Competency Based Learning Materials (CBLMs)

Web Design

Level-3

Module 3: Working with CSS

Code: CBLM-ICT-WD-03-L3-EN-V1

National Skills Development Authority
Prime Minister's Office
Government of the People's Republic of Bangladesh

Copyright

National Skills Development Authority
Prime Minister's Office
Level: 10-11, Biniyog Bhaban,
E-6 / B, Agargaon, Sher-E-Bangla Nagar Dhaka-1207, Bangladesh.
Email: ec@nsda.gov.bd
Website: www.nsda.gov.bd.
National Skills Portal: http:\\skillsportal.gov.bd

The CBLM on "Work with CSS" is developed based on NSDA approved Competency Standards and Competency Based Curriculum under Web Design Level-3 Occupation. It contains the information required to implement the Web Design Level-3 standard.

This document has been prepared by NSDA with the help of relevant experts, trainers/professionals.

All Government-Private-NGO training institutes in the country accredited by NSDA can use this CBLM to implement skill-based training of Web Design level-3 course.

Approved by

---th Executive Committee (EC) Meeting of NSDA

Held on ----------

How to use this Competency Based Learning Materials (CBLMs)

The module, Working with CSS contains training materials and activities for you to complete. These activities may be completed as part of structured classroom activities or you may be required you to work at your own pace. These activities will ask you to complete associated learning and practice activities in order to gain knowledge and skills you need to achieve the learning outcomes.

1. Review the **Learning Activity** page to understand the sequence of learning activities you will undergo. This page will serve as your road map towards the achievement of competence.

2. Read the **Information Sheets.** This will give you an understanding of the jobs or tasks you are going to learn how to do. Once you have finished reading the **Information Sheets** complete the questions in the **Self-Check.**

3. **Self-**Checks are found after each **Information Sheet**. **Self-Checks** are designed to help you know how you are progressing. If you are unable to answer the questions in the **Self-Check** you will need to re-read the relevant **Information Sheet**. Once you have completed all the questions check your answers by reading the relevant **Answer Keys** found at the end of this module.

4. Next move on to the **Job Sheets. Job Sheets** provide detailed information about *how to do the job* you are being trained in. Some **Job Sheets** will also have a series of **Activity Sheets**. These sheets have been designed to introduce you to the job step by step. This is where you will apply the new knowledge you gained by reading the Information Sheets. This is your opportunity to practise the job. You may need to practise the job or activity several times before you become competent.

5. Specification **sheets**, specifying the details of the job to be performed will be provided where appropriate.

6. A review of competency is provided on the last page to help remind if all the required assessment criteria have been met. This record is for your own information and guidance and is not an official record of competency

When working though this Module always be aware of your safety and the safety of others in the training room. Should you require assistance or clarification please consult your trainer or facilitator.

When you have satisfactorily completed all the Jobs and/or Activities outlined in this module, an assessment event will be scheduled to assess if you have achieved competency in the specified learning outcomes. You will then be ready to move onto the next Unit of Competency or Module

Table of Contents

Module Content

Unit Title: Work with CSS
Unit Code: OU- ICT-WD-03-L3-V1
Module Title: Working with CSS

Module Descriptor: This module encompasses the necessary knowledge, skills, and attitudes (KAS) for establishing work with CSS. It specifically includes interpreting and applying CSS, a responsive approach, and a CSS grid.

Nominal Hours: 20

Learning Outcomes:

Upon completion of this module the trainees will be able to:

1. Interpret CSS
2. Apply CSS
3. Use a responsive approach.
4. Use CSS grid.

Assessment Criteria:

1. CSS (Cascading Style Sheets) is interpreted.
2. Types of CSS are identified.
3. Syntax of CSS is interpreted.
4. Selector of CSS is interpreted.
5. CSS file is created.
6. CSS file is integrated.
7. CSS is implemented as per layout.
8. CSS box model and positioning is applied.
9. CSS transition and gradients are applied.
10. 2D/3D transformation and animation is applied.
11. Responsive layout is defined.
12. Media Query is interpreted with CSS.
13. Media query is implemented.
14. Responsive approach is applied on a webpage.
15. CSS grid is interpreted.
16. CSS Grid container is defined.
17. CSS grid items are identified.
18. CSS grid is applied.

Learning Outcome 1: Interpret CSS

Assessment Criteria	1. CSS (Cascading Style Sheets) is interpreted. 2. Types of CSS are identified. 3. Syntax of CSS is interpreted. 4. Selector of CSS is interpreted.
Conditions and Resources	1. Applicable tools, utensils, and equipment as prescribed by competency standard. 2. Supply materials 3. Relevant ingredients 4. CBLM related to the learning outcome. 5. Instructions, job sheets, activity sheets, and standard operating procedures 6. Personal protective equipment 7. Module/reference
Contents	1 CSS (Cascading Style Sheets) 2 Types of CSS 3 CSS Syntax 4 CSS Selectors
Learning Materials	1. CBLM 2. Handouts 3. Books, Manuals 4. Module/ Reference 5. Paper 6. Pen
Training Methods	1. Discussion 2. Presentation 3. Demonstration 4. Guided Practice 5. Individual Practice 6. Project Work 7. Problem Solving 8. Brainstorming
Assessment Methods	1. Written Test 2. Demonstration 3. Oral Questioning

Learning Experience 1: Interpret CSS

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about setup web design environment	1. Instructor will provide the learning materials setup Interpret CSS
2. Read the **Information sheet/s**	2. Information Sheet No:1 Interpret CSS
3. Complete the **Self-Checks & Answer key sheets.**	3. Self-Check No: 1- Interpret CSS Answer key No. 1- Interpret CSS
4. Read the **Job/ Task sheet** and **Specification Sheet**	4. Job/ task sheet and specification sheet Task Sheet No:1-1: Set up the Project

Information Sheet 1: Interpret CSS

Learning Objective:

After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

1.1 CSS (Cascading Style Sheets)
1.2 Types of CSS
1.3 CSS Syntax
1.4 CSS Selectors

1.1 CSS (Cascading style sheets):

Cascading Style Sheets (CSS) is a stylesheet language used for describing the presentation and formatting of a document written in HTML or XML. It is commonly used in web development to control the appearance of web pages, including elements such as layout, colors, fonts, and animations.

CSS works by defining rules that determine how specific elements in an HTML document should be displayed. These rules consist of selectors and declarations. Selectors target specific HTML elements, while declarations specify the desired styles for those elements.

1.1.1 Here's a breakdown of the main components of CSS:

- Selectors: Selectors are used to target HTML elements for styling. They can target elements based on their type (e.g., **p** for paragraphs), class (e.g., .**my-class**), ID (e.g., **#my-id**), attributes, or hierarchical relationships between elements (e.g., **div > p** selects paragraphs that are direct children of a div).

- **Declarations:** Declarations define the styles to be applied to the selected elements. A declaration consists of a property and a value separated by a colon. For example, color: red; sets the text color of the selected elements to red.

- **Properties:** Properties define the aspects of an element's appearance that can be modified, such as color, font-size, background-image, margin, and many others.
- **Values:** Values are assigned to properties to specify the desired style. For example, color: red; sets the color property to the value red.

- **Style Rules:** Style rules combine selectors and declarations to define how specific elements should be styled. Multiple style rules can be grouped together in a CSS file or embedded within the <style> tag in an HTML document.

1.1.2 CSS provides a wide range of features and capabilities, including:

- **Box Model:** CSS allows you to control the size, padding, margin, and border of elements.
- **Layout:** CSS provides various techniques to position and arrange elements on a web page, including floating, positioning, and flexible box layout (flexbox) or grid layout (CSS Grid).
- **Typography:** CSS enables you to control the font family, size, weight, style, and other text-related properties.
- **Colors and Backgrounds:** CSS provides options to set colors for text, backgrounds, and borders, including gradients and transparency.
- **Transitions and Animations:** CSS allows you to create smooth transitions and animations, defining how elements should change over time.
- **Media Queries:** CSS supports media queries that let you define different styles for different screen sizes or devices, enabling responsive web design.

CSS can be included in an HTML document using inline styles, internal stylesheets, or external stylesheets. External stylesheets are commonly used, where a separate CSS file is linked to the HTML document, allowing consistent styling across multiple web pages.

1.1.3 Why Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
</style>
</head>
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Figure 1: CSS Structure

1.1.4 Output:

Figure 2: CSS Structure Output

1.1.5 CSS Solved a Big Problem:

HTML was NEVER intended to contain tags for formatting a web page!

HTML was created to describe the content of a web page, like:

```
<h1>This is a heading</h1>
<p>This is a paragraph. </p>
```

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process. To solve this problem, the World Wide Web Consortium (W3C) created CSS.

CSS removed the style formatting from the HTML page!

1.2 Types of CSS

Cascading Style Sheet (CSS) is used to set the style in web pages that contain HTML elements. It sets the background color, font-size, font-family, color, … etc. properties of elements on a web page.

There are three types of CSS which are given below:

- Inline CSS
- Internal or Embedded CSS
- External CSS

- **Inline CSS:** Inline CSS contains the CSS property in the body section attached to the element is known as inline CSS. This kind of style is specified within an HTML tag using the style attribute.

Example: This example shows the application of inline-css

```
<!DOCTYPE html>
<html>
<head>
<title>Inline CSS</title>
</head>
```

```
<body>
<p style="color:#009900; font-size:50px;
font-style:italic; text-align:center;">
GeeksForGeeks
</p>
</body>
</html>
```

Output

:

Figure 3: Inline CSS Output

- **External CSS:** External CSS contains separate CSS files that contain only style properties with the help of tag attributes (For example class, id, heading, … etc). CSS property is written in a separate file with a .css extension and should be linked to the HTML document using a link tag. It means that, for each element, style can be set only once and will be applied across web pages.

  Example: The file given below contains CSS property. This file saves with .css extension. For Ex: geeks.css

```
body {
background-color:powderblue;
}
.main {
text-align:center;
}
.GFG {
color:#009900;
font-size:50px;
font-weight:bold;
}
#geeks {
font-style:bold;
```

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

Below is the HTML file that is making use of the created external style sheet.

- link tag is used to link the external style sheet with the html webpage.
- href attribute is used to specify the location of the external style sheet file.

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="geeks.css" />
</head>

<body>
<div class="main">
<div class="GFG">GeeksForGeeks</div>
<div id="geeks">
A computer science portal for geeks
</div>
</div>
</body>
</html>
```

Figure 4: External CSS Output

Priorities of CSS: Inline CSS has the highest priority, then comes Internal/Embedded followed by External CSS which has the least priority. Multiple style sheets can be defined

on one page. For an HTML tag, styles can be defined in multiple style types and follow the below order.

- As Inline has the highest priority, any styles that are defined in the internal and external style sheets are overridden by Inline styles.
- Internal or Embedded stands second in the priority list and overrides the styles in the external style sheet.
- External style sheets have the least priority. If there are no styles defined either in inline or internal style sheet then external style sheet rules are applied for the HTML tags.

Inline CSS:

Inline CSS refers to the practice of including CSS styles directly within individual HTML elements using the style attribute. With inline CSS, you can apply specific styles to an element without the need for an external or internal stylesheet. For this CSS style, you'll only need to add the **style** attribute to each HTML tag, without using selectors.

This CSS type is not really recommended, as each HTML tag needs to be styled individually. Managing your website may become too hard if you only use inline CSS.

```
<p style="color: blue; font-size: 16px;">This is a paragraph with inline CSS</p>
```

In the above example, the **style** attribute is added to the **<p>** (paragraph) element. Within the **style** attribute, CSS properties and values are specified, separated by semicolons. In this case, the text color is set to blue **(color: blue**;) and the font size is set to 16 pixels (**font-size: 16px**;).

Inline CSS can be applied to any HTML element and allows you to override or add specific styles directly to that element. This approach can be useful when you want to apply unique or temporary styles to individual elements without affecting the rest of the page. However, it is generally not recommended for large-scale styling because it can make the code harder to maintain and reuse.

It's worth noting that inline CSS takes precedence over other CSS rules, including styles defined in an external stylesheet or internal <**style**> tags. Therefore, if conflicting styles are applied to the same element through different methods, the inline CSS will take priority.

However, inline CSS in HTML can be useful in some situations. For example, in cases where you don't have access to CSS files or need to apply styles for a single element only. Let's take a look at an example. Here, we add an inline CSS to the **<p>** and **<h1>** tag:

```
<!DOCTYPE html>
<html>
<body style="background-color:black;">

<h1 style="color:white;padding:30px;">Hostinger Tutorials</h1>
<p style="color:white;">Something usefull here.</p>

</body>
</html>
```

Advantages of Inline CSS:

- You can easily and quickly insert CSS rules to an HTML page. That's why this method is useful for testing or previewing the changes, and performing quick-fixes to your website.
- You don't need to create and upload a separate document as in the external style.

Disadvantages of Inline CSS:

- Adding CSS rules to every HTML element is time-consuming and makes your HTML structure messy.
- Styling multiple elements can affect your page's size and download time.

Embedded/ Internal:

Internal or embedded CSS requires you to add <style> tag in the <head> section of your HTML document.

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

Here's how you can use internal CSS:

- Open your HTML page and locate <head> opening tag.
- Put the following code right after the <head> tag

```
<style type="text/css">
```

- Add CSS rules on a new line. Here's an example:

```
body {
background-color: blue;
}
h1 {
color: red;
padding: 60px;
}
```

1. Type the closing tag:

```
</style>
```

2. Your HTML file will look like this:

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
body {
background-color: blue;
}
h1 {
color: red;
padding: 60px;
}
</style>
</head>
<body>
<h1>Hostinger Tutorials</h1>
<p>This is our paragraph.</p>
</body>
</html>
```

Here's an example of embedded CSS:

```
<!DOCTYPE html>
<html>
<head>
<title>Embedded CSS Example</title>
<style>
p {
color: blue;
font-size: 16px;
}
h1 {
color: red;
font-size: 24px;
}
</style>
</head>
<body>
<h1>This is a heading with embedded CSS</h1>
<p>This is a paragraph with embedded CSS</p>
</body>
</html>
```

In the above example, the CSS styles are defined within the <**style**> tags in the <**head>** section. The **p** selector specifies that all **<p>** (**paragraph**) elements should have a text color of blue and a font size of 16 pixels. Similarly, the h1 selector specifies that all **<h1**> (**heading level 1**) elements should have a text color of red and a font size of **24** pixels.

Embedded CSS allows you to keep the styles within the HTML file itself, making it convenient for small-scale projects or when you want to quickly apply styles to a single web page. However, it can become less manageable as the project grows larger or when you need to maintain consistent styles across multiple pages.

When using embedded CSS, the styles defined within the <**style**> tags will take precedence over external stylesheets but will be overridden by inline styles. It follows the cascading nature of CSS, where the order of style application is inline styles < **embedded styles** > external stylesheets.

For more complex or extensive styling, it is generally recommended to use external stylesheets, as they offer better organization, reusability, and ease of maintenance across multiple HTML files.

Advantages of Internal CSS:

- You can use class and ID selectors in this style sheet. Here's an example:

```
.class {
property1 : value1;
property2 : value2;
property3 : value3;
}
#id {
property1 : value1;
property2 : value2;
property3 : value3;
}
```

- Since you'll only add the code within the same HTML file, you don't need to upload multiple files.

Disadvantages of Internal CSS:

- Adding the code to the HTML document can increase the page's size and loading time.

External CSS:

External CSS refers to the practice of linking a separate CSS file to an HTML document. This method allows you to define styles in a separate file and apply them consistently across multiple HTML files. It promotes better organization, reusability, and maintainability of styles in web development.

Here's how external CSS is used:

- Create a CSS file: First, create a separate file with a .**css** extension, such as **styles.css**. This file will contain all the CSS rules and styles.

- Link the CSS file to HTML: In the <**head**> section of your HTML document, add a <**link**> tag that references the **CSS** file. The **href** attribute should point to the location of the CSS file.

```
<!DOCTYPE html>
<html>
<head>
<title>External CSS Example</title>
<link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph</p>
</body>
</html>
```

- Define styles in the CSS file: Open the **CSS** file (**styles.css** in this example) and define the styles using CSS selectors, properties, and values.

```
/* styles.css */

h1 {
color: red;
font-size: 24px;
}

p {
color: blue;
font-size: 16px;
}
```

In the above example, the CSS file (styles.css) defines styles for <**h1**> and <**p**> elements. The <**h1>** elements will have a red text color and a font size of **24** pixels, while the **<p>** elements will have a blue text color and a font size of 16 pixels.

External CSS provides several advantages:

- **Reusability**: You can link the same CSS file to multiple HTML documents, ensuring consistent styles across the website.

- **Maintainability**: Keeping styles separate from the HTML makes it easier to manage and update styles without modifying each HTML file.
- **Performance**: By using an external CSS file, the browser can cache the file, resulting in faster subsequent page loads.

It's common practice to organize the CSS file into sections or groups based on the elements or components they style, making it easier to find and modify styles when needed.

Overall, external CSS is widely used in web development to achieve scalable, maintainable, and consistent styling across web pages.

1.3 CSS Syntax

To interpret CSS syntax, it's important to understand the various components and rules that make up valid CSS code. Here's a breakdown of the key aspects of CSS syntax:

CSS Syntax

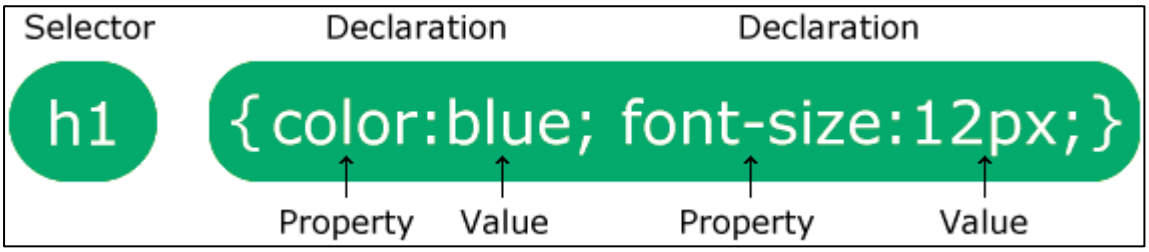


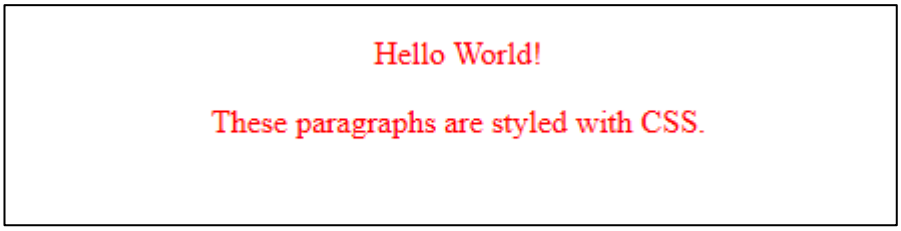
Figure 5: CSS Syntax

- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  color: red;
  text-align: center;
}
</style>
</head>
<body>

<p>Hello World!</p>
<p>These paragraphs are styled with CSS.</p>

</body>
</html>
```

Example

Output:



Explained

- p is a selector in CSS (it points to the HTML element you want to style: <p>).
- color is a property, and red is the property value
- text-align is a property, and center is the property value

1.4 CSS Selectors:

CSS selectors are patterns used to select and target specific HTML elements to apply styles or perform actions. They allow you to define which elements should be affected by the CSS rules you write. CSS provides a wide range of selectors to accommodate different targeting needs. Here are some commonly used CSS selectors:

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

This page will explain the most basic CSS selectors.

The CSS element Selector

The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

```
p {
  text-align: center;
  color: red;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<p>Every paragraph will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```

Output:

Every paragraph will be affected by the style.

Me too!

And me!

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```

Output:

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

```
.center {
 text-align: center;
  color: red;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>

</body>
</html>
```

Output:

The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

```
* {
 text-align: center;
  color: blue;
}
```

```html
<!DOCTYPE html>
<html>
<head>
<style>
* {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>

<h1>Hello world!</h1>

<p>Every element on the page will be affected by the style.</p>
<p id="para1">Me too!</p>
<p>And me!</p>

</body>
</html>
```

Output:

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
h1 {
text-align: center;
color: red;
}

h2 {
text-align: center;
color: red;
}

p {
text-align: center;
color: red;
}
```

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

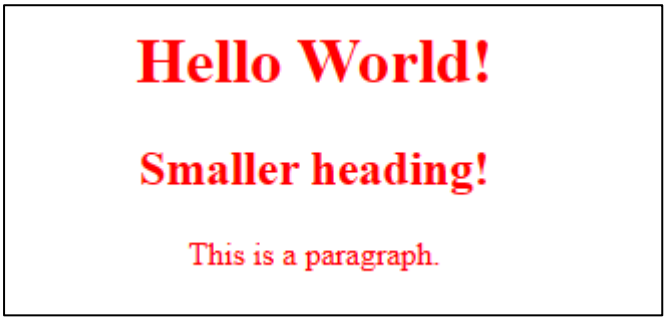
Example

In this example we have grouped the selectors from the code above:

```
 h1, h2, p {
   text-align: center;
   color: red;
                                                    }
```

```
<!DOCTYPE html>
<html>
<head>
<style>
h1, h2, p {
  text-align: center;
  color: red;
}
</style>
</head>
<body>

<h1>Hello World!</h1>
<h2>Smaller heading!</h2>
<p>This is a paragraph.</p>

</body>
</html>
```

Output:



All CSS Simple Selectors

Selector	Example	Example description
#id	#firstname	Selects the element with id="firstname"
.class	.intro	Selects all elements with class="intro"
element.class	p.intro	Selects only <p> elements with class="intro"
*	*	Selects all elements
element	p	Selects all <p> elements
element,element,...	div, p	Selects all elements and all <p> elements

Figure 6: CSS Simple Selectors

Self-Check Sheet 1: Interpret CSS

1 What does CSS stand for?

Answer:

2 What is the purpose of CSS?

Answer:

3 What are the three ways to include CSS in an HTML document?

Answer:

4 What is the difference between class and ID selectors in CSS?

Answer:

5 How can you select all <a> (anchor) elements within a specific element?

Answer:

6 How do you select an element with the ID "my-element" using CSS?

Answer:

7 What is the CSS box model?

Answer:

8 How can you center a block-level element horizontally in CSS?

Answer:

9 How can you apply multiple CSS classes to an element?

Answer:

10 How do you specify a font-family in CSS?

Answer:

Answer Key 1: Interpret CSS

1 What does CSS stand for?
Answer: CSS stands for Cascading Style Sheets.

2 What is the purpose of CSS?
Answer: The purpose of CSS is to describe the presentation and formatting of a document written in HTML or XML, including elements such as layout, colors, fonts, and animations.

3 What are the three ways to include CSS in an HTML document?
Answer: Inline CSS, internal/embedded CSS, and external CSS

4 What is the difference between class and ID selectors in CSS?
Answer: Class selectors target elements based on a specific class attribute value, while ID selectors target elements based on a specific ID attribute value. Class selectors can be used on multiple elements, while ID selectors should be unique within the document.

5 How can you select all <a> (anchor) elements within a specific element?
Answer: By using the descendant selector: div a

6 How do you select an element with the ID "my-element" using CSS?
Answer: By using the ID selector: #my-element

7 What is the CSS box model?
Answer: The CSS box model is a concept that describes how elements are displayed and spaced, including properties such as content, padding, border, and margin

8 How can you center a block-level element horizontally in CSS?
Answer: By setting its left and right margins to auto and giving it a specific width

9 How can you apply multiple CSS classes to an element?
Answer: By adding multiple class names to the class attribute, separated by spaces

10 How do you specify a font-family in CSS?
Answer: By using the font-family property. For example: font-family: Arial, sans-serif;

Task Sheet 1.1 Set up the Project

Objectives: The objective of this task is to understand and interpret Cascading Style Sheets (CSS), which is a stylesheet language used to describe the presentation of HTML documents. By completing this task, you will be able to apply various CSS properties and techniques to style and format web pages effectively.

Working Procedure:

1. Create a new folder on your computer for this project.
2. Inside the project folder, create an HTML file (e.g., "index.html") and a CSS file (e.g., "styles.css").
3. Use a code editor of your choice (e.g., Visual Studio Code, Sublime Text) to work on the files.

Note: Remember to save your progress regularly and continue learning by exploring additional CSS concepts and advanced techniques. The more you practice, the more proficient you will become in interpreting and using Cascading Style Sheets to create visually appealing and responsive web pages.

Learning Outcome 2: Apply CSS

Assessment Criteria	1. CSS file is created. 2. CSS file is integrated. 3. CSS is implemented as per layout. 4. CSS box model and positioning is applied. 5. CSS transition and gradients are applied. 6. 2D/3D transformation and animation is applied.
Conditions and Resources	1. Applicable tools, utensils, and equipment as prescribed by competency standards. 2. Supply materials 3. Relevant ingredients 4. CBLM related to the learning outcome. 5. Instructions, job sheets, activity sheets, and standard operating procedures 6. Personal protective equipment 7. Module/reference
Contents	1 CSS File 2 Integrating CSS Files 3 CSS for Layout 4 CSS Box Model and Positioning 5 CSS Transitions and Gradients 6 2D/3D Transformations and Animations
Learning Materials	1. CBLM 2. Handouts 3. Books, Manuals 4. Module/ Reference 5. Paper 6. Pen
Training Methods	1. Discussion 2. Presentation 3. Demonstration 4. Guided Practice 5. Individual Practice 6. Project Work 7. Problem Solving 8. Brainstorming
Assessment Methods	1. Written Test 2. Demonstration 3. Oral Questioning

Learning Experience 2: Apply CSS

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about work with CSS	1. Instructor will provide the learning materials apply CSS
2. Read the **Information sheet/s**	2. Information Sheet No:2 Apply CSS
3. Complete the **Self-Checks & Answer key sheets.**	3. Self-Check No: 2- Apply CSS Answer key No. 2- Apply CSS
4. Read the **Job/ Task sheet** and **Specification Sheet**	4. Job/ task sheet and specification sheet Job Sheet No:1-1: Apply Cascading Style Sheets (CSS) to Style and Enhance the Appearance of an HTML Document. Specification Sheet: 1-1: Apply Cascading Style Sheets (CSS) to Style and Enhance the Appearance of an HTML Document.

Information Sheet 2: Apply CSS

Learning Objective:

After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

2.1 CSS File
2.2 Integrating CSS Files
2.3 CSS for Layout
2.4 CSS Box Model and Positioning
2.5 CSS Transitions and Gradients
2.6 2D/3D Transformations and Animations

2.1 CSS File

Creating a CSS (Cascading Style Sheets) file is a straightforward process. CSS is used to define the visual styles and layout of web documents, including HTML pages. Here's a step-by-step guide on creating a CSS file:

In this session we are going to write and save our first CSS file. Let's begin by opening a text editing program. If you are on a Microsoft Windows PC open the program named Notepad (hold down the Windows Key on your keyboard and press R, then type notepad and press enter). If you are using a Macintosh computer, launch the application named "TextEdit" (which can be found in your Apps folder).

2.1.1 Let's Write Our First Bit of CSS

Let's imagine we have a simple web page with a heading, and we want the heading to be orange and center aligned. Add the following code into your new blank text document:

```
h1 {
color: orange;
text-align: center;
}
```

Hopefully, you remember this code from our previous lesson. The task for today is to save our CSS file and link it to an HTML page.

Step 1: Saving the CSS File

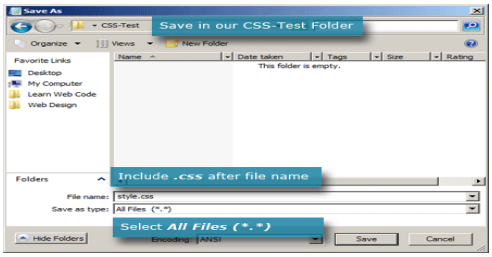


Figure 7: Save CSS File

Save the file: Save the new file with a .css extension. For example, you can name it **styles.css** or any other relevant name. Make sure to save it in a location where it will be accessible by your HTML files.

Step 2: Linking CSS File to an HTML Page

Link the CSS file to your HTML document: To apply the CSS styles to your HTML document, you need to link the CSS file to it. In the <head> section of your HTML file, add a <link> tag with the rel, type, and href attributes. Here's an example

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

The HTML File:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>CSS-Test</title>
</head>
<body>
<h1>CSS-Test</h1>
<div id="box-one">
<p>This is box one.</p>
</div>
<div id="box-two">
<p>This is box two.</p>
</div>
</body>
</html>
```

Step 3: Now save this document in us

"CSS-Test" folder and name it "index.htm".

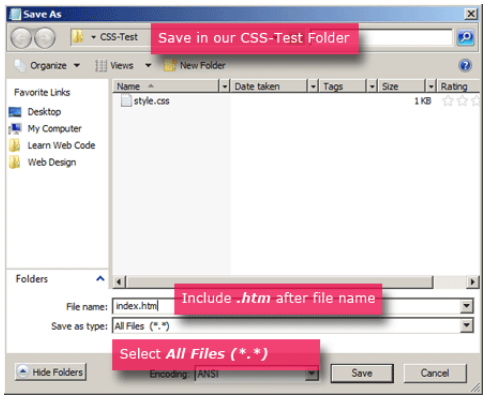
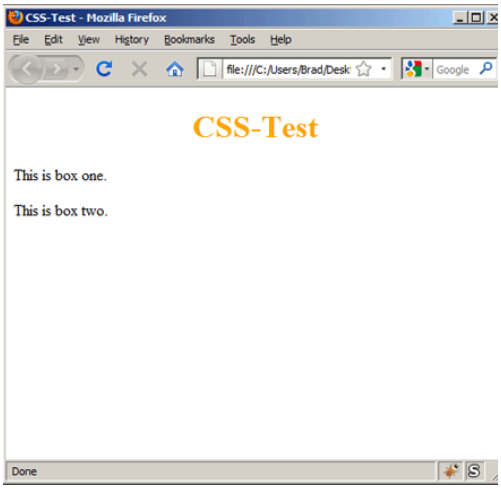


Figure 8: Save CSS File in "CSS Test" Folder

Linking the Two Files Together



We still need to tell the web browser to load our "style.css" file when the "index.htm" page is viewed. Add the following code to "index.htm" directly above our </head> closing tag:

```
<link rel="stylesheet" href="style.css">
```

This line of code tells our browser that we want to link a Style Sheet, that it's located in the same folder as our HTML page, and that it's named "style.css".

Now, when you view "index.htm" page in a web browser you should see a centered, orange heading:

Let's Style Those Two Boxes

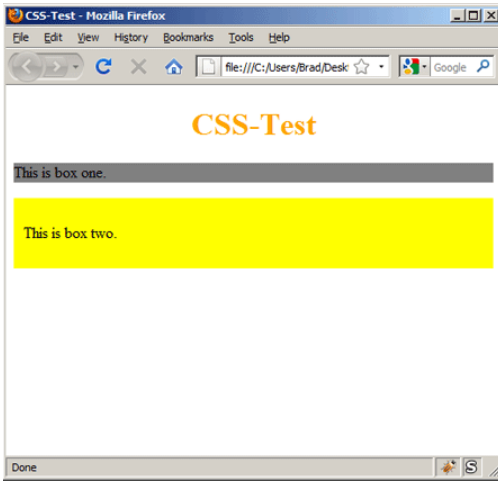
If you look at the code of our HTML page, you'll see two elements. We added an HTML attribute of "id" for these two elements and assigned them values of "box-one" and "box-two." We can use an element's "id" to select and style it with CSS. For example, let's make the first box gray, and the second box yellow. Add the following code to your CSS file, directly below our original <h1> rule:

```
#box-one
{
background-color: gray;
}
#box-two {
background-color: yellow;
padding: 10px;
}
```

When an element has an "id" we can access it with a CSS selector by placing a pound sign (#) in front of its id value. So to select the first element we simply type "#box-one" and then begin our CSS Rule.

Our New Fancy Boxes

When you save your CSS file and refresh our HTML page in a web browser you should see something very similar to this:



That's it! Your CSS file is now ready to be used to style your HTML document. Remember to refer to the appropriate selectors in your CSS file to target the desired HTML elements and apply the desired styles.

2.2 Integrating CSS Files

Integrating CSS files into your HTML document involves linking the CSS file(s) to your HTML file. Here's a step-by-step guide on how to integrate CSS files:

- **Create or locate your CSS file:** If you haven't already created a CSS file, follow the instructions in the previous response to create one. Make sure you know the file path or location of your CSS file.
- **Open your HTML file:** Open the HTML file you want to integrate the CSS file into using a text editor.
- **Identify the <head> section**: In the HTML file, locate the <**head**> section. This is where you typically include meta tags, title, and other document-related information.
- **Link the CSS file**: Inside the <**head**> section, add a <**link**> tag with the following attributes:
- **rel:** Specifies the relationship between the HTML file and the linked file. In this case, set it to "**stylesheet**".
- **type:** Specifies the MIME type of the linked file. For CSS files, set it to "**text/css**".
- **href:** Specifies the path or URL to the CSS file. Provide the appropriate path to your CSS file.

Here's an example of the <link> tag:

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

Replace "path/to/styles.css" with the actual path or URL to your CSS file.

- Save your HTML file: Save the changes made to your HTML file.

Now, when you open your HTML file in a web browser, it will load the CSS file specified in the <**link**> tag, and the styles defined in the CSS file will be applied to the HTML elements according to the selectors used in the CSS rules.

You can also link multiple CSS files to a single HTML file by including multiple <**link**> tags, each pointing to a different CSS file. This allows you to modularize your stylesheets and keep them organized.

2.3 CSS for Layout

Implementing CSS for layout involves using CSS properties and techniques to structure and position elements on a webpage. Here are some commonly used CSS techniques for layout:

2.3.1 Box Model: The box model is fundamental to CSS layout. Each HTML element is represented as a rectangular box, consisting of content, padding, border, and margin. You can control the size and spacing of elements using properties like width, height, padding, border, and margin.

2.3.2 Display Property: The display property specifies how an element should be displayed. The most common values are:

- block: Makes an element a block-level element, taking up the full width of its parent container.
- inline: Makes an element an inline-level element, allowing other elements to be next to it.
- inline-block: Combines aspects of block and inline, allowing an element to have block properties (e.g., width, height) while still being inline.
- none: Hides an element by removing it from the document flow.

2.3.3 Positioning: The position property controls how elements are positioned within their parent container. Common values include:

- static: The default positioning. Elements follow the normal flow of the document.
- relative: Elements are positioned relative to their normal position using properties like top, bottom, left, and right.
- absolute: Elements are positioned relative to the nearest positioned ancestor. If there is no positioned ancestor, it's relative to the document body.
- fixed: Elements are positioned relative to the browser window, so they stay in the same position even when scrolling.

2.3.4 Floats: The float property allows elements to be positioned side by side, either to the left or right of their container. Floated elements are removed from the normal document flow, affecting the positioning of other elements around them.

2.3.5 Flexbox: Flexbox is a powerful CSS layout module that provides flexible and responsive layouts. By setting the display property of a container to flex, you can use properties like flex-direction, justify-content, align-items, and flex to control the positioning and alignment of child elements.

2.3.6 Grid Layout: CSS Grid Layout allows you to create complex two-dimensional layouts. By setting the display property of a container to grid, you can define rows and columns and control the placement of elements within the grid using properties like grid-template-rows, grid-template-columns, and grid-area.

These are just a few of the many CSS techniques available for layout. Experimenting with these properties and modules will give you greater control over the structure and positioning of elements on your webpage. It's recommended to combine these techniques based on your specific layout requirements.

2.4 CSS Box Model and Positioning

The CSS Box Model: All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

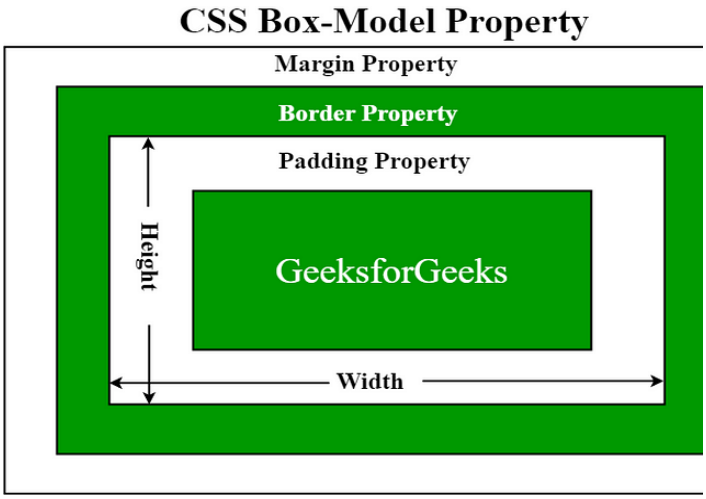


Figure 9: CSS Box Model and Positioning

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

- **Content —** The content of the box, where text and images appear
- **Padding —** Clears an area around the content. The padding is transparent
- **Border —** A border that goes around the padding and content
- **Margin —** Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

2.4.1 Width and Height of an Element

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

Important: When you set the width and height properties of an element with CSS, you just set the width and height of the content area. To calculate the full size of an element, you must also add padding, borders and margins.

Example

This element below will have a total width of 350px:

```
div {
width: 320px;
padding: 10px;
border: 5px solid gray;
margin: 0;
}
```

2.4.2 Understanding CSS Positioning

Positioning elements with CSS in web development isn't as easy as it seems. Things can get quickly complicated as your project gets bigger and without having a good understanding of how CSS deals with aligning HTML elements, you won't be able to fix your alignment issues.

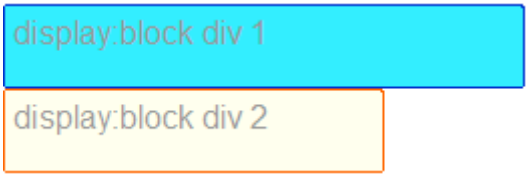
There are different ways/methods for positioning elements with pure CSS. Using CSS float, display and position properties are the most common methods. In this article, I will be explaining one of the most confusing ways for aligning elements with pure CSS: the position property.

It is worth noting that there are basically two types of ways to display an element in CSS: block and inline.

2.4.3 Display:block;

Block can be, quite literally, seen as a block. It has a specified width and height, optionally controlled by its content, but dimensions set by CSS have prevalence. Another property of elements with display:block is that they do not allow elements on the same "line" as their own.

So two simple divs with display block look like this:



So while an element might have a small width, The next element will always be placed under it.

2.4.4 Display:inline;

Display: inline is somewhat the opposite. Elements with display:inline always take their width and height from the contents, and will ignore any dimensions specified in the CSS. Another opposite is that, as the name suggest, these elements are displayed in-line, so they will always be next to the preceding element, providing that the preceding element is inline as well and that there is enough width left. If the width of the "line" is filled, an element with display: inline will wrap to the next line.

That looks like this:

There is a multitude of other display: properties such as table and inline-block.

2.4.5 Flow

With positioning, the elusive "Flow" of CSS positioning comes into play. The flow in CSS is the logical way in which elements get placed on your screen. For example, when you turn off the CSS on this page, you see the HTML in its original order. That order is the flow, and dictates which element gets rendered and placed were. It's very simple: the flow of a page is from the top of the HTML to the bottom of it.

The position Property:
The position property specifies the type of positioning method used for an element.

There are five different position values:

- static
- relative
- fixed
- absolute
- sticky

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

Now let's move on with the position property values.

position: static;

- HTML elements are positioned static by default.
- Static positioned elements are not affected by the top, bottom, left, and right properties.
- An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page.

```
div {
position: static;
}
```

position: relative;

An element with position: relative; is positioned relative to its normal position.

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

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

position: fixed;

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

```
div {
position: fixed;
top: 10px;
right: 20px;
}
```

A fixed element does not leave a gap in the page where it would normally have been located.

position: absolute;

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

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

```
div {
position: absolute;
top: 50px;
left: 100px;
}
```

position: sticky;

An element with position: sticky; is positioned based on the user's scroll position.

A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Overlapping Elements

When elements are positioned, they can overlap other elements.

The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order.

2.5 CSS Transitions and Gradients

CSS Transitions and Gradients are powerful features that allow you to add smooth animations and gradient backgrounds to your elements. Here's an overview of how to apply CSS Transitions and Gradients:

2.5.1 CSS Transitions:

CSS Transitions enable smooth animations when a CSS property changes its value over a specified duration. You can specify which properties to transition, the duration of the transition, and the timing function that controls the transition speed.

Here's an example of applying a transition effect to a <**div**> element when its background color changes:

```
div {
transition: background-color 0.3s ease;
}
div:hover {
background-color: blue;
}
```

In this example, when you hover over the element, the background color will smoothly transition to blue over a duration of 0.3 seconds with an ease timing function. You can adjust the transition duration, timing function, and other properties as needed.

2.5.2 CSS Gradients:

CSS Gradients allow you to create smooth color transitions between two or more specified colors. You can apply gradients to backgrounds, borders, or even text.

Here's an example of applying a linear gradient background to a element:

```
div {
background-image: linear-gradient(to right, red, blue);
}
```

In this example, the element will have a linear gradient background that transitions from red to blue from left to right. You can customize the gradient direction and add more color stops to create complex gradients.

Additionally, you can apply radial gradients using the radial-gradient() function and add color stops to control the smooth transition between colors.

```
div {
background-image: radial-gradient(circle, yellow, orange, red);
}
```

In this example, the element will have a radial gradient background that transitions from yellow to orange to red in a circular pattern.

Remember to adjust the CSS selectors and properties to target the appropriate HTML elements and achieve the desired effects. CSS Transitions and Gradients offer a wide range of possibilities to enhance the visual appeal of your webpages.

2.6 2D/3D Transformations and Animations

CSS 2D/3D Transformations and Animations allow you to manipulate and animate elements in a visually appealing manner. Here's an overview of how to apply these CSS features:

2.6.1 CSS 2D/3D Transformations:

CSS Transformations allow you to modify the position, size, and rotation of elements in 2D or 3D space. You can use properties like transform, translate, rotate, scale, and more to apply transformations.

CSS also supports 3D transformations.
Mouse over the elements below to see the difference between a 2D and a 3D transformation:

In this chapter you will learn about the following CSS property:

2.6.2 Transform

Browser Support
The numbers in the table specify the first browser version that fully supports the property.

Property					
transform	36.0	10.0	16.0	9.0	23.0

2.6.3 CSS 3D Transforms Methods

With the CSS transform property you can use the following 3D transformation methods:

- rotateX()
- rotateY()
- rotateZ()

2.6.4 The rotateX() Method

The rotateX() method rotates an element around its X-axis at a given degree:

Example

```
#myDiv {
  transform: rotateX(150deg);
}
```

Code:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 300px;
  height: 100px;
  background-color: yellow;
  border: 1px solid black;
}
#myDiv {
  transform: rotateX(150deg);
}
</style>
</head>
<body>
<h1>The rotateX() Method</h1>
<p>The rotateX() method rotates an element around its X-axis at a given
degree.</p>
<div>
This a normal div element.
</div>
<div id="myDiv">
This div element is rotated 150 degrees.
</div>
```

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

Output:

The rotateX() Method

The rotateX() method rotates an element around its X-axis at a given degree.

This a normal div element.

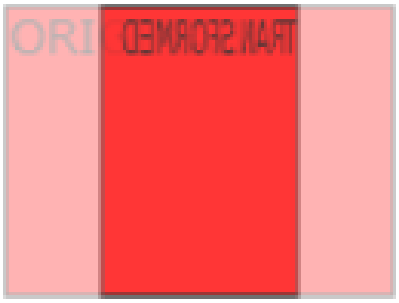
This div element is rotated 150 degrees.

2.6.5 The rotateY() Method

The rotateY() method rotates an element around its Y-axis at a given degree:

Example

```
#myDiv {
  transform: rotateY(150deg);
}
```

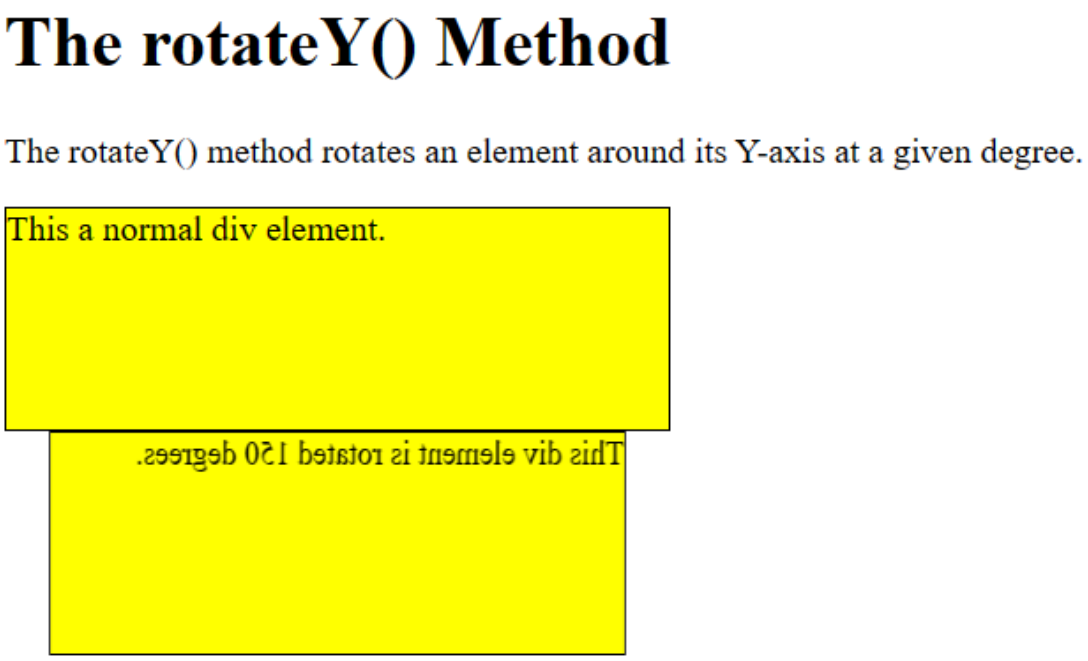


Code:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 300px;
  height: 100px;
  background-color: yellow;
  border: 1px solid black;
}

#myDiv {
  transform: rotateY(150deg);
}
</style>
</head>
<body>

<h1>The rotateY() Method</h1>

<p>The rotateY() method rotates an element around its Y-axis at a given degree.</p>

<div>
```

```
This a normal div element.
</div>

<div id="myDiv">
This div element is rotated 150 degrees.
</div>

</body>
</html>
```

Output:



2.6.6 The rotateZ() Method

The rotateZ() method rotates an element around its Z-axis at a given degree:

Example

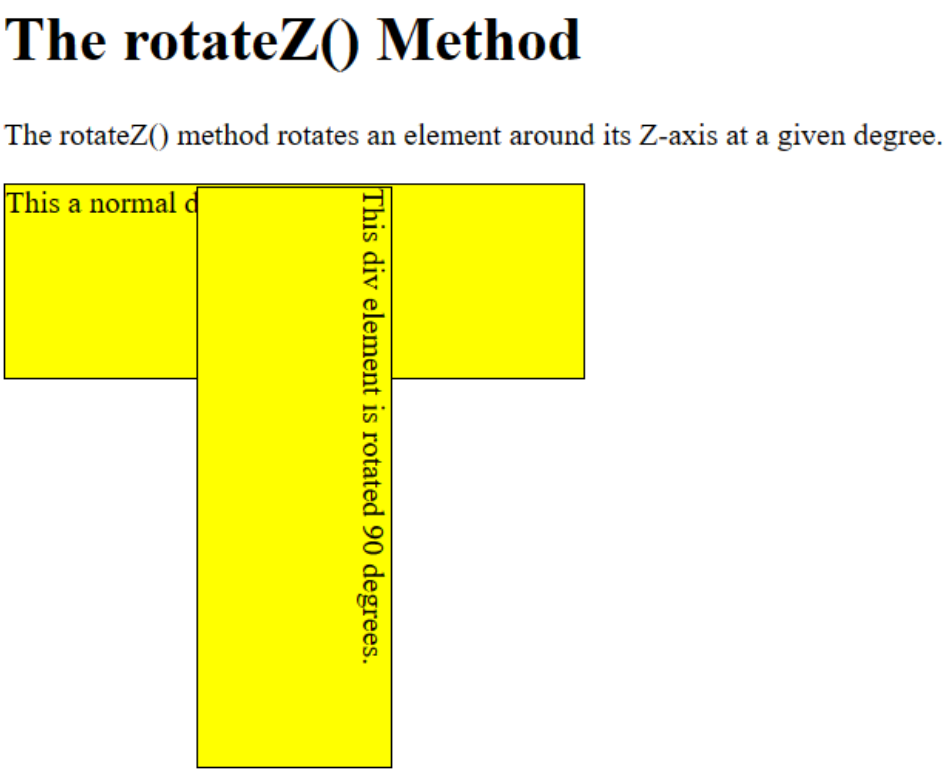
```
#myDiv {
  transform: rotateZ(90deg);
}
```

Code:

```
<!DOCTYPE html>
```

```
<html>
<head>
<style>
div {
  width: 300px;
  height: 100px;
  background-color: yellow;
  border: 1px solid black;
}
#myDiv {
  transform: rotateZ(90deg);
}
</style>
</head>
<body>
<h1>The rotateZ() Method</h1>
<p>The rotateZ() method rotates an element around its Z-axis at a given degree.</p>

<div>
This a normal div element.
</div>
<div id="myDiv">
This div element is rotated 90 degrees.
</div>
</body>
</html>
```

Output:



2.6.7 CSS Transform Properties

Property	Description
transform	Applies a 2D or 3D transformation to an element
transform-origin	Allows you to change the position on transformed elements
transform-style	Specifies how nested elements are rendered in 3D space
perspective	Specifies the perspective on how 3D elements are viewed
perspective-origin	Specifies the bottom position of 3D elements
backface-visibility	Defines whether or not an element should be visible when not facing the screen

Figure 10: CSS Transform Properties

2.6.8 CSS 3D Transform Methods

Function	Description
matrix3d (*n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n*)	Defines a 3D transformation, using a 4x4 matrix of 16 values
translate3d(*x,y,z*)	Defines a 3D translation
translateX(*x*)	Defines a 3D translation, using only the value for the X-axis
translateY(*y*)	Defines a 3D translation, using only the value for the Y-axis
translateZ(*z*)	Defines a 3D translation, using only the value for the Z-axis
scale3d(*x,y,z*)	Defines a 3D scale transformation
scaleX(*x*)	Defines a 3D scale transformation by giving a value for the X-axis
scaleY(*y*)	Defines a 3D scale transformation by giving a value for the Y-axis
scaleZ(*z*)	Defines a 3D scale transformation by giving a value for the Z-axis
rotate3d(*x,y,z,angle*)	Defines a 3D rotation
rotateX(*angle*)	Defines a 3D rotation along the X-axis
rotateY(*angle*)	Defines a 3D rotation along the Y-axis
rotateZ(*angle*)	Defines a 3D rotation along the Z-axis
perspective(*n*)	Defines a perspective view for a 3D transformed element

Figure 11: CSS 3D Transform Methods

2.6.9 CSS Animations:

CSS Animations allow you to create dynamic and interactive effects by specifying keyframes and animating CSS properties over time. You can define keyframes using the @keyframes rule and apply animations using the animation property.

Here's an example of applying a CSS animation to a element:

```
div {
  animation-name: slide;
  animation-duration: 2s;
  animation-timing-function: ease;
  animation-delay: 1s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
```

```
@keyframes slide {
  0% { transform: translateX(0); }
  50% { transform: translateX(200px); }
  100% { transform: translateX(0); }
}
```

In this example, the <**div**> element will slide horizontally back and forth continuously. The animation is defined using the **@keyframes** rule with three keyframes representing the initial position, midpoint, and final position of the animation. Properties like **animation-duration, animation-timing-function, animation-delay, animation-iteration-count**, and **animation-direction** control various aspects of the animation.

You can also animate other CSS properties like opacity, color, width, and more to create a wide range of visual effects.

Remember to adjust the CSS selectors and properties to target the desired HTML elements and achieve the desired transformations and animations. Experimentation and creativity will help you create engaging and interactive web experiences.

Self-Check Sheet 2: Apply CSS

1 What is the purpose of CSS in web development?

Answer:

2 How do you create a CSS file?

Answer:

3 How do you integrate a CSS file into an HTML document?

Answer:

4 What are the four components of the CSS Box Model?

Answer:

5 What does the position property control in CSS?

Answer:

6 What is the purpose of CSS transitions?

Answer:

7 What are CSS gradients used for?

Answer:

8 How can you apply a 2D transformation to an element using CSS?

Answer:

9 What does the @keyframes rule define in CSS animations?

Answer:

10 How can you control the duration of a CSS animation?

Answer:

Answer Key 2: Apply CSS

1 What is the purpose of CSS in web development?
Answer: CSS (Cascading Style Sheets) is used for styling and formatting HTML elements, allowing developers to control the visual appearance of webpages.

2 How do you create a CSS file?
Answer: You can create a CSS file by creating a new text file with a .css extension and saving it with appropriate CSS rules and styles

3 How do you integrate a CSS file into an HTML document?
Answer: You can integrate a CSS file into an HTML document by using the <link> tag with the appropriate rel, type, and href attributes to specify the path or URL to the CSS file

4 What are the four components of the CSS Box Model?
Answer: The four components of the CSS Box Model are content, padding, border, and margin

5 What does the position property control in CSS?
Answer: The position property controls the positioning method of an element, allowing you to specify if it should be static, relative, absolute, or fixed

6 What is the purpose of CSS transitions?
Answer: CSS transitions allow smooth animations when a CSS property changes its value over a specified duration, enhancing the user experience

7 What are CSS gradients used for?
Answer: CSS gradients are used to create smooth color transitions between two or more specified colors, enabling the creation of attractive backgrounds and effects

8 How can you apply a 2D transformation to an element using CSS?
Answer: You can apply a 2D transformation to an element using the transform property with functions like translate (), rotate (), and scale ().

9 What does the @keyframes rule define in CSS animations?
Answer: The @keyframes rule defines the intermediate steps or keyframes of an animation, allowing you to specify different styles at different points during the animation.

10 How can you control the duration of a CSS animation?
Answer: You can control the duration of a CSS animation using the animation-duration property, specifying the time it takes for the animation to complete.

Job Sheet 2.1 Apply Cascading Style Sheets (CSS) to Style and Enhance the Appearance of an HTML Document.

Objectives: The objective of this task is to apply Cascading Style Sheets (CSS) to style and enhance the appearance of an HTML document. By completing this task, you will gain hands-on experience in using CSS selectors, properties, and techniques to design visually appealing and responsive web pages.

Working procedure:

Task 1: Set Up the Project

1. Create a new folder on your computer for this project.
2. Inside the project folder, create an HTML file (e.g., "index.html") and a CSS file (e.g., "styles.css")
3. Use a code editor of your choice (e.g., Visual Studio Code, Sublime Text) to work on the files.

Task 2: Link CSS to HTML

1. Open the "index.html" file and link the CSS file in the <head> section using the <link> element.

Task 3: Apply Basic Styles

1. Use CSS to set font styles (font-family, font-size, font-weight) for different HTML elements (e.g., headings, paragraphs).
2. Change the background color or add a background image to the body or specific sections.

Task 4: Box Model Styling

1. Apply padding and margins to elements to control spacing and layout.
2. Set the border properties to add borders around elements.

Task 5: CSS Selectors

1. Use different CSS selectors to target specific HTML elements, classes, IDs, or attributes.
2. Apply styles to selected elements using the chosen selectors.

Task 6: Layout and Positioning

1. Experiment with CSS positioning properties (position, float, display) to control the layout of elements.
2. Create a simple layout using float or flexbox to arrange elements horizontally or vertically.

Task 7: Style Links and Buttons

1. Customize link styles (visited, hover, active) to make them stand out.
2. Apply styles to buttons to make them visually appealing and interactive.

Task 8: Test and Debug

1. Save your CSS file and refresh the "index.html" page in your web browser.
2. Use browser developer tools to inspect and debug any styling issues.

Task 9: Revise and Improve

1. Review your CSS code and refine it to achieve the desired appearance and layout.
2. Optimize CSS code by removing redundant styles and grouping similar styles together.

Task 10: Practice and Experiment

1. Regularly practice CSS by working on various layout designs and web page elements.
2. Experiment with different CSS properties and techniques to expand your skills.

Remember to save your progress regularly and keep experimenting with CSS to gain more proficiency in applying styles. As you encounter new challenges, research and learn about advanced CSS concepts to further enhance your web development skills.

Learning Outcome 3: Use a Responsive Approach

Assessment Criteria	1 Responsive layout is defined. 2 Media Query is interpreted with CSS. 3 Media query is implemented. 4 Responsive approach is applied on a webpage.
Conditions and Resources	1 Applicable tools, utensils, and equipment as prescribed by competency standard. 2 Supply materials 3 Relevant ingredients 4 CBLM related to the learning outcome. 5 Instructions, job sheets, activity sheets, and standard operating procedures 6 Personal protective equipment 7 Module/reference
Contents	1 Responsive Layout. 2 Media Queries with CSS. 3 Media Queries. 4 Responsive Approach to a Webpage.
Learning Materials	1 CBLM 2 Handouts 3 Books, Manuals 4 Module/ Reference 5 Paper 6 Pen
Training Methods	1 Discussion 2 Presentation 3 Demonstration 4 Guided Practice 5 Individual Practice 6 Project Work 7 Problem Solving 8 Brainstorming
Assessment Methods	1. Written Test 2. Demonstration 3. Oral Questioning

Learning Experience 3: Use a Responsive Approach

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
a) Student will ask the instructor about work with CSS	1. Instructor will provide the learning materials use a responsive approach
b) Read the **Information sheet/s**	2. Information Sheet No: 3: Use a responsive approach
c) Complete the **Self-Checks & Answer key sheets.**	3. Self-Check No: 3: Use a responsive approach Answer key No. 3: Use a responsive approach
d) Read the **Job/ Task sheet** and **Specification Sheet**	4. Job/ task sheet and specification sheet Job Sheet No:3 Implement a Responsive Approach in CSS to Create a Web page.

Information Sheet 3: Use a Responsive Approach

Learning Objective:

After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

3.1 Responsive Layout.
3.2 Media Queries with CSS.
3.3 Media Queries.
3.4 Responsive Approach to a Webpage.

3.1 Defining a Responsive Layout

A responsive layout refers to the design and development approach that aims to create web pages or applications that adapt and respond effectively to various screen sizes, devices, and orientations. The goal is to provide an optimal user experience, regardless of whether the user is accessing the content on a desktop computer, laptop, tablet, or smartphone.

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation.

The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities. One may also have to consider the settings on their devices; if they have a VPN for iOS on their iPad, for example, the website should not block the user's access to the page. In other words, the website should have the technology to automatically respond to the user's preferences. This would eliminate the need for a different design and development phase for each new gadget on the market.

Here are some key principles and techniques commonly used in creating responsive layouts:

- **Fluid Grids:** Instead of fixed pixel-based layouts, responsive designs use fluid grids that are based on relative proportions. This allows elements to resize and reposition themselves dynamically based on the screen size.

- **Flexible Images and Media:** Images and media elements, such as videos or embedded content, are made responsive by adjusting their size and scaling proportionally to fit different devices.

- **Media Queries:** CSS media queries are used to apply different styles and layouts based on the characteristics of the user's device, such as screen width, height, and orientation. Media queries enable developers to target specific devices or ranges of screen sizes and adapt the design accordingly.
- **Breakpoints:** Breakpoints are specific points in the responsive design where the layout and content rearrange or change to better fit the screen size. These breakpoints are often determined based on common device widths or popular screen resolutions.
- **Mobile-First Approach:** With the increasing dominance of mobile devices, many designers and developers adopt a mobile-first approach. This involves designing and developing for smaller screens initially and then progressively enhancing the layout and features as the screen size increases.
- **Content Prioritization:** Responsive layouts often involve prioritizing content based on its importance and relevance to the user. This ensures that critical information is readily accessible, even on smaller screens, while less essential content may be rearranged or hidden.

- By employing these techniques and principles, a responsive layout can provide a seamless and optimized user experience across a wide range of devices, improving accessibility and user engagement.

3.1.1 The Concept of Responsive Web Design

Transplant this discipline onto Web design, and we have a similar yet whole new idea. Why should we create a custom Web design for each group of users; after all, architects don't design a building for each group size and type that passes through it? Like responsive architecture, Web design should automatically adjust. It shouldn't require countless custom-made solutions for each new category of users.

Obviously, we can't use motion sensors and robotics to accomplish this the way a building would. Responsive Web design requires a more abstract way of thinking. However, some ideas are already being practiced: fluid layouts, media queries and scripts that can reformat Web pages and mark-up effortlessly (or automatically).

But responsive Web design is not only about adjustable screen resolutions and automatically resizable images, but rather about a whole new way of thinking about design. Let's talk about all of these features, plus additional ideas in the making.

3.1.2 Adjusting Screen Resolution

With more devices come varying screen resolutions, definitions and orientations. New devices with new screen sizes are being developed every day, and each of these devices may be able to handle variations in size, functionality and even color. Some are in landscape, others in portrait, still others even completely square. As we

know from the rising popularity of the iPhone, iPad and advanced smartphones, many new devices are able to switch from portrait to landscape at the user's whim. How is one to design for these situations?

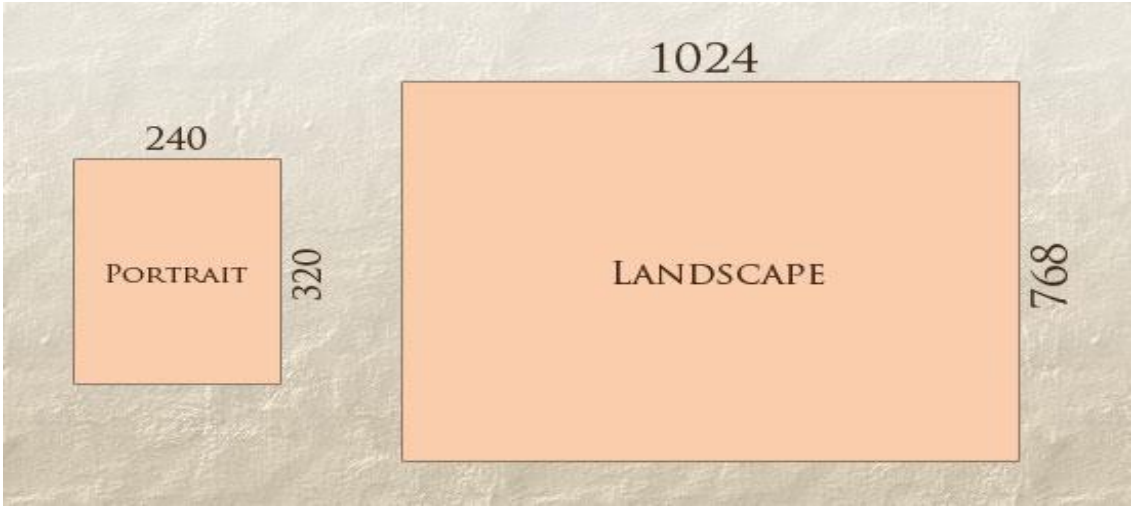


Figure 12: Adjusting Screen Resolution

In addition to designing for both landscape and portrait (and enabling those orientations to possibly switch in an instant upon page load), we must consider the hundreds of different screen sizes. Yes, it is possible to group them into major categories, design for each of them, and make each design as flexible as necessary. But that can be overwhelming, and who knows what the usage figures will be in five years? Besides, many users do not maximize their browsers, which itself leaves far too much room for variety among screen sizes.
Morten Hjerde and a few of his colleagues identified statistics on about 400 devices sold between 2005 and 2008. Below are some of the most common:

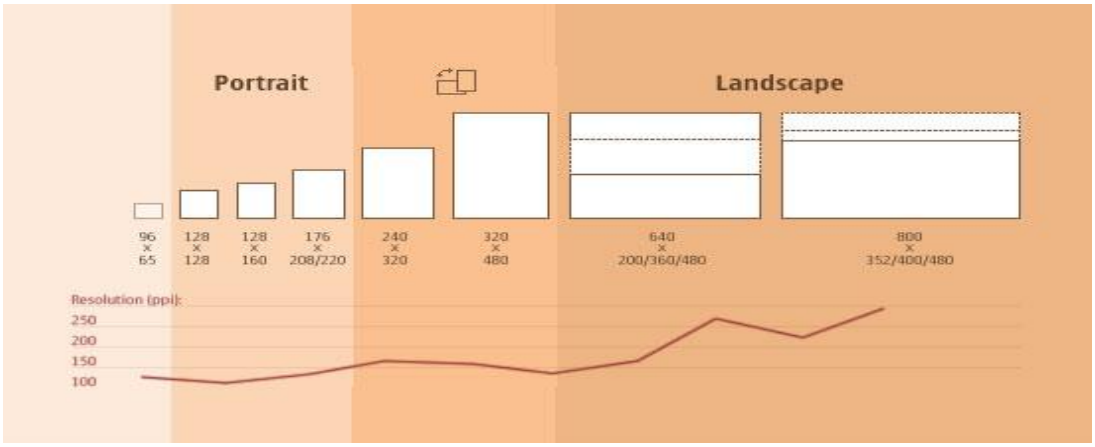


Figure 13: Screen Size

Since then even more devices have come out. It's obvious that we can't keep creating custom solutions for each one. So, how do we deal with the situation?

3.2 Media Queries with CSS

Media queries in CSS are a powerful tool that allows you to apply different styles and layouts based on the characteristics of the user's device or viewport. They enable you to create responsive designs that adapt to various screen sizes, resolutions, orientations, and even specific features of the device.

To interpret media queries in CSS, you typically use the @media rule followed by a media type or feature expression. Here's the basic syntax:

```
@media mediaType and (mediaFeature) {
/* CSS rules and styles */
}
```

Let's break down the different components:

a) **@media:** This is the keyword that signifies the start of a media query.

b) **mediaType:** It represents the general category of the device or media being targeted. Some commonly used media types include:

- all: Applies to all devices and media types.
- screen: Applies to devices with a screen, such as desktops, laptops, tablets, and smartphones.
- print: Applies when printing the document.
- speech: Applies to screen readers and other speech synthesis devices.

c) **mediaFeature:** It specifies the specific characteristics or conditions that must be met for the styles within the media query to apply. Here are a few examples of media features:

- width: Specifies the width of the viewport.
- height: Specifies the height of the viewport.
- orientation: Determines the orientation of the device (landscape or portrait).
- aspect-ratio: Represents the aspect ratio of the viewport.
- resolution: Indicates the pixel density of the output device.

You can combine multiple media features within a media query using logical operators like and, not, and only to create complex conditions. Here's an example:

```
@media screen and (max-width: 768px) and (orientation: landscape) {
/* CSS rules and styles for screens with a maximum width of 768px in landscape orientation */
}
```

In the example above, the styles within the media query will only be applied if the device has a screen, the viewport width is 768 pixels or less, and the orientation is landscape.

By utilizing media queries effectively, you can tailor your website or application's layout, typography, images, and other design elements to provide an optimal user experience across different devices and screen sizes.

3.3 Implementing Media Queries

Implementing media queries involves writing CSS rules within the appropriate media query blocks to target specific device characteristics or viewport conditions. CSS3 supports all of the same media types as CSS 2.1, such as screen, print and handheld, but has added dozens of new media features, including max-width, device-width, orientation and color. New devices made after the release of CSS3 (such as the iPad and Android devices) will definitely support media features. So, calling a media query using CSS3 features to target these devices would work just fine, and it will be ignored if accessed by an older computer browser that does not support CSS3.

n Ethan Marcotte's article, we see an example of a media query in action:

```
<link rel="stylesheet" type="text/css"
media="screen and (max-device-width: 480px)"
href="shetland.css" />
```

This media query is fairly self-explanatory: if the browser displays this page on a screen (rather than print, etc.), and if the width of the screen (not necessarily the viewport) is 480 pixels or less, then load shetland.css.

New CSS3 features also include orientation (portrait vs. landscape), device-width, min-device-width and more. Look at "The Orientation Media Query" for more information on setting and restricting widths based on these media query features.

One can create multiple style sheets, as well as basic layout alterations defined to fit ranges of widths — even for landscape vs. portrait orientations. Be sure to look at the section of Ethan Marcotte's article entitled "Meet the media query" for more examples and a more thorough explanation.

Multiple media queries can also be dropped right into a single style sheet, which is the most efficient option when used:

```
/* Smartphones (portrait and landscape) ---------- */
@media only screen
and (min-device-width : 320px)
```

```
and (max-device-width : 480px) {
/* Styles */
}

/* Smartphones (landscape) ---------- */
@media only screen
and (min-width : 321px) {
/* Styles */
}
/* Smartphones (portrait) ---------- */
@media only screen
and (max-width : 320px) {
/* Styles */
}
```

The code above is from a free template for multiple media queries between popular devices by Andy Clark. See the differences between this approach and including different style sheet files in the mark-up as described in his book "Hardboiled Web Design."

3.3.1 CSS3 MEDIA QUERIES:

Above are a few examples of how media queries, both from CSS 2.1 and CSS3 could work. Let's now look at some specific how-to's for using CSS3 media queries to create responsive Web designs. Many of these uses are relevant today, and all will definitely be usable in the near future.

The min-width and max-width properties do exactly what they suggest. The min-width property sets a minimum browser or screen width that a certain set of styles (or separate style sheet) would apply to. If anything is below this limit, the style sheet link or styles will be ignored. The max-width property does just the opposite. Anything above the maximum browser or screen width specified would not apply to the respective media query.

Note in the examples below that we're using the syntax for media queries that could be used all in one style sheet. As mentioned above, the most efficient way to use media queries is to place them all in one CSS style sheet, with the rest of the styles for the website. This way, multiple requests don't have to be made for multiple style sheets.

```
@media screen and (min-width: 600px) {
.hereIsMyClass {
width: 30%;
float: right;
```

```
}
}
```

The class specified in the media query above (hereIsMyClass) will work only if the browser or screen width is above 600 pixels. In other words, this media query will run only if the minimum width is 600 pixels (therefore, 600 pixels or wider).

```
@media screen and (max-width: 600px) {
.aClassforSmallScreens {
clear: both;
font-size: 1.3em;
}
}
```

Now, with the use of max-width, this media query will apply only to browser or screen widths with a maximum width of 600 pixels or narrower.

While the above min-width and max-width can apply to either screen size or browser width, sometimes we'd like a media query that is relevant to device width specifically. This means that even if a browser or other viewing area is minimized to something smaller, the media query would still apply to the size of the actual device. The min-device-width and max-device-width media query properties are great for targeting certain devices with set dimensions, without applying the same styles to other screen sizes in a browser that mimics the device's size.

```
@media screen and (max-device-width: 480px) {
.classForiPhoneDisplay {
font-size: 1.2em;
}
}

@media screen and (min-device-width: 768px) {
.minimumiPadWidth {
clear: both;
margin-bottom: 2px solid #ccc;
}
}
```

For the iPad specifically, there is also a media query property called orientation. The value can be either landscape (horizontal orientation) or portrait (vertical orientation).

```
@media screen and (orientation: landscape) {
.iPadLandscape {
width: 30%;
float: right;
}
}

@media screen and (orientation: portrait) {
.iPadPortrait {
clear: both;
}
}
```

Unfortunately, this property works only on the iPad. When determining the orientation for the iPhone and other devices, the use of max-device-width and min-device-width should do the trick.

There are also many media queries that make sense when combined. For example, the min-width and max-width media queries are combined all the time to set a style specific to a certain range.

```
@media screen and (min-width: 800px) and (max-width: 1200px) {
.classForaMediumScreen {
background: #cc0000;
width: 30%;
float: right;
}
}
```

The above code in this media query applies only to screen and browser widths between 800 and 1200 pixels. A good use of this technique is to show certain content or entire sidebars in a layout depending on how much horizontal space is available.

Some designers would also prefer to link to a separate style sheet for certain media queries, which is perfectly fine if the organizational benefits outweigh the efficiency lost. For devices that do not switch orientation or for screens whose browser width cannot be changed manually, using a separate style sheet should be fine.

You might want, for example, to place media queries all in one style sheet (as above) for devices like the iPad. Because such a device can switch from portrait to landscape in an instant, if these two media queries were placed in separate style sheets, the

website would have to call each style sheet file every time the user switched orientations. Placing a media query for both the horizontal and vertical orientations of the iPad in the same style sheet file would be far more efficient.

Another example is a flexible design meant for a standard computer screen with a resizable browser. If the browser can be manually resized, placing all variable media queries in one style sheet would be best.

Nevertheless, an organization can be key, and a designer may wish to define media queries in a standard HTML link tag:

```
<link rel="stylesheet" media="screen and (max-width: 600px)" href="small.css" />
<link rel="stylesheet" media="screen and (min-width: 600px)" href="large.css" />
<link rel="stylesheet" media="print" href="print.css" />
```

3.4 Responsive Approach to a Webpage

To apply a responsive approach to a webpage, you can follow these general steps:

- Design with Mobile-First in Mind: Start by designing your webpage for smaller screens first, considering the constraints of mobile devices. This approach ensures that your design focuses on essential content and functionality.

- Use Fluid Grids and Flexible Layouts: Instead of fixed-width layouts, employ fluid grids that adapt to different screen sizes. Use relative units like percentages and ems rather than pixels for sizing elements. This allows them to adjust proportionally as the viewport changes.

- Responsive Typography: Set font sizes, line heights, and other typographic properties using relative units. This enables the text to resize appropriately based on the viewport size.

- Responsive Images and Media: Use CSS techniques like max-width: 100% on images to ensure they scale down proportionally on smaller screens. For media elements like videos or embedded content, utilize responsive techniques or libraries to adapt their size and behavior based on the viewport.

- Media Queries: Utilize CSS media queries to define specific styles and layouts for different viewport sizes or device characteristics. Adjust the design as needed to provide an optimal user experience across various devices.

- Breakpoints: Identify breakpoints or specific screen sizes where the design needs to adapt significantly. Determine these breakpoints based on common device widths

or popular screen resolutions. Use media queries to apply specific styles at these breakpoints to ensure a smooth transition between different layouts.

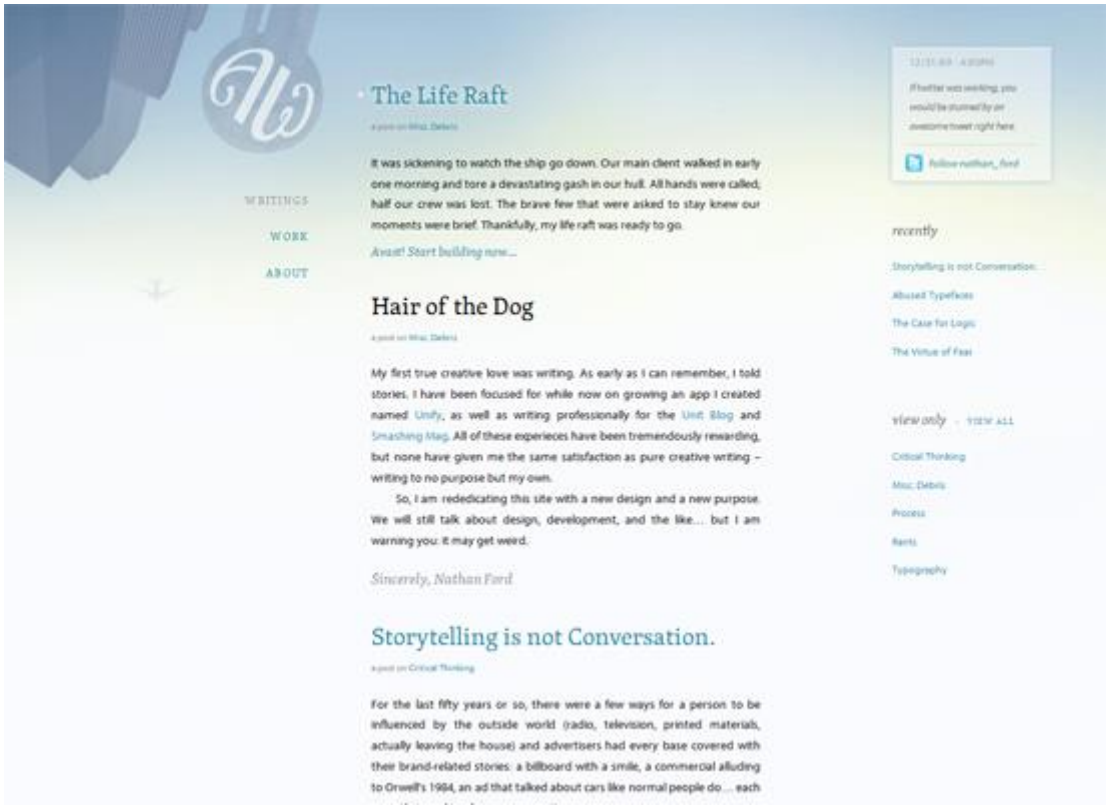
- Content Prioritization: Prioritize your content based on its importance and relevance. Make sure critical information is easily accessible on smaller screens. Consider rearranging or hiding less essential content to improve the user experience on mobile devices.

- Test and Iterate: Regularly test your webpage on different devices, screen sizes, and orientations. Make adjustments as necessary to ensure a consistent and optimized experience across various platforms. This iterative process helps fine-tune your responsive design.

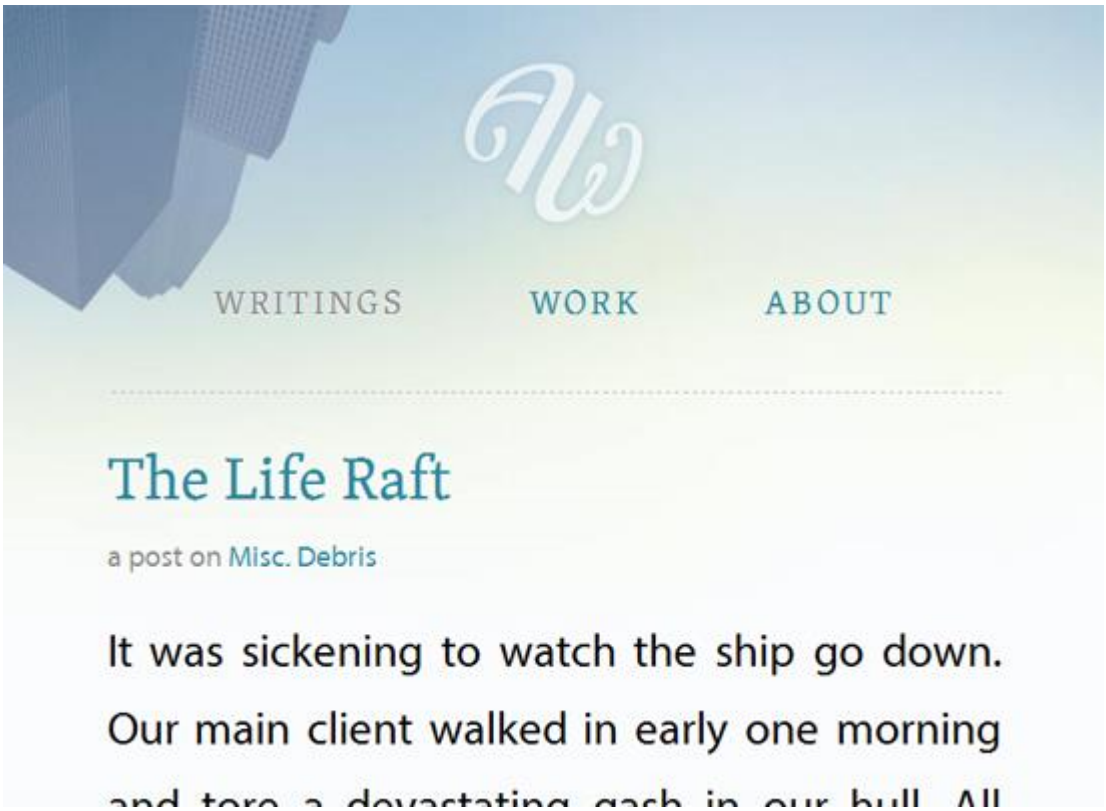
  Remember, implementing a responsive approach requires a combination of design principles, CSS techniques, and thorough testing. By focusing on flexibility, adaptability, and prioritizing user experience, you can create a webpage that provides an optimal viewing experience across a wide range of devices and screen sizes.

  Below we have a few examples of responsive Web design in practice today. For many of these websites, there is more variation in structure and style than is shown in the pairs of screenshots provided. Many have several solutions for a variety of browsers, and some even adjust elements dynamically in size without the need for specific browser dimensions. Visit each of these, and adjust your browser size or change devices to see them in action.

  Art Equals Work is a simple yet great example of responsive Web design. The first screenshot below is the view from a standard computer screen dimension. The website is flexible with browser widths by traditional standars, but once the browser gets too narrow or is otherwise switched to a device with a smaller screen, then the layout switches to a more readable and user-friendly format. The sidebar disappears, navigation goes to the top, and text is enlarged for easy and simple vertical reading.



Equals Work" website view from a standard computer screen dimension.



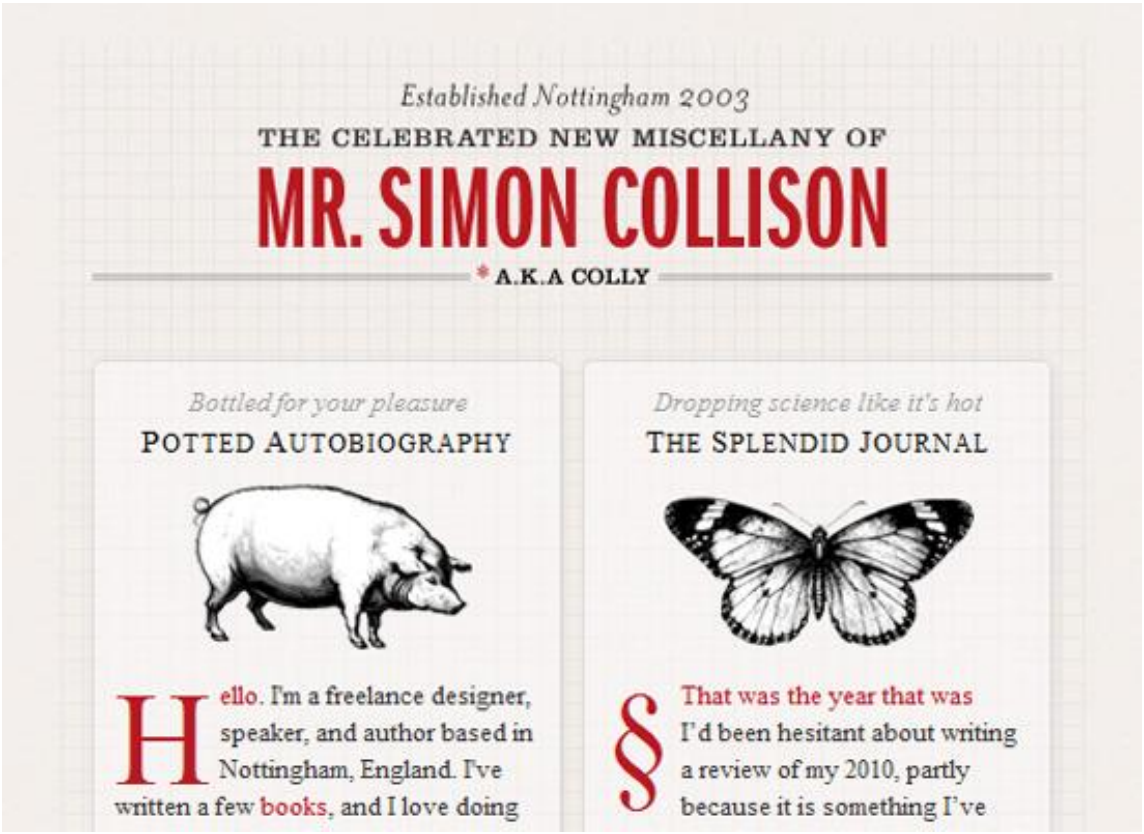
"Art Equals Work" website layout switches for simple vertical reading once the browser gets too narrow.

With Think Vitamin, we see a similar approach. When on a smaller screen or browser, the sidebar and top bar are removed, the navigation simplifies and moves directly above the content, as does the logo. The logo keeps its general look yet is modified for a more vertical orientation, with the tagline below the main icon. The white space around the content on larger screens is also more spacious and interesting, whereas it is simplified for practical purposes on smaller screens.

With four relatively content-heavy columns, it's easy to see how the content here could easily be squished when viewed on smaller devices. Because of the easy organized columns, though, we can also collapse them quite simply when needed, and we can stack them vertically when the space doesn't allow for a reasonable horizontal span. When the browser is minimized or the user is on a smaller device, the columns first collapse into two and then into one. Likewise, the horizontal lines for break points also change in width, without changing the size or style of each line's title text.


Established Nottingham 2003
THE CELEBRATED NEW MISCELLANY OF
MR. SIMON COLLISON
* A.K.A COLLY
POTTED AUTOBIOGRAPHY
THE SPLENDID JOURNAL
EXHAUSTIVE ARCHIVES
AVAILABLE FOR HIRE
EXTERNAL REFERENCES (VIEW ALL)
LANYRD PROFILE
NEW ADVENTURES
FLICKR PHOTOGRAPHS
TWITTER HAPPIER
LAST.FM SCROBBLES
PINBOARD BOOKMARKS
DRIBBBLE SHOTS
GOWALLA PASSPORT


THINK VITAMIN
THE WEB PRACTITIONER'S BLOG
Home
Membership
About
Online Conferences
Podcast
Archive
Write For Us
Communicating Freelance Development
4
By Nick Pellant
20 December 2010 | Category:
Business



As one can see, the main navigation here is the simple layout of t-shirt designs, spanning both vertically and horizontally across the screen. As the browser or screen gets smaller, the columns collapse and move below. This happens at each break point when the layout is stressed, but in between the break points, the images just change proportionally in size. This maintains balance in the design, while ensuring that any images (which are essential to the website) don’t get so small that they become unusable.

Self-Check Sheet 3: Use a responsive approach

1. What is a responsive layout?

   Answer:

2. Why is responsive design important?

   Answer:

3. What are media queries?

   Answer:

4. How do media queries work?

   Answer:

5. What is a mobile-first approach?

   Answer:

6. What are fluid grids?

   Answer:

7. How do media queries handle different screen resolutions?

   Answer:

8. What is the purpose of breakpoints in responsive design?

   Answer:

9. How can media queries be used to target specific devices?

   Answer:

10. What is the role of content prioritization in responsive design?

    Answer:

Answer Key 3: Use a responsive approach

1. What is a responsive layout?
   Answer: A responsive layout is a design approach that adapts to different screen sizes and devices to provide an optimal user experience

2. Why is responsive design important?
   Answer: Responsive design ensures that websites and applications are accessible and usable on various devices, improving user experience and engagement

3. What are media queries?
   Answer: Media queries are CSS rules that allow different styles and layouts to be applied based on the characteristics of the user's device or viewport.

4. How do media queries work?
   Answer: Media queries use conditional statements to evaluate device properties like screen width, height, and orientation, and apply specific styles accordingly

5. What is a mobile-first approach?
   Answer: A mobile-first approach involves designing and developing for mobile devices first, then progressively enhancing the design for larger screens

6. What are fluid grids?
   Answer: Fluid grids are flexible layouts based on relative proportions that allow elements to resize and reposition themselves according to the screen size

7. How do media queries handle different screen resolutions?
   Answer: Media queries can use the resolution media feature to target devices with specific pixel densities and adjust the styles accordingly

8. What is the purpose of breakpoints in responsive design?
   Answer: Breakpoints are specific points in a responsive design where the layout and content rearrange or change to better fit different screen sizes

9. How can media queries be used to target specific devices?
   Answer: Media queries can combine multiple conditions to target specific devices based on screen size, resolution, and other characteristics

10. What is the role of content prioritization in responsive design?
    Answer: Content prioritization ensures that essential information is readily accessible, even on smaller screens, while less important content may be rearranged or hidden

Job Sheet 3.1 Implement a Responsive Approach in CSS to Create a Web page

Objectives: In this task, you will learn and implement a responsive approach in CSS to create a web page that adapts and displays correctly on various devices and screen sizes.

Working Procedure:

Step 1: Set Up the Project

1. Create a new folder for the project and set up the basic file structure.
2. Inside the project folder, create an HTML file (index.html) and a CSS file (styles.css).
3. Link the CSS file to the HTML file using the <link> tag.

Step 2: Create the Base Layout

4. Start by designing the base layout of the web page using HTML and CSS.
5. Build a simple, fixed-width layout that looks good on larger screens.

Step 3: Implement Responsive Design

6. Introduce the concept of media queries. In your CSS file, create your first media query targeting smaller screens (e.g., mobile phones).
7. Modify the layout to make it more mobile-friendly, such as adjusting font sizes, spacing, and reducing column widths.
8. Test the changes by resizing the browser window to see how the layout adapts.

Step 4: Add More Breakpoints

9. Continue to add more media queries for different screen sizes (e.g., tablets and laptops).
10. Make appropriate adjustments to the layout and styles for each breakpoint to ensure a seamless user experience across devices.

Step 5: Use Relative Units

11. Replace fixed pixel values with relative units such as percentages, em, or rem to make elements adjust proportionally based on the screen size.

Step 6: Consider Touch Devices

12. Add touch-friendly features like increasing the size of buttons and links to improve usability on touchscreens.

Step 7: Test on Real Devices

13. Test your responsive web page on real devices, including smartphones, tablets, and laptops, to ensure compatibility and responsiveness.

Step 8: Debug and Optimize

14. Debug any layout issues or inconsistencies across devices and browsers.

15. Optimize your CSS and HTML to ensure efficient loading and smooth user experience.

Step 9: Documentation

16. Document the changes you made to the CSS file, explaining the media queries and responsive design techniques you used.

Learning Outcome 4: Use CSS Grid

Assessment Criteria	1. CSS grid is interpreted. 2. CSS Grid container is defined. 3. CSS grid items are identified. 4. CSS grid is applied.
Conditions and Resources	1. Applicable tools, utensils, and equipment as prescribed by competency standards. 2. Supply materials 3. Relevant ingredients 4. CBLM related to the learning outcome. 5. Instructions, job sheets, activity sheets, and standard operating procedures 6. Personal protective equipment 7. Module/reference
Contents	1 CSS Grid 2 CSS Grid Containers 3 CSS Grid Items 4 Applying CSS Grid
Learning Materials	1 CBLM 2 Handouts 3 Books, Manuals 4 Module/ Reference 5 Paper 6 Pen
Training Methods	1 Discussion 2 Presentation 3 Demonstration 4 Guided Practice 5 Individual Practice 6 Project Work 7 Problem Solving 8 Brainstorming
Assessment Methods	1 Written Test 2 Demonstration 3 Oral Questioning

Learning Experience 4: Use CSS Grid

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Steps	Resources specific instructions
1. Student will ask the instructor about work with CSS	1. Instructor will provide the learning materials use CSS grid
2. Read the **Information sheet/s**	2. Information Sheet No: 4: Use CSS Grid
3. Complete the **Self-Checks & Answer key sheets.**	**3.** Self-Check No: 4: Use CSS Grid Answer key No. 4: Use CSS Grid
4. Read the **Job/ Task sheet** and **Specification Sheet**	4. Job/ task sheet and specification sheet Job Sheet No: 4 Create a Responsive Web Layout using the power of Grid-Based Design

Information Sheet 4: Use CSS Grid

Learning Objective:

After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

4.1 Interpreting CSS Grid
4.2 Defining CSS Grid Containers
4.3 Identifying CSS Grid Items
4.4 Applying CSS Grid

4.1 Interpreting CSS Grid

CSS Grid is a powerful layout system in CSS that allows you to create complex grid-based layouts for web pages. When interpreting CSS Grid, there are several key concepts and properties to consider:

- **Container:** CSS Grid requires a container element that serves as the parent for the grid items. The container is defined using the display: grid; property. It establishes the grid formatting context for its child elements.

- **Grid Tracks:** CSS Grid divides the container into horizontal and vertical tracks. Tracks are the columns and rows of the grid. You can define the size and number of tracks using properties like grid-template-columns and grid-template-rows. You can specify sizes using fixed values (pixels, percentages) or flexible units (fr).

- **Grid Items:** Grid items are the child elements of the grid container. They are placed within the grid tracks and can span multiple tracks. To position grid items, you can use properties like grid-column-start, grid-column-end, grid-row-start, and grid-row-end. The shorthand property grid-column and grid-row can also be used for simplicity.

- **Grid Areas:** Grid areas are rectangular areas within the grid layout that can contain grid items. You can assign names to grid areas using the grid-template-areas property. By specifying the area names for grid items, you can easily position them within the grid.

- **Grid Lines:** Grid lines are the dividing lines between the tracks. They can be referenced using numerical indices or named lines. For example, you can refer to the first column using grid-column-start: 1; or to a named line using grid-column-start: start;.

- **Grid Template:** The grid template is a shorthand way to define the number of columns and rows, as well as their sizes and alignment. For example, grid-template: 1fr 2fr / 100px 1fr; creates two rows (1fr and 2fr) and two columns (100px and 1fr).

- **Grid Gap:** Grid gap refers to the space between grid tracks, both horizontally and vertically. You can control the size of the gap using properties like grid-gap, grid-column-gap, and grid-row-gap.

These are some of the fundamental concepts and properties in CSS Grid. Understanding these concepts will help you interpret and work with CSS Grid layouts effectively.

4.2 Defining CSS Grid Containers

To define a CSS Grid container, you need to follow these steps:

- Select the HTML element that will serve as the container for your grid layout. This could be a <**div**>, a <**section**>, or any other suitable element.

- Apply the **display**: **grid**; property to the selected container element. This sets the container as a grid formatting context and allows you to define grid-related properties.

- Optionally, specify the size and alignment of the grid tracks. You can use the **grid-template-columns** property to define the number, size, and alignment of the columns, and the grid-template-rows property to define the number, size, and alignment of the rows. Here's an example:

```
.container {
display: grid;
grid-template-columns: 1fr 1fr 1fr; /* Three columns with equal width */
grid-template-rows: 100px auto; /* First row with fixed height, second row takes remaining space */
}
```

In the above example, the container element is selected with the class .**container**. It is set as a grid using **display: grid**;. The **grid-template-columns** property creates three columns with an equal width of 1fr each. The **grid-template-rows** property sets the height of the first row to 100px and allows the second row to take the remaining space (auto).

- If needed, define named grid areas using the grid-template-areas property. This allows you to create a visual map of your grid layout. You can assign names to different areas and then assign those names to grid items. Here's an example:

```
.container {
display: grid;
grid-template-columns: 1fr 1fr;
grid-template-rows: auto;
```

```
grid-template-areas:
"header header"
"sidebar main"
"footer footer";
}
```

In this example, the grid areas are defined using the grid-template-areas property. The areas are named "header", "sidebar", "main", and "footer". You can then assign these area names to specific grid items.

- Place the grid items within the container by using properties like grid-column-start, grid-column-end, grid-row-start, and grid-row-end. You can also use shorthand properties like grid-column and grid-row. Here's an example:

```
.item {
grid-column: 1 / 3; /* Item spans from column 1 to 3 */
grid-row: 2; /* Item is placed in row 2 */
}
```

In this example, the .**item** class represents a grid item. The **grid-column** property is set to **1 / 3**, which means the item spans from column 1 to column 3. The **grid-row** property is set to 2, so the item is placed in row 2.

By following these steps, you can define a CSS Grid container and customize its layout according to your requirements.

4.3 Identifying CSS Grid Items

Identifying CSS Grid items involves selecting and applying styles to specific elements within the grid layout. Here's how you can identify and work with CSS Grid items:

- Select Grid Items: Identify the elements that you want to target as grid items within the grid container. These elements will be direct children of the grid container element.

- Apply Grid Item Styles: To apply styles specifically to grid items, you can use the element selector or a class selector along with additional CSS properties. Here's an example:

```
.container {
display: grid;
grid-template-columns: 1fr 1fr;
grid-template-rows: 100px auto;
```

```
}

.item {
background-color: lightblue;
padding: 10px;
}
```

In this example, the .container class represents the grid container, and the .item class is applied to the grid items. The .item class has specific styles like background-color and padding applied to it.

- Positioning Grid Items: To position grid items within the grid layout, you can use properties like grid-column-start, grid-column-end, grid-row-start, grid-row-end, or the shorthand properties grid-column and grid-row. These properties define where the grid item should start and end in terms of grid lines. For example:

```
.item {
grid-column-start: 1;
grid-column-end: 3;
grid-row-start: 2;
grid-row-end: 3;
}
```

In this case, the grid item with the .item class starts at column line 1, ends at column line 3, starts at row line 2, and ends at row line 3.

- Applying Grid Area Names: If you have defined named grid areas using the grid-template-areas property on the container, you can assign these area names to specific grid items. This provides an alternative way to position grid items. Here's an example:

```
.container {
display: grid;
grid-template-columns: 1fr 1fr;
grid-template-rows: 100px auto;
grid-template-areas:
"header header"
"sidebar main"
"footer footer";
}
```

```
.item {
grid-area: main;
}
```

In this example, the grid item with the .item class is assigned the main area using the grid-area property. This positions the item within the defined main area of the grid layout.

By identifying grid items and applying appropriate styles and positioning properties, you can control the appearance and placement of elements within a CSS Grid layout.

4.4 Applying CSS Grid

To apply CSS Grid to a web page layout, you need to follow these steps:

- Identify the Container: Select the HTML element that will serve as the container for your grid layout. This element will be the parent for all grid items. It could be a , a <section>, or any other suitable element.

- Apply Grid Display: Add the display: grid; property to the container element. This sets the container as a grid formatting context and enables the use of grid-related properties. For example:

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

- Define Grid Template Columns and Rows: Specify the size and number of columns and rows in the grid layout using the grid-template-columns and grid-template-rows properties. You can use fixed values (pixels, percentages) or flexible units (fr) to define the sizes. For example:

```
.container {
display: grid;
grid-template-columns: 1fr 1fr 1fr; /* Three columns with equal width */
grid-template-rows: auto 200px; /* First row with auto height, second row with fixed height of 200px */
}
```

- Position Grid Items: Place the grid items within the container using properties like grid-column-start, grid-column-end, grid-row-start, and grid-row-end. You can also use shorthand properties like grid-column and grid-row for simplicity. For example:

```
.item {
grid-column-start: 1;
grid-column-end: 3;
grid-row-start: 2;
grid-row-end: 4;
}
```

In this example, the. item class represents a grid item. It starts at the first column line, ends at the third column line, starts at the second row line, and ends at the fourth row line.

- Customize Grid Gaps: Optionally, adjust the space between grid tracks (columns and rows) using properties like grid-column-gap and grid-row-gap, or the shorthand property grid-gap. For example:

```
.container {
display: grid;
grid-template-columns: 1fr 1fr;
grid-template-rows: auto;
grid-column-gap: 20px; /* Adds a 20px gap between columns */
grid-row-gap: 10px; /* Adds a 10px gap between rows */
}
```

- Add Grid Item Styling: Apply specific styles to the grid items using class selectors or element selectors. You can customize the appearance, positioning, and other properties of the grid items. For example:

```
.item {
background-color: lightblue;
padding: 10px;
}
```

In this example, the **.item** class is applied to the grid items and has styles like background color and padding applied to it.

By following these steps, you can apply CSS Grid to create flexible and powerful grid layouts for your web page. Remember to adjust the grid template, position the grid items, and customize styles according to your design requirements.

Grid Container

Create a grid container by setting the display property with a value of grid or inline-grid. All direct children of grid containers become grid items.

display: grid

Grid items are placed in rows by default and span the full width of the grid container.

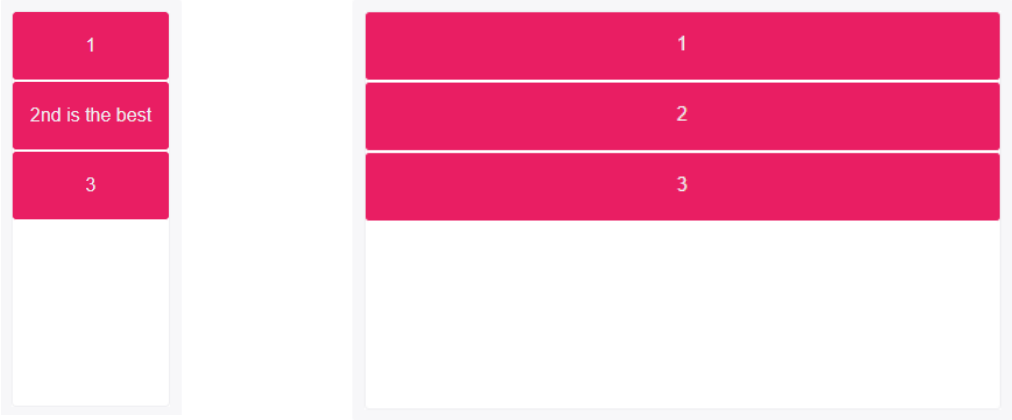
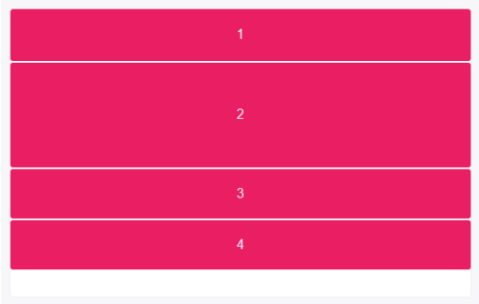


Figure 14: Display Grid

display: inline-grid:

Explicit Grid



Explicitly set a grid by creating columns and rows with the grid-template-columns and grid-template-rows properties.

grid-template-rows: 50px 100px

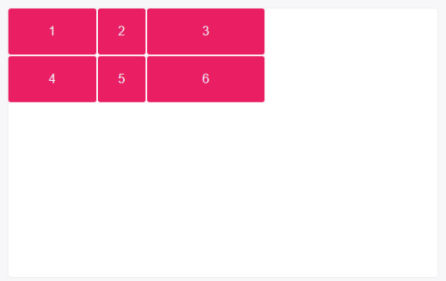
A row track is created for each value specified for grid-template-rows. Track size values can be any non-negative, length value (px, %, em, etc.)

Items 1 and 2 have fixed heights of 50px and 100px.

Because only 2 row tracks were defined, heights of items 3 and 4 are defined by the contents of each.

grid-template-columns: 90px 50px 120px

Like rows, a column track is created for each value specified for grid-template-columns.

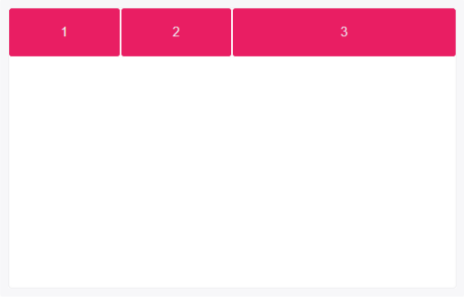


Items 4, 5 and 6 were placed on a new row track because only 3 column track sizes were defined; and because they were placed in column tracks 1, 2 and 3, their column sizes are equal to items 1, 2 and 3.

Grid items 1, 2 and 3 have fixed widths of 90px, 50px and 120px respectively.

grid-template-columns: 1fr 1fr 2fr

The fr unit helps create flexible grid tracks. It represents a fraction of the available space in the grid container (works like Flexbox's unitless values).



In this example, items 1 and 2 take up the first two (of four) sections while item 3 takes up the last two.

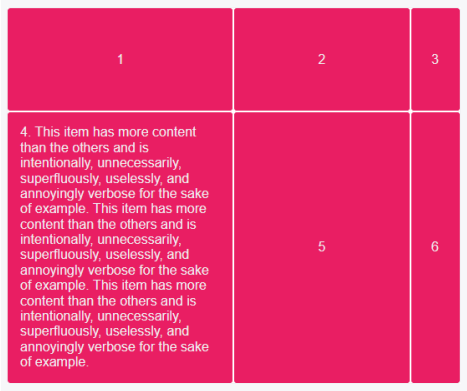
grid-template-columns: 3rem 25% 1fr 2fr

fr is calculated based on the remaining space when combined with other length values.

In this example, 3rem and 25% would be subtracted from the available space before the size of fr is calculated:
1fr = ((width of grid) - (3rem) - (25% of width of grid)) / 3

Minimum and Maximum Grid Track Sizes

Tracks sizes can be defined to have a minimum and/or maximum size with the minmax() function




```
grid-template-rows: minmax(100px, auto);
grid-template-columns: minmax(auto, 50%) 1fr 3em;
```

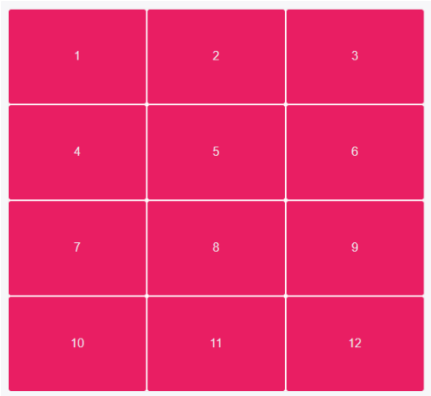
The minmax() function accepts 2 arguments: the first is the minimum size of the track and the second the maximum size. Alongside length values, the values can also be auto, which allows the track to grow/stretch based on the size of the content.

In this example, the first row track is set to have a minimum height of 100px, but its maximum size of auto will allow the row track to grow it the content is taller than 100px.

The first column track has a minimum size of auto, but its maximum size of 50% will prevent it from getting no wider than 50% of the grid container width.

Repeating Grid Tracks

Define repeating grid tracks using the repeat() notation. This is useful for grids with items with equal sizes or many items.

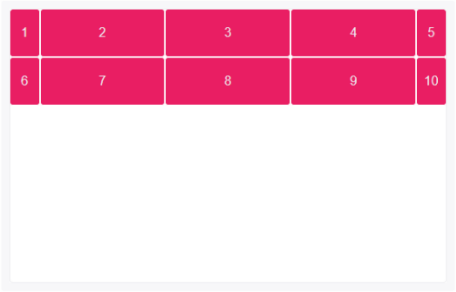



```
grid-template-rows: repeat(4, 100px);
grid-template-columns: repeat(3, 1fr);
```

The repeat() notation accepts 2 arguments: the first represents the number of times the defined tracks should repeat, and the second is the track definition.

grid-template-columns: 30px repeat(3, 1fr) 30px

repeat() can also be used within track listings.



In this example, the first and last column tracks have widths of 30px, and the 3 column tracks in between, created by repeat(), have widths of 1fr each.

Grid Gaps (Gutters)

The grid-column-gap and grid-row-gap properties create gutters between columns and rows.

Grid gaps are only created in between columns and rows, and not along the edge of the grid container.

Self-Check Sheet 4: Use CSS Grid

1. What is CSS Grid?
   Answer:

2. How do you define a CSS Grid container?
   Answer:

3. How can you specify the number of columns and rows in a Grid layout?
   Answer:

4. What is the purpose of using the “Grid-Gap” properly in CSS Grid?
   Answer:

5. How do you place items in specific grid cells using CSS grid?
   Answer:

6. What is the difference between “Grid-template-areas” and “Grid-template-columns” in CSS grid?
   Answer:

7. How can you create a responsive CSS grid layout?
   Answer:

8. What is the purpose of the “Justify-items and “Align-items” properties in CSS grid?
   Answer:

9. How do you create a nested grid within a CSS grid layout?
   Answer:

10. What are some browser compatibility considerations when using CSS grid?
    Answer:

Answer Key 4: Use CSS Grid

1. What is CSS Grid?
   Answer: CSS Grid is a layout system in CSS that allows you to create two-dimensional grid-based layouts for arranging content on a web page.

2. How do you define a CSS Grid container?
   Answer: To define a CSS Grid container, you use the display: grid; property on the parent element containing the grid items.

3. How can you specify the number of columns and rows in a Grid layout?
   Answer: You can use the grid-template-columns and grid-template-rows properties to specify the size of the columns and rows respectively.

4. What is the purpose of using the "Grid-Gap" properly in CSS Grid?
   Answer: The grid-gap property is used to create spacing between grid items and defines the gap between columns and rows in a CSS Grid layout.

5. How do you place items in specific grid cells using CSS grid?
   Answer: You can use the grid-column and grid-row properties to position grid items into specific cells within the grid.

6. What is the difference between "Grid-template-areas" and "Grid-template-columns" in CSS grid?
   Answer: Grid-template-areas defines named grid areas, while grid-template-columns defines the size of the columns in the grid.

7. How can you create a responsive CSS grid layout?
   Answer: You can use media queries and flexible units like percentages to adapt the grid layout to different screen sizes and devices.

8. What is the purpose of the "Justify-items and "Align-items" properties in CSS grid?
   Answer: justify-items controls the alignment of grid items along the inline (horizontal) axis, while align-items controls the alignment along the block (vertical) axis.

9. How do you create a nested grid within a CSS grid layout?
   Answer: By defining a grid item as another grid container using display: grid; you can create a nested grid within a CSS Grid layout.

10. What are some browser compatibility considerations when using CSS grid?
    Answer: CSS Grid is supported by all modern browsers, but for older versions of Internet Explorer, partial support is available using vendor prefixes like -ms-grid. Consider using fallbacks or alternative layouts for browsers that do not fully support CSS Grid.

Job Sheet 4.1 Create a Responsive Web Layout using the power of Grid-Based Design.

Objectives: In this task, you will learn and implement CSS Grid to create a responsive web layout using the power of grid-based design.

Working Procedure:

Step 1: Set Up the Project

1. Create a new folder for the project and set up the basic file structure.
2. Inside the project folder, create an HTML file (index.html) and a CSS file (styles.css).
3. Link the CSS file to the HTML file using the <link> tag.

Step 2: Define the Grid Container

4. Open the HTML file and create a container div element that will act as the grid container.
5. In the CSS file, set the container's display property to grid using the display property.

Step 3: Create Grid Columns and Rows

6. Decide on the number and size of columns and rows for your layout.
7. Use the grid-template-columns and grid-template-rows properties to define the layout structure. Experiment with different units (pixels, percentages, or fr units) to create a flexible design.

Step 4: Position Elements with Grid Lines

8. Add some content to the grid container by creating child elements (grid items).
9. Use the grid-column and grid-row properties to position the items within the grid using grid lines.

Step 5: Implement Responsive Behavior

10. Introduce media queries to create a responsive design. Target different screen sizes (e.g., mobile, tablet, desktop) using appropriate breakpoints.
11. Adjust the grid structure and item placement in each media query to ensure an optimal layout on various devices.

Step 6: Add Gaps and Spacing

12. Utilize the grid-gap property to add spacing between grid items. Experiment with different values (pixels, percentages, or em units) to achieve the desired visual effect.

Step 7: Test and Debug

13. Test your CSS Grid layout on different browsers and devices to ensure compatibility.
14. Debug any layout issues or inconsistencies across different scenarios.

Step 8: Documentation

15. Document the changes you made to the CSS file, explaining the CSS Grid properties used and how they contribute to the layout's responsiveness.

Review of Competency

Below is your assessment rating for module **Work with CSS**

Assessment of Performance Criteria	Yes	No
CSS (Cascading Style Sheets) is interpreted.		
Types of CSS are identified.		
Syntax of CSS is interpreted.		
Selector of CSS is interpreted.		
CSS file is created.		
CSS file is integrated.		
CSS is implemented as per layout.		
CSS box model and positioning is applied.		
CSS transition and gradients are applied.		
2D/3D transformation and animation is applied.		
Responsive layout is defined.		
Media Query is interpreted with CSS.		
Media query is implemented.		
Responsive approach is applied on a webpage.		
CSS grid is interpreted.		
CSS Grid container is defined.		
CSS grid items are identified.		
CSS grid is applied.		

I now feel ready to undertake my formal competency assessment.

Signed:

Date:

Reference

https://www.google.com/search?q=Data+Layer&tbm=isch&ved=2ahUKEwj0o9DW1tGBAxUUjmMGHaBZBEIQ2-cCegQIABAA&oq=Data+Layer&gs_lcp=CgNpbWcQAzIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgARQ1QZY7xNg0xZoAHAAeACAAdcBiAHXCJIBBTAuNC4ymAEAoAEBqgELZ3dzLXdpei1pbWfAAQE&sclient=img&ei=KLwXZfTvKpScjuMPoLORkAQ&bih=629&biw=1366&hl=en#imgrc=G30Pvdc79Lk2QM

https://www.google.com/search?q=XAMPP%2C+WAMP%2C+MAMP%2C+and+LAMP&sca_esv=569660528&hl=en&tbm=isch&source=hp&biw=1366&bih=629&ei=I7wXZc-IFfbi2roPyK2AwAI&iflsig=AO6bgOgAAAAAZRfKMy7gVqgROlsU4in5HyWY6hDrV-nv&ved=0ahUKEwiPponU1tGBAxV2sVYBHcgWACgQ4dUDCAc&uact=5&oq=XAMPP%2C+WAMP%2C+MAMP%2C+and+LAMP&gs_lp=EgNpbWciG1hBTVBQLCBXQU1QLCBNQU1QLCBhbmQgTEFNUEiBElCcBVjpD3ABeACQAQCYAa4BoAHMAqoBAzAuMrgBA8gBAPgBAvgBAYoCC2d3cy13aXotaW1nqAIA&sclient=img#imgrc=NhjcFF8drA0YHM

https://www.google.com/imghp?hl=en&tab=ri&ogbl

Development of CBLM:

The Competency Based Learning Material (CBLM) of '**Work with CSS (**Occupation: Web Design, Level-3**)** for National Skills Certificate is developed by NSDA with the assistance of SIMEC System, ECF consultancy & SIMEC Institute JV (Joint Venture Firm) in the month of June 2023 under the contract number of package SD-9A dated 07th May 2023.

Sl No.	Name & Address	Designation	Contact number
1	Khondoker Ali Asgor Pavel	Writer	01711 873 008
2	Fatema Tuj Johura Arzu	Editor	01912 464 747
3	Md. Amir Hossain	Co-Ordinator	01631 670 445
4	Md. Saif Uddin	Reviewer	01723 004 419