Short Answers

1.What is the Box Model? Describe each part

The model model refers to the layered structure in HTML elements that is also directly related to how it looks, or its CSS. The idea is that everything is box-shaped, as in a square or rectagle. The outer layer is the margin, which is invisible even if you give the element color--it's like white spacing spacing. The next layer in is the border, which is visible if we set it to be something like 1px solid black. The next layer in is the padding, which is visible if we give it color. Then the most inner layer is the actual content. Each layer takes up space, forming the entire overall dimensions of the "box."

2. What is the difference between margin and padding? Can padding be negative? Can margin be negative?

Margin is the outer most layer in the Box Model, outside the border, while padding is the layer inside the border (and directly outside the content area). If you were in a futuristic armor suit, padding is like the insides of the suit, that sticks to your body (making it nice and soft), while margin would be like it's invisible outer force field that prevents others from getting too close to you (and the border would be the suit itself). If you turned off the margin by setting it to 0, others would be able to get close enough to touch your suit. Padding cannot be negative, but margin can. If two elements stacked on top of each other had positive margins, such as 20px and 30px, the spacing between the two would take the higher of the two, giving the total margin between the two 30 px (not 50 px). If one margin was negative and another positive, such as -20px and 30px, it would add them together, giving it a total margin of 10px apart. However, both elements still retain their own full margins. This only applies to sibling elements that are on top of each other (vertical), not next to each other (horizontal). This is called margin collapsing, and it does not apply to padding.

3. Assume an element has the following properties. What is the element size (width & height)? Please explain how you arrived at your solution.

```
width: 100px; height: 100px;
margin: 20px 50px;
padding: 10px 10px;
border: 2px solid black;
```

This element's box-sizing is not set to "border-box," which means we need to add every layer to get the total. The width and height is the inner content by itself, and we would add the padding, border, and margin onto it. The width would be 100px + 10px (* 2) + 2px (* 2) + 50px (* 2), which is 224px. The height would be 100px + 10px (* 2) + 2px (* 2) + 20px (* 2), which is 164px. The element would be 224px by 164px.

4. Assume an element has the following properties. What is the element size (width & height)? Please explain how you arrived at your solution.

```
  width: 100px; height: 100px;
  margin: 20px 50px;
  padding: 10px 10px;
  border: 2px solid black;
  box-sizing: border-box;
```

Because this element's box-sizing is "border-box," it means that the set width and height now include the content, padding, and border. The margin is outside of this. So the width would be 100px + 50px (* 2), or 200px. And the height would be 100px + 20px (* 2), or 140px. The element would be 200px by 140px. Setting it to border-box makes it easier to calculate the overall look and size of an element since it's defined up to and including its borders. In this case, the border is still 2px all around, the padding is still 10px all around, but the actual content area will shrink or grow, making up for the rest of the pixels to make it reach 100px.

5. What is the difference between "position: fixed" and "position: absolute"?

The fixed and absolute positions are used to give an element an offset, which is to move it from it's original position. Both values take the element "out of flow," or in other words, takes it out of the HTML page, allowing other elemnts to fill its original space. The difference though, is that the offset of fixed elements is relative to the viewport. When we start scrolling up/down or left/right, the fixed element will stay in that position. With absolutely placed elements, the offset is relative to its nearest positioned ancestor, or parent element. If there is none, it will be relative to the viewport or HTML document (the outermost `<html>` element). The easiest way to setup an absolutely placed element is to put it in a `<div>`, and give its parent `<div>` the `position: relative`. Now its child element will always be positioned relative to this parent `<div>`.

6. What does z-index do?

Z-index specifies the stacking order of an element, as in which element is placed in front or on top of others, as opposed to under or behind. This only works on positioned elements such as absolute, relative, fixed, or sticky (not static), and flex items. And it should also work with CSS grid items. The higher the z-index number we give an element, the higher or more in front it is. By default every element's z-index is 0, so if we want an element to always be on top, just give it a higher score. If we want an element to go behind, just give it a lower score. We can also just give it a negative z-index.

7. What is Flexbox?

Flexbox is the Flexible Box Layout Module, designed to create flexible, responsive layouts that adjust based on content size or the viewport. It is one-dimensional, focusing on either a single row or column at a time. Flexbox uses a content-first approach, where the layout adapts to the size of its contents. By setting `display: flex` on a parent container, all direct children become flex items. The layout direction can be set with `flex-direction`, which can be either row (default) for a horizontal layout or column for a vertical layout. When using row, the main axis runs horizontally, and `justify-content` aligns items left, center, or right, while `align-items` aligns them vertically (top, center, or bottom). In column direction, the main axis is vertical, so `justify-content` aligns items top, center, or bottom, and `align-items` aligns items horizontally (left, center, or right).

8. What is Grid?

Grid is the Grid Layout Module, which makes it easier to design grid-based layouts using rows and columns, making it a two-dimensional layout system. It uses a layout-first approach, where the grid structure (rows, columns, and gaps) is defined first, and then content is placed into specific grid cells or areas. You define columns and rows with `grid-template-columns` and `grid-template-rows`, and you can specify the size of the gaps between them using `grid-gap`. The grid lines are numbered starting from 1 or -1, and goes up to the number of rows/columns + 1 (for the total length). This allows for precise placement of items. You can give rows, columns, and gaps different width/height proportions, such as fixed sizes, percentages, or flexible values (like fr units). Additionally, you can make individual grid cells flex containers, which allows you to use Flexbox properties within those grid cells.

9. What are the major differences between CSS transitions and animations?

Both CSS transition and animation are used to move an element from an initial state to a final state. But the differences between transition and animation are many. Transitions can only run once, require a trigger to run (such as hover or focus), runs forward when triggered and reverse when the trigger is removed, is easier with JavaScript, and is best for creating a simple change from one state to another. Animations come with intermediate steps in between (such as at 25%, at 50%, at 75%, and at 100%), can loop many times or infinitely (due to its count property), can be triggered or can run automatically when the page loads, can run forwards, in reverse, or in alternating directions, is more difficult to use with JavaScript, and is best for creating a complex series of movements.

10. What is responsive web design (RWD)? What are some examples of RWD on a website? How do we achieve this?

Responsive web design (RWD) is a design appraoch where we use HTML and CSS to make the website UI look great on all devices and screen sizes. It uses a mobile-first approach, which means we design for mobile view first, then tablet, then desktop (go from smallest to biggest). The viewport property in the `<meta>` tag that goes in the head of the HTML document is used to recognize device sizes. We use media queries to set breakpoints at which the layout will be adjusted. For example, between desktop and tablet, we might set the breakdown point at around 800px in width, and between tablet and mobile, we might set a breakdown point at around 600px or 560px. On a full, desktop view, we may want to show everything, including the standard horizontal navigation menu, but on mobile, we may want to hide elements, for example, put the navigation menu in a popup menu itself so it's just a button that doesn't take up too much space, or use Flexbox and set the content direction to column so content flows top down. Usually the height is not as big a concern as the width. These days there are so many different devices (with different widths) that it can certainly be a headache to test for, or to optimize, a website for all of them, so we may just want to start with the popular devices.

11. What is a CSS preprocessor? What are the advantages and disadvantages, if any, to using them over plain CSS?

A CSS preprocessor is a program with its own unique, CSS-based syntax that is transpiled into regular CSS. Most CSS preprocessors' syntax will include features that make the CSS structure more readable and easier to maintain. The main purpose of using a CSS preprocessor is to make writing and managing CSS more efficient, maintainable, and scalable, especially for larger and more complex projects. Some of the most popular CSS proprocessors are SASS/SCSS, LESS, and Stylus. There may be a learning curve when it

comes to using CSS preprocessors, and it usually involves a build or compile step to turn it into regular CSS for browsers to understand.

12. What is the usage of '&' in SCSS?

The `&` character refers to the parent selector in nested CSS blocks, which is useful for pseudo-class/elements. For example, we could set a button, give it a background color, and then use the `&` to give it the pseudo-class hover effect as in the following code:

```
.button {
  background-color: blue;
  &:hover {
    background-color: darkblue;
  }
}
```

13. What is @mixin? What are some use cases?

The `@mixin` keyword defines a reusable block of CSS rules that can be included throughout the stylesheet. It can accept arguments and does not get directly transpiled to CSS. For example, we can create a basic-styles mixin that sets `font-size` and `font-weight`, while allowing the color property to be dynamically defined by passing a value. We can then use this basic-styles mixin in elements like `<h5>` or classes like `.paragraph` (by using the `@include` keyword), passing in a color value (blue or red) as an argument to customize the color of the text in those elements as in the following code:

```
@mixin basic-styles($color) {
  color: $color;
  font-size: 16px;
  font-weight: bold;
}

h5 {
  @include basic-styles(blue);
}

.paragraph {
  @include basic-styles(red);
}
```

14. What is @extend? What are some use cases?

The `@extend` keyword makes a rule inherit the declarations from another rule or placeholder. It can be paired with the placeholder character `%`, which defines a rule that can be reused (extended) throughout the stylesheet. The placeholder does not accept arguments and does not get transpiled into CSS. It can also extend mixins and classes, but it cannot extend IDs or tags. For example, the placeholder `%shared-styles` sets the `color` and `font-weight` properties. The `.heading` class and `<button>` tags are

grouped together (using a comma) and both extend the `%shared-styles` placeholder, which means they both inherit the red color and bold font weight. At the same time, the `.heading` class and `<button>` tag may each have their own styling properties as in the following code:

```
%shared-styles {
  color: red;
  font-weight: bold;
}

.heading,
button {
  @extend %shared-styles;
}

.heading {
  font-size: 24px;
}

button {
  border: 1px solid black;
}
```

15. What is the usage of !important? What are some use cases?

The `!important` rule in CSS is used to prioritize a particular style (property: value). It will override other values of the same property. But it does not override in-line styling. You can override one `!important` with another `!important` on a style property with the same or higher priority. But we should avoid using too many `!important`s because you could be applying styles that could work, but end up not working because somewhere else you set a `!important` and you forgot about it.

16. Could you explain accessibility, usability, and inclusion? Give some examples of each one in terms of web design.

Accessibility means making a website equally accessible to people with physical or mental disabilities. One of the most basic things we can do is set an alt alltribute for people who are blind, allowing a screenreader to read the alt text in place of the image. Other things we can do include making the colors color-friendly to people who are color-blind, or creating better dark/light color contrasts. We probably want to stay away from creating a website that purposefully blinks or flickers too much since that isn't good on anyone's eyes and may cause seizures.

Usability means design that is effective, efficient, and satisfying. This includes user experience design. An example could be to make the "add to shopping cart button" on a website big enough so it can be easily seen and accessible for everyone. A website would not be very "usable" if people didn't know how to sign up on it, navigate it, or purchase products on it. Usability may impact everyone, and not specifically those with disabilities.

Inclusion means universal design, or design for all. It means embracing diversity and ensuring the involvement of everyone to the greatest extent possible. Specifically, this would include people of different

demographics, socioeconomic statuses, languages, physical or mental disabilities, and so on.