Unit-3 CSS

CSS is a styling language used to control the presentation of a document written in HTML or XML. It describes how elements should be rendered on the screen, in print, or in other media.

Syntax:

CSS syntax consists of a selector and a declaration block:

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

Terminology:

- Selector: Specifies the element(s) you want to style.
- Declaration: A property-value pair within curly braces.
- Property: The attribute you want to style (e.g., color, font-size).
- Value: The specific value assigned to a property (e.g., red, 16px).

Naming Conventions:

Use meaningful names: Choose class and ID names that describe the content or function.

Avoid generic names: Be specific to reduce conflicts.

Use dashes (-) or underscores (_) for multi-word names: e.g., .my-class or .my_class.

CSS File Linking:

Link your CSS file to your HTML file using the <link> tag in the <head> section:

Syntax:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type=”text/css” href="styles.css">
  <!-- Other head elements -->
</head>
```

```
<body>
  <!-- Body content -->
</body>
</html>
```

Types of CSS:

1. **Inline CSS:**
   - Applied directly to an HTML element using the style attribute.
   - Overrides external and internal styles.

   Example: This is a paragraph with inline CSS.

2. **Internal or Embedded CSS:**
   - Placed within the <style> tag in the head section of an HTML document.
   - Applies styles to the entire document.

   Example:

```
<head>
  <style>
    p {
      color: red;
      font-size: 18px;
    }
  </style>
</head>
```

3. **External CSS:**
   - Defined in a separate CSS file and linked to an HTML document.
   - Encourages separation of content and presentation.

   Example HTML link:

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

   Example CSS file (styles.css):

```
p {
```

```
  color: green;
  font-size: 20px;
}
```

Types of Selectors:

1. **Element Selector:**

   Syntax: Targets all instances of a specific HTML element.

   Example: Selects all paragraphs and makes the text red.

```
p {
  color: red;
}
```

2. **Class Selector:**

   Syntax: Targets elements with a specific class attribute.
   Example: Applies a style to all elements with the class "my-class."

```
.my-class {
  font-size: 18px;
}
<p class="my-class">This paragraph has a special style.</p>
```

3. **ID Selector:**

   Syntax: Targets a single element with a specific ID attribute.
   Example: Styles the element with the ID "my-id."

```
#my-id {
  background-color: yellow;
}
<div id="my-id">This element has a unique style.</div>
```

4. **Pseudo-selectors:**

   Syntax: Selects elements based on their state or position.
   Examples:

   :hover - Styles an element when the mouse is over it.
   :active - Styles an element being activated (clicked).
   :nth-child - Selects elements based on their position in a parent.

```
li:nth-child(odd) {
background-color: lightgray;
```

```
}
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

Selectors Best Practice:

Keep it simple: Avoid overly complex selectors for better performance.

Specificity: ID selectors are more specific than class selectors.

Order of importance: Inline styles > Internal styles (in the <head>) > External styles.

CSS Comments:

```
/* This is a comment */
selector {
  property: value; /* Inline comment */
}
```

Inheritance in CSS:

Inheritance is a key concept in CSS that defines how properties are passed from parent elements to their child elements within the HTML document tree. This means that if a parent element has a specific style applied, its child elements will inherit that style unless explicitly overridden. However, not all CSS properties are inherited; some must be explicitly set on each element.

Here are some key points about inheritance in CSS:

1. **Inherited Properties:**

   Examples of Inherited Properties:

   color

   font-family

   font-size

   line-height

2. **Non-Inherited Properties:**
   Examples of Non-Inherited Properties:
   - border
   - margin
   - padding
   - background
3. **Controlling Inheritance:**
   Use the inherit keyword to explicitly specify that an element should inherit a value from its parent.

```
.child-element {
  color: inherit; /* Inherits color from the parent */
}
```

4. **Overriding Inherited Properties:**
   You can override inherited properties by applying a new style to a specific element.

```
.child-element {
  color: red; /* Overrides inherited color */
}
```

5. **Universal Selector (*):**
   The universal selector can be used to apply a style to all elements, influencing inheritance.

```
* {
  font-family: Arial, sans-serif; /* Applies to all elements */
}
```

Colors:

Color Property:

```
selector {
  color: red; /* Specifies the text color */
}
```

Background-color Property:

```
selector {
  background-color: #e0e0e0; /* Specifies the background color */
}
```

Backgrounds:

Background-image Property:

```
selector {
  background-image: url('background.jpg'); /* Specifies a background image */
```

}

Background-repeat Property:

selector {

background-repeat: no-repeat; /* Specifies how the background image should repeat */

}

Fonts:

Font-family Property:

selector {

font-family: 'Arial', sans-serif; /* Specifies the font family */

}

Font-size Property:

selector {

font-size: 16px; /* Specifies the font size */

}

Line-height:

Line-height Property:

selector {

line-height: 1.5; /* Specifies the height of a line of text */

}

Text Properties:

Text-align Property:

selector {

text-align: center; /* Specifies the alignment of text */

}

Text-decoration Property:

selector {

text-decoration: underline; /* Specifies the decoration of text (underline, overline, line-through) */ }

Text-transform Property:

```
selector {
  text-transform: uppercase; /* Specifies the capitalization of text */
}
```

Alignment:

Vertical-align Property:

```
selector {
vertical-align: middle; /* Aligns an inline element vertically */
}
```

Margin and Padding for Alignment:

```
selector {
  margin: 10px; /* Adds margin around an element */
  padding: 5px; /* Adds padding inside an element */
}
```

Opacity:

Opacity Property:

```
selector {
opacity: 0.8; /* Specifies the opacity of an element (values between 0 and 1) */
}
```

1. **Borders:**
   a. **Border Property:**
   ```
   selector {
     border: 1px solid #000; /* Specifies the width, style, and color of the border */
   }
   ```
   b. **Border-radius Property:**
   ```
   selector {
     border-radius: 5px; /* Rounds the corners of an element */
   }
   ```
2. **Margin, Padding, Height/Width:**
   a. **Margin Property:**
   ```
   selector {
     margin: 10px; /* Adds space outside an element */
   ```

```
}
```

b. **Padding Property:**

```
selector {
  padding: 5px; /* Adds space inside an element */
}
```

c. **Height/Width Property:**

```
selector {
  height: 100px;
  width: 200px;
}
```

3. **Box Model:**

The box model is comprised of content, padding, border, and margin.

```
div {
      width: 300px;
      border: 15px solid green;
      padding: 50px;
      margin: 20px;
}
```

4. **Outline:**

Outline Property:

```
selector {
outline: 2px dashed #00f; /* Specifies the width, style, and color of an outline */
}
```

5. **Icons:**

Icons can be added using font-based icon libraries or custom image-based icons.

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstr
ap/3.3.7/css/bootstrap.min.css">
</head>
<body>

<i class="glyphicon glyphicon-cloud"></i>
<i class="glyphicon glyphicon-remove"></i>
<i class="glyphicon glyphicon-user"></i>
<i class="glyphicon glyphicon-envelope"></i>
<i class="glyphicon glyphicon-thumbs-up"></i>

</body>
</html>
```

6. **Links:**
   Link Styling:
   ```
   a {
     text-decoration: none; /* Removes underlines from links */
     color: #0066cc; /* Sets link color */
   }
   ```

7. **Lists:**

   List-style Property:

   ```
   ul {

     list-style: square inside; /* Specifies the style, position of list item markers */

   }
   ```

8. **Tables:**
   Table Properties:
   ```
   table {
     border-collapse: collapse; /* Collapses table borders into a single border */
     width: 100%;
   }
   ```

9. **Display:**
   Display Property:
   ```
   selector {
     display: block; /* Changes the default display behavior (block, inline, inline-block) */
   }
   ```

10. **Max-width:**
    Max-width Property:
    ```
    selector {
      max-width: 600px; /* Specifies the maximum width of an element */
    }
    ```

11. **Position:**
    Position Property:
    ```
    selector {
      position: relative; /* Specifies the positioning method (relative, absolute, fixed) */
      top: 10px; /* Adjusts the element's position relative to its normal position */
    }
    ```

12. **Overflow:**
    Overflow Property:
    ```
    selector {
      overflow: hidden; /* Specifies how content should behave when it's larger than its container */
    }
    ```

13. **Float:**
    Float Property:
    ```
    selector {
      float: left; /* Allows an element to be positioned to the left or right of its container */
    }
    ```

14. **Inline-block:**
    Inline-block Property:
    ```
    selector {
      display: inline-block; /* Allows an element to be displayed as an inline-level block container */
    }
    ```

1. Responsive Design:
Responsive design is an approach to web design that makes web pages render well on a variety of devices and window or screen sizes. It ensures a seamless user experience regardless of whether the website is accessed on a desktop, tablet, or mobile device.

2. Mobile-Friendly and Mobile-First:
Mobile-Friendly Design: Ensures that a website looks and functions well on mobile devices. This often involves using responsive design principles.

Mobile-First Design: A design strategy where the initial focus is on designing for mobile devices, and then scaling up for larger screens. It promotes starting with the smallest screen size and progressively enhancing the design for larger screens.

3. Flexible and Fluid Layouts:
Flexible Layouts: Use relative units like percentages for widths, allowing content to adapt to different screen sizes.

Fluid Layouts: Use relative units for widths and heights, creating designs that stretch and shrink with the browser window.

4. Media Queries:
Media queries are CSS rules that apply styles based on the characteristics of the device or browser. They are used to implement responsive design.

css
Copy code
```
@media screen and (max-width: 600px) {
  /* Styles for screens up to 600px wide */
}
```
5. Testing Responsive Design:
Browser Developer Tools: Use browser developer tools to simulate different devices and screen sizes.

Real Devices: Test on actual mobile devices to ensure accuracy.

Online Tools: Use online tools like BrowserStack or Responsinator for cross-browser and device testing.

6. Introduction to CSS Frameworks (Bootstrap, Foundation):
Bootstrap:

Developed by Twitter, Bootstrap is a widely used front-end framework.
It provides a grid system, pre-styled components, and a responsive design structure.
Components like navigation bars, modals, and carousels are readily available.

Foundation:
Developed by Zurb, Foundation is a responsive front-end framework.
It offers a flexible grid system and a variety of UI components.
Foundation allows customization and can be used for mobile-first design.
Key Advantages of CSS Frameworks:
Rapid Development: Frameworks provide pre-built components, saving development time.

Consistency: Ensures consistency in design and layout across different projects.

Responsive Design: Comes with built-in responsive design features for mobile-friendly websites.

1. Viewport Meta Tag:

Include the viewport meta tag in the <head> of your HTML document. This tag helps browsers render the page properly on different devices.

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

2. Fluid Grid Layout:

Use a fluid grid layout to create flexible and proportionate columns. Instead of fixed pixel values, use percentages for widths.

```
.container {
  width: 100%;
  max-width: 1200px; /* Set a maximum width for larger screens */
  margin: 0 auto; /* Center the container */
}

.column {
  width: 100%;
  float: left;
}

@media (min-width: 768px) {
  .column {
```

```
    width: 48%; /* Two columns on larger screens */
  }
}
```

3. Media Queries:

Employ media queries to apply different styles based on the device's characteristics. Adjust layout, font sizes, and other properties for different screen sizes.

```
/* Small screens (phones) */
@media only screen and (max-width: 600px) {
  body {
    font-size: 14px;
  }
}
/* Medium screens (tablets) */
@media only screen and (min-width: 601px) and (max-width: 1024px) {
  body {
    font-size: 16px;
  }
}

/* Large screens (desktops) */
@media only screen and (min-width: 1025px) {
  body {
    font-size: 18px;
  }
}
```

4. Flexible Images:

Ensure that images scale proportionally within their containers using the max-width: 100% CSS rule.

```
css
```

Copy code

```
img {
  max-width: 100%;
  height: auto;
}
```

5. Mobile-First Approach:

Start with styles for smaller screens and progressively enhance for larger screens using media queries. This ensures a mobile-friendly design.

6. Testing:

Regularly test your responsive design using browser developer tools, real devices, and online testing tools to ensure a consistent and optimized experience across various devices.

7. Flexbox and Grid:

Consider using CSS Flexbox and Grid layout for more advanced and flexible control over your page structure.

Example HTML Structure:

html

Copy code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <div class="column">Content 1</div>
    <div class="column">Content 2</div>
```

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

Testing responsive design is crucial to ensure that your web application or site looks and functions well across various devices and screen sizes. Here are several methods to test and debug your responsive design using CSS:

1. Browser Developer Tools:

Most modern browsers come equipped with built-in developer tools that include features for testing responsive designs. Access these tools by right-clicking on your webpage and selecting "Inspect" or "Inspect Element." In the developer tools, look for a device toolbar or a responsive design mode.

Chrome DevTools:

Open Chrome DevTools (F12 or right-click and select "Inspect").

Click on the "Toggle Device Toolbar" icon (or press Ctrl + Shift + M).

Select different devices or customize your viewport size.

Firefox DevTools:

Open Firefox DevTools (F12 or right-click and select "Inspect Element").

Click on the "Toggle Responsive Design Mode" icon (or press Ctrl + Shift + M).

Choose various devices or set a custom size.

Edge DevTools:

Open Edge DevTools (F12 or right-click and select "Inspect Element").

Click on the "Toggle Device Toolbar" icon.

Choose devices or set a custom size.

2. Real Devices:

Testing on actual devices provides the most accurate representation of how your site will behave. Use smartphones, tablets, and other devices to physically interact with your site.

3. Online Responsive Design Testing Tools:

Several online tools help you test your website's responsiveness across different devices and screen sizes:

Responsinator

BrowserStack

Am I Responsive?

Responsive Design Checker

4. Emulators and Simulators:

Emulators: Software that mimics the behavior of mobile devices. Examples include Android Emulator and Xcode iOS Simulator.

Simulators: Tools that simulate device behavior. Examples include Electric Mobile Studio and Ripple Emulator.

5. CSS Validation:

Use CSS validators to check your stylesheets for errors and potential issues. While this won't directly test responsiveness, it helps ensure your CSS is well-formed and doesn't contain errors that could impact layout.

6. Performance Testing:

Test your website's performance on different networks using tools like Google PageSpeed Insights or Lighthouse.

7. User-Agent Switchers:

Use browser extensions or plugins to switch user agents, simulating different devices. This is useful for testing how your site responds to various browsers.

8. Viewport Testing:

Manually resize your browser window to observe how your layout adjusts at different breakpoints.

9. Cross-Browser Testing:

Ensure compatibility across different browsers by testing your site on various browsers like Chrome, Firefox, Safari, and Edge.

10. Usability Testing:

Invite users to test your responsive design and provide feedback. Observing how real users interact with your site on different devices can uncover usability issues.