# Overview of HTML

# Introduction to HTML

- HTML stands for Hypertext Markup Language - it is a markup language for creating web pages and applications.
- HTML contains a particular syntax - namely **elements** and **attributes** - that web browsers parse in order to render the content of the webpage.
- With HTML, the structure and content of a webpage is defined. Styling and dynamic behavior are introduced with CSS and JavaScript, respectively.

# HTML5

HTML5 introduced a new **DOCTYPE** declaration `<!DOCTYPE html>` and the character encoding (charset) declaration `<meta charset="UTF-8">`. The `<DOCTYPE>` declaration is used to inform the browser about the version of HTML used in the document. It is known as the Document Type Declaration (DTD). It just instructs the browser about the document type. A **character encoding** is an approach of converting bytes into characters. For validating the HTML document, a program must choose a character encoding.

HTML5 also introduced features to allow us to embed audio and video files on the web page and provides the support to run JavaScript in the background.

Take a look at the structure of the HTML5 file given below:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title of the document</title>
  </head>

  <body>
    Content of the document......
  </body>
</html>
```

**NOTE:** HTML5 uses UTF-8 as a default character encoding.

`<!DOCTYPE html>` should be in the beginning of the document before any tags.

# Helpful Resources

- Mozilla

# HTML Tags

## Common HTML Tags

There are a vast number of HTML tags you could use on your webpage, but below are listed the most common:

- `<div>` - defines a "division of the page"
- `<p>` - defines a paragraph
- `<span>` - an inline tag for grouping text or elements
- `<b>` - bold text
- `<i>` - italicized text
- `<h1>`, `<h2>`, ... `<h6>` - these are headings, h1 is largest and h6 is smallest
- `<br>` - line break
- `<table>` - defines a table
- `<img src="URL">`
- `<ol>` - an ordered list
- `<ul>` - an unordered list
- `<a href="">` - makes a hyperlink

### Hyperlinks

To make a hyperlink in a webpage, use the `<a>` tag:

```
<p> Here is a <a href="www.google.com">link to Google!</a></p>
```

### Tables

To create a table, use the following markup. `<tr>` defines a table row, `<td>` defines a table cell, and `<th>` is used for table headers.

```
<table>
  <thead>
    <tr>
      <th>Id</th>
      <th>Name</th>
    </tr>
  </thead>
  <tbody>
```

```
    <tr>
      <td>1</td>
      <td>Alice</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Bob</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Charlie</td>
    </tr>
  </tbody>
</table>
```

## Lists

There are two options for making lists in HTML - ordered or unordered lists. Ordered lists are defined with `<ol>`, unordered lists are defined with `<ul>`, and the list items within either are denoted with `<li>`:

```
<ol> <!-- ordered lists render as 1, 2, 3, etc.. -->
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ol>
<ul> <!-- list items in here will just be bullet points -->
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ul>
```

# HTML5 Semantic Elements

Semantic Elements defines the meaning for the web page rather than just presentation.

- The HTML elements like `<div>` and `<span>` are not related to the content on the web page. This kind of elements are called as non-semantic elements.

- The HTML elements like `<form>`, `<table>`, and `<article>` are used to define the content and on the webpage. This kind of elements are called as semantic elements.

We will discuss about the following semantic elements that helps to define the element's purpose on the webpage:

```
<section>
<article>
<header>
<footer>
<nav>
<aside>
<figure>
<figcaption>
<details>
<mark>
<summary>
<time>
```

## Section

The HTML5 `<section>` tag defines a thematic grouping of content. In a document, we have sections like chapters, headers, footers, introduction, content, contact information, etc.

*Example*

```
<section>
  <h1>Protocol</h1>
  <p>A protocol is a standard set of rules that allow electronic devices to communica
te with each other.</p>
</section>
```

## Article

The `<article>` element represents a section of content that forms an independent part of a document or site such as Forum post, Blog post, Newspaper article, etc.,

*Example*

```
<article>
  <h1>What WHO do?</h1>
```

```
  <p>WHO works worldwide to promote health, keep the world safe, and serve the vulner
able. Our goal is to ensure that a billion more people have universal health coverage
, to protect a billion more people from health emergencies, and provide a further bil
lion people with better health and well-being..</p>

</article>
```

## Header

The `<header>` element specifies a header for a document or section.

*Example*

```
<!-- example defines a header for an article -->

<article>

  <header>

    <h1>World Health Organisation</h1>

    <p>What we do?</p>

  </header>

  <p>WHO works worldwide to promote health, keep the world safe, and serve the vulner
able. Our goal is to ensure that a billion more people have universal health coverage
, to protect a billion more people from health emergencies, and provide a further bil
lion people with better health and well-being..</p>

</article>
```

## Footer

The `<footer>` element used to define the footer for a document or section. It contains information about the author of the document, copyright information, links to terms of use, contact information, etc.

*Example*

```
<footer>

  <p>Posted by: someone </p>

  <p>Contact information: <a href="mailto:someone@example.com">

  someone@example.com</a>.</p>

</footer>
```

## Navigation

The `<nav>` element is for major navigation blocks that specify a set of navigation links.

*Example*

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

## Aside

The `<aside>` element is used to identify content that is related to the primary content of the web page. The content inside the `<aside>` element does not constitute the primary content of the page. For example, we can have author information, related links, related content, and advertisements.

*Example*

```
<p>I went TajMahal for this summer</p>


<aside>
  <h4>Taj Mahal</h4>
  <p>The Taj Mahal is an ivory-white marble mausoleum on the south bank of the Yamuna river in the Indian city of Agra.</p>
</aside>
```

## HTML figure and figcaption Elements

The `<figure>` element describes some flow content, optionally with a caption, that is self-contained and referenced as a single unit from the main flow of the document.
The `<img>` and `<figcaption>` elements are grouped in a `<figure>` element. We use the `<img>` element to insert an image on the web page. To add the visual explanation of the image, we need a caption for that image. This can be achieved in the HTML5 by using `<figcaption>` element.

*Example*

```
<figure>
  <img src="WorldMap.jpg" alt="WorldMap">
  <figcaption>Fig1. - World Map </figcaption>
</figure>
```

## Other HTML Elements

- `<details>` - Used to add details that user can view or hide
- `<mark>` - Used to highlight or mark the text.
- `<summary>` - Defines a obvious heading for a `<details>` element
- `<time>` - Used to add a date/time.

# HTML5 Audio tag

The HTML5 `<audio>` element used to embed audio in a web page.

**Example**

```
<body>


<h1>The audio element</h1>

<p>Click the Play button:</p>

<audio controls>

  <source src="River.ogg" type="audio/ogg">

  Invalid audio!!! - Browser doesn't support.

</audio>


</body>
```

In the above example, we have two tags, `<audio>` and `<source>`.

The `<audio>` element defines sound content and it has a *controls* attribute that adds audio controls, like play, pause, and volume. Any text within the `<audio>` and `</audio>` displayed on the browser only if the audio was not supported by the browser.

The `<source>` element defines the media resources for the audio files and it has attributes such as *src* and *type*. The *src* is used to specify the file format of the audio content and *type* specifies the media types that `<audio>` element supports.

The file format supported by `<audio>` element and respective media types are tabulated below:

| File Format | Media Type |
|---|---|
| MP3 | audio/mpeg |
| OGG | audio/ogg |
| WAV | audio/wav |

# HTML5 Video tag

The `<video>` element used to embed a video on a web page, such as a movie clip or other video streams.

**Example**

```
<body>


<h1>The video element</h1>
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
    Invalid audio!!! - Browser doesn't support.
</video>


</body>
```

HTML5 Video Attributes used in the above example are described below:

| Attribute | Value | Description |
|---|---|---|
| src | URL | Specifies the URL of the video file |
| controls | controls | Specifies the video controls such as a play/pause button, etc. |
| height | pixels | Sets the height of the video player |
| width | pixels | Sets the width of the video player |

Note: Any text between the `<video>` and `</video>` tags will be displayed in browsers that do not support the `<video>` element.

The MIME Types supported by the `<video>` element is tabulated below:

| Format | MIME-type |
|---|---|
| MP4 | video/mp4 |
| WebM | video/webm |
| Ogg | video/ogg |

## HTML Document Structure & DOM

# Starting HTML Documents

Every HTML document (ending with `.html` extension) should begin with a special tag known as the DOCTYPE declaration - this lets browsers know what kind of document it is using (HTML, in our case) as well as which version of the markup language is being used. For HTML5, the newest version and the one which we'll be using, the DOCTYPE syntax is:

```
<!DOCTYPE html>
```

The doctype declaration tag does not have a closing tag and it is not self-closing either.

Next, the tag which begins the root of our HTML document is the `<html>` tag. Everything about our webpage will be nested within this tag.

Within the `html` element we have two important tags - the `<head>` and the `<body>` tags. The `head` element will contain all the metadata associated with this page, including the title,

character encoding, and references to external style sheets. The `body` element contains the actual content of our page that will be rendered on the screen by the browser.

## Hello World Example

Let's write a simple webpage that will show off our knowledge of HTML thus far. Open up a new file in a simple text editor and save the file as `hello-world.html`. Then write the following HTML markup and save it:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
    <meta charset="utf-8">
  </head>
  <body>
    <div>
      <!-- THIS IS A COMMENT! -->
      <p>This is my first paragraph written in HTML</p>
    </div>
  </body>
</html>
```

The only tag you may be unfamiliar with is the `meta` tag above - this defines the character encoding that the file will be using. Also, the `<!-- ... -->` syntax denotes a comment.

If you now open your `hello-world.html` file in your browser of choice, you'll see your webpage rendered. It may look ugly now, but we'll address that when we talk about CSS.

## Elements and Attributes

# Elements

HTML is composed of many different **elements** - these provide the structure of the document. Elements are defined within HTML files using **tags** - for example, one very common tag is the `<div>` tag. The tag is enclosed within angle brackets. Most elements have a closing tag which define the end of the element, using the backslash notation - for example, a closing "div" tag would be `</div>`. HTML elements may be *nested* within other elements, like so:

```
<div> text inside the div but outside the paragraph
  <p> this text is inside the paragraph </p>
```

```
</div>
```

In order for HTML to be **valid**, tags must be properly nested - an outer tag cannot be closed before an inner one. For example, the following markup would not be considered valid:

```
<div> invalid!
  <p> cannot close the div here => </div>
</p>
```

Not all elements have closing tags, some are self-closing. For example, the `<img />` tag, which defines an image.

# Attributes

HTML elements can also have **attributes** defined within the tag - these are key/value pairs that give metadata about the tag that are important for the browser to know. For example, image elements must have a URL which the browser can call to retrieve the image file to display on the page - we use the `src` attribute to do this: `<img src="/URL/to/get/cat.png" alt="cool cat!" />`. As you may have guessed, the `alt` attribute specifies an alternative text to show when the image cannot be displayed.

## Global Attributes

**Global** attributes are those that can be applied to any element on the page. Some important global attributes are:

- class
- id
- hidden
- lang
- style
- tabindex
- title

There are also many attributes that should be applied to only certain elements, including the `src` and `alt` attributes shown above. We'll discuss more about these when relevant.

# Inline and Block Elements

Before listing all the HTML elements available to use, it's important to know the difference between inline and block-level elements.

Block-level elements are those that will render on new lines in blocks by default, instead of rendering within the line itself like inline elements do.

One example of a block element is

, and a common inline element is . Try them out on your webpage and notice the difference.

# HTML Forms

An **HTML form** is *a section of a document* that contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus, etc. Using these elements the page can collect information from a user which is typically submitted to a web server. To create a form, you would use the `<form>` tag.

*Why use an HTML Form?*

- We use forms to collect some information/data form the user.
    - For example: If a user wants to purchase some items on the internet, he or she must fill out the form which will collect information such as the shipping address and payment details so that the item can be sent to the given address.

# Attributes Used in HTML Forms

There are several attributes that you can use on the `<form>` tag and on `<input>` elements. We will cover:

- action
- target
- name
- method
- value
- placeholder
- required

Take a look at the following example and find the different attributes:

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


<form action="/test.php" target="_blank" method="GET">
  Username:<br />
  <input type ="text" name="username" placeholder="Username" required/>
  <br />
  Password:<br />
  <input type ="password" name="password" />
```

```
  <br /><br />
  <input type ="submit" value ="Submit" />
</form>


</body>
</html>
```

## The Action Attribute

The action attribute indicates where the form data will be processed. Typically the value is a URL of a server. Generally, the form data is sent to a webpage on the webserver after the user clicks on the submit button.

In the above code, after clicking on the *submit* button, the form data would be sent to a page called *test.php*.

## The Target Attribute

The Target attribute is used to specify whether the submitted result will open in the current window, a new tab or on a new frame. The default value used is "self" which results in the form submission in the same window. To make the result display in a new browser tab, the value should be set to "blank".

In the above code, after clicking on the *submit* button, the result will open in a new browser tab. Most often this attribute is not present and the default value of "self" is used.

## Name Attribute

The name attribute should be provided for each input element. It is not required, but the value provides a label for the data once the form is submitted. If the name attribute is not specified in an input field then the data of that field will not be sent.

In the above code, after clicking the *submit* button, the form data will be sent to a page called */test.php*. The data sent will include the *username* and *password* fields.

## The Method Attribute

The method attribute is used to specify the HTTP method used to send data while submitting the form. There are only two options available: GET and POST.

**GET** - When using the GET method, after the submission of the form, the form values will be visible in the address bar of the browser.

**POST** – When using the POST method, after the submission of the form, the form values will NOT be visible in the address bar of the browser.

## The value Attribute

The value attribute specifies an initial value for an input field. It also serves as the attribute to use when providing a button label for submit and reset input elements.

In the above example, there are no default values.

## The placeholder Attribute

The placeholder attribute specifies a hint that describes the expected value of the input field (a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value. The placeholder attribute works with the following input types: text, search, url, tel, email, and password.

In the above example, the text field has a placeholder of "Username".

## The required Attribute

The required attribute indicates an input field that must be filled out before submitting the form. In most modern browsers, it will prevent the user from submitting the form until an acceptable value is entered. The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

In the above example, only the text field is required.
**NOTE**: The required attribute doesn't have a value portion. You only need to specify the word 'required'.

## The min and max Attributes

The min and max attributes specify the minimum and maximum values for an input field. The min and max attributes work with the following input types: number, range, date, datetime-local, month, time and week.

**Tip**: Use the max and min attributes together to create a range of legal values. (For example: Set a maximum date or a minimum date)

*Example*

```
<form>
  <label for="datemax">Enter a date before 1980-01-01:</label>
  <input type="date" id="datemax" name="datemax" max="1979-12-31"><br><br>


  <label for="datemin">Enter a date after 2000-01-01:</label>
  <input type="date" id="datemin" name="datemin" min="2000-01-02"><br><br>


  <label for="quantity">Quantity (between 1 and 5):</label>
  <input type="number" id="quantity" name="quantity" min="1" max="5">
```

```
</form>>
```

Input Elements and Input Types

# Input Element in HTML Forms

An HTML form collects information from elements. You will specify an addition **type** attribute to indicate which field to display. Various fields can be created such as a text field, checkbox, password field, or radio button.

## Text Field

A text field is a one-line input field that allows the user to input a line of text. Text Fields are created by specifying the type attribute value as "text".

The below example will display a text field with the label *Email Id:*

**Example**

```
<form>
    <label for="email-input">Email Id:</label><br>
    <input type="text" name="email-input" />
</form>
```

## Password Field

Password fields are a type of text field in which the text entered is masked using asterisk or dots. This prevents others form viewing the screen to see what is typed in. Also, it's created by specifying the type attribute value as "password".

**Example**

```
    <form>
        <label for="user-password">Password:</label><br/>
        <input type="password" name="user-password"/>
    </form>
```

**NOTE**: Although a password field is masked, it is NOT encrypted. You will have to use other measures such as HTTPS to encrypt data once the HTML form is submitted.

## Radio Buttons

Radio Buttons are used to let the user select exactly one value from a list of predefined options. It is created by specifying the type attribute value as "radio".

**Example**

```
<form>
    SELECT GENDER
    <br/>
    <input type="radio" name="gender" id="male">
    <label for="male">Male</label><br>
    <input type="radio" name="gender" id="female">
    <label for="female">Female</label>
</form>
```

**NOTE**: A form may have multiple sets of radio buttons. In order to make sure the user only selects one option from a given set, each radio element must have matching `name` attributes. In the example above, both buttons have a `name` attribute value as **gender**.

# Checkboxes

Checkboxes are used to let the user select one or more values from a pre-defined set of options. The type attribute value for checkboxes input control is "checkbox".

**Example**

```
<form>
    <b>SELECT SUBJECTS</b>
    <br>
    <input type="checkbox" name="subject" id="math" />
    <label for="math">Math</label>
    <input type="checkbox" name="subject" id="science" />
    <label for="science">Science</label>
    <input type="checkbox" name="subject" id="english" />
    <label for="english">English</label>
</form>
```

**NOTE**: Just like radio buttons, a form may have multiple sets of checkboxes. In order to make sure the user selects options related to a given set, each checkbox element must have matching `name` attributes. In the example above, each checkbox has a `name` attribute value as **subject**.

# File select boxes

File select boxes are used to allow the user to select a local file on their computer and send it as an attachment to the webserver. It is similar to a text box with a button that allows the user to browse for a file. Instead of browsing for the file, the path and the name of the file can also be written. They are created by specifying a type attribute value as "file".

**Example**

```
<form>
    <label for="fileselect">Upload:</label>
    <input type="file" />
</form>
```

# Text area

A text area is a multi-line text input control which allows users to provide a paragraph or multiple lines of text. It is created by using the "textarea" element.

This is one of the few input controls that DO NOT use the `<input>` element.

You can control the size of a text area by adding attributes "rows" and "cols" to specify the number of rows and columns of text it supports. Most often text area elements are resizable, but the default size is managed by those two attributes.

**Example**

```
<form>
    <label for="Description">Description:</label>
    <textarea rows="5" cols="50" name="Description"
                        id="Description"></textarea>
</form>
```

# Select Boxes (Drop-downs)

Select boxes are used to allow users to select one or more options from a drop-down list. Select boxes are created by using two elements: `<select>` and `<option>`.
The `<select>` element defines a drop-down while list items are defined within the select element using `<option>` elements.

**Example**

```
<form>
    <label for="country">Country:</label>
    <select name="country" id="country">
```

```
            <option value="United States">United States</option>
            <option value="Canada">Canada</option>
            <option value="Mexico">Mexico</option>
        </select>
    </form>
```

## Reset and Submit Buttons

The **submit** button allows the user to send the form data to the web server. You can define a submit button by specifying the type attribute as "submit".

The **reset** button is used to reset the form data and will display any default values. You can define a reset button by specifying the type attribute as "reset".

**Example**

```
    <form action="test.php" method="post" id="users">
        <label for="username">Username:</label>
        <input type="text" name="username" id="Username" />
        <input type="submit" value="Submit" />
        <input type="reset" value="Reset" />
    </form>
```

# HTML5 Validation

When we submit a form, the data in the form will be sent to the server, before that we need to make sure that all the required details are filled out also in the correct format. The process of ensuring or validating the data before submitting to the server is called Client-side form validation.

Let us discuss form validation. For example, we submit any registration form, we may come across such messages listed below:

- "This field is required" (it can't be blank).
- "Please enter the valid phone number" (it should contain the only number).
- "Invalid email address" (it should be in "lmn@asd.com" format).
- "Your password must include one number, one uppercase letter, one lowercase letter, and one special character".

The browser displays the above messages by validating the data entered the registration form. It checks whether the data is in the correct format and satisfies the constraints set by

the application or not. If the browser validates the data, then it is called as *client-side validation*. Validation done by the server is called as *server-side validation*.

There are two different types of client-side validation.

1. **Built-in form validation** - It uses HTML5 form validation features.
2. **JavaScript validation** - It is coded using JavaScript. This validation is completely customizable.

# Built-in form validation

We have attributes that can be used with the form elements for validation. Some of the attributes are listed below:

- **required**: Used when the user must fill the field before submitting the form.
- **minlength and maxlength**: Used to specify the minimum and maximum length of the text.
- **min and max**: Used to specify the min and max values for the numerical fields.
- **type**: Defines the data should be a number or an email address or other predefined type.
- **pattern**: Defines a pattern (regular expression) the entered data needs to follow.

If the data entered in a form satisfies the constraints are considered **valid**. If not, it is considered **invalid**. We use: `valid` and: `invalid` CSS pseudo-class to differentiate between valid and invalid input fields. The`: invalid` CSS pseudo-class used to select and style form `<input>` elements whose value is *invalid* according to the validation attributes specified in the `<input>` element. Similarly, the: valid CSS pseudo-class selects and styles the valid form input elements.

For example, the email inputs `(<input type="email">)` whose value does not match a valid email address pattern then the style defined by the: `invalid` CSS pseudo-class. Below, we have HTML and CSS code for this example.

```
HTML:


 <form>
     <label>
        <span>Email:</span>
        <input type="email" id="email" placeholder="name@domain.com">
    </label>
</form>


CSS:


//:valid
```

```
.valid {
    input[type=email]:valid { border-color: $g; }
}
//:invalid
.invalid {
    input[type=email]:invalid { border-color: $r; }
```

# Overview of CSS

CSS stands for **Cascading Style Sheets** - it is a language for styling HTML documents by specifying certain rules for layout and display in key/value pairs. Style Sheets are a simple and powerful method of allowing attachment of rendering information to HTML documents. It used to style the webpages by setting background-color, font color, font size, font family, etc.

A CSS consists of a *set of rules* that defines the styles for a web page. A CSS style rule composed of **selectors** and **declarations**. The selector is an HTML Element *like h3 used in the below example*. The declaration is comprised of a property and a value surrounded by curly braces. In the below example font-family, font-style and color were properties of the selector h3. Arial, italic and red were the values assigned, respectively, to the properties.

*Example:*

```
  h3 {
      font-family: Arial;
      font-style: italic;
      color: red
      }
```

# CSS Box Model

The CSS box model used to determine how our web page is rendered by browser. It considers each element on the page as a box, CSS allows you to apply different properties that define where and how that element appears. Web pages are made up of rectangular boxes arranged and related to each other.

Every box has 4 parts - **margin**, **border**, **padding** and **content**. The margin is an outermost box, inside that the border, then padding, then the content is innermost. All box sizes/formatting can be styled with CSS.

**Margin** - It is a space between border and margin. It is useful to separate the element from its neighbors. The dimensions are given by the margin-box width and the margin-box height.

**Border** - It is the area between the box's padding and margin. Its dimensions are given by the width and height of the border.

**Padding** - It is a space around the content area and within the border-box. Its dimensions are given by the width of the padding-box and the height of the padding-box.

**Content** - It consists of content like text, image, or other media content. It is bounded by the content edge and its dimensions are given by content-box width and height.

## Inline, Internal, and External Style sheets

# Types of CSS

There are three types of CSS which are given below:

- Inline CSS
- Internal or Embedded CSS
- External CSS

## Inline CSS

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

**Example:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Inline CSS</title>
</head>
<body>
    <p style = "color:#009900;
            font-size:50px;
       font-style:italic;
       text-align:center;">
```

```
                            Hello World
        </p>
</body>
</html>
```

## Internal or Embedded CSS

This can be used when a single HTML document must be styled **uniquely**. The CSS rule set should be within the HTML file in the *head section* i.e. the CSS is embedded within the HTML file.

**Example:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Internal CSS</title>
    <style>
        .main {
            text-align:center;
        }
        .hi{
            color:#009900;
            font-size:50px;
            font-weight:bold;
        }
        .greeting {
            font-style:bold;
            font-size:20px;
        }
    </style>
</head>
<body>
    <div class = "main">
        <div class ="hi"> Hi, Everyone!!</div>
        <div class ="greeting">
            Hello World!!
```

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

## External CSS

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

**Example:** The file given below contains CSS property. This file should be saved with a .css extension. For Ex: **style.css**

```
body {
        background-color:powderblue;
        }
    .main {
        text-align:center;
        }
    .hi {
            color:#009900;
        font-size:50px;
        font-weight:bold;
        }
    #greeting {
        font-style:bold;
        font-size:20px;
        }
```

Below, we have HTML file that makes use of the above created external style sheet (style.css). This can be achieved by using `<link>` tag. The `<link>` element has *rel* and *href* attributes. The *rel* specifies the relationship between the current document and the linked document. In this case, *rel* attribute value will be *style sheet* because we going to add the external style sheet to the HTML document. The *href* attribute is used to specify the location of the external style sheet file.

```
<html>
    <head>
```

```
            <link rel="style sheet" href="style.css"/>
        </head>
        <body>
            <div class = "main">
                <div class ="hi"> Hi, Everyone !</div>
                <div id = "greeting ">
                    Hello world !!
                </div>
            </div>
        </body>
    </html>
```

# CSS Properties

There are a number of CSS properties that you can use to style our webpage. Here we'll discuss some of the CSS properties such as Border, Padding, Margin, display, position, color, and text-align.

## CSS Border Property

The CSS border property allows to style the border area of a box. The properties and corresponding values with examples covered under the CSS border are tabulated below:

| Property | values | Usage | Example |
|---|---|---|---|
| border-width | medium, thin, thick, length | Used to define the border area of a box | `div { border-width: medium 10px thick 15px; }` |
| border-style | none, hidden, dashed, dotted, double, groove, inset, outset, ridge and solid | sets the style of a box's border | `p { border-style: dotted; }` |
| border-color | hex-value for colors | specify the color of a box's border | `p { border-style: solid; border-color: #ff0000; }` |

# CSS Padding Property

The CSS padding property allow you to set the padding area for an element that separates its border from its content. The padding property can take one, two, three, or four values separated by white spaces as listed in the below table. Depending on the list of property values, the HTML element has the padding area on the top, bottom, right, and left.

| Examples | Explanation |
|---|---|
| `p { padding: 70px; }` | Sets the padding for an `<p>` element to 70 pixels for all four sides |
| `p { padding: 35px 70px; }` | Sets the padding for an `<p>` element to 35 pixels for top and bottom and 70 pixels for right and left sides. |
| `p { padding: 35px 70px 40px; }` | Sets the padding for an `<p>` element to 35 pixels for the top, 70 pixels for the left and right side and 40 pixels for the bottom. |
| `p { padding: 35px 70px 40px 80px; }` | Sets the padding for an `<p>` element to 35 pixels for the top, 70 pixels for the right side, 40 pixels for the bottom and 80 pixels for the left side. |

The padding property is a shorthand property for the padding-top, padding-right, padding-bottom, and padding-left properties. The below examples set padding on a specific side for the HTML element.

*Examples*:

```
h1 {
        padding-bottom: 10px;
}
p {
        padding-top: 20px;
        padding-left: 50px;
 }
```

# CSS Margin Property

The CSS margin property is similar to the CSS border property, but it sets the margins around the sides of an element's box instead of the border. It also takes one, two, three, or four values separated by white spaces. The shorthand properties are margin-top, margin-right, margin-bottom, and margin-left to set a margin on respective sides.

*Example:*

```
p {
    margin-left: 10px;
    margin-right: 30px;
}
h1{
    margin: 25px 50px;
}
```

## CSS Display Property

The display property controls the display behavior of an element. The CSS display property sets whether an element is treated as a block or inline elements and the layout used for its children, such as flow layout, flex or grid.

There are two types of HTML elements: **inline-level elements** and **block-level elements**. The differences between these elements affect how you use the box model. Both Inline and block-level elements appear within the body of an HTML page. But, inline-level elements are used to create a short structure that can have data and other inline elements. Inline level elements include `<b>`, `<big>`,`< i>`, `<small>`, `<tt>`, `<abbr>`, `<acronym>`, `<code>`, `<strong>`, etc.
Block-level elements used to create larger structures than inline elements also it starts on new lines by default whereas inline-level elements not. Block elements include `<p>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`,`<h6>`, `<ol>`, `<ul>`, `<pre>`, `<address>`, `<blockquote>`, `<dl>`, `<div>`, `<fieldset>`, `<form>`, `<hr>`, `<table>`, etc. .

The Syntax for the display property is `selector {display: value}`. The property values and description with examples are tabulated below:

| Property value | Description | Example |
|---|---|---|
| block | behaves likes block-level elements | `a {display: block;}` |
| inline | behaves like inline-level elements | `ul li { display: inline; }` |
| none | elements doesn't generate boxes | `h1 { display: none;}` |

## CSS Position Property

The position property defines how an element will be arranged on a page. The Syntax for the position property is `selector {position: value}`. The property values are static, relative, absolute, fixed, or inherit.

static - The element's box is arranged automatically consistent with the normal flow.

relative - The element's box position is relative to its normal flow position. You can adjust the normal flow position by using the top, bottom, left and right properties.

absolute - The element's box arranged to an absolute position with reference to its containing block. Its containing block is that the nearest ancestor element that has its position property set to relative, absolute, or fixed. The top, right, bottom, and left properties are used to set the offset of the element's box with reference to its containing block.

fixed - The element's box position is offset from its browser window by using the top, right, bottom, and left properties. The element's box won't move when the browser window is scrolled.

inherit - The inherit keyword is employed to specify that the value for this property should be taken from the parent element. If inherit is used with the root element, then the initial value for this property is going to be used.

*Example:*

```
a {position: static;}

div {position: relative; top: 20px; left: 50px;}

h1 {position: absolute; top: 30px; left: 20px;}

div {position: fixed; top: 325px; left: 60px;}
```

## CSS Color property

The color property is used to specify the foreground color of text. The color properties are set using 5 different color notation types which is listed below:

```
a {color: red;}

div {color: #3c5;}

h1 {color: #ffa500;}

div {color: rgb(100,20,255);}

#id1 {color: rgb(30%,50%,70%);}
```

## CSS text-align property

The text-align property is used to align the content inside the element. The text inside the element can be aligned in 4 ways - left, right, center and justify.

*Example:* The text-align properties are set to left, right, justify, and center.

```
 div {text-align:left;}

 h1 {text-align: right;}

  p {text-align: justify;}

  div {text-align: center;}
```

# Element Selector

The element selector selects HTML elements by their name / tag name *like a, h1, div, p etc*.

*Example:* Here, we use `<p>` as an element selector. The text inside the `<p>` will be center-aligned also blue color.

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
    text-align: center;
    color: blue;
}
</style>
</head>
<body>
<p>This style will be applied on every paragraph.</p>
<p> Here also</p>
</body>
</html>
```

## Class and Id Selectors

# Class Selector

In the CSS, the class selector is a name preceded by a period (“.”). It uses the class attribute of an HTML element to match the specific HTML element. We can have a Class selector specific to an HTML element *like we have `p.class` in the below example*.

In the below example, we have two class selectors inside the `<style>` element. The class selector `.intro` is applied to the element which has an attribute called `class`, whose value is `intro` and the `p.intro` class selector is applied to the `<p>` element which has an attribute called `class`, whose value is `intro`. Also, the `<p>` element without the `class` attribute doesn't get affected.

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

```
<style>
.intro {
  text-align: center;
  color: red;
}

p.intro {
  text-align: center;
  color: blue;
}
</style>
</head>
<body>

<h1 class="intro">Red and center-aligned heading</h1>
<p class="intro">blue and center-aligned paragraph.</p>
<p> this will not be affected </p>
</body>
</html>
```

# ID Selector

In the CSS, the ID selector is a name preceded by a hash character (“#”). It uses the id attribute of an HTML element to match the specific HTML element. The **id** of an element should be unique within a page, so the id selector is used to select one unique element.

*Example:* Here, we use `#para1` as an ID selector. Inside the body, we have two `<p>` elements. The CSS style rule applied to the element which has an attribute called `id`, whose value is `para1`. Therefore, `Hello World!` will be center-aligned also blue color.

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
  text-align: center;
  color: blue;
```

```
}
</style>
</head>
<body>


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


</body>
</html>
```

*NOTE:* The id name should start with the alphabet, not with numbers. Also, the HTML element without the 'id' attribute doesn't get affected.

# Responsive Web design

Responsive Web design is the approach that allows websites and pages to render (or display) on all devices and screen sizes by automatically adapting to the screen, whether it's a desktop, laptop, tablet, or smartphone. Responsive web design works through CSS, using various settings to serve different style properties depending on the screen size, orientation, resolution, color capability, and other characteristics of the user's device. It is a combination of flexible grids, flex boxes, flexible images, and media queries.

# CSS3 Media Queries

CSS3 supports responsive web design, all kinds of transitions, transformations, and animations and provides box-sizing tools that enable the user to adjust the size of any element without changing the dimensions or padding of the element.

Media queries allow you to customize the presentation of your web pages for a specific range of devices like mobile phones, tablets, desktops, etc. without any change in markups. It composed of a media type and expressions that check for the conditions of particular media features. It is a logical expression that is either true or false.

**Syntax:** A media query consists of an optional **media type** and any number of **media feature** expressions. Multiple queries are often combined in various ways by using **logical operators**. Media queries are case-insensitive. A media query is true if the media sort of the media query matches the media sort of the device and every one expression within the media query are true. It uses the `@media` rule to incorporate a block of CSS properties as long as a particular condition is true. Queries involving unknown media types are always false.

```
@media not|only mediatype and (mediafeature and|or|not mediafeature) {
  CSS-Code;
}
```

Media Types - It describes the category of a device.

- `all` - used for all media type devices
- `print` - used for printers
- `screen` - used primarily for screens like computer screens, tablets, smart-phones, etc.
- `speech` - used for screen readers that "reads" the page aloud

Media features - It describe specific characteristics of the user agent, output device, or environment. Some of the media features are `grid`, `height`, `width`, `hover`, `max-aspect-ratio`, `max-color`, `max-color-index`, `max-height`, etc.

Logical Operators - It used to compose a media query. Logical Operators used in media queries are `not`, `and`, and `only`.

*Example:* It changes the background color of the `<body>` element to "red" and the font style to "Arial" when the browser window is 600px wide or less.

```
@media only screen and (max-width: 600px) {
  body {
    background-color: red;
    font-family: Arial;
  }
}
```

# Bootstrap

Bootstrap is an open-source framework and mobile-first approach for developing responsive websites. It is a front-end framework programmed to support both HTML5 and CSS3. It comprises the list of components such as Typography, Code, Table, Forms, Button, Images, Icons, etc. A responsive website is a website that automatically adjusts the screen size and looks good on all devices, from smartphones to a desktop. It is easy to use, saves time and customizable. Bootstrap 4 is the newest version of Bootstrap.

## How to Download Bootstrap?

There are two ways to download and start using Bootstrap for our website:

1. Download Bootstrap 4 - You can download the bootstrap from getbootstrap.com.
2. Include Bootstrap from a CDN - You can skip the download with BootstrapCDN by copying the links and paste it in the head section of the html code.

# Bootstrap grid systems

Bootstrap grid system consists of series of containers, rows, and columns to layout and align content. It creates a responsive layout and built with grid and flex boxes. Bootstrap classifies the screen sizes ranging from extra small to extra-large based on the pixels. The transition between the various screen sizes is known as breakpoints. Bootstrap grid system provides a set of responsive classes to specify the screen size.

## Bootstrap containers

Containers are the basic layout elements used to wrap the content in the website. The container is the root of the Bootstrap 4 grid system which consists of all the elements and controls the layout width of a page.

Bootstrap provides two container classes:

- `.container` - used to provide the responsive fixed width container
- `.container-fluid` - used to provides a full-width container that spans the entire width of the viewport.

```
<div class="container">
  Hello! I am in a Fixed container.
</div>


<div class="container-fluid">
  Hello! I am in a Fluid container.
</div>
```

## Bootstrap rows

Bootstrap rows are horizontal slices of the screen. They are only used for containing columns or a wrapper for columns. They have to be placed in containers to avoid the horizontal scroll on the page. The bootstrap columns should be children for the row, to align properly. `.row` class is used to create the rows inside the container.

```
<div class="row">
  ...
</div>
```

## Bootstrap columns

`.col` class sets the width for the column dynamically that means we can set the width of each column in a row. Grid system supports a maximum of 12 columns in a row and anything after that will be shifted to a next row.

You can set the `size` for the column (ranging from 1 to 12). *For example:* The .col-4 class creates 3 equal columns (because 12/4= 3 columns). The col-6 class creates 2 equal columns. We can also set different sizes for them.

```
<div class="row">
  <div class="col-5">
    This is col-5
  </div>
  <div class="col-7">
    This is col-7
  </div>
</div>
```

Also, you can set the *Breakpoints* for columns and used to specify the screen resolution. `.col-[breakpoint]` class is used to define the behavior for the columns in the displayed devices. There are 4 different breakpoint class listed below:

- `.col-sm` - used for small devices where the screen width is equal to or greater than 576px
- `.col-md` - used for medium devices where the screen width equal to or greater than 768px
- `.col-lg`- used for large devices where the screen width equal to or greater than 992px
- `.col-xl` - used for extra-large devices where the screen width equal to or greater than 1200px

```
<div class="row">
 <div class="col-lg">
   This is a column
 </div>
</div>
```

We can combine the sizes and breakpoints and use a single class with the format `.col-[breakpoint]-[size]`. Below we have a simple example for Bootstrap Grid System.

```
<!-- Using bootstrap container class -->
<html>
<head>
  <title>Bootstrap Grid System </title>

  <!-- Add Bootstrap Links -->
```

```
  <link rel="style sheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>


<body>
    <div class="container" >
      <div class="row" >
            <div class="col-4 border" >
              This is a col-lg-4
            </div>
            <div class="col-3 border">
              This is a col-lg-3
            </div>
        </div>
        <div class="row">
            <div class="col-sm-6 border">
              This is a col-lg-6
            </div>
            <div class="col-sm-6 border">
              This is a col-lg-6
            </div>
        </div>
    </div>
</body>
</html>
```

# Bootstrap Tables

The bootstrap provides classes to style of HTML tables. To change the default style of HTML table, add bootstrap `.table` class to the `<table>` element. Bootstrap table classes used to create striped table rows, add borders to the cell and the tables, add the colors to the table rows, table headers and also allow us to differentiate handover rows, etc.

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

```
 <table class="table ">
    ...
 </table>
</div>
```

The following classes are used with `.table` class like `<table class="table [class-name]">` to add styles to the table:

- `.table-dark` - used to get a dark background table with the light text.
- `.table-bordered` - adds the border on all sides of the table and cells.
- `.table-striped` - used to add zebra-stripes table rows.
- `.table-hover` - applies the hover effect to the table rows and cells, i.e., you will get grey background when a cursor rolls over a cell or a row.
- `.thead-light` - used within `<thead>`, it makes the table header to appear light.
- `.thead- dark` - used within `<thead>`, it makes the table header to appear dark gray.

*Contextual classes* is used to color the table rows `<tr>` and cells `<td>` individually. We can used these contextual classes in `<tr>` and `<tr>` of the `<table>` element: `.table-warning`, `.table-primary`, `.table-info`, `.table-success`, `.table-danger`, `.table-active`, `.table-light`, `.table-secondary` and `.table-dark`. (For example: `<tr class="table-primary">`).

# Bootstrap Alerts

Bootstrap alerts provide the contextual feedback messages for user actions on the page. Alert boxes contain text that needs the user's attention. The `.alert` class is display the alert message on website. The `.alert` class is used as base class then followed by contextual classes. The contextual alert classes are: `.alert-success`, `.alert-info`, `.alert-warning`, `.alert-danger`, `.alert-primary`, `.alert-secondary`, `.alert-light` and `.alert-dark`. Also, we can dismiss/close any alert messages by using `.alert-dismissible` class. We can animate alerts when dismissing by using `.fade` and `.show` classes. Just add the `data-dismiss="alert"` attribute and `class = "close"` attribute to the `<button>` element to close the alert message boxes.

In the below example, we are creating a success alert message box by adding the contextual class .alert-success to the .alert base class.

```
<div class="alert alert-success alert-dismissible fade show"> <strong>Success!</stron
g>
Your details submitted successfully.
<button type="button" class="close" data-dismiss="alert">&times;</button>
</div>
```

# Bootstrap forms

Bootstrap forms allow us to create elegant forms on the website. We can style the all textual form controls like input, select, textarea, etc., by using `.form-control class`. There are 3 different types of form layout - Vertical Form layout, Horizontal Form layout, Inline Form layout.

Vertical Form Layout - This is a default form layout provided by Bootstrap.

Horizontal Form Layout - In this layout, labels and form controls are aligned side-by-side by using the Bootstrap grid classes. The `.row` class and the `.col-*-*` grid classes used on the form groups to define the width of the layout. Also, to center them vertically we use `.col-form-label` on the `<label>`elements.

Inline Form layout - In this layout, a series of labels, form controls, and buttons are displayed in a single horizontal row. The `.form-inline` class is used within `&lt;form&gt;` element to create inline form layout.

We can stack the checkboxes or radio buttons vertically (line by line), by applying the class `.d-block` on each `<label>` element.

Below, we have a simple example for Horizontal Form Layout.

```
  <div class="form-group row">
        <label for="inputEmail" class="col-sm-2 col-form-label">Email</label>
        <div class="col-sm-10">
            <input type="email" class="form-control" id="inputEmail" placeholder="Ema
il">
        </div>
  </div>
```

# Bootstrap buttons

Bootstrap includes several button styles in which each styles serves a semantic purpose. The `.btn` classes are designed to be used with the `<button>`, `<a>` and `<input>` element. The contextual classes used with the `.btn` classes are `btn-primary`, `btn-secondary`, `btn-success`, `btn-danger`, `btn-warning`, `btn-info`, `btn-dark`, `btn-light`, and `btn-link`. When using button classes on `<a>` elements, we should use `role="button"` attribute to convey the purpose. The `.btn-outline-*` used to remove all background colors on any button.
The `.btn-lg` or `.btn-sm` classes are used to create larger and smaller buttons.
The `.active` and `.disabled` class are used to represent the active and disabled state programmatically.

Example:

```
<button type="button" class="btn btn-primary btn-lg">Large button</button>
<button type="button" class="btn btn-outline-secondary btn-lg">Large button</button>
```

# Bootstrap Navbar

Bootstrap allows us to create a responsive navigation header that includes support for navigation, branding, collapse plugin and more. Navbars require a wrapping `.navbar` with `.navbar-expand{-sm|-md|-lg|-xl}` for responsive collapsing and color scheme classes. Some of the navbars sub-components are listed below with their purposes.

- `.navbar-brand` - used with most elements that contain the name of the company, product, or project.
- `navbar-toggler` - used for collapse plugin and other navigation toggling behaviors (allows to change the position on mobile devices)
- `.form-inline` - used with any form-controls and actions.
- `.navbar-text` - used to align the text vertically also makes it centered.
- `.collapse.navbar-collapse` - used for grouping and hiding navbar contents depending upon the screen resolution.

# Bootstrap Colors

Bootstrap provides color utility classes that support for styling links with hover states.

Text color - used to set the color for text of an element. The contextual text color classes are `.text-muted`, `.text-primary`, `.text-success`, `.text-info`, `.text-warning`, `.text-danger`, `.text-secondary`, `.text-white`, `.text-dark`, `.text-body`, and `.text-light`.

Background color - used to set the background color for an element. The contextual background color classes are similar to the contextual text color classes, here we use `.bg-*` instead of `text-*`.

Example:

```
<!-- setting text color-->
<p class="text-success">.text-success</p>
<p class="text-danger">.text-danger</p>
<p class="text-warning">.text-warning</p>


<!--setting background and text color-->
<div class="bg-light text-dark">.bg-light</div>
<div class="bg-dark text-white">.bg-dark</div>
<div class="bg-white text-dark">.bg-white</div>
<div class="bg-transparent text-dark">.bg-transparent</div>
```

# Intro to Amazon Simple Storage Service (Amazon S3)

Amazon Simple Storage Service (Amazon S3) is an **object storage service** that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics.

## S3 as an Object Storage System

Amazon S3 is an object storage system which means the data is stored as individual objects rather than in some kind of hierarchy like you would see in a file system or directory structure.

- Each individual object is put into a **bucket** and you connect to Amazon S3 using a URL.
- The URL will have the name of your object and the name of your bucket. The bucket is just the container in which you put your objects.
- You use what's called a REST API to connect to S3 using a URL.
- Your browser does a HTTP PUT request and it puts the objects in the bucket.

## Use Cases for S3

You can store any type of file in S3.

- Backup and Storage
- Application Hosting
- Media Hosting
- Software Delivery - host software apps that your customers can download
- Static Website Hosting - You can configure a static website to run from an S3 bucket.

## References

- S3 Documentation

# AWS S3 Bucket Configuration

- To upload your data (photos, videos, documents etc.) to Amazon S3, you must first create an S3 bucket in one of the AWS Regions. You can then upload your data objects to the bucket.

- Every object you store in Amazon S3 resides in a bucket. You can use buckets to group related objects in the same way that you use a directory to group files in a file system.
- Amazon S3 creates buckets in the AWS Region that you specify. You can choose any AWS Region that is geographically close to you to optimize latency, minimize costs, or address regulatory requirements.

## To Create a Bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose `Create bucket`.
3. In `Bucket name`, enter a DNS-compliant name for your bucket.

   The bucket name must:

   - Be unique across all of Amazon S3.
   - Be between 3 and 63 characters long.
   - Not contain uppercase characters.
   - Start with a lowercase letter or number.

4. In `Region`, choose the AWS Region where you want the bucket to reside. 5. In `Bucket settings for Block Public Access`, uncheck the `Block Public Access` settings that you want to apply to the bucket. 6. Choose `Create bucket`.

## Deleting a Bucket

Follow the steps provided in the AWS User Guide for deleting a bucket

**Once you have created a bucket, you can use it to host static websites.**

# Hosting Static Sites on S3

You can use Amazon S3 to host a static website. On a *static website*, individual webpages include static content.

By contrast, a *dynamic website* relies on server-side processing, including server-side scripts such as PHP, JSP, or ASP.NET. Amazon S3 does not support server-side scripting, but AWS has other resources for hosting dynamic websites. To learn > more about website hosting on AWS, see Web Hosting.

- To configure your bucket for static website hosting, you can use the AWS Management Console without writing any code.
- You can also create, update, and delete the website configuration programmatically by using the AWS SDKs. The SDKs provide wrapper classes around the Amazon S3

REST API. If your application requires it, you can send REST API requests directly from your application.
- To host a static website on Amazon S3, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket.

## Steps to Host a Static Website on your Previously Created S3 Bucket

1. Click on your bucket in your S3 Management Console
2. Select `Overview` > `Create Folder` - this is where you will upload static HTML files from your computer to be hosted on the S3 bucket.
3. Copy and Paste the following dummy HTML text into a file titled `index.html` on your computer.

```
<html>
<head>
    <title>Here's a title, displayed at the top of the window.</title>
</head>
<body>
    <h1>A main heading, usually the same as the title.</h1>
    <p>Be <b>bold</b> in stating your key points. Put them in a list: </p>
    <ul>
        <li>The first item in your list</li>
        <li>The second item; <i>italicize</i> key words</li>
    </ul>

    <p>Add a link to your favorite <a href="https://revature.com/learn-to-code/">Web site</a>.
        Break up your page with a horizontal rule or two. </p>
    <hr>
    <p>Finally, link to <a href="page2.html">another page</a> in your own Web site.</p>
    <!-- A copyright notice.-->
    <p>&#169; Revature 2020</p>
</body>

</html>
```

4. Select `Upload` and upload the `index.html` file you just saved to your computer.

5. Select `Properties`
6. Select `Static website hosting`
7. Select `Use this bucket to host a website` > enter the name of the file we're hosting: `index.html`.
8. Return to your bucket > click on the uploaded `index.html` file > click `Make public`.
9. Visit the `Object URL` and you will see your static webpage hosted on your configured S3 bucket!

## References

- Configuring a bucket as a static website using the AWS Management Console