to indicate to the user that it is checked.
- **`:valid` Pseudo-class**: This pseudo-class targets the input fields that meet the validation criteria.
- **`:invalid` Pseudo-class**: This pseudo-class targets the input fields that do not meet the validation criteria.
- **`:in-range` and `:out-of-range` Pseudo-classes**: These pseudo-classes apply styles to elements based on whether their values are within or outside specified range constraints.

- `:required` **Pseudo-class**: This pseudo-class targets `input` elements that have the `required` attribute. It signals to the user that they must fill out the field to submit the form.
- `:optional` **Pseudo-class**: This pseudo-class applies styles input elements that are not required and can be left empty.
- `:autofill` **Pseudo-class**: This pseudo-class applies styles to input fields that the browser automatically fills with saved data.

Location Pseudo-classes

- **Location Pseudo-classes**: These pseudo-classes are used for styling links and elements that are targeted within the current document.
- `:any-link` **Pseudo-class**: This pseudo-class is a combination of the `:link` and `:visited` pseudo-classes. So, it matches any anchor element with an href attribute, regardless of whether it's visited or not.
- `:link` **Pseudo-class**: This pseudo-class allows you to target all unvisited links on a webpage. You can use it to style links differently before the user clicks on them.
- `:local-link` **Pseudo-class**: This pseudo-class targets links that point to the same document. It can be useful when you want to differentiate internal links from external ones.
- `:visited` **Pseudo-class**: This pseudo-class targets a link the user has visited.
- `:target` **Pseudo-class**: This pseudo-class is used to apply styles to an element that is the target of a URL fragment.

Tree-structural Pseudo-classes

- **Tree-structural Pseudo-classes**: These pseudo-classes allow you to target and style elements based on their position within the document tree.
- `:root` **Pseudo-class**: This pseudo-class is usually the root `html` element. It helps you target the highest level in the document so you can apply a common style to the entire document.
- `:empty` **Pseudo-class**: Empty elements, that is, elements with no children other than white space, are also included in the document tree. That's why there's an `:empty` pseudo-class to target empty elements.
- `:nth-child(n)` **Pseudo-class**: This pseudo-class allows you to select elements based on their position within a parent.

- `:nth-last-child(n)` **Pseudo-class**: This pseudo-class enables you to select elements by counting from the end.
- `:first-child` **Pseudo-class**: This pseudo-class selects the first element in a parent element or the document.
- `:last-child` **Pseudo-class**: This pseudo-class selects the last element in a parent element or the document.
- `:only-child` **Pseudo-class**: This pseudo-class selects the only element in a parent element or the document.
- `:first-of-type` **Pseudo-class**: This pseudo-class selects the first occurrence of a specific element type within its parent.
- `:last-of-type` **Pseudo-class**: This pseudo-class selects the last occurrence of a specific element type within its parent.
- `:nth-of-type(n)` **Pseudo-class**: This pseudo-class allows you to select a specific element within its parent based on its position among siblings of the same type.
- `:only-of-type` **Pseudo-class**: This pseudo-class selects an element if it's the only one of its type within its parent.

Functional Pseudo-classes

- **Functional Pseudo-classes**: Functional pseudo-classes allow you to select elements based on more complex conditions or relationships. Unlike regular pseudo-classes which target elements based on a state (for example, `:hover`, `:focus`), functional pseudo-classes accept arguments.
- `:is()` **Pseudo-class**: This pseudo-class takes a list of selectors (ex. `ol`, `ul`) and selects an element that matches one of the selectors in the list.

Example Code

```
<p class="example">This text will change color.</p>
<p>This text will not change color.</p>
<p>This text will not change color.</p>
<p class="this-works-too">This text will change color.</p>
```

Example Code

```
p:is(.example, .this-works-too) {
    color: red;
}
```

- `:where()` **Pseudo-class**: This pseudo-class takes a list of selectors (ex. `ol`, `ul`) and selects an element that matches one of the selectors in the list. The difference between `:is` and `:where` is that the latter will have a specificity of 0.

  Example Code

  ```
  :where(h1, h2, h3) {
      margin: 0;
      padding: 0;
  }
  ```

- `:has()` **Pseudo-class**: This pseudo-class is often dubbed the `"parent"` selector because it allows you to style elements that contain child elements specified in the selector list.

  Example Code

  ```
  article:has(h2) {
      border: 2px solid hotpink;
  }
  ```

- `:not()` **Pseudo-class**: This pseudo-class is used to select elements that do not match the provided selector.

  Example Code

  ```
  p:not(.example) {
    color: blue;
  }
  ```

Pseudo-elements

- `::before` **Pseudo-element**: This pseudo-element uses the `content` property to insert cosmetic content like icons just before the element.
- `::after` **Pseudo-element**: This pseudo-element uses the `content` property to insert cosmetic content like icons just after the element.
- `::first-letter` **Pseudo-element**: This pseudo-element targets the first letter of an element's content, allowing you to style it.

- `::marker` **Pseudo-element**: This pseudo-element lets you select the marker (bullet or numbering) of list items for styling.

CSS Colors Review

Color Theory

- **Color Theory Definition**: This is the study of how colors interact with each other and how they affect our perception. It covers color relationships, color harmony, and the psychological impact of color.
- **Primary Colors**: These colors which are yellow, blue, and red, are the fundamental hues from which all other colors are derived.
- **Secondary Colors**: These colors result from mixing equal amounts of two primary colors. Green, orange, and purple are examples of secondary colors.
- **Tertiary Colors**: These colors result from combining a primary color with a neighboring secondary color. Yellow-Green, Blue-Green, and Blue-Violet are examples of tertiary colors.
- **Warm Colors**: These colors which include reds, oranges, and yellows, evoke feelings of comfort, warmth, and coziness.
- **Cool Colors**: These colors which include blues, green, and purples, evoke feelings of calmness, serenity, and professionalism.
- **Color Wheel**: The color wheel is a circular diagram that shows how colors relate to each other. It's an essential tool for designers because it helps them to select color combinations.
- **Analogous Color Schemes**: These color schemes create cohesive and soothing experiences. They have analogous colors, which are adjacent to each other in the color wheel.
- **Complementary Color Schemes**: These color schemes create high contrast and visual impact. Their colors are located on the opposite ends of the color wheel, relative to each other.
- **Triadic Color Scheme**: This color scheme has vibrant colors. They are made from colors that are approximately equidistant from each other. If they are connected, they form an equilateral triangle on the color wheel.
- **Monochromatic Color Scheme**: For this color scheme, all the colors are derived from the same base color by adjusting its lightness, darkness, and saturation. This evokes a feeling of unity and harmony while still creating contrast.

Different Ways to Work with Colors in CSS

- **Named Colors**: These colors are predefined color names recognized by browsers. Examples include `blue`, `darkred`, `lightgreen`.

- **`rgb()` Function**: RGB stands for Red, Green, and Blue — the primary colors of light. These three colors are combined in different intensities to create a wide range of colors. the `rgb()` function allows you to define colors using the RGB color model.

```css
p {
  color: rgb(255, 0, 0);
}
```

- **`rgba()` Function**: This function adds a fourth value, alpha, that controls the transparency of the color. If not provided, the alpha value defaults to `1`.

```css
div {
  background-color: rgba(0, 0, 255, 0.5);
}
```

- **`hsl()` Function**: HSL stands for Hue, Saturation, and Lightness — three key components that define a color.

```css
p {
  color: hsl(120, 100%, 50%);
}
```

- **`hsla()` Function**: This function adds a fourth value, alpha, that controls the opacity of the color.

```css
div {
  background-color: hsla(0, 100%, 50%, 0.5);
}
```

- **Hexadecimal**: A hex code (short for hexadecimal code) is a six-character string used to represent colors in the RGB color model. The "hex" refers to the base-16 numbering system, which uses digits 0 to 9 and letters A to F.

```css
h1 {
  color: #FF5733; /* A reddish-orange color */
}

p {
  background-color: #4CAF50; /* A shade of green */
}
```

Linear and Radial Gradients

- **Linear Gradients**: These gradients create a gradual blend between colors along a straight line. You can control the direction of this line using keywords like `to top`, `to right`, `to bottom right`, or angles like `45deg`. You can use any valid CSS color and as many color stops as you would like.

```css
.linear-gradient {
  background: linear-gradient(45deg, red, #33FF11, rgba(100,
100, 255, 0.5));
  height: 40vh;
}
```

- **Radial Gradients**: These gradients create circular or elliptical gradients that radiate from a central point.

```css
.radial-gradient {
  background: radial-gradient(circle, red, blue);
  height: 40vh;
}
```

Styling Forms Review

Best Practices for Styling Inputs

- **Styling Inputs**: As with all text elements, you need to ensure the styles you apply to text inputs are accessible. This means the font needs to be adequately sized, and the color needs to have sufficient contrast with the background. Input elements are also focusable. When you are editing your styles, you should take care that you preserve a noticeable indicator when the element has focus, such as a bold border.

Using `appearance: none` for Inputs

- **`appearance: none`**: Browsers apply default styling to a lot of elements. The `appearance: none` CSS property gives you complete control over the styling, but comes with some caveats. When building custom styles for input elements, you will need to make sure focus and error indicators are still present.

Commons Issues Styling `datetime-local` and `color` Properties

- **Common Issues**: These special types of inputs rely on complex pseudo-elements to create things like date and color pickers. This presents a significant challenge for styling these inputs. One challenge is that the default styling is entirely browser-dependent, so the CSS you write to make the picker look the way you intend may be entirely different on another browser.

CSS Layouts and Effects Review

CSS Overflow Property

- **Definition**: Overflow refers to the way elements handle content that exceeds, or "overflows", the size of the containing element. Overflow is two-dimensional.
- **`overflow-x`**: The x-axis determines horizontal overflow.
- **`overflow-y`**: the y-axis determines vertical overflow.
- **`overflow`**: Shorthand property for `overflow-x` and `overflow-y`. If given one value, both overflows will use it. If given two values, the `overflow-x` will use the first, and the `overflow-y` will use the second.

CSS Transform Property

- **Definition**: This property enables you to apply various transformations to elements, such as rotating, scaling, skewing, or translating (moving) them in 2D or 3D space.
- **`translate()` Function**: This function is used to move an element from its current position.
- **`scale()` Function**: This function allows you to change the size of an element.
- **`rotate()` Function**: This function allows you to rotate an element.
- **`skew()` Function**: This function allows you to skew an element.
- **Transforms and Accessibility**: If you're using transform to hide or reveal content, make sure that the content is still accessible to screen readers and keyboard navigation. Hidden content should be truly hidden, such as by using `display: none` or `visibility: hidden`, rather than simply being visually moved off-screen.

The Box Model

- **Definition**: In the CSS box model, every element is surrounded by a box. This box consists of four components: the content area, `padding`, `border`, `margin`.
- **Content Area**: The content area is the innermost part of the box. It's the space that contains the actual content of an element, like text or images.

- **`padding`**: The padding is the area immediately after the content area. It's the space between the content area and the border of an element.
- **`border`**: The border is the outer edge or outline of an element in the CSS box model. It's the visual boundary of the element.
- **`margin`**: The margin is the space outside the border of an element. It determines the distance between an element and other elements around it.

Margin Collapse

- **Definition**: This behavior occurs when the vertical margins of adjacent elements overlap, resulting in a single margin equal to the larger of the two. This behavior applies only to vertical margins (top and bottom), not horizontal margins (left and right).

The `content-box` and `border-box` Property Values

- **`box-sizing` Property**: This property is used to determine how the final `width` and `height` are calculated for an HTML element.
- **`content-box` Value**: In the `content-box` model, the `width` and `height` that you set for an element determine the dimensions of the content area but they don't include the `padding`, `border`, or `margin`.
- **`border-box` Value**: With `border-box`, the `width` and `height` of an element include the content area, the `padding`, and the `border`, but they don't include the `margin`.

CSS Reset

- **Definition**: A CSS reset is a stylesheet that removes all or some of the default formatting that web browsers apply to HTML elements. Third party options for CSS resets include `sanitize.css` and `normalize.css`.

CSS Filter Property

- **Definition**: This property can be used to create various effects such as blurring, color shifting, and contrast adjustment.
- **`blur()` Function**: This function applies a Gaussian blur to the element. The amount is defined in pixels and represents the radius of the blur.

- `brightness()` **Function**: This function adjusts the brightness of the element. A value of 0% will make the element completely black, while values over 100% will increase the brightness.
- `contrast()` **Function**: This function adjusts the contrast of the element. A value of 0% will make the element completely grey, 100% will make the element appear normally, and values greater than 100% will increase the contrast.
- `grayscale()` **Function**: This function converts the element to grayscale. The amount is defined as a percentage, where 100% is completely grayscale and 0% leaves the image unchanged.
- `sepia()` **Function**: This function applies a sepia tone to the element. Like grayscale, it uses a percentage value.
- `hue-rotate()` **Function**: This function applies a hue rotation to the element. The value is defined in degrees and represents a rotation around the color circle.

CSS Typography Review

Introduction to Typography

- **Definition**: Typography is the art of choosing the right fonts and format to make text visually appealing and easy to read. "Type" refers to how the individual characters are designed and arranged.
- **Typeface Definition**: A typeface is the overall design and style of a set of characters, numbers, and symbols. It's like a blueprint for a family of fonts.
- **Font Definition**: A font is a specific variation of a typeface with specific characteristics, such as size, weight, style, and width.
- **Serif Typeface**: This typeface has a classical style with small lines at the end of characters. Serif typefaces are commonly used for printed materials, like books.
- **Sans Serif Typeface**: This typeface has a more modern look, without the small lines at the end of characters. Sans Serif typefaces are commonly used in digital design because they are easy to read on screen. Some examples include Helvetica, Arial, and Roboto.
- **Font Weight**: This is the thickness of the characters, including light, regular, bold, and black.
- **Font Style**: This is the slant and orientation of the characters, like italic and oblique.
- **Font Width**: This is the horizontal space occupied by characters, like condensed and expanded.
- **Baseline**: This is the imaginary line on which most characters rest.
- **Cap Height**: This is the height of uppercase letters, measured from the baseline to the top.
- **X-height**: This is the average height of lowercase letters, excluding ascenders and descenders.
- **Ascenders**: These are the parts of lowercase letters that extend above the x-height, such as the tops of the letters 'h', 'b', 'd', and 'f'.
- **Descenders**: These are the parts of lowercase letters that extend below the baseline, such as the tails of 'y', 'g', 'p', and 'q'.
- **Kerning**: This is how space is adjusted between specific pairs of characters to improve their readability and aesthetics. For example, reducing the space between the letters A and V.

- **Tracking**: This is how space is adjusted between all characters in a block of text. It's essentially the distance between the characters. It affects how dense and open the text will be.
- **Leading**: This is the vertical space between lines of text. It's measured from the baseline of one line to the baseline of the next line.
- **Best Practices with Typography**: You should choose clear and simple fonts to make your designs easy to understand. This is particularly important for the main text of your website. Users are more likely to engage with your content if the font is easy to read. You should use two or three fonts at most to create a visual consistency. Using too many fonts can make the text more difficult to read and weaken your brand's identity. This can also impact the user experience by increasing the load time of the website. You can use font size to create a visual hierarchy for headings, subheadings, paragraphs, and other elements. For example, the main heading on a webpage should have a larger font, followed by subheadings with smaller font sizes. This will give every element in the hierarchy a specific font size that helps users navigate through the structure visually.

CSS `font-family` Property

- **Definition**: A font family is a group of fonts that share a common design. All the fonts that belong to the same family are based on the same core typeface but they also have variations in their style, weight, and width. You can specify multiple font families in order of priority, from highest to lowest, by separating them with commas. These alternative fonts will act as fallback options. You should always include a generic font family at the end of the font-family list.

Example Code

```
font-family: Arial, Lato;
```

- **Common Font Families**: Here are some common font families used in CSS: serif, sans-serif, monospace, cursive, fantasy

Web Safe Fonts

- **Definition**: These fonts are a subset of fonts that are very likely to be installed on a computer or device. When the browser has to render a font, it tries to find the font file on the user's system. But if the font is not found, it will usually fall back to a default system font.
- **Web Safe Fonts**:

- Sans-serif fonts are commonly used for web development because they don't have small "feet" or lines at the end of the characters, so they're easy to read on screen. Some examples of web-safe sans-serif fonts are: Arial, Verdana, and Trebuchet MS.
- In contrast, serif fonts do have small "feet" at the end of the characters, so they're commonly used for traditional print. Web safe serif fonts include: Times New Roman and Georgia.

At-rules and the `@font-face` At-rule

- **Definition**: At-rules are statements that provide instructions to the browser. You can use them to define various aspects of the stylesheet, such as media queries, keyframes, font faces, and more.
- **`@font-face`**: This allows you to define a custom font by specifying the font file, format, and other important properties, like weight and style. For the `@font-face` at-rule to be valid, you also need to specify the `src` property. This property contains references to the font resources.

Example Code

```
@font-face {
  font-family: "MyCustomFont";
  src: url("path/to/font.woff2"),
       url("path/to/font.woff"),
       url("path/to/font.otf");
}
```

- **Font Formats**: For each font resource, you can also specify the format. This is optional. It's a hint for the browser on the font format. If the format is omitted, the resource will be downloaded and the format will be detected after it's downloaded. If the format is invalid, the resource will not be downloaded. Possible font formats include `collection`, `embedded-opentype`, `opentype`, `svg`, `truetype`, `woff`, and `woff2`.

Example Code

```
@font-face {
  font-family: "MyCustomFont";
  src: url("path/to/font.woff2") format("woff2"),
       url("path/to/font.otf") format("opentype"),
       url("path/to/font.woff") format("woff");
}
```

- **`woff` and `woff2`**: `woff` stands for "Web Open Font Format." It's a widely used font format developed by Mozilla in collaboration with Type Supply, LettError, and other organizations. The difference between `woff` and `woff2` is the algorithm used to compress the data.
- **OpenType**: This is a format for scalable computer fonts developed by Microsoft and Adobe that allows users to access additional features in a font. It's widely used across major operating systems.
- **`tech()`**: This is used to specify the technology of the font resource. This is optional.

Example Code

```
@font-face {
  font-family: "MyCustomFont";
  src: url("path/to/font.woff2") format("woff2"),
       url("path/to/font.otf") format("opentype") tech(color-
COLRv1),
       url("path/to/font.woff") format("woff");
}
```

Working With External Fonts

- **Definition**: An external font is a font file that is not included directly within your project files.They're usually hosted on a separate server. To include these external fonts inside your project, you can use a `link` element or `@import` statement inside your CSS.
- **Google Fonts**: This is a Google service that offers a collection of fonts, many of which are designed specifically for web development.

Here is an example using the `link` element:

Example Code

```
<link rel="preconnect" href="https://fonts.googleapis.com">
```

```
<link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
<link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wgh
t@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;
1,700;1,900&display=swap" rel="stylesheet">
```

Example Code

```
.roboto-thin {
  font-family: "Roboto", sans-serif;
  font-weight: 100;
  font-style: normal;
}
```

Here is an example using the `@import` statement:

Example Code

```
@import
url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght
@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1
,700;1,900&display=swap');
```

- **Font Squirrel**: This is another popular resource that you can use for adding custom external fonts to your projects.

`text-shadow` Property

- **Definition**: This property is used to apply shadows to text. You need to specify the X and Y offset, which represent the horizontal and vertical distance of the shadow from the text, respectively. These values are required. Positive values of the X offset and Y offset will move the shadow right and down, respectively, while negative values will move the shadow left and up.

Example Code

```
p {
  text-shadow: 3px 2px;
}
```

- **Shadow Color**: You can also customize the color of the shadow by specifying this value before or after the offset. If the color is not specified, the browser will determine the color automatically, so it's usually best to set it explicitly.

Example Code

```
p {
  text-shadow: 3px 2px #00ffc3;
}
```

- **Blur Radius**: The blur radius is optional but will make the shadow look a lot smoother and more subtle. The default value of the radius blur is zero. The higher the value, the bigger the blur, which means that the shadow becomes lighter.

Example Code

```
p {
  text-shadow: 3px 2px 3px #00ffc3;
}
```

- **Applying Multiple Text Shadows**: The text can have more than one shadow. The shadows will be applied in layers, from front to back, with the first shadow at the top.

Example Code

```
p {
  text-shadow:
    3px 2px 3px #00ffc3,
    -3px -2px 3px #0077ff,
    5px 4px 3px #dee7e5;
}
```

CSS Accessibility Review

Color Contrast Tools

- **WebAIM's Color Contrast Checker**: This online tool allows you to input the foreground and background colors of your design and instantly see if they meet the Web Content Accessibility Guidelines (WCAG) standards.
- **TPGi Colour Contrast Analyzer**: This is a free color contrast checker that allows you to check if your websites and apps meet the Web Content Accessibility Guidelines (WCAG) standards. This tool also comes with a Color Blindness Simulator feature which allows you to see what your website or app looks like for people with color vision issues.

Accessibility Tree

Accessibility tree is a structure used by assistive technologies, such as screen readers, to interpret and interact with the content on a webpage.

`max()` Function

The `max()` function returns the largest of a set of comma-separated values:

```
img {
  width: max(250px, 25vw);
}
```

In the above example, the width of the image will be 250px if the viewport width is less than 1000 pixels. If the viewport width is greater than 1000 pixels, the width of the image will be 25vw. This is because 25vw is equal to 25% of the viewport width.

Best Practices with CSS and Accessibility

- **`display: none;`**: Using `display: none;` means that screen readers and other assistive technologies won't be able to access this content, as it is not included in the accessibility tree. Therefore, it is important to use this method only when you want to completely remove content from both visual presentation and accessibility.

- `visibility: hidden;`: This property and value hides the content visually but keeps it in the document flow, meaning it still occupies space on the page. These elements will also no longer be read by screen readers because they will have been removed from the accessibility tree.
- `.sr-only` **CSS class**: This is a common technique used to visually hide content while keeping it accessible to screen readers.

```
.sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  white-space: nowrap;
  border: 0;
}
```

- `scroll-behavior: smooth;`: This property and value enables a smooth scrolling behavior.
- `prefers-reduced-motion` **feature**: This is a media feature that can be used to detect the user's animation preference.

```
@media (prefers-reduced-motion: no-preference) {
  * {
    scroll-behavior: smooth;
  }
}
```

In the above example, smooth scrolling is enabled if the user doesn't have animation preference set on their device.

Hiding Content with HTML Attributes

- **`aria-hidden` attribute**: Used to hide an element from people using assistive technology such as screen readers. For example, this can be used to hide decorative images that do not provide any meaningful content.
- **`hidden` attribute**: This attribute is supported by most modern browsers and hides content both visually and from the accessibility tree. It can be easily toggled with JavaScript.

```html
<p aria-hidden>This paragraph is visible for sighted users, but is hidden from assistive technology.</p>

<p hidden>This paragraph is hidden from both sighted users and assistive technology.</p>
```

Accessibility Issue of the `placeholder` Attribute

Using placeholder text is not great for accessibility. Too often, users confuse the placeholder text with an actual input value - they think there is already a value in the input.

CSS Positioning Review

Working With Floats

- **Definition**: Floats are used to remove an element from its normal flow on the page and position it either on the left or right side of its container. When this happens, the text will wrap around that floated content.

Example Code

```
float: left;
float: right;
```

- **Clearing Floats**: The `clear` property is used to determine if an element needs to be moved below the floated content. When you have multiple floated elements stacked next to each other, there could be issues with overlap and collapsing in the layouts. So a `clearfix` hack was created to fix this issue.

Example Code

```
.clearfix::after {
  content: "";
  display: block;
  clear: both;
}
```

Static, Relative and Absolute Positioning

- **Static Positioning**: This is the normal flow for the document. Elements are positioned from top to bottom and left to right one after another.
- **Relative Positioning**: This allows you to use the `top`, `left`, `right` and `bottom` properties to position the element within the normal document flow. You can also use relative positioning to make elements overlap with other elements on the page.

Example Code

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

- **Absolute Positioning**: This allows you to take an element out of the normal document flow, making it behave independently from other elements.

Example Code

```
.positioned {
  position: absolute;
  top: 30px;
  left: 30px;
  background-color: coral;
}
```

Fixed and Sticky Positioning

- **Fixed Positioning**: When an element is positioned with `position: fixed`, it is removed from the normal document flow and placed relative to the viewport, meaning it stays in the same position even when the user scrolls. This is often used for elements like headers or navigation bars that need to remain visible at all times.

Example Code

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

- **Sticky Positioning**: This type of positioning will act as a relative positioned element as you scroll down the page. If you specify a `top`, `left`, `right` or `bottom` property, then the element will stop acting like a relatively positioned element and start behaving like a fixed position element.

Example Code

```
.positioned {
  position: sticky;
  top: 30px;
  left: 30px;
}
```

Working With the `z-index` Property

- **Definition**: The `z-index` property in CSS is used to control the vertical stacking order of positioned elements that overlap on the page.

Example Code

```
.container {
  position: relative;
  width: 300px;
  height: 300px;
  border: 1px solid black;
}

.box1 {
  position: absolute;
  z-index: 1;
  background: lightcoral;
  top: 20px;
  left: 20px;
  width: 100px;
  height: 100px;
}
```

CSS Attribute Selectors Review

Working with Different Attribute Selectors and Links

- **Definition**: The `attribute` selector allows you to target HTML elements based on their attributes like the `href` or `title` attributes.

Example Code

```
a[href] {
  color: blue;
  text-decoration: underline;
}
```

- **`title` Attribute**: This attribute provides additional information about an element. Here is how you can target links with the `title` attribute:

Example Code

```
a[title] {
  font-weight: bold;
  text-decoration: none;
}
```

- **Combine selectors to match links that have both `href` and `title` attributes**: Combines multiple attribute selectors.

Example Code

```
a[href][title] {
  color: green;
}
```

- **Match a single word within a space-separated list**: Targets links that have a specific class word.

Example Code

```css
a[class~="primary"] {
  color: red;
  font-weight: bold;
}
```

- **Match values that start with a specific prefix**: Targets links starting with https://

Example Code

```css
a[href^="https://"] {
  color: green;
  text-decoration: underline;
}
```

- **Match values that end with a specific suffix**: Targets links ending with .jpg

Example Code

```css
a[href$=".jpg"] {
  color: darkgreen;
  text-decoration: underline dotted;
}
```

- **Match values that contain a substring anywhere**: Targets links that contain https anywhere in the value.

Example Code

```css
a[href*="https"] {
  color: teal;
}
```

- **Summary**: These patterns make it easy to consistently style links based on their attributes and values.

Targeting Elements with the `lang` and `data-lang` Attribute

- **`lang` Attribute**: This attribute is used in HTML to specify the language of the content within an element. You might want to style elements differently based on the language they are written in, especially on a multilingual website.

Example Code

```
p[lang="en"] {
  font-style: italic;
}
```

- **`data-lang` Attribute**: Custom data attributes like the `data-lang` attribute are commonly used to store additional information in elements, such as specifying the language used within a specific section of text. Here is how you can style elements based on the `data-lang` attribute:

Example Code

```
div[data-lang="fr"] {
  color: blue;
}
```

Working with the Attribute Selector, Ordered List Elements and the `type` Attribute

- **`type` Attribute**: When working with ordered lists in HTML, the `type` attribute allows you to specify the style of numbering used, such as numerical, alphabetical, or Roman numerals

```
/*Example targeting uppercase alphabetical numbering*/
ol[type="A"] {
  color: purple;
  font-weight: bold;
}
```

```css
/*Example targeting lowercase Roman numerals*/
ol[type="i"] {
  color: green;
}
```

Responsive Web Design

- **Definition**: The core principle of responsive design is adaptability – the ability of a website to adjust its layout and content based on the screen size and capabilities of the device it's being viewed on.
- **Fluid grids**: These use relative units like percentages instead of fixed units like pixels, allowing content to resize and reflow based on screen size.
- **Flexible images**: These are set to resize within their containing elements, ensuring they don't overflow their containers on smaller screens.

Media Queries

- **Definition**: This allows developers to apply different styles based on the characteristics of the device, primarily the viewport width.

Example Code

```css
@media screen and (min-width: 768px) {
  /* Styles for screens at least 768px wide */
}
```

- `all` **Media Type**: This is suitable for all devices. This is the default if no media type is specified.
- `print` **Media Types**: This is intended for paged material and documents viewed on a screen in print preview mode.
- `screen` **Media Types**: This is intended primarily for screens.
- `aspect-ratio`: This describes the ratio between the width and height of the viewport.

Example Code

```css
@media screen and (aspect-ratio: 16/9) {
  /* Styles for screens with a 16:9 aspect ratio */
}
```

- `orientation`: This is used to indicate whether the device is in landscape or portrait orientation.

Example Code

```
@media screen and (orientation: landscape) {
  /* Styles for landscape orientation */
}
```

- `resolution`: This is used to describe the resolution of the output device in dots per inch (dpi) or dots per centimeter (dpcm).

Example Code

```
@media screen and (min-resolution: 300dpi) {
  /* Styles for high-resolution screens */
}
```

- `hover`: This is used to test whether the primary input mechanism can hover over elements.

Example Code

```
@media (hover: hover) {
  /* Styles for devices that support hover */
}
```

- `prefers-color-scheme`: This is used to detect if the user has requested a light or dark color theme.
- **Media Queries and Logical Operators**: The `and` operator is used to combine multiple media features, while `not` and `only` can be used to negate or isolate media queries.

Example Code

```
@media screen and (min-width: 768px) and (orientation:
landscape) {
  /* Styles for landscape screens at least 768px wide */
}
```

Common Media Breakpoints

- **Definition**: Media breakpoints are specific points in a website's design where the layout and content adjust to accommodate different screen sizes. There are some general breakpoints that you can use to target phones, tablets and desktop screens. But it is not wise to try to chase down every single possible screen size for different devices.

Example Code

```
/* Styles for screens wider than 768px */
@media screen and (min-width: 768px) {
  body {
    font-size: 1.125rem;
  }
}
```

- **Small Devices (smartphones)**: up to 640px
- **Medium Devices (tablets)**: 641px to 1024px
- **Large Devices (desktops)**: 1025px and larger

Mobile first approach

- **Definition**: The `mobile-first` approach is a design philosophy and development strategy in responsive web design that prioritizes creating websites for mobile devices before designing for larger screens.

Example Code

```
/* Base styles for mobile */
.container {
  width: 100%;
  padding: 10px;
}

/* Styles for larger screens */
@media screen and (min-width: 768px) {
  .container {
```

```css
    width: 750px;
    margin: 0 auto;
    padding: 20px;
  }
}

@media screen and (min-width: 1024px) {
  .container {
    width: 960px;
  }
}
```

CSS Variables Review

CSS Custom Properties (CSS Variables)

- **Definition**: CSS custom properties, also known as CSS variables, are entities defined by CSS authors that contain specific values to be reused throughout a document. They are a powerful feature that allows for more efficient, maintainable, and flexible stylesheets. Custom properties are particularly useful in creating themeable designs. You can define a set of properties for different themes:

Example Code

```css
:root {
  --bg-color: white;
  --text-color: black;
}

.dark-theme {
  --bg-color: #333;
  --text-color: white;
}

body {
  background-color: var(--bg-color);
  color: var(--text-color);
}
```

The `@property` Rule

- **Definition**: The `@property` rule is a powerful CSS feature that allows developers to define custom properties with greater control over their behavior, including how they animate and their initial values.

Example Code

```
@property --property-name {
  syntax: '<type>';
  inherits: true | false;
  initial-value: <value>;
}
```

- **`--property-name`**: This is the name of the custom property you're defining. Like all custom properties, it must start with two dashes. `--property-name` can be things like `<color>`, `<length>`, `<number>`, `<percentage>`, or more complex types.
- **`syntax`**: This defines the type of the property.
- **`inherits`**: This specifies whether the property should inherit its value from its parent element.
- **`initial-value`**: This sets the default value of the property.
- **Gradient Example Using the `@property` Rule**: This example creates a gradient that smoothly animates when the element is hovered over.

Example Code

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

Example Code

```
@property --gradient-angle {
  syntax: "<angle>";
  inherits: false;
  initial-value: 0deg;
}

.gradient-box {
  width: 100px;
  height: 100px;
  background: linear-gradient(var(--gradient-angle), red,
blue);
  transition: --gradient-angle 0.5s;
}

.gradient-box:hover {
  --gradient-angle: 90deg;
}
```

- **Fallbacks**: When using the custom property, you can provide a fallback value using the `var()` function, just as you would with standard custom properties:
  - Example Code

  -
    ```
    .button {
      background-color: var(--main-color, #3498db);
    }
    ```
  - **Assignment**

CSS Grid Review

CSS Grid Basics

- **Definition**: CSS Grid is a two-dimensional layout system used to create complex layouts in web pages. Grids will have rows and columns with gaps between them. To define a grid layout, you would set the `display` property to `grid`.

Example Code

```
.container {
  display: grid;
}
```

- **The `fr` (Fractional) Unit**: This unit represents a fraction of the space within the grid container. You can use the `fr` unit to create flexible grids.
- **Creating Gaps Between Tracks**: There are three ways to create gaps between tracks in CSS grid. You can use the `column-gap` property to create gaps between columns. You can use the `row-gap` property to create gaps between rows. Or you can use the `gap` shorthand property to create gaps between both rows and columns.
- **`grid-template-columns`**: This is used to set lines names and sizing for the grid track columns.

Example Code

```
.container {
  display: grid;
  width: 100%;
  grid-template-columns: 30px 1fr;
}
```

- **`grid-template-rows`**: This is used to set lines names and sizing for the grid track rows.
- **`grid-auto-flow`**: The determines how auto placed items fit in the grid.

Example Code

```css
.container {
  display: grid;
  width: 100%;
  grid-auto-flow: column;
}
```

- `grid-auto-columns`: This is used to set the size for columns created implicitly.

Example Code

```css
.container {
  display: grid;
  width: 100%;
  grid-auto-columns: auto;
}
```

- `place-items`: This is used to align items for both block and inline directions.
- `align-items`: This is used to set the alignment for the items in a grid container.
- `repeat()` **Function**: This function is used to repeat sections in the track listing. Instead of writing `grid-template-columns: 1fr 1fr 1fr;` you can use the `repeat()` function instead.

Example Code

```css
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

- **Explicit Grid**: You can specify the number of lines or tracks using the `grid-template-columns` or `grid-template-rows` properties.
- **Implicit Grid**: When items are placed outside of the grid, then rows and columns are automatically created for those outside elements.
- `minmax()` **Function**: This function is used to set the minimum and maximum sizes for a track.

Example Code

```css
.container {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-auto-rows: minmax(150px, auto);
}
```

- **Line-based Placement**: All grids have lines. To specify where the item begins on a line, you can use the `grid-column-start` and `grid-row-start` properties. To specify where the item ends on the line, you can use the `grid-column-end` and `grid-row-end` properties. You can also choose to use the `grid-column` or `grid-row` shorthand properties instead.

Here is an example of using the `grid-column` property to make an element stretch across all columns.

Example Code

```css
.element {
  grid-column: 1 / -1;
}
```

- **`grid-template-areas`**: The property is used to provide a name for the items you are going to position on the grid.

Example Code

```html
<div class="container">
  <div class="header">Header</div>
  <div class="sidebar">Sidebar</div>
  <div class="main">Main Content</div>
  <div class="footer">Footer</div>
</div>
```

Example Code

```css
.container {
  display: grid;
  grid-template-columns: 200px 1fr;
  grid-template-rows: auto 1fr auto;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
  gap: 20px;
}

.header {
  grid-area: header;
  background-color: #4CAF50;
  padding: 10px;
  color: white;
}

.sidebar {
  grid-area: sidebar;
  background-color: #f4f4f4;
  padding: 10px;
}

.main {
  grid-area: main;
  background-color: #e0e0e0;
  padding: 10px;
}
```

```css
.footer {
  grid-area: footer;
  background-color: #4CAF50;
  padding: 10px;
  color: white;
}
```

Debugging CSS

- **DevTools (Developer Tools)**: DevTools allow you to inspect and modify your CSS in real-time. The Styles pane shows all the CSS rules applied to the selected element, including inherited styles. You can toggle individual properties on and off, edit values, and even add new rules directly in the browser. This immediate feedback is incredibly useful for experimenting with different styles without modifying your source code.
- **CSS Validators**: Validators are used to check your CSS against the official specifications and reports any errors or warnings. A popular validator you can use is the W3C CSS Validator.
- **Debugging Responsive Designs**: The DevTools has an option to allow you to simulate how your site looks on various screen sizes and devices. This can help you identify breakpoint issues or styles that don't scale well across different viewport sizes.

CSS Animation Basics

- **Definition**: CSS animations allow you to create dynamic, visually engaging effects on web pages without the need for JavaScript or complex programming. They provide a way to smoothly transition elements between different styles over a specified duration.
- **The `@keyframes` Rule**: This rule defines the stages and styles of the animation. It specifies what styles the element should have at various points during the animation.

Example Code

```
@keyframes slide-in {
  0% {
    transform: translateX(-100%);
  }
  100% {
    transform: translateX(0);
  }
}
```

- `animation` **Property**: This is the shorthand property used to apply animations.
- `animation-name`: This specifies the name for the `@keyframes` rule to use.
- `animation-duration`: This sets how long the animation should take to complete.
- `animation-timing-function`: This defines how the animation progresses over time (such as ease, linear, ease-in-out).
- `animation-delay`: This specifies a delay before the animation starts.
- `animation-iteration-count`: This sets how many times the animation should repeat.
- `animation-direction`: This determines whether the animation should play forwards, backwards, or alternate.
- `animation-fill-mode`: This specifies how the element should be styled before and after the animation.
- `animation-play-state`: This allows you to pause and resume the animation.

Accessibility and the `prefers-reduced-motion` Media Query

- **The `prefers-reduced-motion` Media Query**: One of the primary accessibility concerns with animations is that they can cause discomfort or even physical harm to some users. People with vestibular disorders or motion sensitivity may experience dizziness, nausea, or headaches when exposed to certain types of movement on screen. The `prefers-reduced-motion` media query allows web developers to detect if the user has requested minimal animations or motion effects at the system level.

Example Code

```
.animated-element {
  transition: transform 0.3s ease-in-out;
}

@media (prefers-reduced-motion: reduce) {
  .animated-element {
    transition: none;
  }
}
```

CSS Basics

- **What is CSS?**: Cascading Style Sheets (CSS) is a markup language used to apply styles to HTML elements. CSS is used for colors, background images, layouts and more.
- **Basic Anatomy of a CSS Rule**: A CSS rule is made up of two main parts: a selector and a declaration block. A selector is a pattern used in CSS to identify and target specific HTML elements for styling. A declaration block applies a set of styles for a given selector, or selectors.
- **`meta name="viewport"` Element**: This `meta` element gives the browser instructions on how to control the page's dimensions and scaling on different devices, particularly on mobile phones and tablets.
- **Default Browser Styles**: Each HTML element will have default browser styles applied to them. This usually includes items like default margins and paddings.

Inline, Internal, and External CSS

- **Inline CSS**: These styles are written directly within an HTML element using the `style` attribute. Most of the time you will not be using inline CSS due to separation of concerns.
- **Internal CSS**: These styles are written within the `<style>` tags inside the `head` section of an HTML document. This can be useful for creating short code examples, but usually, you will not use internal CSS.
- **External CSS**: These styles are written in a separate CSS file and linked to the HTML document using the `link` element in the `head` section. For most projects, you will use an external CSS file over internal or inline CSS.

Working With the `width` and `height` Properties

- **`width` Property**: This property specifies the width of an element. If you do not specify a width, then the default is set to `auto`. This lets the browser determine the element's width based on its content, parent, and display type.
- **`min-width` Property**: This property specifies the minimum width for an element.
- **`max-width` Property**: This property specifies the maximum width for an element.
- **`height` Property**: This property specifies the height of an element. Similarly, the height is `auto` by default, which means it will adjust to the content inside.

- **`min-height` Property**: This property specifies the minimum height for an element.
- **`max-height` Property**: This property specifies the maximum height for an element.

Different Types of CSS Combinators

- **Descendant Combinator (` `) *a single space between selectors**: This combinator is used to target elements that are descendants of a specified parent element.
- **Child Combinator (`>`)**: This combinator is used to select elements that are direct children of a specified parent element.
- **Next-sibling Combinator (`+`)**: This combinator selects an element that immediately follows a specified sibling element.
- **Subsequent-sibling Combinator (`~`)**: This combinator selects all siblings of a specified element that come after it.

Inline, Block, and Inline-Block Level Elements

- **Inline Level Elements**: Inline elements only take up as much width as they need and do not start on a new line. These elements flow within the content, allowing text and other inline elements to appear alongside them. Common inline elements are `span`, `anchor`, and `img` elements.
- **Block Level Elements**: Block level elements that take up the full width available to them by default, stretching across the width of their container. Some common block-level elements are `div`, `paragraph`, and `section` elements.
- **Inline-Block Level Elements**: You can set an element to `inline-block` by using the `display` property. These elements behave like inline elements but can have a `width` and `height` set like block-level elements.

Margin and Padding

- **`margin` Property**: This property is used to apply space outside the element, between the element's border and the surrounding elements.
- **`margin` Shorthand**: You can specify 1–4 values to set the margin sides. One value applies to all four sides; two values set `top` and `bottom`,

then `right` and `left`; three values set `top`, horizontal (`right` and `left`), then `bottom`; four values set `top`, `right`, `bottom`, `left`.

- **`padding` Property**: This property is used to apply space inside the element, between the content and its border.
- **`padding` Shorthand**: You can specify 1–4 values to set the padding sides. One value applies to all four sides; two values set `top` and `bottom`, then `right` and `left`; three values set `top`, horizontal (`right` and `left`), then `bottom`; four values set `top`, `right`, `bottom`, `left`.

CSS Specificity

- **Inline CSS Specificity**: Inline CSS has the highest specificity because it is applied directly to the element. It overrides any internal or external CSS. The specificity value for inline styles is (1, 0, 0, 0).
- **Internal CSS Specificity**: Internal CSS is defined within a `style` element in the `head` section of the HTML document. It has lower specificity than inline styles but can override external styles.
- **External CSS Specificity**: External CSS is linked via a `link` element in the `head` section and is written in separate `.css` files. It has the lowest specificity but provides the best maintainability for larger projects.
- **Universal Selector (`*`)**: a special type of CSS selector that matches any element in the document. It is often used to apply a style to all elements on the page, which can be useful for resetting or normalizing styles across different browsers. The universal selector has the lowest specificity value of any selector. It contributes 0 to all parts of the specificity value (0, 0, 0, 0).
- **Type Selectors**: These selectors target elements based on their tag name. Type selectors have a relatively low specificity compared to other selectors. The specificity value for a type selector is (0, 0, 0, 1).
- **Class Selectors**: These selectors are defined by a period (`.`) followed by the class name. The specificity value for a class selector is (0, 0, 1, 0). This means that class selectors can override type selectors, but they can be overridden by ID selectors and inline styles.
- **ID Selectors**: ID selectors are defined by a hash symbol (`#`) followed by the ID name. ID selectors have a very high specificity, higher than type selectors and class selectors, but lower than inline styles. The specificity value for an ID selector is (0, 1, 0, 0).
- **`!important` keyword**: used to give a style rule the highest priority, allowing it to override any other declarations for a property. When used, it forces the browser to apply the specified style, regardless of the specificity of other

selectors. You should be cautious when using `!important` because it can make your CSS harder to maintain and debug.

- **Cascade Algorithm**: An algorithm used to decide which CSS rules to apply when there are multiple styles targeting the same element. It ensures that the most appropriate styles are used, based on a set of well-defined rules.
- **CSS Inheritance**: The process by which styles are passed down from parent elements to their children. Inheritance allows you to define styles at a higher level in the document tree and have them apply to multiple elements without explicitly specifying them for each element.

Design Terminology

- **Layout**: This is how visual elements are arranged on a page or screen to communicate a message. These elements may include text, images, and white space.
- **Alignment**: This is how the elements are placed in relation to one another. Using alignment correctly is helpful for making the design look clean and organized.
- **Composition**: This is the act of arranging elements to create a harmonious design. It determines how elements like images, text, and shapes relate to each other and contribute to the design in an artistic way.
- **Balance**: This is how visual weight is distributed within a composition. Designers aim to create an equilibrium through symmetrical or asymmetrical arrangements.
- **Scale**: This refers to comparing the dimensions or size of one element to that of another.
- **Hierarchy**: This establishes the order of importance of the elements in a design. It's about making sure that the most important information is noticed first.
- **Contrast**: This is the process of creating clear distinctions between the elements. You can do this through variations in color, size, shape, texture, or any other visual characteristic. Strong contrast is also helpful for improving readability.
- **White Space(negative space)**: This is the empty space in a design. It's the area surrounding the elements.
- **UI(User Interface)**: UI includes the visual and interactive elements that users can see on their screens, like icons, images, text, menus, links, and buttons.
- **UX(User Experience)**: UX is about how users feel when using a product or service. An application with a well-designed user experience is intuitive, easy to use, efficient, accessible, and enjoyable.

- **Design Brief**: This is a document that outlines the objectives, goals, and requirements of a project. It is a roadmap that guides the design process and ensures that the final product meets the needs of the client.
- **Vector Based Design**: This involves creating digital art using mathematical formulas to define lines, shapes, and colors.
- **Prototyping**: This refers to the process of creating an interactive model of a product or user interface.

UI Design Fundamentals

- **Good Contrast for Background and Foreground Colors**: It is important to ensure that the background and foreground colors have good contrast to make the text readable. The Web Content Accessibility Guidelines (WCAG) recommend a minimum contrast ratio of 4.5:1 for normal text and 3:1 for large text.
- **Good Visual Hierarchy in Design**: A strong visual hierarchy can provide a clear path for the eye to follow, ensuring that the information you convey is consumed in the order that you intend.
- **Responsive Images**: Responsive images are images that scale to fit the size of the screen they are being viewed on. This is important because it ensures that your images look good on all devices, from desktops to mobile phones.
- **Progressive Enhancement**: This is a design approach that ensures all users, regardless of browser or device, can access the essential content and functionality of an application.
- **User-centered Design**: This is an approach that prioritizes the end user, from their needs to their preferences and limitations. The goal of user-centered design is to craft a web page that is intuitive, efficient to use, and pleasing for your users to interact with.
- **User Research**: This is the systematic study of the people who use your product. The goal is to measure user needs, behaviors, and pain points.
- **User Testing**: This refers to the practice of capturing data from users as they interface with your application.
- **A/B Testing**: This is the process of shipping a new feature to a randomly selected subset of your user base. You can then leverage analytics data to determine if the feature is beneficial.
- **User Requirements**: This refers to the stories or rubric that your application needs to follow. User requirements might be defined by user research, or industry standards. They can even be defined by stakeholder input.
- **Progressive Disclosure**: This is a design pattern used to only show users relevant content based on their current activity and hide the rest. This is done to reduce cognitive load and make the user experience more intuitive.

- **Deferred/Lazy Registration**: This is a UI design pattern that allows users to browse and interact with your application without having to register.

Design Best Practices

- **Dark Mode**: This is a special feature on web applications where you can change the default light color scheme to a dark color scheme. You should use desaturated colors in dark mode. Desaturated colors are colors that are less intense and have a lower saturation level.
- **Breadcrumbs**: This is a navigation aid that shows the user where they are in the site's hierarchy. It is best to place breadcrumbs at the top of the page so users can easily find it. Also, you want to make sure the breadcrumbs are large enough to be easily read, but not so large that they take up too much space on the page.
- **Card Component**: Your card component should be simple in design, not visually cluttered or display too much information. For media, make sure to choose high-quality images and videos to enhance the user experience.
- **Infinite Scroll**: This is a design pattern that loads more content as the user scrolls down the page. You should consider using a load more button because it gives a user control over when they want to see more content. You can also add a back button so users have the ability to go back to the previous page without having to scroll all the way back up.
- **Modal Dialog**: This is a type of pop-up that will display on top of existing page content. Usually the background content will have a dim color overlay in order to help the user better focus on the modal content. Also, it is always a good idea to allow the user to click outside of the modal to close it. When you use the HTML `dialog` element, you will get a lot of the functionality and accessibility benefits built in.
- **Progress Indication for Form Registration**: This is a way to show users how far they are in a process. It can be used in forms, registration, and setup processes. Your design should be simple, easy to find, and make it possible to go back to previous steps.
- **Shopping Cart**: Carts are a place for user to see what item they have already selected on an e-commerce platform. Your carts should always be visible to the user, use a common icon like a cart, bag or basket, and have a clear call-to-action button for users to proceed to checkout.

Common Design Tools

- **Figma**: This cloud-based tool specializes in User Interface and User Experience (UI / UX) design. It enables design and development teams to collaborate from anywhere, offering built-in features including Vector-based design, automatic layout, a commenting and feedback system and more.
- **Sketch**: This is a popular design tool used for its intuitive interface and simplicity, making it ideal for developers who want to quickly create prototypes. It's also widely used by designers for tasks like creating UIs, icons, and web layouts.
- **Adobe XD**: This is a vector-based design and prototyping tool for UI/UX design, known for its seamless integration with other Adobe apps like Photoshop, Illustrator, and After Effects.
- **Canva**: This tool allows you to create a wide range of visual content, including posters, cover photos, presentations, short videos, and more. Its user-friendly and simple design makes it ideal for beginners.

Absolute Units

- **`px` (Pixels)**: This absolute unit is a fixed-size unit of measurement in CSS, providing precise control over dimensions. This means that 1px is always equal to 1/96th of an inch.
- **`in` (Inch)**: This absolute unit is equal to 96px.
- **`cm` (Centimeters)**: This absolute unit is equal to 25.2/64 of an inch.
- **`mm` (Millimeters)**: This absolute unit is equal to 1/10th of a centimeter.
- **`q` (Quarter-Millimeters)**: This absolute unit is equal to 1/40th of a centimeter.
- **`pc` (Picas)**: This absolute unit is equal to 1/6th of an inch.
- **`pt` (Points)**: This absolute unit is equal to 1/72th of an inch.

Relative Units

- **Percentages**: These relative units allow you to define sizes, dimensions, and other properties as a proportion of their parent element. For example, if you set `width: 50%;` on an element, it will occupy half the width of its parent container.
- **`em` Unit**: These units are relative to the font size of the element. If you are using `ems` for the `font-size` property, the size of the text will be relative to the font size of the parent element.
- **`rem` Unit**: These units are relative to the font size of the root element, which is the `html` element.

- **`vh` Unit**: `vh` stands for `"viewport height"` and `1vh` is equal to 1% of the viewport's height.
- **`vw` Unit**: `vw` stands for `"viewport width"` and `1vw` is equal to 1% of the viewport's width.

`calc` Function

- **`calc()` Function**: With the `calc()` function, you can perform calculations directly within your stylesheets to determine property values dynamically. This means that you can create flexible and responsive user interfaces by calculating dimensions based on the viewport size or other elements.

User Action Pseudo-classes

- **Pseudo-classes Definition**: These are special CSS keywords that allow you to select an element based on its specific state or position.
- **User Action Pseudo-classes**: These are special keywords that allow you to change the appearance of elements based on user interactions, improving the overall user experience.
- **`:active` Pseudo-class**: This pseudo-class lets you select the active state of an element, like clicking on a button.
- **`:hover` Pseudo-class**: This pseudo-class defines the hover state of an element.
- **`:focus` Pseudo-class**: This pseudo-class applies styles when an element gains focus, typically through keyboard navigation or when a user clicks into a form input.
- **`:focus-within` Pseudo-class**: This pseudo-class is used to apply styles to an element when it or any of its descendants have focus.

Input Pseudo-classes

- **Input Pseudo-classes**: These pseudo-classes are used to target HTML `input` elements based on the state they are in before and after user interaction.
- **`:enabled` Pseudo-class**: This pseudo-class is used to target form buttons or other elements that are currently enabled.
- **`:disabled` Pseudo-class**: This pseudo-class lets you style an interactive element in disabled mode.

- `:checked` **Pseudo-class**: This pseudo-class is used to indicate to the user that it is checked.
- `:valid` **Pseudo-class**: This pseudo-class targets the input fields that meet the validation criteria.
- `:invalid` **Pseudo-class**: This pseudo-class targets the input fields that do not meet the validation criteria.
- `:in-range` **and** `:out-of-range` **Pseudo-classes**: These pseudo-classes apply styles to elements based on whether their values are within or outside specified range constraints.
- `:required` **Pseudo-class**: This pseudo-class targets `input` elements that have the `required` attribute. It signals to the user that they must fill out the field to submit the form.
- `:optional` **Pseudo-class**: This pseudo-class applies styles to input elements that are not required and can be left empty.
- `:autofill` **Pseudo-class**: This pseudo-class applies styles to input fields that the browser automatically fills with saved data.

Location Pseudo-classes

- **Location Pseudo-classes**: These pseudo-classes are used for styling links and elements that are targeted within the current document.
- `:any-link` **Pseudo-class**: This pseudo-class is a combination of the `:link` and `:visited` pseudo-classes. So, it matches any anchor element with an href attribute, regardless of whether it's visited or not.
- `:link` **Pseudo-class**: This pseudo-class allows you to target all unvisited links on a webpage. You can use it to style links differently before the user clicks on them.
- `:local-link` **Pseudo-class**: This pseudo-class targets links that point to the same document. It can be useful when you want to differentiate internal links from external ones.
- `:visited` **Pseudo-class**: This pseudo-class targets a link the user has visited.
- `:target` **Pseudo-class**: This pseudo-class is used to apply styles to an element that is the target of a URL fragment

Tree-structural Pseudo-classes

- **Tree-structural Pseudo-classes**: These pseudo-classes allow you to target and style elements based on their position within the document tree.

- `:root` **Pseudo-class**: This pseudo-class is usually the root `html` element. It helps you target the highest level in the document so you can apply a common style to the entire document.
- `:empty` **Pseudo-class**: Empty elements, that is, elements with no children other than white space, are also included in the document tree. That's why there's an `:empty` pseudo-class to target empty elements.
- `:nth-child(n)` **Pseudo-class**: This pseudo-class allows you to select elements based on their position within a parent.
- `:nth-last-child(n)` **Pseudo-class**: This pseudo-class enables you to select elements by counting from the end.
- `:first-child` **Pseudo-class**: This pseudo-class selects the first element in a parent element or the document.
- `:last-child` **Pseudo-class**: This pseudo-class selects the last element in a parent element or the document
- `:only-child` **Pseudo-class**: This pseudo-class selects the only element in a parent element or the document
- `:first-of-type` **Pseudo-class**: This pseudo-class selects the first occurrence of a specific element type within its parent.
- `:last-of-type` **Pseudo-class**: This pseudo-class selects the last occurrence of a specific element type within its parent.
- `:nth-of-type(n)` **Pseudo-class**: This pseudo-class allows you to select a specific element within its parent based on its position among siblings of the same type.
- `:only-of-type` **Pseudo-class**: This pseudo-class selects an element if it's the only one of its type within its parent.

Functional Pseudo-classes

- **Functional Pseudo-classes**: Functional pseudo-classes allow you to select elements based on more complex conditions or relationships. Unlike regular pseudo-classes which target elements based on a state (for example, `:hover`, `:focus`), functional pseudo-classes accept arguments.
- `:is()` **Pseudo-class**: This pseudo-class takes a list of selectors (ex. `ol`, `ul`) and selects an element that matches one of the selectors in the list.
- `:where()` **Pseudo-class**: This pseudo-class takes a list of selectors (ex. `ol`, `ul`) and selects an element that matches one of the selectors in the list. The difference between `:is` and `:where` is that the latter will have a specificity of 0.

- `:has()` **Pseudo-class**: This pseudo-class is often dubbed the `"parent"` selector because it allows you to style elements who contain child elements specified in the selector list.

Pseudo-elements

- `::before` **Pseudo-element**: This pseudo-element uses the `content` property to insert cosmetic content like icons just before the element.
- `::after` **Pseudo-element**: This pseudo-element uses the `content` property to insert cosmetic content like icons just after the element.
- `::first-letter` **Pseudo-element**: This pseudo-element targets the first letter of an element's content, allowing you to style it.
- `::marker` **Pseudo-element**: This pseudo-element lets you select the marker (bullet or numbering) of list items for styling.

Color Theory

- **Color Theory Definition**: This is the study of how colors interact with each other and how they affect our perception. It covers color relationships, color harmony, and the psychological impact of color.
- **Primary Colors**: These colors which are yellow, blue, and red, are the fundamental hues from which all other colors are derived.
- **Secondary Colors**: These colors result from mixing equal amounts of two primary colors. Green, orange, and purple are examples of secondary colors.
- **Tertiary Colors**: These colors result from combining a primary color with a neighboring secondary color. Yellow-Green, Blue-Green, and Blue-Violet, are examples of tertiary colors.
- **Warm Colors**: These colors which include reds, oranges, and yellows, evoke feelings of comfort, warmth, and coziness.
- **Cool Colors**: These colors which include blues, green, and purples, evoke feelings of calmness, serenity, and professionalism.
- **Color Wheel**: The color wheel is a circular diagram that shows how colors relate to each other. It's an essential tool for designers because it helps them to select color combinations.
- **Analogous Color Schemes**: These color schemes create cohesive and soothing experiences. They have analogous colors, which are adjacent to each other in the color wheel.
- **Complementary Color Schemes**: These color schemes create high contrast and visual impact. Their colors are located on the opposite ends of the color wheel, relative to each other.

- **Triadic Color Scheme**: This color scheme has vibrant colors. They are made from colors that are approximately equidistant from each other. If they are connected, they form an equilateral triangle on the color wheel.
- **Monochromatic Color Scheme**: For this color scheme, all the colors are derived from the same base color by adjusting its lightness, darkness, and saturation. This evokes a feeling of unity and harmony while still creating contrast.

Different Ways to Work with Colors in CSS

- **Named Colors**: These colors are predefined color names recognized by browsers. Examples include `blue`, `darkred`, `lightgreen`.
- **`rgb()` Function**: RGB stands for Red, Green, and Blue — the primary colors of light. These three colors are combined in different intensities to create a wide range of colors. the `rgb()` function allows you to define colors using the RGB color model.
- **`rgba()` Function**: This function adds a fourth value —alpha— that controls the transparency of the color.
- **`hsl()` Function**: HSL stands for Hue, Saturation, and Lightness — three key components that define a color.
- **`hsla()` Function**: This function adds a fourth value -alpha- that controls the opacity of the color.
- **Hexadecimal**: A hex code (short for hexadecimal code) is a six-character string used to represent colors in the RGB color model. The "hex" refers to the base-16 numbering system, which uses digits 0 to 9 and letters A to F.

Linear and Radial Gradients

- **Linear Gradients**: These gradients create a gradual blend between colors along a straight line. You can control the direction of this line and the colors used.
- **Radial Gradients**: These gradients create circular or elliptical gradients that radiate from a central point.

Best Practices for Styling Inputs

- **Styling Inputs**: As with all text elements, you need to ensure the styles you apply to text inputs are accessible. This means the font needs to be adequately sized, and the color needs to have sufficient contrast with the background. Input elements are also focusable. When you are editing your styles, you should take care that you preserve a noticeable indicator when the element has focus, such as a bold border.

Using `appearance: none` for Inputs

- **`appearance: none`**: Browsers apply default styling to a lot of elements. The `appearance: none` CSS property gives you complete control over the styling, but comes with some caveats. When building custom styles for input elements, you will need to make sure focus and error indicators are still present.

Common Issues Styling `datetime-local` and `color` Properties

- **Common Issues**: These special types of inputs rely on complex pseudo-elements to create things like the date and color pickers. This presents a significant challenge for styling these inputs. One challenge is that the default styling is entirely browser-dependent, so the CSS you write to make the picker look the way you intend may be entirely different on another browser.

CSS Overflow Property

- **Definition**: Overflow refers to the way elements handle content that exceeds, or "overflows", the size of the containing element. Overflow is two-dimensional.
- **`overflow-x`**: The x-axis determines horizontal overflow.
- **`overflow-y`**: the y-axis determines vertical overflow.

CSS Transform Property

- **Definition**: This property enables you to apply various transformations to elements, such as rotating, scaling, skewing, or translating (moving) them in 2D or 3D space.
- **`translate()` Function**: This function is used to move an element from its current position.
- **`scale()` function**: This function allows you to change the size of an element.
- **Transforms and Accessibility**: If you're using transform to hide or reveal content, make sure that the content is still accessible to screen readers and

keyboard navigation. Hidden content should be truly hidden, such as by using `display: none` or `visibility: hidden`, rather than simply being visually moved off-screen.

The Box Model

- **Definition**: In the CSS box model, every element is surrounded by a box. This box consists of four components: the content area, `padding`, `border`, `margin`.
- **Content Area**: The content area is the innermost part of the box. It's the space that contains the actual content of an element, like text or images.
- **`padding`**: The padding is the area immediately after the content area. It's the space between the content area and the border of an element.
- **`border`**: The border is the outer edge or outline of an element in the CSS box model. It's the visual boundary of the element.
- **`margin`**: The margin is the space outside the border of an element. It determines the distance between an element and other elements around it.

Margin Collapse

- **Definition**: This behavior occurs when the vertical margins of adjacent elements overlap, resulting in a single margin equal to the larger of the two. This behavior applies only to vertical margins (top and bottom), not horizontal margins (left and right).

The `content-box` and `border-box` Property Values

- **`box-sizing` Property**: This property is used to determine how the final `width` and `height` are calculated for an HTML element.
- **`content-box` Value**: In the `content-box` model, the `width` and `height` that you set for an element determine the dimensions of the content area but they don't include the `padding`, `border`, or `margin`.
- **`border-box` Value**: With `border-box`, the `width` and `height` of an element include the content area, the `padding`, and the `border`, but they don't include the `margin`.

CSS Reset

- **Definition**: A CSS reset is a stylesheet that removes all or some of the default formatting that web browsers apply to HTML elements. Third party options for CSS resets include `sanitize.css` and `normalize.css`.

CSS Filter Property

- **Definition**: This property can be used to create various effects such as blurring, color shifting, and contrast adjustment.
- **`blur()` Function**: This function applies a Gaussian blur to the element. The amount is defined in pixels and represents the radius of the blur.
- **`brightness()` Function**: This function adjusts the brightness of the element. A value of 0% will make the element completely black, while values over 100% will increase the brightness.
- **`grayscale()` Function**: This function converts the element to grayscale. The amount is defined as a percentage, where 100% is completely grayscale and 0% leaves the image unchanged.
- **`sepia()` Function**: This function applies a sepia tone to the element. Like grayscale, it uses a percentage value.
- **`hue-rotate()` Function**: This function applies a hue rotation to the element. The value is defined in degrees and represents a rotation around the color circle.

Introduction to CSS Flexbox and Flex Model

- **Definition**: CSS flexbox is a one-dimensional layout model that allows you to arrange elements in rows and columns within a container.
- **Flex Model**: This model defines how flex items are arranged within a flex container. Every flex container has two axes: the main axis and the cross axis.

The `flex-direction` Property

- **Definition**: This property sets the direction of the main axis. The default value of `flex-direction` is `row`, which places all the flex items on the same row, in the direction of your browser's default language (left to right or right to left).
- **`flex-direction: row-reverse;`**: This reverses the items in the row.
- **`flex-direction: column;`**: This will align the flex items vertically instead of horizontally.
- **`flex-direction: column-reverse;`**: This reverses the order of the flex items vertically.

The `flex-wrap` Property

- **Definition**: This property determines how flex items are wrapped within a flex container to fit the available space. `flex-wrap` can take three possible values: `nowrap`, `wrap`, and `wrap-reverse`.
- **`flex-wrap: nowrap;`**: This is the default value. Flex items won't be wrapped onto a new line, even if their width exceed the container's width.
- **`flex-wrap: wrap;`**: This property will wrap the items when they exceed the width of their container.
- **`flex-wrap: wrap-reverse;`**: This property will wrap flex items in reverse order.
- **`flex-flow` Property**: This property is a shorthand property for `flex-direction` and `flex-wrap`.

The `justify-content` Property

- **Definition**: This property aligns the child elements along the main axis of the flex container.
- **`justify-content: flex-start;`**: In this case, the flex items will be aligned to the start of the main axis. This could be horizontal or vertical.
- **`justify-content: flex-end;`**: In this case, the flex items are aligned to the end of the main axis, horizontally or vertically.
- **`justify-content: center;`**: This centers the flex items along the main axis.
- **`justify-content: space-between;`**: This will distribute the elements evenly along the main axis.
- **`justify-content: space-around;`**: This will distribute flex items evenly within the main axis, adding a space before the first item and after the last item.
- **`justify-content: space-evenly;`**: This will distributes the items evenly along the main axis.

The `align-items` Property

- **Definition**: This property is used to distribute items along the cross axis. Remember that the cross axis is perpendicular to the main axis.
- **`align-items: center;`**: This is used to center the items along the cross axis.

- `align-items: flex-start;`: This aligns the items to the start of the cross axis.
- `align-items: stretch;`: This is used to stretch the flex items along the cross axis.

Introduction to Typography

- **Definition**: Typography is the art of choosing the right fonts and format to make text visually appealing and easy to read. "Type" refers to how the individual characters are designed and arranged.
- **Typeface Definition**: A typeface is the overall design and style of a set of characters, numbers, and symbols. It's like a blueprint for a family of fonts.
- **Font Definition**: A font is a specific variation of a typeface with specific characteristics, such as size, weight, style, and width.
- **Serif Typeface**: This typeface has a classical style with small lines at the end of characters. Serif typefaces are commonly used for printed materials, like books.
- **Sans Serif Typeface**: This typeface has a more modern look, without the small lines at the end of characters. Sans Serif typefaces are commonly used in digital design because they are easy to read on screen. Some examples include Helvetica, Arial, and Roboto.
- **Font Weight**: This is the thickness of the characters, including light, regular, bold, and black.
- **Font Style**: This is the slant and orientation of the characters, like italic and oblique.
- **Font Width**: This is the horizontal space occupied by characters, like condensed and expanded.
- **Baseline**: This is the imaginary line on which most characters rest.
- **Cap Height**: This is the height of uppercase letters, measured from the baseline to the top.
- **X-height**: This is the average height of lowercase letters, excluding ascenders and descenders.
- **Ascenders**: These are the parts of lowercase letters that extend above the x-height, such as the tops of the letters 'h', 'b', 'd', and 'f'.
- **Descenders**: These are the parts of lowercase letters that extend below the baseline, such as the tails of 'y', 'g', 'p', and 'q'.
- **Kerning**: This is how space is adjusted between specific pairs of characters to improve their readability and aesthetics. For example, reducing the space between the letters A and V.

- **Tracking**: This is how space is adjusted between all characters in a block of text. It's essentially the distance between the characters. It affects how dense and open the text will be.
- **Leading**: This is the vertical space between lines of text. It's measured from the baseline of one line to the baseline of the next line.
- **Best Practices with Typography**: You should choose clear and simple fonts to make your designs easy to understand. This is particularly important for the main text of your website. Users are more likely to engage with your content if the font is easy to read. You should use two or three fonts at most to create a visual consistency. Using too many fonts can make the text more difficult to read and weaken your brand's identity. This can also impact the user experience by increasing the load time of the website. You can use font size to create a visual hierarchy for headings, subheadings, paragraphs, and other elements. For example, the main heading on a webpage should have a larger font, followed by subheadings with smaller font sizes. This will give every element in the hierarchy a specific font size that helps users navigate through the structure visually.

CSS `font-family` Property

- **Definition**: A font family is a group of fonts that share a common design. All the fonts that belong to the same family are based on the same core typeface but they also have variations in their style, weight, and width. You can specify multiple font families in order of priority, from highest to lowest, by separating them with commas. These alternative fonts will act as fallback options. You should always include a generic font family at the end of the font-family list.
- **Common Font Families**: Here are some common font families used in CSS: serif, sans-serif, monospace, cursive, fantasy

Web Safe Fonts

- **Definition**: These fonts are a subset of fonts that are very likely to be installed on a computer or device. When the browser has to render a font, it tries to find the font file on the user's system. But if the font is not found, it will usually fall back to a default system font.
- **Web Safe Fonts**:
- Sans-serif fonts are commonly used for web development because they don't have small "feet" or lines at the end of the characters, so they're easy to read on screen. Some examples of web-safe sans-serif fonts are: Arial, Verdana, and Trebuchet MS.

- In contrast, serif fonts do have small "feet" at the end of the characters, so they're commonly used for traditional print. Web safe serif fonts include: Times New Roman and Georgia.

At-rules and the `@font-face` At-rule

- **Definition**: At-rules are statements that provide instructions to the browser. You can use them to define various aspects of the stylesheet, such as media queries, keyframes, font faces, and more.
- **`@font-face`**: This allows you to define a custom font by specifying the font file, format, and other important properties, like weight and style. For the `@font-face` at-rule to be valid, you also need to specify the `src` property. This property contains references to the font resources.
- **Font Formats**: For each font resource, you can also specify the format. This is optional. It's a hint for the browser on the font format. If the format is omitted, the resource will be downloaded and the format will be detected after it's downloaded. If the format is invalid, the resource will not be downloaded. Possible font formats include `collection`, `embedded-opentype`, `opentype`, `svg`, `truetype`, `woff`, and `woff2`.
- **`woff` and `woff2`**: `woff` stands for "Web Open Font Format." It's a widely used font format developed by Mozilla in collaboration with Type Supply, LettError, and other organizations. The difference between `woff` and `woff2` is the algorithm used to compress the data.
- **OpenType**: This is a format for scalable computer fonts developed by Microsoft and Adobe that allows users to access additional features in a font. It's widely used across major operating systems.
- **`tech()`**: This is used to specify the technology of the font resource. This is optional.

Working With External Fonts

- **Definition**: An external font is a font file that is not included directly within your project files. They're usually hosted on a separate server. To include these external fonts inside your project, you can use a `link` element or `@import` statement inside your CSS.
- **Google Fonts**: This is a Google service that offers a collection of fonts, many of which are designed specifically for web development.
- **Font Squirrel**: This is another popular resource that you can use for adding custom external fonts to your projects.

`text-shadow` Property

- **Definition**: This property is used to apply shadows to text. You need to specify the X and Y offset, which represent the horizontal and vertical distance of the shadow from the text, respectively. These values are required. Positive values of the X offset and Y offset will move the shadow right and down, respectively, while negative values will move the shadow left and up.
- **Shadow Color**: You can also customize the color of the shadow by specifying this value before or after the offset. If the color is not specified, the browser will determine the color automatically, so it's usually best to set it explicitly.
- **Blur Radius**: The blur radius is optional but will make the shadow look a lot smoother and more subtle. The default value of the radius blur is zero. The higher the value, the bigger the blur, which means that the shadow becomes lighter.
- **Applying Multiple Text Shadows**: The text can have more than one shadow. The shadows will be applied in layers, from front to back, with the first shadow at the top.

Color Contrast Tools

- **WebAIM's Color Contrast Checker**: This online tool allows you to input the foreground and background colors of your design and instantly see if they meet the Web Content Accessibility Guidelines (WCAG) standards.
- **TPGi Colour Contrast Analyzer**: This is a free color contrast checker that allows you to check if your websites and apps meet the Web Content Accessibility Guidelines (WCAG) standards. This tool also comes with a Color Blindness Simulator feature which allows you to see what your website or app looks like for people with color vision issues.

Accessibility Tree

Accessibility tree is a structure used by assistive technologies, such as screen readers, to interpret and interact with the content on a webpage.

`max()` Function

The `max()` function returns the largest of a set of comma-separated values:

Example Code

```
img {
```

```css
  width: max(250px, 25vw);
}
```

In the above example, the width of the image will be 250px if the viewport width is less than 1000 pixels. If the viewport width is greater than 1000 pixels, the width of the image will be 25vw. This is because 25vw is equal to 25% of the viewport width.

Best Practices with CSS and Accessibility

- **`display: none;`**: Using `display: none;` means that screen readers and other assistive technologies won't be able to access this content, as it is not included in the accessibility tree. Therefore, it is important to use this method only when you want to completely remove content from both visual presentation and accessibility.
- **`visibility: hidden;`**: This property and value hides the content visually but keeps it in the document flow, meaning it still occupies space on the page. These elements will also no longer be read by screen readers because they will have been removed from the accessibility tree.
- **`.sr-only` CSS class**: This is a common technique used to visually hide content while keeping it accessible to screen readers.

Example Code

```css
.sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  white-space: nowrap;
  border: 0;
}
```

- **`scroll-behavior: smooth;`**: This property and value enables a smooth scrolling behavior.

- `prefers-reduced-motion` **feature**: This is a media feature that can be used to detect the user's animation preference.

Example Code

```
@media (prefers-reduced-motion: no-preference) {
  * {
    scroll-behavior: smooth;
  }
}
```

In the above example, smooth scrolling is enabled if the user doesn't have animation preference set on their device.

Hiding Content with HTML Attributes

- `aria-hidden` **attribute**: Used to hide an element from people using assistive technology such as screen readers. For example, this can be used to hide decorative images that do not provide any meaningful content.
- `hidden` **attribute**: This attribute is supported by most modern browsers and hides content both visually and from the accessibility tree. It can be easily toggled with JavaScript.

Example Code

```
<p aria-hidden>This paragraph is visible for sighted users, but is hidden from assistive technology.</p>

<p hidden>This paragraph is hidden from both sighted users and assistive technology.</p>
```

Accessibility Issue of the `placeholder` Attribute

Using placeholder text is not great for accessibility. Too often, users confuse the placeholder text with an actual input value - they think there is already a value in the input.

Working with Different Attribute Selectors and Links

- **Definition**: The `attribute` selector allows you to target HTML elements based on their attributes like the `href` or `title` attributes.

- **`title` Attribute**: This attribute provides additional information about an element.

Targeting Elements with the `lang` and `data-lang` Attribute

- **`lang` Attribute**: This attribute is used in HTML to specify the language of the content within an element. You might want to style elements differently based on the language they are written in, especially on a multilingual website.
- **`data-lang` Attribute**: Custom data attributes like the `data-lang` attribute are commonly used to store additional information in elements, such as specifying the language used within a specific section of text.

Working with the Attribute Selector, Ordered List Elements and the `type` Attribute

- **`type` Attribute**: When working with ordered lists in HTML, the `type` attribute allows you to specify the style of numbering used, such as numerical, alphabetical, or Roman numerals.

Working With Floats

- **Definition**: Floats are used to remove an element from its normal flow on the page and position it either on the left or right side of its container. When this happens, text will wrap around that floated content.
- **Clearing Floats**: The `clear` property is used to determine if an element needs to be moved below the floated content. When you have multiple floated elements stacked next to each other, there could be issues with overlap and collapsing in the layouts. So a `clearfix` hack was created to fix this issue.

Static, Relative and Absolute Positioning

- **Static Positioning**: This is the normal flow for the document. Elements are positioned from top to bottom, and left to right one after another.
- **Relative Positioning**: This allows you to use the `top`, `left`, `right` and `bottom` properties to position the element within the normal document flow. You can also use relative positioning to make elements overlap with other elements on the page.

- **Absolute Positioning**: This allows you to take an element out of the normal document flow, making it behave independently from other elements.

Fixed and Sticky Positioning

- **Fixed Positioning**: When an element is positioned with `position: fixed`, it is removed from the normal document flow and placed relative to the viewport, meaning it stays in the same position even when the user scrolls. This is often used for elements like headers or navigation bars that need to remain visible at all times.
- **Sticky Positioning**: This type of positioning will act as an relative positioned element as you scroll down the page. If you specify a `top`, `left`, `right` or `bottom` property, then the element will stop acting like a relatively positioned element and start behaving like a fixed position element.

Working With the `z-index` Property

- **Definition**: The `z-index` property in CSS is used to control the vertical stacking order of positioned elements that overlap on the page.

Responsive Web Design

- **Definition**: The core principle of responsive design is adaptability – the ability of a website to adjust its layout and content based on the screen size and capabilities of the device it's being viewed on.
- **Fluid grids**: These use relative units like percentages instead of fixed units like pixels, allowing content to resize and reflow based on screen size.
- **Flexible images**: These are set to resize within their containing elements, ensuring they don't overflow their containers on smaller screens.

Media Queries

- **Definition**: This allows developers to apply different styles based on the characteristics of the device, primarily the viewport width.
- **`all` Media Type**: This is suitable for all devices. This is the default if no media type is specified.
- **`print` Media Types**: This is intended for paged material and documents viewed on a screen in print preview mode.

- `screen` **Media Types**: This is intended primarily for screens.
- `aspect-ratio`: This describes the ratio between the width and height of the viewport.
- `orientation`: This is used to indicate whether the device is in landscape or portrait orientation.
- `resolution`: This is used to describe the resolution of the output device in dots per inch (dpi) or dots per centimeter (dpcm).
- `hover`: This is used to test whether the primary input mechanism can hover over elements.
- `prefers-color-scheme`: This is used to detect if the user has requested a light or dark color theme.
- **Media Queries and Logical Operators**: The `and` operator is used to combine multiple media features, while `not` and `only` can be used to negate or isolate media queries.

Common Media Breakpoints

- **Definition**: Media breakpoints are specific points in a website's design where the layout and content adjust to accommodate different screen sizes. There are some general breakpoints that you can use to target phones, tablets and desktop screens. But it is not wise to try to chase down every single possible screen size for different devices.
- **Small Devices (smartphones)**: up to 640px
- **Medium Devices (tablets)**: 641px to 1024px
- **Large Devices (desktops)**: 1025px and larger

Mobile first approach

- **Definition**: The `mobile-first` approach is a design philosophy and development strategy in responsive web design that prioritizes creating websites for mobile devices before designing for larger screens.

CSS Custom Properties (CSS Variables)

- **Definition**: CSS custom properties, also known as CSS variables, are entities defined by CSS authors that contain specific values to be reused throughout a document. They are a powerful feature that allows for more efficient, maintainable, and flexible stylesheets. Custom properties are particularly useful in creating themeable designs. You can define a set of properties for different themes.

The `@property` Rule

- **Definition**: The `@property` rule is a powerful CSS feature that allows developers to define custom properties with greater control over their behavior, including how they animate and their initial values.

Example Code

```css
@property --property-name {
  syntax: '<type>';
  inherits: true | false;
  initial-value: <value>;
}
```

- **`--property-name`**: This is the name of the custom property you're defining. Like all custom properties, it must start with two dashes. `--property-name` can be things like `<color>`, `<length>`, `<number>`, `<percentage>`, or more complex types.
- **`syntax`**: This defines the type of the property.
- **`inherits`**: This specifies whether the property should inherit its value from its parent element.
- **`initial-value`**: This sets the default value of the property.
- **Fallbacks**: When using the custom property, you can provide a fallback value using the `var()` function, just as you would with standard custom properties.

CSS Grid Basics

- **Definition**: CSS Grid is a two-dimensional layout system used to create complex layouts in web pages. Grids will have rows and columns with gaps between them. To define a grid layout, you would set the `display` property to `grid`.
- **The `fr` (Fractional) Unit**: This unit represents a fraction of the space within the grid container. You can use the `fr` unit to create flexible grids.
- **Creating Gaps Between Tracks**: There are three ways to create gaps between tracks in CSS grid. You can use the `column-gap` property to create

gaps between columns. You can use the `row-gap` property to create gaps between rows. Or you can use the `gap` shorthand property to create gaps between both rows and columns.

- **`repeat()` Function**: This function is used to repeat sections in the track listing. Instead of writing `grid-template-columns: 1fr 1fr 1fr;` you can use the `repeat()` function instead.
- **Explicit Grid**: You can specify the number of lines or tracks using the `grid-template-columns` or `grid-template-rows` properties.
- **Implicit Grid**: When items are placed outside of the grid, then rows and columns are automatically created for those outside elements.
- **`minmax()` Function**: This function is used to set the minimum and maximum sizes for a track.
- **Line-based Placement**: All grids have lines. To specify where the item begins on a line, you can use the `grid-column-start` and `grid-row-start` properties. To specify where the item ends on the line, you can use the `grid-column-end` and `grid-row-end` properties. You can also choose to use the `grid-column` or `grid-row` shorthand properties instead.
- **`grid-template-areas`**: This property is used to provide a name for the items you are going to position on the grid.

CSS Animation Basics

- **Definition**: CSS animations allow you to create dynamic, visually engaging effects on web pages without the need for JavaScript or complex programming. They provide a way to smoothly transition elements between different styles over a specified duration.
- **The `@keyframes` Rule**: This rule defines the stages and styles of the animation. It specifies what styles the element should have at various points during the animation.
- **`animation` Property**: This is the shorthand property used to apply animations.
- **`animation-name`**: This specifies the name for the `@keyframes` rule to use.
- **`animation-duration`**: This sets how long the animation should take to complete.
- **`animation-timing-function`**: This defines how the animation progresses over time (such as ease, linear, ease-in-out).
- **`animation-delay`**: This specifies a delay before the animation starts.

- `animation-iteration-count`: This sets how many times the animation should repeat.
- `animation-direction`: This determines whether the animation should play forwards, backwards, or alternate.
- `animation-fill-mode`: This specifies how the element should be styled before and after the animation.
- `animation-play-state`: This allows you to pause and resume the animation.

Accessibility and the `prefers-reduced-motion` Media Query

- **The `prefers-reduced-motion` Media Query**: One of the primary accessibility concerns with animations is that they can cause discomfort or even physical harm to some users. People with vestibular disorders or motion sensitivity may experience dizziness, nausea, or headaches when exposed to certain types of movement on screen. The `prefers-reduced-motion` media query allows web developers to detect if the user has requested minimal animations or motion effects at the system level