A PRACTICAL APPROACH TO HTML, CSS, JavaScript & Bootstrap

Author: www.teampiccolo.com

First Edition 2020

Table of Contents

Chapter 1: HTML5 and CSS3

HTML and CSS are web technologies used to design and build websites. All websites use HTML and CSS. Html stands for Hyper Text Mark-up Language. It is use for creating web pages and web applications.

CSS stands for Cascading Style Sheet. CSS describes how Html elements are to be displayed on the screen, paper or in other media.

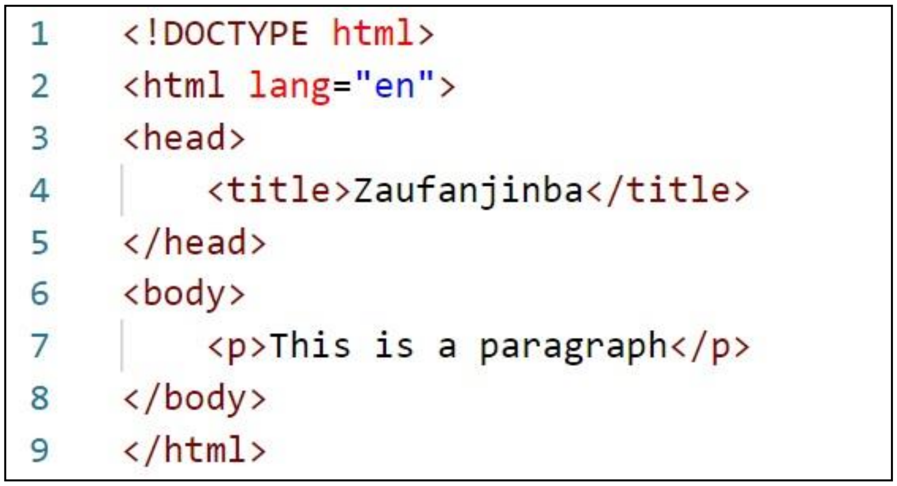
Before we start learning about these two technologies is this chapter, let’s learn about some terminologies.

Web	The **Web** is the common name for the World Wide Web, a subset of the Internet consisting of the pages that can be accessed by a Web browser.
Web Pages	**Web Pages** are what make up a website. For example on www.zaufanjinba.org, you can find the home page, about me page, my mission page, my vision page, foundation page and contact us page.
Websites	A **Website** is a collection of related web pages located under a single domain name. When you combine the home page, about me page, my mission page, my vision page, foundation page and contact us page and store them under a single domain name www.zaufanjinba.org
Web Server	A **web server** is a computer that runs websites.
Text Editors	A **text editor** is a type of computer program that edits plain text. Example of text editors include notepad++, atom, visual studio code, sublime etc.

HTML Introduction

HTML was created by Sir Tim Berners-Lee in late 1991 but was not released officially, which was published in 1995 as HTML 2.0. HTML 4.01 was published in late 1999 and was a major version of HTML. The current version of HTML as the time of writing this book is HTML 5.2 released in 2017.

HTML stands for Hyper Text Markup Language. HTML describes the structure of a Web page. HTML consists of a series of elements. HTML elements tell the browser how to display the content. Don’t worry, we will learn about HTML elements soon.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Zaufanjinba</title>
</head>
<body>
    <p>This is a paragraph</p>
</body>
</html>
```

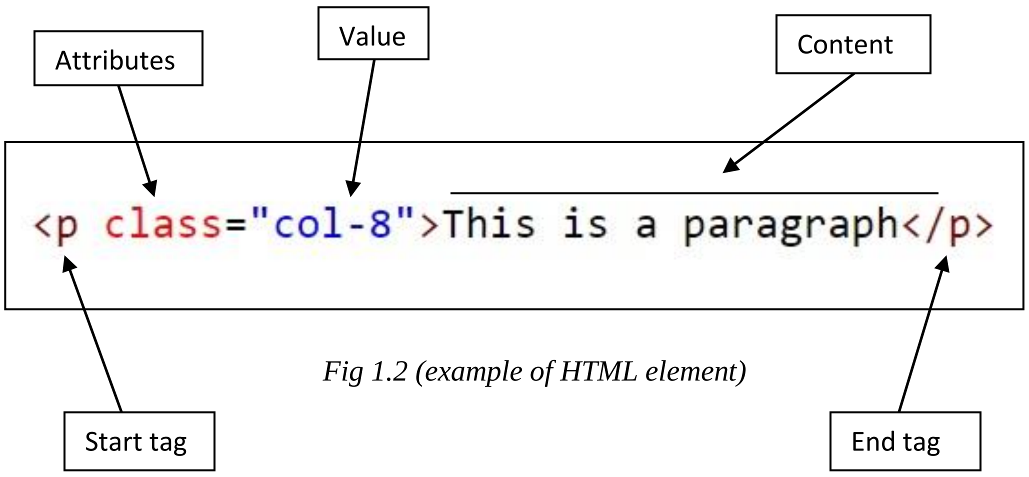
Fig 1.1 (*basic structure of html*)

Foot Note:
*You save an HTML file with the extension of either **.html** or **.htm** but use **.html** as most developers use that.*

HTML Elements

An HTML element is an individual component of an HTML document. It represents semantics, or meaning. For example, the title element represents the title of the document.

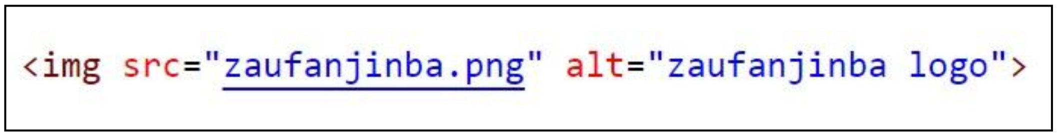
An HTML element is defined by a start tag, some content, and an end tag.



```
<p class="col-8">This is a paragraph</p>
```

Fig 1.2 (example of HTML element)

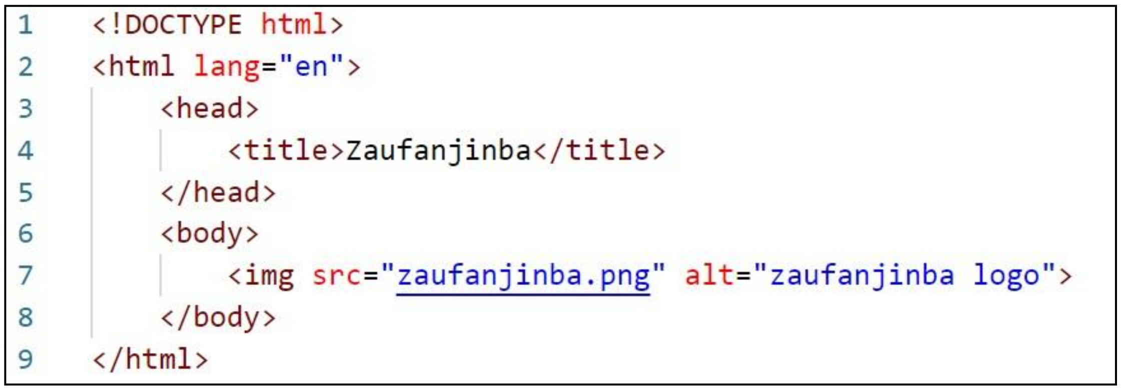
However, not all HTML element require the start tag and the end tag or close tag. HTML elements like <br>, <img>, <link>, <hr> etc. don't have the close tag. They are called **empty element**.



```
<img src="zaufanjinba.png" alt="zaufanjinba logo">
```

Fig 1.3 (example of an empty element)

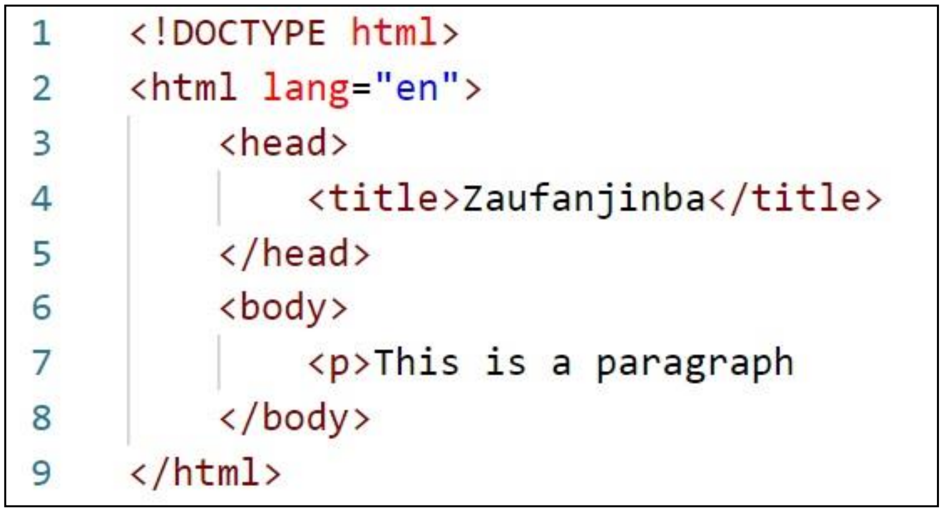
HTML elements can be nested (this means that elements can contain other elements). All HTML documents consist of nested HTML elements. In ***fig 1.4*** below, the title element is nested inside the head element and the head element is nested inside the html element.



```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Zaufanjinba</title>
    </head>
    <body>
        <img src="zaufanjinba.png" alt="zaufanjinba logo">
    </body>
</html>
```

Fig. 1.4 (nested elements)

Never skip the end tag of an HTML element. Some HTML elements will display correctly, even if you forget the end tag. However, never rely on this! Unexpected results and errors may occur if you forget the end tag as in *fig. 1.5* below, the <p> closing tag is missing.



```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Zaufanjinba</title>
    </head>
    <body>
        <p>This is a paragraph
    </body>
</html>
```

Fig 1.5 (skipping the end tag)

Inside the opening tag of image element in *fig. 1.4*, you notice two attributes; src and alt. We will learn about HTML attributes in the next section of this book.

Foot Note:

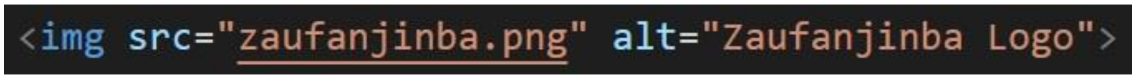
*HTML elements are **not case sensitive**: <P> means the same as <p>. The HTML standard does not require lowercase tags, but W3C **recommends** lowercase in HTML.*

HTML Attributes

All HTML elements can have attributes. Attributes provide additional information about elements. Attributes are always specified in the start tag.

The <img> tag is used to embed an image in an HTML page. The src attribute specifies the path to the image to be displayed. We will learn more about images in HTML.

The required alt attribute for the <img> tag specifies an alternate text for an image, if the image for some reason cannot be displayed. This can be due to slow connection, or an error in the src attribute, or if the user uses a screen reader.

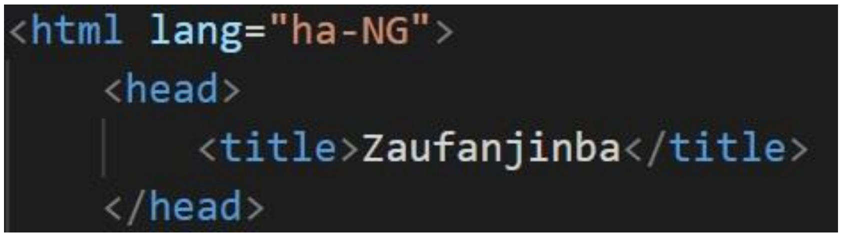


```html
<img src="zaufanjinba.png" alt="Zaufanjinba Logo">
```

Fig 1.6 (html attributes)

The lang attribute inside the <html> tag, is used to declare the language of the Web page. This is meant to assist search engines and browsers.

Country codes can also be added to the language code in the lang attribute. So, the **first two characters** define the **language of the HTML** page, and the **last two characters define the country**. The fig. 1.7 below specifies **Hausa** as the language and **Nigeria** as the country:



```html
<html lang="ha-NG">
    <head>
        <title>Zaufanjinba</title>
    </head>
```

Fig 1.7 (country code)

Foot Note:

*The HTML standard does not require lowercase attribute names. W3C **recommends** lowercase attributes in HTML. The HTML standard does not require quotes around attribute values. W3C **recommends** quotes in HTML. Double quotes around attribute values are the most common in HTML, but single quotes can also be used.*

HTML Head Element

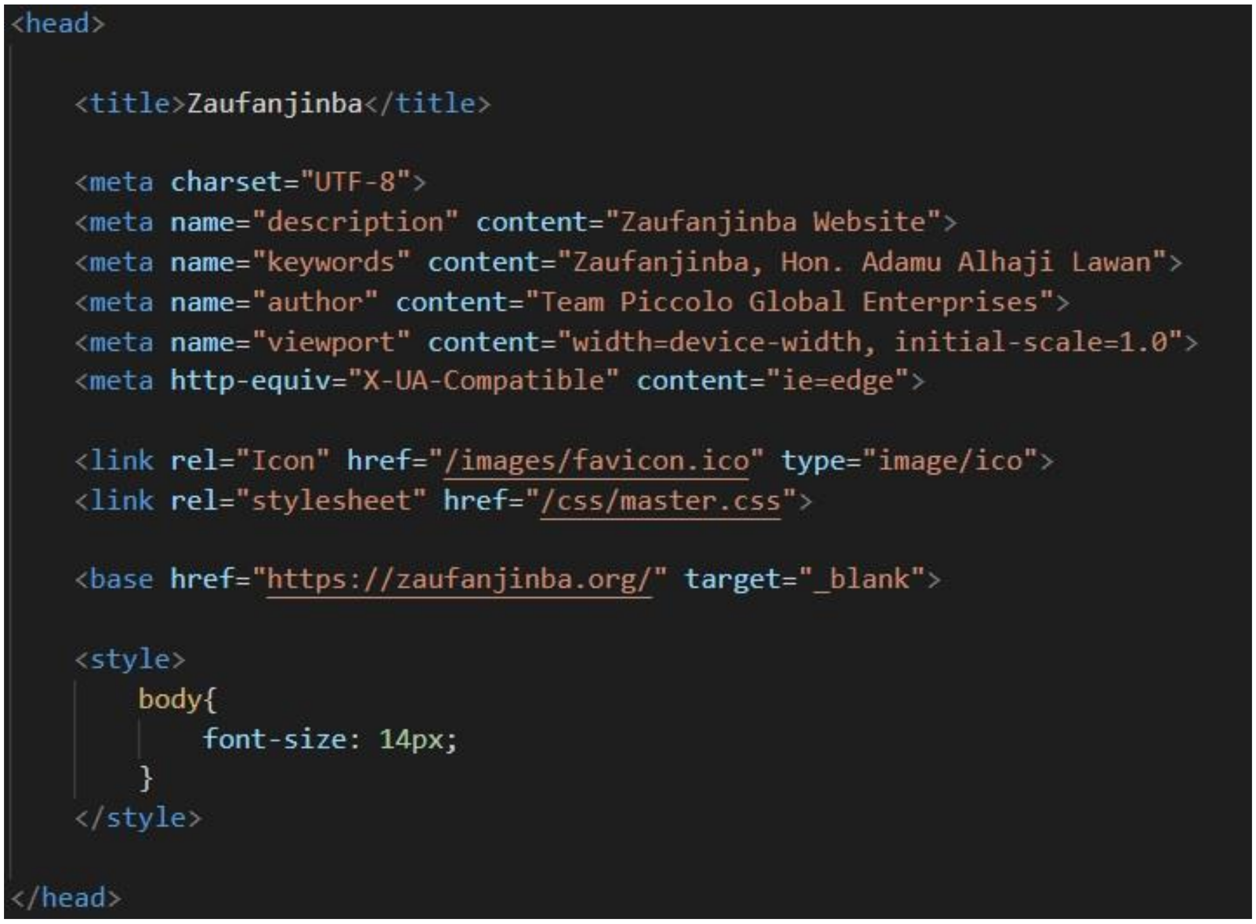
The <head> element is a container for metadata (data about data) and is placed between the <html> tag and the <body> tag.

HTML metadata is data about the HTML document. Metadata is not displayed. Metadata typically define the document title, character set, styles, scripts, and other meta information.

The HTML <head> element is a container for all the head elements: <title>, <style>, <meta>, <link>, <script>, and <base>.

Tag	Description
<head>	Defines information about the document
<title>	Defines the title of a document
<base>	Defines a default address or a default target for all links on a page
<link>	Defines the relationship between a document and an external resource
<meta>	Defines metadata about an HTML document
<script>	Defines a client-side script
<style>	Defines style information for a document

Fig. 1.8 (HTML Head element content)



```
<head>

    <title>Zaufanjinba</title>

    <meta charset="UTF-8">
    <meta name="description" content="Zaufanjinba Website">
    <meta name="keywords" content="Zaufanjinba, Hon. Adamu Alhaji Lawan">
    <meta name="author" content="Team Piccolo Global Enterprises">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <link rel="Icon" href="/images/favicon.ico" type="image/ico">
    <link rel="stylesheet" href="/css/master.css">

    <base href="https://zaufanjinba.org/" target="_blank">

    <style>
        body{
            font-size: 14px;
        }
    </style>

</head>
```

Fig. 1.9 (HTML head element)

The <title> element defines the title of the document, and is required in all HTML documents.

The <meta> element is used to specify which character set is used, page description, keywords, author, and other metadata.

Metadata is used by browsers (how to display content), by search engines (keywords), and other web services.

The <link> element is used to link to page Icon or external style sheets.

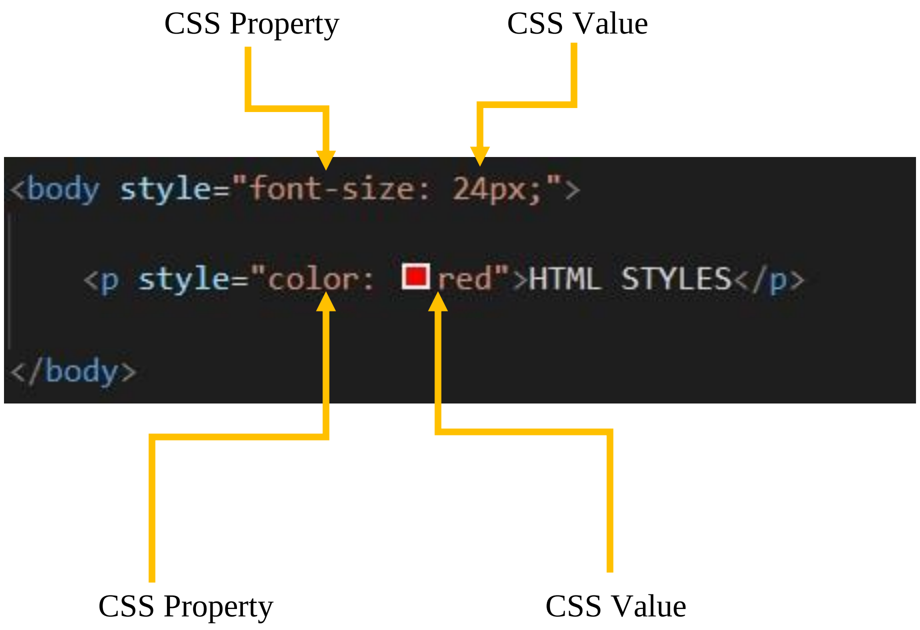
The <style> element is used to define style information for a single HTML page

The <script> element is used to define client-side JavaScript. We have not written any JavaScript code in *fig. 1.9* for now; don’t worry as we progress, we learn JavaScript in the second chapter of this book.

HTML Styles

Every day, web designers are designing beautiful websites and applications. There is a field called UI (User Interface) design in software development. A UI designer main purpose is design beautiful contents. They are a lot of nice designs on the internet. Having a skills in Graphic design will really help you to build fantastic UI.

The HTML style attribute is used to add styles to an element, such as font-size, color, and more.



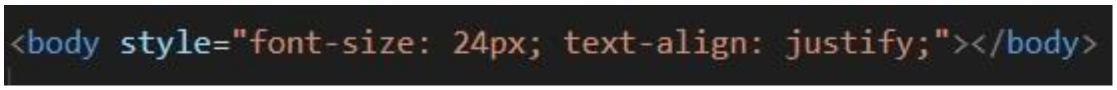
```
<body style="font-size: 24px;">

    <p style="color: red">HTML STYLES</p>

</body>
```

Fig. 1.10 (HTML Styles)

Foot Note:

When you add the style attribute to an HTML element, you first write the CSS property you want to apply to the element then a colon then the CSS value then you terminate it by using a semi-colon. We will learn more about CSS later in this book.

NOTE: You can add more than one CSS property to an element as in fig 1.11 below:



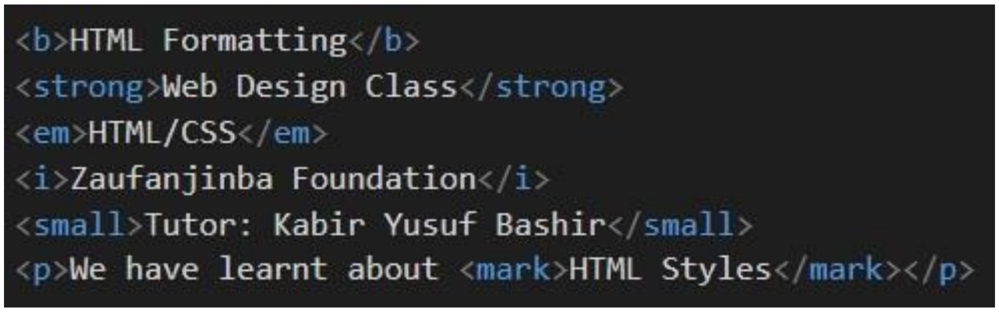
```
<body style="font-size: 24px; text-align: justify;"></body>
```

Fig 1.11 (more than one CSS property)

HTML Formatting

Text formatting is very useful in presenting information. In your paragraph, a need might come to emphasis on a certain point to get the attention of the reader.

HTML provides some elements that helps to make text formatting much easier.



```html
<b>HTML Formatting</b>
<strong>Web Design Class</strong>
<em>HTML/CSS</em>
<i>Zaufanjinba Foundation</i>
<small>Tutor: Kabir Yusuf Bashir</small>
<p>We have learnt about <mark>HTML Styles</mark></p>
```

Fig. 1.12 (HTML formatting)

The HTML <b> element defines bold text, without any extra importance.

The HTML <strong> element defines text with strong importance. The content inside is typically displayed in bold.

The HTML <em> element defines emphasized text. The content inside is typically displayed in italic.

The HTML <i> element defines a part of text in an alternate voice or mood. The content inside is typically displayed in italic.

The HTML <small> element defines smaller text.

The HTML <mark> element defines text that should be marked or highlighted.

The HTML <del> element defines text that has been deleted from a document. Browsers will usually strike a line through deleted text.

The HTML <ins> element defines a text that has been inserted into a document. Browsers will usually underline inserted text.

The HTML <sub> element defines subscript text. Subscript text appears half a character below the normal line, and is sometimes rendered in a smaller font. Subscript text can be used for chemical formulas, like H_2O.

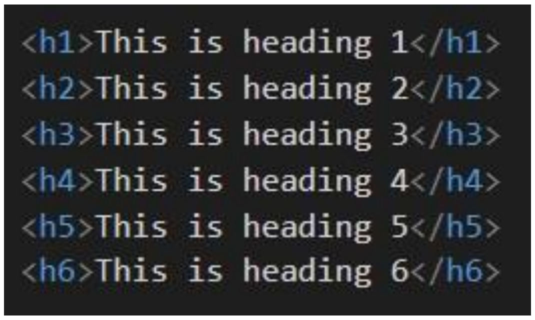
The HTML <sup> element defines superscript text. Superscript text appears half a character above the normal line, and is sometimes rendered in a smaller font. Superscript text can be used for footnotes, like WWW [1]

HTML Heading

HTML headings are titles or subtitles that you want to display on a webpage.

HTML headings are defined with the <h1> to <h6> tags.

<h1> defines the most important heading. <h6> defines the least important heading.



```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
```

Fig 1.13 (HTML Headings)

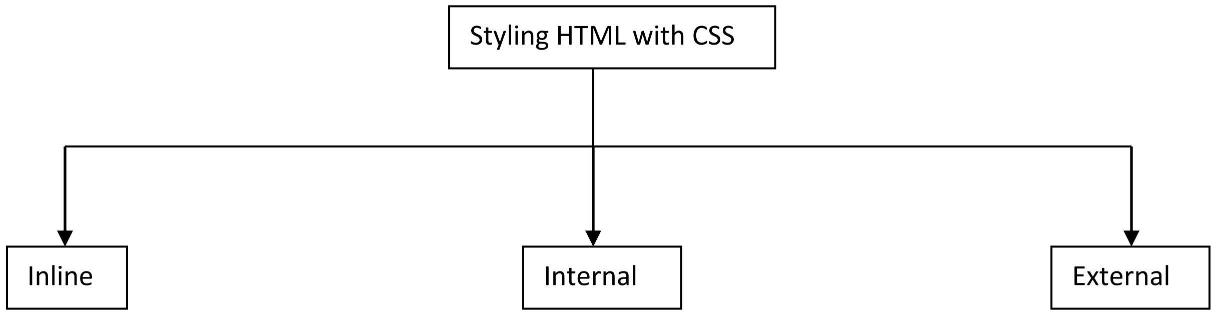
Foot Note:

Search engines use the headings to index the structure and content of your web pages. Users often skim a page by its headings. It is important to use headings to show the document structure. <h1> headings should be used for main headings, followed by <h2> headings, then the less important <h3>, and so on.

*Note: Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold***

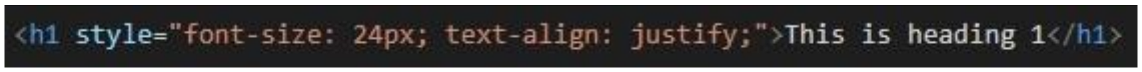
Styling Html with Css (Cascading Style Sheet)

You can add styling to your HTML in three (3) ways and we will explain when to use each of the three (3) ways in your design and which is the best approach.



Inline CSS

An inline CSS is used to apply a unique style to a single HTML element. An inline CSS uses the style attribute of an HTML element. The following example in *fig. 1.14* sets the font-size of the <h1> element to ***24px***, and text-align to ***justify***:



```
<h1 style="font-size: 24px; text-align: justify;">This is heading 1</h1>
```

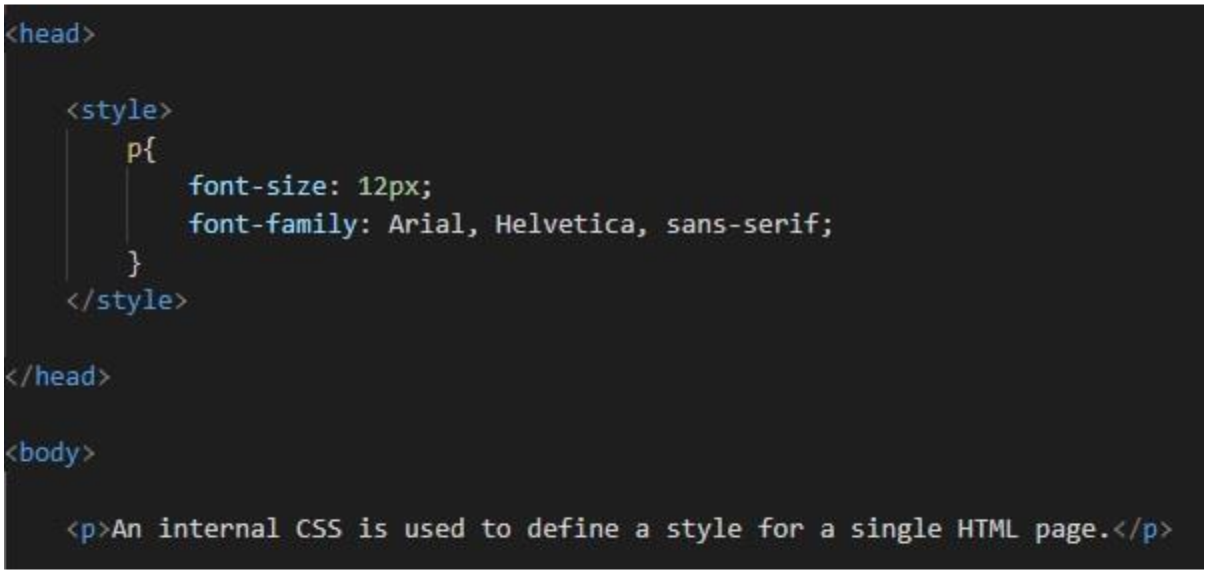
Fig 1.14 (Inline CSS)

Foot Note:

*The word **cascading** means that a style applied to a parent element will also apply to all children elements within the parent. So, if you set the color of the body text to "blue", all headings, paragraphs, and other text elements within the body will also get the same color (unless you specify something else)!*

Internal CSS

An internal CSS is used to define a style for a single HTML page. An internal CSS is defined in the <head> section of an HTML page, within a <style> element. The following example in *fig 1.15* sets the font-size of ALL the <p> elements (on that page) to ***12px***, and the font-family of ALL the <p> elements to ***Arial***, if ***Arial*** is not install on the system; it will set the font-family to ***Helvetica***, if ***Helvetica*** not found then it will be set to ***sans-serif***.

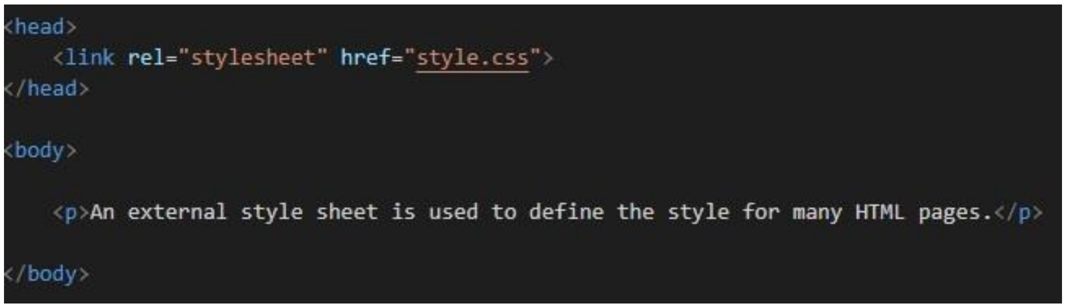


```
<head>

    <style>
        p{
            font-size: 12px;
            font-family: Arial, Helvetica, sans-serif;
        }
    </style>

</head>

<body>

    <p>An internal CSS is used to define a style for a single HTML page.</p>
```

Fig. 1.15 (Internal CSS)

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:

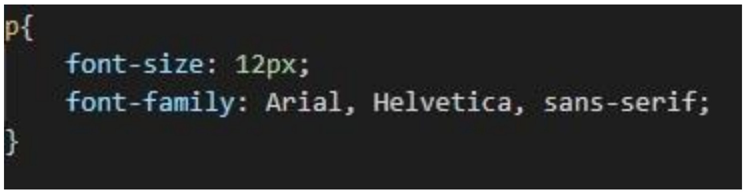


```
<head>
    <link rel="stylesheet" href="style.css">
</head>

<body>

    <p>An external style sheet is used to define the style for many HTML pages.</p>

</body>
```

Fig. 1.16 (External CSS)

The external style sheet can be written in any text editor like Visual Studio Code, Sublime, notepad++, etc. The file must **not contain any HTML code**, and must be saved with a **.css** extension.

Here is how the "styles.css" file looks like as in *fig. 1.17* below:



```
p{
    font-size: 12px;
    font-family: Arial, Helvetica, sans-serif;
}
```

Fig. 1.17 (style.css)

Summary

- When you want to style a particular HTML element on a page; the **Inline CSS** is the best approach to use.
- When you want to style a particular Web Page on your Website; the **Internal CSS** is the best approach to use.
- When you want to style the entire website; the **External CSS** is the best approach to use.

Recommendation

When you want to style an HTML element, a web page or the entire website; the best approach is to use the **External Styling,** because it is makes your code much easier to maintain and it is the standard most developers used.

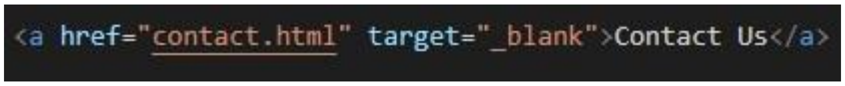
Foot Note:

*With an **external style sheet**, you can change the look of an entire web site, by changing one file.*

HTML Links

Links are found in nearly all web pages. Links allow users to click their way from page to page. HTML links are hyperlinks. You can click on a link and jump to another document or to even another website. When you move the mouse over a link, the mouse arrow will turn into a little hand.

The HTML <a> tag defines a hyperlink as in the *fig. 1.18* below:



```
<a href="contact.html" target="_blank">Contact Us</a>
```

Fig. 1.18 (HTML links)

The href attribute

The most important attribute of the <a> element is the href attribute, which indicates the link's destination. The *link text* is the part that will be visible to the reader. Clicking on the link text, will send the reader to the specified URL address as in *fig. 1.18 (HTML links)* above.

The target attribute

The target attribute specifies where to open the linked document. The target attribute can have one of the following values:

- _self - Default. Opens the document in the same window/tab as it was clicked
- _blank - Opens the document in a new window or tab
- _parent - Opens the document in the parent frame
- _top - Opens the document in the full body of the window

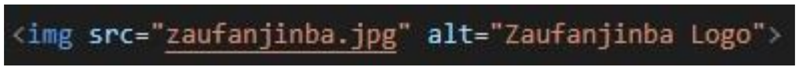
Foot Note:

- *You can use an **Image** or a **Button** as the link also.*
- *You can link to a document like a Portal Document Format (PDF).*

HTML Images

High resolution images improve the design and appearance of your website. They are a lot of free websites on the internet that you can download high resolution images for free. One of my favorite website is **pixabay.com.**

In HTML, images are defined with the <img> tag. The <img> tag is empty, it contains attributes only, and does not have a closing tag.



```
<img src="zaufanjinba.jpg" alt="Zaufanjinba Logo">
```

Fig 1.19 (HTML images)

The src attribute specifies the URL (web address) of the image as in *fig. 1.19* above.

The alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the alt attribute should describe the image:

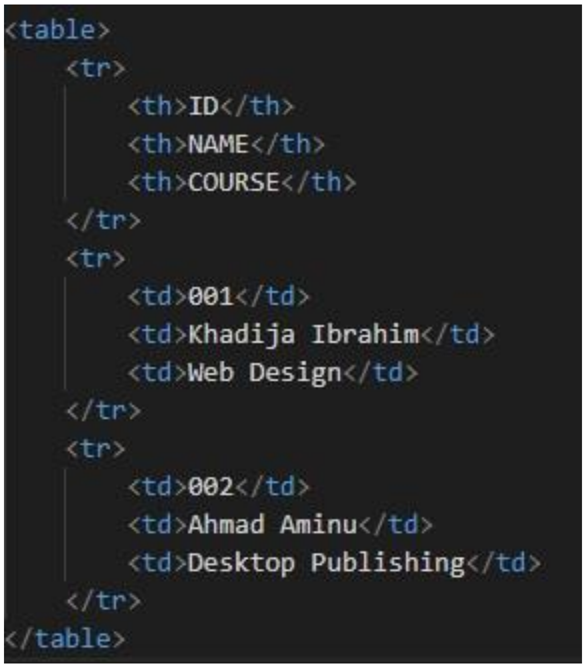
Foot Note:

- *The image can be of any image type example portable network graphic (PNG) etc.*
- *A screen reader is a software program that reads the HTML code, converts the text, and allows the user to "listen" to the content. Screen readers are useful for people who are visually impaired or learning disabled.*

HTML Tables

A table is a combination of rows and columns. A table is used to present information. In HTML, you can present your data using table.

An HTML table is defined with the <table> tag. Each table row is defined with the <tr> tag. A table header is defined with the <th> tag. By default, table headings are bold and centered. A table data/cell is defined with the <td> tag.



```
<table>
    <tr>
        <th>ID</th>
        <th>NAME</th>
        <th>COURSE</th>
    </tr>
    <tr>
        <td>001</td>
        <td>Khadija Ibrahim</td>
        <td>Web Design</td>
    </tr>
    <tr>
        <td>002</td>
        <td>Ahmad Aminu</td>
        <td>Desktop Publishing</td>
    </tr>
</table>
```

Fig 1.20 (HTML Table)

Foot Note:

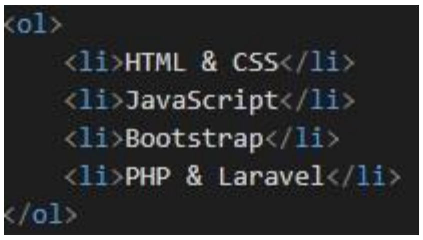
- *The <td> elements are the data containers of the table. They can contain all sorts of HTML elements; text, images, lists, other tables, etc.*

HTML Lists

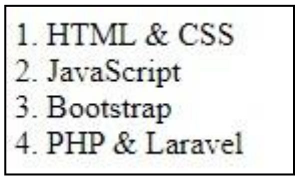
In HTML, we have mainly two (2) types of list. ***Ordered list*** and ***unordered*** list. HTML lists allow web authors to group a set of related items in lists.

Ordered list

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag. The list items will be marked with numbers by default as shown in *fig 1.21* below:



```
<ol>
    <li>HTML & CSS</li>
    <li>JavaScript</li>
    <li>Bootstrap</li>
    <li>PHP & Laravel</li>
</ol>
```

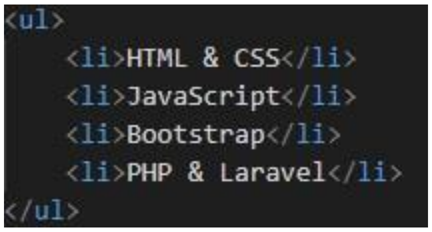


1. HTML & CSS
2. JavaScript
3. Bootstrap
4. PHP & Laravel

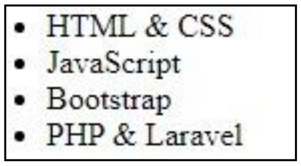
Fig 1.21 (HTML Ordered list)

Unordered list

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag. The list items will be marked with bullets (small black circles) by default as shown below in *fig. 1.22* below:



```
<ul>
    <li>HTML & CSS</li>
    <li>JavaScript</li>
    <li>Bootstrap</li>
    <li>PHP & Laravel</li>
</ul>
```



- HTML & CSS
- JavaScript
- Bootstrap
- PHP & Laravel

Fig 1.22 (HTML Unordered list)

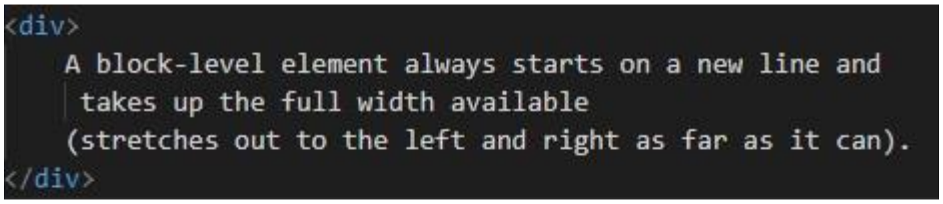
HTML Block and Inline Elements

Every HTML element has a default display value, depending on what type of element it is.

Block Level Element

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can). The <div> element is the widely used block level element. Below are examples of block level elements in HTML:

<address>	<article>	<aside>	<blockquote>	<canvas>	<dd>	<div>
<dl>	<dt>	<fieldset>	<figcaption>	<figure>	<footer>	<form>
<h1>-<h6>	<header>	<hr>	<li>	<main>	<nav>	
<noscript>	<ol>	<p>	<pre>	<section>	<table>	<tfoot>
<ul>	<video>					



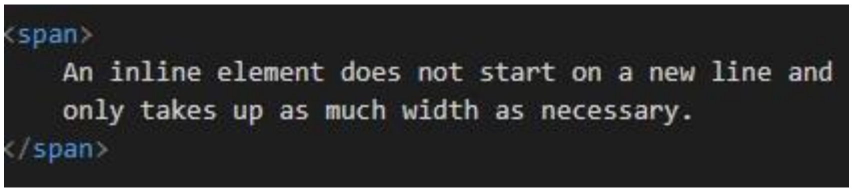
```
<div>
    A block-level element always starts on a new line and
     takes up the full width available
    (stretches out to the left and right as far as it can).
</div>
```

Fig 1.23 (HTML Block level element)

Inline Level Element

An inline element does not start on a new line and only takes up as much width as necessary. The <span> element is the widely used inline level element. Below are examples of inline level elements in HTML:

<a>	<abbr>	<acronym>	<b>	<bdo>	<big>	<br>
<button>	<cite>	<code>	<dfn>	<em>	<i>	<img>
<input>	<kbd>	<label>	<map>	<object>	<output>	<q>
<samp>	<script>	<select>	<small>	<span>	<strong>	<sub>
<sup>	<textarea>	<time>	<tt>	<var>		

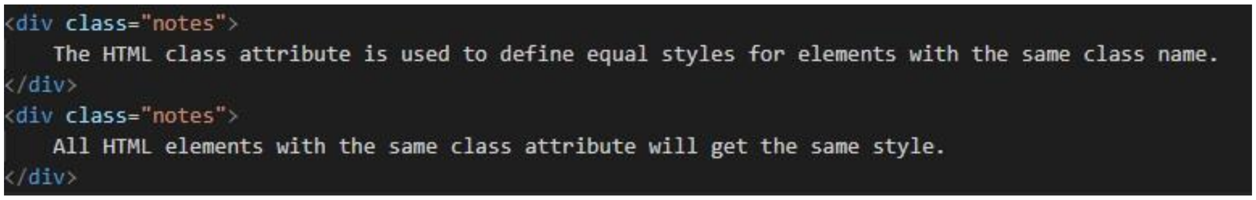


```html
<span>
    An inline element does not start on a new line and
    only takes up as much width as necessary.
</span>
```

Fig 1.24 (HTML Inline level element)

HTML Classes & Id

The HTML class attribute is used to define equal styles for elements with the same class name. All HTML elements with the same class attribute will get the same style.

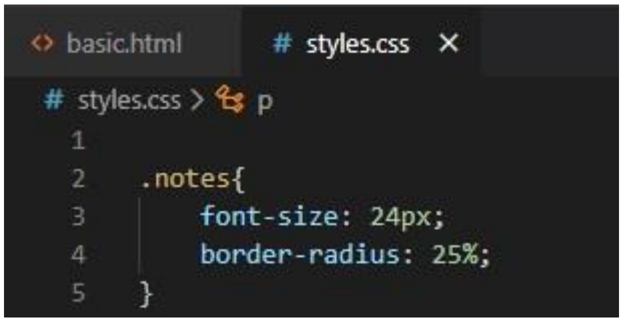


```html
<div class="notes">
    The HTML class attribute is used to define equal styles for elements with the same class name.
</div>
<div class="notes">
    All HTML elements with the same class attribute will get the same style.
</div>
```

Fig 1.25 (HTML class)

Any element with the class of "**notes**" as in f*ig 1.25* above will get same style. This will make your code much easier to maintain.

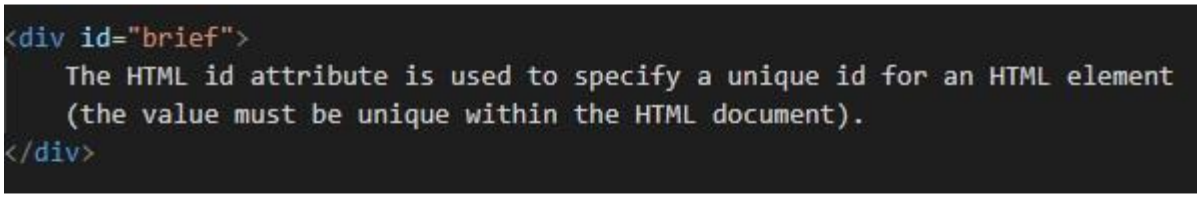
In CSS, to select elements with a specific class, write a period (.) character, followed by the name of the class as shown below in *fig 1.26*:



```css

.notes{
    font-size: 24px;
    border-radius: 25%;
}
```

Fig 1.26 (Selecting a specific element in CSS)

The HTML id attribute is used to specify a unique id for an HTML element (the value must be unique within the HTML document).

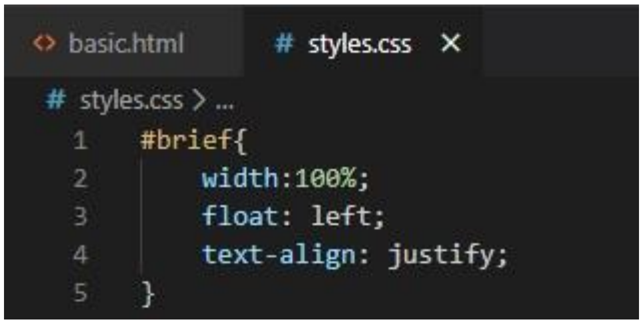


```
<div id="brief">
    The HTML id attribute is used to specify a unique id for an HTML element
    (the value must be unique within the HTML document).
</div>
```

Fig 1.27 (HTML id attribute)

The id attribute is used by CSS or JavaScript to perform certain tasks for the element with the specific id value.

In CSS, to select an element with a specific id, write a hash (#) character, followed by the id of the element as shown below in fig 1.28:

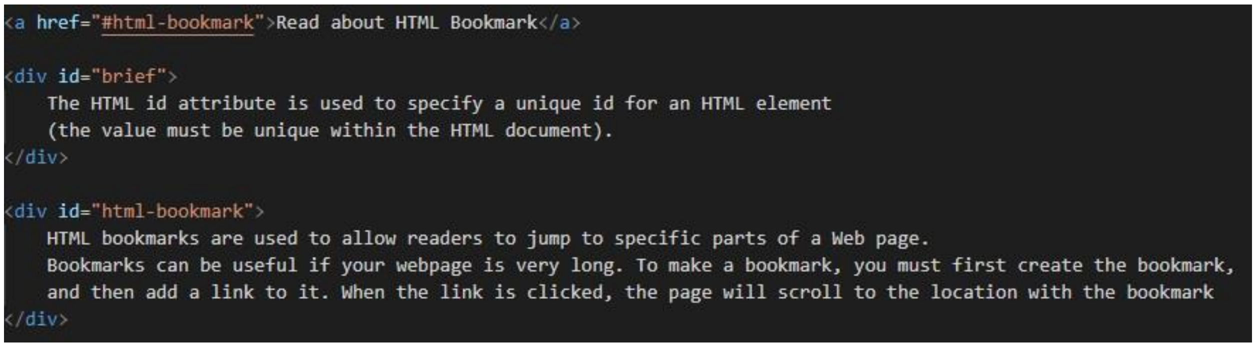


```
#brief{
    width:100%;
    float: left;
    text-align: justify;
}
```

Fig 1.28 (Selecting a specific id in CSS)

Bookmarks with Id and Links

HTML bookmarks are used to allow readers to jump to specific parts of a Web page. Bookmarks can be useful if your webpage is very long. To make a bookmark, you must first create the bookmark, and then add a link to it. When the link is clicked, the page will scroll to the location with the bookmark.



```
<a href="#html-bookmark">Read about HTML Bookmark</a>

<div id="brief">
    The HTML id attribute is used to specify a unique id for an HTML element
    (the value must be unique within the HTML document).
</div>

<div id="html-bookmark">
    HTML bookmarks are used to allow readers to jump to specific parts of a Web page.
    Bookmarks can be useful if your webpage is very long. To make a bookmark, you must first create the bookmark,
    and then add a link to it. When the link is clicked, the page will scroll to the location with the bookmark
</div>
```

Fig 1.29 (HTML bookmarks)

Foot Note:

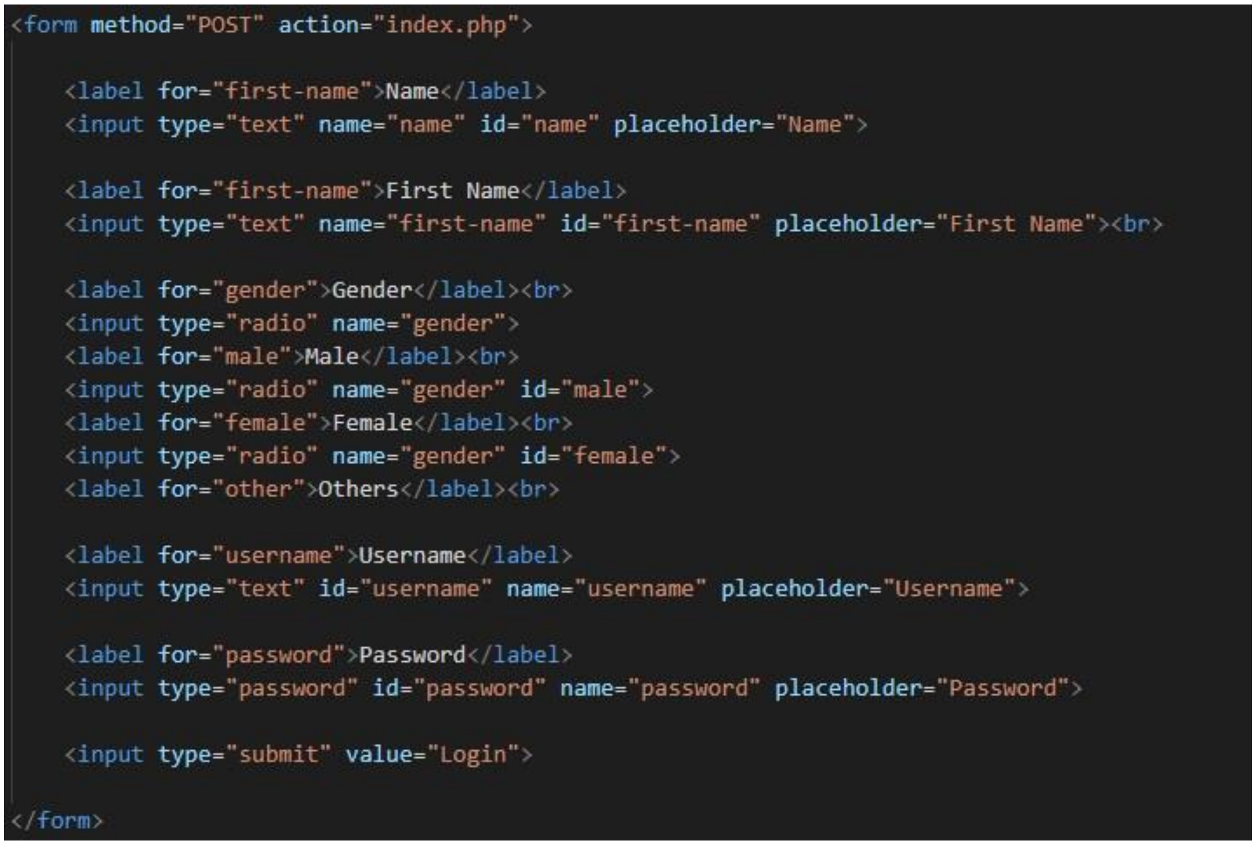
- The class attribute can be used on **any** HTML element.
- The class name is case sensitive. The class “notes” as not same as “Notes”.
- Different tags, like <h1> and <span>, can have the same class name and thereby share the same style.
- The id attribute can be used on any HTML element.
- The id value is case-sensitive.
- The id value must contain at least **one** character, and must **not** contain whitespace (spaces, tabs, etc.)
- An HTML element can only have one unique id that belongs to that single element, while a class name can be used by multiple elements.

HTML Forms

Forms are used to collect information from a user. Example of forms include a survey form, registration form, contact form etc. it depends on the kind of information you want to collect from the user.

An HTML form is used to collect user input. The user input can then be sent to a server for processing.

The HTML <form> element defines a form that is used to collect user input.



```html
<form method="POST" action="index.php">

    <label for="first-name">Name</label>
    <input type="text" name="name" id="name" placeholder="Name">

    <label for="first-name">First Name</label>
    <input type="text" name="first-name" id="first-name" placeholder="First Name"><br>

    <label for="gender">Gender</label><br>
    <input type="radio" name="gender">
    <label for="male">Male</label><br>
    <input type="radio" name="gender" id="male">
    <label for="female">Female</label><br>
    <input type="radio" name="gender" id="female">
    <label for="other">Others</label><br>

    <label for="username">Username</label>
    <input type="text" id="username" name="username" placeholder="Username">

    <label for="password">Password</label>
    <input type="password" id="password" name="password" placeholder="Password">

    <input type="submit" value="Login">

</form>
```

Fig 1.30 (HTML form element)

The method attribute

The method attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data.

The action attribute

The action attribute defines the action to be performed when the form is submitted. Usually, the form data is sent to a page on the server when the user clicks on the submit button. In the example shown above in *fig 1.30*, the form data is sent to a page on the server called "/index.php". This page contains a server-side script that handles the form data

The <label> Element

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 is focused 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

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

The <input> Element

The <input> element is the most important form element. The <input> element is displayed in several ways, depending on the **type** attribute.

The type attribute

The type attribute is used to specific what type of input you want to use. The type attribute has some values below:

button	checkbox	color	date	datetime	email	file
hidden	image	month	number	password	radio	range
reset	search	submit	tel	text	time	url
week						

The name attribute

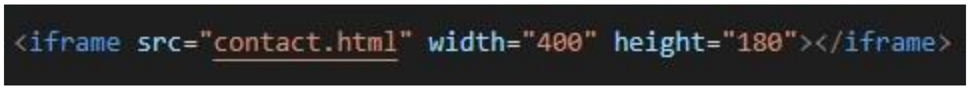
Each input field must have a name attribute to be submitted. If the name attribute is omitted, the data of that input field will not be sent at all.

The placeholder attribute

On any text input, you can also use an attribute called placeholder whose value is text that will be shown in the text box until the user clicks in that area. Older browsers simply ignore this attribute.

HTML Iframes

An HTML iframe is used to display a web page within a web page. An HTML iframe is defined with the <iframe> tag:



```
<iframe src="contact.html" width="400" height="180"></iframe>
```

Fig 1.31 (HTML iframes)

Most developers used the iframe tag to embed Google map on the contact page. You can embed your address using Google map and users can easily reach you.

Exercise on HTML

1. Why are HTML headings important to search engines?
2. Mention the ways to add style to your HTML page.
3. Why do most developers prefer using external styling?
4. If I want to open an HTML page in a new window, which of the value will I add to the target attribute?
5. Explain the alt attribute in <img> element.
6. Distinguish between the ordered list and unordered list in HTML.
7. List 10 examples of both block elements and inline level elements in HTML.
8. Distinguish between the id attribute and class attribute in HTML.
9. Explain the action attribute in <form> element.

CSS Introduction

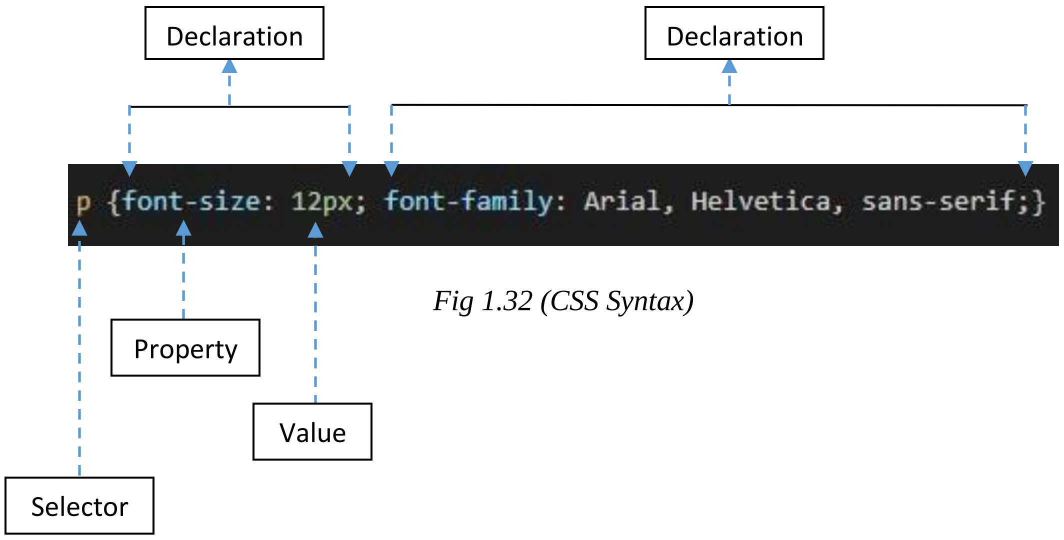
CSS stands for **C**ascading **S**tyle **S**heets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once.

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes. With CSS, you can design styles for different screen size like tablet, iPad, iPhone, Android, desktop, laptop etc. When the website is visited using a laptop, the display varies when visited on an iPhone or a tablet. It is called responsive web design and we will learn about it later in this book, Smile!

The style definitions are normally saved in external **.css** files.

CSS Syntax

A CSS rule-set consists of a selector and a declaration block as shown below in fig 1.32:



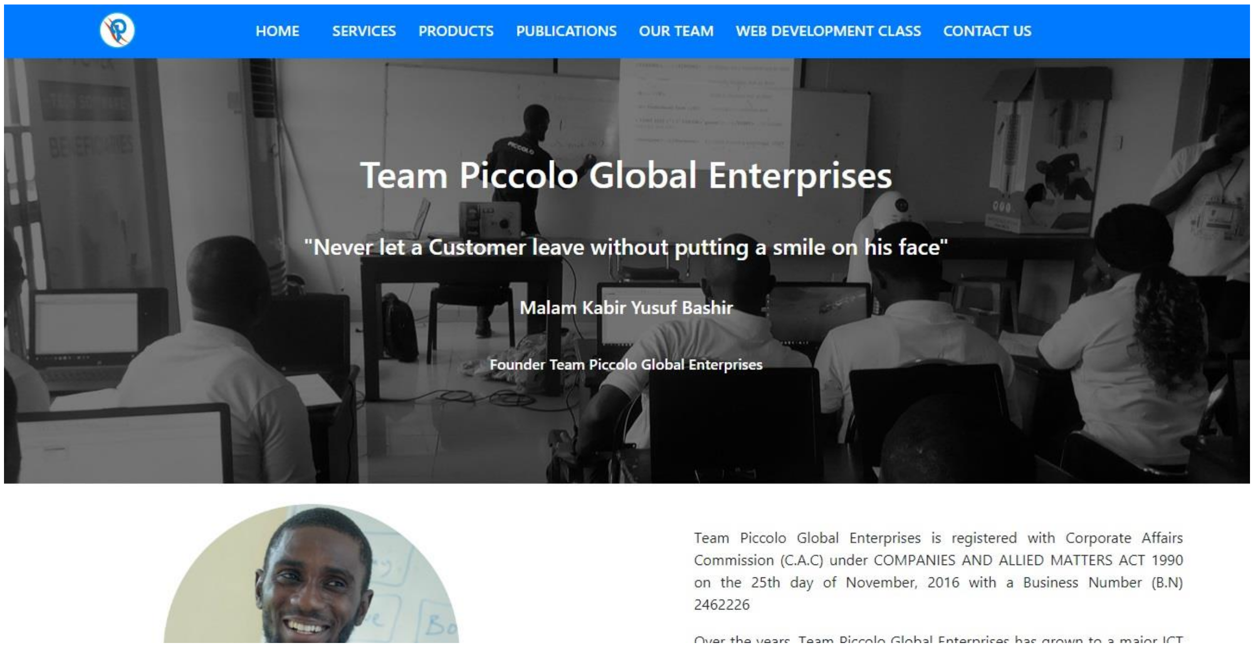
```
p {font-size: 12px; font-family: Arial, Helvetica, sans-serif;}
```

Fig 1.32 (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.

CSS Background

The CSS background properties are used to define the background effects for elements. The background can be an image or a color. In *fig. 1.33* below, an image is used as the background.

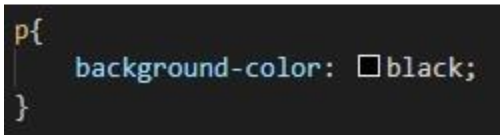

HOME
SERVICES
PRODUCTS
PUBLICATIONS
OUR TEAM
WEB DEVELOPMENT CLASS
CONTACT US
Team Piccolo Global Enterprises
"Never let a Customer leave without putting a smile on his face"
Malam Kabir Yusuf Bashir
Founder Team Piccolo Global Enterprises

Team Piccolo Global Enterprises is registered with Corporate Affairs Commission (C.A.C) under COMPANIES AND ALLIED MATTERS ACT 1990 on the 25th day of November, 2016 with a Business Number (B.N) 2462226

Fig 1.33 (CSS Background)

The background-color property specifies the background color of an element. A color is most often specified by:

- a valid color name - like "black"
- a HEX value - like "#000000"
- an RGB value - like "rgb(0,0,0)"



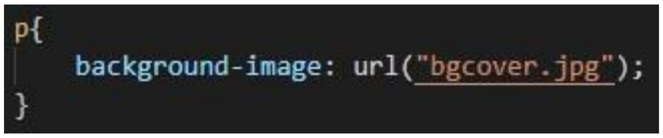
```
p{
    background-color: □black;
}
```

Fig 1.34 (CSS background-color property)

Any of the three (3) methods you decide to choose will work just fine. Whether using a valid color name like “black” or using the hexadecimal value like “#000000” or using RGB value like “rgb (0, 0, 0)”

Note: RGB stands for Red, Green and Blue.

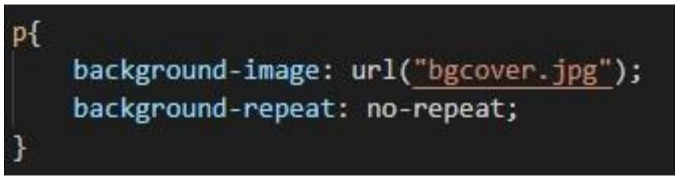
The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element. The background image for a page can be set as shown below in fig. 1.35:



```
p{
    background-image: url("bgcover.jpg");
}
```

Fig. 1.35 (CSS background-image property)

By default, the background-image property repeats an image both horizontally and vertically. You can alter this behavior. If you want the image to repeat horizontally only, there is a property called background-repeat with value of repeat-x. If you want the image to repeat vertically, change the value to repeat-y and if you don’t want the image to repeat, use the value of no-repeat as shown below in fig. 1.36:

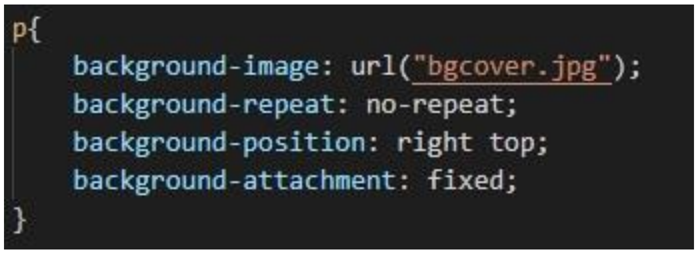


```
p{
    background-image: url("bgcover.jpg");
    background-repeat: no-repeat;
}
```

Fig. 1.36 (CSS background-repeat property)

In some scenarios, the background image might disturb the text on the website or make them not visible. The can alter the position of the image using the `background-position` property. You can get it a value of “right top”, “right bottom”, “left top” or “right top” depending on the position you want to the image to appear. You will see the code in *fig. 1.37* below:

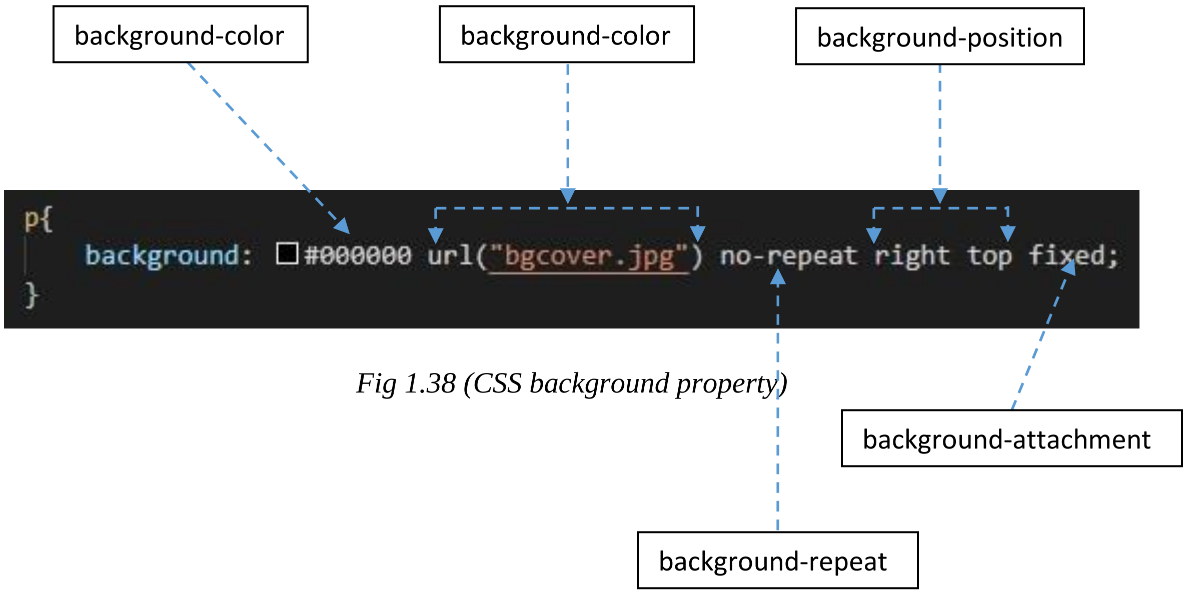
To specify that the background image should be fixed (will not scroll with the rest of the page), use the background-attachment property as shown below in *fig. 1.37*:



```
p{
    background-image: url("bgcover.jpg");
    background-repeat: no-repeat;
    background-position: right top;
    background-attachment: fixed;
}
```

Fig. 1.37 (CSS background-attachment property)

To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property. The shorthand property for background is background as shown below in *fig 1.38*:



```
p{
    background: □#000000 url("bgcover.jpg") no-repeat right top fixed;
}
```

Fig 1.38 (CSS background property)

Foot Note:

- *It does not matter if one of the property values is missing, as long as the other ones are in this order as shown above if fig. 1.38*

CSS Border

The CSS border properties allow you to specify the style, width, and color of an element's border. Below are some of examples of border:

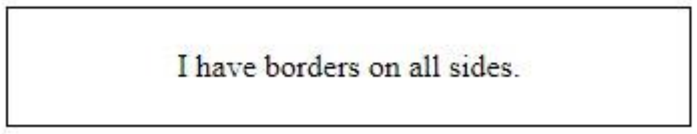


Fig. 1.39 (border on all sides)

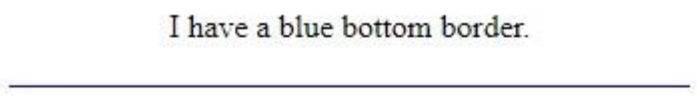


Fig. 1.40 (border-bottom)

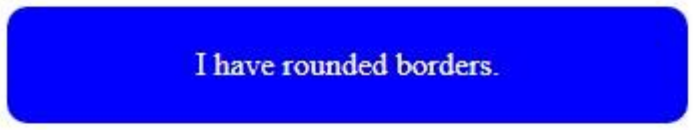


Fig. 1.41 (rounded borders)

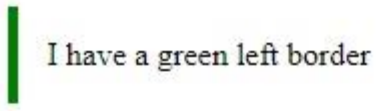


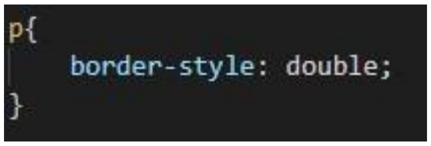
Fig. 1.42 (bottom-left)

The border-style property specifies what kind of border to display. The following values are allowed:

- dotted - Defines a dotted border.
- dashed - Defines a dashed border.
- solid - Defines a solid border.
- double - Defines a double border.

- groove - Defines a 3D grooved border. The effect depends on the border-color value.
- ridge - Defines a 3D ridged border. The effect depends on the border-color value.
- inset - Defines a 3D inset border. The effect depends on the border-color value.
- outset - Defines a 3D outset border. The effect depends on the border-color value.
- none - Defines no border.
- hidden - Defines a hidden border.

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border)

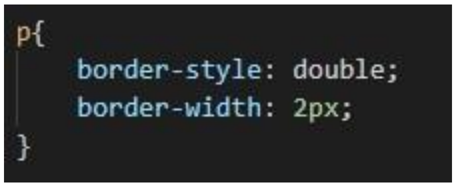


```
p{
    border-style: double;
}
```

Fig. 1.43 (CSS border-style property)

NOTE: *None of the OTHER CSS border properties described below will have ANY effect unless the border-style property is set. Make sure to set it first before any of the other border-style property.*

The border-width property specifies the width of the four borders. The width can be set as a specific size (in px, pt, cm, em, etc.) or by using one of the three pre-defined values: thin, medium, or thick. The border-width property can have from one to four values (for the top border, right border, bottom border, and the left border).



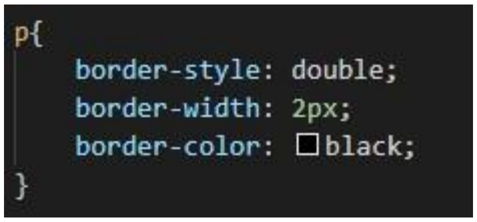
```
p{
    border-style: double;
    border-width: 2px;
}
```

Fig. 1.44 (CSS border-width property)

The border-color property is used to set the color of the four borders. The color can be set by:

- name - specify a color name, like "black"
- Hex - specify a hex value, like "#000000"
- RGB - specify a RGB value, like "rgb(0,0,0)"
- transparent

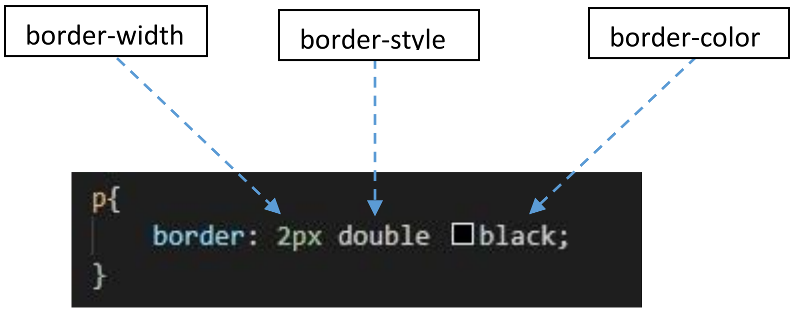
The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border). If border-color is not set, it inherits the color of the element.



```
p{
    border-style: double;
    border-width: 2px;
    border-color: □black;
}
```

Fig. 1.45 (CSS border-color property)

You can shorten the code in *fig. 1.45* above on a single line using the border property.



```
p{
    border: 2px double □black;
}
```

Fig. 1.46 (CSS border property)

CSS Margin

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

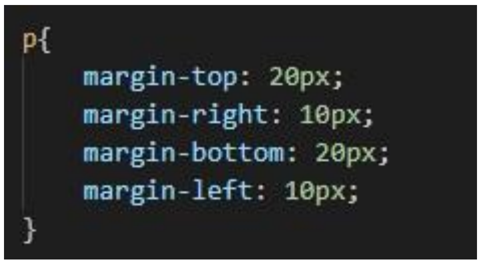
With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

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

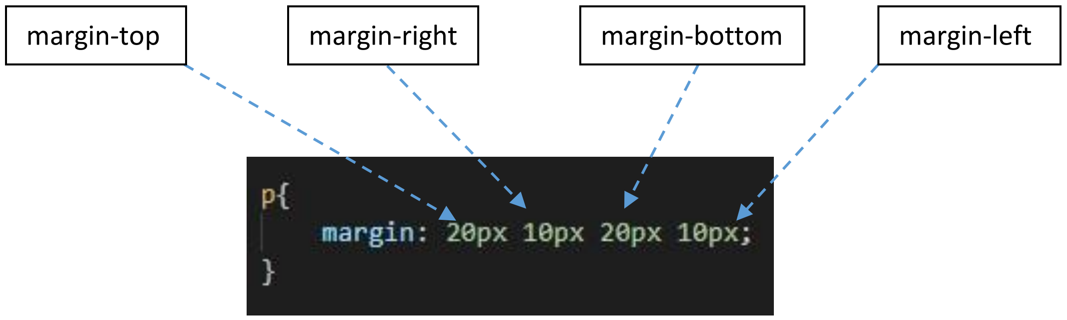
Note: Negative values are allowed.



```
p{
    margin-top: 20px;
    margin-right: 10px;
    margin-bottom: 20px;
    margin-left: 10px;
}
```

Fig. 1.47 (CSS margin property)

You can shorten the code above in *fig. 1.47*, it is possible to specify all the margin properties in one property using the margin property as shown below in *fig. 1.48*:



```
p{
    margin: 20px 10px 20px 10px;
}
```

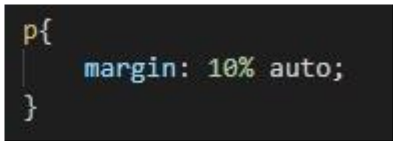
Fig. 1.48 (CSS margin property short form)

If the margin property has three values: **margin: 20px 10px 20px;** top-margin is 20px right and left margins are 10px, bottom-margin is 20px.

If the margin property has two values: **margin: 20px 10px;** top and bottom margins are 25px, right and left margins are 10px.

If the margin property has one value: **margin: 10px;** all four margins are 10px.

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.



```
p{
    margin: 10% auto;
}
```

Fig. 1.49 (auto value)

CSS Padding

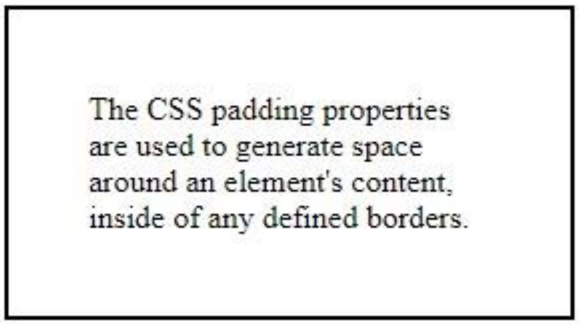

The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

Fig. 1.50 (CSS padding property)

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).

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.

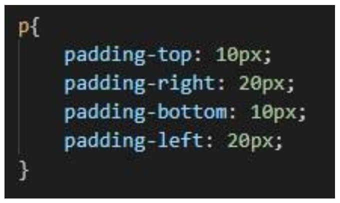
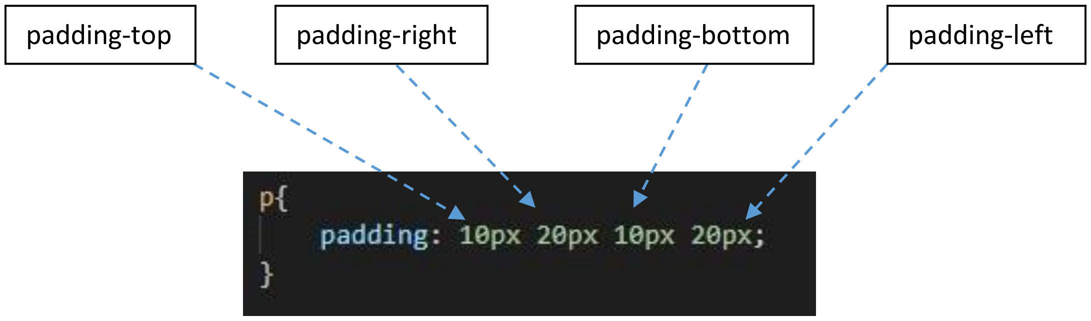

```
p{
    padding-top: 10px;
    padding-right: 20px;
    padding-bottom: 10px;
    padding-left: 20px;
}
```

Fig. 1.51 (CSS padding property)

You can shorten the code above in *fig. 1.51*, it is possible to specify all the margin properties in one property using the padding property as shown below in *fig. 1.52*:



```
p{
    padding: 10px 20px 10px 20px;
}
```

Fig. 1.52 (CSS padding property short form)

If the padding property has four values: **padding: 10px 20px 10px 20px;** top-padding is 10px, right-padding is 20px, bottom-padding is 10px, left-padding is 20px.

If the padding property has three values: **padding: 10px 20px 10px;** top-padding is 10px, right and left paddings are 20px, bottom padding is 10px.

If the padding property has two values: **padding: 10px 20px;** top and bottom paddings are 10px, right and left paddings are 20px.

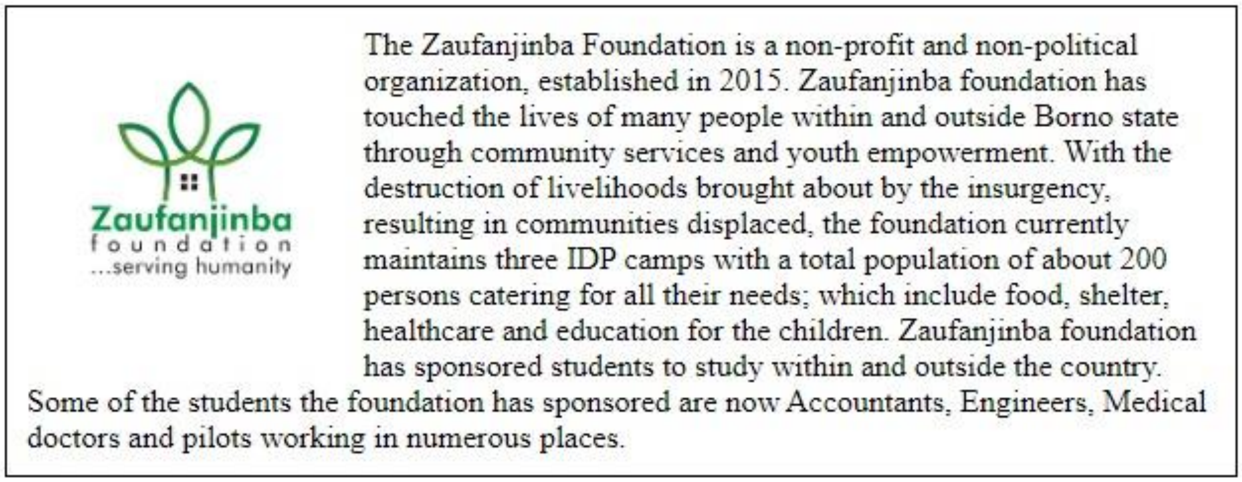
If the padding property has one value: **padding: 20px;** all four paddings are 20px.

Foot Note:

- *The CSS margin properties are used to generate space outside the element. While CSS padding properties are used to generate space inside the element.*
- *You can set "Negative" values to CSS margin properties but you* ***cannot*** *set "Negative" values to CSS padding properties.*
- *The more you used these two (2) CSS properties (margin, padding) in your work, the more it becomes second nature to you. Smile!*

CSS Float

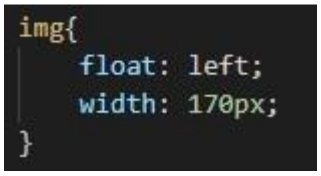
The CSS float property specifies how an element should float. The float property is used for positioning and formatting content e.g. let an image float left to the text in a container. In its simplest use, the float property can be used to wrap text around images.



The Zaufanjinba Foundation is a non-profit and non-political organization, established in 2015. Zaufanjinba foundation has touched the lives of many people within and outside Borno state through community services and youth empowerment. With the destruction of livelihoods brought about by the insurgency, resulting in communities displaced, the foundation currently maintains three IDP camps with a total population of about 200 persons catering for all their needs; which include food, shelter, healthcare and education for the children. Zaufanjinba foundation has sponsored students to study within and outside the country. Some of the students the foundation has sponsored are now Accountants, Engineers, Medical doctors and pilots working in numerous places.

Fig. 1.53 (CSS Float property)

The float property can have one of the following values: **left** - The element floats to the left of its container, **right** - The element floats to the right of its container, **none** - The element does not float (will be displayed just where it occurs in the text). This is default, **inherit** - The element inherits the float value of its parent.



```
img{
    float: left;
    width: 170px;
}
```

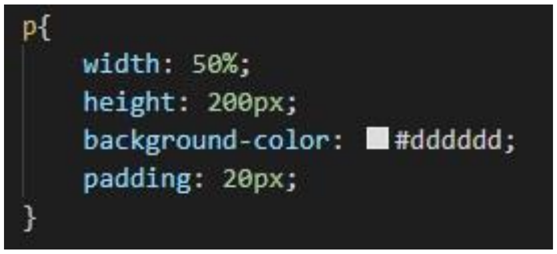
Fig. 1.54 (CSS float property example)

Foot Note: *The CSS clear property specifies what elements can float beside the cleared element and on which side.*

CSS Height and Width

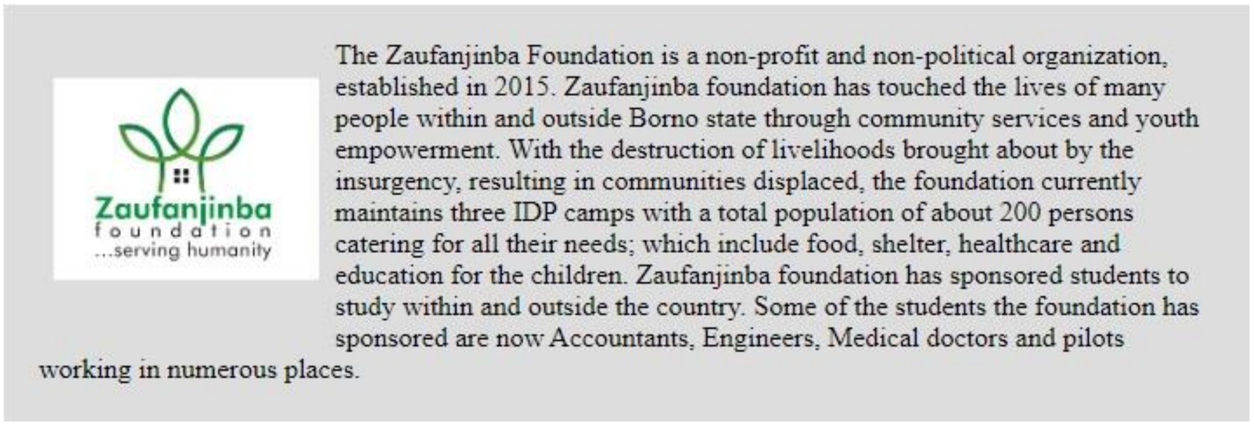
The height and width properties are used to set the height and width of an element. The height and width properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element.

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



```
p{
    width: 50%;
    height: 200px;
    background-color: ■#dddddd;
    padding: 20px;
}
```

Fig. 1.55 (CSS height & width property)



The Zaufanjinba Foundation is a non-profit and non-political organization, established in 2015. Zaufanjinba foundation has touched the lives of many people within and outside Borno state through community services and youth empowerment. With the destruction of livelihoods brought about by the insurgency, resulting in communities displaced, the foundation currently maintains three IDP camps with a total population of about 200 persons catering for all their needs; which include food, shelter, healthcare and education for the children. Zaufanjinba foundation has sponsored students to study within and outside the country. Some of the students the foundation has sponsored are now Accountants, Engineers, Medical doctors and pilots working in numerous places.

Fig. 1.56 (CSS height & width property example)

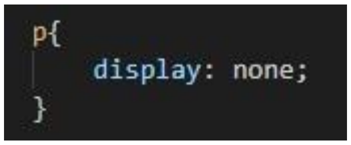
Foot Note:

Remember that the height and width properties do not include padding, borders, or margins! They set the height/width of the area inside the padding, border, and margin of the element!

CSS Display

The display property is the most important CSS property for controlling layout.
The display property specifies if/how an element is displayed. Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline.

Hiding an element can be done by setting the display property to none. The element will be hidden, and the page will be displayed as if the element is not there:



```
p{
    display: none;
}
```

Fig. 1.57 (CSS display property)

CSS Position

The position property specifies the type of positioning method used for an element. There are four different position values: static, relative, fixed, absolute.

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.

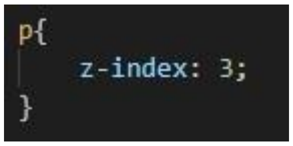
HTML elements are positioned static by default. Static positioned elements are not affected by the top, bottom, left, and right properties. An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

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.

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.

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.

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:



```
p{
    z-index: 3;
}
```

Fig. 1.58 (CSS z-index property)

An element with greater stack order is always in front of an element with a lower stack order.

Foot Note:

If two positioned elements overlap without a z-index specified, the element positioned last in the HTML code will be shown on top.

CSS Text

In this section, we will learn about some CSS properties used for text formatting. CSS properties like color, text-align, text-decoration, text-transform, text-indent, letter-spacing, line-height, direction and word-spacing.

The color property is used to set the color of the text. With CSS, a color is most often specified by: a color name - like "red", a HEX value - like "#ff0000", an RGB value - like "rgb(255,0,0)".

Note:

- The default text color for a page is defined in the body selector.
- For W3C compliant CSS: If you define the color property, you must also define the background-color property.

The text-align property is used to set the horizontal alignment of a text. A text can be left or right aligned, centered, or justified. The value of text-align property includes: left, right, center and justify. If you have used a word processing software like Microsoft word, the text-align property works exactly as in Microsoft word.

The text-decoration property is used to set or remove decorations from text. The value text-decoration: none; is often used to remove underlines from links. The value of text-decoration property includes: underline, line-through, over-line.

Note: It is not recommended to underline text that is not a link, as this often confuses the reader.

The text-transform property is used to specify uppercase and lowercase letters in a text. It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word. The value of text-transform property includes: capitalize, lowercase, none, uppercase, initial, inherit, unset.

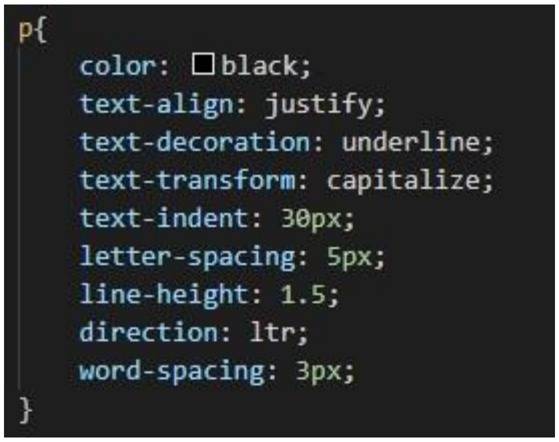
The text-indent property is used to specify the indentation of the first line of a text. You use the unit of measurement like px, cm, em, etc. when specifying the value of the property: text-indent: 30px;

The letter-spacing property is used to specify the space between the characters in a text. You use the unit of measurement like px, cm, em, etc. when specifying the value of the property: letter-spacing: 5px;

The line-height property is used to specify the space between lines. Using the line-height, you **don't** specify the unit of measurement like cm, px or em: line-height: 1.5;

The direction property is used to change the text direction of an element. Some of the values you can use on the direction property includes: rtl and ltr.

The word-spacing property is used to specify the space between the words in a text. You use the unit of measurement like px, cm, em, etc. when specifying the value of the property: word-spacing: 3px;



```
p{
    color: ■black;
    text-align: justify;
    text-decoration: underline;
    text-transform: capitalize;
    text-indent: 30px;
    letter-spacing: 5px;
    line-height: 1.5;
    direction: ltr;
    word-spacing: 3px;
}
```

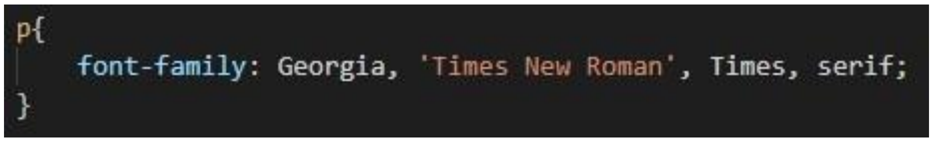
Fig. 1.59 (CSS Text properties)

CSS Font

The CSS font properties define the font family, boldness, size, and the style of a text. With CSS, you can alter the font family of your web page for make your website much attractive and user friendly.

The font family of a text is set with the font-family property. The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on.

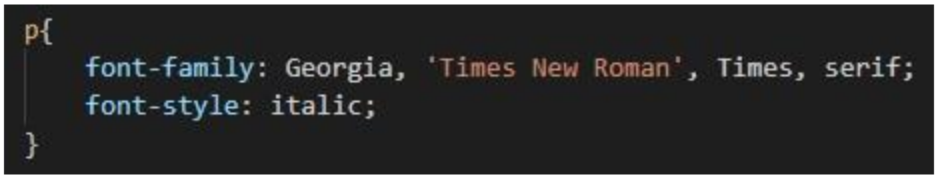
Note: If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".



```
p{
    font-family: Georgia, 'Times New Roman', Times, serif;
}
```

Fig. 1.60 (CSS font-family property)

The font-style property is mostly used to specify italic text. This property has three values: **normal** - The text is shown normally, **italic** - The text is shown in italics, **oblique** - The text is "leaning" (oblique is very similar to italic, but less supported)

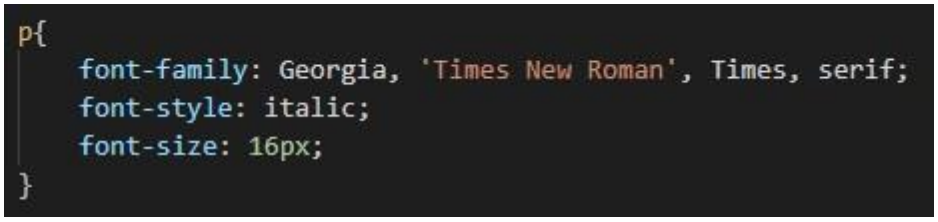


```
p{
    font-family: Georgia, 'Times New Roman', Times, serif;
    font-style: italic;
}
```

Fig. 1.61 (CSS font-style property)

The font-size property sets the size of the text. Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs. You can cm, em, px or % as the unit of measurement.

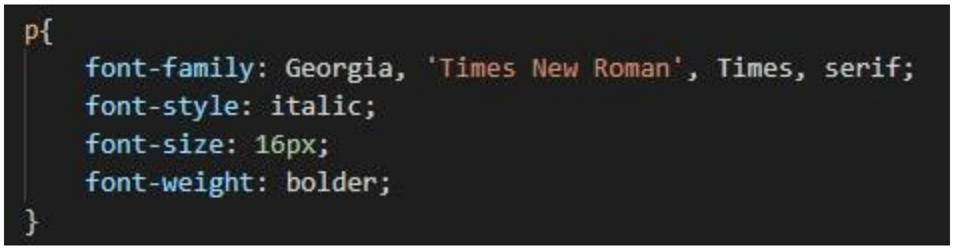


```
p{
    font-family: Georgia, 'Times New Roman', Times, serif;
    font-style: italic;
    font-size: 16px;
}
```

Fig. 1.62 (CSS font-size property)

Note: Many developers use **em** instead of **pixels**. The em size unit is recommended by the W3C.

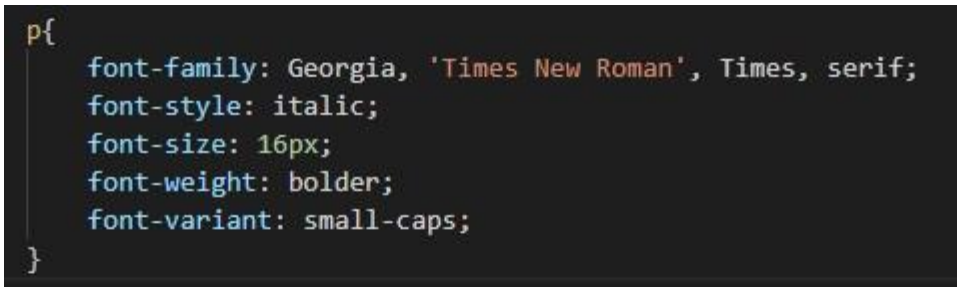
The font-weight property specifies the weight of a font: This property has some values like bold, bolder and normal.



```
p{
    font-family: Georgia, 'Times New Roman', Times, serif;
    font-style: italic;
    font-size: 16px;
    font-weight: bolder;
}
```

Fig. 1.63 (CSS font-weight property)

The font-variant property specifies whether or not a text should be displayed in a small-caps font. In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.



```
p{
    font-family: Georgia, 'Times New Roman', Times, serif;
    font-style: italic;
    font-size: 16px;
    font-weight: bolder;
    font-variant: small-caps;
}
```

Fig. 1.64 (CSS font-variant property)

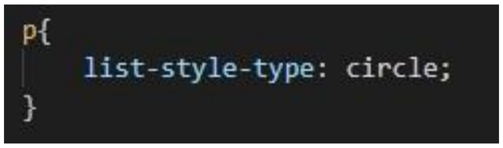
CSS Links

We have already learnt in HTML, there are two main types of lists: **unordered lists** (<ul>) - the list items are marked with bullets and **ordered lists** (<ol>) - the list items are marked with numbers or letters.

The CSS list properties allow you to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items

The list-style-type property specifies the type of list item marker. This property has some values like: circle, decimal, none, square, upper-roman, lower-alpha, etc.



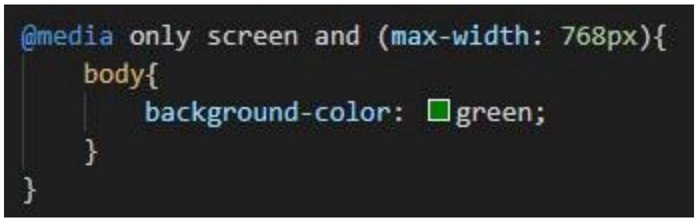
```
p{
    list-style-type: circle;
}
```

Fig. 1.65 (CSS list-style-type property)

CSS Media Queries (Responsive Design)

Media query is a CSS technique introduced in CSS3. It uses the @media rule to include a block of CSS properties only if a certain condition is true.

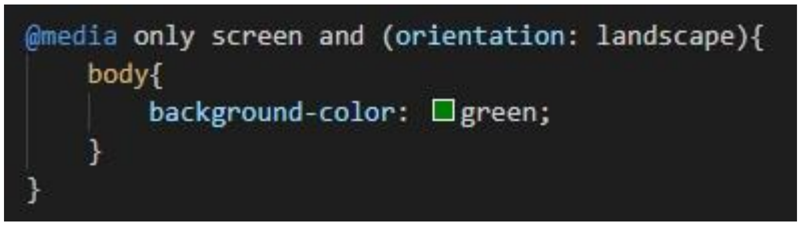
If the browser window is 768px or smaller, the background color will be green as shown in fig. 1.66 below:



```
@media only screen and (max-width: 768px){
    body{
        background-color: green;
    }
}
```

Fig. 1.66 (CSS media query)

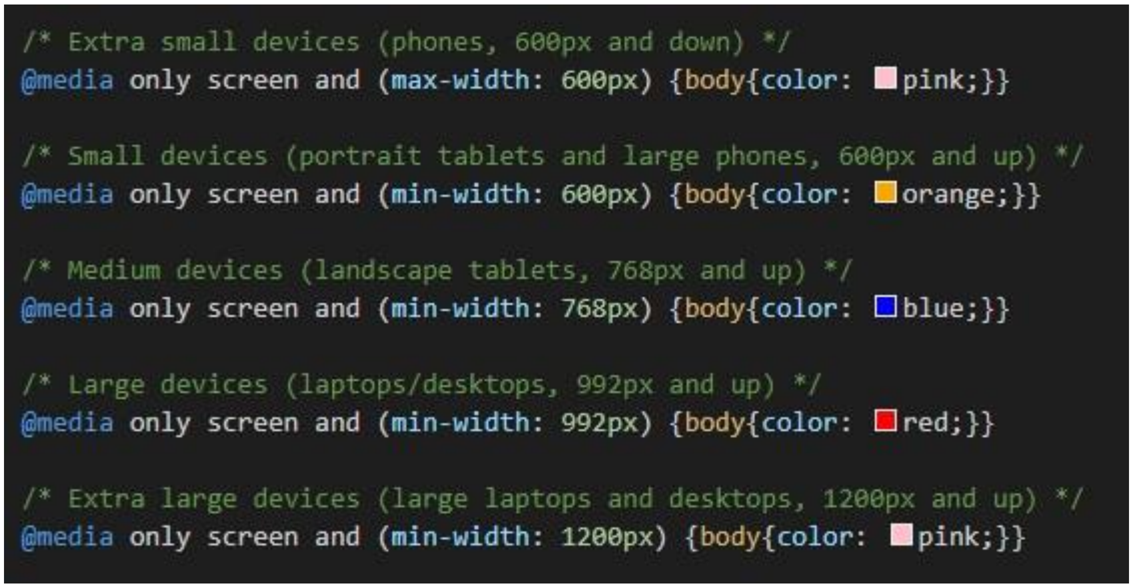
Media queries can also be used to change layout of a page depending on the orientation of the browser. You can have a set of CSS properties that will only apply when the browser window is wider than its height, a so called "**Landscape**" orientation.



```
@media only screen and (orientation: landscape){
    body{
        background-color: green;
    }
}
```

Fig. 1.67 (CSS media query landscape orientation)

You can add as many breakpoints as you like. There are tons of screens and devices with different heights and widths, so it is hard to create an exact breakpoint for each device. To keep things simple you could target five groups as shown below in fig. 1.68.



```
/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {body{color: pink;}}

/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {body{color: orange;}}

/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {body{color: blue;}}

/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {body{color: red;}}

/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {body{color: pink;}}
```

Fig. 1.68 (CSS Media Query breakpoint)

Foot Note:

- *Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices).*
- *Instead of changing styles when the width gets* smaller *than 768px, we should change the design when the width gets* larger *than 768px. This will make our design Mobile First.*

Exercise on CSS

1. To use an image as your background in CSS, which of the CSS property will you use?
2. Distinguish between CSS margin and CSS padding.
3. List all the values of the float property in CSS that you know.
4. Explain CSS width and CSS height.
5. Write short note on CSS display and CSS position.
6. Which of the CSS text formatting property will I use to make my heading underlined?
7. To increase the font-size of a web page, which of the CSS property will you use?
8. What do you understand by responsive web design?
9. Why is it a good practice to start with mobile design first in responsive web design?

Project on HTML & CSS

With your knowledge of HTML & CSS, create a personal portfolio web page using HTML & CSS.

REFERENCES

1. https://www.techopedia.com/definition/5613/web
2. https://en.wikipedia.org/wiki/Text_editor
3. https://www.w3schools.in/html-tutorial/history/#:~:text=HTML%20was%20created%20by%20Sir,evolved%20with%20various%20versions%20updating.
4. https://www.w3schools.com/html/html_intro.asp
5. https://www.tutorialrepublic.com/html-tutorial/html-elements.php
6. https://www.w3schools.com/html/html_elements.asp
7. https://www.w3schools.com/html/html_attributes.asp
8. https://www.w3schools.com/tags/ref_language_codes.asp
9. https://www.w3schools.com/html/html_head.asp
10. https://www.w3schools.com/html/html_styles.asp
11. https://www.w3schools.com/html/html_formatting.asp
12. https://www.w3schools.com/html/html_headings.asp
13. https://www.w3schools.com/html/html_css.asp
14. https://www.w3schools.com/html/html_links.asp
15. https://www.w3schools.com/html/html_images.asp
16. https://www.w3schools.com/html/html_lists.asp
17. https://www.w3schools.com/html/html_blocks.asp
18. https://www.w3schools.com/html/html_classes.asp
19. https://www.w3schools.com/html/html_id.asp
20. https://www.w3schools.com/html/html_forms.asp
21. HTML & CSS design and build websites by Jon Duckett. Forms page 168 definition of placeholder attribute.
22. https://www.w3schools.com/html/html_iframe.asp
23. https://www.w3schools.com/css/default.asp

24. https://www.w3schools.com/css/css_intro.asp
25. https://www.w3schools.com/css/css_syntax.asp
26. https://www.w3schools.com/css/css_background.asp
27. https://www.w3schools.com/css/css_border.asp
28. https://www.w3schools.com/css/css_margin.asp
29. https://www.w3schools.com/css/css_padding.asp
30. https://www.w3schools.com/css/css_float.asp
31. https://www.w3schools.com/css/css_dimension.asp
32. https://www.w3schools.com/css/css_display_visibility.asp
33. https://www.w3schools.com/css/css_positioning.asp
34. https://www.w3schools.com/css/css_text.asp
35. https://www.w3schools.com/css/css_font.asp
36. https://www.w3schools.com/css/css_list.asp
37. https://www.w3schools.com/css/css_rwd_mediaqueries.asp

Chapter 2: JavaScript

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997. ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

JavaScript and Java are completely different languages, both in concept and design.

JavaScript is one of the 3 languages all web developers **must** learn:

- HTML to define the content of web pages.
- CSS to specify the layout of web pages.
- JavaScript to program the behavior of web pages.

Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

Foot Note:

- *You don't have to get or download JavaScript.*
- *JavaScript is already running in your browser on your computer, on your tablet, and on your smart-phone. Free to use for everyone.*
- *JavaScript accepts both double and single quotes.*

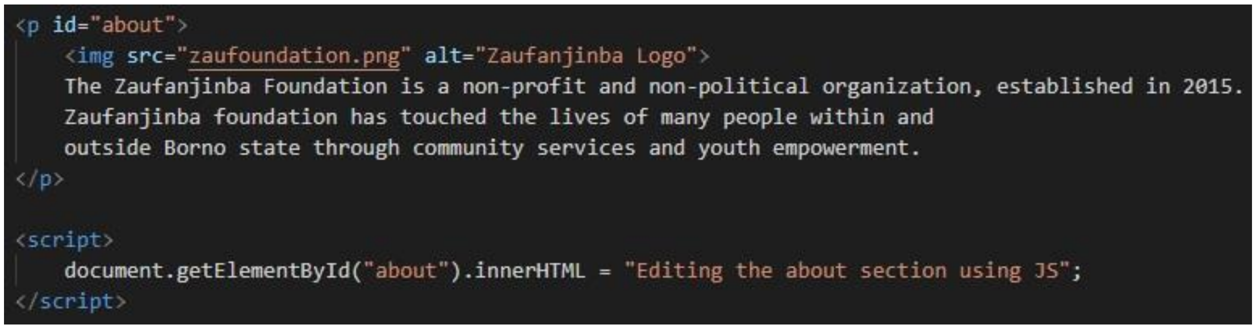
JS Output

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

To access an HTML element, JavaScript can use the document.getElementById(id) method. The id attribute defines the HTML element.

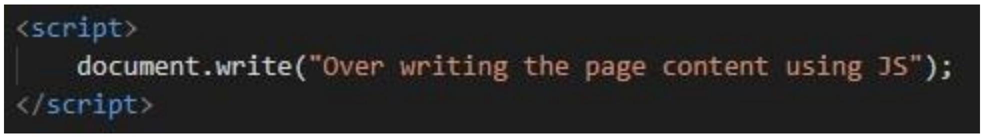
The innerHTML property defines the HTML content.



```
<p id="about">
    <img src="zaufoundation.png" alt="Zaufanjinba Logo">
    The Zaufanjinba Foundation is a non-profit and non-political organization, established in 2015.
    Zaufanjinba foundation has touched the lives of many people within and
    outside Borno state through community services and youth empowerment.
</p>

<script>
    document.getElementById("about").innerHTML = "Editing the about section using JS";
</script>
```

Fig. 2.1 (JS innerHTML method)

For testing purposes, it is convenient to use document.write().



```
<script>
    document.write("Over writing the page content using JS");
</script>
```

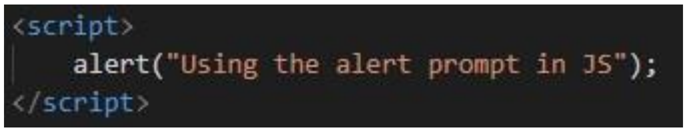
Fig. 2.2 (JS document.write method)

Note:

- Using document.write() after an HTML document is loaded, will delete all existing HTML

You can use an alert box to display data using the window.alert() property in JS. You can skip the window keyword.

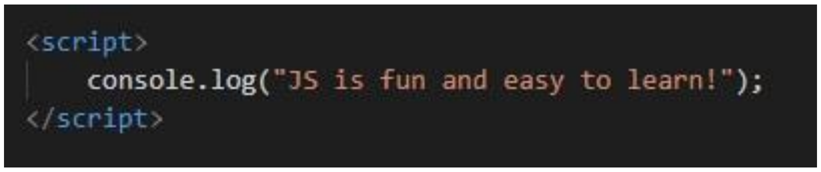
In JavaScript, the window object is the global scope object; that means that variables, properties, and methods by default belongs to the window object. This also means that specifying the window keyword is optional.



```
<script>
    alert("Using the alert prompt in JS");
</script>
```

Fig. 2.3 (JS alert method)

For debugging purposes, you can call the console.log() method in the browser to display data. We will learn about debugging in JS later in this book! Smile.



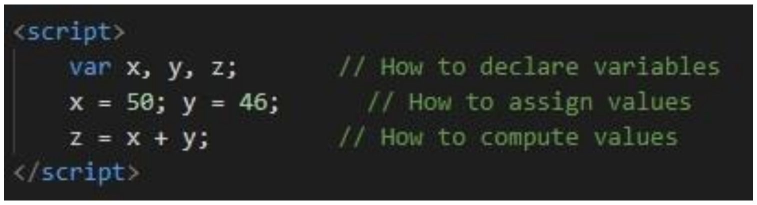
```
<script>
    console.log("JS is fun and easy to learn!");
</script>
```

Fig. 2.4 (JS console.log method)

You can call the window.print() method in the browser to print the content of the current window. If you want to print the content of the web page, you can call the window.print() method.

JS Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed.

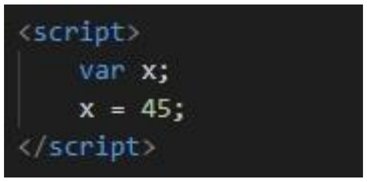


```
<script>
    var x, y, z;         // How to declare variables
    x = 50; y = 46;       // How to assign values
    z = x + y;           // How to compute values
</script>
```

Fig. 2.5 (JS syntax)

JavaScript **keywords** are used to identify actions to be performed. **Keywords** are reserved words in programming. You cannot use a **keyword** when declaring variables in JS, examples of keywords in JS includes: var, let, switch, loop, for, if, etc. The var keyword tells the browser to create variables.

In a programming language, **variables** are used to **store** data values. JavaScript uses the var keyword to **declare** variables. An **equal sign** is used to **assign values** to variables. In *fig. 2.6* below, **x** is defined as a variable. Then, **x** is assigned (given) the value **45**:



```
<script>
    var x;
    x = 45;
</script>
```

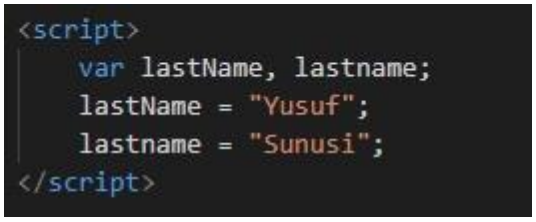
Fig. 2.6 (declaring variable in JS)

Identifiers are names. In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages. In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($). Subsequent characters may be letters, digits, underscores, or dollar signs.

Note: Numbers are not allowed as the first character. This way JavaScript can easily distinguish identifiers from numbers.

All JavaScript identifiers are **case sensitive**. The variables lastName and lastname, are two different variables.



```
<script>
    var lastName, lastname;
    lastName = "Yusuf";
    lastname = "Sunusi";
</script>
```

Fig. 2.7 (JS case sensitivity)

Programmers have used different ways of joining multiple words into one variable name:

- **Hyphens:** first-name, last-name, master-card, inter-city;
- **Underscore:** first_name, last_name, master_card, inter_city;
- **Upper Camel Case (Pascal Case):** FirstName, LastName, MasterCard, InterCity;
- **Lower Camel Case:** JavaScript programmers tend to use camel case that starts with a lowercase letter: firstName, lastName, masterCard, interCity.

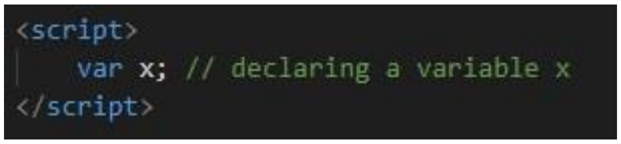
Foot Note:

- ***Hyphens** naming are not allowed in JavaScript. They are reserved for subtractions.*
- *I will be using **Lower Camel Case** naming throughout this book.*

JS Comment

JavaScript comments can be used to explain JavaScript code, and to make it more readable. JavaScript comments can also be used to prevent execution, when testing alternative code. Is a good practice to comment your code as this will make your code much easier to maintain. There are two (2) types of comments in most programming languages: **single line** comment and **block** comment.

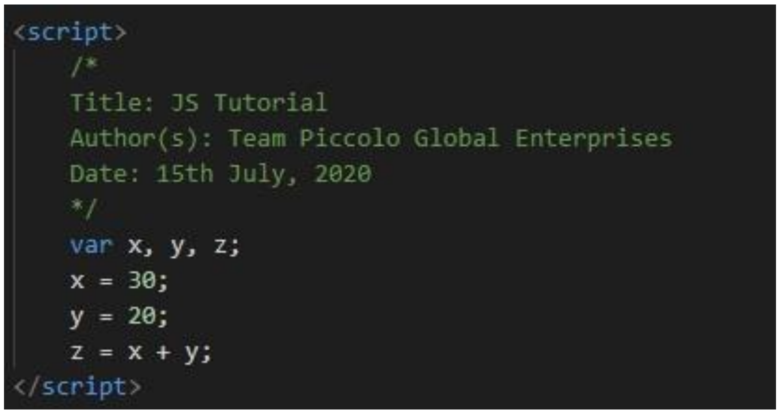
Single line comments start with //. Any text between // and the end of the line will be ignored by JavaScript (will not be executed). The code below in *fig. 2.8* uses a single-line comment.



```
<script>
    var x; // declaring a variable x
</script>
```

Fig. 2.8 (single line comment in JS)

Multi-line comments start with /* and end with */. Any text between /* and */ will be ignored by JavaScript. The code below in *fig 2.9* uses a multi-line comment (a comment block).



```
<script>
    /*
    Title: JS Tutorial
    Author(s): Team Piccolo Global Enterprises
    Date: 15th July, 2020
    */
    var x, y, z;
    x = 30;
    y = 20;
    z = x + y;
</script>
```

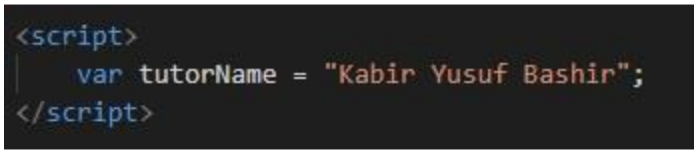
Fig. 2.9 (block comment in JS)

Foot Note:

- *It is most common to use single line comments. Block comments are often used for formal documentation.*

JS Variables

Variables are container for storing data values. In your kitchen, you have a container for salt, sugar or milk; the container of salt is used to store salt and the container of sugar for sugar. This is same as variables in any programming language. In the code below in *fig. 2.10*, the variable **tutorName** stored the value **Kabir Yusuf Bashir**.



```
<script>
    var tutorName = "Kabir Yusuf Bashir";
</script>
```

Fig. 2.10 (JS variables)

Creating a variable in JavaScript is called "declaring" a variable. You declare a JavaScript variable with the var keyword as shown in *fig. 2.10* above.

Variables declared **globally** (outside any function) have **Global Scope. Global** variables can be accessed from anywhere in a JavaScript program.

Variables declared **locally** (inside a function) have **Function Scope**. **Local** variables can only be accessed from inside the function where they are declared.

Variables declared with the var keyword cannot have **Block Scope**. Variables declared inside a block **{ }** can be accessed from outside the block. Before ES2015 JavaScript did not have **Block Scope**.

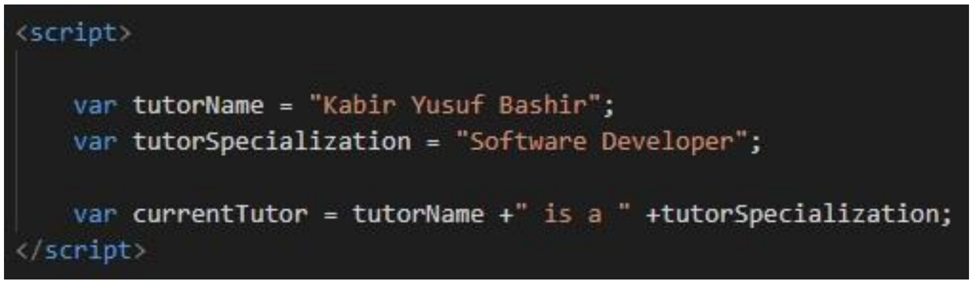
Variables declared with the let keyword can have Block Scope. Variables declared inside a block **{ }** cannot be accessed from outside the block.

Variables defined with const behave like let variables, except they cannot be reassigned. Declaring a variable with const is similar to let when it comes to **Block Scope**.

After the declaration, the variable has no value (technically it has the value of undefined). To **assign** a value to the variable, use the equal sign operator as shown in *fig. 2.10* above.

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator in algebra or mathematics. If you want to use the "equal to" operator in JS, use double equal to operator (= =) or the triple equal to operator (= = =). We will learn more about operators later in this book smile!

You can join two or more strings together under a single variable. The terminology is called **concatenation.** In JS, the plus sign (+) is used to join two or more strings together.



```
<script>

    var tutorName = "Kabir Yusuf Bashir";
    var tutorSpecialization = "Software Developer";

    var currentTutor = tutorName +" is a " +tutorSpecialization;
</script>
```

Fig. 2.11 (JS Concatenation)

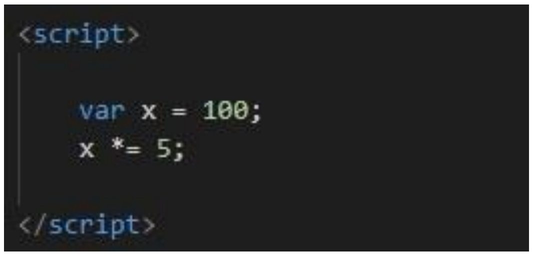
JS Operators

Operators are used to perform some action/operation in any programming language. We will learn about these operators in JS below in this book. We learn some in this section and others fully later in this book. Smile!

- Arithmetic operator
- Assignment operator
- String operator
- Comparison operator
- Logical operator
- Type operator

Arithmetic operators are used to perform arithmetic on numbers. These operators includes: + addition, - subtraction, * multiplication, ** exponentiation, / division, % modulus, ++ increment, -- decrement.

Assignment operators assign values to JavaScript variables. These operators includes: =, +=, -=, *=, /=, %=, **=. In fig. 2.12 below, x is assigned the value of 100. Then we used the *= operator to multiply x by 5 which makes the value of x to be 500.



```
<script>

    var x = 100;
    x *= 5;

</script>
```

Fig. 2.12 (Assignment operator in JS)

String operator is used to concatenate (join) strings together. The + operator is used to add (concatenate) strings. We have learned about the String operator in *fig. 2.11* above.

Note: When used on strings, the + operator is called the concatenation operator.

Comparison operators are used for comparison, you can compare between strings or between numbers. These operators includes: == equal to, === equal value and equal type, != not equal, !== not equal value or not equal type, > greater than, < less than, >= greater than or equal to, <= less than or equal to and ? ternary. We will deep down into comparison operator fully later in this book. Smile!

Logical operators are used to determine the logic between variables or values. For instance, you want to build a login system. You can use the && (AND) logical operator to compare the username and password of the user, if it matches with your credentials in your database, the system logs them in; else, it will display the necessary error! These operators includes: && logical AND, || logical OR and ! logical NOT.

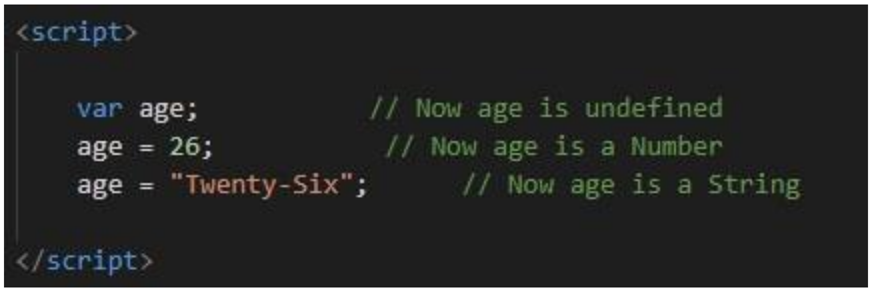
Typeof operator is used to find the data type of a JavaScript variable. These operators includes: typeof (Returns the type of a variable) and instanceof (Returns true if an object is an instance of an object type).

JS Data Types

In programming, data types is an important concept. To be able to operate on variables, it is important to know something about the type. The data types in JS includes:

- Dynamic data type.
- String data type.
- Number data type.
- Boolean data type.
- Array data type.
- Object data type.
- Undefined data type.
- Null data type.

JavaScript **dynamic data type**: This means that the same variable can be used to hold different data types.

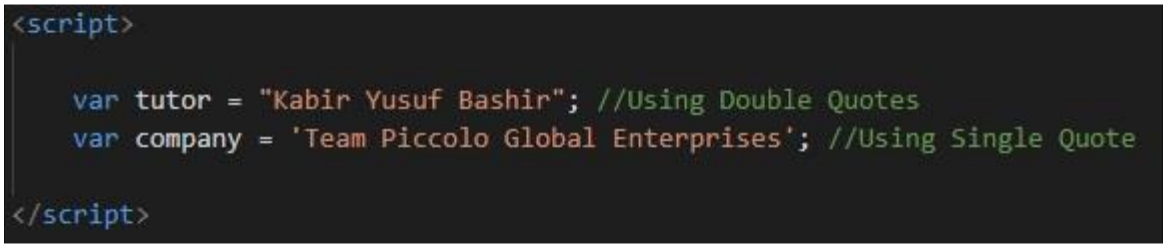


```
<script>

    var age;            // Now age is undefined
    age = 26;            // Now age is a Number
    age = "Twenty-Six";      // Now age is a String

</script>
```

Fig. 2.13 (JS Dynamic Data Type)

Note: the value of **age** is now "Twenty-Six" not 26 because "Twenty-Six" overwrite the value of 26.

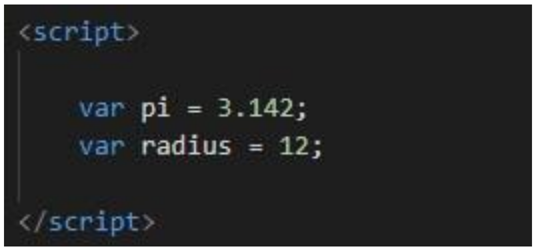
JavaScript **String data type** (or a text string) is a series of characters like "Hon Adamu Alhaji Lawan". Strings are written with quotes. You can use single or double quotes.



```
<script>

    var tutor = "Kabir Yusuf Bashir"; //Using Double Quotes
    var company = 'Team Piccolo Global Enterprises'; //Using Single Quote

</script>
```

Fig. 2.14 (JS String data type)

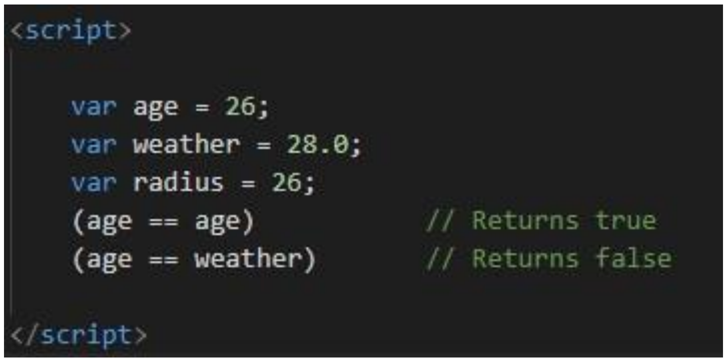
JavaScript **Number data type** is a series of numbers like 2020 or a decimal number like 3.142, etc. Numbers can be written with, or without decimals.



```
<script>

    var pi = 3.142;
    var radius = 12;

</script>
```

Fig. 2.15 (JS Number data type)

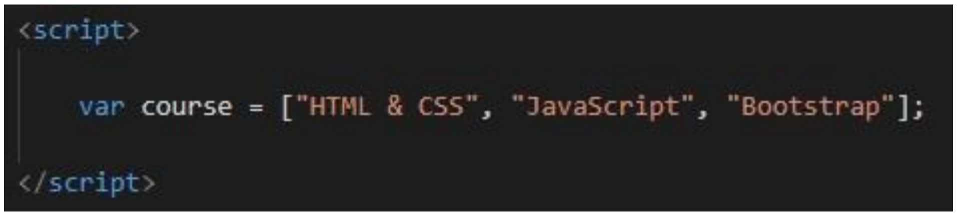
JavaScript **Boolean data type** can only have two values: true or false. Booleans are often used in conditional testing.



```
<script>

    var age = 26;
    var weather = 28.0;
    var radius = 26;
    (age == age)            // Returns true
    (age == weather)        // Returns false

</script>
```

Fig. 2.16 (JS Boolean data type)

JavaScript **array data type**: are written with square brackets. An array stores multiple values in one single variable. Array items are separated by commas. The following code below in *fig. 2.17* declares (creates) an array called course, containing three items (course names).

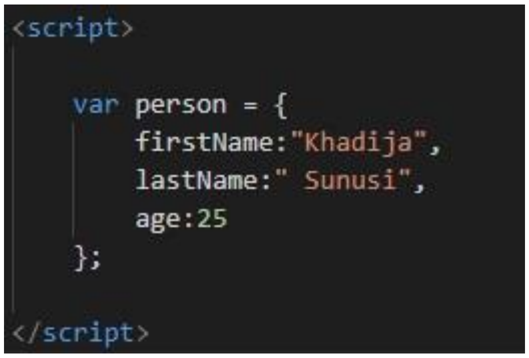


```
<script>

    var course = ["HTML & CSS", "JavaScript", "Bootstrap"];

</script>
```

Fig. 2.17 (JS Array data type)

Note: Array indexes are zero-based, which means the first item is [0], second is [1], and so on

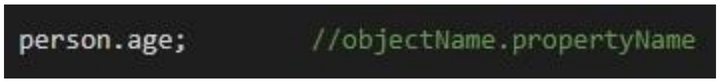
JavaScript **Object data type**: are variables too. But objects can contain many values. An object is a container which encloses data and behavior together. In real life, a person is an **object**. A person has **properties** like firstName, lastName and age, and **methods** like eat, sleep or jog.



```
<script>

    var person = {
        firstName:"Khadija",
        lastName:" Sunusi",
        age:25
    };

</script>
```

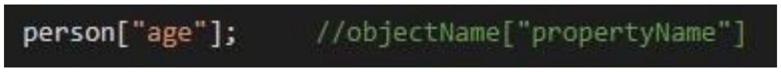
Fig. 2.18 (JS object data type)

Note: You can access object properties in two ways:



```
person.age;         //objectName.propertyName
```

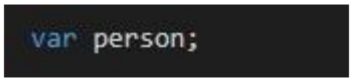
Fig. 2.19 (access object properties I)



```
person["age"];     //objectName["propertyName"]
```

Fig. 2.20 (access object properties II)

JavaScript **Undefined data type**: In JavaScript, a variable without a value, has the value undefined. The type is also undefined. Any variable can be emptied, by setting the value to undefined.



```
var person;
```

Fig. 2.21 (JS undefined data type)

JavaScript **Null data type**: In JavaScript null is "nothing". It is supposed to be something that doesn't exist. Unfortunately, in JavaScript, the data type of null is an object.

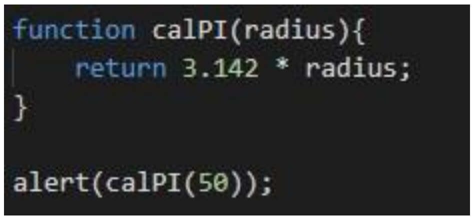
Foot Note:

- *Undefined and Null are equal in value but different in type.*
- *You should use **objects** when you want the element names to be **strings (text)**.*
- *You should use **arrays** when you want the element names to be **numbers**.*

JS Functions

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**. **Function names** can contain letters, digits, underscores, and dollar signs (same rules as variables). The parentheses may include parameter names separated by commas: ***(parameter1, parameter2 ...)***. The code to be executed, by the function, is placed inside curly brackets: **{ }**



```
function calPI(radius){
    return 3.142 * radius;
}

alert(calPI(50));
```

Fig. 2.22 (JS function)

When JavaScript reaches a return statement, the function will stop executing. If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller".

Foot Note:

- *You can reuse code: Define the code once, and use it many times. You can use the same code many times with different arguments, to produce different results.*
- *Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.*
- *Variables declared within a JavaScript function, become **LOCAL** to the function. Local variables can only be accessed from within the function.*

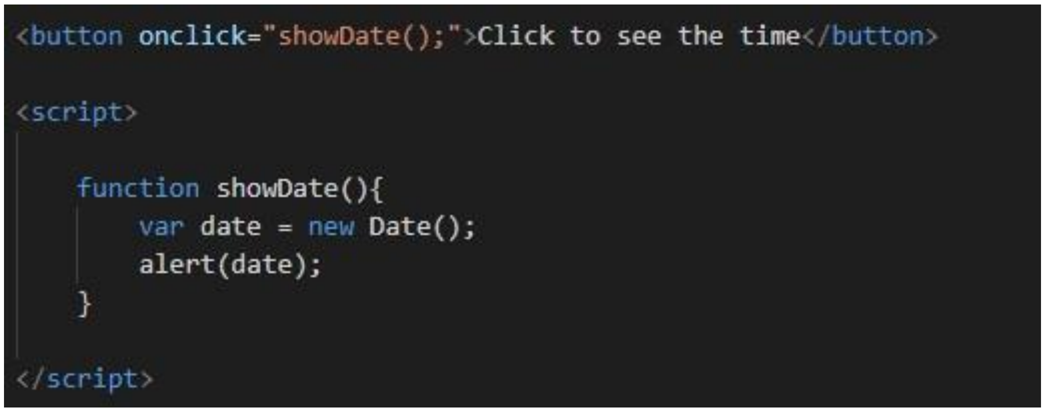
JS Events

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- etc.

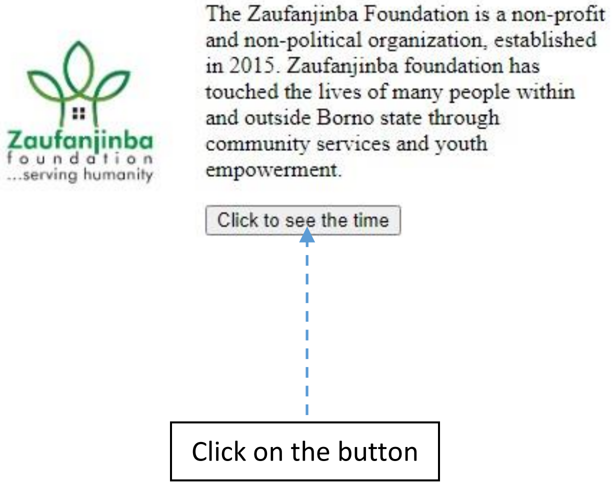
JavaScript lets you execute code when events are detected. Below are the examples of some common HTML events.

Events	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
Onkeydown	The user pushes a keyboard key
Onload	The browser has finished loading the page

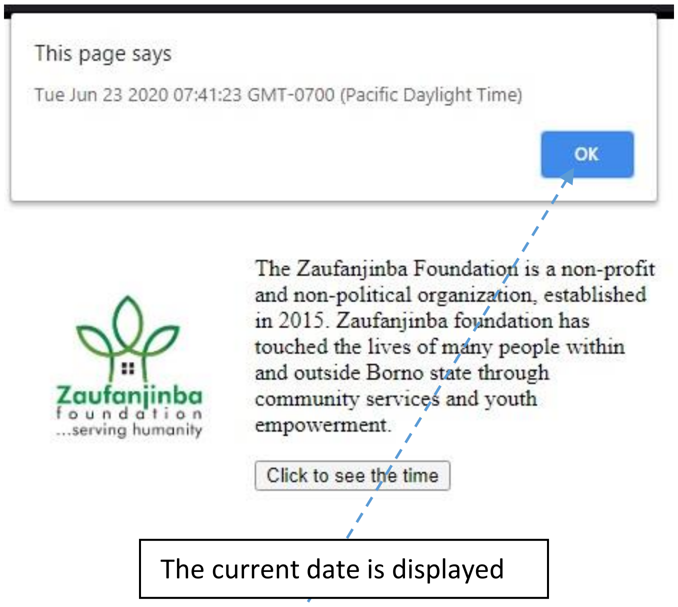


```
<button onclick="showDate();">Click to see the time</button>

<script>

    function showDate(){
        var date = new Date();
        alert(date);
    }

</script>
```

Fig. 2.23 (JS onclick event code)



The Zaufanjinba Foundation is a non-profit and non-political organization, established in 2015. Zaufanjinba foundation has touched the lives of many people within and outside Borno state through community services and youth empowerment.



This page says

Tue Jun 23 2020 07:41:23 GMT-0700 (Pacific Daylight Time)

The Zaufanjinba Foundation is a non-profit and non-political organization, established in 2015. Zaufanjinba foundation has touched the lives of many people within and outside Borno state through community services and youth empowerment.

Fig. 2.24 (when the button is clicked) *Fig. 2.25 (an alert box appears with the current date)*

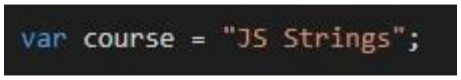
Foot Note:

- *HTML events are "**things**" that happen to HTML elements.*
- *When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.*
- *We will learn more about events in HTML DOM section later in this book. Smile!*

JS Strings

We have started learning about strings in JS when we learnt about JS data types. In this section, we will learn more about strings in JS.

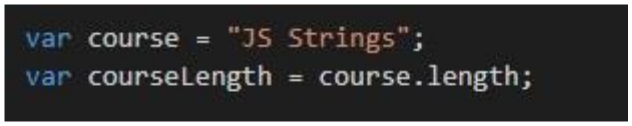
JavaScript strings are used for storing and manipulating text. A JavaScript string is zero or more characters written inside quotes. It can be placed inside a single or double quotes.



```
var course = "JS Strings";
```

Fig. 2.26 (JS Strings)

To find the length of a string, use the built-in **length** property as shown below in *fig. 2.27*:

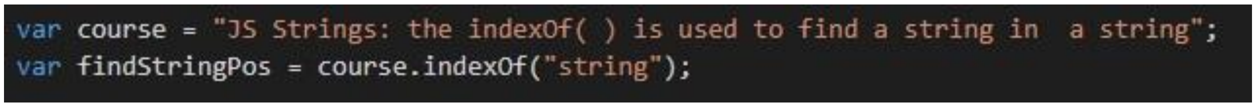


```
var course = "JS Strings";
var courseLength = course.length;
```

Fig. 2.27 (JS String length property)

Note: the length property in JS can be used when you want to limit the character of variable. Maybe you want the password input field to have minimum characters of 3; the length property is your best solution!

The **indexOf ( )** method is used to find a string in a string. The indexOf ( ) method returns the index of (the position of) the **first** occurrence of a specified text in a string.



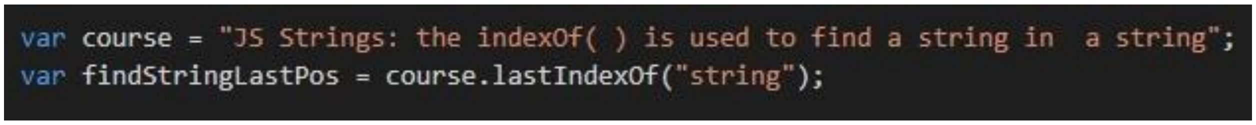
```
var course = "JS Strings: the indexOf( ) is used to find a string in  a string";
var findStringPos = course.indexOf("string");
```

Fig. 2.28 (JS String indexOf ( ) property)

Foot Note:

- *JavaScript counts positions from zero. 0 is the first position in a string, 1 is the second, and 2 is the third and so on…*

The **lastIndexOf ( )** method returns the index of the **last** occurrence of a specified text in a string.

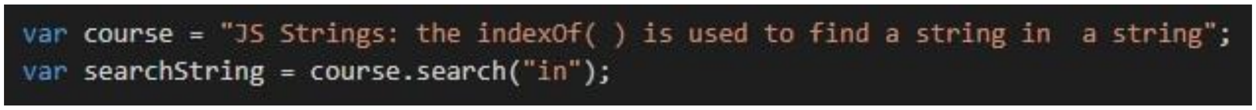


```
var course = "JS Strings: the indexOf( ) is used to find a string in  a string";
var findStringLastPos = course.lastIndexOf("string");
```

Fig 2.29 (JS String lastIndexOf ( ) property)

Note: Both the indexOf ( ), and the lastIndexOf ( ) methods return -1 if the text is **not found.**

The **search ( )** method is used to search for a string in a string. The search ( ) method searches a string for a specified value and returns the position of the match.



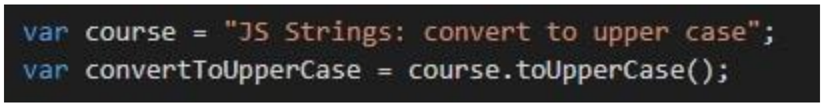
```
var course = "JS Strings: the indexOf( ) is used to find a string in  a string";
var searchString = course.search("in");
```

Fig. 2.30 (JS String search ( ) method)

Note:

- The two methods, indexOf ( ) and search ( ), are equal.
- They accept the same arguments (parameters), and they return the same value.
- The two methods are equal, but the search ( ) method can take much more powerful search values.

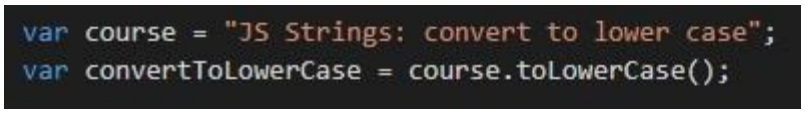
The **toUpperCase ( )** method is used to convert string to upper case characters. A string is converted to upper case with toUpperCase ( ).



```
var course = "JS Strings: convert to upper case";
var convertToUpperCase = course.toUpperCase();
```

Fig. 2.31 (JS String toUpperCase ( ) method)

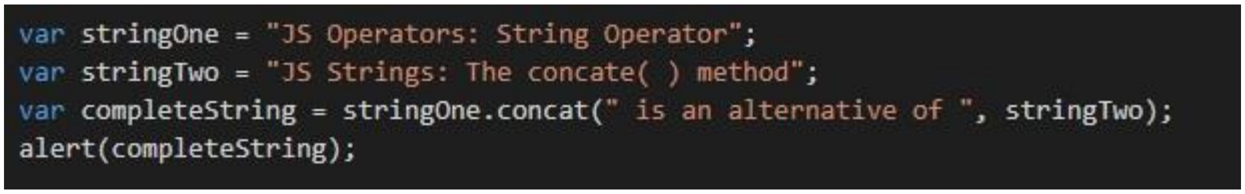
The **toLowerCase ( )** method is used to convert string to lower case characters. A string is converted to lower case with toLowerCase ( ).



```
var course = "JS Strings: convert to lower case";
var convertToLowerCase = course.toLowerCase();
```

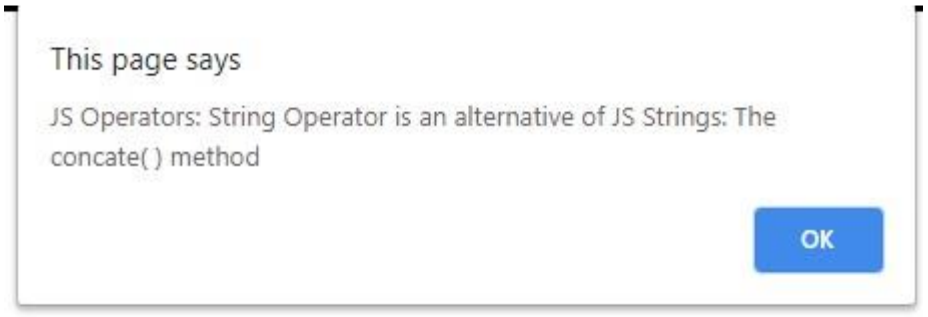
Fig. 2.32 (JS String toLowerCase ( ) method)

The **concat ( )** method is used to join two or more strings. It works same as the plus operator (+) which is used for concatenation. The concat ( ) is an alternative to the sting operator (+). You can read more about the string operator (+) in **JS operators** above (string operator).



```
var stringOne = "JS Operators: String Operator";
var stringTwo = "JS Strings: The concate( ) method";
var completeString = stringOne.concat(" is an alternative of ", stringTwo);
alert(completeString);
```

Fig. 2.33 (JS String concat ( ) method)



This page says

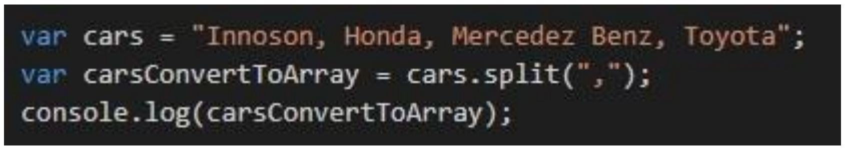
JS Operators: String Operator is an alternative of JS Strings: The concate( ) method

Fig. 2.34

*An alert box showing **stringOne** and **stringTwo** joined together using the concat ( ) method.*

The Zaufanjinba Foundation is a non-profit and non-political organization, established in 2015. Zaufanjinba foundation has touched the lives of many people within and outside Borno state through community services and youth empowerment.

The **split ( )** method is used to convert a string to an array.



```
var cars = "Innoson, Honda, Mercedez Benz, Toyota";
var carsConvertToArray = cars.split(",");
console.log(carsConvertToArray);
```

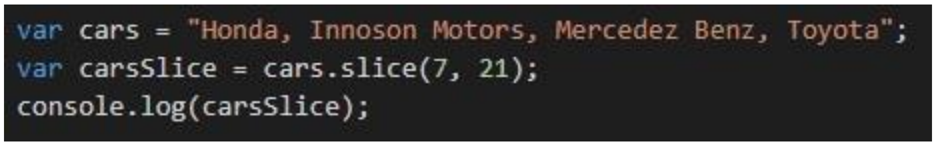
Fig. 2.35 (JS String split ( ) method)

Note:

- If the separator is omitted, the returned array will contain the whole string in index [0].
- If the separator is "", the returned array will be an array of single characters.
- We will learn more about the console.log ( ) method in **JS Debugging**.

There are 3 methods for **extracting a part of a string**: slice (start, end), substring (start, end), substr (start, length).

The **slice ( )** method extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the starting index (position), and the ending index (position). The code below in *fig. 2.36* contains a variable name **cars** with some list of cars stored as the value of the variable. We used the slice ( ) method to extract "**Innoson Motors**" from the variable cars.



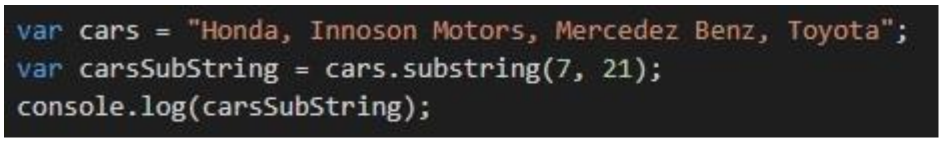
```
var cars = "Honda, Innoson Motors, Mercedez Benz, Toyota";
var carsSlice = cars.slice(7, 21);
console.log(carsSlice);
```

Fig. 2.36 (JS String slice ( ) method)

Note:

- If a parameter is negative, the position is counted from the end of the string.
- To get same result as the code in *fig. 2.36*, change the slice ( ) method position to slice (38, -23).
- If you omit the second parameter, the method will slice out the rest of the string

The **substring ( )** method is similar to slice ( ). The difference is that substring ( ) cannot accept negative indexes.

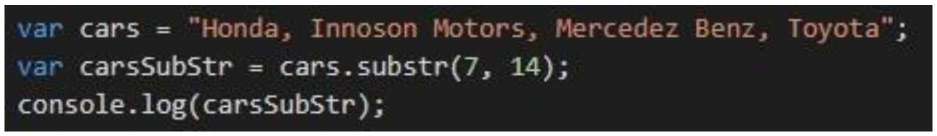


```
var cars = "Honda, Innoson Motors, Mercedez Benz, Toyota";
var carsSubString = cars.substring(7, 21);
console.log(carsSubString);
```

Fig. 2.37 (JS String substring ( ) method)

Note: the only difference between the slice ( ) method and substring ( ) method is that the substring ( ) method cannot accept negative indexed.

The **substr ( )** is similar to slice ( ). The difference is that the second parameter specifies the **length** of the extracted part. The code below in *fig. 2.38* extracted "**Innoson Motors**" from the variable **cars**.



```
var cars = "Honda, Innoson Motors, Mercedez Benz, Toyota";
var carsSubStr = cars.substr(7, 14);
console.log(carsSubStr);
```

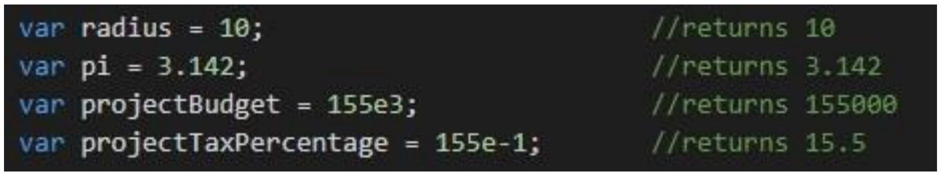
Fig. 2.38 (JS String substr ( ) method)

Foot Note:

- *If the first parameter is negative, the position counts from the end of the string.*
- *The second parameter cannot be negative, because it defines the length.*
- *If you omit the second parameter, substr ( ) will slice out the rest of the string.*

JS Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals. Extra-large or extra small numbers can be written with scientific (exponent) notation. All the variables below in *fig. 2.39* are valid numbers in JS.

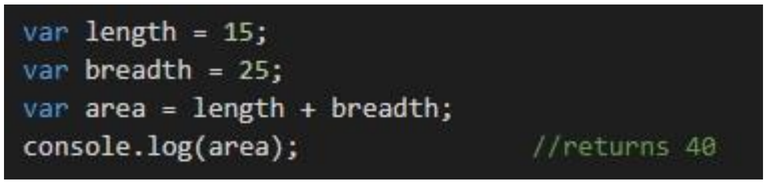


```
var radius = 10;                          //returns 10
var pi = 3.142;                           //returns 3.142
var projectBudget = 155e3;                //returns 155000
var projectTaxPercentage = 155e-1;        //returns 15.5
```

Fig. 2.39 (JS Numbers)

Note:

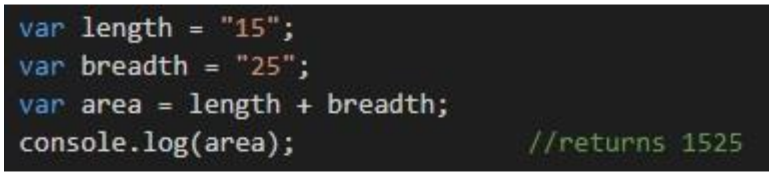
- JavaScript uses the + operator for both addition and concatenation.
- Numbers are added. Strings are concatenated.



```
var length = 15;
var breadth = 25;
var area = length + breadth;
console.log(area);               //returns 40
```

Fig. 2.40 (JS Numbers + operator)

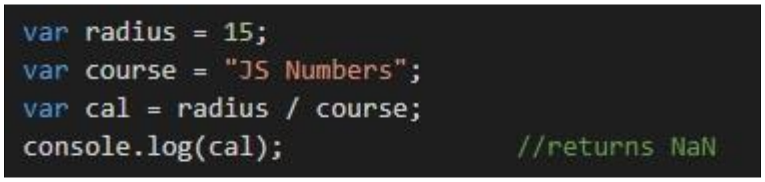
- If you add two strings, the result will be a string concatenation.



```
var length = "15";
var breadth = "25";
var area = length + breadth;
console.log(area);               //returns 1525
```

Fig. 2.41 (JS string concatenation)

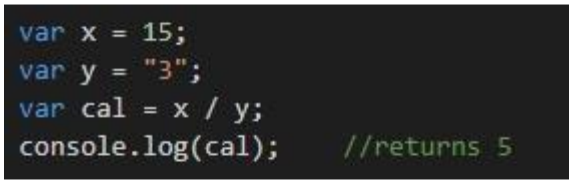
NaN is a JavaScript reserved word indicating that a number is not a legal number. Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number). In the code below in fig. 2.42, we try to divide the variable radius (**15**) by the variable course ("**JS Numbers**"). It returns NaN because you cannot divide 15 by "JS Numbers".



```
var radius = 15;
var course = "JS Numbers";
var cal = radius / course;
console.log(cal);               //returns NaN
```

Fig. 2.42 (JS NaN)

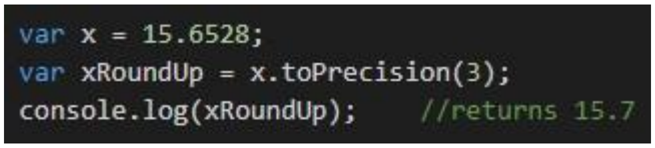
Note: However, if the string contains a numeric value, the result will be a number.



```
var x = 15;
var y = "3";
var cal = x / y;
console.log(cal);    //returns 5
```

Fig. 2.43 (JS Numbers II)

The **toPrecision ( )** method returns a string, with a number written with a specified length. The toPrecision ( ) method is used to round-up a number. In the code below in *fig. 2.44*, the variable **x** has 6 digits but we round-it up to 3 digits using the toPrecision ( ) which returns **15.7**:



```
var x = 15.6528;
var xRoundUp = x.toPrecision(3);
console.log(xRoundUp);    //returns 15.7
```

Fig. 2.44 (JS Number toPrecision ( ) method)

There are 3 JavaScript methods that can be used to convert variables to numbers:

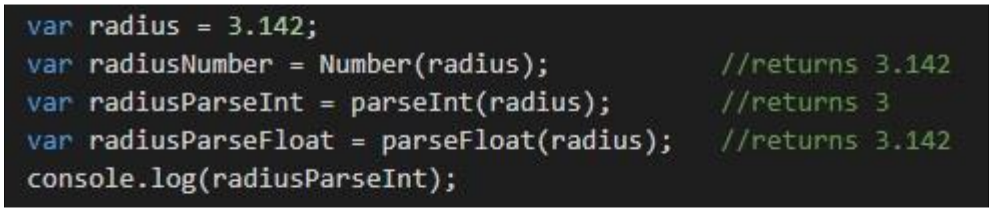
- The Number ( ) method
- The parseInt ( ) method
- The parseFloat ( ) method

The **Number ( ) method** can be used to convert JavaScript variables to numbers.

The **parseInt ( ) method** parses a string and returns a whole number. Spaces are allowed. Only the first number is returned.

The **parseFloat ( )** method parses a string and returns a number. Spaces are allowed. Only the first number is returned.

Note: If the number cannot be converted, NaN (Not a Number) is returned.

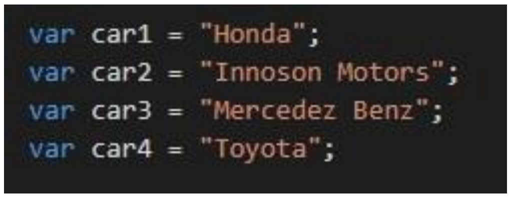


```
var radius = 3.142;
var radiusNumber = Number(radius);          //returns 3.142
var radiusParseInt = parseInt(radius);      //returns 3
var radiusParseFloat = parseFloat(radius);  //returns 3.142
console.log(radiusParseInt);
```

Fig. 2.45 (JS Number ( ), parseInt ( ) & parseFloat ( ) methods)

JS Arrays

An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like *fig. 2.46* below:

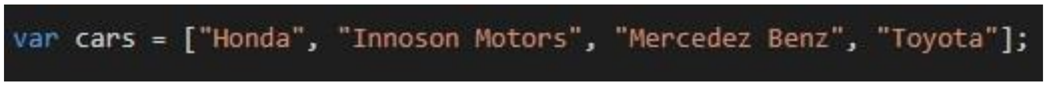


```
var car1 = "Honda";
var car2 = "Innoson Motors";
var car3 = "Mercedez Benz";
var car4 = "Toyota";
```

Fig. 2.46 (JS Arrays)

However, what if you want to loop through the cars and find a specific one? And what if you had not 4 cars, but 40000? **The solution is an array!**

An array can hold many values under a single name, and you can access the values by referring to an index number.



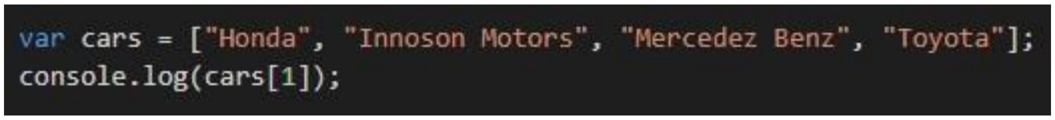
```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota"];
```

Fig. 2.47 (JS Arrays II)

You can access an array element by referring to the **index number**.

Note: Array indexes start with 0. [0] is the first element. [1] is the second element.

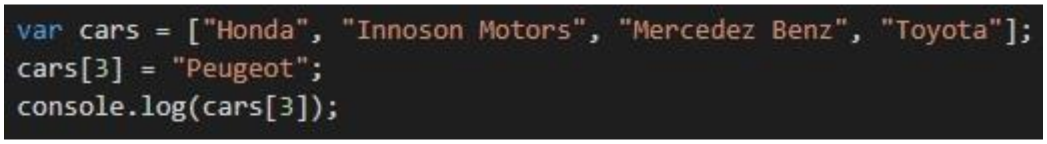
If we want to access "**Innoson Motors**" from the array **cars**, we access it using the array name and the index [1] as shown below in *fig. 2.48*:



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota"];
console.log(cars[1]);
```

Fig. 2.48 (JS array: access array element)

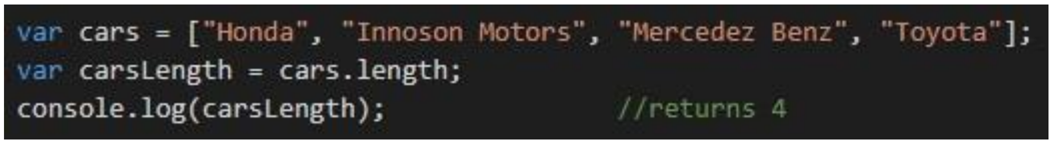
You can change the value of the fourth element in the cars array above in *fig. 2.48* by accessing the element index and assigning a new value to it as shown below in *fig. 2.49*:



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota"];
cars[3] = "Peugeot";
console.log(cars[3]);
```

Fig. 2.49 (JS array: changing the value of an array element)

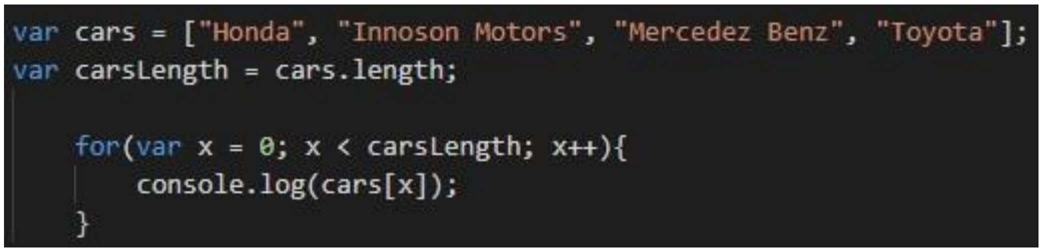
You can get the total elements contained in an array by using the **length property**.
The length property of an array returns the length of an array (the number of array elements).
Let's try to get the length of the array cars above:



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota"];
var carsLength = cars.length;
console.log(carsLength);                //returns 4
```

Fig. 2.50 (JS array: length property)

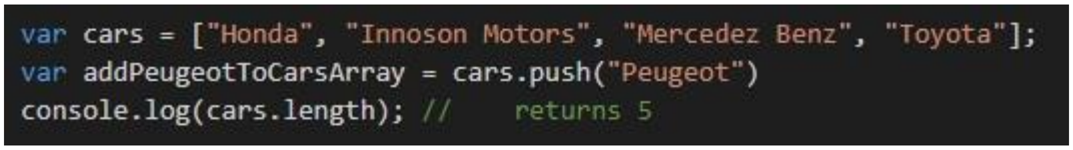
You can loop through the cars array above using loop. The safest way to loop through an array is to use for loop. We will learn more about loops in JS later in this book. Smile!



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota"];
var carsLength = cars.length;

    for(var x = 0; x < carsLength; x++){
        console.log(cars[x]);
    }
```

Fig. 2.51 (JS array: looping through an array)

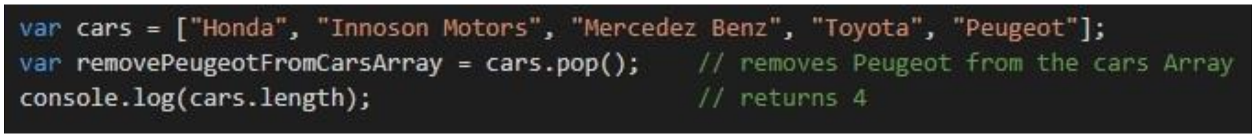
Sometimes a need might come to add a new element to an array; in JS, the **push ( ) method** is used. The push ( ) method adds a new element to an array (at the end).



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota"];
var addPeugeotToCarsArray = cars.push("Peugeot")
console.log(cars.length); //    returns 5
```

Fig. 2.52 (JS Array: push ( ) method)

To remove the new element you inserted into the cars array, you can use the **pop ( ) method**. The pop ( ) method removes the last element from an array.

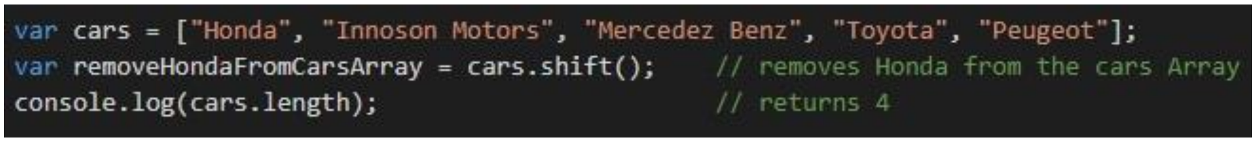


```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota", "Peugeot"];
var removePeugeotFromCarsArray = cars.pop();    // removes Peugeot from the cars Array
console.log(cars.length);                       // returns 4
```

Fig. 2.53 (JS Array: pop ( ) method)

The pop ( ) method removes the last element from an array but in some scenarios, we might want to remove the first element from the array. You can use the shift ( ) method in JS.

The **shift ( ) method** removes the first array element and "shifts" all other elements to a lower index. Shifting is equivalent to popping, working on the first element instead of the last.

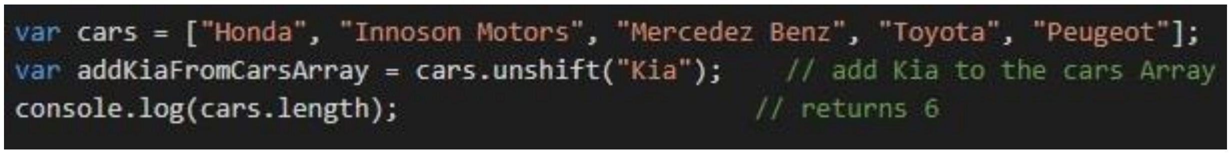


```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota", "Peugeot"];
var removeHondaFromCarsArray = cars.shift();    // removes Honda from the cars Array
console.log(cars.length);                       // returns 4
```

Fig. 2.54 (JS Array: shift ( ) method)

To add an element to the beginning of an array, you can use **unshift ( ) method** in JS.

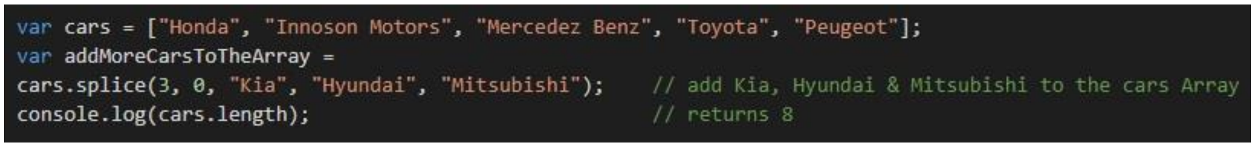
The unshift ( ) method adds a new element to an array (at the beginning)



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota", "Peugeot"];
var addKiaFromCarsArray = cars.unshift("Kia");    // add Kia to the cars Array
console.log(cars.length);                        // returns 6
```

Fig. 2.55 (JS Array: unshift ( ) method)

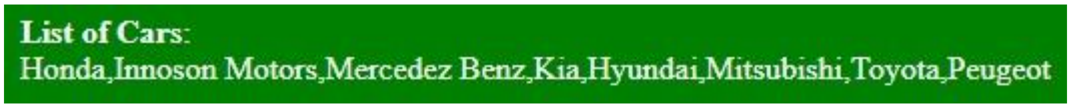
You can also add new elements to an array using the **splice ( ) method**. The splice ( ) method can be used to add new items to an array. This method contain some parameters as shown below in *fig. 2.56*:



```
var cars = ["Honda", "Innoson Motors", "Mercedez Benz", "Toyota", "Peugeot"];
var addMoreCarsToTheArray =
cars.splice(3, 0, "Kia", "Hyundai", "Mitsubishi");    // add Kia, Hyundai & Mitsubishi to the cars Array
console.log(cars.length);                             // returns 8
```

Fig. 2.56 (JS Array: splice ( ) method)

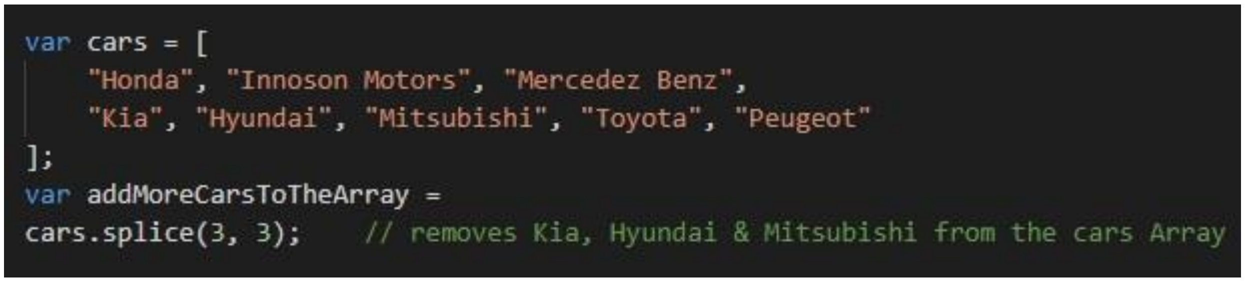
The first parameter (2) defines the position **where** new elements should be **added** (spliced in). The second parameter (0) defines **how many** elements should be **removed**. The rest of the parameters (“Kia”, “Hyundai”, “Mitsubishi”) define the new elements to be **added**.



```
List of Cars:
Honda,Innoson Motors,Mercedez Benz,Kia,Hyundai,Mitsubishi,Toyota,Peugeot
```

Fig. 2.57 (JS Array: cars added using the splice ( ) method)

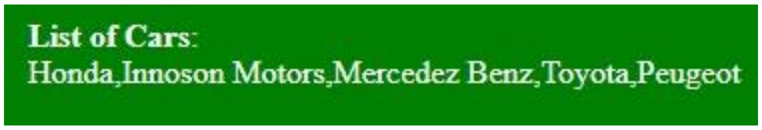
You can also use the splice ( ) method to remove an element from an array. With clever parameter setting, you can use splice ( ) to remove elements without leaving "holes" in the array. Let's try to remove an element from the cars array using the splice ( ) method as shown below in *fig. 2.58*:



```
var cars = [
    "Honda", "Innoson Motors", "Mercedez Benz",
    "Kia", "Hyundai", "Mitsubishi", "Toyota", "Peugeot"
];
var addMoreCarsToTheArray =
cars.splice(3, 3);    // removes Kia, Hyundai & Mitsubishi from the cars Array
```

Fig. 2.58 (JS Array: using the splice ( ) method to remove an element in an array)

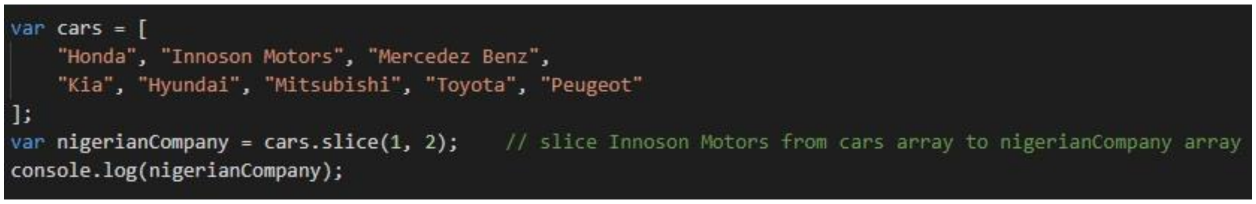
The first parameter (3) defines the position where new elements should be **added** (spliced in). The second parameter (3) defines **how many** elements should be **removed**. The rest of the parameters are omitted. No new elements will be added.



```
List of Cars:
Honda,Innoson Motors,Mercedez Benz,Toyota,Peugeot
```

Fig. 2.59 (JS Array: cars array after removing some elements using splice ( ) method)

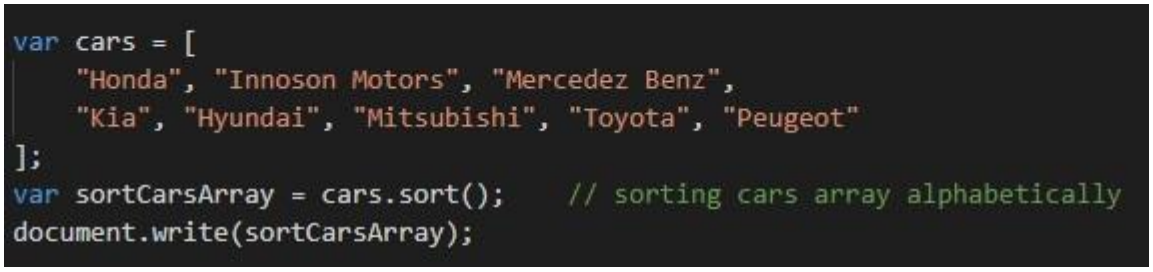
You can use **slice ( ) method** to slices out a piece of an array into a new array. The slice ( ) method can take two arguments like slice (1, 2). The method then selects elements from the start argument, and up to (but not including) the end argument.



```
var cars = [
    "Honda", "Innoson Motors", "Mercedez Benz",
    "Kia", "Hyundai", "Mitsubishi", "Toyota", "Peugeot"
];
var nigerianCompany = cars.slice(1, 2);    // slice Innoson Motors from cars array to nigerianCompany array
console.log(nigerianCompany);
```

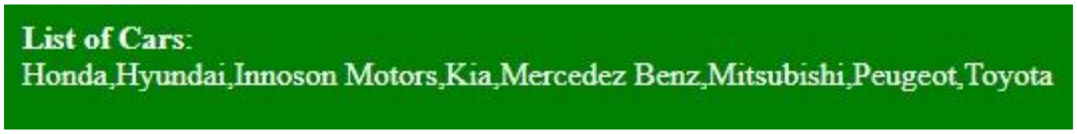
Fig. 2.60 (JS Array: slicing Innoson Motors from the cars array)

You can sort an array in alphabetical order using the **sort ( ) method**. The sort ( ) method sorts an array alphabetically.



```
var cars = [
    "Honda", "Innoson Motors", "Mercedez Benz",
    "Kia", "Hyundai", "Mitsubishi", "Toyota", "Peugeot"
];
var sortCarsArray = cars.sort();    // sorting cars array alphabetically
document.write(sortCarsArray);
```

Fig. 2.61 (JS Arrays: sort ( ) method)

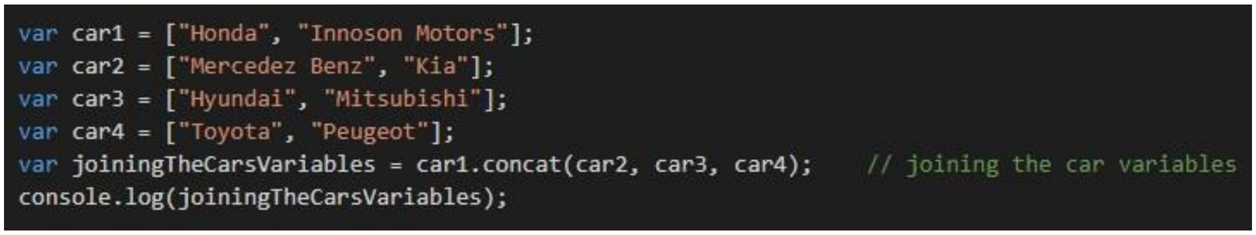


```
List of Cars:
Honda,Hyundai,Innoson Motors,Kia,Mercedez Benz,Mitsubishi,Peugeot,Toyota
```

Fig. 2.62 (JS Arrays: the cars array sorted in alphabetical order)

Note: use the sort ( ) method only on strings but not numbers, as this will produce incorrect result.

The **concat ( ) method** creates a new array by concatenating two arrays.



```
var car1 = ["Honda", "Innoson Motors"];
var car2 = ["Mercedez Benz", "Kia"];
var car3 = ["Hyundai", "Mitsubishi"];
var car4 = ["Toyota", "Peugeot"];
var joiningTheCarsVariables = car1.concat(car2, car3, car4);    // joining the car variables
console.log(joiningTheCarsVariables);
```

Fig. 2.63 (JS Arrays: concat ( ) method)

Summary:

Push ( ) method adds a new element to an array (at the end) whereas the Pop ( ) method removes an element from an array (at the end).

Unshift ( ) method adds a new element to an array (at the beginning) whereas the shift ( ) method removes an element from an array (at the beginning).

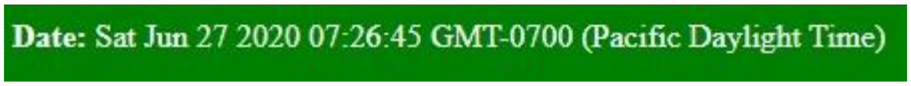
Splice ( ) method can both be used to add and remove an element from array depending on the parameters you passed.

Slice ( ) method is used to slice a piece of an array to a new array.

You can use the concat ( ) method to join two arrays together.

JS Dates

JavaScript **Date Object** lets us work with dates. By default, JavaScript will use the browser's time zone and display a date as a full text string as shown below in *fig. 2.64*:

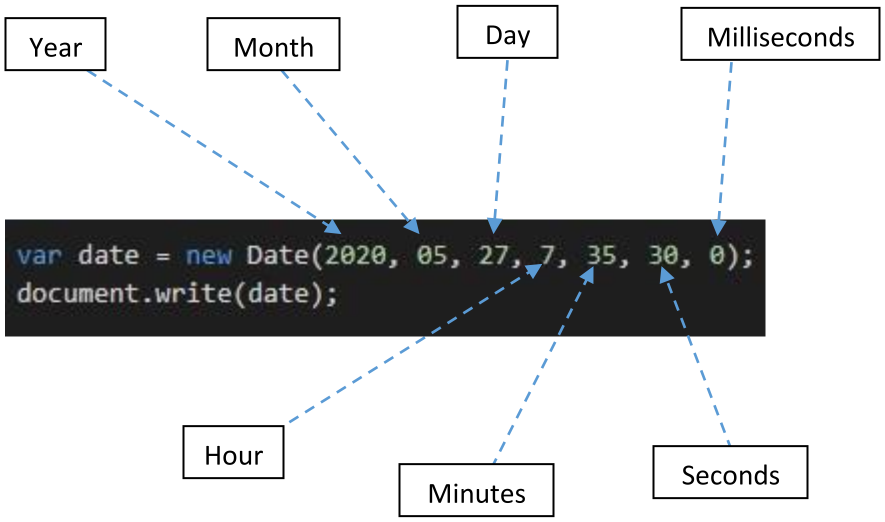


```
Date: Sat Jun 27 2020 07:26:45 GMT-0700 (Pacific Daylight Time)
```

Fig. 2.64 (JS Date)

Date objects are created with the new Date ( ) constructor. There are **4 ways** to create a new date object:

- new Date ( )
- new Date (*year, month, day, hours, minutes, seconds, milliseconds*)
- new Date (*milliseconds*)
- new Date (*date string*)

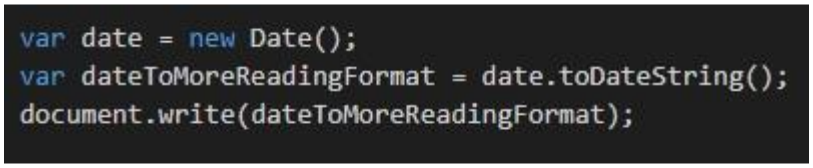


```
var date = new Date(2020, 05, 27, 7, 35, 30, 0);
document.write(date);
```

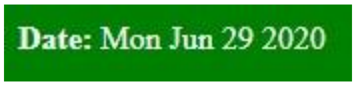
Fig. 2.65 (JS Date II)

Note: JavaScript counts months from 0 to 11. January is 0. December is 11. You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

The **toDateString ( ) method** converts a date to a more readable format as shown below in *fig. 2.66*:



```
var date = new Date();
var dateToMoreReadingFormat = date.toDateString();
document.write(dateToMoreReadingFormat);
```



```
Date: Mon Jun 29 2020
```

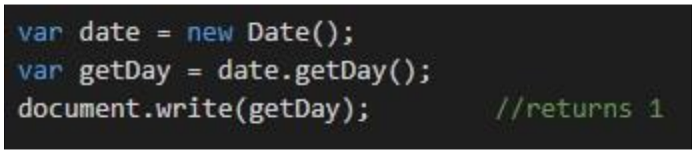
Fig. 2.66 (JS Date: toDateString ( ) method)

Get methods are used for getting a part of a date. Here are the most common (alphabetically).

Method	**Description**
getDate ( )	Get the day as a number (1-31)
getDay ( )	Get the weekday as a number (0-6)
getFullYear ( )	Get the four digit year (yyyy)
getHours ( )	Get the hour (0-23)
getMilliseconds ( )	Get the milliseconds (0-999)
getMinutes ( )	Get the minutes (0-59)
getMonth ( )	Get the month (0-11)
getSeconds ( )	Get the seconds (0-59)

getTime ( )	Get the time (milliseconds since January 1, 1970)

Date methods let you get and set date values (years, months, days, hours, minutes, seconds, milliseconds).

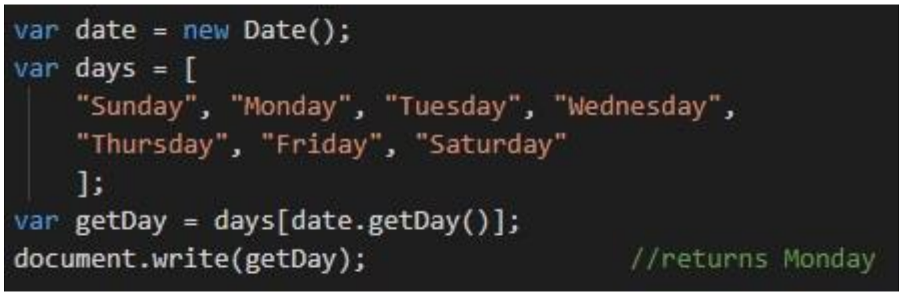


```
var date = new Date();
var getDay = date.getDay();
document.write(getDay);          //returns 1
```

Fig. 2.67 (JS Date: getDay ( ) method)

Note: In JavaScript, the first day of the week (0) means "Sunday", even if some countries in the world consider the first day of the week to be "Monday".

You can use an array of names, and getDay ( ) to return the weekday as a name as shown below in *fig. 2.68*:

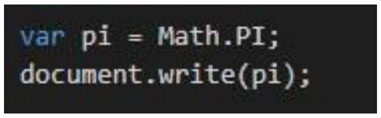


```
var date = new Date();
var days = [
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"
    ];
var getDay = days[date.getDay()];
document.write(getDay);                //returns Monday
```

Fig. 2.68 (JS Date: getting the day of the week)

JS Maths

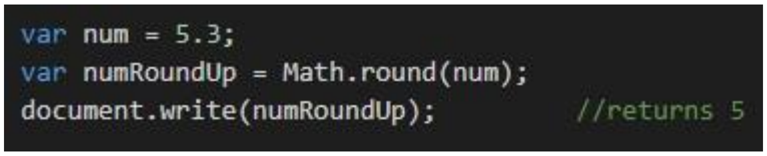
The JavaScript Math object allows you to perform mathematical tasks on numbers. The **Math.PI ( )** is used to return the pie value as shown below in *fig. 2.69*:



```
var pi = Math.PI;
document.write(pi);
```

Fig. 2.69 (JS Maths: Math.PI ( ) method)

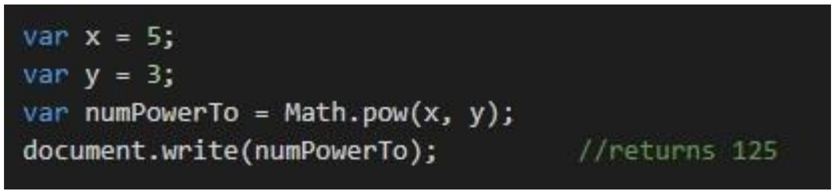
Math.round (x) returns the value of x rounded to its nearest integer.



```
var num = 5.3;
var numRoundUp = Math.round(num);
document.write(numRoundUp);         //returns 5
```

Fig. 2.70 (JS Maths: Math.round ( ) method)

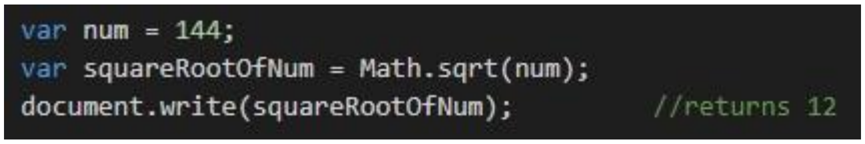
Math.pow (x, y) returns the value of x to the power of y:



```
var x = 5;
var y = 3;
var numPowerTo = Math.pow(x, y);
document.write(numPowerTo);         //returns 125
```

Fig. 2.71 (JS Maths: Math.pow ( ) method)

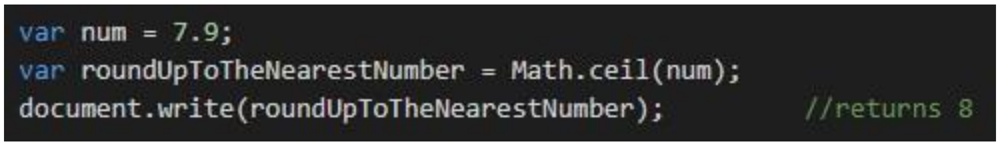
Math.sqrt (x) returns the square root of x:



```
var num = 144;
var squareRootOfNum = Math.sqrt(num);
document.write(squareRootOfNum);         //returns 12
```

Fig. 2.72 (JS Maths: Math.sqrt ( ) method)

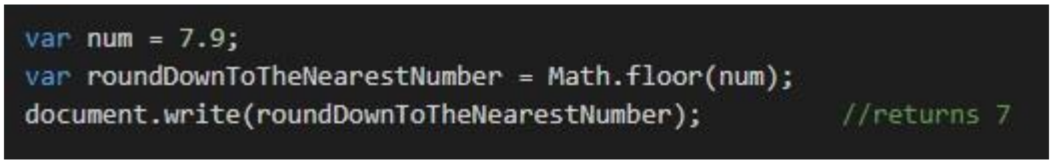
Math.ceil (x) returns the value of x rounded **up** to its nearest integer:



```
var num = 7.9;
var roundUpToTheNearestNumber = Math.ceil(num);
document.write(roundUpToTheNearestNumber);          //returns 8
```

Fig. 2.73 (JS Maths: Math.ceil ( ) method)

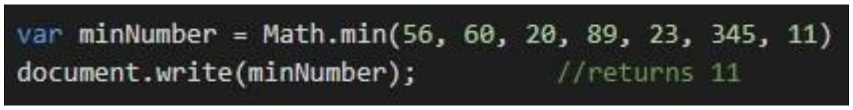
Math.floor (x) returns the value of x rounded **down** to its nearest integer:



```
var num = 7.9;
var roundDownToTheNearestNumber = Math.floor(num);
document.write(roundDownToTheNearestNumber);          //returns 7
```

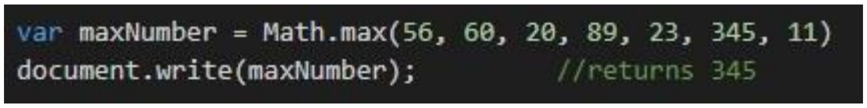
Fig. 2.74 (JS Maths: Math.floor ( ) method)

Math.min ( ) and **Math.max ( )** can be used to find the lowest or highest value in a list of arguments:



```
var minNumber = Math.min(56, 60, 20, 89, 23, 345, 11)
document.write(minNumber);          //returns 11
```

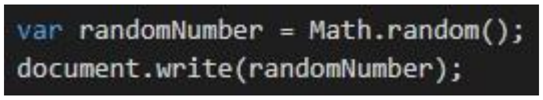
Fig. 2.75 (JS Maths: Math.min ( ) method)



```
var maxNumber = Math.max(56, 60, 20, 89, 23, 345, 11)
document.write(maxNumber);          //returns 345
```

Fig. 2.76 (JS Maths: Math.max ( ) method)

Math.random ( ) returns a random number between 0 (inclusive), and 1 (exclusive).



```
var randomNumber = Math.random();
document.write(randomNumber);
```

Fig. 2.77 (JS Maths: Math.random ( ) method)

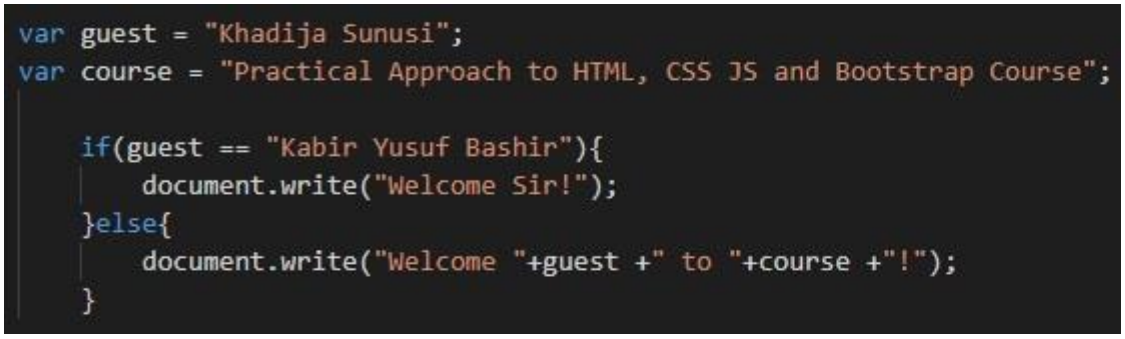
Note: Math.random ( ) always returns a number lower than 1.

JS Comparison and Logical Operator

Comparison operators are used in logical statements to determine equality or difference between variables or values. Comparison and Logical operators are used to test for true or false.

Comparison operators includes: == equal to, === equal value and equal type, != not equal to, !== not equal value and not equal type, > greater than, < less than, >= greater than or equal to and <= less than or equal to.

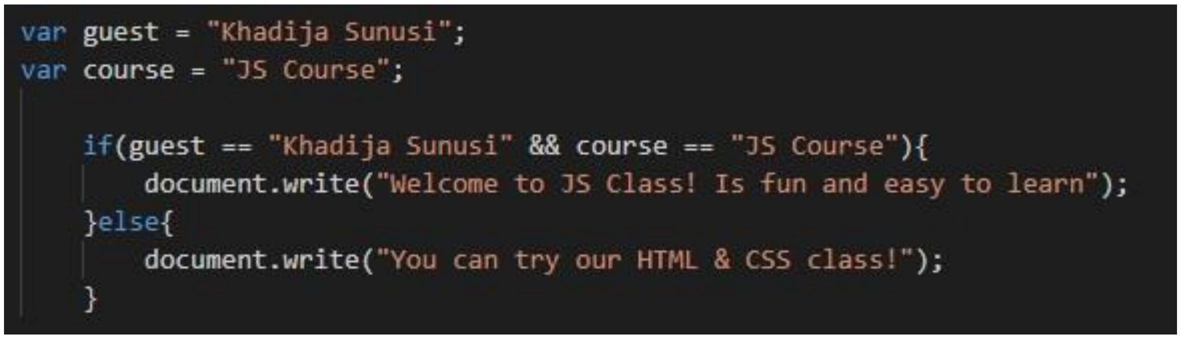
The code below in *fig. 2.78* compare a variable **guest**, if the **guest** is “**Kabir Yusuf Bashir**”, it will print “**Welcome Sir!**”, else, it will print “**Welcome guest to course**”.



```
var guest = "Khadija Sunusi";
var course = "Practical Approach to HTML, CSS JS and Bootstrap Course";

    if(guest == "Kabir Yusuf Bashir"){
        document.write("Welcome Sir!");
    }else{
        document.write("Welcome "+guest +" to "+course +"!");
    }
```

Fig. 2.78 (JS Comparison operator: == equal to)

Logical operators are used to determine the logic between variables or values. Logical operators includes: && AND, || OR and ! NOT.

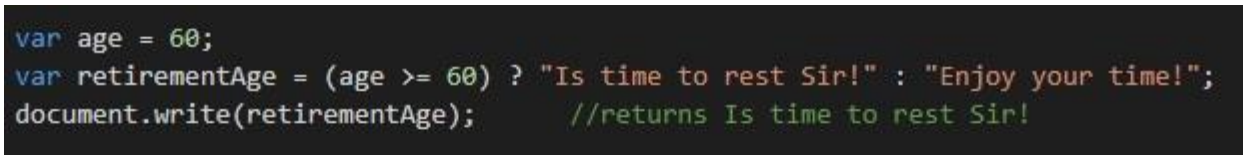


```
var guest = "Khadija Sunusi";
var course = "JS Course";

    if(guest == "Khadija Sunusi" && course == "JS Course"){
        document.write("Welcome to JS Class! Is fun and easy to learn");
    }else{
        document.write("You can try our HTML & CSS class!");
    }
```

Fig. 2.79 (JS Logical operator: && AND)

Don’t worry if you find it difficult understanding the code in ***fig. 2.78*** and ***fig. 2.79***, it will become second nature as you continue reading this book. Smile!

JavaScript also contains a **conditional (Ternary) operator** that assigns a value to a variable based on some condition.



```
var age = 60;
var retirementAge = (age >= 60) ? "Is time to rest Sir!" : "Enjoy your time!";
document.write(retirementAge);      //returns Is time to rest Sir!
```

Fig. 2.80 (JS conditional operator)

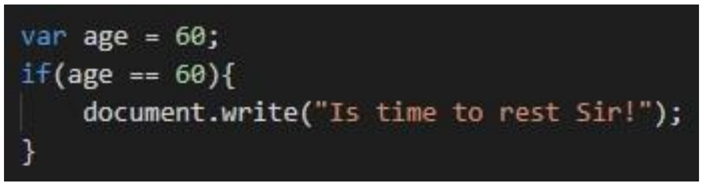
Note: If the variable **age** is a value greater than or equal to **60**, the value of the variable **retirementAge** will be "**Is time to rest Sir!**", otherwise the value of **retirementAge** will be "**Enjoy your time!**".

JS Conditions

Conditional statements are used to perform different actions based on different conditions. Very often when you write code, you want to perform different actions for different decisions. In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The **if statement** specify a block of JavaScript code to be executed if a condition is true.

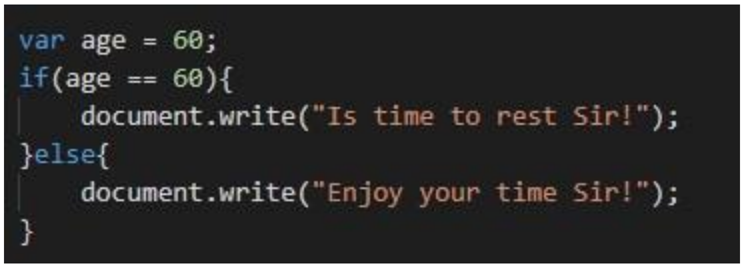


```
var age = 60;
if(age == 60){
    document.write("Is time to rest Sir!");
}
```

Fig. 2.81 (JS Conditions: if statement)

Note: that **if** is in lowercase letters. Uppercase letters (**If** or **IF**) will generate a JavaScript error.

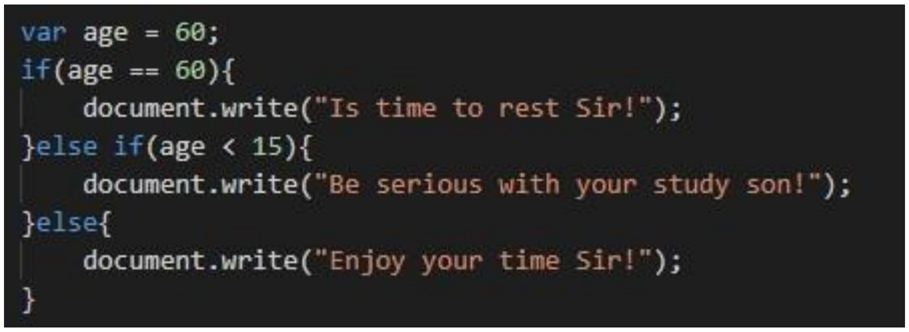
The **else statement** specify a block of code to be executed if the condition is false.



```
var age = 60;
if(age == 60){
    document.write("Is time to rest Sir!");
}else{
    document.write("Enjoy your time Sir!");
}
```

Fig. 2.82 (JS Conditions: else statement)

Use the **else if statement** to specify a new condition if the first condition is false.



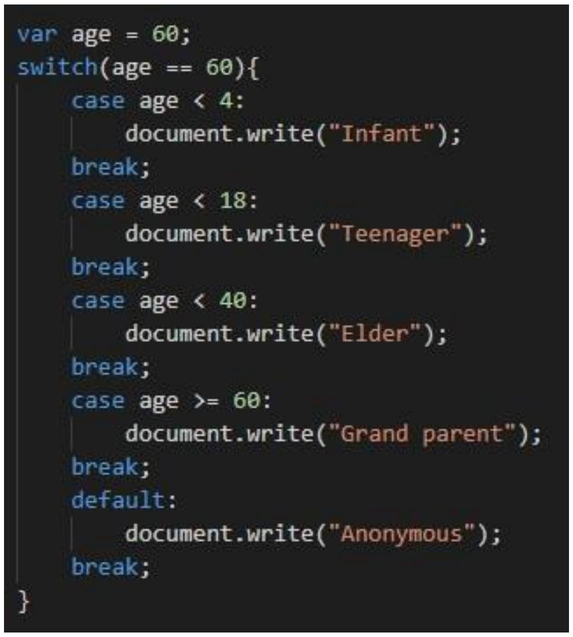
```
var age = 60;
if(age == 60){
    document.write("Is time to rest Sir!");
}else if(age < 15){
    document.write("Be serious with your study son!");
}else{
    document.write("Enjoy your time Sir!");
}
```

Fig. 2.83 (JS Conditions: else if statement)

JS Switch statement

The switch statement is used to perform different actions based on different conditions.

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.



```
var age = 60;
switch(age == 60){
    case age < 4:
        document.write("Infant");
    break;
    case age < 18:
        document.write("Teenager");
    break;
    case age < 40:
        document.write("Elder");
    break;
    case age >= 60:
        document.write("Grand parent");
    break;
    default:
        document.write("Anonymous");
    break;
}
```

Fig. 2.84 (JS switch statement)

When JavaScript reaches a **break keyword**, it breaks out of the switch block. This will stop the execution of inside the block. It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Note: If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

The **default keyword** specifies the code to run if there is no case match.

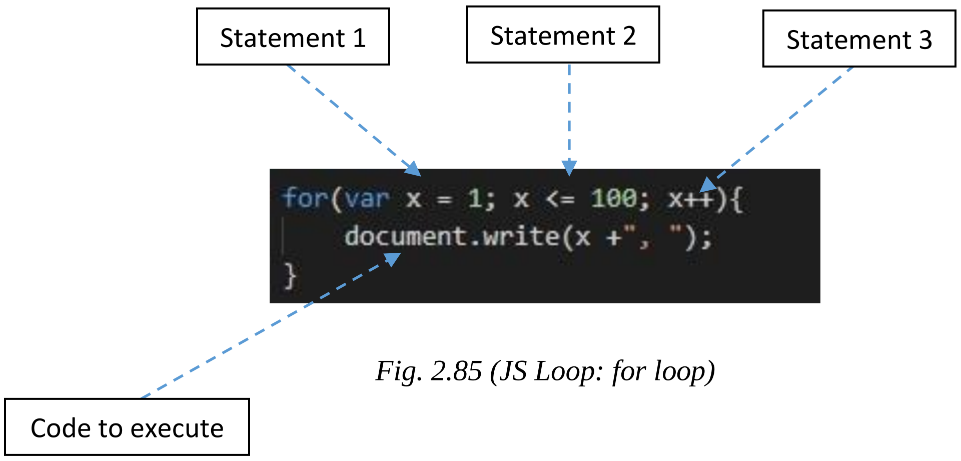
If multiple cases matches a case value, the **first** case is selected. If no matching cases are found, the program continues to the **default** label. If no default label is found, the program continues to the statement(s) **after the switch**.

JS Loop

We have used loops when we were learning arrays. Loops are handy, if you want to run the same code over and over again, each time with a different value.

JavaScript supports different kinds of loops:

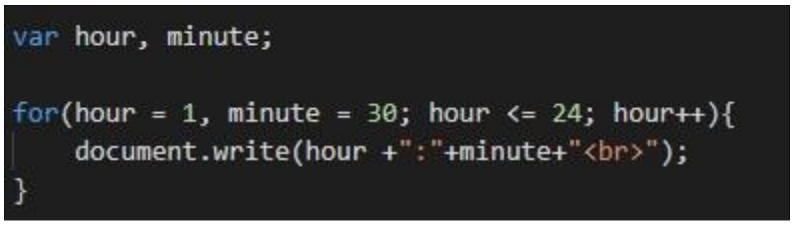
- for - loops through a block of code a number of times.
- while - loops through a block of code while a specified condition is true.
- do/while - also loops through a block of code while a specified condition is true.



```
for(var x = 1; x <= 100; x++){
    document.write(x +", ");
}
```

Fig. 2.85 (JS Loop: for loop)

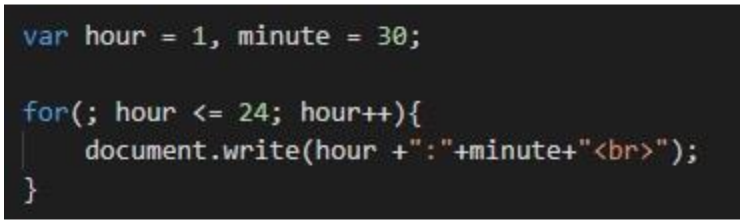
Statement 1 is executed (one time) before the execution of the code block. **Statement 1** sets a variable before the loop starts (**var x = 1**). Normally you will use **statement 1** to initialize the variable used in the loop (x = 1). This is **not always the case**, JavaScript doesn't care. **Statement 1 is optional**. You can initiate many values in statement 1 (separated by comma).



```
var hour, minute;

for(hour = 1, minute = 30; hour <= 24; hour++){
    document.write(hour +":"+minute+"<br>");
}
```

Fig. 2.86 (JS Loop: for loop initiating many values)

Note: And you can omit statement 1 (like when your values are set before the loop starts).

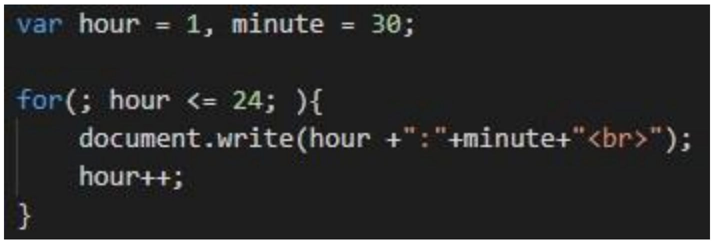


```
var hour = 1, minute = 30;

for(; hour <= 24; hour++){
    document.write(hour +":"+minute+"<br>");
}
```

Fig. 2.87 (JS Loop: for loop omitting statement 1)

Statement 2 defines the condition for executing the code block. **Statement 2** defines the condition for the loop to run (**x must be less than or equal to 100**). Often statement 2 is used to evaluate the condition of the initial variable. This is not always the case, JavaScript doesn't care. **Statement 2 is also optional**. If statement 2 returns true, the loop will start over again, if it returns false, the loop will end. If you omit statement 2, you must provide a break inside the loop. Otherwise the loop will never end. This will **crash your browser**. We will learn about breaks later in this book.

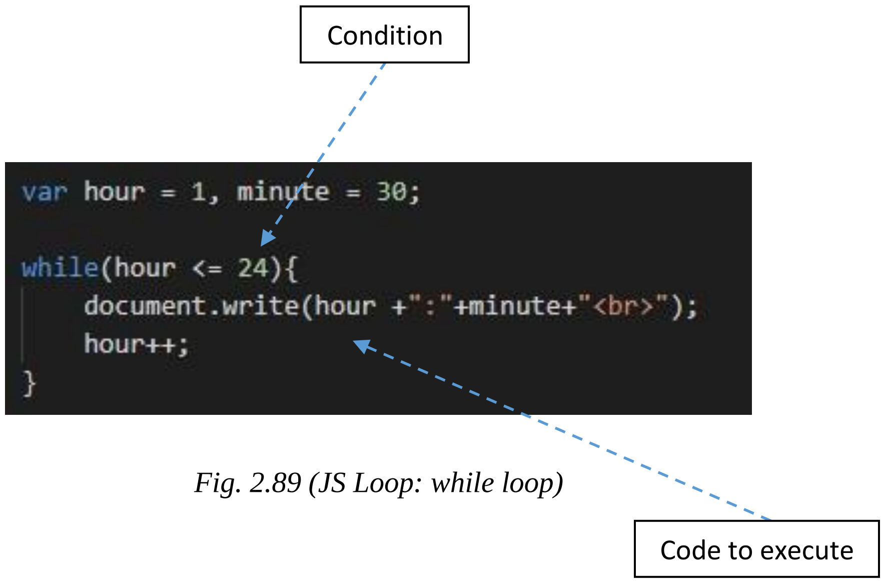
Statement 3 is executed (every time) after the code block has been executed. **Statement 3** increases a value (**x++**) each time the code block in the loop has been executed. Often statement 3 increments the value of the initial variable. This is not always the case, JavaScript doesn't care, and **statement 3 is optional**. **Statement 3** can do anything like negative increment (i--), positive increment (i = i + 15), or anything else. Statement 3 can also be omitted (like when you increment your values inside the loop) as shown below in *fig. 2.88*:



```
var hour = 1, minute = 30;

for(; hour <= 24; ){
    document.write(hour +":"+minute+"<br>");
    hour++;
}
```

Fig. 2.88 (JS Loop: for loop omitting statement 3)

The **while loop** loops through a block of code as long as a specified condition is true.

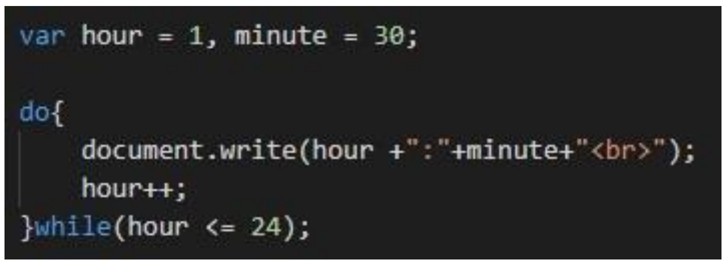


```
var hour = 1, minute = 30;

while(hour <= 24){
    document.write(hour +":"+minute+"<br>");
    hour++;
}
```

Fig. 2.89 (JS Loop: while loop)

Note: If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The **do/while loop** is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

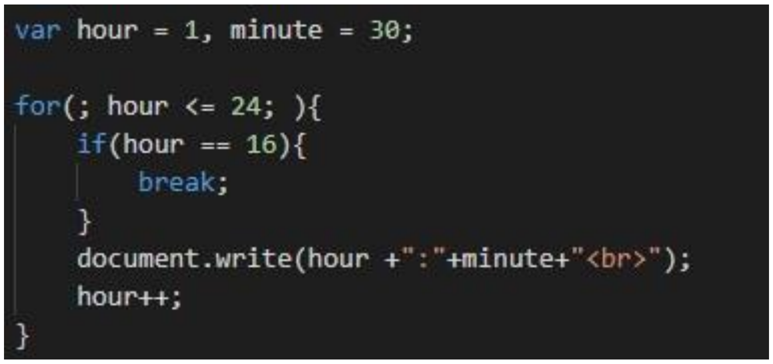


```
var hour = 1, minute = 30;

do{
    document.write(hour +":"+minute+"<br>");
    hour++;
}while(hour <= 24);
```

Fig. 2.90 (JS Loop: do/while loop)

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

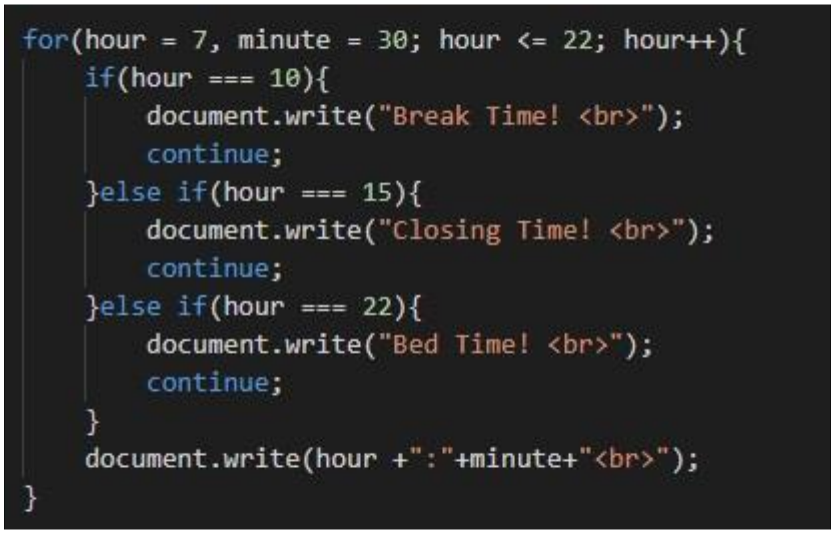
The **break statement** "jumps out" of a loop. The break statement breaks the loop and continues executing the code after the loop (if any). The for loop below in *fig. 2.91* breaks when the hour reaches 16, the loop will stop executing.



```
var hour = 1, minute = 30;

for(; hour <= 24; ){
    if(hour == 16){
        break;
    }
    document.write(hour +":"+minute+"<br>");
    hour++;
}
```

Fig. 2.91 (JS Loop: break statement)

The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop. The code below in *fig. 2.92* when the hour reaches **10**, it will print “**Break time**” and continue. When it reaches **15**, it will print “**Closing time**” and continue. When it reaches **22**, it will print “**Bed Time**”:

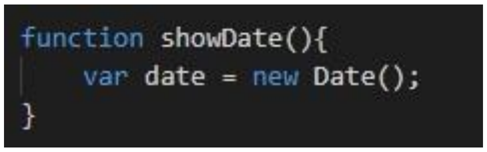


```
for(hour = 7, minute = 30; hour <= 22; hour++){
    if(hour === 10){
        document.write("Break Time! <br>");
        continue;
    }else if(hour === 15){
        document.write("Closing Time! <br>");
        continue;
    }else if(hour === 22){
        document.write("Bed Time! <br>");
        continue;
    }
    document.write(hour +":"+minute+"<br>");
}
```

Fig. 2.92 (JS Loop: continue statement)

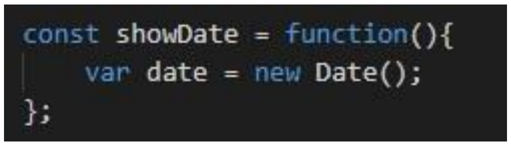
JS Arrow Function

Arrow functions were introduced in ES6. Arrow functions allow us to write shorter function syntax.



```
function showDate(){
    var date = new Date();
}
```

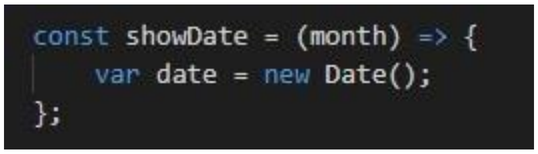
Fig. 2.93 (JS function: function declaration)



```
const showDate = function(){
    var date = new Date();
};
```

Fig. 2.94 (JS function: function expression)

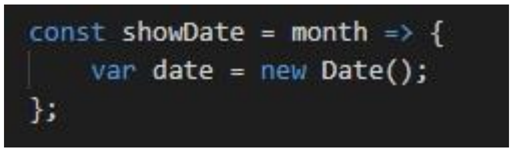
Note: you have to put a semi-colon at the end of a "function expression" but in "function declaration" you don't.



```
const showDate = (month) => {
    var date = new Date();
};
```

Fig. 2.95 (JS function: arrow function)

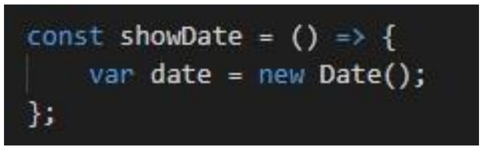
Note: if you have only one parameter, you don't need the bracket. You can write the function as shown below in *fig. 2.96*:



```
const showDate = month => {
    var date = new Date();
};
```

Fig. 2.96 (JS function: arrow function II)

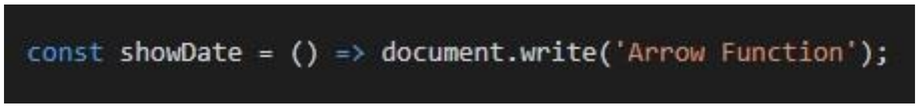
Note: if they are not parameters, you must include the bracket as shown below in *fig. 2.97*:



```
const showDate = () => {
    var date = new Date();
};
```

Fig. 2.97 (JS function: arrow function III)

Note: if you are having a single return statement, you can write the function as shown below in *fig. 2.98*:



```
const showDate = () => document.write('Arrow Function');
```

Fig. 2.98 (JS function: arrow function IV)

JS Debugging

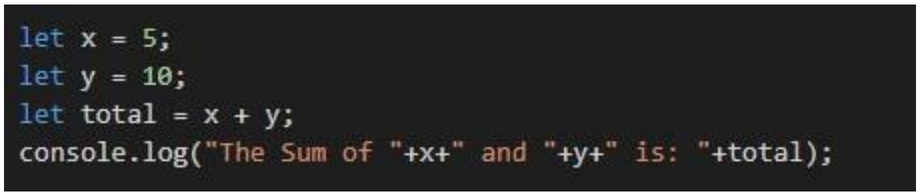
Programming code might contain syntax errors, or logical errors. Many of these errors are difficult to diagnose.

Often, when programming code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors. Searching for (and fixing) errors in programming code is called code debugging.

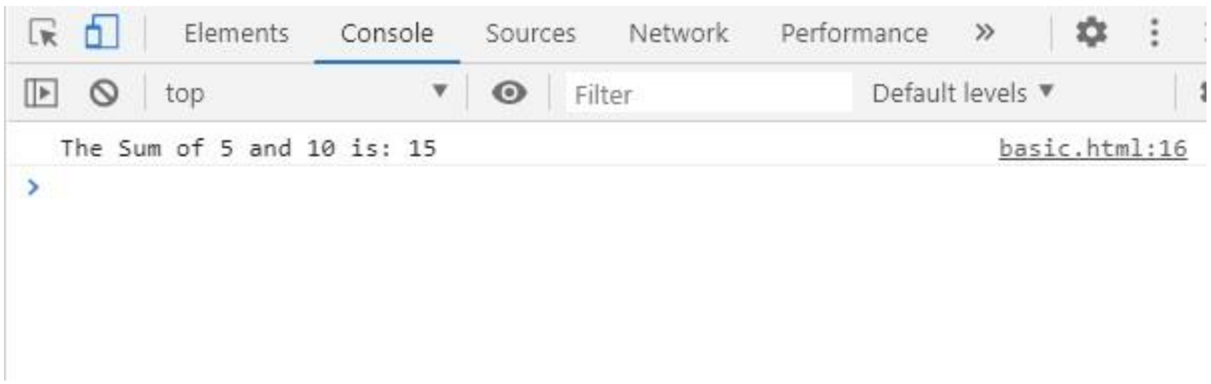
Debugging is not easy. All modern browsers have a built-in JavaScript debugger. Built-in debuggers can be turned on and off, forcing errors to be reported to the user.

You activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

If your browser supports debugging, you can use console.log ( ) to display JavaScript values in the debugger window.



```
let x = 5;
let y = 10;
let total = x + y;
console.log("The Sum of "+x+" and "+y+" is: "+total);
```



```
The Sum of 5 and 10 is: 15                         basic.html:16
```

Fig. 2.99 (JS Debugging)

JS HTML DOM (Document Object Model)

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

When a web page is loaded, the browser creates a Document Object Model of the page.
The HTML DOM model is constructed as a tree of Objects.

As a browser loads a web page, it creates a model of that page. The model is called DOM tree and it is stored in the browser’s memory. It consists of four main types of nodes:

- **Document Node**: it represents the entire page, when you access any element, attribute or text node. You navigate to it via the document node.
- **Element Node**: HTML elements describe the structure of an HTML page. To access the DOM tree, you start by looking for elements. Once you find the element you want, then you can access its text and attribute nodes if you want.
- **Attribute Node**: the opening tags of HTML elements can carry attributes and these are represented by attribute nodes in the DOM tree. Attribute nodes are not children of the element that carries them; they are part of that element.
- **Text Node**: once you have accessed an element node, you can then reach the text within that element. This is stored in its own text node.

Accessing and updating the DOM tree involves two steps:

- Locate the node that represents the element you want to work with.
- Use its text content, child elements and attributes.

Step 1: Access the elements.

a) Select an individual element node:

    i. getElementById ( )

    ii. querySelector ( ): uses a CSS selector and returns the first matching element.

b) select multiple elements (node lists)

    i. getElementsByClassName ( )

    ii. getElementsByTagName ( )

    iii. querySelectorAll ( ): uses a CSS selector to select all matching element.

c) traversing between element nodes

    i. parentNode

    ii. previousSibling / nextSibling

    iii. firstChild / lastChild

Step 2: Work with those elements

d) Access / update text nodes:

    i. nodeValue: this property lets you access or update contents of a text node.

e) Work with HTML content:

    i. innerHTML

    ii. createTextNode ( )

    iii. createElement ( )

    iv. appendChild ( ) / removeChild ( )

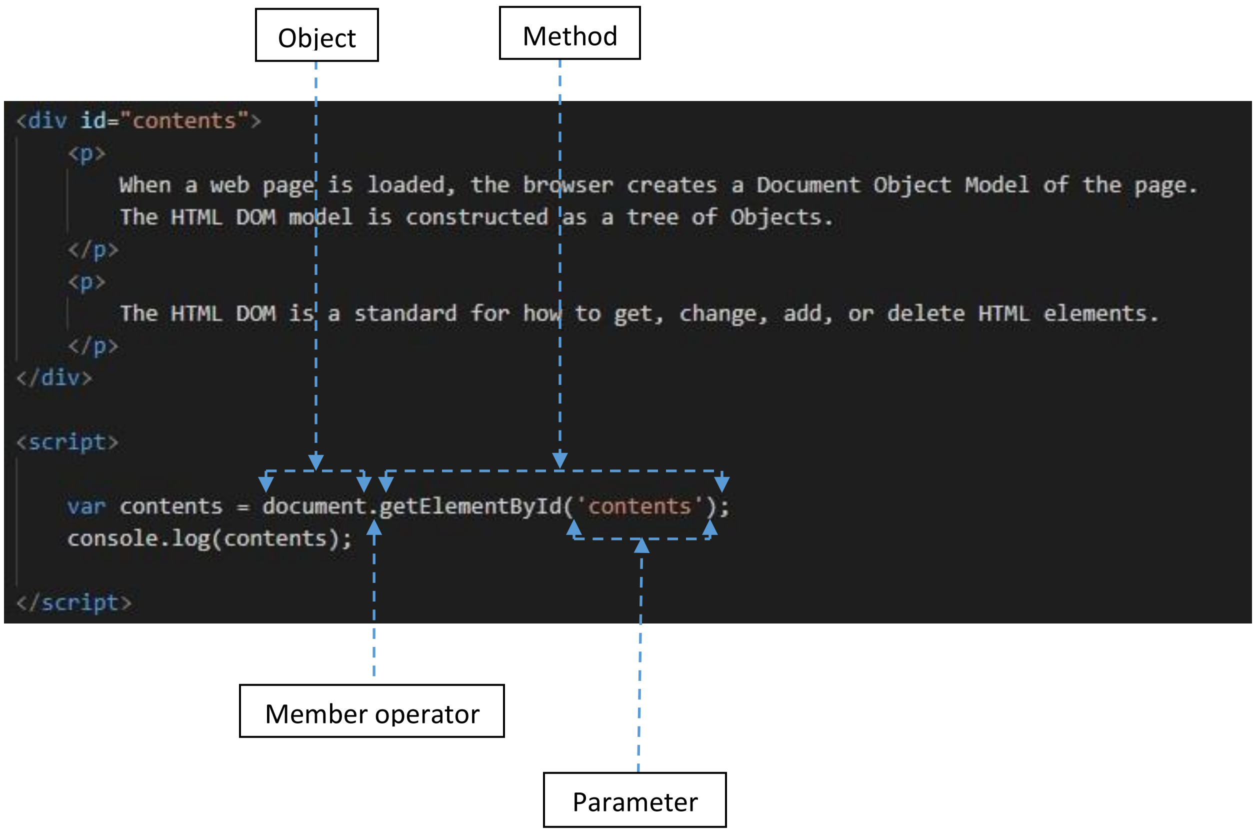
f) Access or update attribute values

  i. hasAttribute ( )

  ii. getAttribute ( )

  iii. setAttribute ( )

  iv. removeAttribute ( )



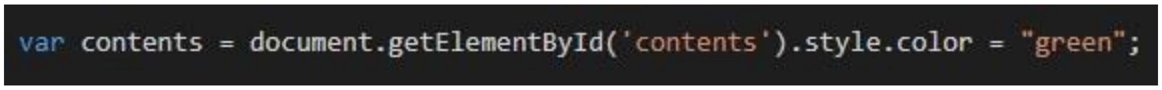
```
<div id="contents">
    <p>
        When a web page is loaded, the browser creates a Document Object Model of the page.
        The HTML DOM model is constructed as a tree of Objects.
    </p>
    <p>
        The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
    </p>
</div>

<script>

    var contents = document.getElementById('contents');
    console.log(contents);

</script>
```

Fig. 2.100 (JS HTML DOM: getElementById ( ))

Foot note:

- *The dot notation indicates that method (on the right) is being applied to the node on the left of the period.*

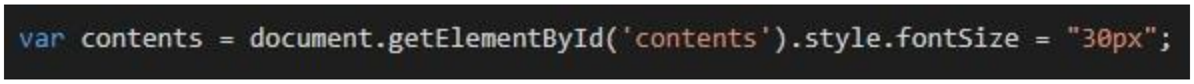
The HTML DOM also allows JavaScript to change the style of HTML elements. Let's try to change the text color of the div (contents) above in *fig. 2.100*:



```
var contents = document.getElementById('contents').style.color = "green";
```

Fig. 2.101 (JS HTML DOM: style property)

Note: when you applied the style property to an element, you can use any of the CSS properties you want. But **note** when the CSS property contains **two values** or **more** for instance font-size or background-color; if you want to attached that property to the element, using camel case naming like this fontSize or backgroundColor.

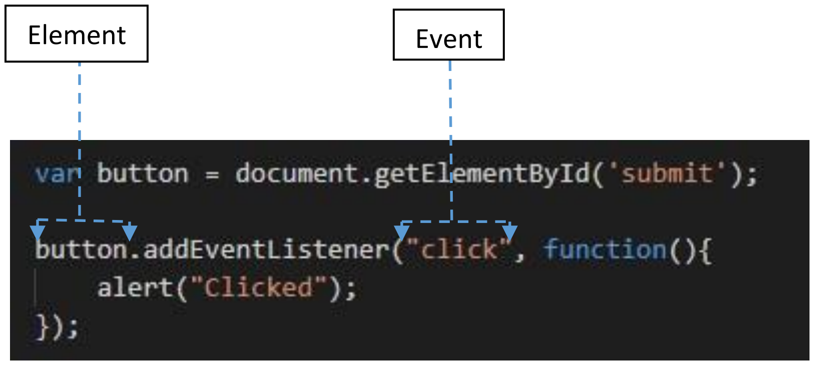


```
var contents = document.getElementById('contents').style.fontSize = "30px";
```

Fig. 2.102 (JS HTML DOM: joining two or more CSS properties)

JS Event Listeners

Event listeners are a more recent approach to handling events. They can deal with more than one function at a time but they are not supported in older browsers. You can add many event handlers to one element.



```
var button = document.getElementById('submit');

button.addEventListener("click", function(){
    alert("Clicked");
});
```

Fig. 2.103 (JS Event listeners)

Note: if you want need to remove an event listener, there is a function called removeEventListener ( ) which removes the event listener from the specified element.

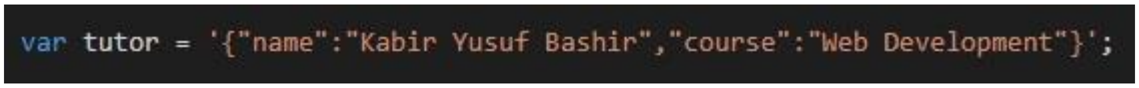
JS JSON

JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a lightweight data interchange format. JSON is "self-describing" and easy to understand.

JSON is a format for storing and transporting data. JSON is often used when data is sent from a server to a web page.

JSON has some syntax format below:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays



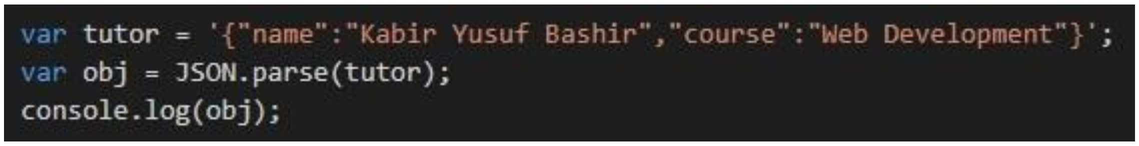
```
var tutor = '{"name":"Kabir Yusuf Bashir","course":"Web Development"}';
```

Fig. 2.104 (JS JSON)

Foot Note:

- *JSON names require double quotes. JavaScript names do not.*

A common use of JSON is to read data from a web server, and display the data in a web page. The JavaScript built-in function JSON.parse ( ) to convert the string into a JavaScript object.



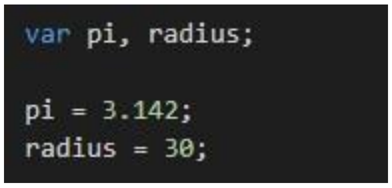
```
var tutor = '{"name":"Kabir Yusuf Bashir","course":"Web Development"}';
var obj = JSON.parse(tutor);
console.log(obj);
```

Fig. 2.105 (JS JSON: JSON.parse ( ))

JS Best Practices

It is a good coding practice to put all declarations at the top of each script or function. This will:

- Give cleaner code
- Provide a single place to look for local variables
- Make it easier to avoid unwanted (implied) global variables
- Reduce the possibility of unwanted re-declarations

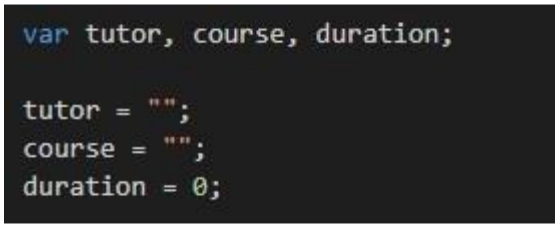


```
var pi, radius;

pi = 3.142;
radius = 30;
```

Fig. 2.106 (JS Best Practice: variable declaration)

It is a good coding practice to initialize variables when you declare them. This will:

- Give cleaner code
- Provide a single place to initialize variables
- Avoid undefined values



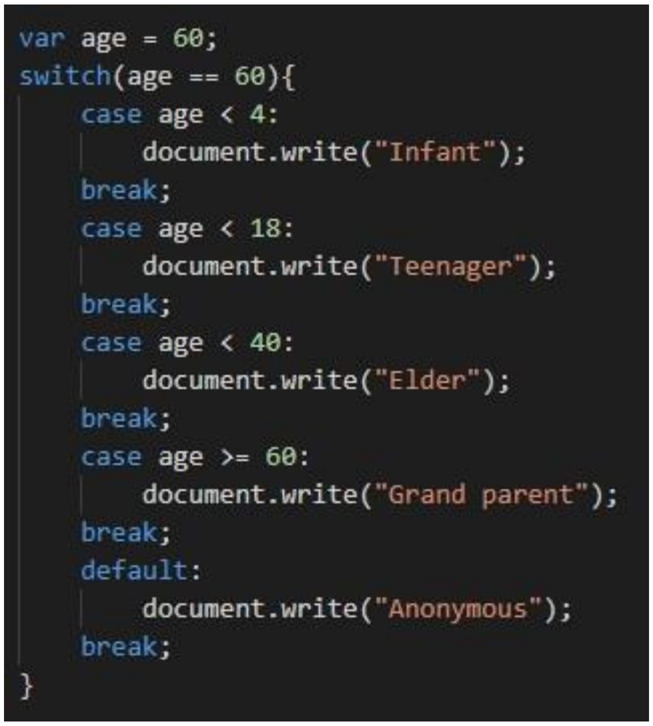
```
var tutor, course, duration;

tutor = "";
course = "";
duration = 0;
```

Fig. 2.107 (JS Best Practice: Initializing variables)

Note: Initializing variables provides an idea of the intended use (and intended data type).

Use = = = Comparison when comparing variables, the = = = operator forces comparison of values and type.

Always end your switch statements with a default. Even if you think there is no need for it.



```
var age = 60;
switch(age == 60){
    case age < 4:
        document.write("Infant");
    break;
    case age < 18:
        document.write("Teenager");
    break;
    case age < 40:
        document.write("Elder");
    break;
    case age >= 60:
        document.write("Grand parent");
    break;
    default:
        document.write("Anonymous");
    break;
}
```

Fig. 2.108 (JS Best Practice: ending switch statement with a default)

Exercise on JavaScript

1. What is a JavaScript identifier?
2. What is a variable?
3. Explain any of the two (2) JavaScript operators you know.
4. Explain JavaScript string data type.
5. What are JavaScript events?
6. Mention the three (3) JavaScript methods used to convert variables to number.
7. To get the length of an array, what JavaScript method will you use?
8. Write short note on JavaScript date object.
9. Write short note on JavaScript Logical operators.
10. Write short note on JavaScript DOM.

Project on JavaScript

Using your personal portfolio web page you created in chapter 1; create three (3) different styles for the web page; use JavaScript to change the style of your web page to the first style when the time is from 00:00 to 11:59; and change the style of your web page to the second style when the time is from 12:00 to 18:59; and lastly change the style of your web page to the third style when the time is from 19:00 to 23:59.

REFERENCES

1. https://www.w3schools.com/js/default.asp
2. https://www.w3schools.com/js/js_intro.asp
3. https://www.w3schools.com/js/js_output.asp
4. https://www.w3schools.com/js/js_syntax.asp
5. https://www.w3schools.com/js/js_comments.asp
6. https://www.w3schools.com/js/js_variables.asp
7. https://www.w3schools.com/js/js_operators.asp
8. https://www.w3schools.com/js/js_datatypes.asp
9. https://www.w3schools.com/js/js_functions.asp
10. https://www.w3schools.com/js/js_objects.asp
11. https://www.w3schools.com/js/js_events.asp
12. https://www.w3schools.com/js/js_strings.asp
13. https://www.w3schools.com/js/js_string_methods.asp
14. https://www.w3schools.com/js/js_numbers.asp
15. https://www.w3schools.com/js/js_number_methods.asp
16. https://www.w3schools.com/js/js_arrays.asp
17. https://www.w3schools.com/js/js_array_methods.asp
18. https://www.w3schools.com/js/js_dates.asp
19. https://www.w3schools.com/js/js_date_formats.asp
20. https://www.w3schools.com/js/js_math.asp
21. https://www.w3schools.com/js/js_comparisons.asp
22. https://www.w3schools.com/js/js_if_else.asp
23. https://www.w3schools.com/js/js_switch.asp
24. https://www.w3schools.com/js/js_loop_for.asp
25. https://www.w3schools.com/js/js_loop_while.asp

26. https://www.w3schools.com/js/js_break.asp
27. https://www.w3schools.com/js/js_let.asp
28. https://www.w3schools.com/js/js_arrow_function.asp
29. https://www.w3schools.com/js/js_const.asp
30. https://www.w3schools.com/js/js_debugging.asp
31. https://www.w3schools.com/js/js_htmldom.asp
32. Jon Duckett JavaScript & jQuery
33. https://www.w3schools.com/js/js_htmldom_css.asp
34. https://www.w3schools.com/js/js_htmldom_eventlistener.asp
35. https://www.w3schools.com/js/js_json.asp
36. https://www.w3schools.com/js/js_best_practices.asp

Chapter 3: Bootstrap

Bootstrap was developed by Mark Otto and Jacob Thornton at Twitter, and released as an open source product in August 2011 on GitHub.

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites.

Bootstrap is a free front-end framework for faster and easier web development. Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins. Bootstrap also gives you the ability to easily create responsive designs.

Bootstrap 4 is the newest version of Bootstrap; with new components, faster style sheet and more responsiveness. Bootstrap 4 supports the latest, stable releases of all major browsers and platforms. However, Internet Explorer 9 and down is not supported.

Advantages of Bootstrap:

- **Easy to use:** Anybody with just basic knowledge of HTML and CSS can start using Bootstrap
- **Responsive features:** Bootstrap's responsive CSS adjusts to phones, tablets, and desktops
- **Mobile-first approach:** In Bootstrap 3, mobile-first styles are part of the core framework
- **Browser compatibility:** Bootstrap is compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Safari, and Opera)

There are two ways to start using Bootstrap on your own web site. You can:

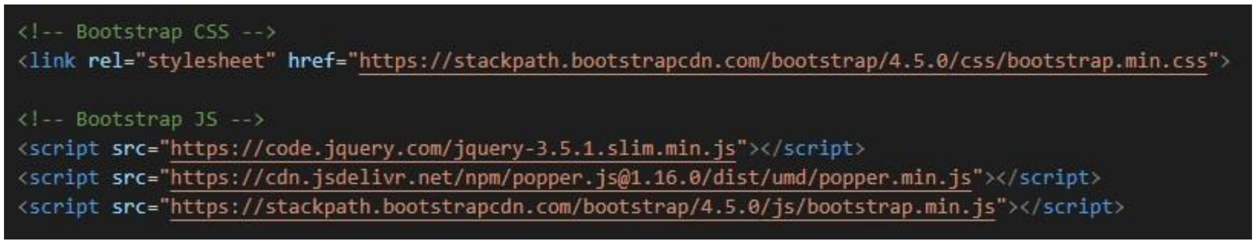
- Download Bootstrap from getbootstrap.com
- Include Bootstrap from a CDN

Note: Bootstrap is completely free to download and use!

BS Get Started

If you want to download and host Bootstrap yourself, go to getbootstrap.com, and follow the instructions there.

If you don't want to download and host Bootstrap yourself, you can include it from a CDN (Content Delivery Network).



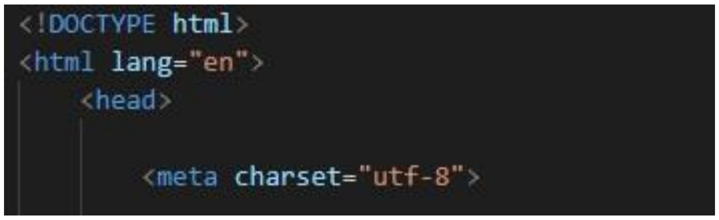
```
<!-- Bootstrap CSS -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">

<!-- Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"></script>
```

Fig. 3.001 (BS Get Started: including Bootstrap from CDN)

One advantage of using the Bootstrap CDN than downloading Bootstrap from getbootstrap.com, is faster loading time. Also, most CDN's will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time.

To create a web page with Bootstrap 4, follow the steps below:

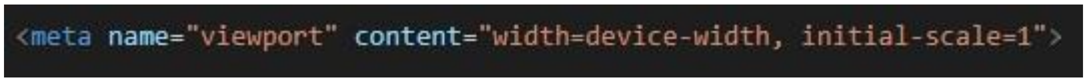
- Bootstrap 4 uses HTML elements and CSS properties that require the HTML5 doctype. Always include the HTML5 doctype at the beginning of the page, along with the lang attribute and the correct character set:



```
<!DOCTYPE html>
<html lang="en">
    <head>

        <meta charset="utf-8">
```

Fig. 3.002 (BS Get Started: adding HTML5 doctype)

- Bootstrap 4 is designed to be responsive to mobile devices. Mobile-first styles are part of the core framework. To ensure proper rendering and touch zooming, add the following <meta> tag inside the <head> element:



```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

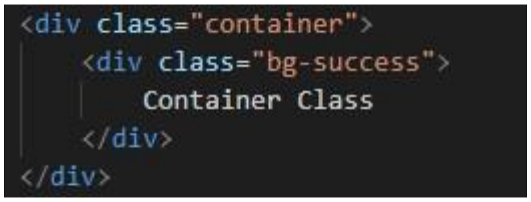
Fig. 3.003 (BS Get Started: adding meta tag for viewport)

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

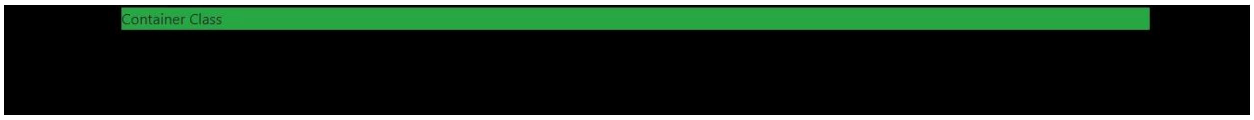
The initial-scale=1 part sets the initial zoom level when the page is first loaded by the browser.

- Bootstrap 4 also requires a containing element to wrap site contents. There are two container classes to choose from:

  1. The .container class provides a responsive **fixed width container**
  2. The .container-fluid class provides a **full width container**, spanning the entire width of the viewport.

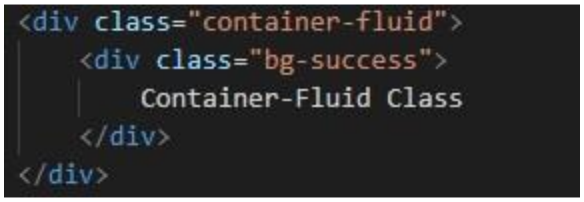


```
<div class="container">
    <div class="bg-success">
        Container Class
    </div>
</div>
```

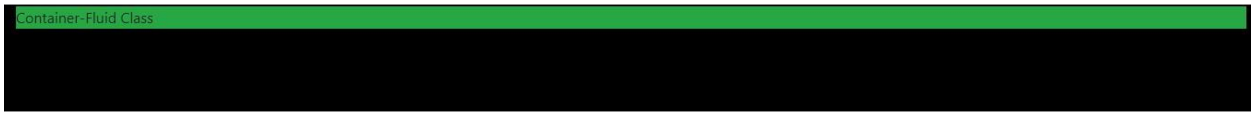


Container Class

Fig. 3.004 (BS container class)



```html
<div class="container-fluid">
    <div class="bg-success">
        Container-Fluid Class
    </div>
</div>
```



Container-Fluid Class

Fig. 3.005 (BS container-fluid class)

Note: don’t worry if you find it difficult understanding the codes above, trust me, it will become second nature as you continue reading this book. Smile!

BS Grid System

Bootstrap's grid system is built with flexbox and allows up to 12 columns across the page. If you do not want to use all 12 columns individually, you can group the columns together to create wider columns. The grid system is responsive, and the columns will re-arrange automatically depending on the screen size. Make sure that the sum adds up to 12 or fewer (it is not required that you use all 12 available columns).

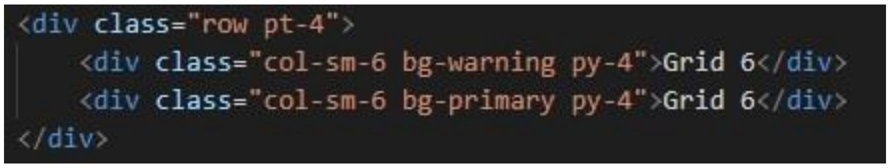
The Bootstrap 4 grid system has five classes:

- .col- (extra small devices - screen width less than 576px)
- .col-sm- (small devices - screen width equal to or greater than 576px)
- .col-md- (medium devices - screen width equal to or greater than 768px)
- .col-lg- (large devices - screen width equal to or greater than 992px)
- .col-xl- (xlarge devices - screen width equal to or greater than 1200px)

The classes above can be combined to create more dynamic and flexible layouts.

Tip: Each class scales up, so if you wish to set the same widths for sm and md, you only need to specify sm.

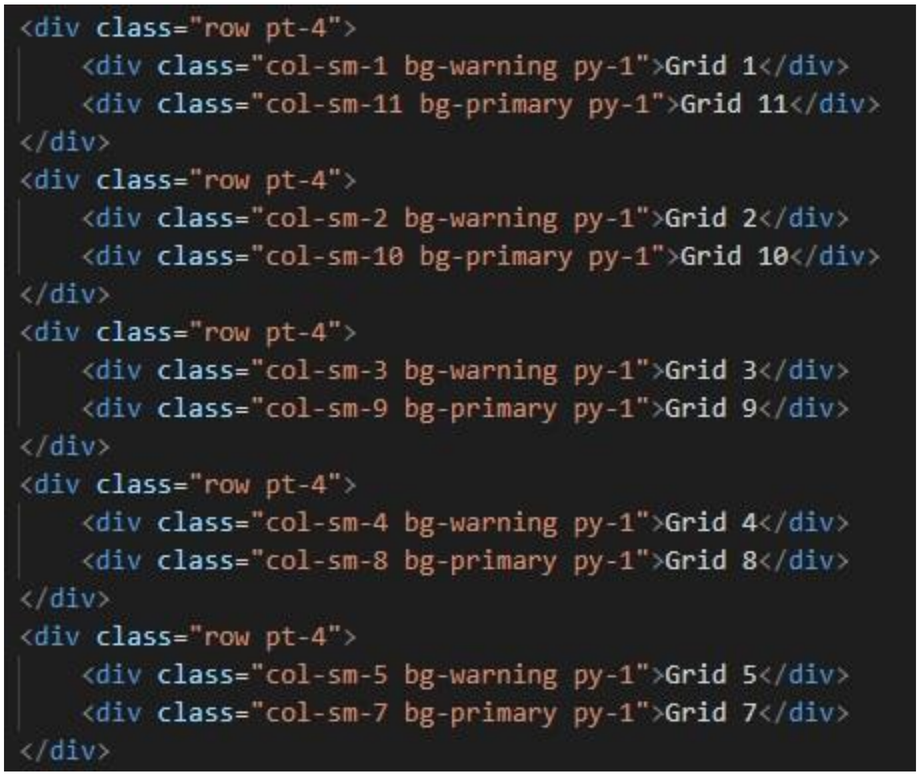
First; create a row (<div class="row">). Then, add the desired number of columns (tags with appropriate .col-*-* classes). Note that numbers in .col-*-* should always add up to 12 for each row as show below in *fig. 3.006*:



```
<div class="row pt-4">
    <div class="col-sm-6 bg-warning py-4">Grid 6</div>
    <div class="col-sm-6 bg-primary py-4">Grid 6</div>
</div>
```

Fig. 3.006 (BS Grid System)

Note: In *fig. 3.006* above, we created two divs inside the row and each div has a class of col-sm-6. When you add the two columns, they will add up to 12.



```
<div class="row pt-4">
    <div class="col-sm-1 bg-warning py-1">Grid 1</div>
    <div class="col-sm-11 bg-primary py-1">Grid 11</div>
</div>
<div class="row pt-4">
    <div class="col-sm-2 bg-warning py-1">Grid 2</div>
    <div class="col-sm-10 bg-primary py-1">Grid 10</div>
</div>
<div class="row pt-4">
    <div class="col-sm-3 bg-warning py-1">Grid 3</div>
    <div class="col-sm-9 bg-primary py-1">Grid 9</div>
</div>
<div class="row pt-4">
    <div class="col-sm-4 bg-warning py-1">Grid 4</div>
    <div class="col-sm-8 bg-primary py-1">Grid 8</div>
</div>
<div class="row pt-4">
    <div class="col-sm-5 bg-warning py-1">Grid 5</div>
    <div class="col-sm-7 bg-primary py-1">Grid 7</div>
</div>
```

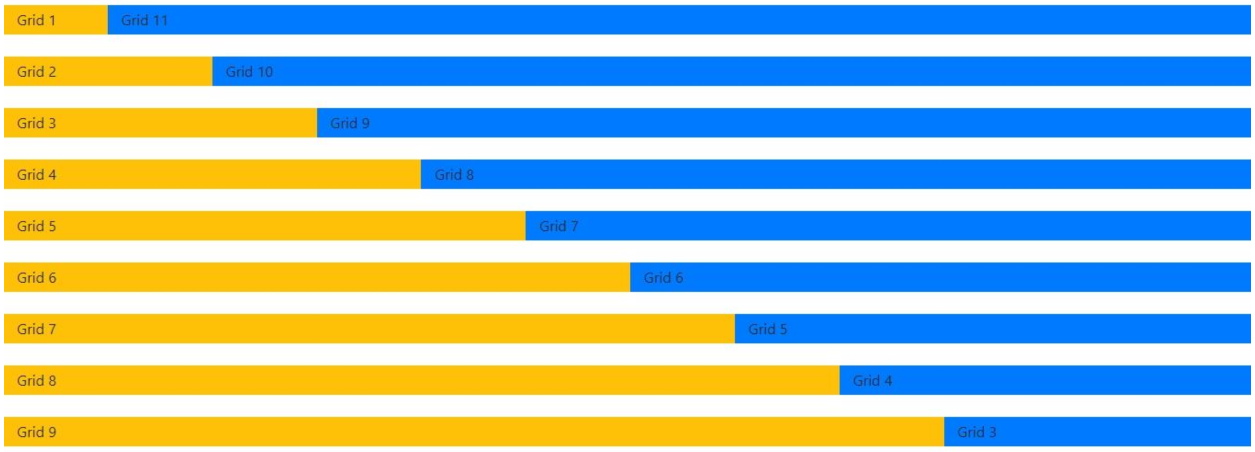


Fig. 3.007 (BS Grid System: 12 Grid system)

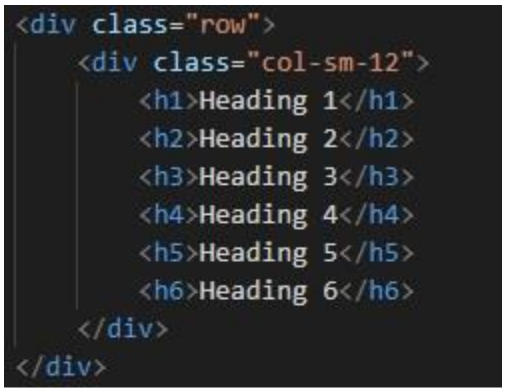
Foot Note:

- *Inside a row, you can have more than two divs just make sure they add up to 12.*

BS Text/Typography

Bootstrap 4 uses a default font-size of 16px, and its line-height is 1.5. The default font-family is "Helvetica Neue", Helvetica, Arial, sans-serif. In addition, all <p> elements have margin-top: 0 and margin-bottom: 1rem (16px by default)

Bootstrap 4 styles HTML headings (<h1> to <h6>) with a bolder font-weight and an increased font-size.



```
<div class="row">
    <div class="col-sm-12">
        <h1>Heading 1</h1>
        <h2>Heading 2</h2>
        <h3>Heading 3</h3>
        <h4>Heading 4</h4>
        <h5>Heading 5</h5>
        <h6>Heading 6</h6>
    </div>
</div>
```

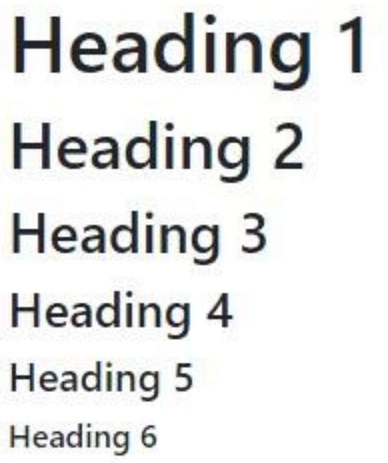
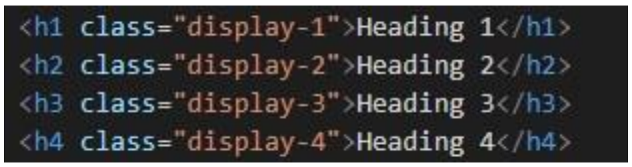


Fig. 3.008 (BS Typography: headings)

Display headings are used to stand out more than normal headings (larger font-size and lighter font-weight), and there are four classes to choose from: .display-1, .display-2, .display-3, and .display-4.



```
<h1 class="display-1">Heading 1</h1>
<h2 class="display-2">Heading 2</h2>
<h3 class="display-3">Heading 3</h3>
<h4 class="display-4">Heading 4</h4>
```

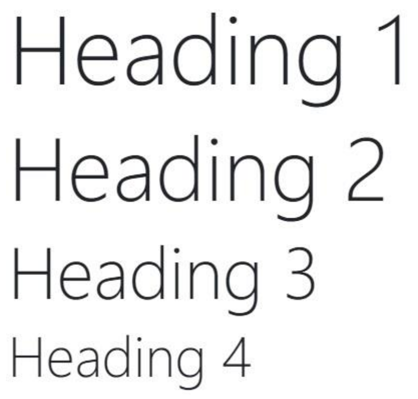


Fig. 3.008 (BS Typography: display class)

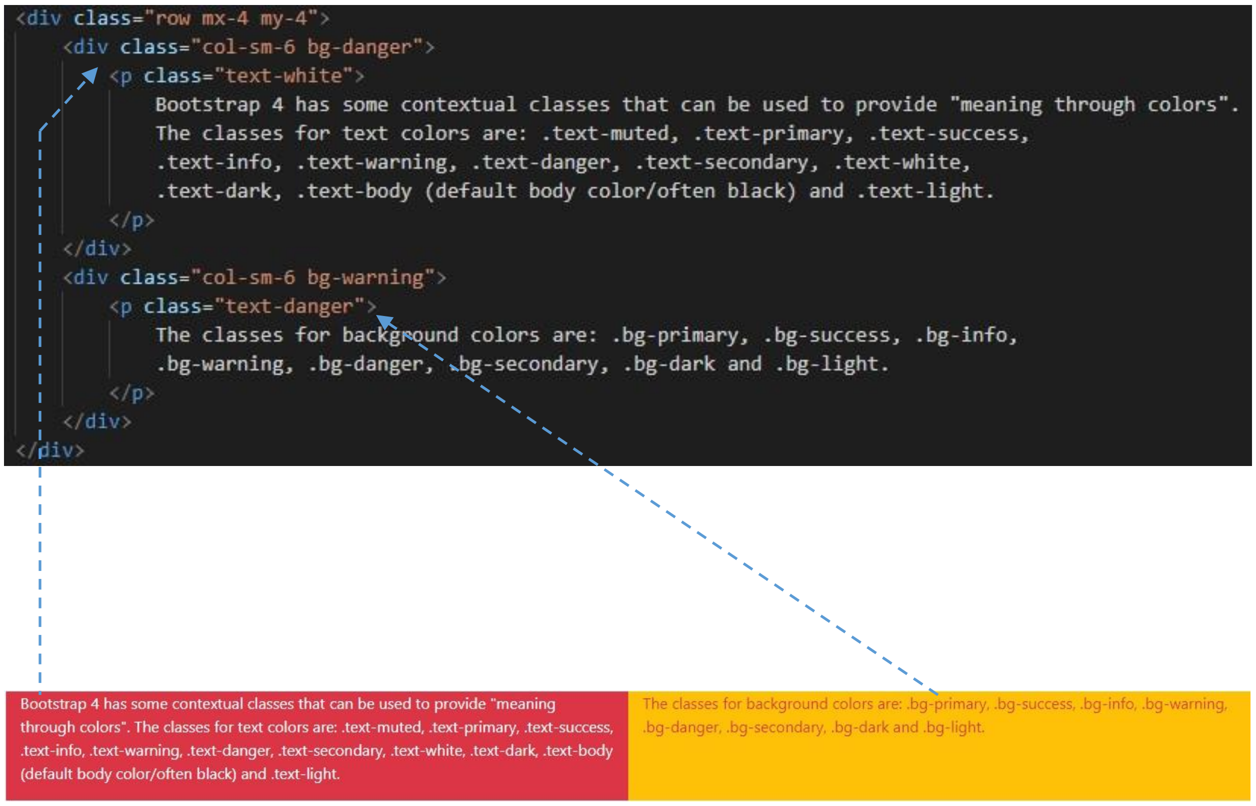
The Bootstrap 4 classes below can be added to style HTML elements further:

.font-weight-bold (Bold text), .font-weight-bolder (Bolder text), .font-italic (Italic text), .font-weight-light (Light weight text), .font-weight-lighter (Lighter weight text), .font-weight-normal (Normal text), .lead (Makes a paragraph stand out), .small (Indicates smaller text [set to 80% of the size of the parent]), .text-left (Indicates left-aligned text), .text-*-left (Indicates left-aligned text on small, medium, large or xlarge screens), .text-break (Prevents long text from breaking layout), .text-center (Indicates center-aligned text), .text-*-center (Indicates center-aligned text on small, medium, large or xlarge screens), .text-decoration-none (Removes the underline from a link), .text-right (Indicates right-aligned text), .text-*-right (Indicates right-aligned text on small, medium, large or xlarge screens), .text-justify (Indicates justified text), .text-monospace (Monospaced text), .text-nowrap (Indicates no wrap text), .text-lowercase (Indicates lowercased text), .text-reset (Resets the color of a text or a link [inherits the color from its parent]), .text-uppercase (Indicates uppercased text), .text-capitalize (Indicates capitalized text), .initialism (Displays the text inside an <abbr> element in a slightly smaller font size), .list-unstyled (Removes the default list-style and left margin on list items [works on both <ul> and <ol>]). This class only applies to immediate children list items (to remove the default list-style from any nested lists, apply this class to any nested lists as well) .list-inline (Places all list items on a single line [used together with .list-inline-item on each <li> elements]), .pre-scrollable (Makes a <pre> element scrollable)

BS Colors

Bootstrap 4 has some contextual classes that can be used to provide "meaning through colors". The classes for text colors are: .text-muted, .text-primary, .text-success, .text-info, .text-warning, .text-danger, .text-secondary, .text-white, .text-dark, .text-body (default body color/often black) and .text-light.

The classes for background colors are: .bg-primary, .bg-success, .bg-info, .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light.



```
<div class="row mx-4 my-4">
    <div class="col-sm-6 bg-danger">
        <p class="text-white">
            Bootstrap 4 has some contextual classes that can be used to provide "meaning through colors".
            The classes for text colors are: .text-muted, .text-primary, .text-success,
            .text-info, .text-warning, .text-danger, .text-secondary, .text-white,
            .text-dark, .text-body (default body color/often black) and .text-light.
        </p>
    </div>
    <div class="col-sm-6 bg-warning">
        <p class="text-danger">
            The classes for background colors are: .bg-primary, .bg-success, .bg-info,
            .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light.
        </p>
    </div>
</div>
```

Bootstrap 4 has some contextual classes that can be used to provide "meaning through colors". The classes for text colors are: .text-muted, .text-primary, .text-success, .text-info, .text-warning, .text-danger, .text-secondary, .text-white, .text-dark, .text-body (default body color/often black) and .text-light.

The classes for background colors are: .bg-primary, .bg-success, .bg-info, .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light.

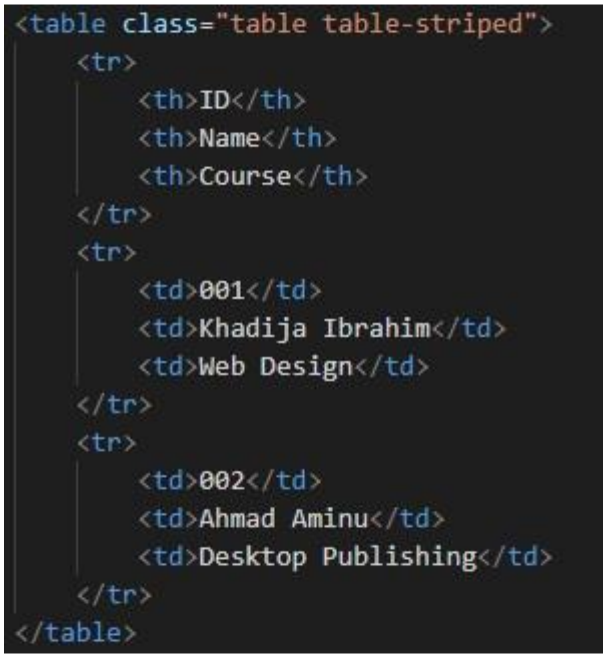
Fig. 3.010 (BS Colors)

BS Tables

We have learned about tables in the first chapter of this book. With Bootstrap, you can create an awesome table without writing any CSS property. A basic Bootstrap 4 table has a light padding and horizontal dividers.

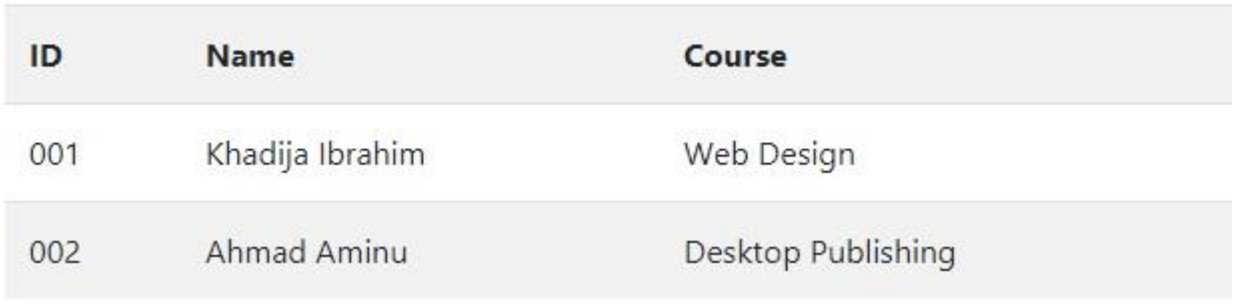
The .table class adds basic styling to a table.

The .table-striped class adds zebra-stripes to a table.



```html
<table class="table table-striped">
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Course</th>
    </tr>
    <tr>
        <td>001</td>
        <td>Khadija Ibrahim</td>
        <td>Web Design</td>
    </tr>
    <tr>
        <td>002</td>
        <td>Ahmad Aminu</td>
        <td>Desktop Publishing</td>
    </tr>
</table>
```

Fig. 3.011 (BS Tables: table-striped class)



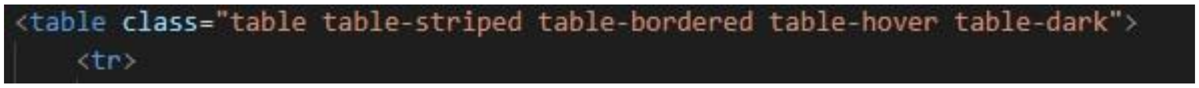
ID	Name	Course
001	Khadija Ibrahim	Web Design
002	Ahmad Aminu	Desktop Publishing

Note: whenever you want to use any of the table classes to your table, always start with the table class, then write whatever class you want to apply to the table.

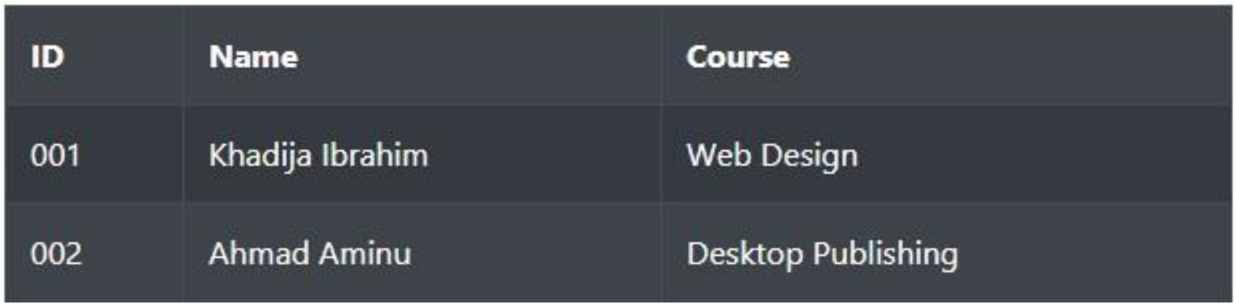
The .table-bordered class adds borders on all sides of the table and cells.

The .table-hover class adds a hover effect (grey background color) on table rows.

The .table-dark class adds a black background to the table.



```
<table class="table table-striped table-bordered table-hover table-dark">
    <tr>
```



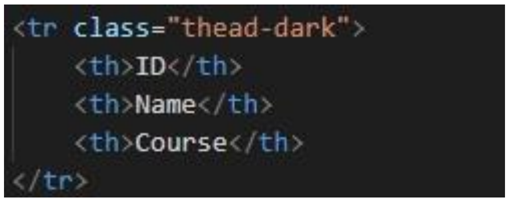
ID	Name	Course
001	Khadija Ibrahim	Web Design
002	Ahmad Aminu	Desktop Publishing

Fig. 3.012 (BS Tables: table-dark class)

The .table-borderless class removes borders from the table.

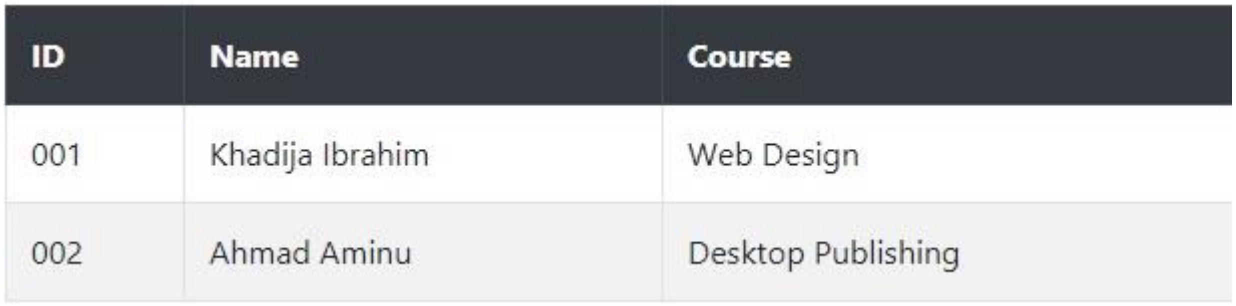
Contextual classes can be used to color the whole table (<table>), the table rows (<tr>) or table cells (<td>). The contextual classes that can be used are: .table-primary (Blue: Indicates an important action), .table-success (Green: Indicates a successful or positive action), .table-danger (Red: Indicates a dangerous or potentially negative action), .table-info (Light blue: Indicates a neutral informative change or action), .table-warning (Orange: Indicates a warning that might need attention), .table-active (Grey: Applies the hover color to the table row or table cell), .table-secondary (Grey: Indicates a slightly less important action), .table-light (Light grey table or table row background), .table-dark (Dark grey table or table row background).

The .thead-dark class adds a black background to table headers, and the .thead-light class adds a grey background to table headers



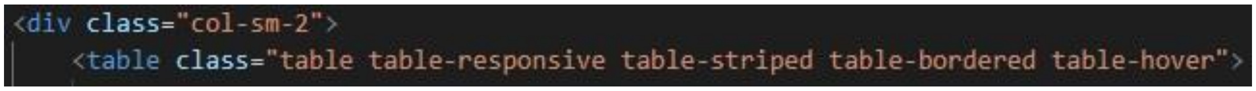
```
<tr class="thead-dark">
    <th>ID</th>
    <th>Name</th>
    <th>Course</th>
</tr>
```

Fig. 3.013 (BS Tables: thead-dark class)

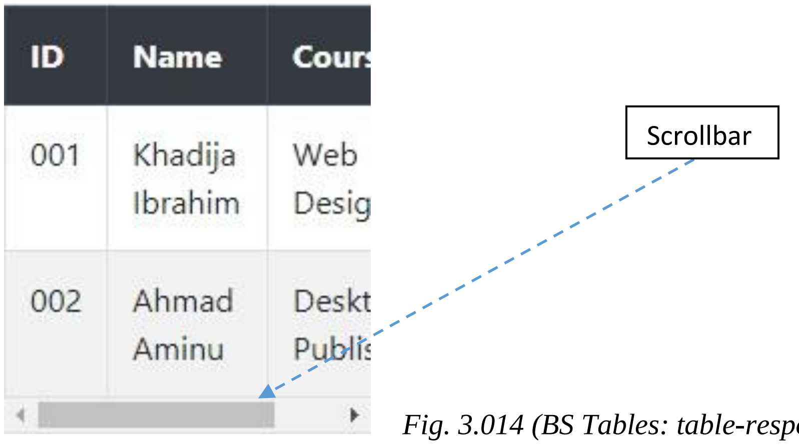


ID	Name	Course
001	Khadija Ibrahim	Web Design
002	Ahmad Aminu	Desktop Publishing

The .table-responsive class adds a scrollbar to the table when needed (when it is too big horizontally).



```
<div class="col-sm-2">
    <table class="table table-responsive table-striped table-bordered table-hover">
```



ID	Name	Cour
001	Khadija Ibrahim	Web Desig
002	Ahmad Aminu	Deskt Publis

Fig. 3.014 (BS Tables: table-responsive class)

BS Images

The .**rounded class** adds rounded corners to an image.

```
<img class="rounded" src="covid-19.jpg" alt="Covid 19">
```

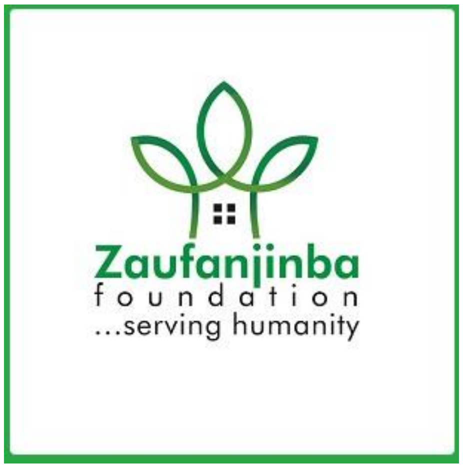


Fig. 3.015 (BS Images: rounded class)

The .**rounded-circle class** shapes the image to a circle.

```
<img class="rounded-circle" src="zaufoundation.png" alt="Foundation Logo">
```

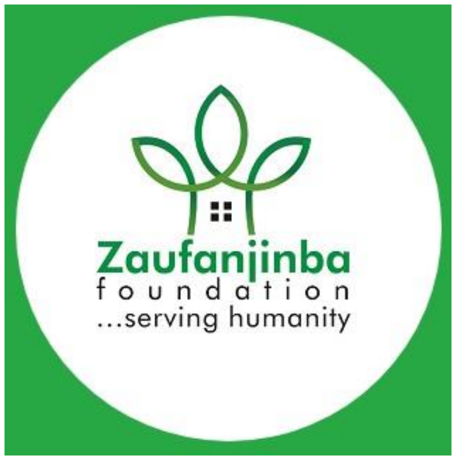


Fig. 3.016 (BS Images: rounded-circle class)

The **.img-thumbnail class** shapes the image to a thumbnail (bordered).

Float an image to the right with the **.float-right class** or to the left with **.float-left class**.

```
<div class="col-sm-6 p-4 bg-success text-white">
    <img class="rounded-circle float-right" src="zaufoundation.png" alt="Foundation Logo">
    Zaufanjinba foundation has touched the lives of many people within and
    outside Borno state through community services and youth empowerment.
    With the destruction of livelihoods brought about by the insurgency,
    resulting in communities displaced, the foundation currently maintains
    three IDP camps with a total population of about 200 persons catering
    for all their needs; which include food, shelter, healthcare and education
    for the children.
</div>
```

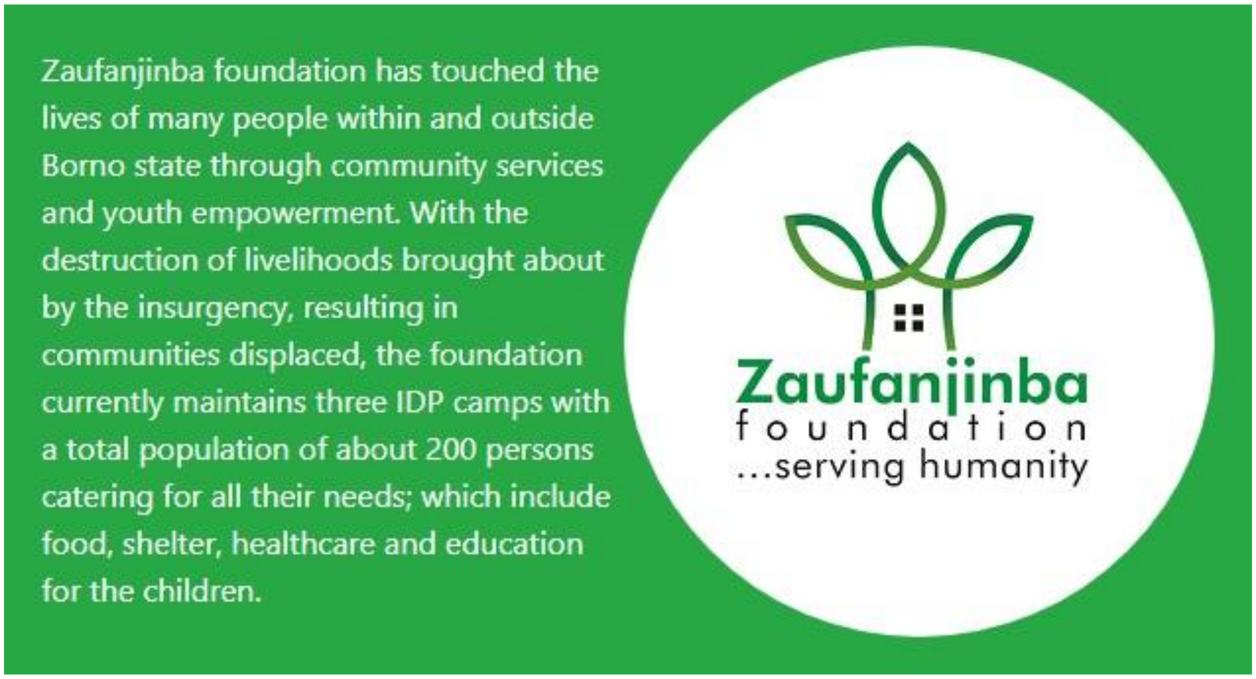

Zaufanjinba foundation has touched the lives of many people within and outside Borno state through community services and youth empowerment. With the destruction of livelihoods brought about by the insurgency, resulting in communities displaced, the foundation currently maintains three IDP camps with a total population of about 200 persons catering for all their needs; which include food, shelter, healthcare and education for the children.

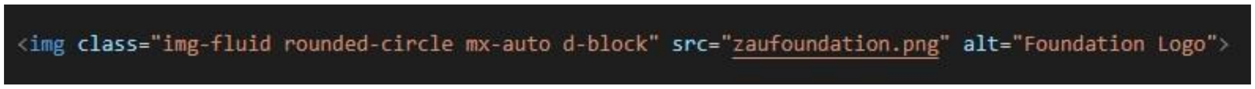
Fig. 3.017 (BS Images: float-right class)

You center an image by adding the utility class **.mx-auto class** and **.d-block class** to the image.

```
<img class="rounded-circle mx-auto d-block" src="zaufoundation.png" alt="Foundation Logo">
```

Fig. 3.018 (BS Images: mx-auto and d-block class)

Images come in all sizes. So do screens. Responsive images automatically adjust to fit the size of the screen. Create responsive images by adding an **.img-fluid class** to the <img> tag. The image will then scale nicely to the parent element.



```
<img class="img-fluid rounded-circle mx-auto d-block" src="zaufoundation.png" alt="Foundation Logo">
```

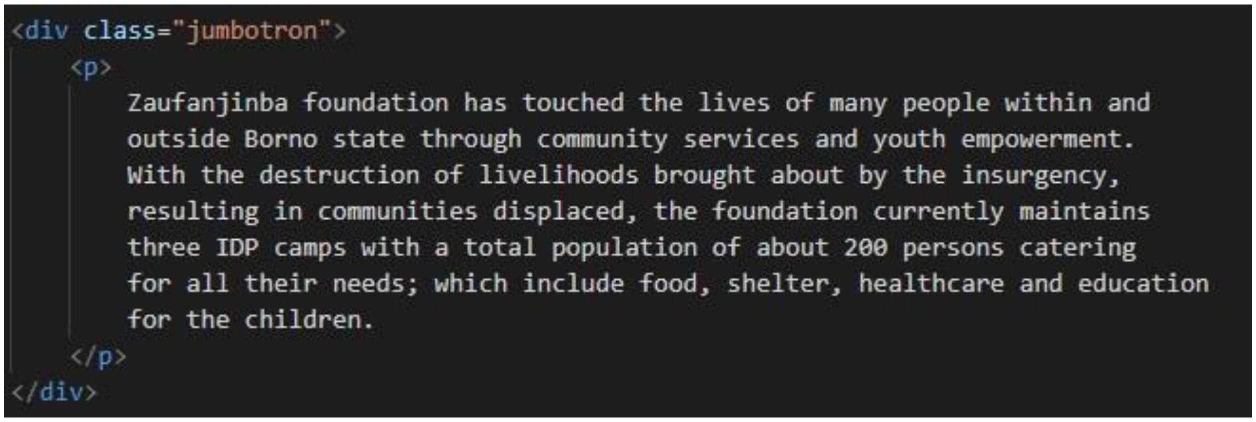
Fig. 3.019 (BS Images: img-fluid class)

The .img-fluid class applies max-width: 100%; and height: auto; to the image

BS Jumbotron

A jumbotron indicates a big grey box for calling extra attention to some special content or information.

Use a <div> element with class .jumbotron to create a jumbotron

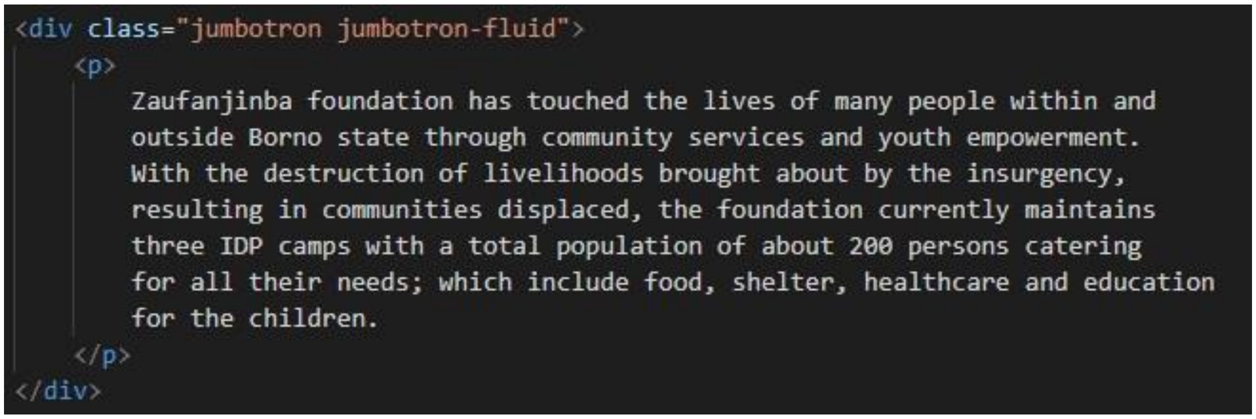


```
<div class="jumbotron">
    <p>
        Zaufanjinba foundation has touched the lives of many people within and
        outside Borno state through community services and youth empowerment.
        With the destruction of livelihoods brought about by the insurgency,
        resulting in communities displaced, the foundation currently maintains
        three IDP camps with a total population of about 200 persons catering
        for all their needs; which include food, shelter, healthcare and education
        for the children.
    </p>
</div>
```

Fig. 3.020 (BS Jumbotron)

Note: Inside a jumbotron you can put nearly any valid HTML, including other Bootstrap elements/classes.

If you want a full-width jumbotron without rounded borders, add the .**jumbotron-fluid class** and a .container or .container-fluid inside of it.



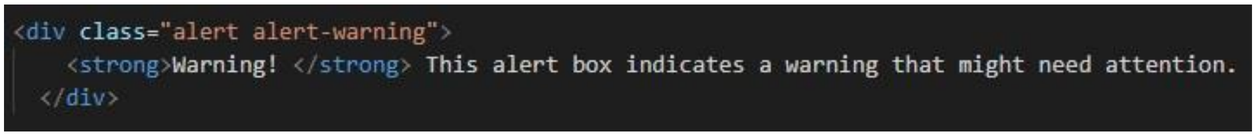
```
<div class="jumbotron jumbotron-fluid">
    <p>
        Zaufanjinba foundation has touched the lives of many people within and
        outside Borno state through community services and youth empowerment.
        With the destruction of livelihoods brought about by the insurgency,
        resulting in communities displaced, the foundation currently maintains
        three IDP camps with a total population of about 200 persons catering
        for all their needs; which include food, shelter, healthcare and education
        for the children.
    </p>
</div>
```

Fig. 3.021 (BS Jumbotron: jumbotron-fluid class)

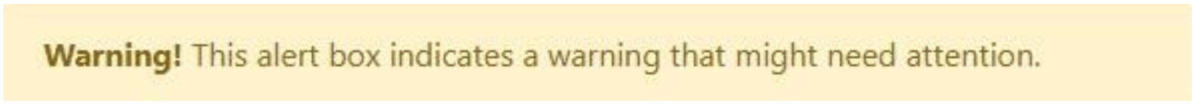
BS Alerts

Bootstrap 4 provides an easy way to create predefined alert messages.

Alerts are created with the **.alert class**, followed by one of the contextual classes .alert-success, .alert-info, .alert-warning, .alert-danger, .alert-primary, .alert-secondary, .alert-light or .alert-dark.



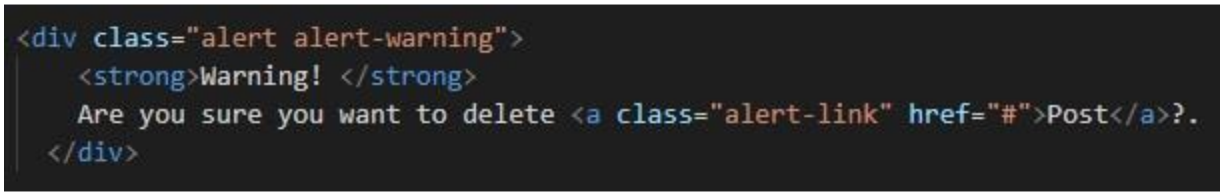
```
<div class="alert alert-warning">
    <strong>Warning! </strong> This alert box indicates a warning that might need attention.
  </div>
```



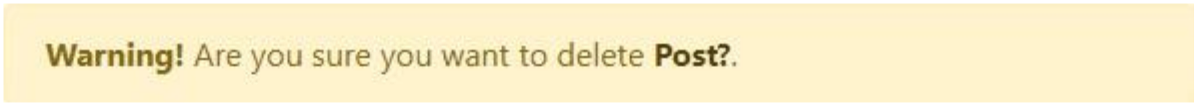
Warning! This alert box indicates a warning that might need attention.

Fig. 3.022 (BS Alerts: alert-warning class)

You can add the **alert-link class** to any links inside the alert box to create "matching colored links".



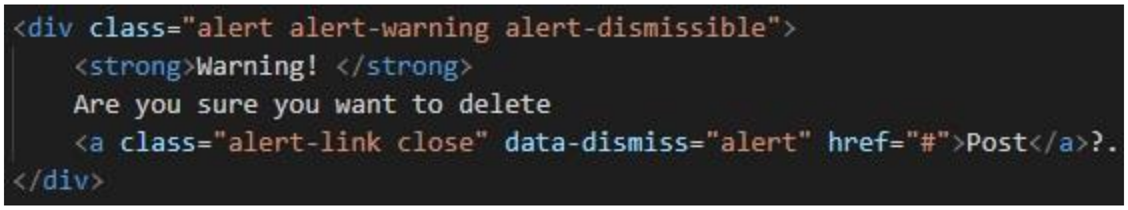
```
<div class="alert alert-warning">
    <strong>Warning! </strong>
    Are you sure you want to delete <a class="alert-link" href="#">Post</a>?.
  </div>
```



Warning! Are you sure you want to delete **Post?**.

Fig. 3.023 (BS Alerts: alert-link)

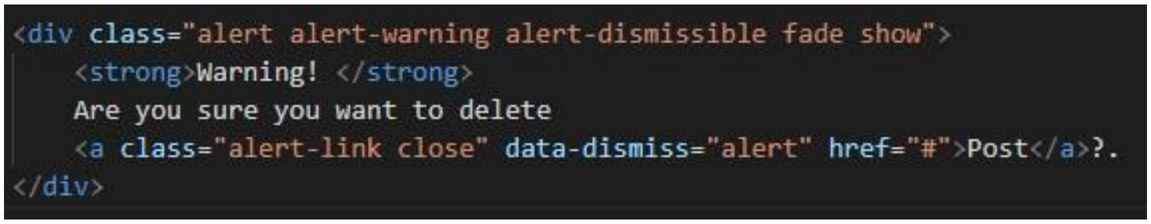
To close the alert message, add an .alert-dismissible class to the alert container. Then add class="close" and data-dismiss="alert" to a link or a button element (when you click on this the alert box will disappear).



```
<div class="alert alert-warning alert-dismissible">
    <strong>Warning! </strong>
    Are you sure you want to delete
    <a class="alert-link close" data-dismiss="alert" href="#">Post</a>?.
</div>
```

Fig. 3.024 (BS Alerts: alert-dismissible class)

You can add an animation when closing the alert box. The .fade and .show classes adds a fading effect when closing the alert message.

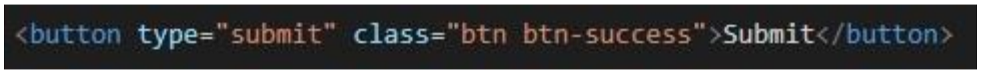


```
<div class="alert alert-warning alert-dismissible fade show">
    <strong>Warning! </strong>
    Are you sure you want to delete
    <a class="alert-link close" data-dismiss="alert" href="#">Post</a>?.
</div>
```

Fig. 3.025 (BS Alerts: fade and show class)

BS Buttons

Buttons are created with the .btn class, followed by one of the contextual classes .btn-success, .btn -info, .btn -warning, .btn -danger, .btn -primary, .btn -secondary, .btn -light or .btn -dark.



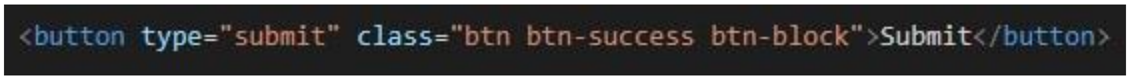
```
<button type="submit" class="btn btn-success">Submit</button>
```

Fig. 3.026 (BS Buttons: btn-success)

Note: The button classes can be used on <a>, <button>, or <input> elements

You can use the **.btn-lg class** for large buttons or **.btn-sm class** for small buttons

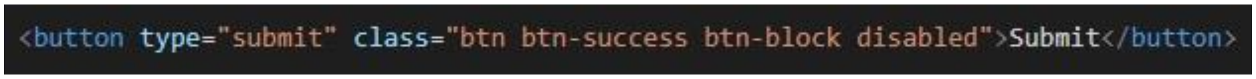
To create a block-level button, add **.btn-block class** to the button, its spans the entire width of the parent element.



```
<button type="submit" class="btn btn-success btn-block">Submit</button>
```

Fig. 3.027 (BS Buttons: btn-block class)

A button can be set to an active (appear pressed) or a disabled (unclickable) state: The class .active makes a button appear pressed, and the disabled attribute makes a button unclickable.



```
<button type="submit" class="btn btn-success btn-block disabled">Submit</button>
```

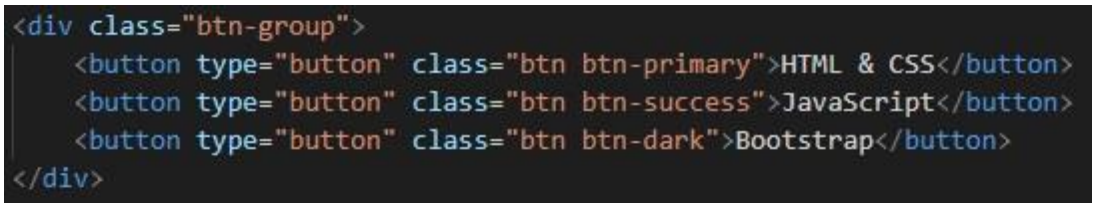
Fig. 3.028 (BS Buttons: disabled class)

Note: <a> elements do not support the disabled attribute and must therefore use the .disabled class to make it visually appear disabled.

BS Button Group

Bootstrap 4 allows you to group a series of buttons together (on a single line) in a button group.

Use a <div> element with class .btn-group to create a button group.



```
<div class="btn-group">
    <button type="button" class="btn btn-primary">HTML & CSS</button>
    <button type="button" class="btn btn-success">JavaScript</button>
    <button type="button" class="btn btn-dark">Bootstrap</button>
</div>
```

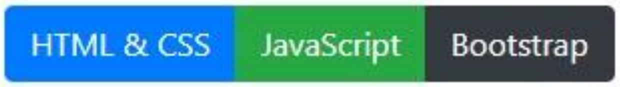
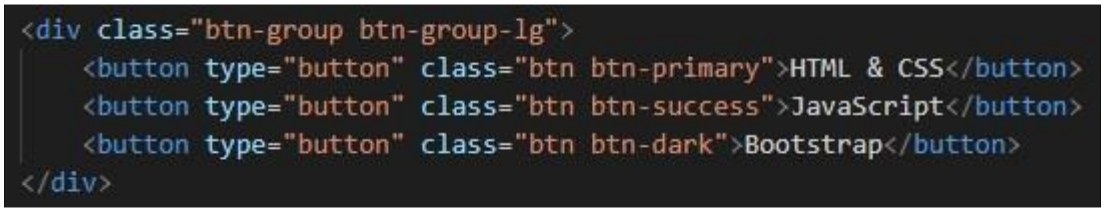


Fig. 3.029 (BS Button Group)

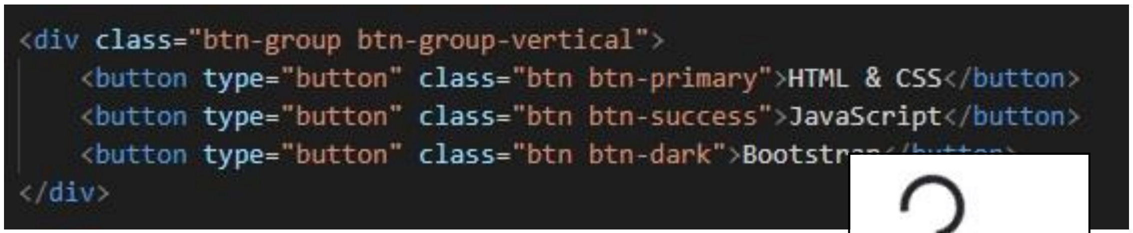
Note: Instead of applying button sizes to every button in a group, use class .btn-group-lg for a large button group or the .btn-group-sm for a small button group.



```
<div class="btn-group btn-group-lg">
    <button type="button" class="btn btn-primary">HTML & CSS</button>
    <button type="button" class="btn btn-success">JavaScript</button>
    <button type="button" class="btn btn-dark">Bootstrap</button>
</div>
```

Fig. 3.030 (BS Button Group: btn-group-lg)

Bootstrap 4 also support vertical button group. Use the class .btn-group-vertical to create a vertical button group.



```
<div class="btn-group btn-group-vertical">
    <button type="button" class="btn btn-primary">HTML & CSS</button>
    <button type="button" class="btn btn-success">JavaScript</button>
    <button type="button" class="btn btn-dark">Bootstr
</div>
```

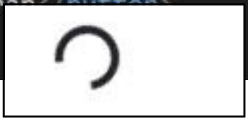
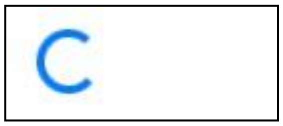
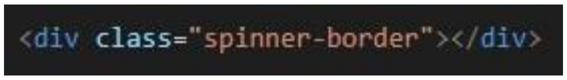


Fig. 3.031 (BS Button Group: btn-group-vertical)

BS Spinners

To create a spinner/loader, use the .**spinner-border class**.

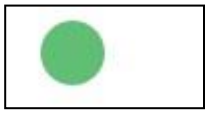


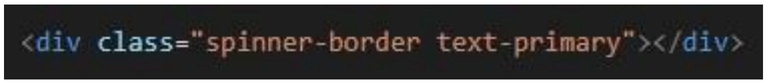


```
<div class="spinner-border"></div>
```

Fig. 3.032 (BS Spinners)

You can use any of the text color utilities to add a color to the spinner.

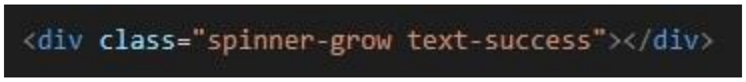




```
<div class="spinner-border text-primary"></div>
```

Fig. 3.033 (BS Spinners: adding color to a spinner)

You can use the .**spinner-grow class** if you want the spinner/loader to grow instead of "spin".

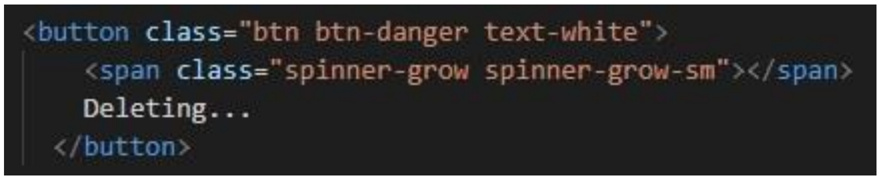


```
<div class="spinner-grow text-success"></div>
```

Fig. 3.034 (BS Spinners: spinner-grow class)

You can create smaller spinners using the .**spinner-border-sm** or .**spinner-grow-sm**.

You can also add spinners to a button, with or without text.



```
<button class="btn btn-danger text-white">
    <span class="spinner-grow spinner-grow-sm"></span>
    Deleting...
  </button>
```

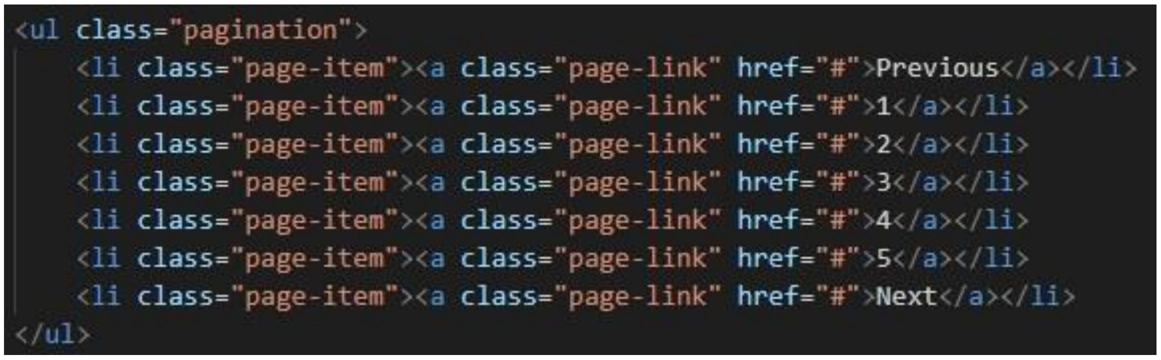
Fig. 3.035 (BS Spinners: adding spinner to a button)

BS Pagination

If you have a web site with lots of pages, you may wish to add some sort of pagination to each page or you are query the records of a table from your database, you can use pagination to display to load a certain number of records on a page. This will improve your website performance and loading speed.

To create a basic pagination, add the .pagination class to an <ul> element.

Then add the .page-item to each <li> element and a .page-link class to each link inside <li>:



```
<ul class="pagination">
    <li class="page-item"><a class="page-link" href="#">Previous</a></li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item"><a class="page-link" href="#">4</a></li>
    <li class="page-item"><a class="page-link" href="#">5</a></li>
    <li class="page-item"><a class="page-link" href="#">Next</a></li>
</ul>
```

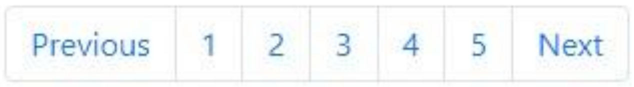
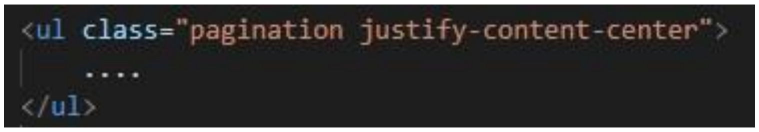


Fig. 3.036 (BS Pagination)

Use the .active class to "highlight" the current page. To disable a link or make it unclickable, use the .disabled class.

Note: You can use utility classes to change the alignment of the pagination. Like the .justify-content-center class to center the pagination or .justify-content-end class to move the pagination to the right.



```
<ul class="pagination justify-content-center">
    ....
</ul>
```

Fig. 3.037 (BS pagination: centered pagination)

BS Cards

A card in Bootstrap 4 is a bordered box with some padding around its content. It includes options for headers, footers, content, colors, etc.

A basic card is created with the .card class, and content inside the card has a .card-body class.

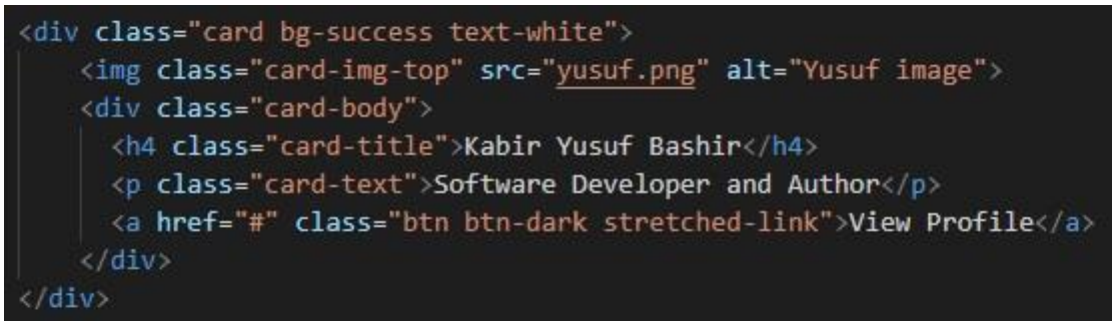
To add a header to the card, use the .card-header class and the .card-footer class to add footer to the card.

You can add a background color to the card, use contextual classes (.bg-primary, .bg-success, .bg-info, .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light.

You can add .card-img-top or .card-img-bottom to an <img> to place the image at the top or at the bottom inside the card.

You can add the .stretched-link class to a link inside the card, and it will make the whole card clickable and hoverable (the card will act as a link).

You can turn an image into a card background and use .card-img-overlay to add text on top of the image.



```
<div class="card bg-success text-white">
    <img class="card-img-top" src="yusuf.png" alt="Yusuf image">
    <div class="card-body">
      <h4 class="card-title">Kabir Yusuf Bashir</h4>
      <p class="card-text">Software Developer and Author</p>
      <a href="#" class="btn btn-dark stretched-link">View Profile</a>
    </div>
</div>
```

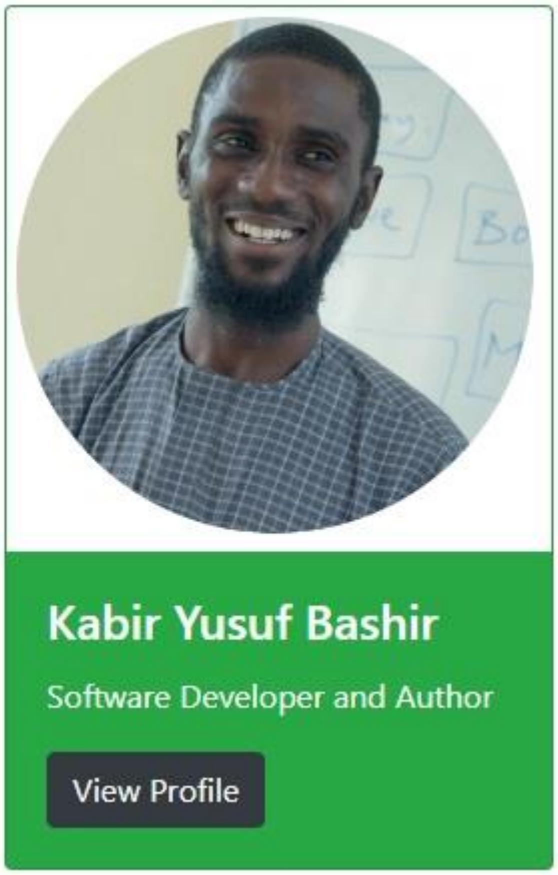


Fig. 3.038 (BS Cards)

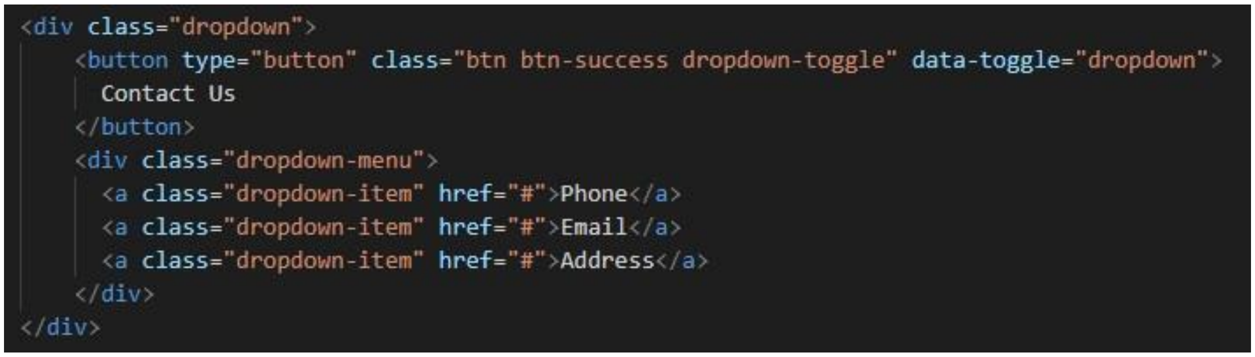
BS Dropdown

A dropdown menu is a toggleable menu that allows the user to choose one value from a predefined list. Dropdowns are useful when creating your website navigation.

The .dropdown class indicates a dropdown menu.

To open the dropdown menu, use a button or a link with a class of .dropdown-toggle and the data-toggle="dropdown" attribute.

Add the .dropdown-menu class to a <div> element to actually build the dropdown menu. Then add the .dropdown-item class to each element (links or buttons) inside the dropdown menu as shown below in *fig. 3.039*:



```
<div class="dropdown">
    <button type="button" class="btn btn-success dropdown-toggle" data-toggle="dropdown">
      Contact Us
    </button>
    <div class="dropdown-menu">
      <a class="dropdown-item" href="#">Phone</a>
      <a class="dropdown-item" href="#">Email</a>
      <a class="dropdown-item" href="#">Address</a>
    </div>
</div>
```

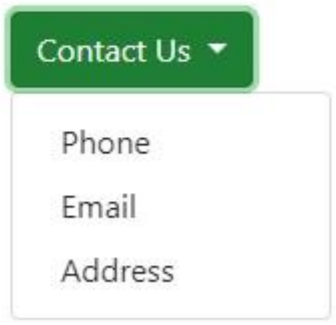
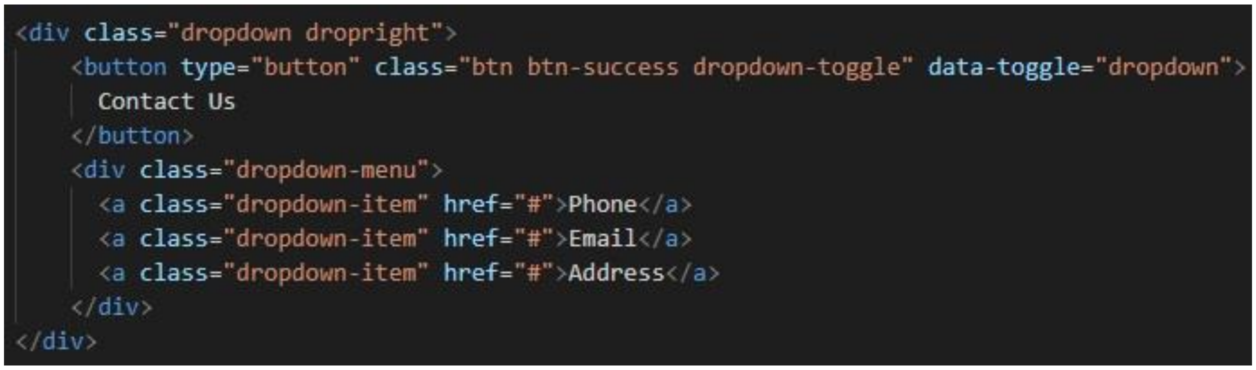


Fig. 3.039 (BS Dropdown)

You can also create a "dropright" or "dropleft" menu, by adding the .dropright or .dropleft class to the dropdown element. Note that the caret/arrow is added automatically.



```html
<div class="dropdown dropright">
    <button type="button" class="btn btn-success dropdown-toggle" data-toggle="dropdown">
      Contact Us
    </button>
    <div class="dropdown-menu">
      <a class="dropdown-item" href="#">Phone</a>
      <a class="dropdown-item" href="#">Email</a>
      <a class="dropdown-item" href="#">Address</a>
    </div>
</div>
```

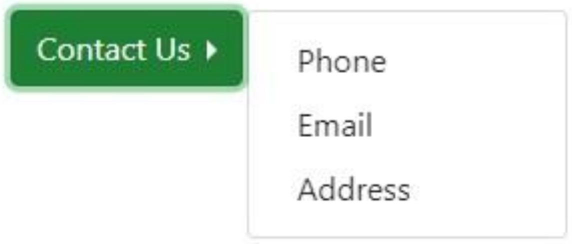
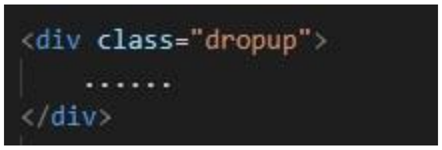


Fig. 3.040 (BS Dropdown: dropright class)

Note: If you want the dropdown menu to expand upwards instead of downwards, change the <div> element with class="dropdown" to "dropup"

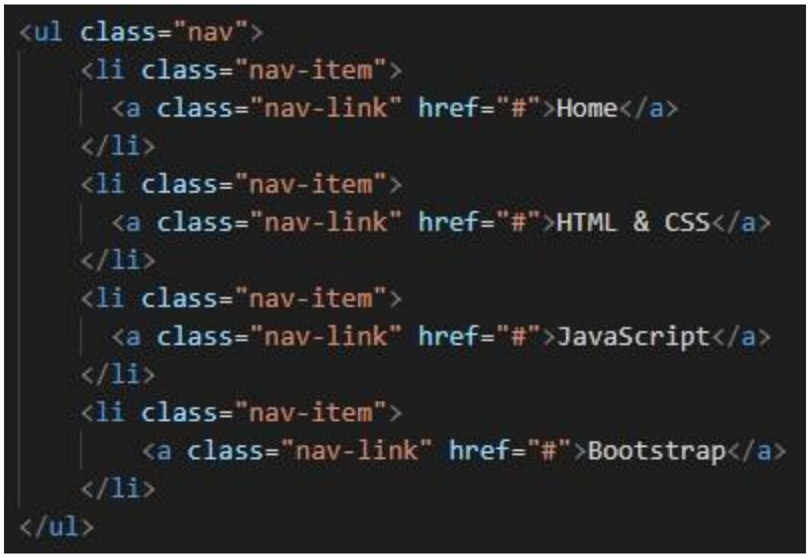


```html
<div class="dropup">
    ......
</div>
```

Fig. 3.041 (BS Dropdown: dropup class)

BS Navs

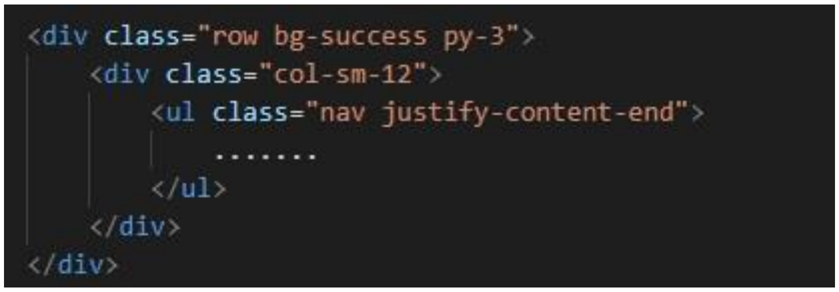
To create a simple horizontal menu, add the .nav class to a <ul> element, followed by .nav-item for each <li> and add the .nav-link class to their links.



```
<ul class="nav">
    <li class="nav-item">
      <a class="nav-link" href="#">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">HTML & CSS</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">JavaScript</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">Bootstrap</a>
    </li>
</ul>
```

Fig. 3.042 (BS Nav)

Normally by default, the nav is aligned to the left. You can align the nav to the center using the .justify-content-center class or to the right using the .justify-content-end class.



```
<div class="row bg-success py-3">
    <div class="col-sm-12">
        <ul class="nav justify-content-end">
            .......
        </ul>
    </div>
</div>
```

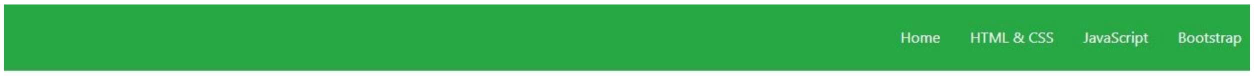
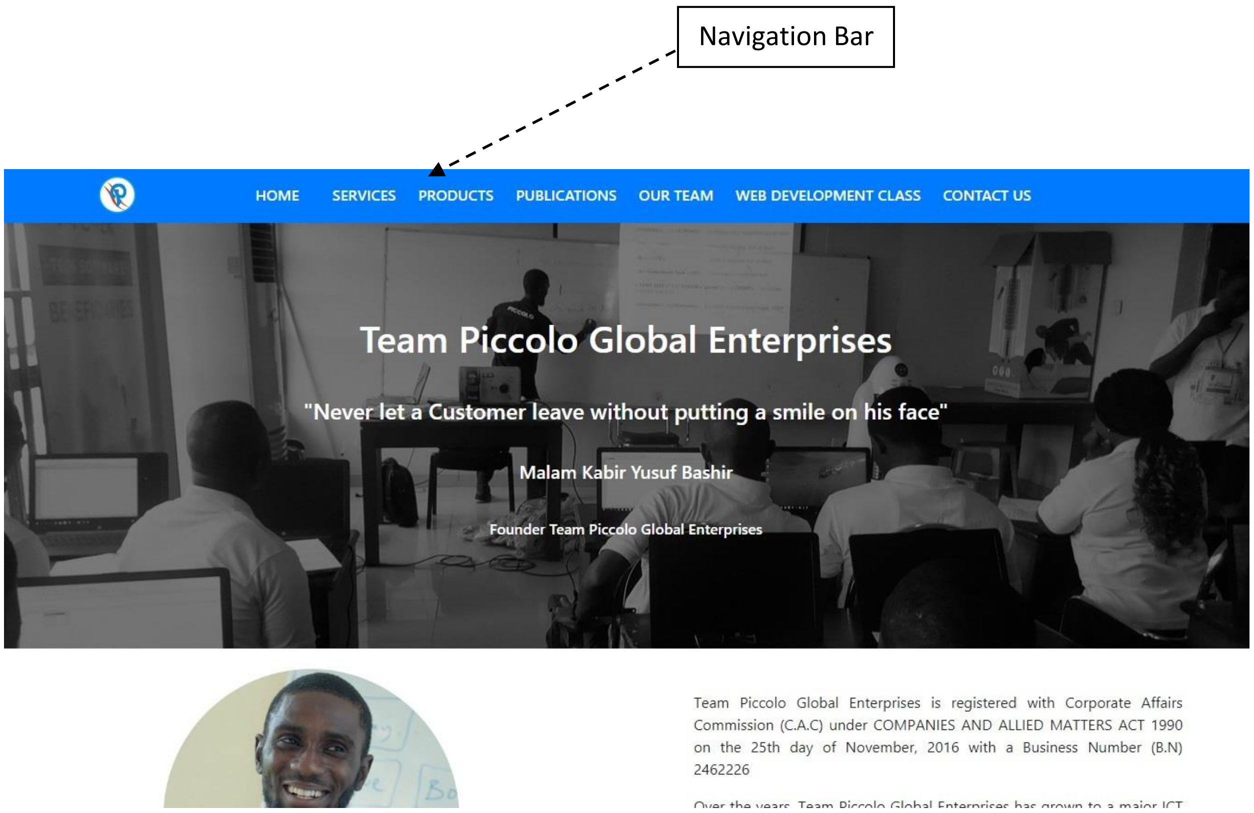


Fig. 3.043 (BS Nav: aligning nav to the right)

BS Navigation Bar

A navigation bar is a navigation header that is placed at the top of the page.



Team Piccolo Global Enterprises

"Never let a Customer leave without putting a smile on his face"

Malam Kabir Yusuf Bashir

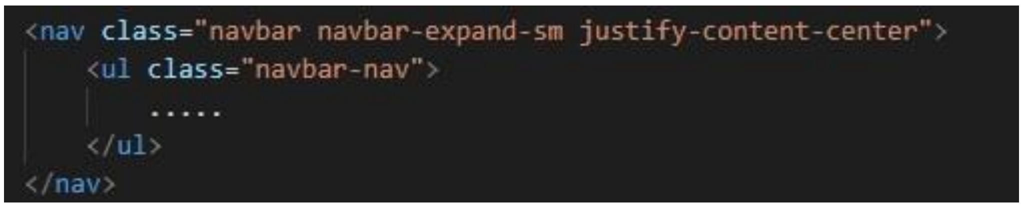
Founder Team Piccolo Global Enterprises

Team Piccolo Global Enterprises is registered with Corporate Affairs Commission (C.A.C) under COMPANIES AND ALLIED MATTERS ACT 1990 on the 25th day of November, 2016 with a Business Number (B.N) 2462226

Over the years Team Piccolo Global Enterprises has grown to a major ICT

A standard navigation bar is created with the .navbar class, followed by a responsive collapsing class: .navbar-expand-xl|lg|md|sm (stacks the navbar vertically on extra-large, large, medium or small screens).

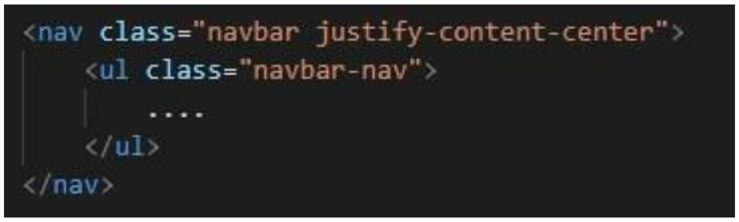
To add links inside the navbar, use a <ul> element with class="navbar-nav". Then add <li> elements with a .nav-item class followed by an <a> element with a .nav-link class



```
<nav class="navbar navbar-expand-sm justify-content-center">
    <ul class="navbar-nav">
        .....
    </ul>
</nav>
```

Fig. 3.045 (BS NavBar)

If you want a vertical navigation bar, remove the .navbar-expand-xl|lg|md|sm class.



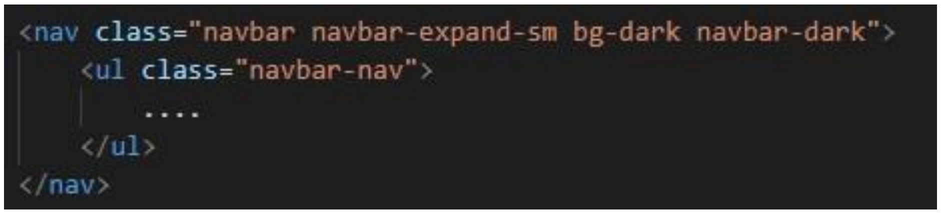
```
<nav class="navbar justify-content-center">
    <ul class="navbar-nav">
        ....
    </ul>
</nav>
```

Fig. 3.046 (BS NavBar: vertical navigation bar)

To center the navigation bar, use the .justify-content-center class as in *fig. 3.046* above:

Use any of the .bg-color classes to change the background color of the navbar (.bg-primary, .bg-success, .bg-info, .bg-warning, .bg-danger, .bg-secondary, .bg-dark and .bg-light)

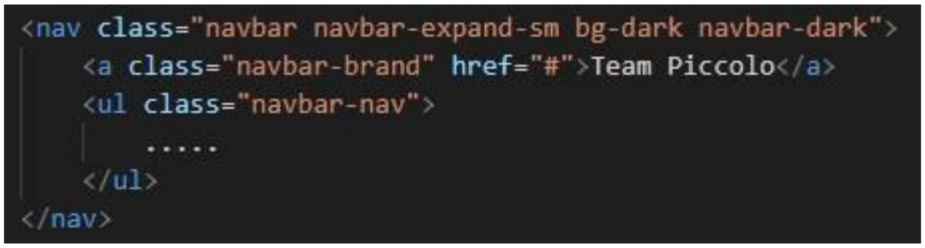
Tip: Add a **white** text color to all links in the navbar with the .navbar-dark class, or use the .navbar-light class to add a **black** text color.



```
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <ul class="navbar-nav">
        ....
    </ul>
</nav>
```

Fig. 3.047 (BS NavBar: navbar-dark class)

You can add your company name, brand, project name or logo to the navigation bar using the .navbar-brand class.

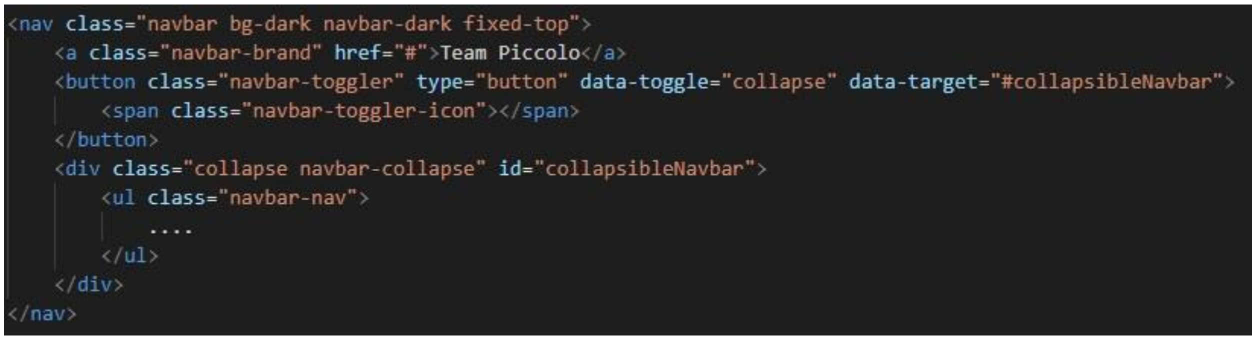


```
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <a class="navbar-brand" href="#">Team Piccolo</a>
    <ul class="navbar-nav">
        .....
    </ul>
</nav>
```

Fig. 3.048 (BS NavBar: navbar-brand class)

Very often, especially on small screens, you want to hide the navigation links and replace them with a button that should reveal them when clicked on.

To create a collapsible navigation bar, use a button with class="navbar-toggler", data-toggle="collapse" and data-target="#*thetarget*". Then wrap the navbar content (links, etc.) inside a div element with class="collapse navbar-collapse", followed by an id that matches the data-target of the button: "*thetarget*".



```
<nav class="navbar bg-dark navbar-dark fixed-top">
    <a class="navbar-brand" href="#">Team Piccolo</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#collapsibleNavbar">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="collapsibleNavbar">
        <ul class="navbar-nav">
            ....
        </ul>
    </div>
</nav>
```

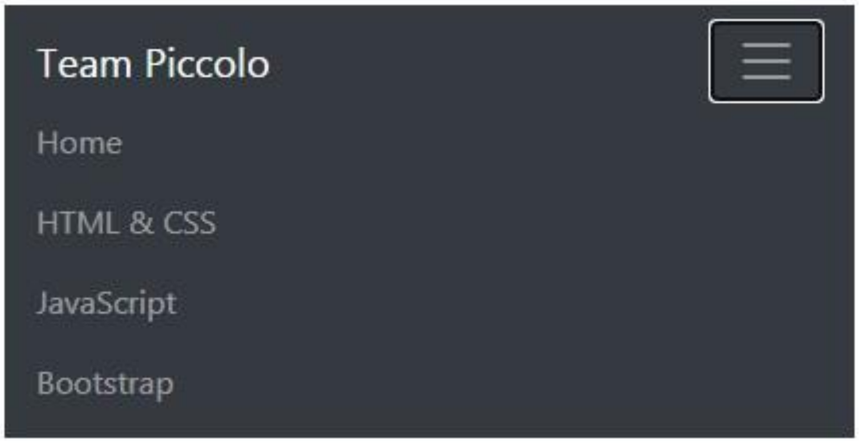


Fig. 3.049 (BS NavBar: collapsible navbar)

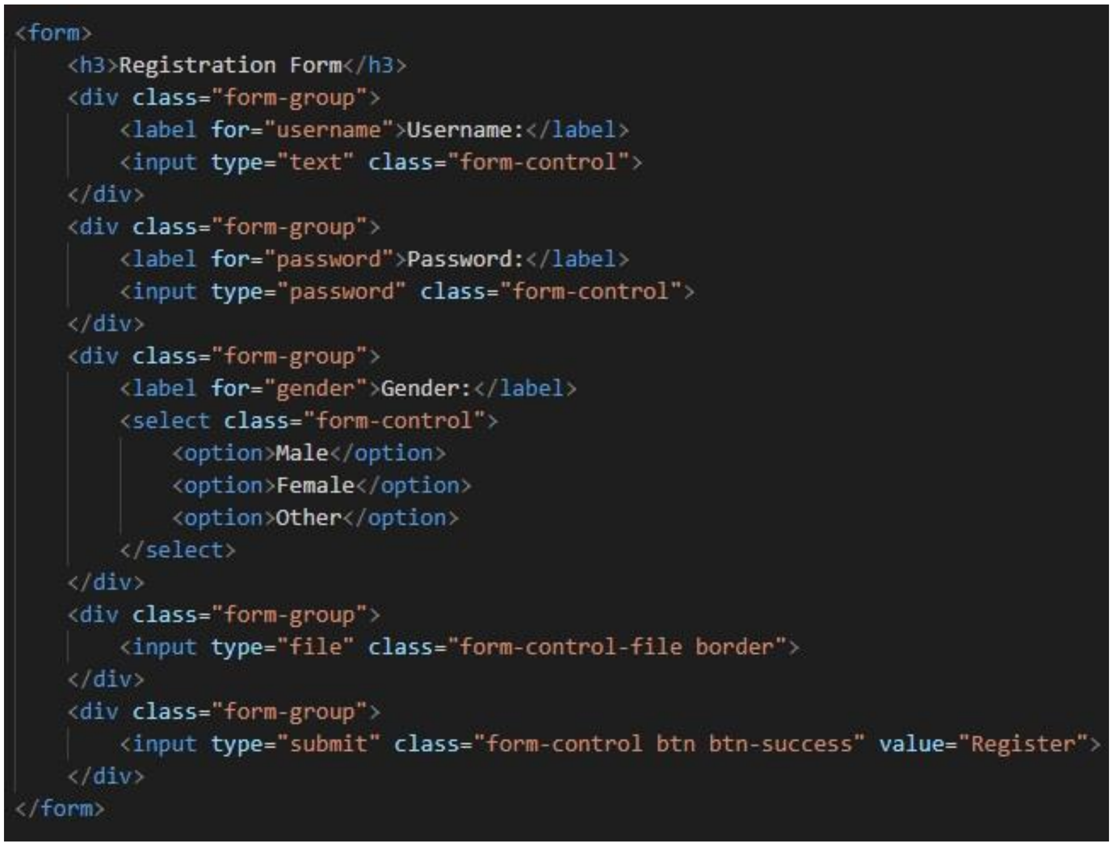
The navigation bar can also be fixed at the top or at the bottom of the page as shown above in *fig. 3.049*, the navigation bar is fixed at the top of the page. The .fixed-top class makes the navigation bar fixed at the **top** and .fixed-bottom class to make the navbar stay at the **bottom** of the page. A fixed navigation bar stays visible in a fixed position (top or bottom) independent of the page scroll.

BS Forms

Bootstrap provides two types of form layouts: Stacked (full-width) form and Inline form. Form controls automatically receive some global styling with Bootstrap

All textual <input>, <textarea>, and <select> elements with class .form-control have a width of 100%.

Bootstrap supports the following form controls: input, textarea, checkbox, radio and select. Bootstrap supports all the HTML5 input types: text, password, datetime, datetime-local, date, month, time, week, number, email, url, search, tel, and color.



```
<form>
    <h3>Registration Form</h3>
    <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" class="form-control">
    </div>
    <div class="form-group">
        <label for="gender">Gender:</label>
        <select class="form-control">
            <option>Male</option>
            <option>Female</option>
            <option>Other</option>
        </select>
    </div>
    <div class="form-group">
        <input type="file" class="form-control-file border">
    </div>
    <div class="form-group">
        <input type="submit" class="form-control btn btn-success" value="Register">
    </div>
</form>
```

Fig. 3.050 (BS Forms)

The .form-group class in *fig. 3.050* above, is used to wrapper elements together, this ensure proper margins.

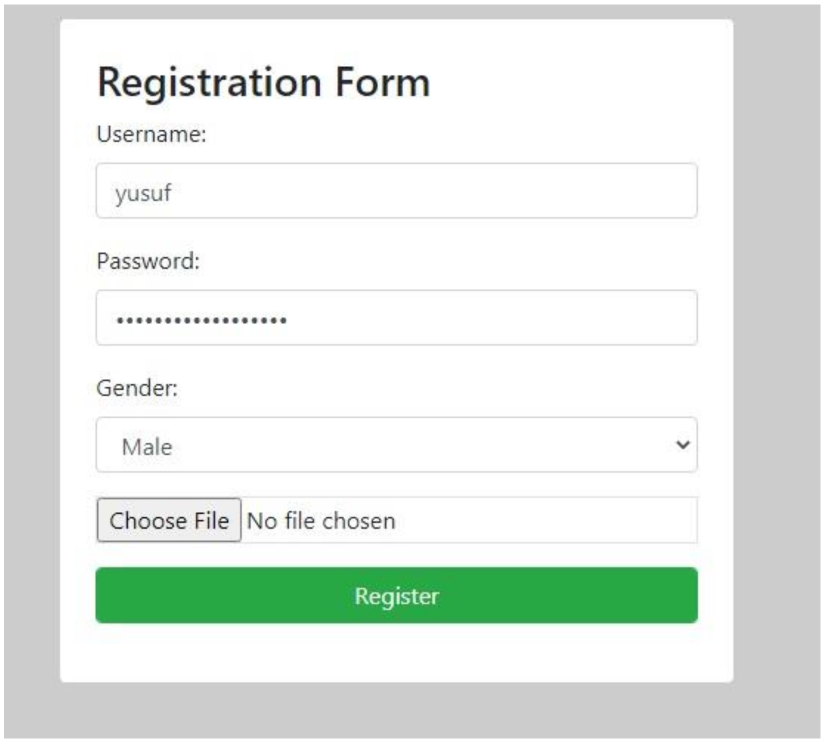
The code in *fig. 3.50* above, contains four input elements and one select element; one of type="text", one of type="password", one of type="file" and of type="submit".

The .form-control class styles inputs with full-width and proper padding, etc.

Note: Inputs will NOT be fully styled if their type is not properly declared!

You can change the size of the form control with .form-control-sm or .form-control-lg.

You can add the .form-control-range class to input type="range" or .form-control-file to input type="file" to style a range control or a file field with full-width.



Registration Form

Username:

yusuf

Password:

••••••••••••••••••

Gender:

Male

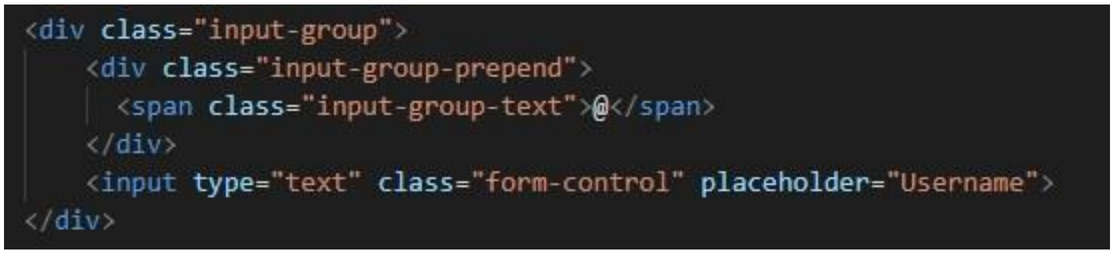
Fig. 3.051 (BS Forms: a simple registration form)

BS Inputs Group

The .input-group class is a container to enhance an input by adding an icon, text or a button in front or behind the input field as a "help text".

Use .input-group-prepend to add the help text in front of the input, and .input-group-append to add it behind the input.

At last, add the .input-group-text class to style the specified help text.



```
<div class="input-group">
    <div class="input-group-prepend">
      <span class="input-group-text">@</span>
    </div>
    <input type="text" class="form-control" placeholder="Username">
</div>
```

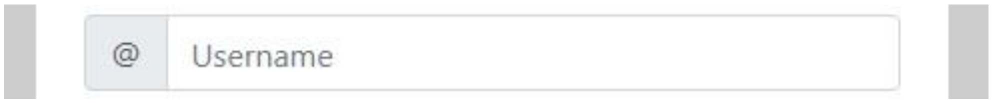
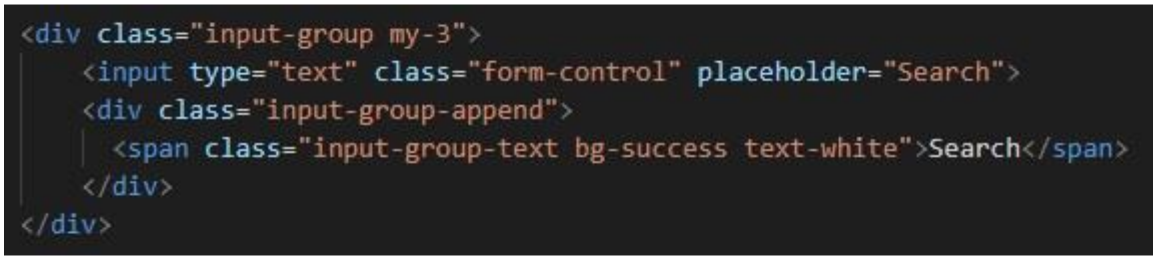


Fig. 3.052 (BS Input Group: input-group-prepend)



```
<div class="input-group my-3">
    <input type="text" class="form-control" placeholder="Search">
    <div class="input-group-append">
      <span class="input-group-text bg-success text-white">Search</span>
    </div>
</div>
```

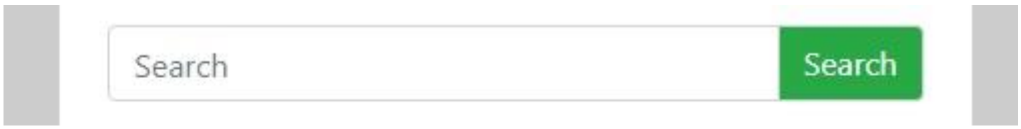


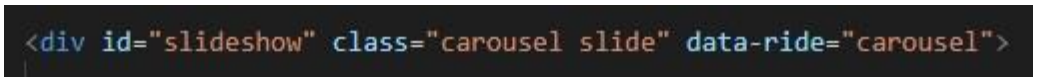
Fig. 3.052 (BS Input Group: input-group-append)

BS Carousel

To create a slideshow on your web page, you can easily use the bootstrap carousel. It is easy to create and use on your web page. A Carousel is a slideshow for cycling through elements.

To create a carousel on your web page using bootstrap, follow the steps below:

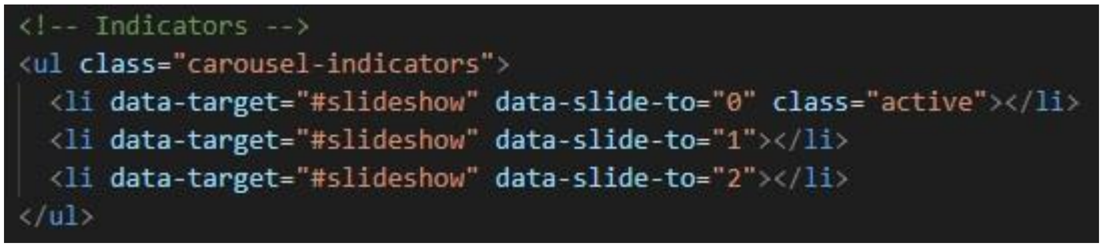
- Create a <div> with the class of .carousel and a class of slide. Give the <div> an id and an attribute of data-ride="carousel". In this example the value of our is "slideshow"



```
<div id="slideshow" class="carousel slide" data-ride="carousel">
```

Fig. 3.053 (BS Carousel: carousel class)

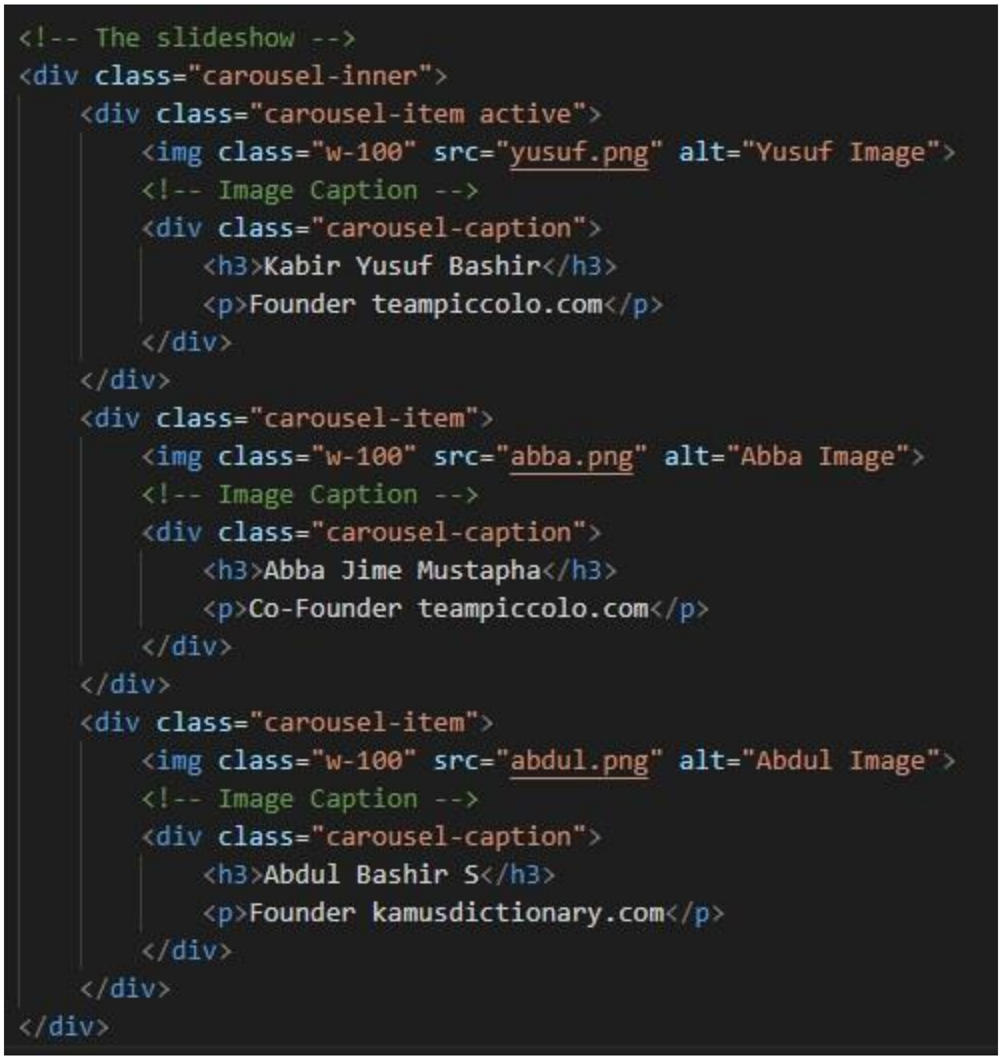
- Next is to create an **indicator** inside the <div> you created in *fig. 3.053* above. Create unordered list using the HTML <ul> element with a class of carousel-indicators. This create little dots at the bottom of each slide (which indicates how many slides there are in the carousel, and which slide the user are currently viewing). Inside the <ul> element, create a <li> element depending on the number of slides on your page. In this example we have three slides on the page so we will be needing three <li>s inside the <ul>. Give each of the <li> elements an attribute of data-target with the value of the id you given your <div> in *fig. 3.053* above and an attribute of data-slide-to with the value of the sequence of your slides starting with 0. In this example the data-target="#slideshow".



```
<!-- Indicators -->
<ul class="carousel-indicators">
  <li data-target="#slideshow" data-slide-to="0" class="active"></li>
  <li data-target="#slideshow" data-slide-to="1"></li>
  <li data-target="#slideshow" data-slide-to="2"></li>
</ul>
```

Fig. 3.054 (BS Carousel: carousel-indicators class)

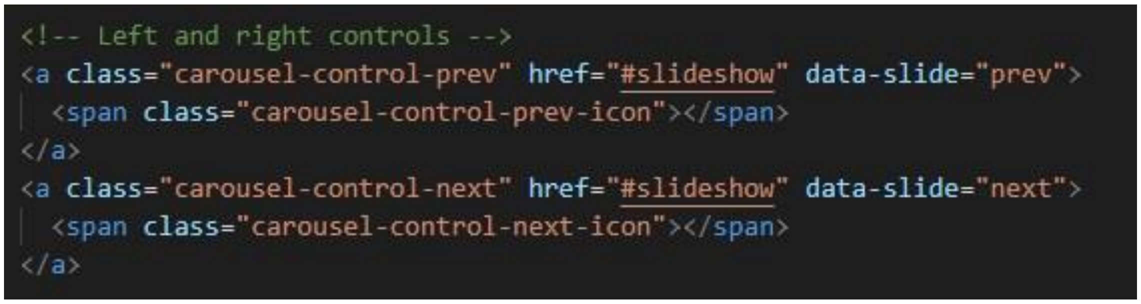
- Next create a <div> with a class of .carousel-inner. Inside the <div>, we will add our slides to the carousel. To add slides to the carousel, create a <div> with the class of .carousel-item. The <div> specifies the content of each slide. Use an <img> element to insert an image to the slide. If the slide has a **caption,** create a <div> with class of .carousel-caption inside the <div> with the class .carousel-item.



```html
<!-- The slideshow -->
<div class="carousel-inner">
    <div class="carousel-item active">
        <img class="w-100" src="yusuf.png" alt="Yusuf Image">
        <!-- Image Caption -->
        <div class="carousel-caption">
            <h3>Kabir Yusuf Bashir</h3>
            <p>Founder teampiccolo.com</p>
        </div>
    </div>
    <div class="carousel-item">
        <img class="w-100" src="abba.png" alt="Abba Image">
        <!-- Image Caption -->
        <div class="carousel-caption">
            <h3>Abba Jime Mustapha</h3>
            <p>Co-Founder teampiccolo.com</p>
        </div>
    </div>
    <div class="carousel-item">
        <img class="w-100" src="abdul.png" alt="Abdul Image">
        <!-- Image Caption -->
        <div class="carousel-caption">
            <h3>Abdul Bashir S</h3>
            <p>Founder kamusdictionary.com</p>
        </div>
    </div>
</div>
```

Fig. 3.055 (BS Carousel: carousel-item and carousel-caption class)

- Lastly is our controls. We will need two controls, previous and next. We will use an <a> element. Create an <a> element with a class of .carousel-control-prev, give it an attribute of data-slide=”prev” and a href=”#slideshow”. Follow same procedure to create the **next** control as shown below in *fig. 3.056*:



```
<!-- Left and right controls -->
<a class="carousel-control-prev" href="#slideshow" data-slide="prev">
  <span class="carousel-control-prev-icon"></span>
</a>
<a class="carousel-control-next" href="#slideshow" data-slide="next">
  <span class="carousel-control-next-icon"></span>
</a>
```

Fig. 3.056 (BS carousel: carousel-control-prev and carousel-control-next)

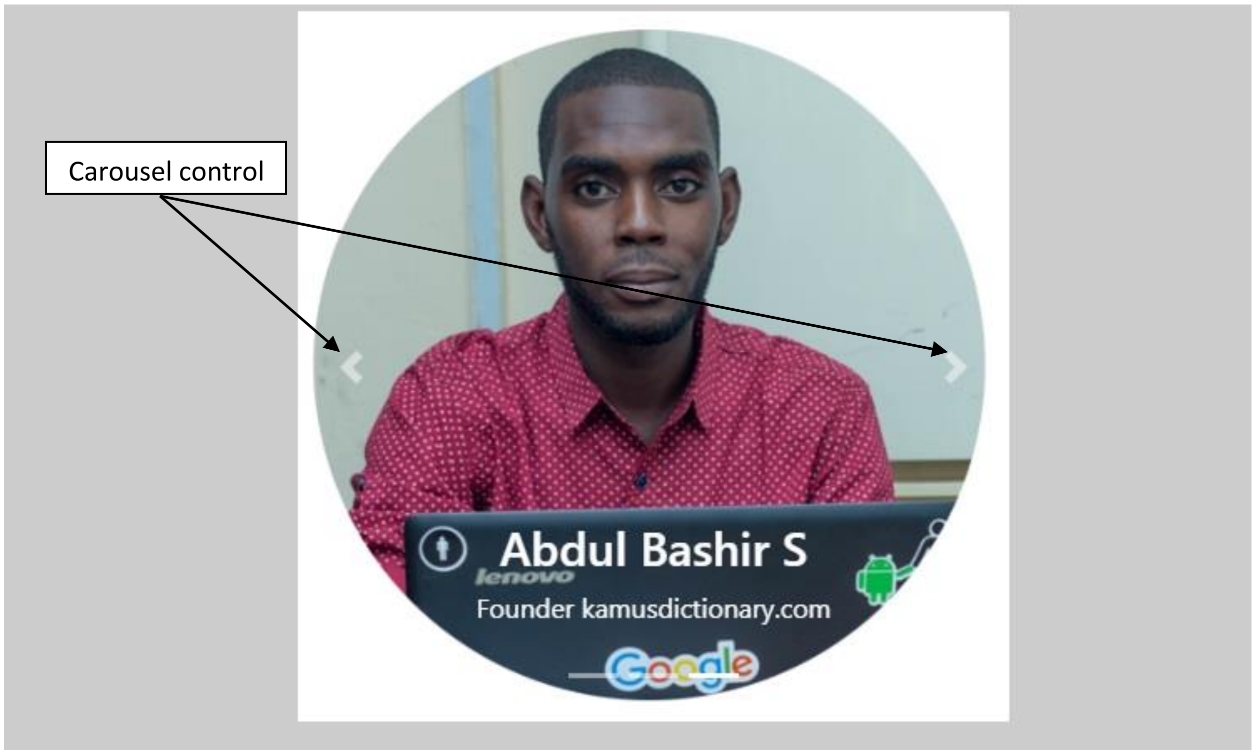


Fig. 3.057 (BS carousel: slideshow)

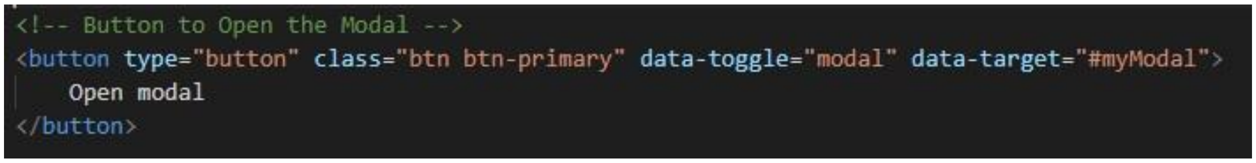
Foot Note: *The .slide class in fig. 3.053 above adds a CSS transition and animation effect when sliding from one item to the next. You can remove this class if you do not want this effect.*

BS Modal

A Modal component is a dialog box/popup window that is displayed on top of the current page.

To create a modal in bootstrap, follow the steps below:

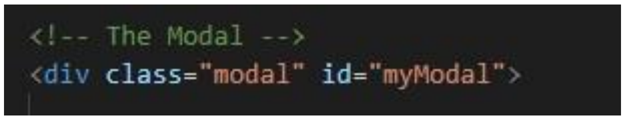
- Create a <button> with attributes of data-toggle=”modal” and data-target=”#myModal”. The data-target attribute value should be unique as you will be referencing it in your modal. You can style the <button> with the classes you wished to apply to the <button>.



```
<!-- Button to Open the Modal -->
<button type="button" class="btn btn-primary" data-toggle="modal" data-target="#myModal">
    Open modal
</button>
```

Fig. 3.058 (BS Modal: creating button)

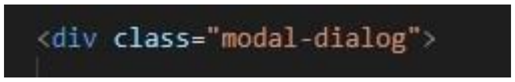
- Next is to create a <div> with class of .modal and an id attribute of **myModal**. The value of your id should be same of the value of the data-target in *fig. 3.058* above:



```
<!-- The Modal -->
<div class="modal" id="myModal">
```

Fig. 3.059 (BS Modal: creating the modal div)

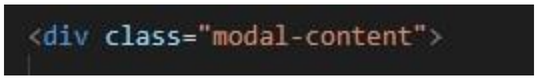
- Inside the <div> we created in *fig. 3.059*, create a <div> with the class .modal-dialog. This will be our dialog, it will contain the modal content.



```
<div class="modal-dialog">
```

Fig. 3.060 (BS Modal: creating modal dialog)

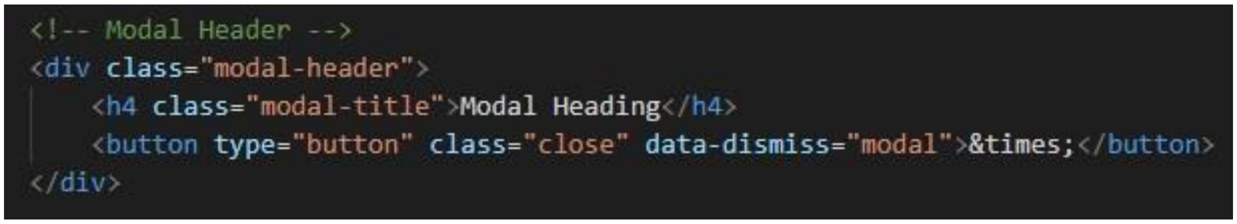
- Inside the modal-dialog, let's create a <div> with the class of .modal-content. Inside the modal-content <div>, we will create three <div>s; one for modal header, modal body and modal footer.



```
<div class="modal-content">
```

Fig. 3.061 (BS Modal: creating modal content)

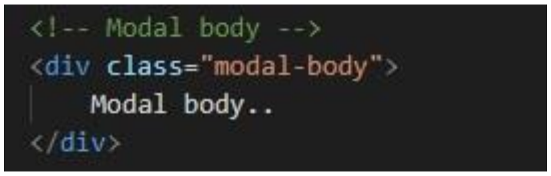
- Let's start by creating the modal header. Create a <div> with class of .modal-header. Inside the <div>, you can use any of the heading element in HTML for the modal header. Then create a <button> with class of .close and an attribute of data-dismiss="modal". The button is used to close the modal.



```
<!-- Modal Header -->
<div class="modal-header">
    <h4 class="modal-title">Modal Heading</h4>
    <button type="button" class="close" data-dismiss="modal">&times;</button>
</div>
```

Fig. 3.062 (BS Modal: creating modal header)

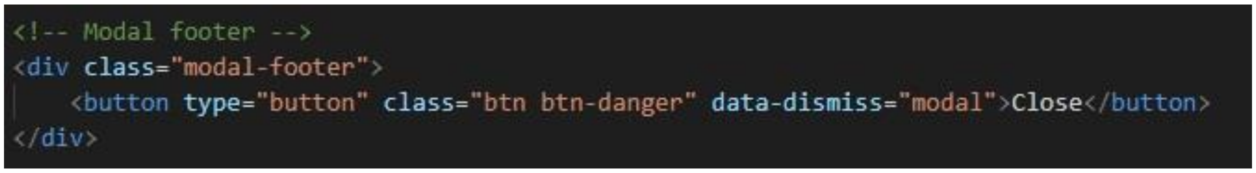
- After the modal header, next is the modal body. Create a <div> with the class of .modal-body. You can put any valid HTML element inside the <div>.



```
<!-- Modal body -->
<div class="modal-body">
    Modal body..
</div>
```

Fig. 3.063 (BS Modal: creating modal body)

- Lastly is the modal footer. Create a <div> with class of .modal-footer. Create a <button> just like the <button> we created inside the modal header in *fig. 3.062* above:



```
<!-- Modal footer -->
<div class="modal-footer">
    <button type="button" class="btn btn-danger" data-dismiss="modal">Close</button>
</div>
```

Fig. 3.064 (BS Modal: creating modal footer)

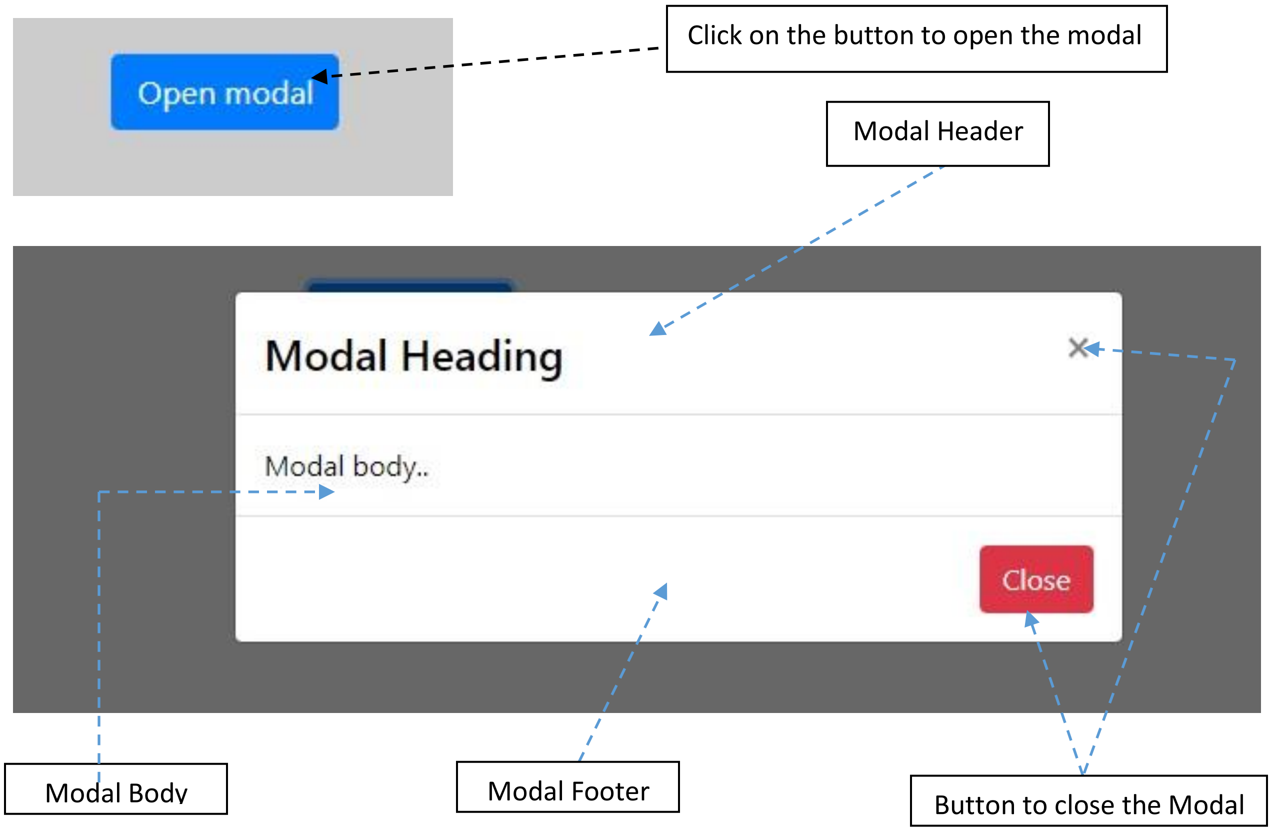


Fig. 3.065 (BS Modal: BS Modal)

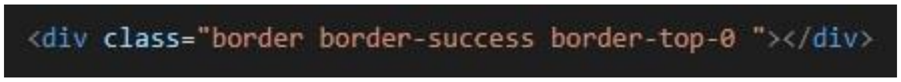
Foot Note:

- *You can use the .fade class to add a fading effect when opening and closing the modal.*
- *You can also add .modal-dialog-scrollable to .modal-dialog to make the dialog scrollable. This will only make the modal scrollable, instead of the page itself.*

BS Utilities

Bootstrap 4 has a lot of utility/helper classes to quickly style elements without using any CSS code. Let’s learn some of Bootstrap utilities:

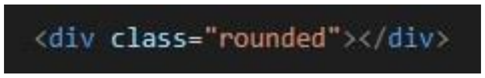
1. The border classes to add or remove borders from an element. You can add a color to the border with any of the contextual border color classes like .border-success for green or .border-primary for blue color. For example if you don’t want border at the top of a box, you can add a class .border-top-0 to remove the border-top or .border-left-0 to remove the border-left.



```
<div class="border border-success border-top-0 "></div>
```

Fig. 3.066 (BS utilities: border class)

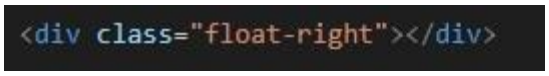
2. The rounded classes is used to make a box or div rounded or create a border radius to the <div>. You can use either of these classes: .rounded-lg, .rounded-top, .rounded-left, .rounded-right, .rounded-bottom, .rounded-circle, .rounded-0 and .rounded-sm.



```
<div class="rounded"></div>
```

Fig. 3.067 (BS utilities: rounded class)

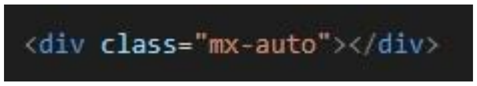
3. You can float an element to the right with the .float-right class or to the left with .float-left, and clear floats with the .clearfix class.



```
<div class="float-right"></div>
```

Fig. 3.068 (BS utilities: float-right class)

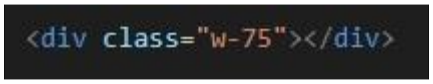
4. You can center an element with the .mx-auto class (adds margin-left and margin-right: auto)



```
<div class="mx-auto"></div>
```

Fig. 3.069 (BS utilities: mx-auto class)

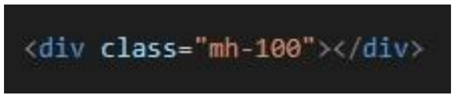
5. You can set the width of an element with the w-* classes (.w-25, .w-50, .w-75, .w-100, .mw-100)



```
<div class="w-75"></div>
```

Fig. 3.070 (BS utilities: w-75 class)

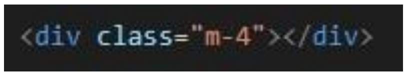
6. You can set the height of an element with the h-* classes (.h-25, .h-50, .h-75, .h-100, .mh-100)



```
<div class="mh-100"></div>
```

Fig. 3.071 (BS utilities: mh-100 class)

7. You can create spacing using either padding or margin. You can set the padding of an element with the p-* classes (.p-0, .p-1, .p-2, .p-3, .p-4, .p-5, .p-auto). You can specific the padding either on the top, left, right or bottom. For example .pt-3 adds a padding-top with a size of three and .pr-1 adds a padding-right with a size of one. To specific a margin, change the **p** to **m** as in example above. For example .m-4 adds a margin on all the sides (top, left, right and bottom) with a size of four and .ml-3 adds a margin-left with a size of three.



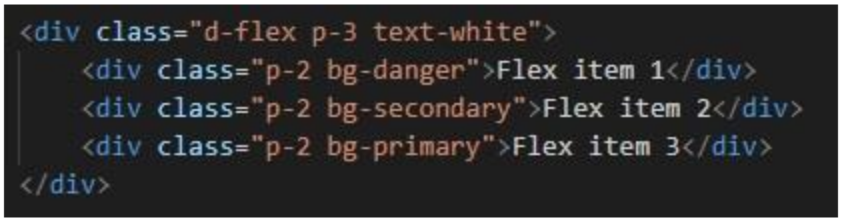
```
<div class="m-4"></div>
```

Fig. 3.072 (BS utilities: m-4 class)

BS Flex

Bootstrap flex classes are used to control the layout of Bootstrap 4 components. Flex Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

To create a flexbox container and transform the children into flex items, use the d-flex class.



```
<div class="d-flex p-3 text-white">
    <div class="p-2 bg-danger">Flex item 1</div>
    <div class="p-2 bg-secondary">Flex item 2</div>
    <div class="p-2 bg-primary">Flex item 3</div>
</div>
```

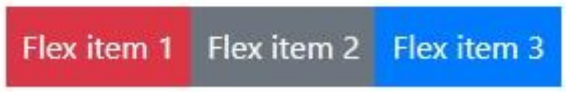
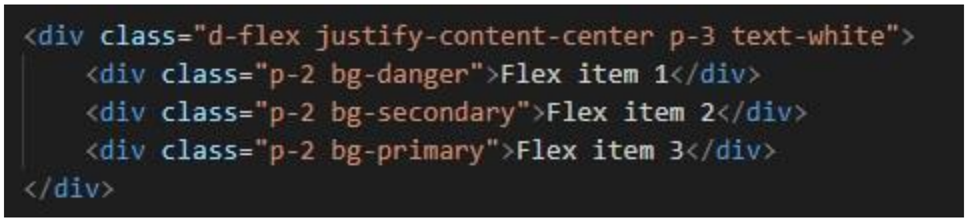


Fig. 3.073 (BS flex: d-flex class)

By default, the flex items are display horizontally. But you can use .flex-row to display the flex items horizontally (side by side), this is default. You can use .flex-row-reverse to right-align the horizontal direction, this makes the flex item aligned to the right.

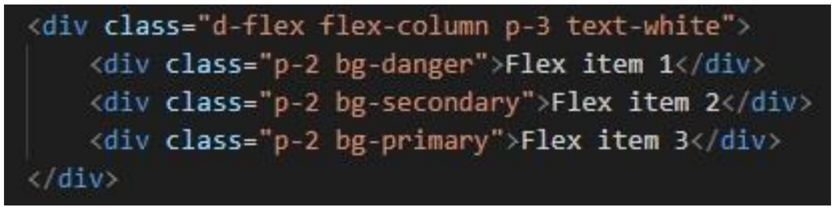
You can use the .justify-content-* classes to change the alignment of flex items. Valid classes are start (default), end, center, between or around.



```
<div class="d-flex justify-content-center p-3 text-white">
    <div class="p-2 bg-danger">Flex item 1</div>
    <div class="p-2 bg-secondary">Flex item 2</div>
    <div class="p-2 bg-primary">Flex item 3</div>
</div>
```

Fig. 3.074 (BS flex: justify-content-center class)

You can use .flex-column to display the flex items vertically (on top of each other), or .flex-column-reverse to reverse the vertical direction.



```
<div class="d-flex flex-column p-3 text-white">
    <div class="p-2 bg-danger">Flex item 1</div>
    <div class="p-2 bg-secondary">Flex item 2</div>
    <div class="p-2 bg-primary">Flex item 3</div>
</div>
```

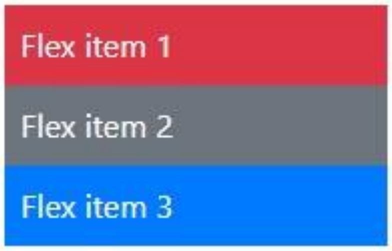
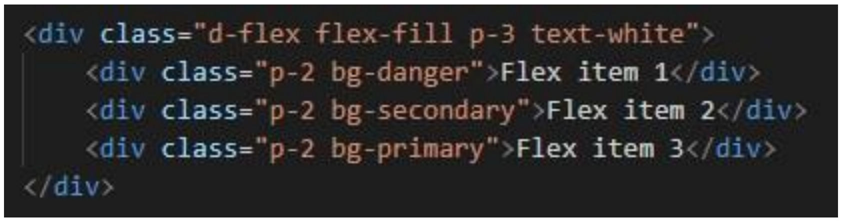


Fig. 3.075 (BS flex: flex-column class)

You use .flex-fill on flex items to force them into equal widths.



```
<div class="d-flex flex-fill p-3 text-white">
    <div class="p-2 bg-danger">Flex item 1</div>
    <div class="p-2 bg-secondary">Flex item 2</div>
    <div class="p-2 bg-primary">Flex item 3</div>
</div>
```

Fig. 3.076 (BS flex: flex-fill class)

Exercise on bootstrap

1. Explain any of the ways used to add bootstrap to your web page.
2. Bootstrap requires a containing element to wrap site contents; mention either any of the two (2) classes you can use.
3. List any five (5) of the bootstrap class you can use to add color to your web page using bootstrap.
4. What class will you add to a <table> element to make it responsive using bootstrap?
5. What class will you use to make an image a circle using bootstrap?
6. Write short note on bootstrap buttons.
7. How do you create a loader/spinner in bootstrap?
8. Write short note on bootstrap dropdown.
9. What is a bootstrap carousel?
10. Explain about bootstrap flex.

Project on Bootstrap

With your knowledge of HTML, CSS, JavaScript and Bootstrap; design Zaufanjinba foundation website exactly like the link here www.zaufanjinba.org using bootstrap.

REFERENCES

1. https://www.w3schools.com/bootstrap4/default.asp
2. https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp
3. https://www.w3schools.com/bootstrap4/bootstrap_containers.asp
4. https://www.w3schools.com/bootstrap4/bootstrap_grid_basic.asp
5. https://www.w3schools.com/bootstrap4/bootstrap_typography.asp
6. https://www.w3schools.com/bootstrap4/bootstrap_colors.asp
7. https://www.w3schools.com/bootstrap4/bootstrap_tables.asp
8. https://www.w3schools.com/bootstrap4/bootstrap_images.asp
9. https://www.w3schools.com/bootstrap4/bootstrap_jumbotron.asp
10. https://www.w3schools.com/bootstrap4/bootstrap_alerts.asp
11. https://www.w3schools.com/bootstrap4/bootstrap_buttons.asp
12. https://www.w3schools.com/bootstrap4/bootstrap_button_groups.asp
13. https://www.w3schools.com/bootstrap4/bootstrap_spinners.asp
14. https://www.w3schools.com/bootstrap4/bootstrap_pagination.asp
15. https://www.w3schools.com/bootstrap4/bootstrap_cards.asp
16. https://www.w3schools.com/bootstrap4/bootstrap_dropdowns.asp
17. https://www.w3schools.com/bootstrap4/bootstrap_navs.asp
18. https://www.w3schools.com/bootstrap4/bootstrap_navbar.asp
19. https://www.w3schools.com/bootstrap4/bootstrap_forms.asp
20. https://www.w3schools.com/bootstrap4/bootstrap_forms_input_group.asp
21. https://www.w3schools.com/bootstrap4/bootstrap_carousel.asp
22. https://www.w3schools.com/bootstrap4/bootstrap_modal.asp
23. https://www.w3schools.com/bootstrap4/bootstrap_utilities.asp
24. https://www.w3schools.com/bootstrap4/bootstrap_flex.asp