Web Application Programming

Week 2: Web Applications Front-End Development Continued

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com or jose@kumiuniversity.ac.ug

Agenda

1. Summary of previous lecture
2. CSS and HTML basics for responsive web page development
3. Responsive web design principles

Summary of Previous Lecture

1. Course Overview(Course Description, Objectives, Learning outcomes, A glance Course Content, Course requirements, Course length,Text Books and References)
2. Introduction to Web Apps (Definition, characteristics, types of web applications-dag deeper into PWAs, web application frameworks, steps to developing web Apps etc.)
3. HTML (Definition and looked at its components, sampled basic structure of html etc.) and
4. CSS basics(Defined CSS, why use it, looked at CSS syntax, CSS Selectors, Applied basic CSS onto html page)

More on CSS basics - CSS Selectors

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

We said we CSS selectors can divided into five categories:

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

This lecture will explain the Pseudo-Class selectors and Pseudo-Element selectors.

More on CSS basics - Pseudo-classes

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

1. Style an element when a user mouse over it
2. Style visited and unvisited links differently
3. Style an element when it gets focus

More on CSS basics -Pseudo-classes Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {
  property: value;
}
```

All CSS Pseudo Classes

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"

(https://www.w3schools.com/css/css_pseudo_classes.asp)

All CSS Pseudo Classes+

:last-child	p:last-child	Selects every <p> elements that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute specified

(https://www.w3schools.com/css/css_pseudo_classes.asp)

All CSS Pseudo Classes+

:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

(https://www.w3schools.com/css/css_pseudo_classes.asp)

More on CSS basics - Anchor Pseudo-classes

Links can be displayed in different ways:

Example

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}
```

```
/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

Note: `a:hover` MUST come after `a:link` and `a:visited` in the CSS definition in order to be effective!
`a:active` MUST come after `a:hover` in the CSS definition in order to be effective! Pseudo-class names are not case-sensitive.

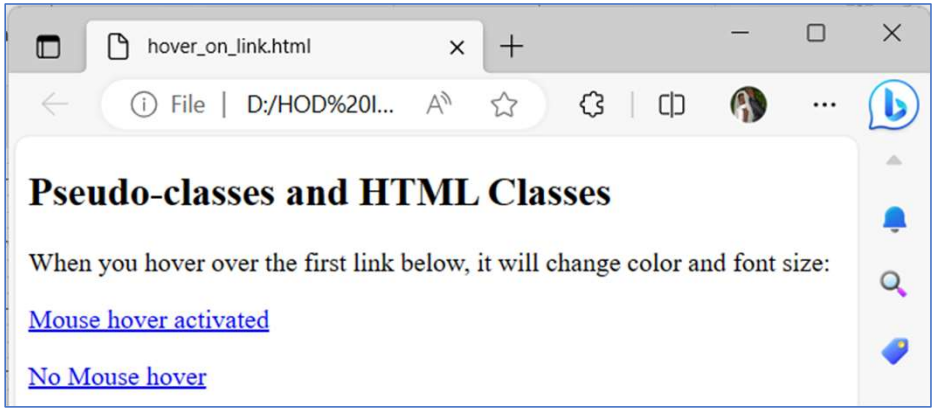
More on CSS basics - Anchor Pseudo-classes -hover_on_link.html

Homee.php | Internal Styles.html | myStyles.css | colorsValues.html | Responsive web page.html | hover_on_link.html

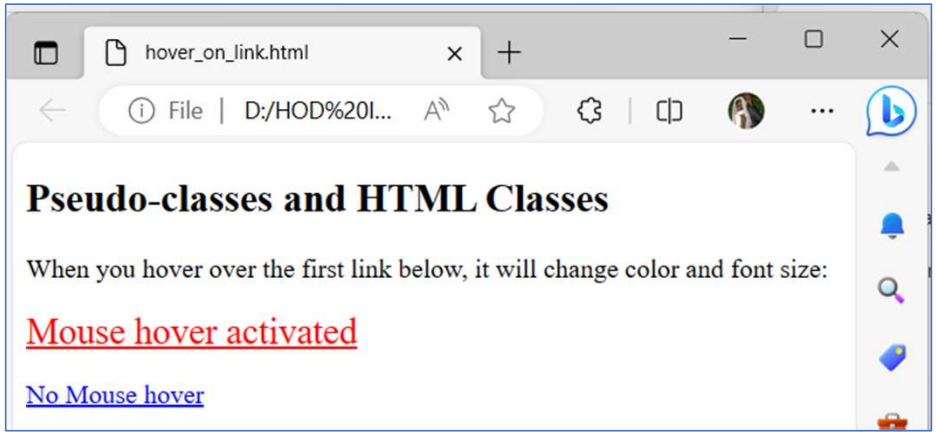
```
1  <!DOCTYPE html><html><head><style>
2  a.highlight:hover {
3    color: #ff0000;
4    font-size: 22px;
5  }
6  </style></head><body>
7
8  <h2>Pseudo-classes and HTML Classes</h2>
9
10 <p>When you hover over the first link below,
11 it will change color and font size:</p>
12
13 <p><a class="highlight" href="css_syntax.asp">Mouse hover activated</a></p>
14
15 <p><a href="default.asp">No Mouse hover</a></p>
16
17 </body></html>
```

More on CSS basics - Anchor Pseudo-classes -hover_on_link.html

Display before mouse hover



Display On mouse hover



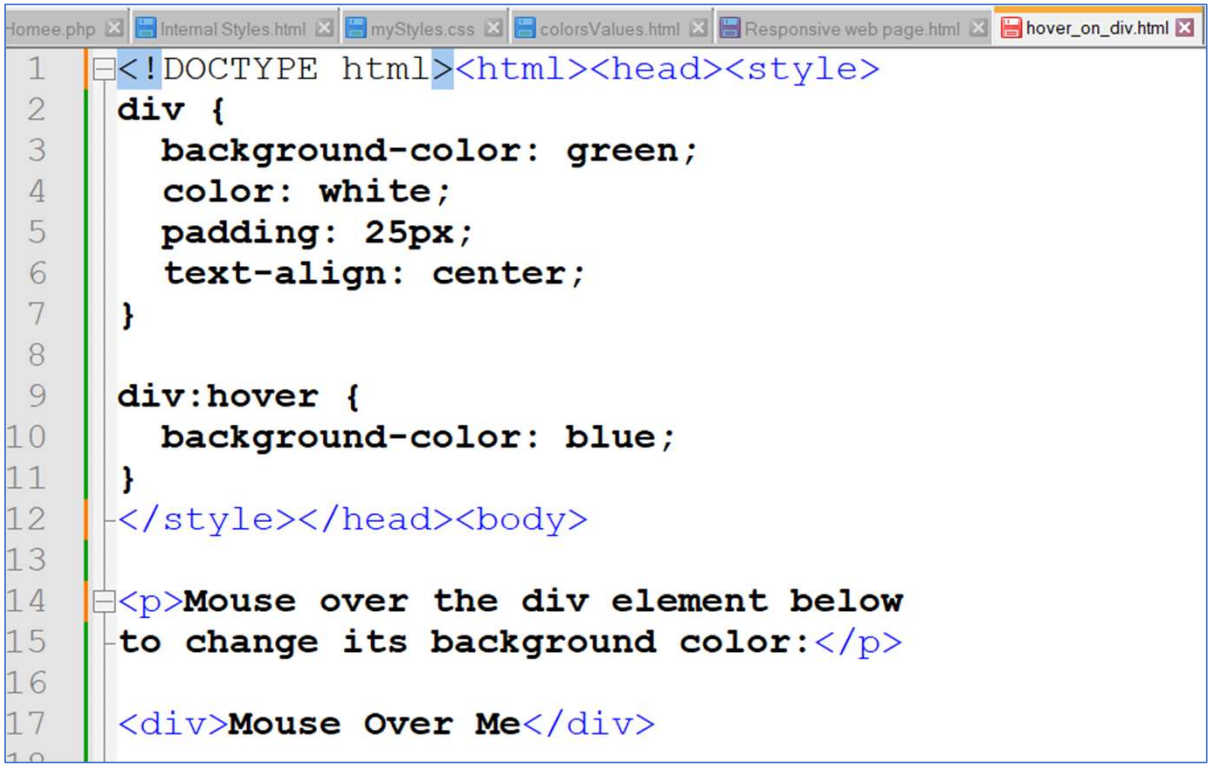
More on CSS basics - Pseudo-classes + Hover on a div

Example two

**Hover on **

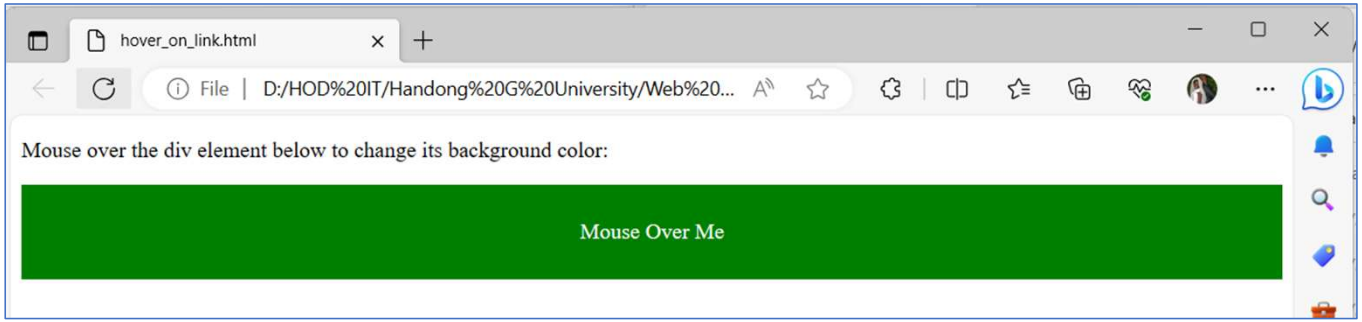
An example of using the :hover pseudo-class on a element:

```
div:hover {
  background-color: blue;
}
```

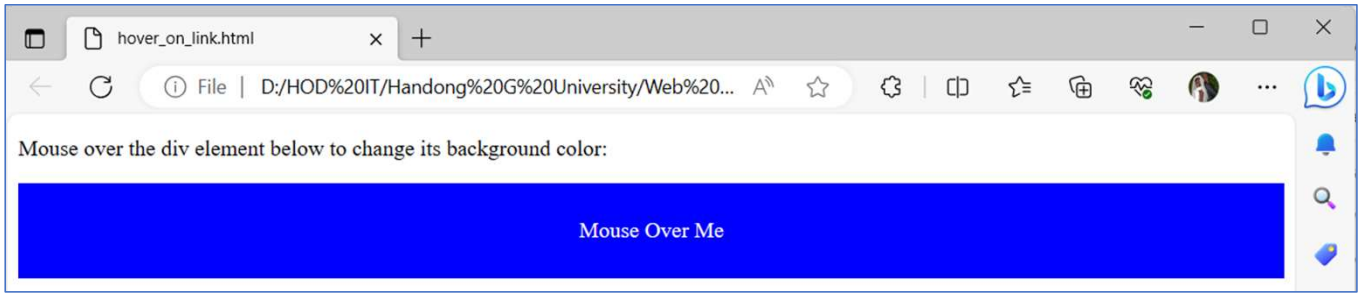


More on CSS basics - Pseudo-classes + Hover on a div code output

Display before mouse hover



Display on mouse hover



CSS Pseudo Elements Selectors

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

1. Style the first letter, or line, of an element
2. Insert content before, or after, the content of an element

Syntax

The syntax of pseudo-elements:

```
selector::pseudo-element {
  property: value;
}
```

All CSS Pseudo Elements

Below is the list of all Pseudo Elements.

Selector	Example	Example description
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::marker	::marker	Selects the markers of list items
::selection	p::selection	Selects the portion of an element that is selected by a user

(https://www.w3schools.com/css/css_pseudo_classes.asp)

CSS Pseudo Elements Selector example-PseudoElement.html file

The ::first-line Pseudo-element

The *::first-line* pseudo-element is used to add a special style to the first line of a text.

The following example formats the first line of the text in all <p> elements:

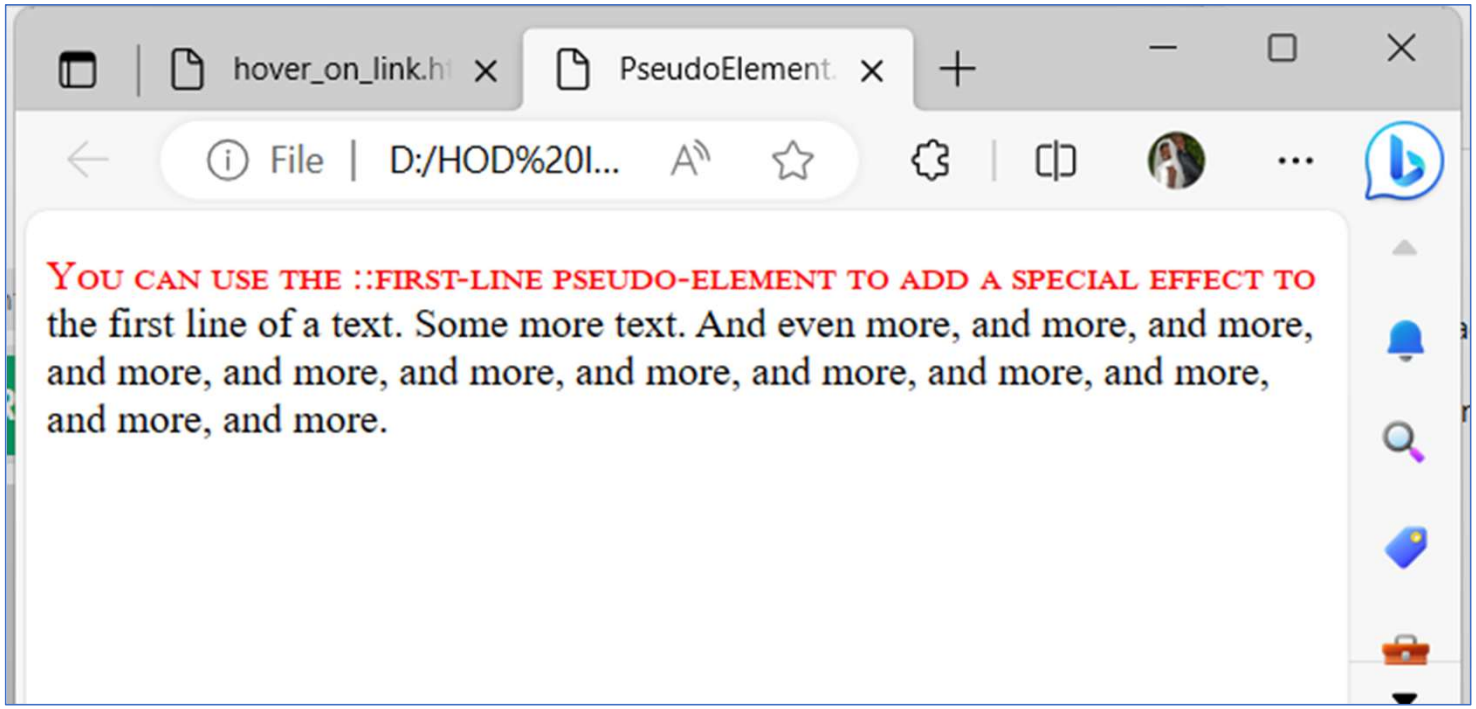
```
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
```

Example

Homee.php | Internal Styles.html | myStyles.css | colorsValues.html | Responsive web page.html | PseudoElement.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <style>
5  p::first-line {
6    color: #ff0000;
7    font-variant: small-caps;
8  }
9  </style>
10 </head>
11 <body>
12
13 <p>You can use the ::first-line pseudo-element
14 to add a special effect to the first line of a
15 text. Some more text. And even more, and more,
16 and more, and more, and more, and more, and more,
17 and more, and more, and more, and more, and more.</p>
18
19 </body>
20 </html>
```

CSS Pseudo Elements Selector example-PseudoElement.html Output



Properties of The ::first-line pseudo-elemen

Note: The ::first-line pseudo-element can only be applied to block-level elements.

The following properties apply to the ::first-line pseudo-element:

1. font properties
2. color properties
3. background properties
4. word-spacing
5. letter-spacing
6. text-decoration
7. vertical-align
8. text-transform
9. line-height
10. clear

Notice the double colon notation - ::first-line versus :first-line

The double colon replaced the single-colon notation for pseudo-elements in CSS3. This was an attempt from W3C to distinguish between **pseudo-classes** and **pseudo-elements**.

The single-colon syntax was used for both pseudo-classes and pseudo-elements in CSS2 and CSS1.

For backward compatibility, the single-colon syntax is acceptable for CSS2 and CSS1 pseudo-elements.

More on CSS basics - How to attach CSS to a web page

When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.

Ways to Insert CSS

There are three ways of inserting a style sheet:

1. Inline CSS
2. Internal/Document level CSS
3. External CSS

Attaching Inline CSS

An inline style may be used to apply a unique style for a single element. E.g.

<h1 style="color: blue; font-style: italic;">Kumi University</h1>

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property. Inline styles are defined within the "style" attribute of the relevant element:

```
<!DOCTYPE html>
<html>
<body>

<h1 style="color:blue; text-align:center;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>

</body>
</html>
```

Attaching Internal/Document level CSS

The internal style is defined inside the <style> element, inside the head section. An internal style sheet may be used if one single HTML page has a unique style.

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

body { background-color: linen;}
h1 { color: maroon; margin-left: 40px;}

</style>
</head>

<body>

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

</body>
</html>
```

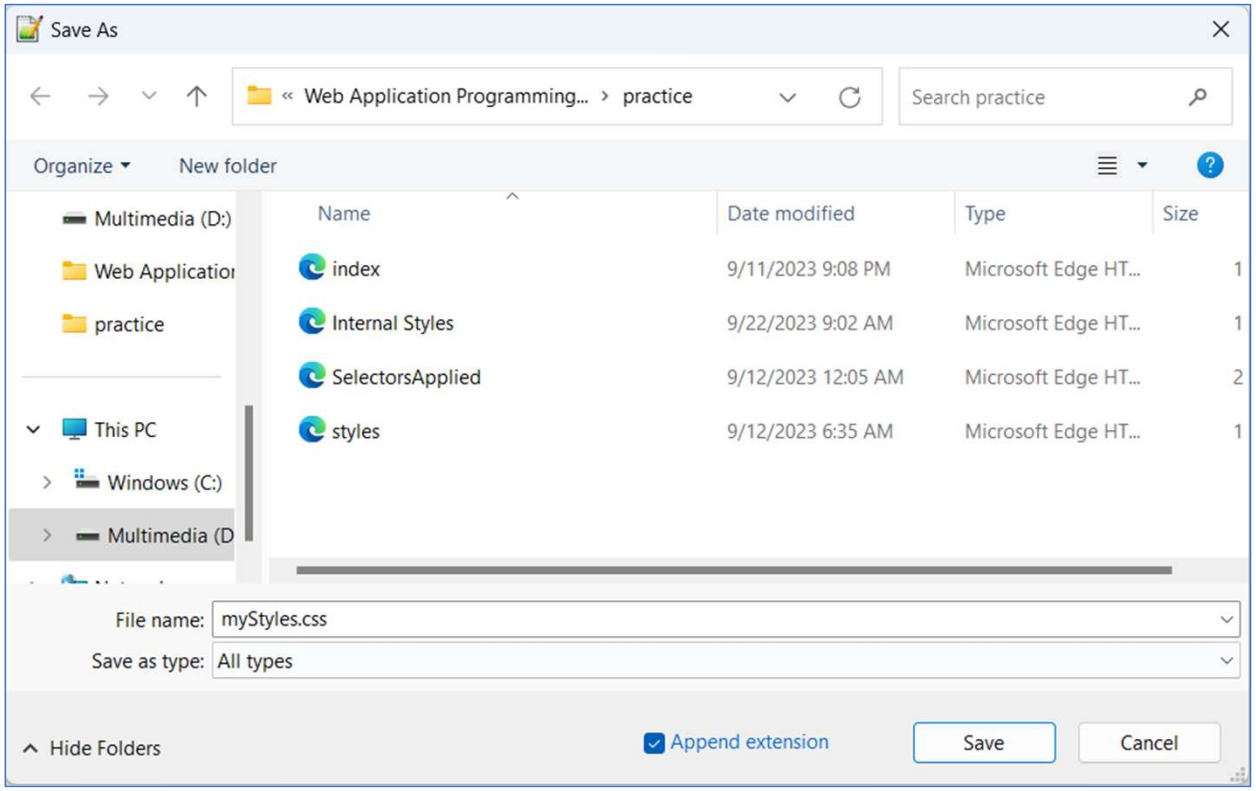
Attaching External CSS

With an external style sheet, one can change the look of an entire website by changing just one file!

Each HTML page must include a reference to the external style sheet file inside the <link> element/tag, inside the head section.

Attaching External CSS
Creating an external CSS file

1. Open Text editor
2. Create your styles
3. Save the file As e.g. myStyles.css (Click File, Select **Save As**, Browse storage area, Enter the name of the file e.g. **myStyles.css** in the File **Name Box**, Select File type as **All Types**, then click Save Button)



Attaching External CSS into html File –myStyles.css

Each HTML page must include a reference to the external style sheet file inside the <link> element/tag, inside the head section.

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

<link rel="stylesheet" href="mystyles.css">

</head>
<body>

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

</body>
</html>
```

Cascading Order

Which style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

Applying Multiple Style Sheets onto an html page

If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used. E.g.

1. Assume that an **external style sheet** has the following style for the **<h1>** element:

```
h1 {
  color: navy;
}
```

2. Then, assume that an **internal style sheet** also has the following style for the **<h1>** element:

```
h1 {
  color: orange;
}
```

Example 1

If the internal style is defined after the link to the external style sheet, the <h1> elements will be "orange":

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
<style>
h1 {
  color: orange;
}
</style>
</head>
```

Applying Multiple Style Sheets onto html page

If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used. E.g.

1. Assume that an **external style sheet** has the following style for the **<h1>** element:

```
h1 {
  color: navy;
}
```

2. Then, assume that an **internal style sheet** also has the following style for the **<h1>** element:

```
h1 {
  color: orange;
}
```

Example 2

However, if the internal style is defined before the link to the external style sheet, the <h1> elements will be "navy":

```
<head>
<style>
h1 {
  color: orange;
}
</style>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

Values - Lengths

- **Lengths** [a number + unit identifier]

  Unit identifier can be

  em (font size of the relevant font),

  ex (x-height of the relevant font),

  px (pixels),

  in (inches), **cm** (centimeter), **mm**,

  pt (points, =1/72 **in**), **pc** (picas, 1 pc = 12 pt)

  E.g.

```
h1 { margin: 0.5em }, h1 { margin: 1ex },
p { font-size: 12px }, h2 { line-height: 3cm }, h4
{ font-size: 12pt }, h4 { font-size: 1pc }
```

CSS Colors

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

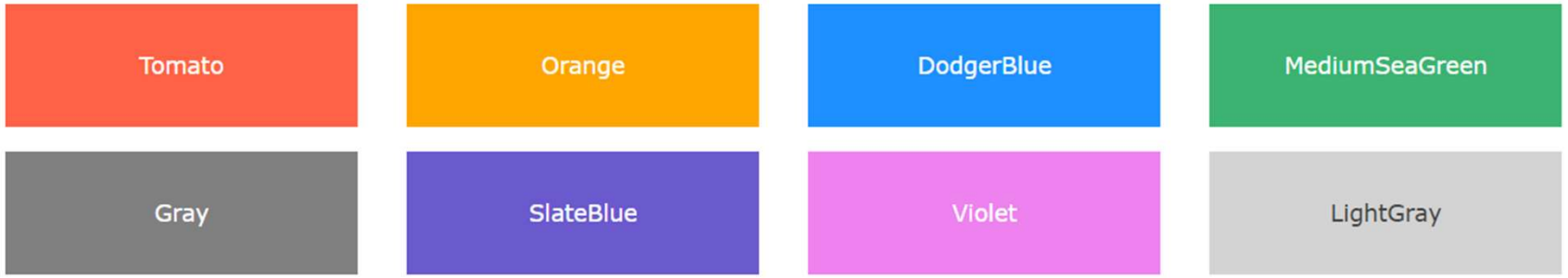
1. **RGB:** Red, Green, Blue **Example:** rgb(255, 0, 0)
2. **HEX:** Hexadecimal **Example**: #FF0000
3. **HSL:** Hue, Saturation, Lightness **Example:** hsl(0, 100%, 50%)
4. **RGBA**: Red, Green, Blue, Alpha (transparency) **Example:** rgba(255, 0, 0, 0.5)
5. **HSLA**: Hue, Saturation, Lightness, Alpha (transparency) **Example:** hsla(0, 100%, 50%, 0.5)

These color representations are commonly used in web design and development to specify colors for elements on a webpage or in a stylesheet.

CSS Colors+

Color Names

In CSS, a color can be specified by using a predefined color name



See all colors supported by CSS here (https://www.w3schools.com/colors/colors_names.asp)

CSS Colors+

Lets try it here- colors.html file

```
Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
mee.php | Internal Styles.html | myStyles.css | colors.html

1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <h1 style="background-color:Tomato;">Tomato</h1>
6  <h1 style="background-color:Orange;">Orange</h1>
7  <h1 style="background-color:DodgerBlue;">DodgerBlue</h1>
8  <h1 style="background-color:MediumSeaGreen;">MediumSeaGreen</h1>
9  <h1 style="background-color:Gray;">Gray</h1>
10 <h1 style="background-color:SlateBlue;">SlateBlue</h1>
11 <h1 style="background-color:Violet;">Violet</h1>
12 <h1 style="background-color:LightGray;">LightGray</h1>
13
14 </body>
15 </html>
```

Output

CSS Text and Border Color

You can set Text color as below

```
<h1 style="color:Tomato;">First color</h1>
<p style="color:DodgerBlue;">Text color based on color names</p>
<p style="color:MediumSeaGreen;">Done well</p>
```

Your can set Border Color as below

```
<h1 style="border:2px solid Tomato;">Border one</h1>
<h1 style="border:2px solid DodgerBlue;">Border two</h1>
```

CSS Color Values

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Same as color name "Tomato", but 50% transparent

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

CSS Color Values+Sample code+output

```
<!DOCTYPE html>
<html>
<body>

<p>Same as color name "Tomato":</p>

<h1 style="background-color:rgb(255, 99, 71);">rgb(255, 99, 71)</h1>
<h1 style="background-color:#ff6347;">#ff6347</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">hsl(9, 100%, 64%)</h1>

<p>Same as color name "Tomato", but 50% transparent:</p>
<h1 style="background-color:rgba(255, 99, 71, 0.5);">rgba(255, 99, 71, 0.5)</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">hsla(9, 100%, 64%, 0.5)</h1>

<p>In addition to the predefined color names,
colors can be specified using RGB, HEX, HSL,
or even transparent colors using RGBA or HSLA color values.</p>

</body>
</html>
```

Output

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Same as color name "Tomato", but 50% transparent:

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

In addition to the predefined color names, colors can be specified using RGB, HEX, HSL, or even transparent colors using RGBA or HSLA color values.

CSS Box Properties

To Manage information in a box-form well, one has to know which properties to deal with, which induced;

1. margin : <length>
2. border : <style><width><color>
3. padding : <length>
4. width & height : <length>

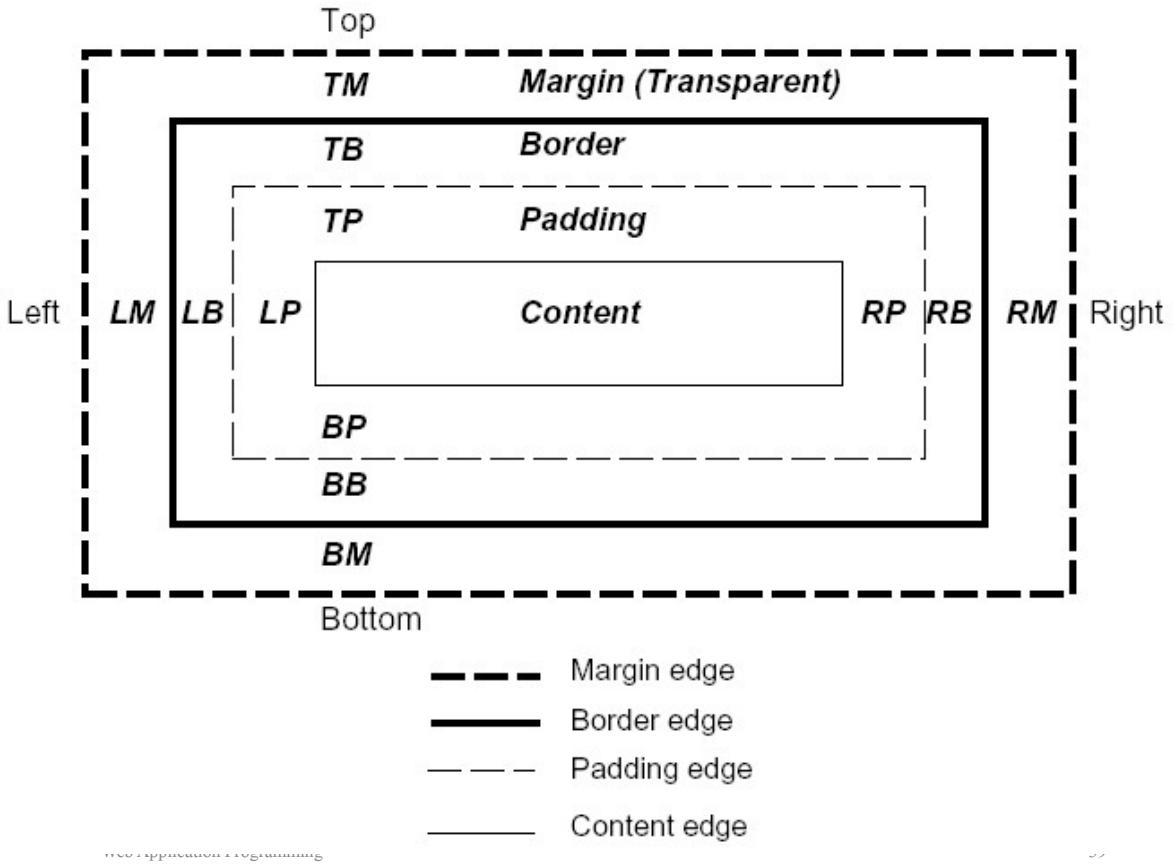
Examples:

```
p{
        margin: 50px;
        padding: 30px;
        float: right;
        height: 200px;
        width: 400px;
        border: 5px solid #006633;
}
```

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.

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:



CSS Box Model+

Explanation of the different parts:

1. **Content** - The content of the box, where text and images appear
2. **Padding** - Clears an area around the content. The padding is transparent
3. **Border** - A border that goes around the padding and content
4. **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.

CSS Box model++

Demonstration of the box model

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

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

CSS Box model+++
Calculating the width of the box

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 will have a total width of 350px:

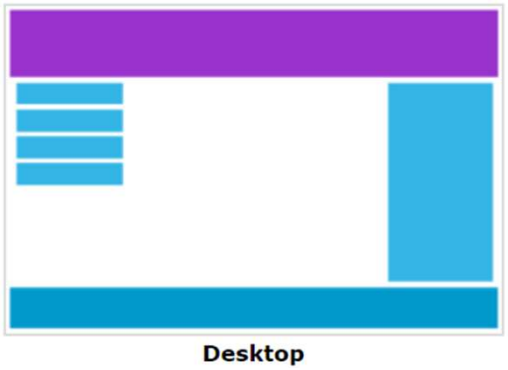
```
div {
  width: 320px;
  padding: 10px;
  border: 5px solid gray;
  margin: 0;
}
```

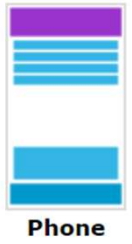
Calculation

320px (width)
+ 20px (left + right padding)
+ 10px (left + right border)
+ 0px (left + right margin)
= 350px

Responsive Web Design

Responsive web design is a kind of web application design that enables web pages/apps look good on all devices such as; desktops, tablets, and phones.





(https://www.w3schools.com/css/css_rwd_intro.asp)

Your web page should look good, and be easy to use, regardless of the device.

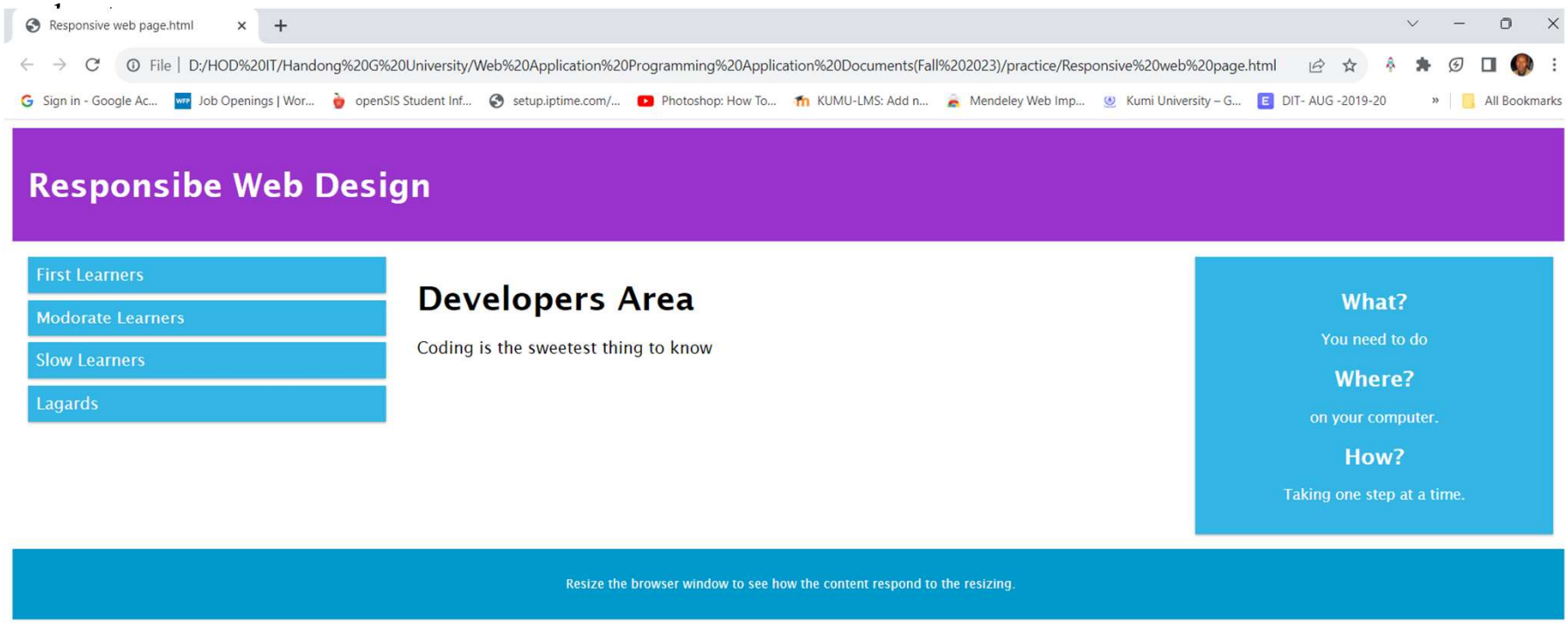
Responsive web design uses only HTML and CSS.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:

Responsive Web Design+

It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

Don't worry if you don't understand the example below, we will break down the code, step-by-step, in the next



Responsive web design principles

Responsive web application design is crucial for providing a consistent and user-friendly experience across various devices and screen sizes. Here are some key principles to consider when designing a responsive web application

1. **Mobile-First Approach:** Start designing for mobile devices first and then progressively enhance for larger screens. This ensures that your application is optimized for smaller screens and can adapt to larger ones.
2. **Fluid Grid Layout:** Use a fluid grid system (e.g., CSS Grid or Flexbox) that automatically adjusts the layout and spacing based on screen size. Avoid fixed-width layouts that can lead to horizontal scrolling on smaller screens.
3. **Media Queries:** Use media queries in your CSS to apply different styles and layouts based on the screen size and device characteristics. This allows you to create breakpoints where the design adjusts to fit the screen.

Responsive web design principles+

4. **Flexible Images and Media:** Ensure that images and media elements can scale or adapt to different screen sizes. Use responsive image techniques like srcset and picture elements to serve appropriately sized images based on the device's resolution.
5. **Progressive Enhancement:** Start with a solid foundation of HTML and CSS that works on all devices, and then add JavaScript enhancements as needed. This ensures that your application is functional even without JavaScript or on older devices.
6. **Touch-Friendly Design:** Consider touch gestures and interactions for mobile devices. Use larger touch targets, avoid hover-dependent interactions, and make sure the application is easy to navigate with touchscreens.
7. **Optimize Performance:** Pay attention to performance optimization by reducing unnecessary assets, compressing images, and minimizing HTTP requests. Faster loading times are especially critical on mobile devices with slower connections.

Responsive web design principles++

8. **Content Prioritization:** Prioritize and organize content based on its importance. Consider what information should be displayed prominently on smaller screens and what can be hidden or shown on demand.
9. **Consistent Navigation:** Maintain a consistent navigation structure across different screen sizes. Use responsive navigation menus, such as hamburger menus or off-canvas menus, for mobile devices.
10. **Testing Across Devices:** Test your responsive design on various real devices and screen sizes, including smartphones, tablets, laptops, and desktops. Consider using browser developer tools and emulators to simulate different devices.
11. **User Feedback and Usability Testing:** Gather feedback from users and conduct usability testing to identify any issues with the responsive design and make necessary improvements.

Responsive web design principles+++

12. **Accessibility:** Ensure that your web application is accessible to users with disabilities. Use semantic HTML, provide alternative text for images, and follow accessibility guidelines (e.g., WCAG) to make your application inclusive.
13. **Cross-Browser Compatibility:** Test your application on different web browsers (e.g., Chrome, Firefox, Safari, Edge) to ensure compatibility and consistent rendering.
14. **Performance Monitoring:** Continuously monitor your application's performance on different devices and screen sizes, and make adjustments as needed to maintain optimal speed and usability.
15. **Scalability:** Design your application to be scalable, allowing it to grow and adapt to changing requirements and new devices without requiring a complete redesign.

By following these responsive web application design principles, you can create a user-friendly and adaptable interface that provides a seamless experience across a wide range of devices and screen sizes.

Responsive Web App Dev't+ Viewport

The viewport is the user's visible area of a web page. The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, **web pages were designed only for computer screens**, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

Responsive Web App Dev't+
Setting the Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the **<meta>** tag.

You should include the following **<meta>** viewport element in all your web pages:

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

This gives the browser instructions on how to control the page's dimensions and scaling.

The **width=device-width** part sets the width of the page to follow the screen-width of the device (which varys depending on the device).

The **initial-scale=1.0** part sets the initial zoom level when the page is first loaded by the browser.

Below is an example of a web page without the viewport meta tag, and the same web page with the viewport meta tag:

Responsive Web App Dev't+ Setting the Viewport+

Here is an example of a web page without the viewport meta tag, and the same web page with the viewport meta tag:

Without the viewport meta tag

Note! the the difference in the way the image and the text balances the screen of the two codes

With the viewport meta tag

(https://www.w3schools.com/css/css_rwd_viewport.asp)

Responsive Web App Dev't+
Size Content to The Viewport

Users are used to scroll websites vertically on both desktop and mobile devices - but not horizontally!

So, if the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience.

Some additional rules to follow:

1. Do NOT use large fixed width elements - For example, if an image is displayed at a width wider than the viewport it can cause the viewport to scroll horizontally. Remember to adjust this content to fit within the width of the viewport.

2. Do NOT let the content rely on a particular viewport width to render well - Since screen dimensions and width in CSS pixels vary widely between devices, content should not rely on a particular viewport width to render well.

3. Use CSS media queries to apply different styling for small and large screens - Setting large absolute CSS widths for page elements will cause the element to be too wide for the viewport on a smaller device. Instead, consider using relative width values, such as width: 100%. Also, be careful of using large absolute positioning values. It may cause the element to fall outside the viewport on small devices.

Responsive Web Design - Grid-View

What is a Grid-View?

Many web pages are based on a grid-view, which means that the page is divided into columns:

12 Columns

Responsive Web Design - Grid-View

Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.

A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window

Building a Responsive Grid-View – build_rwp.html

Lets start building a responsive grid-view.

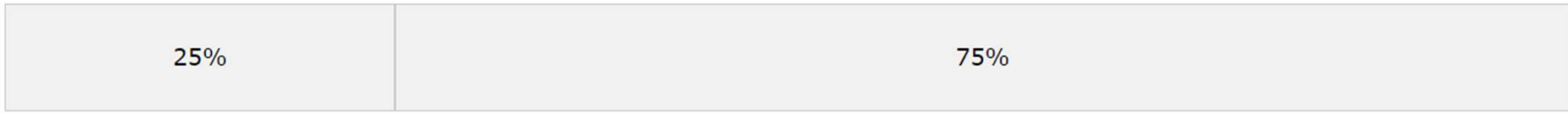
First ensure that all HTML elements have the box-sizing property set to border-box. This makes sure that the padding and border are included in the total width and height of the elements.

Add the following code in your CSS:

```
* {
  box-sizing: border-box;
}
```

Building a Responsive Grid-View+ build_rwp.html

The following example shows a simple responsive web page, with two columns:




```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```

The example above is fine if the web page only contains two columns. However, we want to use a responsive grid-view with 12 columns, to have more control over the web page.

Building a Responsive Grid-View++ On a page with 12 Grids

First we must calculate the percentage for one column: 100% / 12 columns = 8.33%.

Then we make one class for each of the 12 columns, class="col-" and a number defining how many columns the section should span as seen below;

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

NOTE! All these columns should be floating to the left, and have a padding of 15px:

```
[class*="col-"] {
  float: left;
  padding: 15px;
  border: 1px solid red;
}
```

Building a Responsive Grid-View++

Each row should be wrapped in a . The number of columns inside a row should always add up to 12:

```
<div class="row">
  <div class="col-3">...</div> <!-- 25% -->
  <div class="col-9">...</div> <!-- 75% -->
</div>
```

The columns inside a row are all floating to the left, and are therefore taken out of the flow of the page, and other elements will be placed as if the columns do not exist. To prevent this, we will add a style that clears the flow:

```
.row::after {
  content: "";
  clear: both;
  display: table;
}
```

Building a Responsive Grid-View+++

We also want to add some styles and colors to make it look better:

```
html {
  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

```
.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color :#33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0
1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
  background-color: #0099cc;
}
```

Building a Responsive Grid-View+ build_rwp.html full code

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}

.row::after {
  content: "";
  clear: both;
  display: table;
}

[class*="col-"] {
  float: left;
  padding: 15px;
}

.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
```

Building a Responsive Grid-View+ build_rwp.html full code

```
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

html {
  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
  color: #ffffff;
```

Building a Responsive Grid-View+ build_rwp.html full code

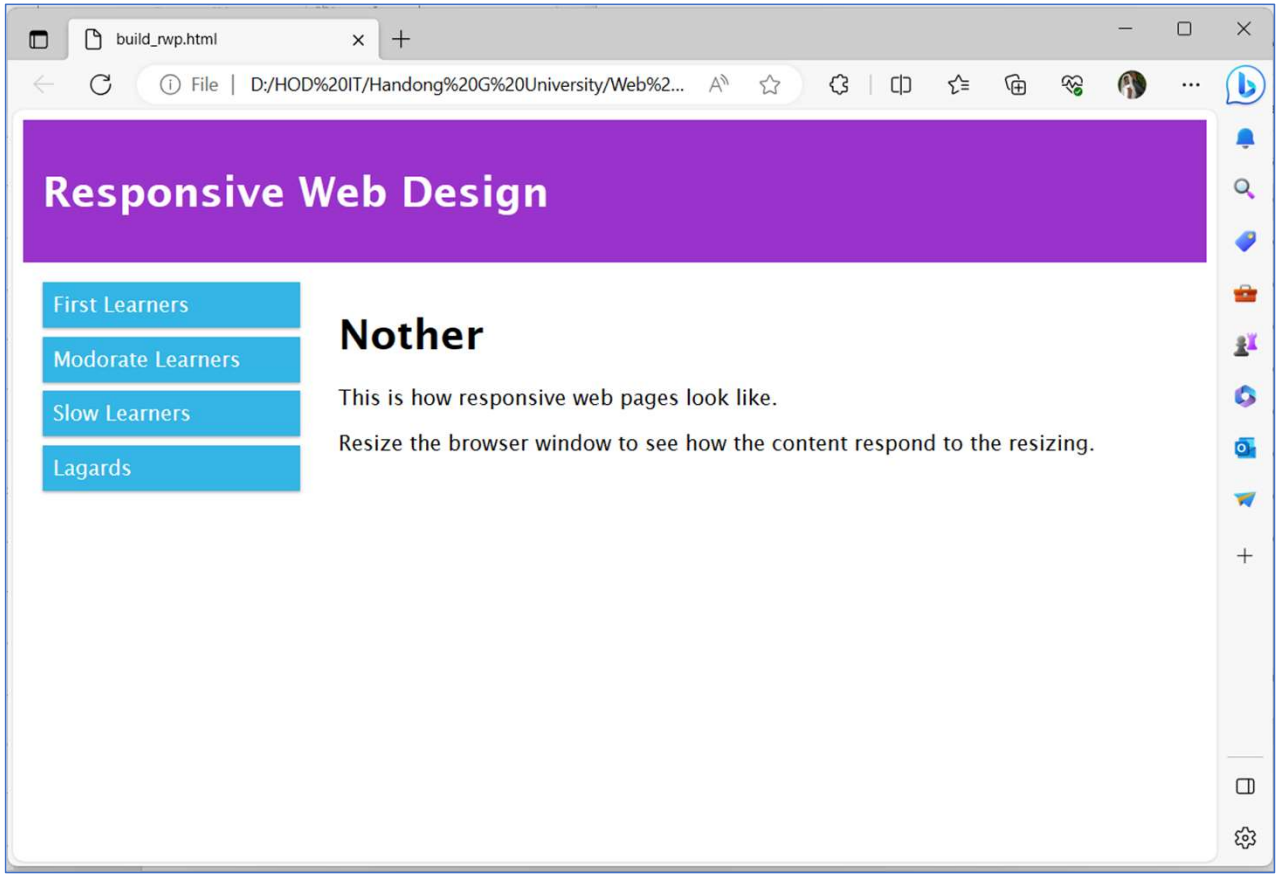
```
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
    background-color: #0099cc;
}
</style>
</head>
<body>

<div class="header">
    <h1>Responsive Web Design</h1>
</div>

<div class="row">
    <div class="col-3 col-s-3 menu">
        <ul>
            <li>First Learners</li>
            <li>Modorate Learners</li>
            <li>Slow Learners</li>
            <li>Lagards </li>
        </ul>
    </div>
```

Building a Responsive Grid-View+ build_rwp.html full code

```
79      <div class="col-9">
80        <h1>Nother</h1>
81        <p>This is how responsive web pages look like.</p>
82        <p>Resize the browser window to see how the content respond to the resizing.</p>
83      </div>
84    </div>
85
86    </body></html>
```

End of Code

Building a Responsive Grid-View+ build_rwp.html full code+Output

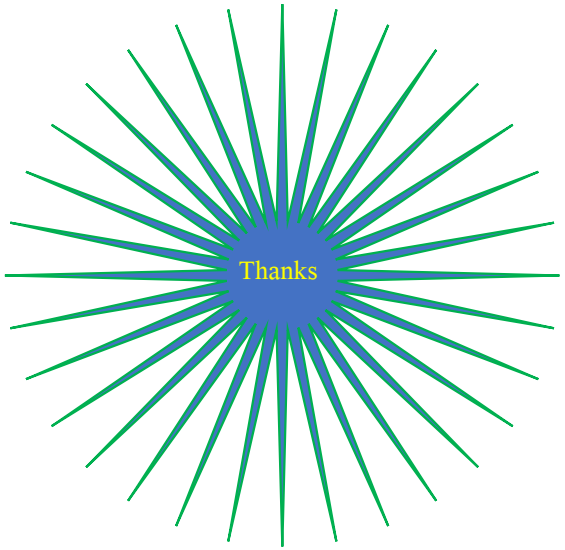


Summary

Summary

1. CSS basics for responsive web page development(Pseudo-class selectors, Pseudo-elements selectors, looked at how to attach CSS onto an html file etc.)
2. Responsive web design principles(Mobile-First Approach, Fluid Grid Layout, Media Queries etc. and actually built a responsive web page)

Thank you for Listening



References

Web app development in 2023 Brewster, C. (2023). : Everything you need to know. Trio. https://www.trio.dev/front-end/resources/web-app-development

Advantages and disadvantages of Web applications OpenAI. (2021, September 11). . OpenAI Knowledge Base.

HTML introduction. Introduction to HTML. (n.d.). https://www.w3schools.com/html/html_intro.asp

CSS introduction. (n.d.). https://www.w3schools.com/css/css_intro.asp