Background and Overview:

What is a web server?

The term web server can refer to hardware or software, or both of them working together.

On the hardware side, a web server is a computer that stores web server software and a website's component files (for example, HTML documents, images, CSS stylesheets, and JavaScript files). A web server connects to the Internet and supports physical data interchange with other devices connected to the web.

On the software side, a web server includes several parts that control how web users access hosted files. At a minimum, this is an HTTP server. An HTTP server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages). An HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

At the most basic level, whenever a browser needs a file that is hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct (hardware) web server, the (software) HTTP server accepts the request, finds the requested document, and sends it back to the browser, also through HTTP. (If the server doesn't find the requested document, it returns a 404 response instead.) To publish a website, you need either a static or a dynamic web server.

A static web server, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files as-is to your browser.

A dynamic web server consists of a static web server plus extra software, most commonly an application server and a database. We call it "dynamic" because the application server updates the hosted files before sending content to your browser via the HTTP server.

The language of the web:

Hypertext Markup Language, or HTML, uses tags that enclose plain text. The tags describe how the text should appear and the function of the text. The web browser looks at the tags and displays them accordingly. A simple example of HTML text is: Do you want to have lunch? **HTML (Hypertext Markup Language):**

HTML is the code that is used to structure a web page and its content.

What is HTML?

HTML is the standard markup language for Web pages. With HTML you can create your own website.

HTML (Hyper Text Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page's appearance/presentation (CSS) or functionality/behavior (JavaScript).

"Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web.

Content of HTML:

HTML uses "markup" to annotate text, images, and other content for display in a Web browser. HTML markup includes special "elements" such as <head>, <title>, <body>, <header>, <footer>, <article>,

<section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li> and many others.

An HTML element is set off from other text in a document by "tags", which consist of the element name surrounded by "<" and ">". The name of an element inside a tag is case insensitive. That is, it can be written in uppercase, lowercase, or a mixture. For example, the <title> tag can be written as <Title>, <TITLE>, or in any other way.

HTML Styles – CSS

CSS stands for Cascading Style Sheets.

CSS is the language we use to style an HTML document.

CSS describes how HTML elements should be displayed.

CSS saves a lot of work. It can control the layout of multiple web pages all at once.

What is CSS?

Cascading Style Sheets (CSS) is used to format the layout of a webpage.

With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more!

HTML Forms:

An HTML form is used to collect user input. The user input is most often sent to a server for processing.

The <form> Element:

The HTML <form> element is used to create an HTML form for user input.

The <form> element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.

The <input> Element:

The HTML <input> element is the most used form element.

An <input> element can be displayed in many ways, depending on the type attribute.

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)

<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

Text Fields:

The <input type="text"> defines a single-line input field for text input.

```
<form>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname">
</form>
```

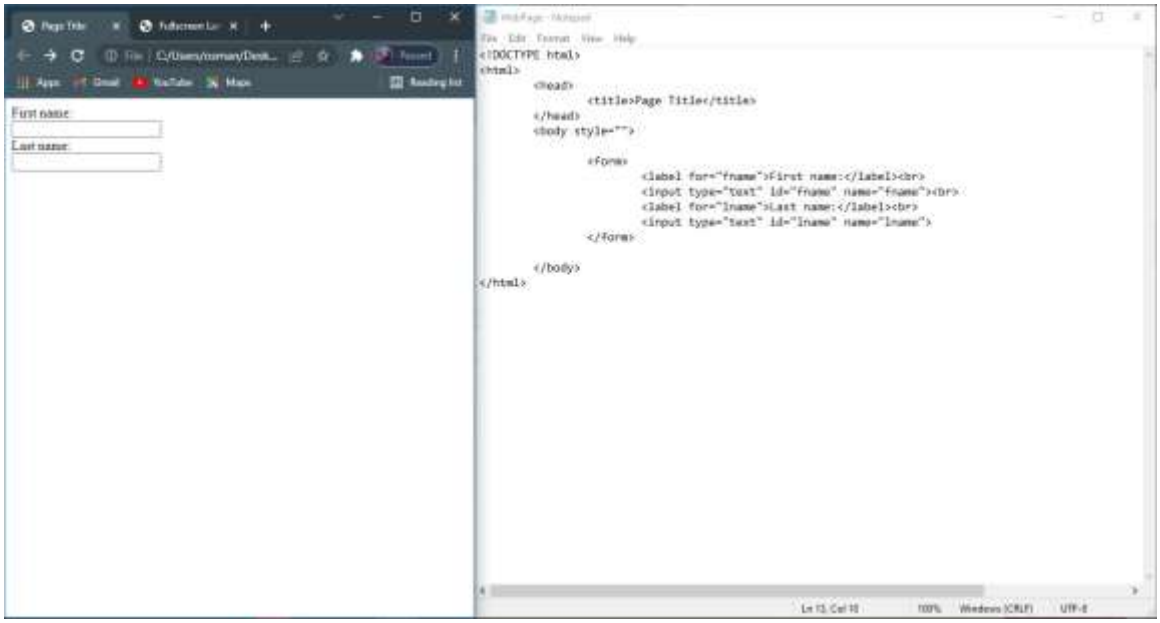


Fig. 1 (Input Field Form)

The form itself is not visible. Also note that the default width of an input field is 20 characters.

The <label> Element:

Notice the use of the <label> element in the example above.

The <label> tag defines a label for many form elements.

The <label> element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.

The <label> element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the <label> element, it toggles the radio button/checkbox.

The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

Radio Buttons:

The <input type="radio"> defines a radio button.
Radio buttons let a user select ONE of a limited number of choices.

```
<form>
        <input type="radio" id="html" name="fav_language" value="HTML">
        <label for="html">HTML</label><br>
        <input type="radio" id="css" name="fav_language" value="CSS">
        <label for="css">CSS</label><br>
        <input type="radio" id="javascript" name="fav_language" value="JavaScript">
        <label for="javascript">JavaScript</label> </form>
```

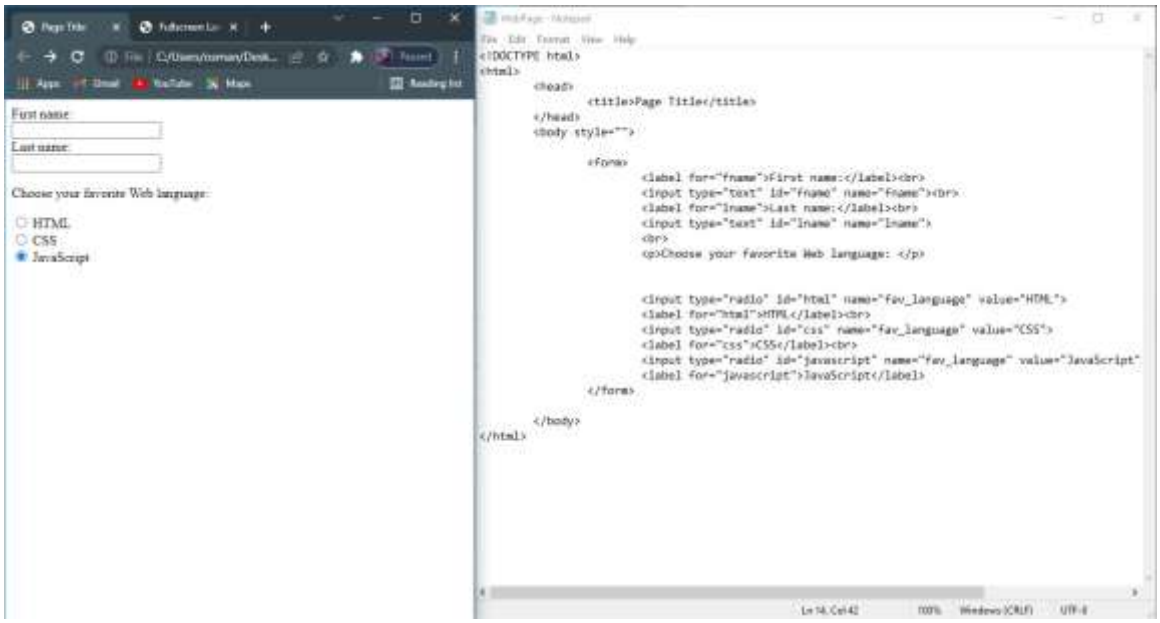


Fig. 2 (Radio Buttons)

Checkboxes:

The <input type="checkbox"> defines a checkbox.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

```
<form>
        <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
        <label for="vehicle1"> I have a bike</label><br>
```

```
<input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
<label for="vehicle2"> I have a car</label><br>
<input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
<label for="vehicle3"> I have a boat</label>
</form>
```

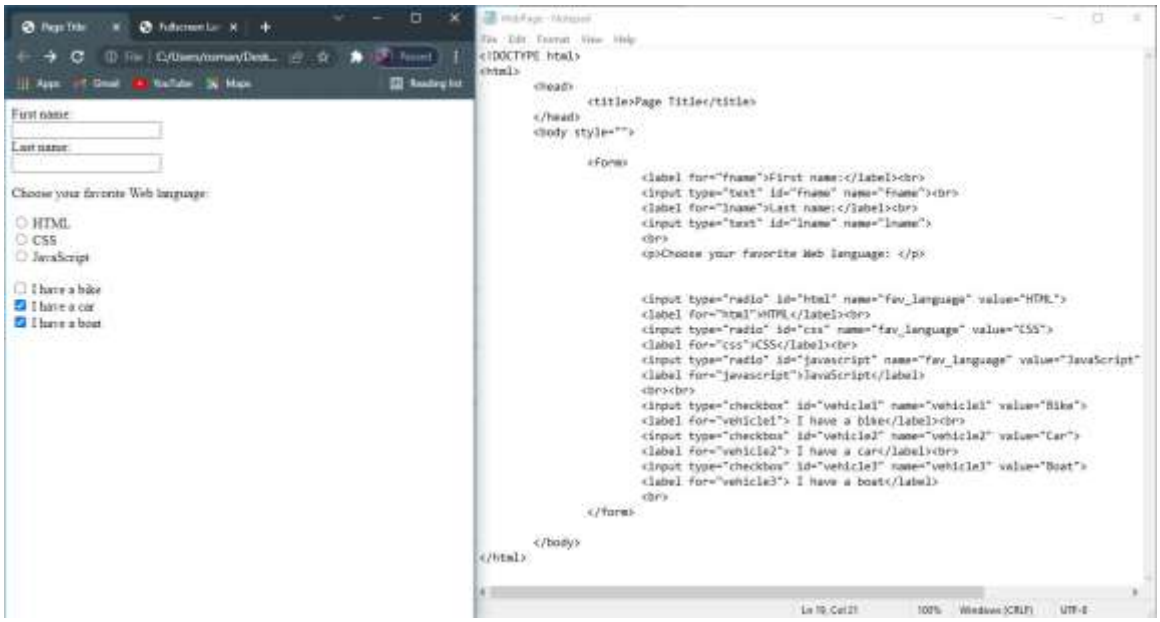


Fig. 3 (Form Check Box)

The Submit Button:

The <input type="submit"> defines a button for submitting the form data to a form-handler.

The form-handler is typically a file on the server with a script for processing input data.

The form-handler is specified in the form's action attribute.

```
<input type="submit" value="Submit">
```

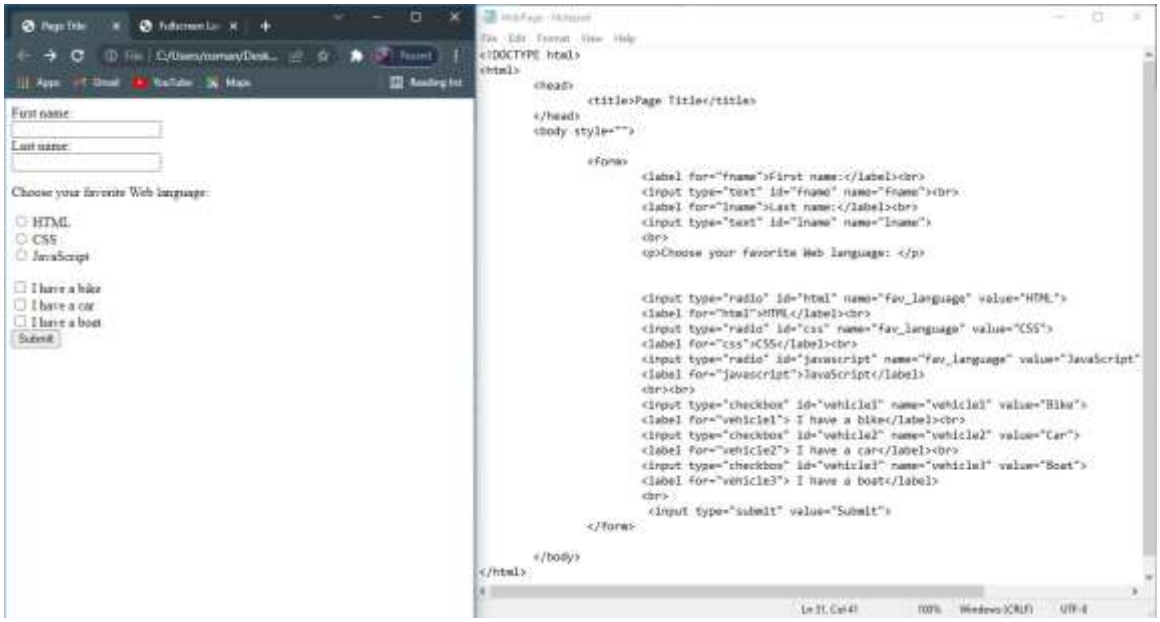


Fig. 4 (The Submit Button)

HTML Input Types:

This chapter describes the different types for the HTML <input> element.

HTML Input Types

Here are the different input types you can use in HTML:

- <input type="button">
- <input type="checkbox">
- <input type="color">
- <input type="date">
- <input type="datetime-local">
- <input type="email">
- <input type="file">
- <input type="hidden">
- <input type="image">
- <input type="month">
- <input type="number">
- <input type="password">
- <input type="radio">
- <input type="range">
- <input type="reset">
- <input type="search">
- <input type="submit">
- <input type="tel">
- <input type="text">
- <input type="time">
- <input type="url">
- <input type="week">

Input Type Text:

<input type="text"> defines a single-line text input field.

Input Type Password:

<input type="password"> defines a password field.

Example:

<label for="pwd">Password:</label>

<input type="password" id="pwd" name="pwd"> **Input**

Type Submit:

<input type="submit"> defines a button for submitting form data to a form-handler.

Input Type Button:

<input type="button"> defines a button.

Example:

<input type="button" value="Click Me!">

Input Type Date:

The <input type="date"> is used for input fields that should contain a date.

Depending on browser support, a date picker can show up in the input field.

Example:

```
<form>
  <label for="birthday">Birthday:</label>
  <input type="date" id="birthday" name="birthday">
</form>
```

Input Type Range

The <input type="range"> defines a control for entering a number whose exact value is not important (like a slider control). Default range is 0 to 100. However, you can set restrictions on what numbers are accepted with the min, max, and step attributes:

Example:

```
<form>
  <label for="vol">Volume (between 0 and 50): </label>
  <input type="range" id="vol" name="vol" min="0" max="50">
</form>
```

Input Type Search:

The <input type="search"> is used for search fields (a search field behaves like a regular text field).

Example:

```
<form>
  <label for="search">Search Google:</label>
  <input type="search" id="search" name="search">
</form>
```

Web design and development tools in the browser:

Modern web browsers do more than display web pages; they also include tools for developing and troubleshooting web pages. Some browser tools for testing and debugging sites include: **Internet Explorer developer tools**

Microsoft Internet Explorer includes built-in developer tools. You can access these tools by choosing Tools > Developer Tools or by using the keyboard shortcut F12.

Safari

The Apple Safari browser includes built-in developer tools that are not enabled by default. To enable the developer tools, choose Safari > Preferences. In the Preferences menu, select the Advanced menu and then select the check box labeled Show Developer menu in menu bar.

The Firefox Firebug extension

The Firebug extension is an option for extending the Mozilla Firefox browser. You can download it at http://getfi rebug.com.

Chrome

The Google Chrome browser includes built-in developer tools. To access these tools, click the Page menu at the top-right corner of the browser window, then choose Developer > Developer Tools. You can also right-click on any element and select Inspect Element. They include tools for inspecting your site's HTML and CSS as they exist inside the browser, instead of just viewing the original source code. This is particularly helpful with dynamic sites, complex sites, and sites that use frameworks such as ASP or PHP.

Inspecting HTML elements

You can inspect HTML elements by clicking on them. The code for a selected element is highlighted in a content pane. Use this to quickly identify the exact code that references an element so you can quickly understand information such as the width and height of an object on the page, such as an image or div element.

Inspecting CSS properties

Quickly access CSS styles that are associated with a selected element to see the exact CSS rules associated with your selection and examine the cascade of rules so you understand whether a style is associated with an internal style or external style sheet. This can make it easier to debug.

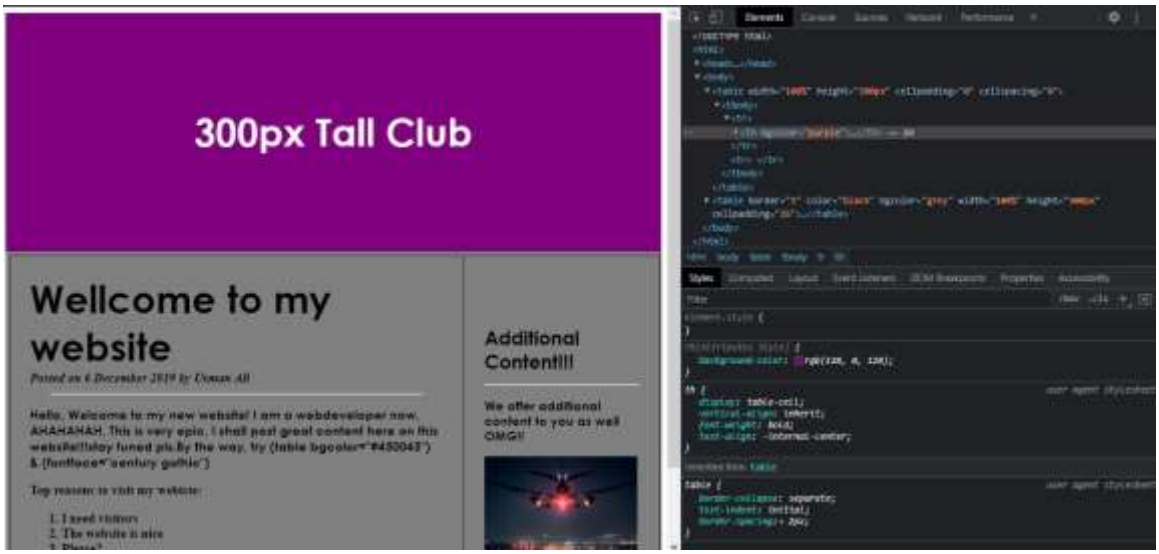


Fig. 8 (Inspecting in Browser)

HTML Styles – CSS

CSS stands for Cascading Style Sheets.

CSS is the language we use to style an HTML document.

CSS describes how HTML elements should be displayed.

CSS saves a lot of work. It can control the layout of multiple web pages all at once.

What is CSS?

Cascading Style Sheets (CSS) is used to format the layout of a webpage.

With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more!

Using CSS

CSS can be added to HTML documents in 3 ways:

- Inline - by using the style attribute inside HTML elements
- Internal - by using a <style> element in the <head> section
- External - by using a <link> element to link to an external CSS file

The most common way to add CSS, is to keep the styles in external CSS files. Previously, in the manual inline styling used. Now we're going to use External CSS files for styling.

External CSS:

An external style sheet is used to define the style for many HTML pages. To use an external style sheet, add a link to it in the <head> section of each HTML page. Here index.html is a html file and there is a link to external stylesheet css.

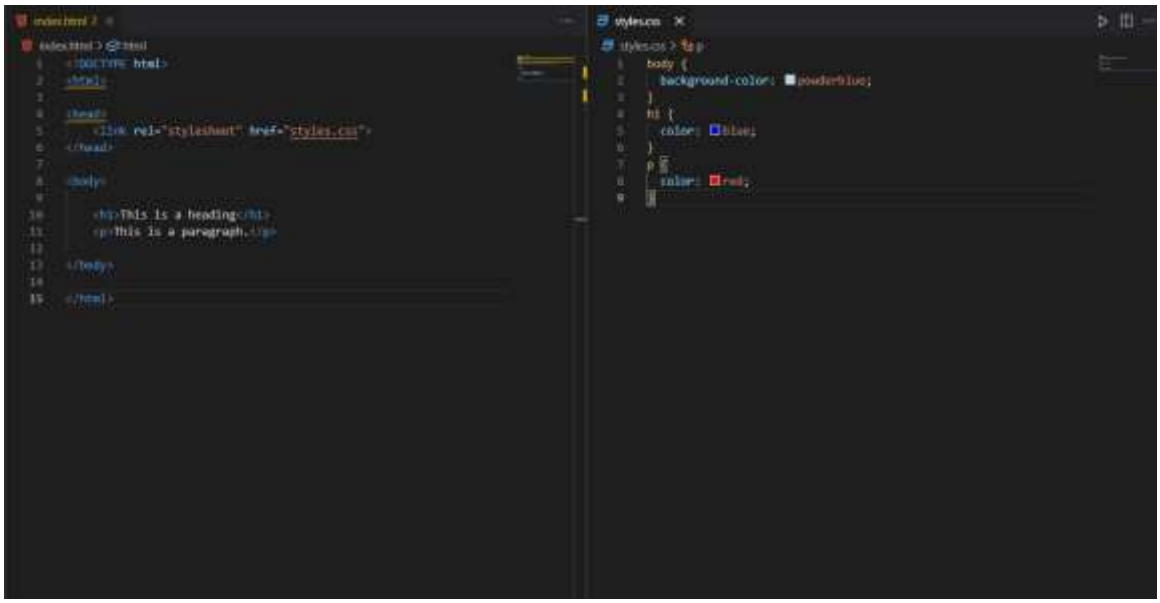


Fig. 9 (External CSS File)

Output of the files in the browser.

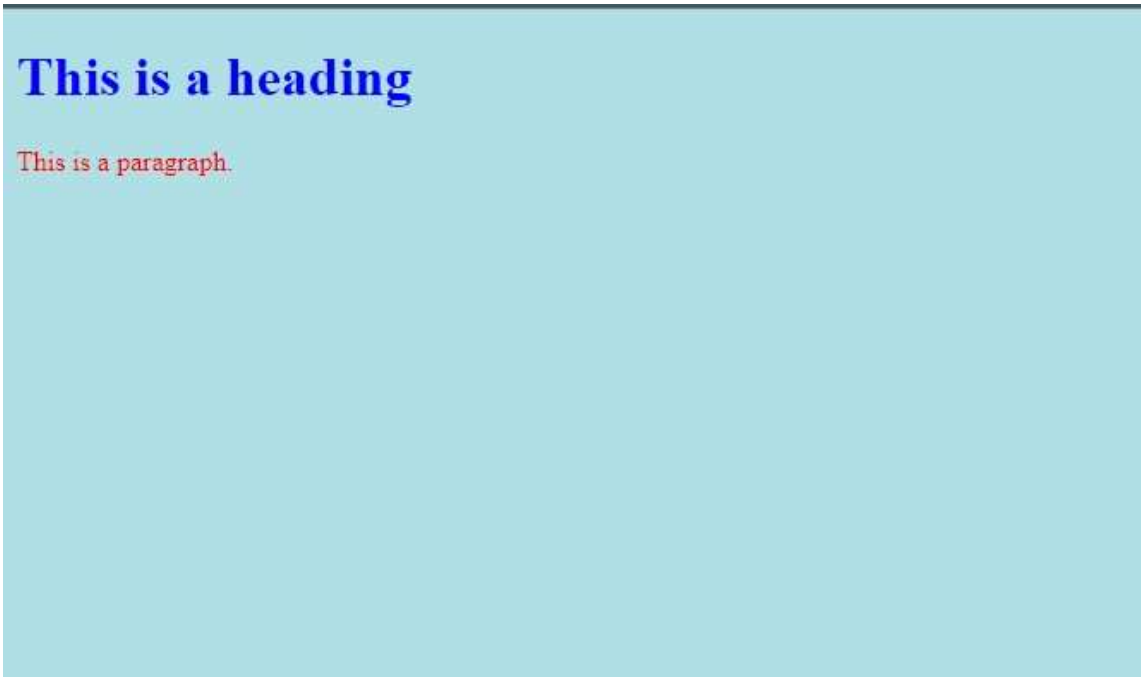


Fig. 10 (Browser Output)

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>

This is a paragraph.

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!

CSS Syntax:

A CSS rule consists of a selector and a declaration block.

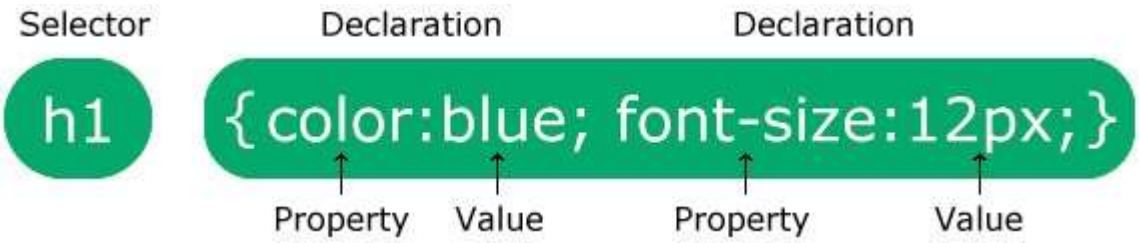


Fig. 11 (Understanding CSS Property)

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.

Example:

In this example all <p> elements will be center-aligned, with a red text color:

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

Example 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 **CSS**

Selectors:

A CSS selector selects the HTML element(s) you want to style.

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

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;
}
```

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;
}
```

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. **Example**

In this example all HTML elements with class="center" will be red and center-aligned:

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

The CSS Universal Selector

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

Example

The CSS rule below will affect every HTML element on the page:

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

CSS Comments:

CSS comments are not displayed in the browser, but they can help document your source code. **CSS**

Comments

Comments are used to explain the code, and may help when you edit the source code at a later date. Comments are ignored by browsers.
A CSS comment is placed inside the <style> element, and starts with /* and ends with */:

Example

```
/* This is a single-line comment */
p {   color: red;
}
```

CSS Margin:

The CSS margin properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

- margin-top

- margin-right
- margin-bottom
- margin-left

All the margin properties can have the following values:

auto - the browser calculates the margin length

- specifies a margin in px, pt, cm, etc.

% - specifies a margin in % of the width of the containing element inherit -

specifies that the margin should be inherited from the parent element Tip:

Negative values are allowed.

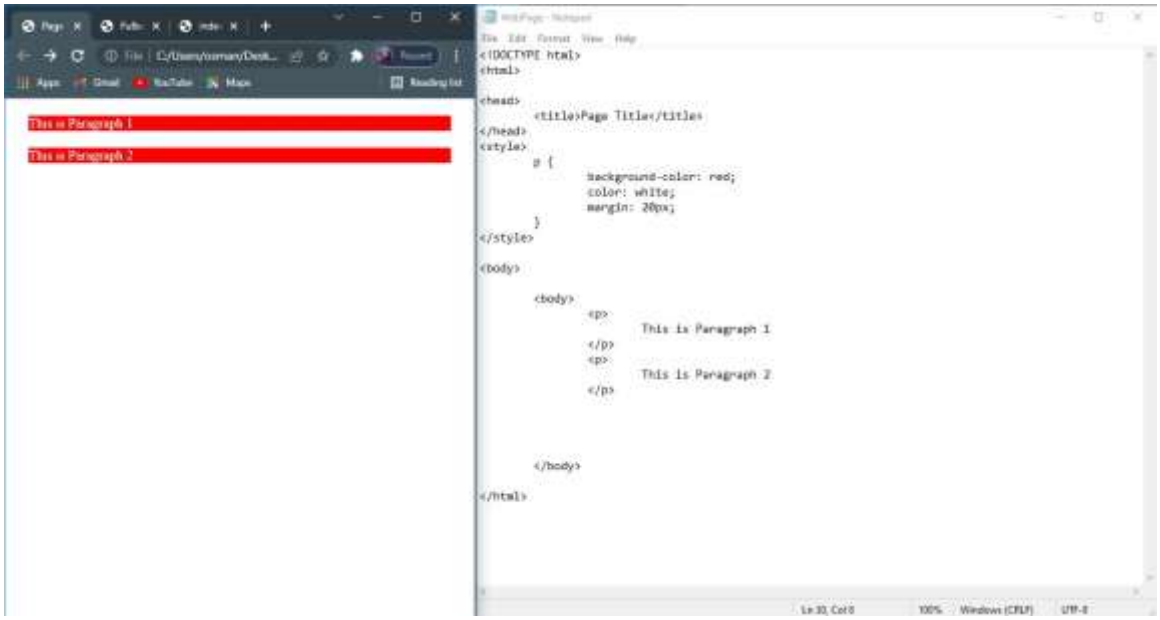


Fig. 12 (Margins in HTML)

Margin - Shorthand Property

To shorten the code, it is possible to specify all the margin properties in one property.

The margin property is a shorthand property for the following individual margin properties:

- margin-top
- margin-right
- margin-bottom
- margin-left

So, here is how it works:

If the margin property has four values:

margin: 25px 50px 75px 100px;

- top margin is 25px
- right margin is 50px
- bottom margin is 75px
- left margin is 100px

Example

Use the margin shorthand property with four values:

```
p {
  margin: 25px 50px 75px 100px;
}
```

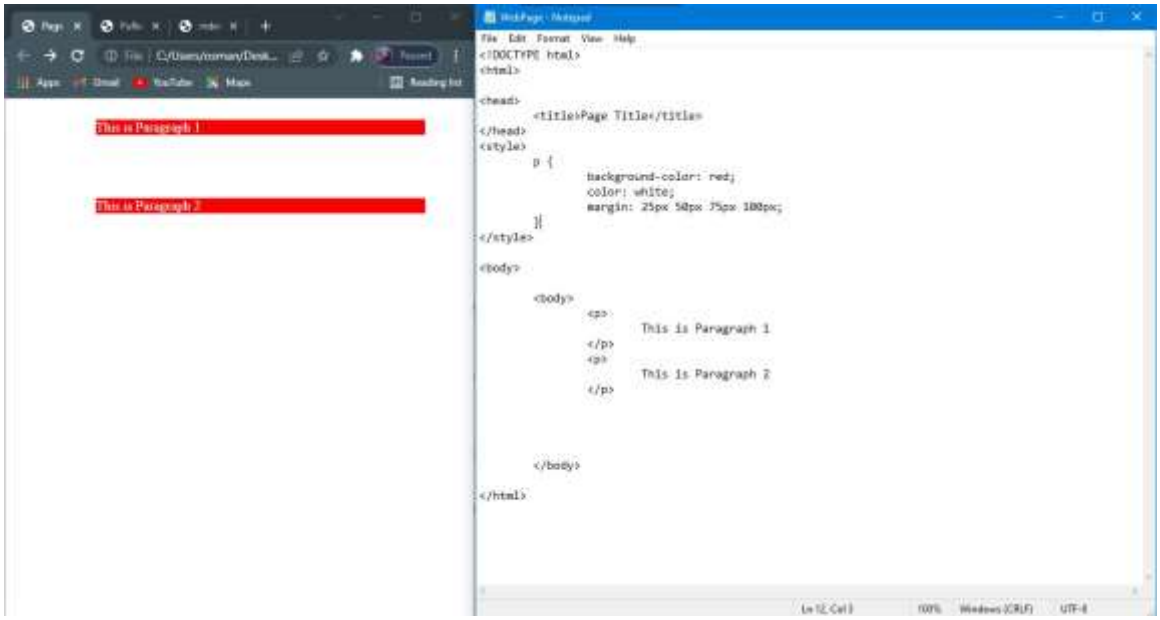


Fig. 13 (Shorthand Margin Property)

The auto Margin Value

You can set the margin property to auto to horizontally center the element within its container. The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

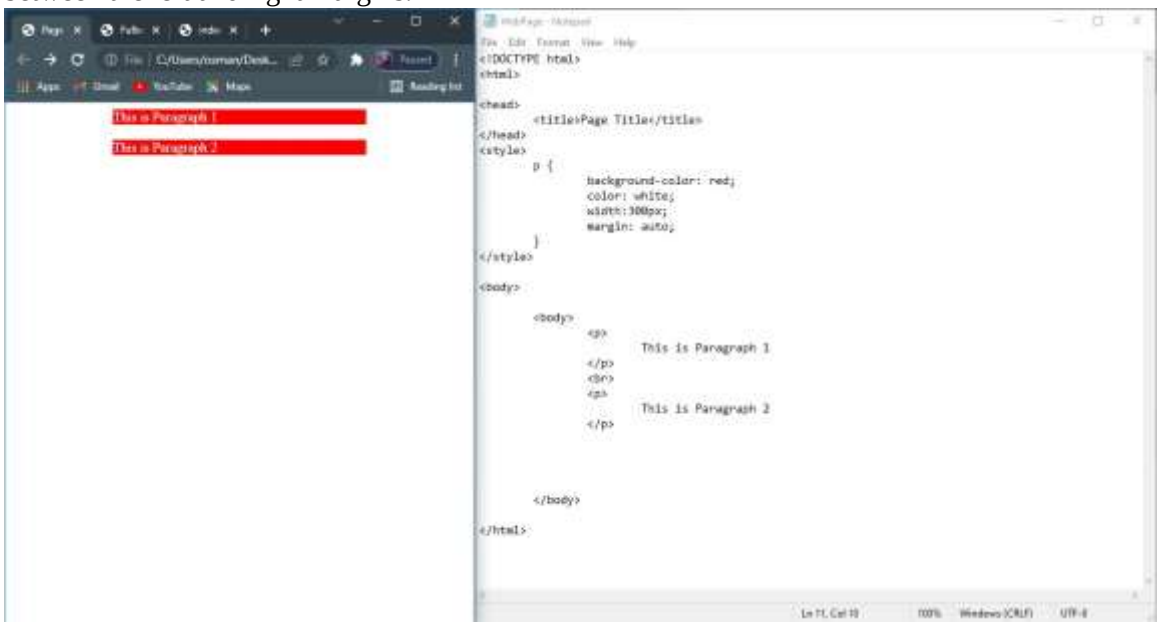


Fig. 14 (Auto margin)

CSS Padding:

Padding is used to create space around an element's content, inside of any defined borders.

The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- padding-top
- padding-right
- padding-bottom
- padding-left

All the padding properties can have the following values: length

- specifies a padding in px, pt, cm, etc.

% - specifies a padding in % of the width of the containing element inherit -

specifies that the padding should be inherited from the parent element Note:

Negative values are not allowed.

Example

Set different padding for all four sides of a element: div

{

padding-top: 50px;
padding-right: 30px;
padding-bottom: 50px;
padding-left: 80px;

}

Padding - Shorthand Property:

To shorten the code, it is possible to specify all the padding properties in one property.

The padding property is a shorthand property for the following individual padding properties:

- padding-top
- padding-right
- padding-bottom
- padding-left

So, here is how it works:

If the padding property has four values:

padding: 25px 50px 75px 100px;

top padding is 25px right

padding is 50px bottom padding

is 75px left padding is 100px

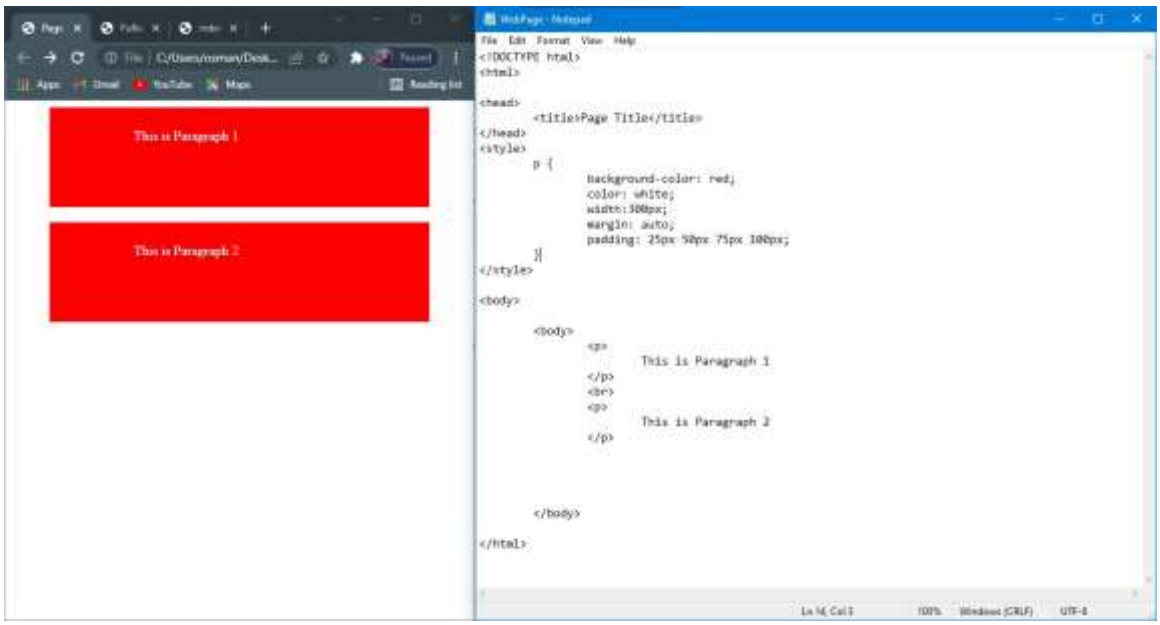


Fig. 15 (Short hand Property Padding)

Padding and Element Width:

The CSS width property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element (the box model).
So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result. **Example:**

Here, the element is given a width of 300px. However, the actual width of the element will be 350px (300px + 25px of left padding + 25px of right padding):

div { width:

300px;

padding: 25px;

}

CSS Height, Width and Max-width

The CSS height and width properties are used to set the height and width of an element.

The CSS max-width property is used to set the maximum width of an element.

CSS height and width Values

The height and width properties may have the following values:

auto - This is default. The browser calculates the height and width length
- Defines the height/width in px, cm etc.
% - Defines the height/width in percent of the containing block initial
- Sets the height/width to its default value
inherit - The height/width will be inherited from its parent value

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
</style>
</head>
<body>

<h2>Set the height and width of an element</h2>

<div>This div element has a height of 200px and a width of 50%.</div>

</body>
</html>
```

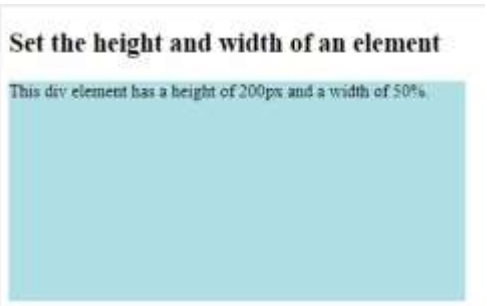


Fig. 16 (Height & Width)

Setting max-width:

The max-width property is used to set the maximum width of an element.
The max-width can be specified in length values, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).

The problem with the above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.
Using max-width instead, in this situation, will improve the browser's handling of small windows.

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.

Fig. 17 (CSS Box Model)

Explanation of the different parts:

- 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.

CSS Layout - The position Property

The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

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. **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: This element has position: static; Here is the CSS that is used.

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.

This element has position: relative;

Here is the CSS that is used: **Example**

```
div. relative {

position: relative;

left: 30px;

  border: 3px solid #73AD21;
```

} **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.

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.

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Here is a simple example:

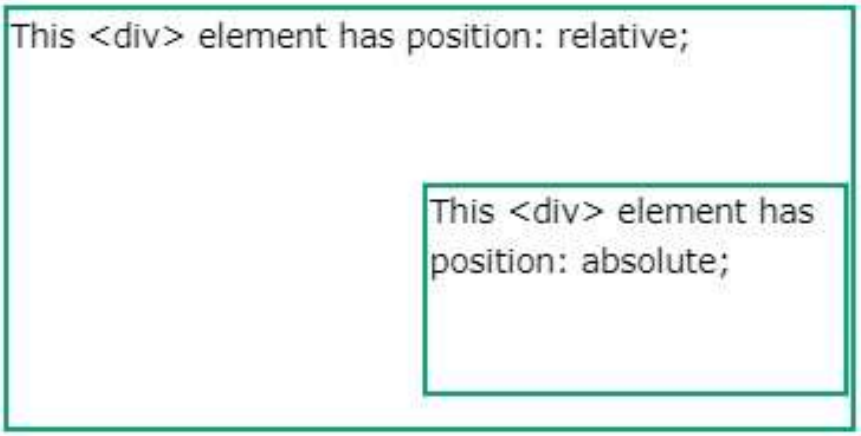


Fig. 18 (Absolute Positioning)

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).

Div Tag:

The tag defines a division or a section in an HTML document.
The tag is used as a container for HTML elements - which is then styled with CSS or manipulated with JavaScript.
The tag is easily styled by using the class or id attribute.
Any sort of content can be put inside the tag!

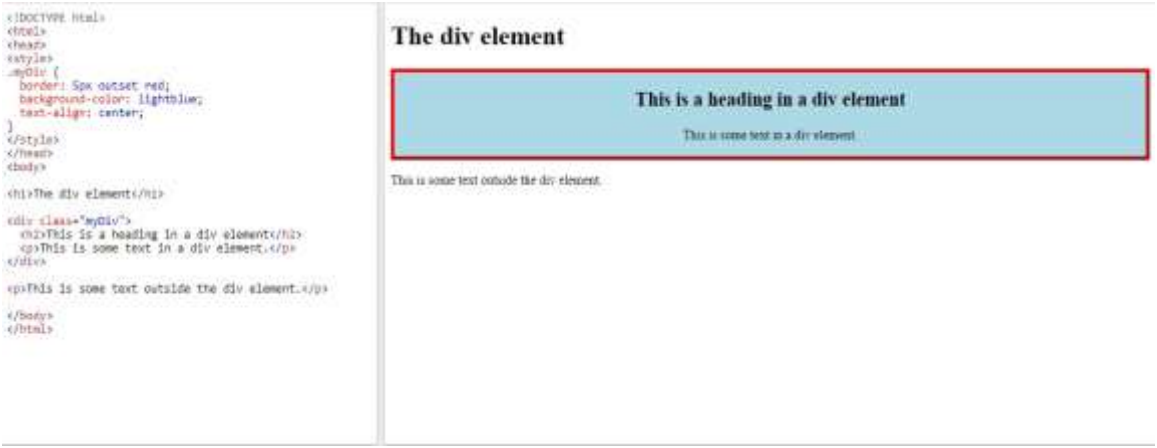


Fig. 19 (Sticky Positioning)

CSS Layout - Horizontal & Vertical Align:

Center Align Elements:

To horizontally center a block element (like), use margin: auto;
Setting the width of the element will prevent it from stretching out to the edges of its container.

Fig. 20 (Center Align Element)

The element will then take up the specified width, and the remaining space will be split equally between the two margins:

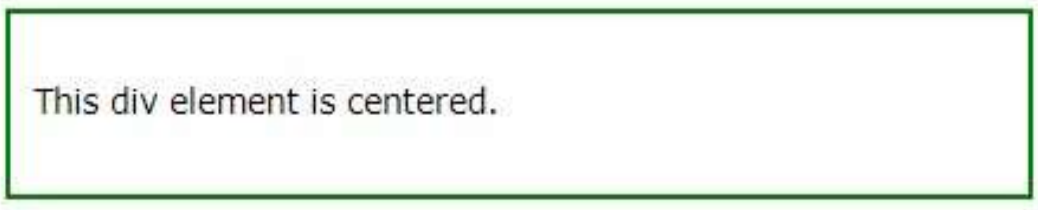


Fig. 21 (Vertical Center Align)

Code Example

.center {

margin: auto;

width: 50%;

border: 3px

solid green;

padding: 10px;

}

Center aligning has no effect if the width property is not set (or set to 100%).

Center Align Text:

To just center the text inside an element, use text-align: center;

Fig. 22 (Text Align Center)

Code Example:
.center { text-align:
center; border: 3px
solid green; }

Center an Image

To center an image, set left and right margin to auto and make it into a block element.

Code Example img { display: block; margin-left: auto; margin-right: auto;

width: 40%;

}

Fig. 23 (Center an Image)

Center Vertically - Using padding

There are many ways to center an element vertically in CSS. A simple solution is to use top and bottom padding:

Fig. 24 (Align Using Padding)

Code Example

```
.center {  padding: 70px
0;  border: 3px solid
green; }
```

To center both vertically and horizontally, use padding and text-align: center.

It can be center vertically by using:

- Using position & transform
- Line Height
- Flexbox

CSS Combinators:

A combinator is something that explains the relationship between the selectors.

A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.

There are four different combinators in CSS:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~) **Descendant Selector:**

The descendant selector matches all elements that are descendants of a specified element.

The following example selects all <p> elements inside elements:

Example:

```
div  p
{
 background-color: yellow; }
```

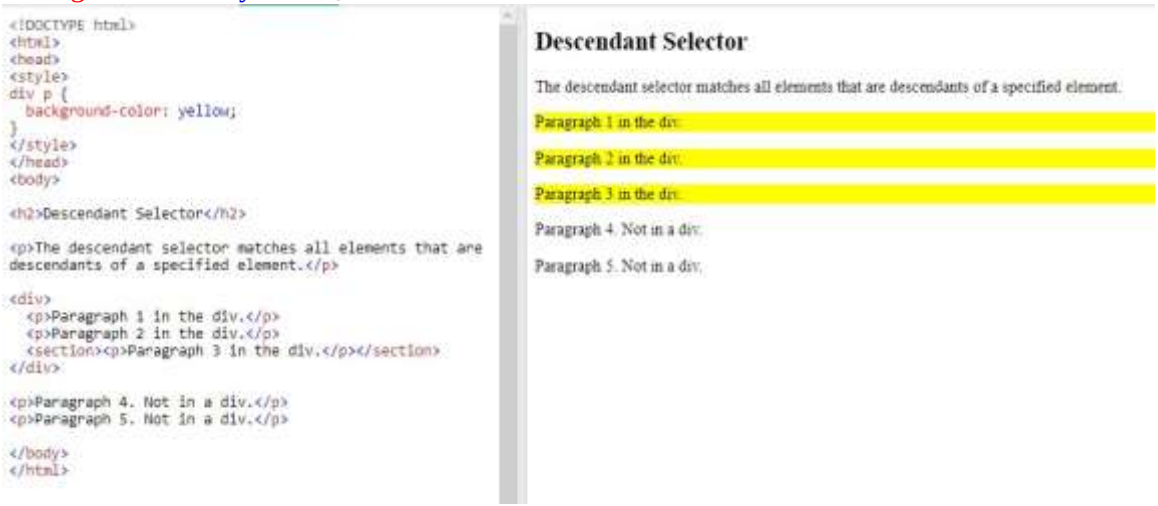


Fig. 26 (Descendant Selector)

Child Selector (>):

The child selector selects all elements that are the children of a specified element.

The following example selects all <p> elements that are children of a element:

Example:

```
div > p {
 background-color: yellow;
```

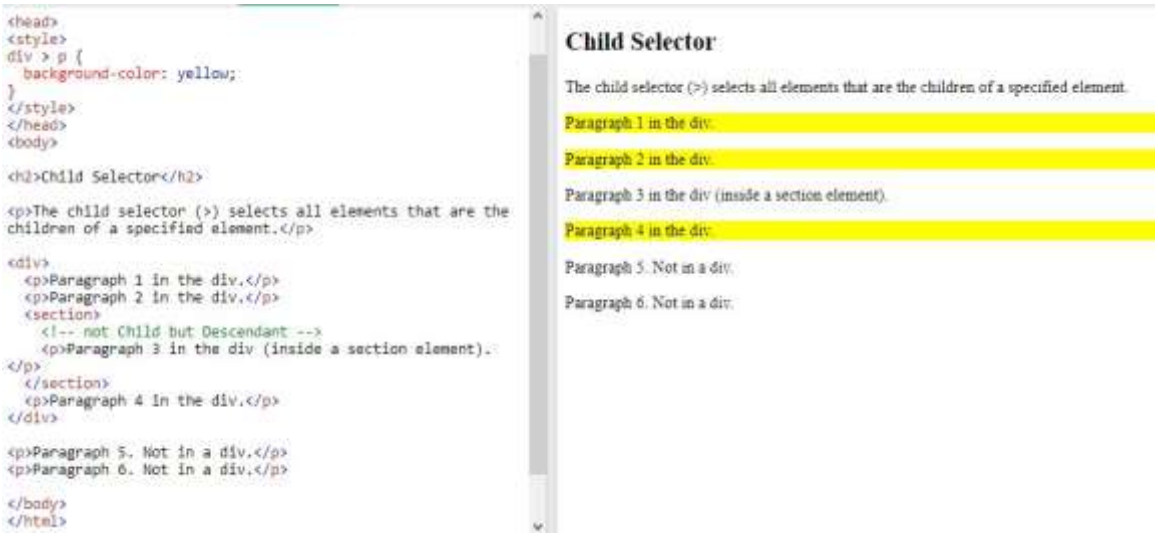


Fig. 27 (Child Selector)

Adjacent Sibling Selector (+):

The adjacent sibling selector is used to select an element that is directly after another specific element. Sibling elements must have the same parent element, and "adjacent" means "immediately following". The following example selects the first <p> element that are placed immediately after elements:

Example:

```
div + p {
  background-color: yellow; }
```

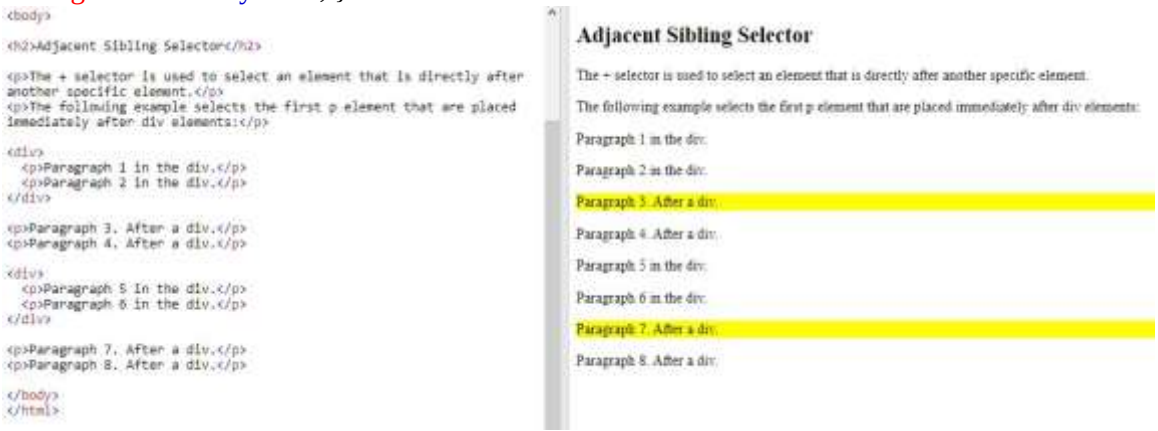


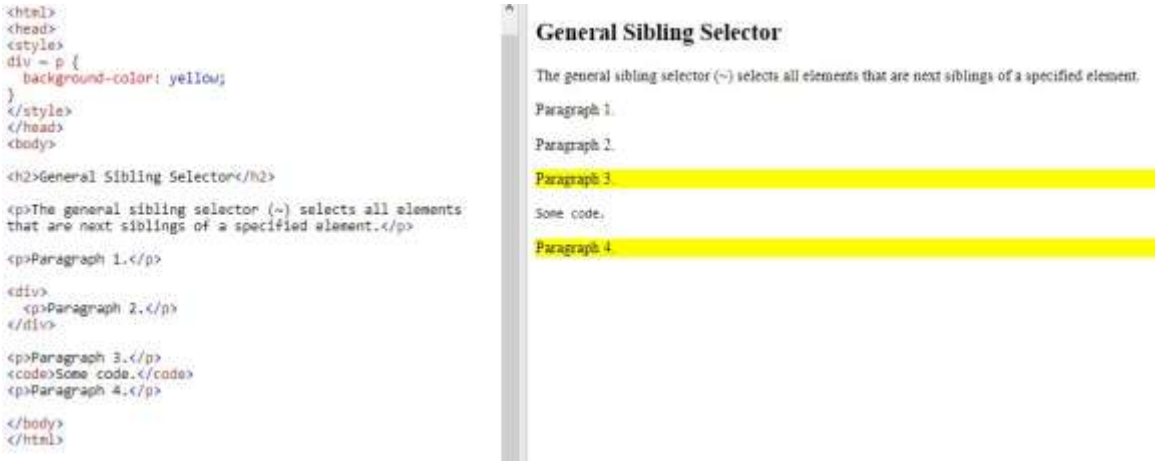
Fig. 28 (Adjacent Sibling Selector)

General Sibling Selector (~):

The general sibling selector selects all elements that are next siblings of a specified element. The following example selects all <p> elements that are next siblings of elements:

Example:

```
div ~ p {
  background-color: yellow;
```



```
<html>
<head>
<style>
div ~ p {
  background-color: yellow;
}
</style>
</head>
<body>

<h2>General Sibling Selector</h2>

<p>The general sibling selector (~) selects all elements
that are next siblings of a specified element.</p>

<p>Paragraph 1.</p>

<div>
  <p>Paragraph 2.</p>
</div>

<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>

</body>
</html>
```


Fig. 29 (General Sibling Selector)

CSS 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:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus **Syntax:**

The syntax of pseudo-classes:

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

Anchor Pseudo-classes:

Links can be displayed in different ways:

```
/* 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.

Pseudo-classes and HTML Classes:

Pseudo-classes can be combined with HTML classes:

When you hover over the link in the example, it will change color:

Example:

```
a.highlight:hover {
color: #ff0000; }
```

CSS - The :first-child Pseudo-class:

The :first-child pseudo-class matches a specified element that is the first child of another element.

Match the first <p> element

In the following example, the selector matches any <p> element that is the first child of any element:

Example:

```
p:first-child {
color: blue; }
```

Match the first <i> element in all <p> elements

In the following example, the selector matches the first <i> element in all <p> elements:

Example:

```
p i:first-child {
color: blue; }
```

CSS Pseudo Classes:

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <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
:last-child	p:last-child	Selects every <p> elements that is the last child 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-child	p:only-child	Selects every <p> element that is the only child of its parent
:required	input:required	Selects <input> elements with a "required" attribute specified

:visited	a:visited	Selects all visited links

CSS Flexbox:

What is CSS Flexbox?

CSS flexbox is a one-dimensional layout pattern that makes it easy to design flexible and effective layouts. The use of flexbox ensures that elements are properly placed and are predictable. Flex items are positioned inside a flex container along a flex line. By default, there is only one flex line per flex container.

Basics and Terminology

Let's get ourselves familiar with the basic terminology that is common in flexbox.

- **main-axis:** The main-axis is the primary axis of the flex container along which flex items are aligned.
- **main-start | main-end:** The flex items are placed or set inside the container beginning with main-start and going up to main-end.
- **cross-axis:** Cross-axis is referred to as the axis perpendicular to the main axis, and its direction depends on the main axis direction.

CSS Flex Container:

Parent Element (Container)

Like we specified in the previous chapter, this is a flex container (the blue area) with three flex items.

```
<html>
<head>
<style>
.flex-container {
  display: flex;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>Create a Flex Container</h1>

<div class="flex-container">
```

```
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

</body>
</html>
```

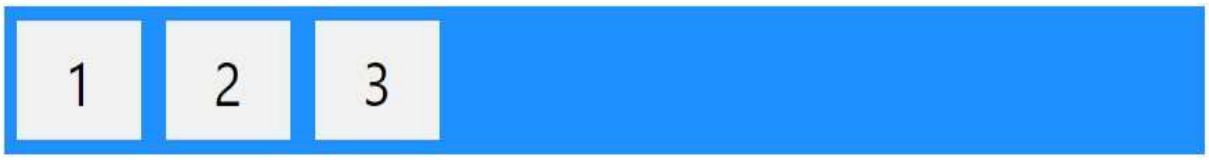


Fig. 30 (Flex Container)

The flex container becomes flexible by setting the display property to flex:

Example:

```
.flex-container
{   display: flex;
}
```

The flex container properties are:

- flex-direction
- flex-wrap
- flex-flow
- justify-content
- align-items
- align-content

The flex-direction Property:

The flex-direction property defines in which direction the container wants to stack the flex items.

Fig. 31 (Flex Container)

Example

The column value stacks the flex items vertically (from top to bottom):

```
.flex-container
{  display: flex;
 flex-direction:  column;
}
```

The flex-direction Property:

The "flex-direction: row;" stacks the flex items horizontally (from left to right):

Fig. 32 (Flex Container)

Example

The "flex-direction: row;" stacks the flex items horizontally (from left to right):

```
flex-container      {
display: flex;    flex-
direction: row; }
```

The row-reverse value stacks the flex items horizontally (but from right to left):

```
.flex-container
{  display: flex;
 flex-direction: row-reverse; }
```

The flex-wrap Property

The flex-wrap property specifies whether the flex items should wrap or not. The flex-wrap property is a sub-property of the Flexible Box Layout module. It defines whether the flex items are forced in a single line or can be flowed into multiple lines. If set to multiple lines, it also defines the cross-axis which determines the direction new lines are stacked in.

Example:

The wrap value specifies that the flex items will wrap if necessary.

```
.flex-container  {
display:      flex;
flex-wrap:  wrap;
}
```

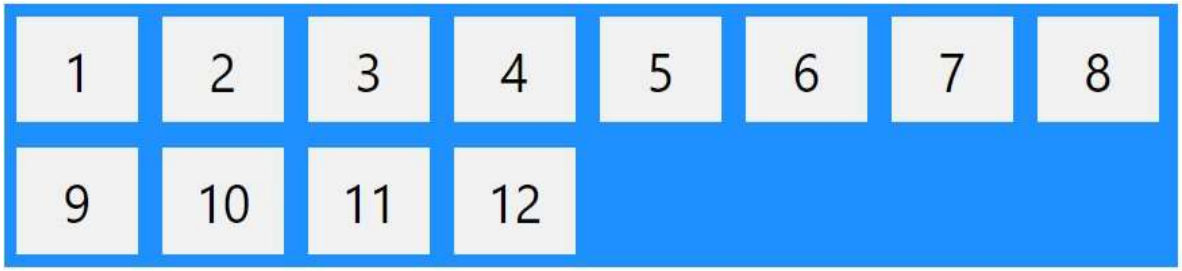


Fig. 33 (Flex Container)

The flex-wrap Property

The "flex-wrap: nowrap;" specifies that the flex items will not wrap (this is default):

Fig. 34 (Flex Container)

All items are in one row.

The justify-content Property:

The justify-content property is used to align the flex items:

Fig. 35 (Flex Container)

Example

The center value aligns the flex items at the center of the container:

```
.flex-container
{   display: flex;
  justify-content: center;
}
```

Fig. 36 (Flex Container)

Example

The flex-start value aligns the flex items at the beginning of the container (this is default):

```
.flex-container
{   display: flex;
  justify-content:   flex-start;
}
```

Fig. 37 (Flex Container)

Example:

The space-around value displays the flex items with space before, between, and after the lines:

```
.flex-container
{  display: flex;
 justify-content:  space-around;
}
```

The align-items Property:

The align-items property is used to align the flex items.

Fig. 38 (Flex Container)

In these examples we use a 200 pixels high container, to better demonstrate the align-items property.

Example:

The center value aligns the flex items in the middle of the container:

```
.flex-container     {
display:          flex;
height:         200px;
align-items: center;
}
```

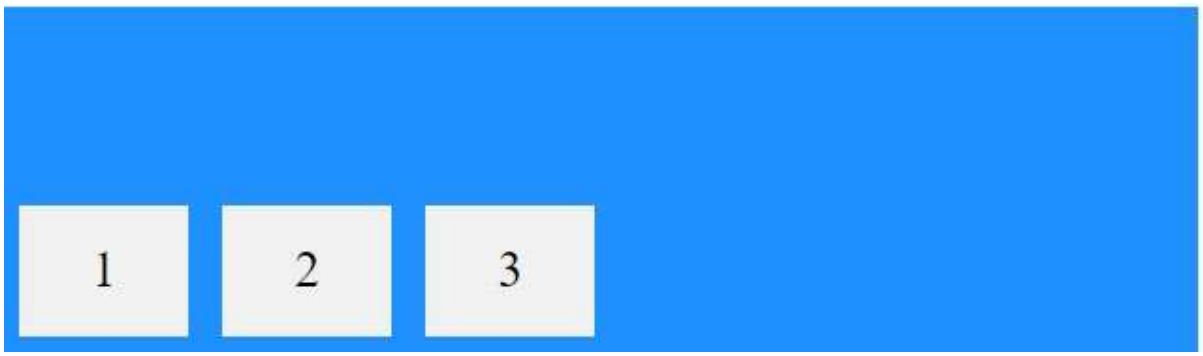


Fig. 39 (Flex Container)

Example:

The flex-end value aligns the flex items at the bottom of the container:

```
.flex-container        {
display: flex;  height:
200px;    align-items:
flex-end; }
```

The CSS Flexbox Container Properties:

The following table lists all the CSS Flexbox Container properties:

Property	Description
align-content	Modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns flex lines
align-items	Vertically aligns the flex items when the items do not use all available space on the cross-axis
display	Specifies the type of box used for an HTML element
flex-direction	Specifies the direction of the flexible items inside a flex container
flex-flow	A shorthand property for flex-direction and flex-wrap
flex-wrap	Specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line
justify-content	Horizontally aligns the flex items when the items do not use all available space on the main-axis

CSS Media Queries

CSS2 Introduced Media Types:

The @media rule, introduced in CSS2, made it possible to define different style rules for different media types.

Examples: You could have one set of style rules for computer screens, one for printers, one for handheld devices, one for television-type devices, and so on. Unfortunately, these media types never got a lot of support by devices, other than the print media type.

CSS3 Introduced Media Queries:

Media queries in CSS3 extended the CSS2 media types. Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

- width and height of the viewport
- width and height of the device
- orientation (is the tablet/phone in landscape or portrait mode?)

resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

Min Width to Max Width:

You can also use the (max-width:) and (min-width:) values to set a minimum width and a maximum width. For example, when the browser's width is between 600 and 900px, change the appearance of a element:

```
@media screen and (max-width: 900px) and (min-width: 600px)
{  div.example {    font-size: 50px;    padding: 50px;    border:
8px solid black;
   background: yellow;
  }
}
```

Media Queries Simple Examples:

One way to use media queries is to have an alternate CSS section right inside your style sheet.

The following example changes the background-color to light-green if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the background-color will be pink):

Example:

```
@media  screen  and  (min-width:  480px)  {
body {
   background-color: lightgreen;
  }

}
```

Media Queries for Menus:

In this example, we use media queries to create a responsive navigation menu, that varies in design on different screen sizes.

Fig. 40 (Media Query)

```
/* The navbar container */
.topnav  {      overflow:
hidden;     background-
color: #333;
}


/*  Navbar  links  */
.topnav  a  {     float:
```

```
left;  display: block;
color: white;  text-
align:  center;
padding: 14px 16px;
text-decoration:
none;
}

/* On screens that are 600px wide or less, make the menu links stack on top of each other instead of
next to each other */
@media screen and (max-width: 600px) {
 .topnav  a  {
float: none;
  width: 100%;
 }
}
```

Media Queries for Columns:

A common use of media queries, is to create a flexible layout. In this example, we create a layout that varies between four, two and full-width columns, depending on different screen sizes:

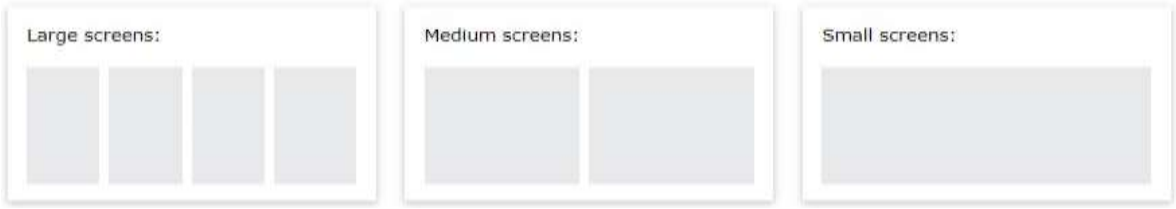


Fig. 41 (Media Query)

Example:

```
/* Create four equal columns that floats next to each other */
.column {
 float: left;
 width: 25%;
}

/* On screens that are 992px wide or less, go from four columns to two columns */
@media screen and (max-width: 992px) {
 .column  {
width: 50%;
 }

}

/* On screens that are 600px wide or less, make the columns stack on top of each other instead of next
to each other */
@media screen and (max-width: 600px) {
 .column {
  width: 100%;
```

```
  }
}
```

Flex Box Responsive:

A more modern way of creating column layouts, is to use CSS Flexbox (see example below).

```
/* Container for flexboxes */
.row {   display:
flex;        flex-
wrap: wrap;
}

/* Create four equal columns */
.column      {
flex:     25%;
padding:
20px;
}


/* On screens that are 992px wide or less, go from four columns to two columns */
@media screen and (max-width: 992px) {
  .column {
    flex: 50%;
  }

}


/* On screens that are 600px wide or less, make the columns stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .row {
    flex-direction: column;
  }

}
```