

## **ASHOK IT – UI Development Notes – HTML, CSS, JavaScript**

### **What is the internet?**

The Internet is a network where a group of computers are connected together. Systems connected in that network are able to communicate with each other and exchange data.

### **What is the web?**

Web is the medium to share information across the internet. But what is this information? It can be generally represented using text, images, video, audio and other media.

The Internet is very powerful because we can communicate with any system in that network. We can issue commands to any computer in the world when connected to this network – Internet.

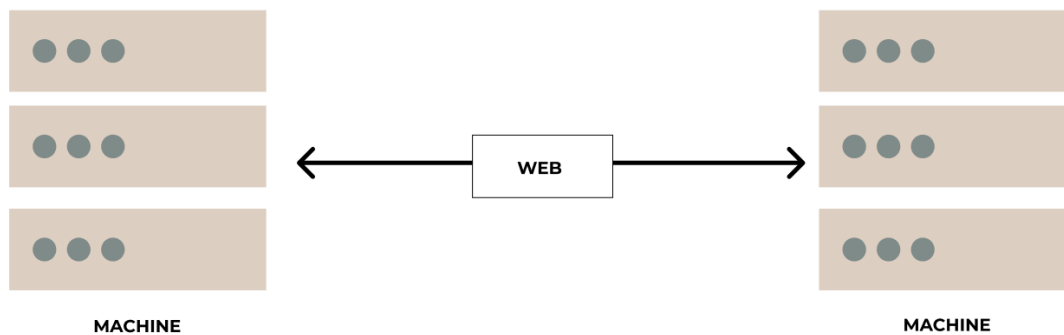
Today, we can use web applications like WhatsApp, Gmail, shop online using Amazon and do many things because all these computers are connected together in a single network and we can use it to do whatever we want. This global network is just a way computers are connected to exchange information in one network.

### **How is data exchanged on the web ?**

Web uses a protocol called HTTP (HyperText Transport Protocol) to access the information from any computer on the internet. A protocol just means a set of rules. Hyper Text just means a special text which has a link to other resources on the Internet. It can include video, images, sounds etc.

So, HTTP is the set of rules using which Hyper Text is downloaded from the server and displayed on the browser. Once this hypertext is stored in the server that is connected to the internet then using the HTTP protocol this Hyper Text can be exchanged between the computers. Basically it carries the Hyper Text between two computers to establish a communication.

## HYPER TEXT TRANSFER PROTOCOL



### How does this HTTP work?

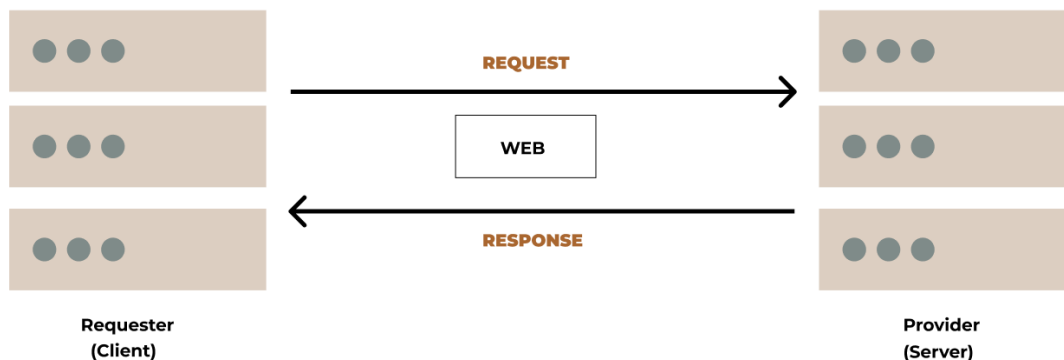
HTTP simply works using a request – response approach. In simple words, one machine asks and the other machine serves.

The client machine asks for some data by making a request to the server machine, and once the server receives this request, it serves by sending HyperText in response.

The client (requestor) initiates the request for some resource on the server (provider), and once the provider receives the request it will search for the resources and if found sends back that resource to the requestor.

So basically, if there is no requester, the provider does nothing. Some one has to initiate a request for the provider to respond.

## HYPER TEXT TRANSFER PROTOCOL



### **So who is the requestor / client ?**

Requestor generally is the browser that initiates the request to the provider. So, all the actions that happen in the browser or at the requestor side are called Client Side.

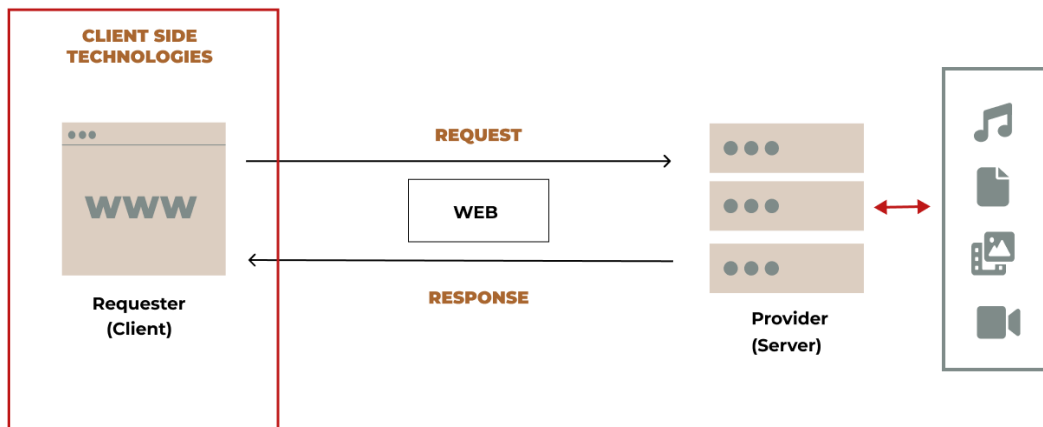
All the actions that take place at your computer before the request is made to the provider is generally referred to as the Client Side. For example, if your browser executes a code on your computer then we say the code is executed at client side.

### **So who is the provider / server ?**

Anything that happens at the server is called Server Side. Once the request is sent to the server or provider then all the operations or steps that the server executes is referred to as Server side.

Server side code executes when the Client makes a request to the server and asks to provide a resource or perform some operation. The output of the code is sent back to the client by the server.

**So we will be exploring and learning about the client side technologies.**



### What are the client side technologies?

Client side technologies are those which are used to build a website on our client machine. At the core, there are three technologies which are used to build a website which are responsible for working on the three parts that go into building a webpage.

1. First the basic skeleton of the webpage. For example the text, images, etc and how they are put together on the page. So basically the content and structure of the webpage. This is done using HTML, which stands for **HyperText Markup Language**.
2. Now the page is created but looks bad. How do we color elements, adjust the sizing, position, etc of the elements? In other words, how can we style the web page? This is done using CSS, which stands for **Cascading Style Sheets**.
3. The web page has its content and is structured. We can style it however we want it using CSS. Now how can we make it interactive and dynamic? Upto this point, we can only just read and view content on the web page. Web

applications generally have a lot of behavior and provide feedback or perform operations upon user interaction at runtime. For example, validating a form, changing the page data, hiding and showing things, downloading data in the background etc. This is handled by **JavaScript**, which is a crucial part in building dynamic and interactive applications on the client side.



**We will start by learning HTML and then move on to CSS, then to Javascript.**

### **So what is HTML?**

HTML stands for Hyper Text Markup Language. So, HTML is a Markup Language that is used to mark the contents and then tell the browser how to display them on the screen.

It is a markup language that our browser understands and knows how to display the contents. In HTML, we can do the following things:

- Write how to display the contents on the browser
- Structure the contents as required
- Provide links to resources on another server
- Embed Videos and Audio from the server
- And more ...

The main purpose of HTML is to tell the browser how the data is displayed and in which location.

A simple HTML page would look like this in code.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Document</title>
  </head>
  <body>
    <h1>So what is HTML?</h1>
    <p>
      HTML stands for Hyper Text Markup Language. So, HTML is a Markup
      Language that is used to mark the contents and then tell the browser how to
      display them on the screen.
    </p>
    <p>
      The main purpose of HTML is to tell the browser how the data is
      displayed and in which location.
    </p>
  </body>
</html>
```

The HTML page is made up of tags, which represent different elements that we can create on a web page.

### **What is a Tag?**

A Tag is the text between the left angle bracket (<) and the right angle bracket (>). There are starting tags (such as <p>) and there are ending tags (</p>). A starting tag will have just the element name inside the angle brackets, like so, <p>, and the closing tag will have a forward slash and the element name in between the angle brackets, like so, </p>. We can put content in between the opening and closing tags. That content can be text, other HTML tags, etc.

For every opening tag there should be a closing tag. If we define an opening tag of an element inside the tags of another element, we have to make sure that we close our element tag before the closing of the other element.

However some tags can be self closing, which means they do not need a closing tag and we cannot put content in between those opening and closing tags (closing tag should be avoided here). We will use them standalone or give them content through some other way.

### **What is an Attribute?**

An Attribute is a name=value pair on the element tag. Attributes help to show additional details about the element.

For example, `<p width="200">My first paragraph</p>`

Here the attribute "width" tells the browser how wide the paragraph should be. We will learn more about these in detail later in the course.

### **Structure of the HTML Page**

Using Markups in HTML, Elements can be arranged in such a manner that it can be logically shown how the final output looks like on a computer screen.

Elements, tags, attributes of HTML were supposed to indicate how the title of the page looks, how the heading should look and in which order.

HTML markup language uses a markup structure to organize the elements in the page. Consider this example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello World!</title>
  </head>
  <body>
```

```
<h1>This is a heading</h1>
<p>This is a paragraph</p>
</body>
</html>
```

Do not worry about the HTML but focus on how the elements are organized to tell the browser about the purpose of each element and structure of each element. The main purpose is to tell the browser how to display the content on the computer screen so you organize the markups in the same way it should be displayed. Sequencing is important which decides how the elements are organized on the page. There is also a parent child relation between the elements.

Here `<head>` and `<body>` elements are child elements of `<html>`. `<p>` and `<h1>` are child elements of the `<body>` parent element.

## Parts of the HTML page

Consider this example HTML page:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Document</title>
  </head>
  <body>
    <h1>So what is HTML?</h1>
    <p>
      HTML stands for Hyper Text Markup Language.
    </p>
  </body>
</html>
```

## DOCTYPE



!DOCTYPE is the first declaration of the HTML page. This will indicate to the browser what type of HTML version we are using. If we want to tell browsers to use different HTML versions then we mention that in this DocType.

`<!DOCTYPE html>`

## **HTML**

This is the root element and tells the browser that this is a HTML document. All the HTML tags should be inside this element. You should not define anything outside this element.

## **HEAD**

Head element is used for the following things:

- Include other supporting files required for this page.
- Tell search engines about your page.
- Set Title for your Page.
- Mention the Meta data about your page.
- All the elements you define in the HEAD tag are not displayed on the page.

## **BODY**

This is the place where you define all the elements. Any element defined under the body will be displayed on the page.

## **HTML Text Elements**

Purpose of HTML Text elements is to display text in a format that is appealing and readable. There are different types of HTML Text Elements that help to display our content like we see the content in the newspaper and magazine.

### **Benefits of this Text Tags:**

- Display Headings and Paragraphs
- Markup the Bold, Italics and Underlining the text.
- Show different format of text like Java code and normal readable text

## Structural Markup

These are the markups that are used to define the text and give a real meaning to the text. Like mentioning the Heading and Paragraphs on the Web Page.

## Semantic Markup

Semantic elements are used to provide extra information to the user by bold text, underlines and italics. They increase the readability and also help to mark the text in the paragraphs or headings. Like marking a Quotation can also be done via the Semantic Markup.

These are the HTML Text Elements that we learnt today:

- Headings
- Paragraph

## Usage of Headings

Headings are used to display the title of the paragraph or show some text in bigger size and bolder. There are 6 headings tags <h1> to <h6>

h1 is the bigger and h6 is the smallest. Headings text tags are always big in size and used to grab the attention of the user or show the purpose of the article. The behavior of the headings can be changed later using CSS which we will learn later. Browser has few default settings to show headings.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Headings</title>
  </head>
  <body>
    <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>
</body>
</html>
```

## Usage of Paragraphs

Paragraphs is the place where you put most of your content to display as a paragraph on the browser.

Paragraph tags help to organize the content nicely into a small container which makes content easy to read and edit.

TAG: <p>

ELEMENT: <p>sometext</p>

Example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Paragraphs</title>
  </head>
  <body>
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptasiure
      nesciunt fugit harum ipsam aliquam accusantium cumque aperiam!
    </p>
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptasiure
      nesciunt fugit harum ipsam aliquam accusantium cumque aperiam!
    </p>
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptasiure
      nesciunt fugit harum ipsam aliquam accusantium cumque aperiam!
    </p>
  </body>
</html>
```

## Single Line Breaks

Line Breaks helps to break the line and start the text on a new line. The more breaks you have the more new lines are added. This is done by using the tag <br>.

Please note that <br> is a self closing tag. You can also write it as <br/>.

Eg:

<p>

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

</p>

### Output:

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser.

It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

## Strong Text

Strong tag helps to bold and highlight the text to emphasize some text in an element. It is often used in the Paragraphs. This is done using the <strong> tag. The text that we want to emphasize should be wrapped with this <strong> tag.

Eg:

<p>

**The HyperText Markup Language or HTML** is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

</p>

### Output:

**The HyperText Markup Language or HTML** is the standard markup language for documents designed to be displayed in a web browser.

It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

### **Emphasis Text**

Emphasis is used to emphasize a word or line in the paragraphs. It is similar to italics but this is mainly used to emphasize a word or line of text. It is often used in the Paragraphs. The text that we want to emphasize should be wrapped with this `<em>` tag.

Eg:

```
<p>I <strong>love</strong>to write <em>HTML5</em>!</p>
```

Output:

I **love** to write *HTML5*!

### **Underline Text**

Underline will simply underline the text, and it can be done by wrapping text with the `<u>` tag.

Eg:

```
<p>I <u><strong>love</strong></u> to write <em>HTML5</em>!</p>
```

Output: I **love** to write *HTML5*!

### **Italics**

Italics tag will simply italicize the text. It is often used in the Paragraphs. It can be done by wrapping the text with the `<i>` tag.

Eg:

```
<p>
  I <i><u><strong>love</strong></u></i> to write <em>HTML5</em>!
</p>
```

Output: I *love* to write *HTML5*!

### Code

Code tag is used to display the programming source code on the web page. Browser treats the code tag as special and displays the code as it is written in the html. However, it ignores the new lines. So, the code is displayed in one line.

```
<code>
function add(x, y){
  return a + b;
}
</code>
```

Output: function add(x, y){ return a + b; }

### Preformatted Tag

pre tag is used to display the white space and it will retain the indentation of the format that is written in the html file. Browser does not format anything inside the pre tag but instead it will try to print as it is including the spacing and formatting. This can be done by wrapping the text with a <pre> tag.

Eg:

```
<pre>
```

```
public void add(int a, int b) {  
    return a + b;  
}
```

```
int c = add(1, 2);  
</pre>
```

Output:

```
public void add(int a, int b) {  
    return a + b;  
}
```

```
int c = add(1, 2);
```

## **Anchor Tag**

The <a> tag defines a hyperlink, which is used to link from one page to another. The most important attribute of the <a> element is the href attribute, which indicates the link's destination.

The anchor tag also allows you to link to a specific section of the page. Anchor tags help users to jump on the specific section of the same page.

id – This id attribute is first assigned to the tag to mark as an anchor.

Using <a> tag, we can provide link to that specific location of the page

**Eg:**

```
<a href="https://www.google.com" target="_blank">Search Google!</a>
```

This example above will take us to the Google website in a new tab. The value “\_blank” to the attribute target will make sure this site opens up in a new tab.

Output: [Search Google!](https://www.google.com)

When clicked, this will open <https://www.google.com/> in a new tab.

### **Embedding an image into an HTML page**

<img> tag is used to embed images in the html page. The image could be in the same server or it could be in a different server.

Images are not technically inserted into a web page; images are linked to web pages. The <img> tag creates a holding space for the referenced image.

The <img> tag has two required attributes:

- src – Specifies the path to the image
- alt – Specifies an alternate text for the image, if the image for some reason cannot be displayed

Note: Also, always specify the width and height of an image. If width and height are not specified, the page might flicker while the image loads.

Eg:

```

```



Output:



## HTML Lists

HTML lists allow us to group a set of related items in lists. There are different types of lists in HTML. We will look at the two majorly used types lists used.

1. Ordered Lists
2. Unordered Lists

### Ordered Lists

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.

Eg:

```
<ol>  
  <li>title</li>  
  <li>description</li>  
</ol>
```

Output:

1. title
2. description

### **Unordered Lists**

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.

Eg:

```
<ul>
  <li>title</li>
  <li>description</li>
</ul>
```

Output:

- title
- description

### **HTML Tables**

HTML tables allow web developers to arrange data into rows and columns. It is similar to showing data in a database format or excel sheet format.

#### **Table tags:**

- `<table>` tag is used to define the table
- `<th>` is used to define heading.
- `<tr>` is used to mention rows.
- `<td>` represents columns in the table.

- TAG: <table>, <th>, <tr>, <td>

Eg:

```
<table>
  <tr>
    <th>ID</th>
    <th>Name</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Person 1</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Person 2</td>
  </tr>
</table>
```

Output:

ID	Name
1	Person 1
2	Person 2

### **Table with borders**

Border is an attribute of the table html element. It specifies the border width to draw the line around the table. Border will help to display the lines of the table to understand the width and height of each cell. You can mention the width of the border with this value

Eg:

```
<table border="1">
  <tr>
    <th>ID</th>
    <th>Name</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Person 1</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Person 2</td>
  </tr>
</table>
```

Output:

ID	Name
1	Person 1
2	Person 2

### **Table with Header and Footer**

Table has two important sections apart from row and column and that is header and footer. Header and footer as they are used to specify some text before row and after row. Like headings and subtotals.

Tags: <table>, <caption>, <thead>, <tfoot>, <tbody>

Eg:

```
<table border="1">
  <caption>Employee Salary</caption>
  <thead>
    <tr>
      <td>Month</td>
      <td>Salary</td>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>2022</td>
      <td>1,00,000</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Jan</td>
      <td>50,000</td>
    </tr>
    <tr>
      <td>Feb</td>
      <td>50,000</td>
    </tr>
  </tbody>
</table>
```

Output:

Month	Salary
Jan	50,000
Feb	50,000

2022	1,00,000
------	----------

### RowSpan and ColSpan

RowSpan and ColSpan is to add spacing to the cell. If you want to merge two cells together then this attribute will help.

rowspan and colspan are attribute you can use in the <td> or <tr> elements

To make a cell span over multiple columns, use the colspan attribute:

```
<table border="1">
  <tr>
    <th colspan="2">Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Smith</td>
    <td>43</td>
  </tr>
  <tr>
    <td>Yuvraj</td>
    <td>Singh</td>
    <td>51</td>
  </tr>
</table>
```

Output:

Name		Age
John	Smith	43
Yuvraj	Singh	51

To make a cell span over multiple rows, use the rowspan attribute:

```
<table>
  <tr>
    <th colspan="2">Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td rowspan="2">John</td>
    <td>Smith</td>
    <td>43</td>
  </tr>
  <tr>
    <td>Singh</td>
    <td>51</td>
  </tr>
</table>
```

Output:

Name		Age
John	Smith	43
	Singh	51

## **Forms**

Form elements are the basic building blocks of an HTML page. This is the most common element that you will be using for any website development project.

Common scenario where you will use form elements:

- Contact Form
- Newsletter Form
- Register or Login Form
- Personal Information Form

Forms are used to collect information from the user and post the data to the script running on the server. The script on the server can save the data in the database or file.

With forms, you can restrict the data submitted to the server and use different form elements to capture different types of data from users. For example you can collect email id, phone number or Date from the form.

Below are the list of Form Elements which are most commonly used:

- Basic Form Elements
- Input Elements
- Select Element
- Checkboxes
- Text Area
- Other Form Elements
- Form Action – GET
- Form Action – POST



## Basic Form Elements

The HTML `<form>` element can contain one or more of the following form elements:

- `<input>`
- `<label>`
- `<select>`
- `<textarea>`
- `<button>`
- `<fieldset>`
- `<legend>`
- `<datalist>`
- `<output>`
- `<option>`
- `<optgroup>`

### The `<input>` Element

The `<input>` element can be displayed in several ways, depending on the type attribute.

### The `<label>` Element

The `<label>` element defines a label for several form elements.

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

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

Eg:

```
<form>
  <label for="email">Email Id:</label><br>
  <input type="text" id="email" name="email"><br><br>
  <input type="submit" value="Submit">
</form>
```

Output:

Email Id:

Submit

### The <select> Element

The <select> element defines a drop-down list:

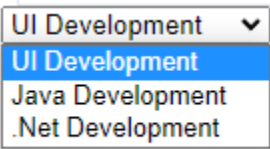
### The <option> Element

The <option> defines an option that can be selected. By default, the first item in the drop-down list is selected.

Eg:

```
<label for="cars">Choose a Course:</label>
<select id="cars" name="cars">
  <option value="volvo">UI Development</option>
  <option value="saab">Java Development</option>
  <option value="fiat">.Net Development</option>
</select>
```

Output:

Choose a Course: 

## Radio Buttons Form Elements

Radio buttons are used to give options for users to pick one option from the various options provided. Users can only select one from the options. This too is done using the `<input>` tag, with the addition of an attribute `type="radio"`.

```
<form>
  <p>Please select your favorite Web language:</p>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

Output:

Please select your favorite Web language:

- ☐ HTML
- ☐ CSS
- ☐ JavaScript

## Checkbox Form Elements

Checkbox buttons are used to pick many options from the list of the items displayed. Users can only select one or more from the options. This done by mentioning the attribute `type="checkbox"` on the `<input>` tag

`type = checkbox` will change the input type to checkbox button.  
`checked` attribute will select the checkbox button

Eg:

```
<form>
  <input type="checkbox" id="html" name="html" value="0" />
  <label for="html"> HTML</label><br />
  <input type="checkbox" id="css" name="css" value="1" />
  <label for="css"> CSS</label><br />
  <input type="checkbox" id="javascript" name="javascript" value="2" />
  <label for="javascript"> Javascript</label><br /><br />
  <input type="submit" value="Submit" />
</form>
```

### TextArea Form Elements

Text area is similar to a text box but you can enter multiple lines in the text area and also control the width and height of the box by specifying the rows and columns attribute of it. Textarea element is represented by the <textarea> tag.

This is mostly used to capture long text from users on the form.

Eg:

```
<form>

  <p><label for="textarea-demo">Learn Web Development</label></p>

  <textarea id="textarea-demo" name="textarea-demo" rows="4" cols="50">
    Lorem ipsum dolor sit, amet consectetur adipisicing elit. Placeat nostrum laboriosam,
    labore molestias commodi possimus minus esse nihil vero nobis saepe libero harum omnis
    pariatur consequatur debitis enim inventore recusandae quisquam sapiente ipsam amet
    blanditiis! Voluptates molestiae maxime dolorem, soluta quod, dolorum quia non quo ab
    eligendi reiciendis! Quia, amet!
  </textarea>

</form>
```

Output:

Learn Web Development

```
  Lorem ipsum dolor sit, amet consectetur  
adipiscing elit. Placeat nostrum laboriosam,  
labore molestias commodi possimus minus esse nihil  
vero nobis saepe libero harum omnis pariatur
```

## HTML Block Level Elements

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element. A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: `<p>` and `<div>`.

The `<p>` element defines a paragraph in an HTML document.

The `<div>` element defines a division or a section in an HTML document.

Eg:

```
<p style="border: 1px solid black">Hello World</p>
```

```
<div style="border: 1px solid black">Hello World</div>
```

Hello World

Hello World

## Inline Elements

An inline element does not start on a new line. An inline element only takes up as much width as necessary.

This is a `<span>` element inside a paragraph.

```
<p>
```

This is an inline span

`<span style="border: 1px solid black">`Hello World`</span>` element inside a paragraph.

```
</p>
```

```
<p>
```

The SPAN element is an inline element, and will not start on a new line and only takes up as much width as necessary.

```
</p>
```

Output:

This is an inline span Hello World element inside a paragraph.

The SPAN element is an inline element, and will not start on a new line and only takes up as much width as necessary.

## DIV HTML Tag

`<div>` tags are called Division Tags they help to divide the tags into groups and apply special formatting on them.

`<div>` acts as a container to wrap the elements together.

Difference between DIV and P:

- `<div>` tags are container tags to hold group of elements
- `<p>` tags are used to tell the browser that it is a paragraph. `<p>` tags is used to write paragraphs

It is the semantic difference between the two:

<div> tag means a block of container

<p> tag means a paragraph of content.

<p> tag is wrapped inside the <div> tag as <div> is a container to group elements together.

Eg:

```
<div>
  <p>something</p>
</div>
```

Output:

something

### **SPAN HTML Tag**

<span> tags are like Division Tags, they help group elements.

<span> acts as a container to wrap the elements together.

### **Difference between SPAN and DIV**

<div> tags are block level elements which means it adds a new line for each div closing tag.

<span> tags are in-line which means it does not add the new line and put all the text inline and in same line.

Eg:

```
<p>
```

This is an inline span

```
<span style="border: 1px solid black">Hello World</span> element inside a
paragraph.
```

```
</p>
```

Output:

This is an inline span Hello World element inside a paragraph.

## HTML class Attribute

The HTML class attribute is used to specify a class for an HTML element. Multiple HTML elements can share the same class.

The class attribute is often used to point to a class name in a style sheet. It can also be used by JavaScript to access and manipulate elements with the specific class name.

In the following example we have three <div> elements with a class attribute with the value of "city". All of the three <div> elements will be styled equally according to the .city style definition in the head section:

Eg:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      .city {
        background-color: cornflowerblue;
        color: black;
      }
    </style>
  </head>
  <body>
    <div class="city">
      <h2>Hyderabad</h2>
      <p>Hyderabad is in Telangana</p>
    </div>
```



```
<div class="city">
  <h2>Mumbai</h2>
  <p>Mumbai is in Maharashtra</p>
</div>

<div class="city">
  <h2>Chennai</h2>
  <p>Chennai is in Tamil Nadu</p>
</div>
</body>
</html>
```

Output:

## Hyderabad

Hyderabad is in Telangana

## Mumbai

Mumbai is in Maharashtra

## Chennai

Chennai is in Tamil Nadu

To create a class; write a dot(.) character, followed by a class name. Then, define the CSS properties within curly braces {}

HTML elements can belong to more than one class.

To define multiple classes, separate the class names with a space, e.g. <div class="city main">. The element will be styled according to all the classes specified.

In the following example, the first <h2> element belongs to both the city class and also to the main class, and will get the CSS styles from both of the classes:

### **Structure of the HTML Page**

Using Markups in HTML, Elements can be arranged in such a manner that it can be logically shown how the final output looks like on a computer screen.

Elements, tags, attributes of HTML were supposed to indicate how the title of the page looks, how the heading should look and in which order.

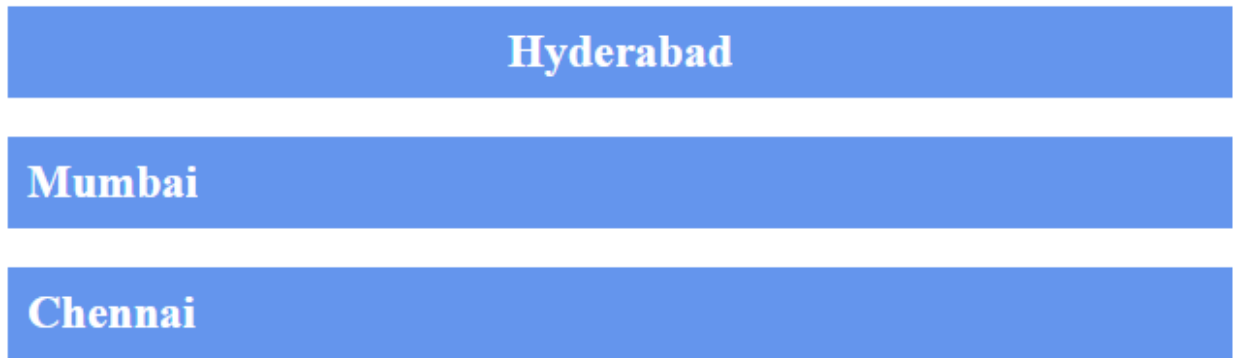
HTML markup language uses a markup structure to organize the elements in the page. Consider this example:

### **Parts of the HTML page**

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      .city {
        background-color: cornflowerblue;
        color: white;
        padding: 10px;
      }

      .main {
        text-align: center;
      }
    </style>
  </head>
  <body>
    <h2 class="city main">Hyderabad</h2>
    <h2 class="city">Mumbai</h2>
    <h2 class="city">Chennai</h2>
  </body>
</html>
```

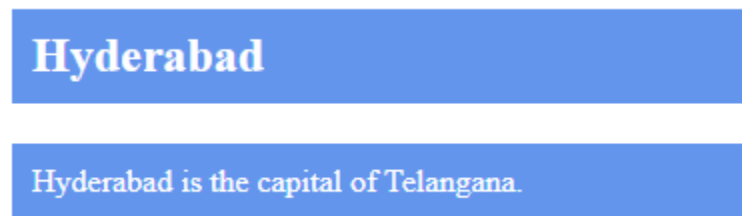
Output:



### **Different Elements Can Share Same Class**

Different HTML elements can point to the same class name.

In the following example, both `<h2>` and `<p>` point to the "city" class and will share the same style:



### **HTML Entities**

Some characters are reserved in HTML. If you use the less than (`<`) or greater than (`>`) signs in your text, the browser might mix them with tags.

Character entities are used to display reserved characters in HTML.

A character entity looks like this:

`&entity_name;`

OR

`&#entity_number;`

**Eg:**

`<h2>A space character: &nbsp;</h2>`

`<h2>The ampersand (and) sign: &amp;</h2>`

`<h2>The double quote sign: &quot;</h2>`

`<h2>The single quote sign: &apos;</h2>`

`<h2>The greater-than sign: &gt;</h2>`

`<h2>The less-than sign: &lt;</h2>`

`<h2>The copyright sign: &copy;</h2>`

`<h2>The registered trademark sign: &reg;</h2>`

Output:

**A space character:**

**The ampersand (and) sign: &**

**The double quote sign: "**

**The single quote sign: '**

**The greater-than sign: >**

**The less-than sign: <**

**The copyright sign: ©**

**The registered trade mark sign: ®**

## HTML Semantics

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of non-semantic elements: `<div>` and `<span>` – Tells nothing about its content.

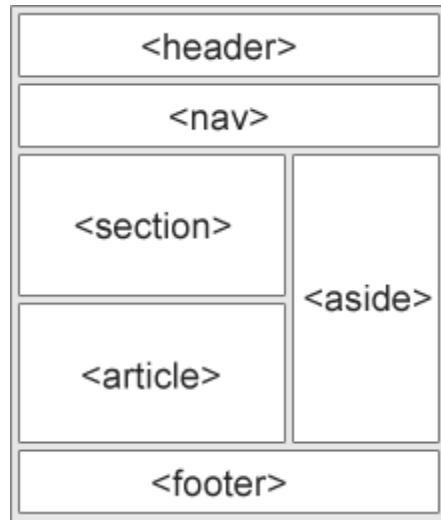
Examples of semantic elements: `<form>`, `<table>`, and `<article>` – Clearly defines its content.

Many web sites contain HTML code like: `<div id="nav">` `<div class="header">` `<div id="footer">` to indicate navigation, header, and footer.

In HTML there are some semantic elements that can be used to define different parts of a web page:

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

A page layout using HTML semantic elements would make sense and give a clear hierarchy to the page. It would look something like this:



## **CSS**

What is CSS? CSS stands for Cascading Style Sheet. It is used to describe how the content should be displayed on the browser, print or screen.

With CSS Language, you can control the layout of the page, color of the text, size of the font, spacing between the text, width and height of the elements and complete presentation of the web page.

In Short, CSS handles the look and feel of the web page. HTML is used to describe the content and CSS is used to display the content in a presentable way.

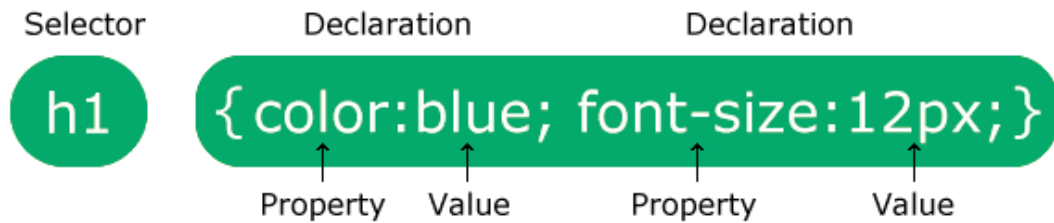
Usage of CSS CSS are written in a file with extension .css and it is linked into the HTML page.

Define the style once and then use it anywhere on your site. Load the CSS once per page and it will manage the entire page layout and presentation. Helps to change the page layout based on the screen the site is viewed on like Mobile, Tablet or Computer Screen.

CSS helps to separate the presentation work from the HTML page and the developers can focus on building the content and displaying it separately. Global Standards also suggest to use CSS and do not use any HTML attributes to style the tag.

Reuse the same CSS for multiple sites to have the same look and feel.

## 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.

Eg:

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

## Example Explained

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



## How To Add CSS to an HTML file

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

- <input>
- <label>
- <select>
- <textarea>
- <button>
- <fieldset>
- <legend>
- <datalist>
- <output>
- <option>
- <optgroup>

## Three Ways to Insert CSS

There are three ways of inserting a style sheet:

1. External CSS
2. Internal CSS
3. Inline CSS

### External CSS

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

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

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

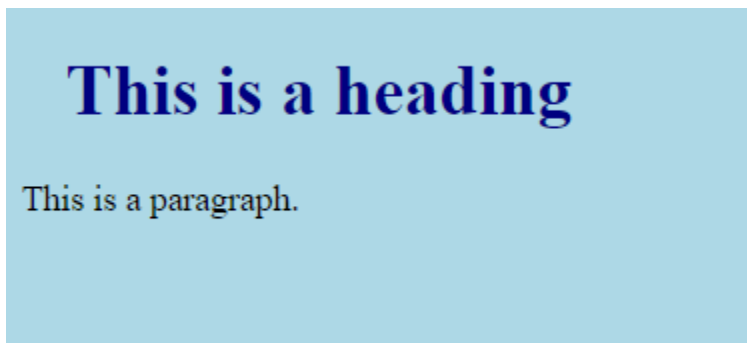
```
<head>
  <link rel="stylesheet" href="externalstyles.css" />
</head>
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>
</body>
</html>
```

### externalstyles.css

```
body {
  background-color: lightblue;
}
```

```
h1 {
  color: navy;
  margin-left: 20px;
}
```

Output:



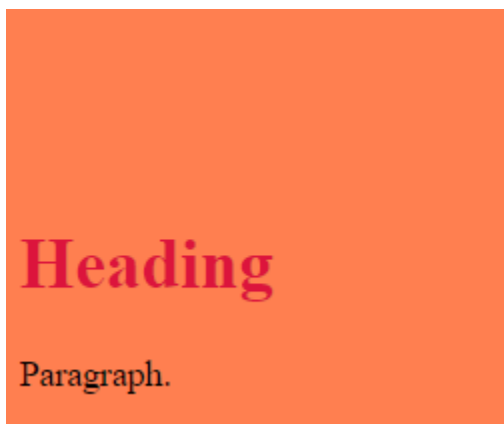
## Internal CSS

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

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

        h1 {
          color: crimson;
          margin-top: 100px;
        }
      }
    </style>
  </head>
  <body>
    <h1>Heading</h1>
    <p>Paragraph.</p>
  </body>
</html>
```

Output:



## Inline CSS

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

```
<!DOCTYPE html>
<html>
  <body>
    <h1 style="color: crimson; text-align: center">Heading</h1>
    <p style="color: royalblue">Paragraph.</p>
  </body>
</html>
```

Output:

Heading

Paragraph.

## CSS Selectors

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

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

We can divide CSS selectors into five categories:

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

## **CSS element Selector**

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

Eg:

```
p {  
  color: crimson;  
}
```

## **CSS id Selector**

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element is unique within a page, so the id selector is used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
#heading {  
  text-align: center;  
  color: royalblue;  
}
```

## **CSS class Selector**

The class selector selects HTML elements with a specific class attribute. To select elements with a specific class, write a dot(.) character, followed by the class name.

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

## **CSS Universal Selector**

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

Eg:

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

## **CSS Grouping Selector**

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

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

```
h1 {  
  color: royalblue;  
}
```

```
h2 {  
  color: royalblue;  
}
```

```
p {  
  color: royalblue;  
}
```

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

To group selectors, separate each selector with a comma.

```
h1, h2, p {  
  color: royalblue;  
}
```

## Fonts

Usage of Text Font or Text word is used to change the behavior of text on the page.

Here are some of the font and text properties. font-family – Specify the font-name to be used.

font-size – Specify the size of the font in pixels or px. It is the same pixel used in MS Word.

font-style – Used to apply the italics, normal or oblique

font-weight – Weight is used to represent how thick the stroke of the font should be. Usually bold, light, medium text-transform – Control the case of text – Uppercase or lowercase.

text-decoration – Decorating the text underline or overline. text-shadow – Add shadow at the back of the text.

Changing Fonts in CSS is no different than modifying a Word document and changing the contents. Most of the words will match with the keywords we use in Microsoft Word Software.

Consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS Fonts</title>
    <style type="text/css">

p {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
  font-size: 12px;
  line-height: 1.5em;
```

```
letter-spacing: 0.2em;
word-spacing: 1em;
text-align: left;
/* left, right, center, justify */
font-style: Oblique;
/* Normal, Italic, Oblique */
font-weight: bold;
/* Light, Medium, Bold, Black */
text-transform: uppercase;
/* lowercase, capitalize */
text-decoration: underline;
/* none, overline, line-through */
text-indent: 500px;
}
```

```
p::first-letter {
    font-size: 200%;
}
```

```
p::first-line {
    text-shadow: none;
}
```

</style>

</head>

<body>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

</p>

</body>

</html>



Output:

**LOREM**  
**IPSUM IS SIMPLY DUMMY TEXT OF THE PRINTING AND**  
**TYPESETTING INDUSTRY. LOREM IPSUM HAS BEEN THE**  
**INDUSTRY'S STANDARD DUMMY TEXT EVER SINCE THE**  
**1500S, WHEN AN UNKNOWN PRINTER TOOK A GALLEY**  
**OF TYPE AND SCRAMBLED IT TO MAKE A TYPE**  
**SPECIMEN BOOK.**

### Usage of Colors

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.

Eg:

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

You can represent colors in CSS using different ways.

Colors can be applied to almost every HTML tag like background, text, border and fill the box.

Mostly commonly used method to represent colors: RGB Value, Hex Code Value and the name of the Color.

RGB is represented with the short form rgb(red, green, blue) and numbers inside it.

RGB(255, 0, 0) – Red, RGB(0, 255, 0) – Green

Hex Code Value is also used to represent the specific colors. The value starts with # and then followed by numbers & characters in Hexadecimals. It is typically 6 digits long.

```
background-color: #ff0000 ;
```

You can use UPPERCASE – FF or lowercase ff to represent the HEX value, but it is good to use lowercase.

Consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS Colors</title>
    <style type="text/css">
      div {
        padding: 20px;
        width: 50%;
        font-size: 20px;
      }
      .blueboxRGB {
        background-color: rgb(65, 105, 225);
        color: white;
      }
      .blueboxHEX {
        background-color: #4169e1;
        color: white;
      }
      .blueboxNAME {
        background-color: royalblue;
        color: white;
      }
    </style>
  </head>
  <body>
```

```
<div class="blueboxRGB">Blue BOX - RGB - rgb(255,0,0)</div>
<hr />
<div class="blueboxHEX">Blue BOX - HEX - #FF0000</div>
<hr />
<div class="blueboxNAME">Blue BOX - NAME - blue</div>
<hr />
</body>
</html>
```

Output:

Blue BOX - RGB - rgb(255,0,0)

---

Blue BOX - HEX - #FF0000

---

Blue BOX - NAME - blue

---

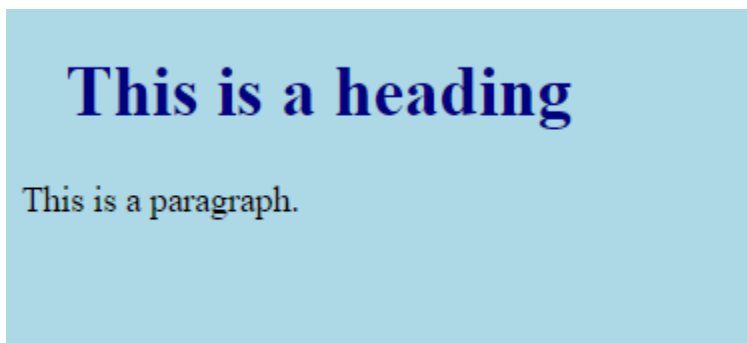
```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="externalstyles.css" />
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

### externalstyles.css

```
body {
  background-color: lightblue;
}
```

```
h1 {
  color: navy;
  margin-left: 20px;
}
```

Output:



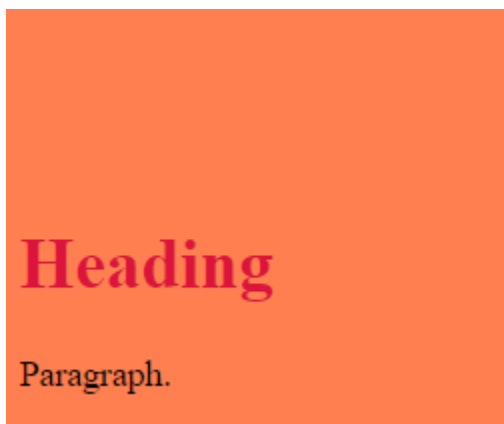
## Internal CSS

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

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

        h1 {
          color: crimson;
          margin-top: 100px;
        }
      }
    </style>
  </head>
  <body>
    <h1>Heading</h1>
    <p>Paragraph.</p>
  </body>
</html>
```

Output:



## Inline CSS

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

```
<!DOCTYPE html>
<html>
  <body>
    <h1 style="color: crimson; text-align: center">Heading</h1>
    <p style="color: royalblue">Paragraph.</p>
  </body>
</html>
```

Output:

Heading

Paragraph.

## CSS Selectors

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

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

We can divide CSS selectors into five categories:

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

## **CSS element Selector**

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

Eg:

```
p {  
  color: crimson;  
}
```

## **CSS id Selector**

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element is unique within a page, so the id selector is used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
#heading {  
  text-align: center;  
  color: royalblue;  
}
```

## **CSS class Selector**

The class selector selects HTML elements with a specific class attribute. To select elements with a specific class, write a dot(.) character, followed by the class name.

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

## **CSS Universal Selector**

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

Eg:

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

## **CSS Grouping Selector**

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

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

```
h1 {  
  color: royalblue;  
}
```

```
h2 {  
  color: royalblue;  
}
```

```
p {  
  color: royalblue;  
}
```

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

To group selectors, separate each selector with a comma.

```
h1, h2, p {  
  color: royalblue;  
}
```



## CSS background:

You can set a background to an HTML element using the CSS background property.

The background property is a shorthand property for:

- background-color
- background-image
- background-position
- background-size
- background-repeat
- background-origin
- background-clip
- background-attachment

It does not matter if one of the values above are missing, e.g. background:#ff0000 url(smiley.gif); is allowed.

The syntax for the background property:

**background:** *bg-color bg-image position/bg-size bg-repeat bg-origin bg-clip bg-attachment* initial|inherit;

**background-color:** Specifies the background color to be used

**background-image:** Specifies ONE or MORE background images to be used

**background-position:** Specifies the position of the background images

**background-size:** Specifies the size of the background images

**background-repeat:** Specifies how to repeat the background images

**background-origin:** Specifies the positioning area of the background images

**background-clip:** Specifies the painting area of the background images

**background-attachment:** Specifies whether the background images are fixed or scrolls with the rest of the page

**initial** Set this property to its default value.

**inherit** Inherits this property from its parent element.

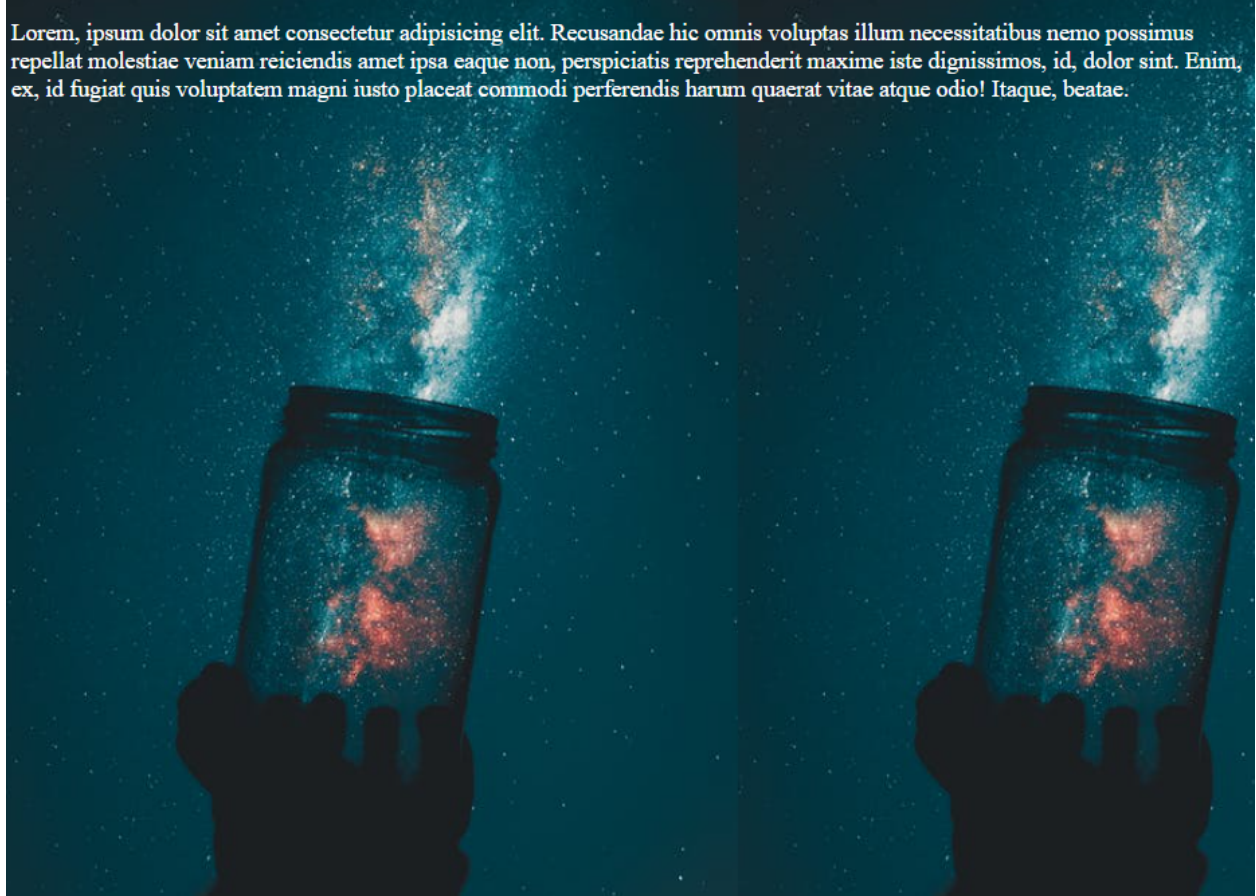
Eg:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background: lightblue

url("https://images.pexels.com/photos/1274260/pexels-photo-1274260.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1");
      }
      p {
        color: white;
      }
    </style>
  </head>
  <body>
    <p>
      Lorem, ipsum dolor sit amet consectetur adipisicing elit. Recusandae hic
      omnis voluptas illum necessitatibus nemo possimus repellat molestiae
      veniam reiciendis amet ipsa eaque non, perspiciatis reprehenderit maxime
      iste dignissimos, id, dolor sint. Enim, ex, id fugiat quis voluptatem
      magni iusto placeat commodi preferendis harum quaerat vitae atque odio!
      Itaque, beatae.
    </p>
  </body>
</html>
```

Output:

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Recusandae hic omnis voluptas illum necessitatibus nemo possimus repellat molestiae veniam reiciendis amet ipsa eaque non, perspiciatis reprehenderit maxime iste dignissimos, id, dolor sint. Enim, ex, id fugiat quis voluptatem magni iusto placeat commodi perferendis harum quaerat vitae atque odio! Itaque, beatae.



We can also just assign a color as a background to our HTML element, like so:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background: coral;
```

```

    }
    p {
        color: white;
    }
</style>
</head>
<body>
    <p>
        Lorem, ipsum dolor sit amet consectetur adipisicing elit. Recusandae hic
        omnis voluptas illum necessitatibus nemo possimus repellat molestiae
        veniam reiciendis amet ipsa eaque non, perspiciatis reprehenderit maxime
        iste dignissimos, id, dolor sint. Enim, ex, id fugiat quis voluptatem
        magni iusto placeat commodi perferendis harum quaerat vitae atque odio!
        Itaque, beatae.
    </p>
</body>
</html>

```

Output:

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Recusandae hic omnis
 voluptas illum necessitatibus nemo possimus repellat molestiae veniam reiciendis
 amet ipsa eaque non, perspiciatis reprehenderit maxime iste dignissimos, id, dolor
 sint. Enim, ex, id fugiat quis voluptatem magni iusto placeat commodi perferendis
 harum quaerat vitae atque odio! Itaque, beatae.

### CSS Borders:

Usage of Borders There is a magical box around every HTML tags. To see the BOX around every HTML tag apply this rule.

```

*{
    border-style: solid;
    border-color: red;
}

```

Always remember that every html tag has a box around it that you can control. Every element is a box which is arranged side by side or on top of each other. You can control that element by controlling that box.

**border-style:** solid; – Controls the style of the line

**border-color:** red; – Color of the border line

**border-width:** 4px; – Width of the Border Line

**border-top-style:** dashed; – Apply the style only to top line

**border-bottom-style:** dotted; – Apply the style only to bottom line

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

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background: coral;
      }
      p {
        color: white;
        border: 2px solid black;
        padding: 1rem;
      }
    </style>
  </head>
  <body>
    <p>
      Lorem, ipsum dolor sit amet consectetur adipiscing elit. Recusandae hic
      omnis voluptas illum necessitatibus nemo possimus repellat molestiae
      veniam reiciendis amet ipsa eaque non
    </p>
  </body>
</html>
```

Output:

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Recusandae hic omnis voluptas illum necessitatibus nemo possimus repellat molestiae veniam reiciendis amet ipsa eaque non

Another example:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Page Description" />
    <title>CSS Box - Border</title>
    <style type="text/css">
      div {
        border-style: solid;
        border-color: red;
        border-width: 4px;
        border-top-style: dashed;
        border-bottom-style: dotted;
        width: 50%;
      }
      .blue-border {
        border: 5px solid blue;
      }
      .green-border {
        border: 5px solid green;
      }
    </style>
  </head>
  <body>
    <div>Border Example with Diff line of 4px width and red in color.</div>
    <div class="blue-border">
```

Border Example with Solid line of 5px width and blue in color.

</div>

<div class="green-border">

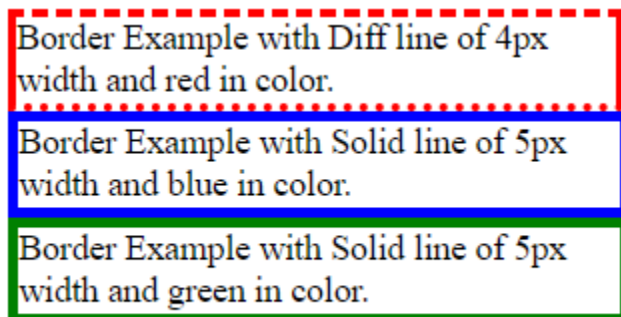
Border Example with Solid line of 5px width and green in color.

</div>

</body>

</html>

Output:

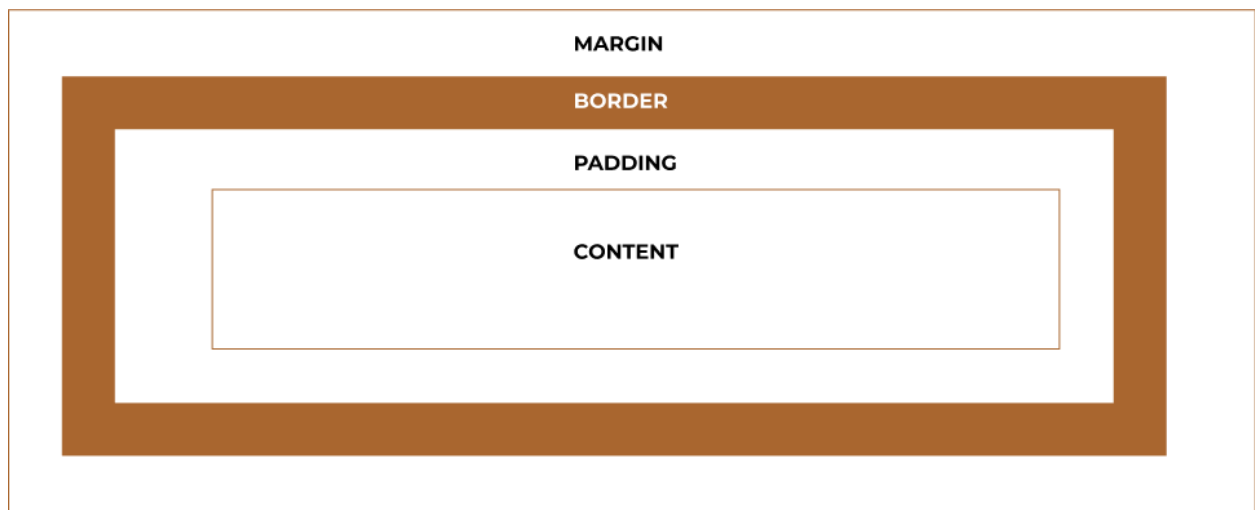


## CSS Box Model

All HTML elements can be considered as boxes.

In CSS, the term "box model" is used when talking about design and layout.

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



Explanation of the different parts:

**Content** - The content of the box, where text and images appear

**Padding** - Clears an area around the content. The padding is transparent

**Border** - A border that goes around the padding and content

**Margin** - Clears an area outside the border. The margin is transparent

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

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



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

Eg:

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

Here the calculation would be

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

The total width of an element should be calculated like this:

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

### **Margin**

With Margin, you can push the boxes around the HTML tag.

You can add Margin on four sides of the box TOP RIGHT BOTTOM LEFT



Margin can be used to add spaces between the boxes and push the boxes around.

Margin Properties:

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

Margin can be set in different ways:

Always think like a clock rotating in clockwise direction Top, Right, Bottom and Left. That is how the parameters settings are done as well.

```
h1 {  
  margin-top: 5px;  
  margin-right: 5px;  
  margin-bottom: 5px;  
  margin-left: 5px;  
}
```

Or

```
h1 {  
  margin: 5px; /*All the 4 sides will have margin of 5 px;*/
```

```
}
```

Or

```
h1 {  
  /* margin: top right bottom left */  
  margin: 5px 5px 5px 5px;  
}
```

Or

```
h1 {  
  /* margin: top left+right bottom */  
  margin: 5px 5px 5px;  
}
```

Consider an example page:

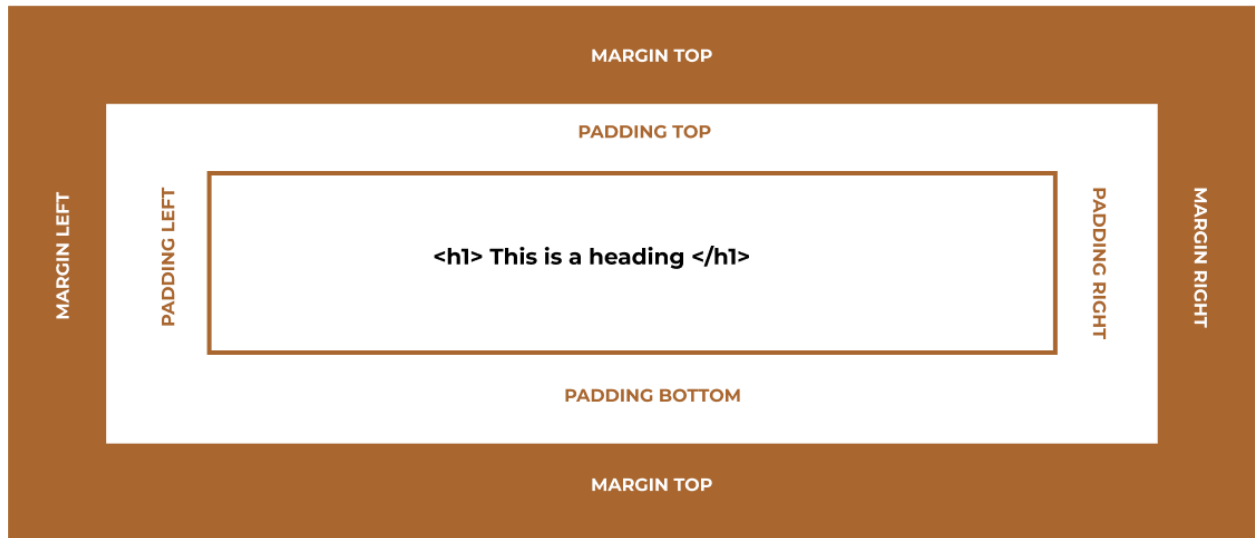
```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>CSS Box - Margin</title>  
    <style type="text/css">  
      .red-border {  
        border: 5px solid crimson;  
        margin-bottom: 25px;  
      }  
      .blue-border {  
        border: 5px solid royalblue;  
        margin-top: 25px;  
        margin-left: 20px;  
      }  
      .green-border {  
        border: 5px solid forestgreen;
```

```
        margin-left: 50px;
        margin-top: 25px;
    }
</style>
</head>
<body>
    <div class="red-border">Margin Example with Solid red line.</div>
    <div class="red-border" style="border-style: dashed">
        Margin Example with Dashed red line.
    </div>
    <div class="red-border" style="border-style: dotted">
        Margin Example with Dotted red line.
    </div>
    <div class="blue-border">Margin Example with Solid Blue line.</div>
    <div class="blue-border" style="border-style: dashed">
        Margin Example with Dashed Blue line.
    </div>
    <div class="blue-border" style="border-style: dotted">
        Margin Example with Dotted Blue line.
    </div>
    <div class="green-border">Margin Example with Solid Green line.</div>
    <div class="green-border" style="border-style: dashed">
        Margin Example with Dashed Green line.
    </div>
    <div class="green-border" style="border-style: dotted">
        Margin Example with Dotted Green line.
    </div>
</body>
</html>
```

## Padding

Padding is the extra space that you have inside the box .

Padding adds extra space inside the box to make it look bigger.



Padding properties are similar to margin properties the only different is that margin add space after the box and padding add space inside the box.

Padding adds space inside the box.

Margin adds space outside the box.

Consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>CSS Box - Padding</title>
    <style type="text/css">
      .red-border {
        border: 5px solid crimson;
        padding-top: 10px;
```

```
}
.blue-border {
  border: 5px solid royalblue;
  padding-left: 10px;
}
.green-border {
  border: 5px solid forestgreen;
  padding: 10px;
}
</style>
</head>
<body>
  <div class="red-border">
    Padding Top Example with Solid red line. No Margin.
  </div>
  <div class="red-border">
    Padding Top Example with Solid red line. No Margin.
  </div>
  <div class="red-border">
    Padding Top Example with Diff red line. No Margin.
  </div>
  <div class="blue-border">
    Padding Left Example with Solid blue line. No Margin.
  </div>
  <div class="blue- border">
    Padding Left Example with Solid blue line. No Margin.
  </div>
  <div class="blue-border">
    Padding Left Example with Solid blue line. No Margin.
  </div>
  <div class="green-border">
    Padding all sides Example with Solid green line. No Margin.
  </div>
  <div class="green-border">
    Padding all sides Example with Solid green line. No Margin.
  </div>
  <div class="green-border">
    Padding all sides Example with Solid green line. No Margin.
```

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

Output:

Padding Top Example with Solid red line. No Margin.

Padding Top Example with Solid red line. No Margin.

Padding Top Example with Diff red line. No Margin.

Padding Left Example with Solid blue line. No Margin.

Padding Left Example with Solid blue line. No Margin.

Padding Left Example with Solid blue line. No Margin.

Padding all sides Example with Solid green line. No Margin.

Padding all sides Example with Solid green line. No Margin.

Padding all sides Example with Solid green line. No Margin.

## Display property

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.

### Display: none;

display: none; is commonly used with JavaScript to hide and show elements without deleting and recreating them. Take a look at our last example on this page if you want to know how this can be achieved.

The <script> element uses display: none; as default.

## Override The Default Display Value

As mentioned, every element has a default display value. However, you can override this.

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.

A common example is making inline <li> elements for horizontal menus:

The following example displays <li> element as a block element. This is by default a block level element.

```
li {  
  display: inline;  
}
```

The following example displays <span> element as a block element. This is by default an inline element.



```
span {  
  display: block;  
}
```

### How to hide an element?

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:

```
h1.hidden {  
  display: none;  
}
```

**visibility:hidden**; also hides an element. However, the element will still take up the same space as before. The element will be hidden, but still affect the layout.

```
h1.hidden {  
  visibility: hidden;  
}
```

**display:** Specifies how an element should be displayed

**visibility:** Specifies whether or not an element should be visible

## CSS Text Alignment

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

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 following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left)

```
<!DOCTYPE html>
<html>
<head>
  <style>
    h1 {
      text-align: center;
    }

    h2 {
      text-align: left;
    }

    h3 {
      text-align: right;
    }
  </style>
</head>
<body>
  <h1>Heading aligned in the center</h1>
  <h2>Heading aligned to the left</h2>
  <h3>Heading aligned to the right</h3>
</body>
</html>
```

Output:

## Heading aligned in the center

### Heading aligned to the left

### Heading aligned to the right

#### CSS Floats

Usage of Floating Box float property to specify the element to float left or right or follow the existing flow of box arrangement.

float: none | left | right | initial | inherit;

**none** – This will follow the default floating of box and it also breaks the existing floating property set by its siblings or parent.

**left and right** – allows to element to float left or right

**initial** – Custom value can be given to float the element inherit – Follows the inheritance property from its parent style.

**clear: both** – property will clear the floating of boxes next to each other.

Consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Floating Example</title>
    <style type="text/css">
      .box {
        height: 50px;
        width: 50px;
        margin-bottom: 10px;
        margin-right: 10px;
      }
      .blue {
        background-color: #5bc0de;
        float: left;
      }
      .green {
        background-color: #5cb85c;
        float: right;
      }
    </style>
  </head>
  <body>
    <div class="box blue"></div>
    <div class="box blue"></div>
    <div class="box blue"></div>
    <div class="box blue"></div>
    <div class="box blue"></div>
    <div class="box green"></div>
    <div class="box green"></div>
    <div class="box green"></div>
  </body>
</html>
```

## Output:



## Floating Images

Text beside the images can be floated to right or left with the float property. Like the articles in the newspaper, you have the image and then text running sometime or left side or right side.

This is achieved by floating the image to left or right and then next text elements will float with it.

Code Snippet to clear floating:

```
.clearfix::after {  
  display: block;  
  clear: both;  
  height: 0;  
}
```

Look at the following code for examples of float:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Floating Example</title>
    <style type="text/css">
      h1 {
        text-align: center;
        text-decoration: underline;
      }
      img {
        margin-right: 20px;
      }
      .clearfix::after {
        /* content: ""; clear: both; display: table; */
        content: ".";
        display: block;
        clear: both;
        height: 0;
        visibility: hidden;
      }
      .float-left {
        float: left;
      }
      .float-right {
        float: right;
      }
    </style>
  </head>
  <body>
    <div class="clearfix">
      
</div>
<hr />
<div class="clearfix">
  
</div>
<hr />
<div class="clearfix">
  
  
  
```

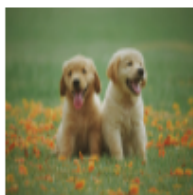
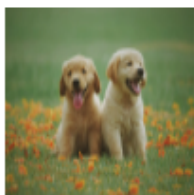
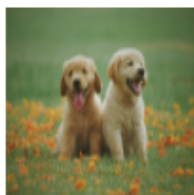
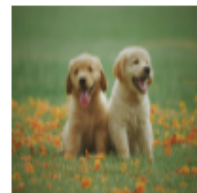
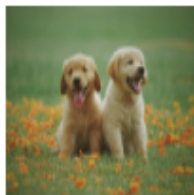
```
</div>
```

```
<hr />
```

```
</body>
```

```
</html>
```

Output:





## **Positioning Elements**

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

There are five different position values:

1. static
2. relative
3. fixed
4. absolute
5. sticky

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

### **Position Static:**

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

### **Position Relative**

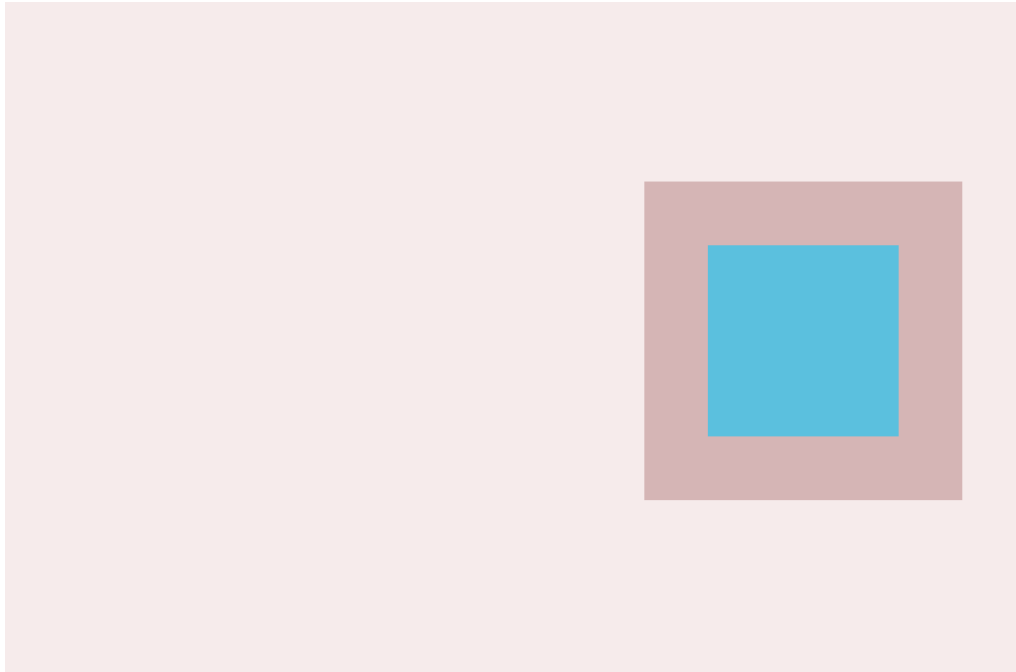
Relative is just to tell the browser to follow the current flow and make it relative to its parent. Wherever the parent element is, relative property will make the child stack them relative to the parent element.

**position: relative** helps to group the parent and child together and flow them next to each other.

Consider the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Fixed Bar Position at Bottom of Page</title>
    <style type="text/css">
      body {
        background: rgba(166, 54, 54, 0.1);
      }
      .container {
        background: rgba(163, 99, 99, 0.4);
        height: 250px;
        width: 250px;
        margin-left: 500px;
        margin-top: 150px;
        position: relative;
      }
      .box {
        height: 150px;
        width: 150px;
        background-color: #5bc0de;
        top: 50px;
        left: 50px;
        position: absolute;
      }
    </style>
  </head>
  <body>
    <div class="container"><div class="box"></div></div>
  </body>
</html>
```

Output:



**Position Fixed:** fixed property on any element can be fixed at any position of the screen.

Once the element property is mentioned as position: fixed then using the following property the element can be moved anywhere on the page.

- top
- right
- bottom
- left

Elements will be fixed and will not move on the page.

It is good when you want to show a header bar or footer announcement of some pages.

Consider the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Fixed Bar Position at Bottom of Page</title>

<style type="text/css">
  body,
  html {
    height: 100%;
  }
  p {
    margin: 0 auto;
    max-width: 600px;
    margin-top: 40px;
    line-height: 1.5;
    font-size: 2rem;
  }
  body {
    font-family: Georgia, serif;
  }
  h1 {
    text-align: center;
  }
  .announcement-bottom {
    background-color: red;
    opacity: 0.85;
    padding: 20px;
    color: rgba(255, 255, 255, 0.9);
    position: fixed;
    bottom: 0; /* Key Property */
    left: 0;
    right: 0;
    text-align: center;
  }
  .announcement-top {
    background: #d4765d;
    position: fixed;
    top: 0; /* Key Property */
    left: 0;
    right: 0; /* z-index: 2; */ /* opacity: .85; */
    text-align: center;
    color: white;
    padding: 20px;
  }
</style>
```

```

</head>
<body>
  <div class="announcement-top">Fixed Header</div>
  <div class="announcementbottom">Fixed Element at end of the Page</div>
  <br /><br />
  <h1>Position: Fixed</h1>
  <p>
    What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and
    typesetting industry. Lorem Ipsum has been the industry's standard dummy
    text ever since the 1500s, when an unknown printer took a galley of type
    and scrambled it to make a type specimen book. It has survived not only
    five centuries, but also the leap into electronic typesetting, remaining
    essentially unchanged. It was popularised in the 1960s with the release of
    Letraset sheets containing Lorem Ipsum passages, and more recently with
    desktop publishing software like Aldus PageMaker including versions of
    Lorem Ipsum. Why do we use it?
  </p>
</body>
</html>

```

## Output

Fixed Header	Fixed Header
<p><b>Position: Fixed</b></p> <p>What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including</p>	<p>What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum. Why do we use it?</p>

## Position Absolute

Positions allow the element to move around the pages.

By default without CSS, browsers will lay all the elements one after the other.

To arrange them into the correct position we use this position declaration.

With position: absolute we break the running flow of browser placement and position based on the body position again.

Example:

```
<h1>This is h1</h1>
```

```
<p>This is paragraph</p>
```

Without CSS, they both appear one after the other. h1 tags starts from the absolute position the parent position and <p> tag will follow the flow and sit after <h1>

If you want to break the <p> flow and want to position somewhere else and follow the new location then we set the position of that element as absolute. Then the browser will start placing them again from top.

**position: absolute** tells the browser to take this element out of the flow and start putting from the absolute body position not the current flow.

Eg:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Fixed Bar Position at Bottom of Page</title>
```

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

```
.container {
```

```
background: #d4765d;
```

```
height: 250px;
```

```
width: 250px;
```

```
margin-left: 300px;
```

```
margin-top: 150px;
```

```
}
```

```
.box {
```

```
height: 150px;
```

```
width: 150px;
background-color: rgba(0, 0, 0, 0.4);
top: 50px;
left: 50px;
position: absolute;
}
</style>
</head>
<body>
  <div class="container"><div class="box"></div></div>
</body>
</html>
```

Output:

**Position Sticky:**

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

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

Consider the following code:

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



```
<style>
div.sticky {
  position: -webkit-sticky;
  position: sticky;
  top: 0;
  padding: 5px;
  background-color: #cae8ca;
  border: 2px solid #4caf50;
}
</style>
</head>
<body>
<p>
  Try to <b>scroll</b> inside this frame to understand how sticky
  positioning works.
</p>

<div class="sticky">I am sticky!</div>

<div style="padding-bottom: 2000px">
<p>
  In this example, the sticky element sticks to the top of the page (top:
  0), when you reach its scroll position.
</p>
<p>Scroll back up to remove the stickyness.</p>
<p>
  Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum
  definitiones no quo, maluisset concludaturque et eum, altera fabulas ut
  quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert
  laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no
  molestiae voluptatibus.
</p>
<p>
  Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum
  definitiones no quo, maluisset concludaturque et eum, altera fabulas ut
  quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert
  laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no
  molestiae voluptatibus.
```

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

Output:

I am sticky!

efficiantur his ad. Eum no molestiae voluptatibus.

Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, malisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

## CSS Pseudo Classes

Pseudo Class Selector are predefined class that are available to use for the HTML tags.

For Example, You want to change the color of link when someone mouse over the link. This is done by a predefined class for the HTML tags called as Pseudo Class Selectors.

<a> anchor tags have hover, visited pseudo class that we can use. Pseudo class are separated with ":" along with the HTML tags.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      a:hover { color: red; }
    </style>
  </head>
  <body>
    <a href="#">Hover on me and I will turn red in color</a>
  </body>
</html>
```

Output:

---

Hover on me and I will turn red in color

You don't have to mention the Pseudo class to the HTML tags attribute. These are the properties of the HTML tags that you are changing.

Another example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pseudo Class Selector Notation</title>
    <style>
      a:link {
        color: blue;
      } /* Visited Link */
      a:visited {
        color: gray;
      } /* On Mouse Over Link */
      a:hover {
        color: red;
      }
    </style>
  </head>
  <body>
    <a href="#">Link</a>
  </body>
</html>
```

```

    } /* Active link that is clicked */
    a:active {
        color: green;
    }
</style>
</head>
<body>
    <h1>Pseudo Class Selector - HTML TAG</h1>
    <a href="#1">Home Page</a> <a href="#2">Contact</a> <a href="#3">About</a>
</body>
</html>

```

Output:

## Pseudo Class Selector - HTML TAG

[Home Page](#) [Contact](#) [About](#)

### Pseudo Elements

Pseudo Elements Selector are rules that you want to add right after the element is closed. Irrespective of what is there after the tag. For Example, You want to add "!!!" after every paragraph ending. Then you can use the Pseudo Element Selector.

Pseudo class are separated with "::" along with the HTML tags

#### Example:

```
p::after { content: "!!!"; }
```

This will add "!!!" after the every paragraph tag.

p::before will apply the before the tag.

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <style>
      p::after {
        content: "!!!";
      }
    </style>
  </head>
  <body>
    <p>See the !!! right after the paragraph</p>
  </body>
</html>
```

Output:

See the !!! right after the paragraph!!!

## **Responsive Web Design**

Responsive web design makes your web page look good on all devices.

Responsive web design uses only HTML and CSS. It does not use JavaScript.

## **Designing For The Best Experience For All Users**

Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device. Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:

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

### **Practices we should use**

- Set the viewport / scale
- Use fluid widths as opposed to fixed
- Media queries – Different css styling for different screen sizes
- Rem units over px
- Mobile first method

## **Viewport**

The viewport is the user's visible area of a web page.

The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

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

## **Setting the Viewport**

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

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

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

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

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

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

## **Grid View**

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

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

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

## Media Query

Media query is a CSS technique introduced in CSS3.

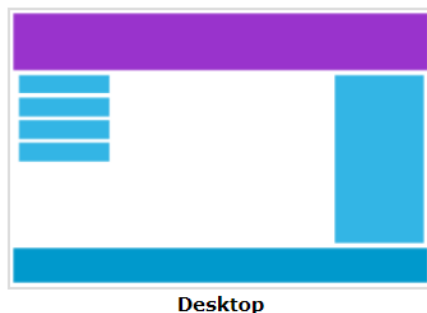
It uses the @media rule to include a block of CSS properties only if a certain condition is true.

Eg:

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

## Adding a breakpoint

We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.



Consider the following code:

When the screen (browser window) gets smaller than 768px, each column should have a width of 100%:

```
<!DOCTYPE html>
```



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

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

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

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

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

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

    .menu li {
      padding: 8px;
      margin-bottom: 7px;
    }
```

```
background-color: #33b5e5;
color: #ffffff;
box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0, 0.24);
}
```

```
.aside {
background-color: #33b5e5;
padding: 15px;
color: #ffffff;
text-align: center;
font-size: 14px;
box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0, 0.24);
}
```

```
.footer {
background-color: #0099cc;
color: #ffffff;
text-align: center;
font-size: 12px;
padding: 15px;
}
```

```
/* For desktop: */
```

```
.col-1 {
width: 8.33%;
}
```

```
.col-2 {
width: 16.66%;
}
```

```
.col-3 {
width: 25%;
}
```

```
.col-4 {
width: 33.33%;
```

```
}
```

```
.col-5 {  
  width: 41.66%;  
}
```

```
.col-6 {  
  width: 50%;  
}
```

```
.col-7 {  
  width: 58.33%;  
}
```

```
.col-8 {  
  width: 66.66%;  
}
```

```
.col-9 {  
  width: 75%;  
}
```

```
.col-10 {  
  width: 83.33%;  
}
```

```
.col-11 {  
  width: 91.66%;  
}
```

```
.col-12 {  
  width: 100%;  
}
```

```
@media only screen and (max-width: 768px) {  
  [class*="col-"] {  
    width: 100%;  
  }  
}
```

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

<body>

<div class="header">
  <h1>City</h1>
</div>

<div class="row">
  <div class="col-3 menu">
    <ul>
      <li>The Flight</li>
      <li>The City</li>
      <li>The Island</li>
      <li>The Food</li>
    </ul>
  </div>

  <div class="col-6">
    <h1>The City</h1>
    <p>Hyderabad is the capital of Telangana</p>
  </div>

  <div class="col-3 right">
    <div class="aside">
      <h2>What?</h2>
      <p>Hyderabad is a city in Telangana</p>
      <h2>Where?</h2>
      <p>Hyderabad is the capital of the Telangana state.</p>
      <h2>How?</h2>
      <p>You can reach Hyderabad airport from all over India.</p>
    </div>
  </div>

</div>

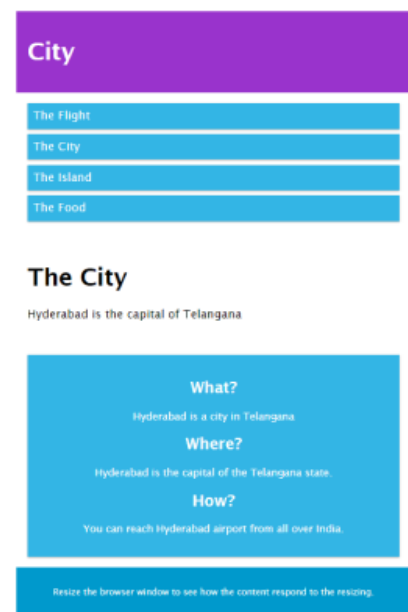
<div class="footer">
```

<p>Resize the browser window to see how the content respond to the resizing.</p>  
</div>

</body>

</html>

Output:



### **Commonly used media queries**

/\* Extra small devices (phones, 600px and down) \*/

@media only screen and (max-width: 600px) {...}

/\* Small devices (portrait tablets and large phones, 600px and up) \*/

@media only screen and (min-width: 600px) {...}

/\* Medium devices (landscape tablets, 768px and up) \*/

@media only screen and (min-width: 768px) {...}

/\* Large devices (laptops/desktops, 992px and up) \*/

@media only screen and (min-width: 992px) {...}

/\* Extra large devices (large laptops and desktops, 1200px and up) \*/

@media only screen and (min-width: 1200px) {...}

### **Hide Elements With Media Queries**

@media only screen and (max-width: 600px) {

div.example {  
display: none;

}

}

## Responsive units

CSS has several different units for expressing a length.

Many CSS properties take "length" values, such as width, margin, padding, font-size, etc.

Length is a number followed by a length unit, such as 10px, 2em, etc.

## Absolute Lengths

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

- cm    centimeters
- mm   millimeters
- in    inches (1in = 96px = 2.54cm)
- px \*   pixels (1px = 1/96th of 1in)
- pt    points (1pt = 1/72 of 1in)
- pc    picas (1pc = 12 pt)

## Relative Lengths

Relative length units specify a length relative to another length property. Relative length units scale better between different rendering mediums.

- em    Relative to the font-size of the element (2em means 2 times the size of the current font)
- ex    Relative to the x-height of the current font (rarely used)
- ch    Relative to the width of the "0" (zero)
- rem   Relative to font-size of the root element
- vw    Relative to 1% of the width of the viewport\*

- `vh` Relative to 1% of the height of the viewport\*
- `vmin` Relative to 1% of viewport's\* smaller dimension
- `vmax` Relative to 1% of viewport's\* larger dimension
- `%` Relative to the parent element

## **CSS Flexbox**

Introduced in CSS3, flexbox is a method for defining how components of a page behave across different devices and screen sizes. Named for its flexible nature, flexbox offers several benefits over old layout methods, including dynamic rearranging, alignment, direction, and container fit.

In an increasingly mobile world, flexbox allows a developer to gain full control over the behavior of page elements indeterminate of what device they are being accessed from.

## **Flexbox Overview**

There are three main components of flexbox:

- flex container — flex items are contained inside a flex container
- flex items — elements whose behavior is being controlled
- direction of flow — controls the direction of flex items

## **Create a Flex Container**

To define and access a container as a flex container you can use `display: flex;`. If no additional rules are set, all direct children will be considered flex items and will be laid horizontally, from left to right. The width of flex items automatically adjusts to fit inside the container.



```
<div class = flexcontainer>
<div class = flexitem1 style=background-color:lightblue;> test1 </div>
<div class = flexitem2 style=background-color:lightgreen;> test2 </div>
<div class = flexitem3 style=background-color:lightslategray;"> test3 </div>
</div>
.flexcontainer {
    display: flex;
}
```



### Defining Flex Direction

To put all flex items in one column, add `flex-direction: column;` to your CSS.

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

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



test1

test2

test3

To reverse the column order of flex items, add `flex-direction: column-reverse;` to your CSS.

```
.flexcontainer {  
  display: flex;  
  flex-direction: column-reverse;  
}
```



test3

test2

test1

To reverse the row order of flex items, add `flex-direction: row-reverse;` to your CSS.

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



test3test2test1

### **Align and Justify Flex Items**

If you want flex items aligned to the right:

```
.flexcontainer {  
  display: flex;  
  justify-content: flex-end;  
}
```



test1test2test3

If you want flex items centered:

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

A horizontal container with three items: 'test1' on a light blue background, 'test2' on a light green background, and 'test3' on a dark blue background. The items are centered together within the container.

test1test2test3

**If you want flex items separated**

```
.flexcontainer {  
  display: flex;  
  justify-content: space-between;  
}
```

A horizontal container with three items: 'test1' on a light blue background, 'test2' on a light green background, and 'test3' on a dark blue background. The items are separated by equal space, with 'test1' on the left, 'test2' in the middle, and 'test3' on the right.

test1test2test3

**To center flex items both horizontally and vertically:**

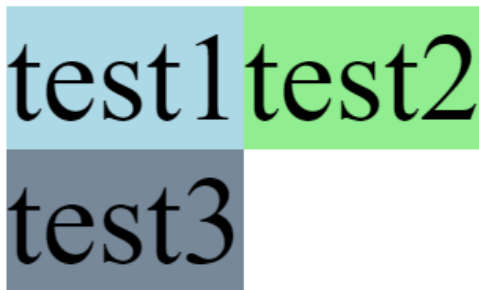
```
.flexcontainer {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
}
```

## Wrapping

Resized to fit into one column or one row when a flex container is not large enough, flex items are not able to wrap by default. By using `flex-wrap: wrap;`, any flex items causing overflow will be wrapped to the next line.

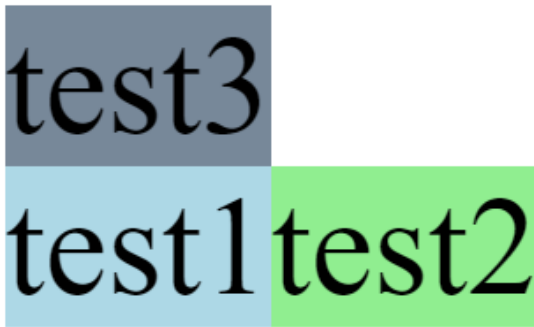
To allow wrapping:

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



**To allow reverse wrapping (overflow item(s) are sent to the line above):**

```
.flexcontainer {  
  display: flex;  
  flex-wrap: wrap-reverse;  
}
```



To control the positioning of flex items when there is wrapping (replace \* with stretch, space-around, space-between, center, flex-end, or flex-start):

```
.flexcontainer {  
  display: flex;  
  flex-wrap: wrap;  
  align-content: *;  
}
```

### **Expanding Flex Items**

When there is leftover space within a flex container, flex-grow can be used to dictate how each item fits in the remaining space.

To grow a flex item (default size is 0):

```
.flexcontainer {  
  display: flex;  
}  
.flexitem1 {  
  flex-grow: 1;  
  border: 3px solid;  
}
```

test1 test2 test3

## Forms & Inputs

### Form text

Block-level or inline-level form text can be created using `.form-text`. Form text below inputs can be styled with `.form-text`. If a block-level element will be used, a top margin is added for easy spacing from the inputs above.

```
<label for="inputPassword5" class="form-label">Password</label>
<input type="password" id="inputPassword5" class="form-control"
aria-describedby="passwordHelpBlock">
<div id="passwordHelpBlock" class="form-text">
  Your password must be 8-20 characters long, contain letters and numbers, and must not
  contain spaces, special characters, or emoji.
</div>
```

Password

Your password must be 8-20 characters long, contain letters and numbers, and must not contain spaces, special characters, or emoji.

### Disabled forms

Add the disabled boolean attribute on an input to prevent user interactions and make it appear lighter.

Add the disabled attribute to a `<fieldset>` to disable all the controls within. Browsers treat all native form controls (`<input>`, `<select>`, and `<button>` elements) inside a `<fieldset disabled>` as disabled, preventing both keyboard and mouse interactions on them.

```
<form>
  <fieldset disabled>
    <legend>Disabled fieldset example</legend>
    <div class="mb-3">
```



```

<label for="disabledTextInput" class="form-label">Disabled input</label>
<input type="text" id="disabledTextInput" class="form-control" placeholder="Disabled
input">
</div>
<div class="mb-3">
  <label for="disabledSelect" class="form-label">Disabled select menu</label>
  <select id="disabledSelect" class="form-select">
    <option>Disabled select</option>
  </select>
</div>
<div class="mb-3">
  <div class="form-check">
    <input class="form-check-input" type="checkbox" id="disabledFieldsetCheck" disabled>
    <label class="form-check-label" for="disabledFieldsetCheck">
      Can't check this
    </label>
  </div>
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</fieldset>
</form>

```

## Disabled fieldset example

Disabled input

Disabled input

Disabled select menu

Disabled select

☐ Can't check this

Submit

## Input group

Easily extend form controls by adding text, buttons, or button groups on either side of textual inputs, custom selects, and custom file inputs. Place one add-on or button on either side of an input. You may also place one on both sides of an input. Remember to place `<label>`s outside the input group.

```
<div class="input-group mb-3">
  <span class="input-group-text" id="basic-addon1">@</span>
  <input type="text" class="form-control" placeholder="Username" aria-label="Username"
aria-describedby="basic-addon1">
</div>
```

```
<div class="input-group mb-3">
  <input type="text" class="form-control" placeholder="Recipient's username" aria-label="Recipient's
username" aria-describedby="basic-addon2">
  <span class="input-group-text" id="basic-addon2">@example.com</span>
</div>
```

```
<label for="basic-url" class="form-label">Your vanity URL</label>
<div class="input-group mb-3">
  <span class="input-group-text" id="basic-addon3">https://example.com/users/</span>
  <input type="text" class="form-control" id="basic-url" aria-describedby="basic-addon3">
</div>
```

```
<div class="input-group mb-3">
  <span class="input-group-text">$</span>
  <input type="text" class="form-control" aria-label="Amount (to the nearest dollar)">
  <span class="input-group-text">.00</span>
</div>
```

```
<div class="input-group mb-3">
  <input type="text" class="form-control" placeholder="Username" aria-label="Username">
  <span class="input-group-text">@</span>
  <input type="text" class="form-control" placeholder="Server" aria-label="Server">
</div>
```

```
<div class="input-group">
  <span class="input-group-text">With textarea</span>
  <textarea class="form-control" aria-label="With textarea"></textarea>
</div>
```

@ Username

Recipient's username
@example.com

Your vanity URL

https://example.com/users/

\$ .00

Username @ Server

With textarea

## Wrapping

Input groups wrap by default via flex-wrap: wrap in order to accommodate custom form field validation within an input group. You may disable this with .flex-nowrap.

```
<div class="input-group flex-nowrap">
  <span class="input-group-text" id="addon-wrapping">@</span>
  <input type="text" class="form-control" placeholder="Username" aria-label="Username"
    aria-describedby="addon-wrapping">
</div>
```

@ Username

## Sizing

Add the relative form sizing classes to the .input-group itself and contents within will automatically resize—no need for repeating the form control size classes on each element.

Sizing on the individual input group elements isn't supported.

```

<div class="input-group input-group-sm mb-3">
  <span class="input-group-text" id="inputGroup-sizing-sm">Small</span>
  <input type="text" class="form-control" aria-label="Sizing example input"
aria-describedby="inputGroup-sizing-sm">
</div>

```

```

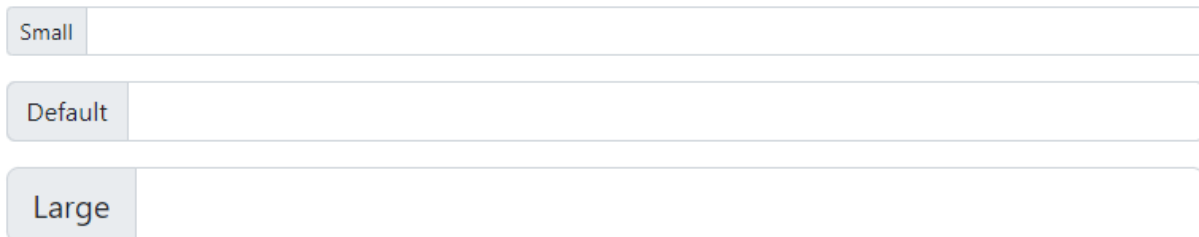
<div class="input-group mb-3">
  <span class="input-group-text" id="inputGroup-sizing-default">Default</span>
  <input type="text" class="form-control" aria-label="Sizing example input"
aria-describedby="inputGroup-sizing-default">
</div>

```

```

<div class="input-group input-group-lg">
  <span class="input-group-text" id="inputGroup-sizing-lg">Large</span>
  <input type="text" class="form-control" aria-label="Sizing example input"
aria-describedby="inputGroup-sizing-lg">
</div>

```



The image displays three examples of Bootstrap input groups. Each example consists of a light gray button-like element on the left and a white text input field on the right. The first example is labeled 'Small', the second 'Default', and the third 'Large'. The 'Large' example shows a significantly larger font size for both the label and the input field compared to the others.

## Checkboxes and radios

Place any checkbox or radio option within an input group's addon instead of text. It is recommended to add `.mt-0` to the `.form-check-input` when there's no visible text next to the input.

```

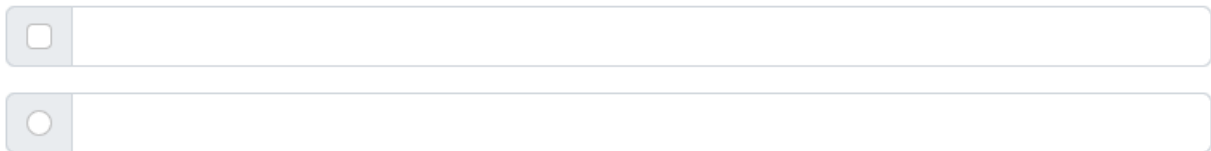
<div class="input-group mb-3">
  <div class="input-group-text">
    <input class="form-check-input mt-0" type="checkbox" value="" aria-label="Checkbox for following text
input">
  </div>
  <input type="text" class="form-control" aria-label="Text input with checkbox">
</div>

```

```

<div class="input-group">
  <div class="input-group-text">
    <input class="form-check-input mt-0" type="radio" value="" aria-label="Radio button for following text
input">
  </div>
  <input type="text" class="form-control" aria-label="Text input with radio button">
</div>

```



## Button addons

```

<div class="input-group mb-3">
  <button class="btn btn-outline-secondary" type="button"
id="button-addon1">Button</button>
  <input type="text" class="form-control" placeholder="" aria-label="Example text with
button addon" aria-describedby="button-addon1">
</div>

```

```

<div class="input-group mb-3">
  <input type="text" class="form-control" placeholder="Recipient's username"
aria-label="Recipient's username" aria-describedby="button-addon2">
  <button class="btn btn-outline-secondary" type="button"
id="button-addon2">Button</button>
</div>

```

```

<div class="input-group mb-3">
  <button class="btn btn-outline-secondary" type="button">Button</button>
  <button class="btn btn-outline-secondary" type="button">Button</button>
  <input type="text" class="form-control" placeholder="" aria-label="Example text with two
button addons">
</div>

```

```

<div class="input-group">
  <input type="text" class="form-control" placeholder="Recipient's username"
  aria-label="Recipient's username with two button addons">
  <button class="btn btn-outline-secondary" type="button">Button</button>
  <button class="btn btn-outline-secondary" type="button">Button</button>
</div>

```

The image shows four visual examples of Bootstrap input groups with buttons:

- Example 1: A button labeled "Button" is positioned to the left of a text input field.
- Example 2: A text input field with the placeholder text "Recipient's username" is followed by a button labeled "Button" on the right.
- Example 3: Two buttons labeled "Button" are positioned to the left of a text input field.
- Example 4: A text input field with the placeholder text "Recipient's username" is followed by two buttons labeled "Button" on the right.

## Buttons with dropdowns

```

<div class="input-group">
  <button class="btn btn-outline-secondary dropdown-toggle" type="button"
  data-bs-toggle="dropdown" aria-expanded="false">Dropdown</button>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="#">Action before</a></li>
    <li><a class="dropdown-item" href="#">Another action before</a></li>
    <li><a class="dropdown-item" href="#">Something else here</a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item" href="#">Separated link</a></li>
  </ul>
  <input type="text" class="form-control" aria-label="Text input with 2 dropdown buttons">
  <button class="btn btn-outline-secondary dropdown-toggle" type="button"
  data-bs-toggle="dropdown" aria-expanded="false">Dropdown</button>
  <ul class="dropdown-menu dropdown-menu-end">
    <li><a class="dropdown-item" href="#">Action</a></li>
    <li><a class="dropdown-item" href="#">Another action</a></li>
    <li><a class="dropdown-item" href="#">Something else here</a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item" href="#">Separated link</a></li>
  </ul>
</div>

```



## Custom forms

Input groups include support for custom selects and custom file inputs. Browser default versions of these are not supported.

```
<div class="input-group mb-3">
  <label class="input-group-text" for="inputGroupSelect01">Options</label>
  <select class="form-select" id="inputGroupSelect01">
    <option selected>Choose...</option>
    <option value="1">One</option>
    <option value="2">Two</option>
    <option value="3">Three</option>
  </select>
</div>
```

```
<div class="input-group mb-3">
  <select class="form-select" id="inputGroupSelect02">
    <option selected>Choose...</option>
    <option value="1">One</option>
    <option value="2">Two</option>
    <option value="3">Three</option>
  </select>
  <label class="input-group-text" for="inputGroupSelect02">Options</label>
</div>
```

```
<div class="input-group mb-3">
  <button class="btn btn-outline-secondary" type="button">Button</button>
```

```

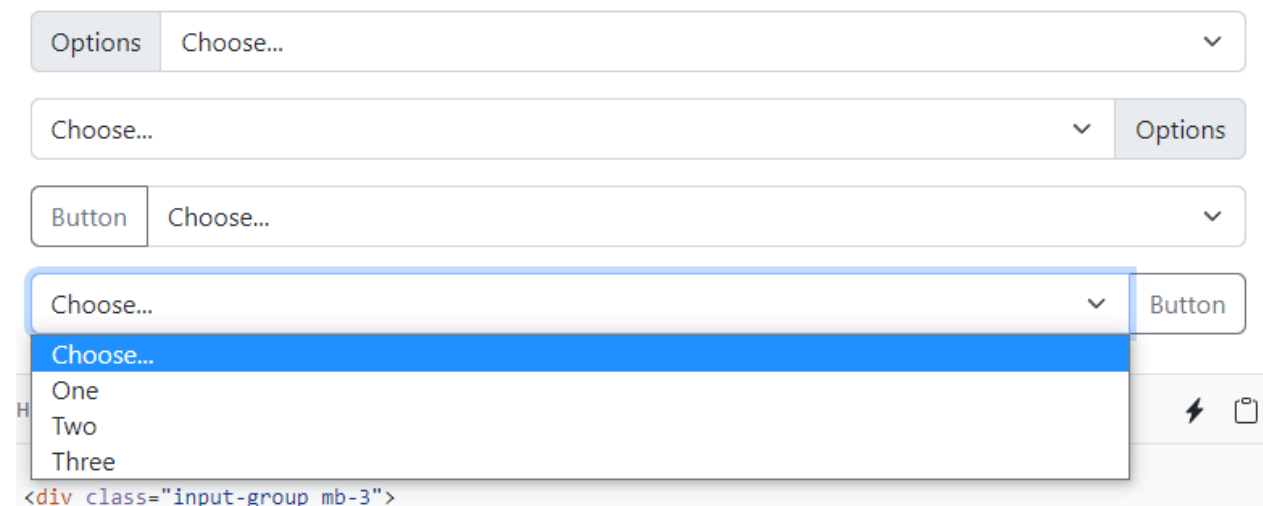
<select class="form-select" id="inputGroupSelect03" aria-label="Example select with button
addon">
  <option selected>Choose...</option>
  <option value="1">One</option>
  <option value="2">Two</option>
  <option value="3">Three</option>
</select>
</div>

```

```

<div class="input-group">
  <select class="form-select" id="inputGroupSelect04" aria-label="Example select with button
addon">
    <option selected>Choose...</option>
    <option value="1">One</option>
    <option value="2">Two</option>
    <option value="3">Three</option>
  </select>
  <button class="btn btn-outline-secondary" type="button">Button</button>
</div>

```





## Tables

Due to the widespread use of `<table>` elements across third-party widgets like calendars and date pickers, Bootstrap's tables are opt-in. Add the base class `.table` to any `<table>`, then extend with our optional modifier classes or custom styles. All table styles are not inherited in Bootstrap, meaning any nested tables can be styled independent from the parent.

Using the most basic table markup, here's how `.table`-based tables look in Bootstrap.

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Jacob</td>
      <td>Thornton</td>
      <td>@fat</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td colspan="2">Larry the Bird</td>
      <td>@twitter</td>
    </tr>
  </tbody>
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

## Accented tables

### Striped rows

Use `.table-striped` to add zebra-stripping to any table row within the `<tbody>`.

```
<table class="table table-striped">
...
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

### Striped columns

Use `.table-striped-columns` to add zebra-stripping to any table column.

```
<table class="table table-striped-columns">
...
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

### Hoverable rows

Add `.table-hover` to enable a hover state on table rows within a `<tbody>`.

```
<table class="table table-hover">
...
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

## Bordered table

Add `.table-bordered` for borders on all sides of the table and cells.

```
<table class="table table-bordered">
```

...

```
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

## Tables without borders

Add `.table-borderless` for a table without borders

```
<table class="table table-borderless">
```

...

```
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

## Small tables

Add `.table-sm` to make any `.table` more compact by cutting all cell padding in half.

```
<table class="table table-sm">
```

```
...
```

```
</table>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

## Table group dividers

Add a thicker border, darker between table groups—`<thead>`, `<tbody>`, and `<tfoot>`—with `.table-group-divider`. Customize the color by changing the `border-top-color` (which we don't currently provide a utility class for at this time).

```
<table class="table">
```

```
<thead>
```

```
<tr>
```

```
<th scope="col">#</th>
```

```
<th scope="col">First</th>
```

```
<th scope="col">Last</th>
```

```
<th scope="col">Handle</th>
```

```
</tr>
```

```
</thead>
```

```
<tbody class="table-group-divider">
```

```
<tr>
```

```
<th scope="row">1</th>
```

```
<td>Mark</td>
```

```
<td>Otto</td>
```

```
<td>@mdo</td>
```

```
</tr>
```

```

<tr>
  <th scope="row">2</th>
  <td>Jacob</td>
  <td>Thornton</td>
  <td>@fat</td>
</tr>
<tr>
  <th scope="row">3</th>
  <td colspan="2">Larry the Bird</td>
  <td>@twitter</td>
</tr>
</tbody>
</table>

```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

## Nesting

Border styles, active styles, and table variants are not inherited by nested tables.

```

<table class="table table-striped">
  <thead>
    ...
  </thead>
  <tbody>
    ...

```

```

<tr>
  <td colspan="4">
    <table class="table mb-0">
      ...
    </table>
  </td>
</tr>
...
</tbody>
</table>

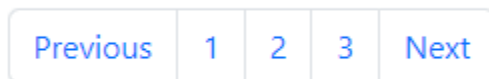
```

#	First	Last	Handle
1	Mark	Otto	@mdo
<div> <div>Header</div> <div>Header</div> <div>Header</div> </div>			
A		First	Last
B		First	Last
C		First	Last
3	Larry	the Bird	@twitter

## Pagination

We use a large block of connected links for our pagination, making links hard to miss and easily scalable—all while providing large hit areas. Pagination is built with list HTML elements so screen readers can announce the number of available links. Use a wrapping `<nav>` element to identify it as a navigation section to screen readers and other assistive technologies.

```
<nav aria-label="Page navigation example">
  <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="#">Next</a></li>
  </ul>
</nav>
```



## Working with icons

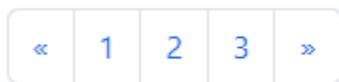
```
<nav aria-label="Page navigation example">
  <ul class="pagination">
    <li class="page-item">
      <a class="page-link" href="#" aria-label="Previous">
        <span aria-hidden="true">&laquo;</span>
      </a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li 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="#" aria-label="Next">
    <span aria-hidden="true">&raquo;</span>
  </a>
</li>
</ul>
</nav>

```



### Disabled and active states

Pagination links are customizable for different circumstances. Use `.disabled` for links that appear un-clickable and `.active` to indicate the current page.

While the `.disabled` class uses `pointer-events: none` to try to disable the link functionality of `<a>`s, that CSS property is not yet standardized and doesn't account for keyboard navigation.

```

<nav aria-label="...">
  <ul class="pagination">
    <li class="page-item disabled">
      <a class="page-link">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item active" aria-current="page">
      <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="#">Next</a>
    </li>
  </ul>
</nav>

```



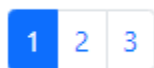
## Sizing

For smaller paginations, add `.pagination-lg` or `.pagination-sm` for additional sizes.

```
<nav aria-label="...">
  <ul class="pagination pagination-lg">
    <li class="page-item active" aria-current="page">
      <span class="page-link">1</span>
    </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>
  </ul>
</nav>
```



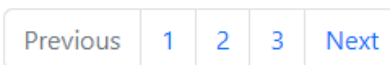
```
<nav aria-label="...">
  <ul class="pagination pagination-sm">
    <li class="page-item active" aria-current="page">
      <span class="page-link">1</span>
    </li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
  </ul>
</nav>
```



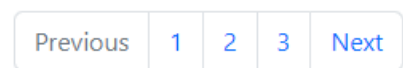
## Alignment

Change the alignment of pagination components with flexbox utilities. For example, with `.justify-content-center`:

```
<nav aria-label="Page navigation example">
  <ul class="pagination justify-content-center">
    <li class="page-item disabled">
      <a class="page-link">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="#">Next</a>
    </li>
  </ul>
</nav>
```



```
<nav aria-label="Page navigation example">
  <ul class="pagination justify-content-end">
    <li class="page-item disabled">
      <a class="page-link">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="#">Next</a>
    </li>
  </ul>
</nav>
```



## Alerts

Alerts are available for any length of text, as well as an optional close button. For proper styling, use one of the eight required contextual classes (e.g., `.alert-success`).

```
<div class="alert alert-primary" role="alert">
  A simple primary alert—check it out!
</div>
<div class="alert alert-secondary" role="alert">
  A simple secondary alert—check it out!
</div>
<div class="alert alert-success" role="alert">
  A simple success alert—check it out!
</div>
<div class="alert alert-danger" role="alert">
  A simple danger alert—check it out!
</div>
<div class="alert alert-warning" role="alert">
  A simple warning alert—check it out!
</div>
<div class="alert alert-info" role="alert">
  A simple info alert—check it out!
</div>
<div class="alert alert-light" role="alert">
  A simple light alert—check it out!
</div>
<div class="alert alert-dark" role="alert">
  A simple dark alert—check it out!</div>
```

A simple primary alert—check it out!

A simple secondary alert—check it out!

A simple success alert—check it out!

A simple danger alert—check it out!

A simple warning alert—check it out!

A simple info alert—check it out!

A simple light alert—check it out!

A simple dark alert—check it out!

## Spinners

Bootstrap “spinners” can be used to show the loading state in your projects. They’re built only with HTML and CSS, meaning you don’t need any JavaScript to create them. You will, however, need some custom JavaScript to toggle their visibility. Their appearance, alignment, and sizing can be easily customized with our amazing utility classes.

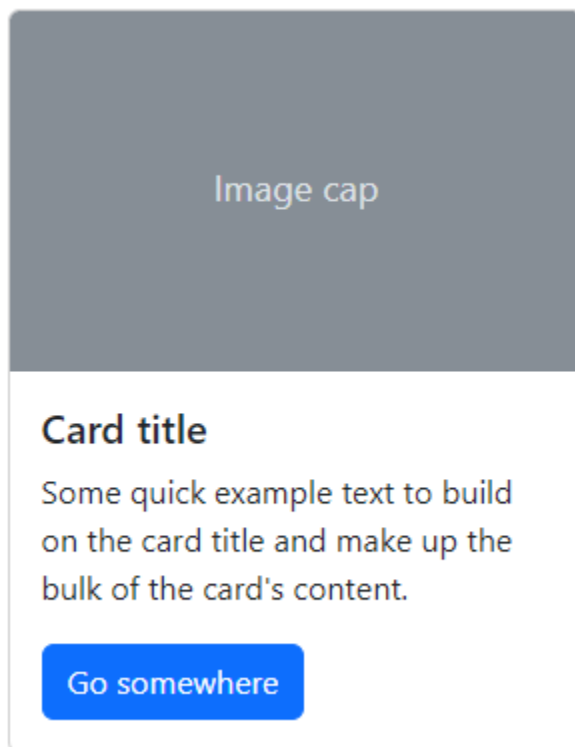
```
<div class="spinner-border" role="status">  
  <span class="visually-hidden">Loading...</span>  
</div>
```



## Cards

A card is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options. If you're familiar with Bootstrap 3, cards replace our old panels, wells, and thumbnails. Similar functionality to those components is available as modifier classes for cards.

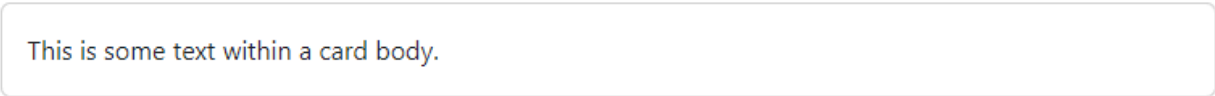
```
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title and make up the
bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```



## Body

The building block of a card is the `.card-body`. Use it whenever you need a padded section within a card.

```
<div class="card">
  <div class="card-body">
    This is some text within a card body.
  </div>
</div>
```



This is some text within a card body.

## Titles, text, and links

Card titles are used by adding `.card-title` to a `<h*>` tag. In the same way, links are added and placed next to each other by adding `.card-link` to an `<a>` tag.

Subtitles are used by adding a `.card-subtitle` to a `<h*>` tag. If the `.card-title` and the `.card-subtitle` items are placed in a `.card-body` item, the card title and subtitle are aligned nicely.

```
<div class="card" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <h6 class="card-subtitle mb-2 text-muted">Card subtitle</h6>
    <p class="card-text">Some quick example text to build on the card title and make up the
bulk of the card's content.</p>
    <a href="#" class="card-link">Card link</a>
    <a href="#" class="card-link">Another link</a>
  </div>
</div>
```

## Card title

### Card subtitle

Some quick example text to build on the card title and make up the bulk of the card's content.

[Card link](#) [Another link](#)

## Images

.card-img-top places an image to the top of the card. With .card-text, text can be added to the card. Text within .card-text can also be styled with the standard HTML tags.

```
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <p class="card-text">Some quick example text to build on the card title and make up the
    bulk of the card's content.</p>
  </div>
</div>
```

## List groups

Create lists of content in a card with a flush list group.

```
<div class="card" style="width: 18rem;">
  <ul class="list-group list-group-flush">
    <li class="list-group-item">An item</li>
    <li class="list-group-item">A second item</li>
    <li class="list-group-item">A third item</li>
  </ul>
</div>
```



An item
A second item
A third item

### Header and footer

Add an optional header and/or footer within a card.

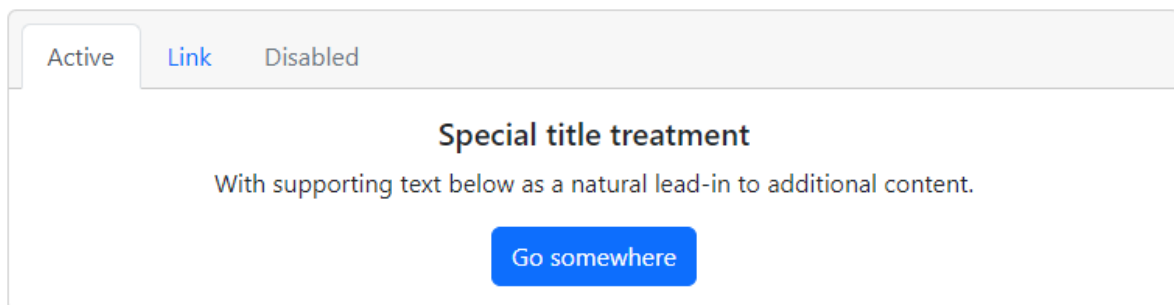
```
<div class="card">
  <div class="card-header">
    Featured
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to additional
content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Featured
<b>Special title treatment</b> With supporting text below as a natural lead-in to additional content. <a href="#">Go somewhere</a>

## Navigation

Add some navigation to a card's header (or block) with Bootstrap's nav components.

```
<div class="card text-center">
  <div class="card-header">
    <ul class="nav nav-tabs card-header-tabs">
      <li class="nav-item">
        <a class="nav-link active" aria-current="true" href="#">Active</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled">Disabled</a>
      </li>
    </ul>
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to additional
content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```



## Images

Cards include a few options for working with images. Choose from appending “image caps” at either end of a card, overlaying images with card content, or simply embedding the image in a card.

### Image caps

Similar to headers and footers, cards can include top and bottom “image caps”—images at the top or bottom of a card.

```
<div class="card mb-3">
  
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">This is a wider card with supporting text below as a natural lead-in
to additional content. This content is a little bit longer.</p>
    <p class="card-text"><small class="text-muted">Last updated 3 mins ago</small></p>
  </div>
</div>
<div class="card">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">This is a wider card with supporting text below as a natural lead-in
to additional content. This content is a little bit longer.</p>
    <p class="card-text"><small class="text-muted">Last updated 3 mins ago</small></p>
  </div>
  
</div>
```

Image cap

### Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago

### Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago

Image cap

## Bootstrap Carousel

The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript. It works with a series of images, text, or custom markup. It also includes support for previous/next controls and indicators.

### Slides only Carousel

```
<div id="carouselExampleSlidesOnly" class="carousel slide" data-bs-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
</div>
```

---

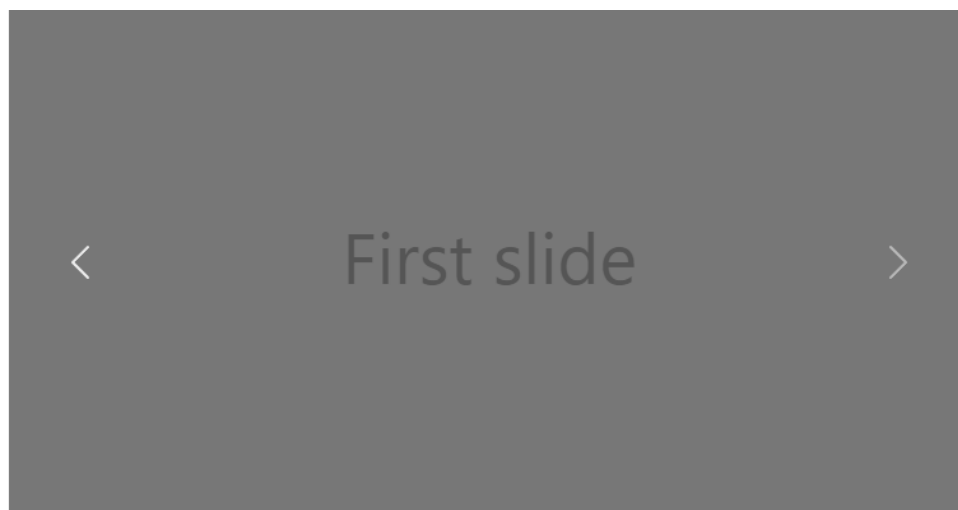


---

## Carousel with controls

Adding in the previous and next controls. We recommend using `<button>` elements, but you can also use `<a>` elements with `role="button"`.

```
<div id="carouselExampleControls" class="carousel slide" data-bs-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
  <button class="carousel-control-prev" type="button"
data-bs-target="#carouselExampleControls" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button"
data-bs-target="#carouselExampleControls" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```



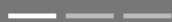
## Carousel with indicators

You can also add the indicators to the carousel, alongside the controls, too.

```
<div id="carouselExampleIndicators" class="carousel slide" data-bs-ride="true">
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="0" class="active" aria-current="true" aria-label="Slide 1"></button>
    <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="1" aria-label="Slide 2"></button>
    <button type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide-to="2" aria-label="Slide 3"></button>
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
  <button class="carousel-control-prev" type="button"
data-bs-target="#carouselExampleIndicators" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button"
data-bs-target="#carouselExampleIndicators" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```



First slide





## Carousel with captions

You can also add the indicators to the carousel, alongside the controls, too.

Add captions to your slides easily with the `.carousel-caption` element within any `.carousel-item`. They can be easily hidden on smaller viewports, as shown below, with optional display utilities. We hide them initially with `.d-none` and bring them back on medium-sized devices with `.d-md-block`.

```
<div id="carouselExampleCaptions" class="carousel slide" data-bs-ride="false">
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0"
class="active" aria-current="true" aria-label="Slide 1"></button>
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1"
aria-label="Slide 2"></button>
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2"
aria-label="Slide 3"></button>
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active">
      
      <div class="carousel-caption d-none d-md-block">
        <h5>First slide label</h5>
        <p>Some representative placeholder content for the first slide.</p>
      </div>
    </div>
    <div class="carousel-item">
      
      <div class="carousel-caption d-none d-md-block">
        <h5>Second slide label</h5>
        <p>Some representative placeholder content for the second slide.</p>
      </div>
    </div>
    <div class="carousel-item">
      
      <div class="carousel-caption d-none d-md-block">
        <h5>Third slide label</h5>
        <p>Some representative placeholder content for the third slide.</p>
      </div>
    </div>
  </div>
</div>
```

```
    </div>
  </div>
</div>
<button class="carousel-control-prev" type="button"
data-bs-target="#carouselExampleCaptions" data-bs-slide="prev">
  <span class="carousel-control-prev-icon" aria-hidden="true"></span>
  <span class="visually-hidden">Previous</span>
</button>
<button class="carousel-control-next" type="button"
data-bs-target="#carouselExampleCaptions" data-bs-slide="next">
  <span class="carousel-control-next-icon" aria-hidden="true"></span>
  <span class="visually-hidden">Next</span>
</button>
</div>
```



## Bootstrap Collapse

The collapse JavaScript plugin is used to show and hide content. Buttons or anchors are used as triggers that are mapped to specific elements you toggle.

Use the following classes on elements to show and hide content

- **.collapse** hides content
- **.collapsing** is applied during transitions
- **.collapse.show** shows content

```
<p>
```

```
  <a class="btn btn-primary" data-bs-toggle="collapse" href="#collapseExample"
  role="button" aria-expanded="false"
  aria-controls="collapseExample">
```

```
    Link with href
```

```
  </a>
```

```
</p>
```

```
<div class="collapse" id="collapseExample">
```

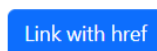
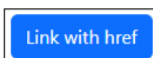
```
  <div class="card card-body">
```

Some placeholder content for the collapse component. This panel is hidden by default but revealed when the user

activates the relevant trigger.

```
  </div>
```

```
</div>
```



Some placeholder content for the collapse component. This panel is hidden by default but revealed when the user activates the relevant trigger.

## Bootstrap Accordion

Build vertically collapsing accordions in combination with Bootstrap's Collapse JavaScript plugin.

The accordion uses collapse internally to make it collapsible. To render an accordion that's expanded, add the `.open` class on the `.accordion`.

```
<div class="accordion" id="accordionExample">
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingOne">
      <button class="accordion-button" type="button" data-bs-toggle="collapse"
data-bs-target="#collapseOne"
      aria-expanded="true" aria-controls="collapseOne">
        Accordion Item #1
      </button>
    </h2>
    <div id="collapseOne" class="accordion-collapse collapse show"
aria-labelledby="headingOne"
      data-bs-parent="#accordionExample">
      <div class="accordion-body">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime in tempore veniam
molestiae similique facere
        quisquam deleniti doloribus ad? Debitis.
      </div>
    </div>
  </div>
  <div class="accordion-item">
    <h2 class="accordion-header" id="headingTwo">
      <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"
data-bs-target="#collapseTwo"
      aria-expanded="false" aria-controls="collapseTwo">
        Accordion Item #2
      </button>
    </h2>
    <div id="collapseTwo" class="accordion-collapse collapse" aria-labelledby="headingTwo"
      data-bs-parent="#accordionExample">
      <div class="accordion-body">
```

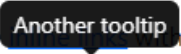
```
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime in tempore veniam molestiae similique facere  
    quisquam deleniti doloribus ad? Debitis.  
  </div>  
</div>  
</div>  
<div class="accordion-item">  
  <h2 class="accordion-header" id="headingThree">  
    <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse"  
data-bs-target="#collapseThree"  
    aria-expanded="false" aria-controls="collapseThree">  
    Accordion Item #3  
  </button>  
</h2>  
  <div id="collapseThree" class="accordion-collapse collapse"  
aria-labelledby="headingThree"  
    data-bs-parent="#accordionExample">  
    <div class="accordion-body">  
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime in tempore veniam molestiae similique facere  
      quisquam deleniti doloribus ad? Debitis.  
    </div>  
  </div>  
</div>  
</div>  
</div>
```

Accordion Item #1	^
Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime in tempore veniam molestiae similique facere quisquam deleniti doloribus ad? Debitis.	
Accordion Item #2	v
Accordion Item #3	v

## Tooltips

Bootstrap Tooltips can be placed on any element.

```
<p class="muted">Placeholder text to demonstrate some <a href="#"  
data-bs-toggle="tooltip"  
  data-bs-title="Default tooltip">inline links</a> with tooltips. This is now just filler, no  
killer. Content placed  
  here just to mimic the presence of <a href="#" data-bs-toggle="tooltip"  
data-bs-title="Another tooltip">real text</a>.  
  And all that just to give you an idea of how tooltips would look when used in real-world  
situations. So hopefully  
  you've now seen how <a href="#" data-bs-toggle="tooltip" data-bs-title="Another one  
here too">these tooltips on  
  links</a> can work in practice, once you use them on <a href="#"  
data-bs-toggle="tooltip"  
  data-bs-title="The last tip!">your own</a> site or project.  
</p>
```

Placeholder text to demonstrate some  tooltips. This is now just filler, no killer. Content placed here just to mimic the presence of [real text](#). And all that just to give you an idea of how tooltips would look when used in real-world situations. So hopefully you've now seen how [these tooltips on links](#) can work in practice, once you use them on [your own](#) site or project.

## Popovers

Bootstrap Popovers can be placed on any element.

```
<button type="button" class="btn btn-secondary m-5" data-bs-container="body"
data-bs-toggle="popover"
  data-bs-placement="top" data-bs-content="Top popover"
aria-describedby="popover337085">
  Popover on top
</button>
<button type="button" class="btn btn-secondary m-5" data-bs-container="body"
data-bs-toggle="popover"
  data-bs-placement="right" data-bs-content="Right popover">
  Popover on right
</button>
<button type="button" class="btn btn-secondary m-5" data-bs-container="body"
data-bs-toggle="popover"
  data-bs-placement="bottom" data-bs-content="Bottom popover">
  Popover on bottom
</button>
<button type="button" class="btn btn-secondary m-5" data-bs-container="body"
data-bs-toggle="popover"
  data-bs-placement="left" data-bs-content="Left popover"
aria-describedby="popover859246">
  Popover on left
</button>
```

Popover on top

Popover on right

Right popover

Popover on bottom

Bottom popover

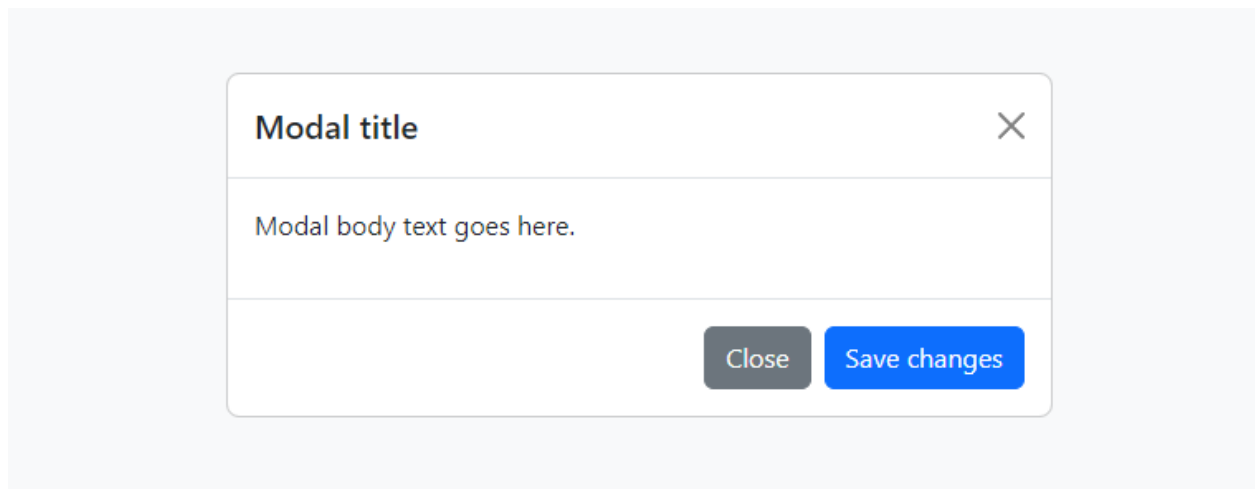
Popover on left



## Bootstrap Modals

Use Bootstrap's JavaScript modal plugin to add dialogs to your site for lightboxes, user notifications, or completely custom content.

```
<div class="modal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Modal title</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <p>Modal body text goes here.</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary"
data-bs-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```



## JavaScript

JavaScript is a programming language that is one of the core technologies of the Web, alongside HTML and CSS. Most of the websites on the internet use JavaScript on the client side for webpage behavior, often incorporating third-party libraries.

The main advantage of JavaScript is that it runs on the user's browser. There is no need to refresh the page or send the page to server and do some calculation.

All the operations happens on Browser so it makes the JavaScript extra ordinary faster than any web language.

Latest Famous Languages like Angular are completely built on JavaScript which handles the front and back of the application from the browser itself. New Web Programming Languages are based on JavaScript.

JavaScript can be written in many different places in an HTML page.

One of the methods of defining the JavaScript is inside the same HTML page.

This type of JavaScript is restricted to page level only means you cannot reuse this code in some other pages. Benefit of using this internal JavaScript is when you want specific changes to apply for that page level only.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="Page Description">
  <title>Internal JavaScript</title>
  <script>
    alert('This is called from Head Section!');
  </script>
</head>

<body>
```

```
<h1>Heading</h1>
<p>Paragraph Text</p>
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
</body>

</html>
```

## **External Javascript**

JavaScript (JS) can be written in another file and included inside the HTML page. This type of external JS is very powerful and helpful technique which is commonly used in every website development.

Benefit of using this external JS is that you have one JS file that is included in all the website pages. By just changing at one place in the JS it will impact the overall site design look and feel.

This is one of the best practice to separate the design with the html tags and store them in an external file and include it in all the HTML pages. External JS filename should be .js and it can be included anywhere inside the HTML page with

```
<script type='text/javascript' src="location of the file"></script>
```

## **SYNTAX:**

```
<head>
  <script type="text/javascript" src="javascript/javascriptcode.js"></script>
</head>
```

<script> tag is used to link the resource to the HTML page. The attribute of script tag will let the browser know what type of resource it is.

src attribute is similar to <img> tag src to map the location of the file in the server with the path and filename. You can even mention the folder name/file name to refer to the file path.

### **Example:**

Consider two files: index.html javascript/javascriptcode.js

- javascriptcode.js file is linked inside the index.html file with <script> tag.
- javascript is the folder name, it could be any name and not mandatory to have that name.
- type attribute tells the type of the content in the file. In this case, it is text/javascript

### **Content of the HTML file:**

```
<!DOCTYPE html>
<html>
<head>
  <title>External JavaScript</title>
  <script type='text/javascript' src="javascript/javascriptcode.js"></script>
</head>

<body>
  <h1>External JavaScript!</h1>
  <p>Paragraph Text</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
</body>
```

</html>

### **Content of the JS file:**

```
alert('I am called from javascriptcode.js file!');
```

### **Inline JavaScript**

Inline JavaScript is defined inside the HTML tag itself like an attribute. Inline JS overrides all the styles defined in internal JS and External JS. Inline JS code is executed first. This is defined in the HTML tag as an attribute.

#### **SYNTAX:**

```
<a href="#" onclick="alert('Welcome  
to JavaScript!');">Click Me</a>
```

onclick is an event for <a> tag and it is called when a user clicks on the link.

Calling this alert is handled by the browser itself. When a user clicks on the link, the browser will raise the click event on this tag and because we ask to raise an alert when this event is raised. This method is called.

Whatever is mentioned inside the onClick value will be executed as JavaScript. JavaScript is written inside this onClick event within double quotes "".

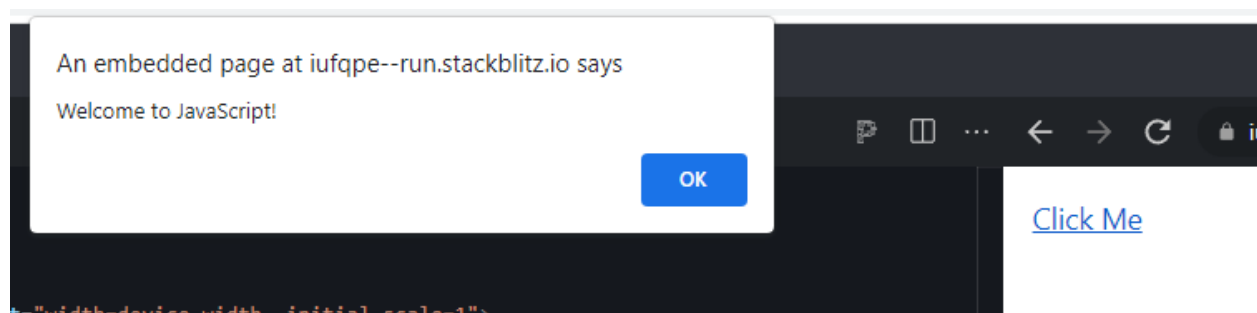
All the JavaScript is exactly similar as mentioned in internal and external JavaScript.

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

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="Page Description">
  <title>Inline JavaScript</title>
</head>

<body>
  <a href="#" onclick="alert('Welcome to JavaScript!');">Click Me</a>
</body>

</html>
```



## Console Tab

Console tab is the output tab in the browser's developer tools where you can write the logs of your program and view it. Even browser will it is own log here if there is any issue in the HTML program.

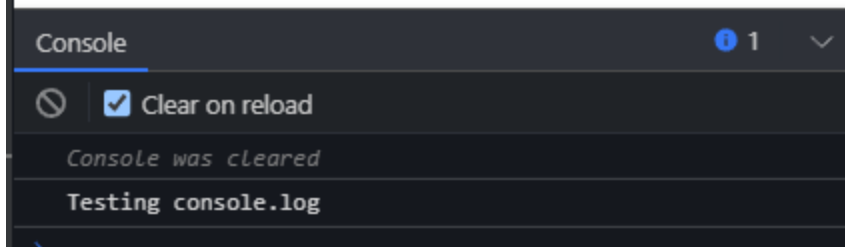
```
console.log('This is a log from JavaScript program');
```

This is the easiest way to debug your program by writing console.log at many places in your code and verify it from the inspect -> console window

# Internal JavaScript!

Paragraph Text

- Item 1
- Item 2
- Item 3



## JavaScript Variables

Variables are containers for storing data (storing data values)

There are 3 ways to declare a JavaScript Variable:

1. Using var
2. Using let
3. Using const

Always declare JavaScript variables with var, let, or const

- The var keyword is used in all JavaScript code from 1995 to 2015.
- The let and const keywords were added to JavaScript in 2015.
- If you want your code to run in an older browser, you must use var.

If you want a general rule: always declare variables with `const`. The `const` variables can be assigned a value only once. Then it cannot be changed, in other words you cannot reassign another value to it.

If you think the value of the variable can change, use `let`.

In this example, `price1`, `price2`, and `total`, are variable

```
const price1 = 5;  
const price2 = 6;  
let total = price1 + price2;
```

Here `price1` and `price2` cannot be changed. These are constant values and cannot be changed. '`total`' can change because it has been declared using the '`let`' keyword, which means that variable can be reassigned to new values.

## JavaScript Identifiers

All JavaScript variables must be identified with unique names. These unique names are called identifiers.

Identifiers can be short names (like `x` and `y`) or more descriptive names (`age`, `sum`, `totalVolume`).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with `$` and `_` (but we will not use it in this tutorial)
- Names are case sensitive (`y` and `Y` are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names



## JavaScript DataTypes

In programming, data types are an important concept. To be able to operate on variables, it is important to know something about the type. Without data types, a computer cannot safely solve this.

```
let x = 16 + "Car";
```

JavaScript will treat the example above as:

```
let x = "16" + "Car";
```

So, x will hold the value of "16Car"

*When adding a number and a string, JavaScript will treat the number as a string.*

## JavaScript Types are Dynamic

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

```
let x;          // Now x is undefined
x = 5;          // Now x is a Number
x = "John";     // Now x is a String
```

## JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes

```
let answer1 = "It's alright";           // Single quote inside double quotes
let answer2 = "He is called 'Johnny'";  // Single quotes inside double quotes
let answer3 = 'He is called "Johnny"';  // Double quotes inside single quotes
```

## JavaScript Numbers

JavaScript has only one type of numbers. Numbers can be written with, or without decimals.

```
let x1 = 34.00;    // Written with decimals
let x2 = 34;       // Written without decimals
```

## JavaScript Booleans

Booleans can only have two values: true or false. Booleans are often used in conditional testing.

```
let x = 5;
let y = 5;
let z = 6;
(x == y)    // Returns true
(x == z)    // Returns false
```

## JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

```
const cars = ["Maruti", "Tata", "Mahindra"];
```

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

## JavaScript Objects

JavaScript objects are written with curly braces {}. Object properties are written as name:value pairs, separated by commas.

Eg:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

## The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable. The typeof operator returns the type of a variable or an expression.

```
typeof ""           // Returns "string"  
typeof "John"       // Returns "string"  
typeof "John Doe"   // Returns "string"
```

```
typeof 0            // Returns "number"  
typeof 314          // Returns "number"  
typeof 3.14         // Returns "number"  
typeof (3)          // Returns "number"  
typeof (3 + 4)      // Returns "number"
```

## Undefined

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

```
let car; // Value is undefined, type is undefined
```

Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

```
car = undefined; // Value is undefined, type is undefined
```

## Empty Values

An empty value has nothing to do with undefined. An empty string has both a legal value and a type.

```
let car = ""; // The value is "", the typeof is "string"
```

## JavaScript Arrays of Objects

JavaScript variables can be objects. Arrays are special kinds of objects. Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array.

```
myArray[0] = Date.now;  
myArray[1] = myFunction;  
myArray[2] = myCars;
```

## Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
cars.length // Returns the number of elements  
cars.sort() // Sorts the array
```

## The length Property

The length property of an array returns the length of an array (the number of array elements).

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let length = fruits.length;
```

### Accessing the First Array Element

JavaScript has only one type of numbers. Numbers can be written with, or without decimals.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

### Accessing the Last Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

### Looping Array Elements

One way to loop through an array, is using a for loop.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;
```

```
let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

### Adding Array Elements

The easiest way to add a new element to an array is using the push() method

Eg:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits.push("Lemon"); // Adds a new element (Lemon) to fruits
```

New element can also be added to an array using the length property:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[fruits.length] = "Lemon"; // Adds "Lemon" to fruits
```

### **When to Use Arrays. When to use Objects.**

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

- JavaScript does not support associative arrays.
- 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.

### **JavaScript new Array()**

An empty value has nothing to do with undefined. An empty string has both a legal value and a type.

```
let car = ""; // The value is "", the typeof is "string"
```

JavaScript has a built-in array constructor `new Array()`. But you can safely use `[]` instead.

These two different statements both create a new empty array named points.

```
const points = new Array(40, 100, 1, 5, 25, 10);
const points = [40, 100, 1, 5, 25, 10];
```

## JavaScript Loops

Loops are used to run the same code over and over again, each time with a different value. Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";
```

We can write:

```
for (let i = 0; i < cars.length; i++) {  
  text += cars[i] + "<br>";  
}
```

### JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- 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 Loop

The for statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {  
  // code block to be executed  
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Eg:

```
for (let i = 0; i < 5; i++) {  
  text += "The number is " + i + "<br>";  
}
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

### Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case, JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

### Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case, JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again, if it returns false, the loop will end.

### Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case, JavaScript doesn't care, and expression 3 is optional.

Expression 3 can do anything like negative increment ( $i--$ ), positive increment ( $i = i + 15$ ), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

## Loop Scope:

### Using var in a loop

```
var i = 5;
```

```
for (var i = 0; i < 10; i++) {  
    // some code  
}
```

```
// Here i is 10
```

### Using let in a loop:

```
let i = 5;
```

```
for (let i = 0; i < 10; i++) {  
    // some code  
}
```

```
// Here i is 5
```

In the first example, using var, the variable declared in the loop redeclares the variable outside the loop. In the second example, using let, the variable declared in the loop does not redeclare the variable outside the loop. When let is used to declare the i variable in a loop, the i variable will only be visible within the loop.

### **For/Of and For/In Loops**

The JavaScript for in statement loops through the properties of an Object:

```
for (key in object) {  
  // code block to be executed  
}
```

Eg:

```
const person = {fname:"John", lname:"Doe", age:25};
```

```
let text = "";  
for (let x in person) {  
  text += person[x];  
}
```

#### **Example Explained**

- The for in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key
- The value of the key is person[x]

## **For In Over Arrays**

The JavaScript for in statement can also loop over the properties of an Array:

```
for (variable in array) {  
  code  
}
```

Eg:

```
const numbers = [45, 4, 9, 16, 25];
```

```
let txt = "";  
for (let x in numbers) {  
  txt += numbers[x];  
}
```

## JavaScript Conditionals:

Conditional statements are used to perform different actions based on different conditions.

### Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

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

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

Eg:

```
if (hour < 18) {  
    greeting = "Good day";  
}
```

## **The else Statement**

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

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

## **The else if Statement**

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

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false and condition2 is false  
}
```

Eg:

```
if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

## Switch Statement

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {  
  case x:  
    // code block  
    break;  
  case y:  
    // code block  
    break;  
  default:  
    // code block  
}
```

This is how it works:

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.

Eg:

```
switch (new Date().getDay()) {  
  case 0:  
    day = "Sunday";  
    break;  
  case 1:  
    day = "Monday";  
    break;  
  case 2:  
    day = "Tuesday";  
    break;  
  case 3:  
    day = "Wednesday";  
    break;
```

```
case 4:
  day = "Thursday";
  break;
case 5:
  day = "Friday";
  break;
case 6:
  day = "Saturday";
}
```

### **The break Keyword**

When JavaScript reaches a “break” keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

### **The default Keyword**

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

Eg:

The `getDay()` method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:



```
switch (new Date().getDay()) {  
  case 6:  
    text = "Today is Saturday";  
    break;  
  case 0:  
    text = "Today is Sunday";  
    break;  
  default:  
    text = "Looking forward to the Weekend";  
}
```

## Array helpers

### forEach()

array.forEach() calls a function on each element in an array. The iterator function deals with the operations that have to be performed. The array takes the parameter, which is the Iterator Function.

**Syntax:** *array.forEach(function(currentValue, index, arr));*

Eg:

```
const arr = [1,2,3,4,5];  
arr.forEach(item => console.log(item))
```

### Output:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

### map()

array.map() loops through each item of the array, same as forEach() but map() returns the value of the array. map() calls the function for every element of the array in a particular order. Let's start with a simple example:

```
const arr = [{name:"Ashish",age:18}, {name:"Amit",age:20}]  
let resArray = []  
for(let i = 0; i < arr.length; i++){  
    resArray.push(arr[i].name)  
}  
console.log(resArray);
```

### Output:

["Ashish","Amit"]

Now, let's approach this using the `array.map()` helper method.

```
array.map(function(currentValue, index, arr));
```

The following code shows the use of the `map()` helper method:

```
const arr = [  
  {name: "Ashish", age: 18},  
  {name: "Amit", age: 20 }  
]  
  
const people = arr.map(person => person.name)  
  
console.log(people);
```

**Output:**

```
["Ashish","Amit"]
```

**filter()**

`array.filter()` contains a boolean condition. `filter()` returns an array consisting of all the elements that are true inside a function. It does not make any change in the original array. `filter()` is often used in filtering and sorting a list.

**Syntax:** `array.filter(function(currentValue, index, arr));`

Example:

```
const arr = [  
  {name: "Ashish", age: 18},  
  {name: "Amit", age: 20 },  
  {name: "Sanjay", age: 20 }  
]  
  
const people = arr.filter(person => person.age == 20)  
  
console.log(people);
```

**Output:**

```
[{name: "Amit", age: 20 }, {name: "Sanjay", age: 20 }]
```

**find()**

array.find() find value in the array (only the first value), if an element in the array satisfies the provided testing function, else undefined if not found. It is really useful in a situation where we have a massive database and we have to find the ids of people.

```
const arr = [  
  {name: "Ashish", age: 18},  
  {name: "Amit", age: 20 },  
  {name: "Sanjay", age: 20 }  
]  
  
const people = arr.find(person => person.name == "Sanjay")  
  
console.log(people);
```

**Output:**

```
[{name: "Sanjay", age: 20 }]
```

### **reduce()**

`array.reduce()` takes two parameters. The first is the accumulator and the second is the initial value that we can set. Let's understand this method with an example:

Eg:

```
array = [1, 2, 3, 4, 5];
```

```
let result = array.reduce((acc, val) => {  
    return acc + val;  
}, 0);
```

```
console.log(result);
```

In this example, we are using `reduce()` helper method that takes two parameters, accumulator and current value, and returns the sum of all elements in an array.

### **Output:**

15

we can probably reimplement all the other helper methods by using `reduce()`. This is how `reduce()` works.

## **some()**

`array.some()` checks whether at least one element in the array passes the test implemented by the provided function. It returns a Boolean value. In `some()`, it is enough that one value will be matched with the condition and it will return `true`. Let's take an example :

```
const arr = [  
  {name: "Ashish", age: 18},  
  {name: "Amit", age: 20 },  
  {name: "Sanjay", age: 20 }  
]
```

```
const matchSome = arr.some(person => person.name == "Ashish")  
console.log(matchSome );
```

## **Output:**

true

In this above example, `some()` will check if any element in an array matches the condition if it matches then the function will return `true` else `false`.

## **every()**

`array.every()` returns true if everything matches the condition. It returns a Boolean value. It is the same as the `some()` helper method but the only difference is it uses Logical AND instead of OR.

Example:

```
const arr = [  
  {name: "Ashish", age: 18},  
  {name: "Amit", age: 20 },  
  {name: "Sanjay", age: 20 }  
]
```

```
const matchEvery = arr.every(person => person.age == 20)  
console.log(matchEvery );
```

**Output:**

false

## **from()**

`array.from()` is the method that creates a new instance of the Array from an array-like or iterable object. This method is a static method that is called using the Array class name.

**Syntax:** *Array.from(arraylike, mapFunc, thisArg)*

In this Syntax :

- `arrayLike` — Array-like or iterable object to convert to an array.
- `mapFunc(optional)` — Map function that is called on each element.
- `thisArg(optional)`—Value to use as this when executing `mapFunc`.

`Array.from()` method of ES6 returns a new Array instance.

Eg:

```
function fromMethod() {  
  return Array.from(arguments);  
}
```

```
console.log(fromMethod(1, "A"));
```

**Output:** `[ 1, 'A' ]`



## **of()**

`Array.of()` method always creates an array that contains the values that you pass to it regardless of the types or the number of arguments. The main difference between `Array.of()` and `Array()` constructor is that when we pass an integer value as a parameter in `Array.of()`, it creates an array with a single element that we pass whereas in `Array()` constructor it creates a length of a given size that is passed as a parameter.

Syntax:

```
Array.of(element0[, element1[, ...[, elementN]]])
```

Eg:

```
let numbers = new Array(2);  
console.log(numbers.length); // 2
```

Now let's see `Array.of()` method example :

```
let numbers = Array.of(3);  
console.log(numbers.length); // 1
```

In this example, we passed the number 3 to the `Array.of()` method. The `Array.of()` method creates an array of one number.

### **findIndex()**

Array.findIndex() method returns the index of the first element in the array that satisfies the provided testing function. Otherwise, it returns -1.

Syntax :

*findIndex(testFn(element[, index[, array]])[, thisArg])*

The above example returns the index of the first occurrence of the number 7 in the numArray array.

Eg:

```
let numArray = [1, 5, 7, 8, 10, 7];  
let index = numArray.findIndex(numArray => numArray === 7);  
console.log(index);
```

The above example returns the index of the first occurrence of the number 7 in the numArray array.

### **Keys()**

Array.keys() method returns a new Array Iterator object that contains the keys for each index in the array. This method never ignores the empty string.

Syntax :

arrayName.keys()

In the above syntax `arrayName` is the name of your array.

`Array.keys()` example :

```
const languages = ["JavaScript", "Java", "C#", "C"];
```

```
let iterator = languages.keys();
```

```
for (let key of iterator) {  
  console.log(key);  
}
```

**Output :**

```
0  
1  
2  
3  
4
```

**values()**

`Array.value()` method returns a new Array object that contains the values for each index in the array.

Syntax :

```
arrayName.values()
```

In the above syntax `arrayName` is the name of your array.

`Array.value()` code example :

```
const array = ["a", "b", "c"];
```

```
const iterator = array.values();
```

```
for (const value of iterator) {  
  console.log(value);  
}
```

**Output :**

a

b

c

## JavaScript Functions

Function is a block of statements that performs an action. You can pass parameters to functions and it can return a value from the function using the "return " keyword.

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: {}

### Syntax:

```
function nameOfFunction(Parameters) {  
    return someValue;  
}
```

Eg:

```
function area(width, height) {  
    return width * height;  
}
```

```
var size = area(10, 20);
```

**Function parameters** are listed inside the parentheses () in the function definition. Function arguments are the values received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables.

### Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

### Function Return

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":

Calculate the product of two numbers, and return the result:

```
let x = myFunction(4, 3); // Function is called, return value will end up in x
```

```
function myFunction(a, b) {  
  return a * b;          // Function returns the product of a and b  
}
```

The result in x will be:

12

## The () Operator Invokes the Function

Using the example below, toCelsius refers to the function object, and toCelsius() refers to the function result.

```
function toCelsius(fahrenheit) {  
  return (5/9) * (fahrenheit-32);  
}
```

```
document.getElementById("demo").innerHTML = toCelsius(77);
```

Accessing a function without () will return the function object instead of the function result.

```
function toCelsius(fahrenheit) {  
  return (5/9) * (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius;
```

## Object Types (Blueprints) (Constructor function)

```
function Person(first, last, age, eye) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eye;  
}
```

Sometimes we need a "blueprint" for creating many objects of the same "type". The way to create an "object type", is to use an object constructor function. In the example above, function `Person()` is an object constructor function. Objects of the same type are created by calling the constructor function with the `new` keyword

```
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Alice", "Mike", 48, "green");
```

JavaScript has built-in constructors for native objects:

```
new String()  // A new String object  
new Number()  // A new Number object  
new Boolean()  // A new Boolean object  
new Object()  // A new Object object  
new Array()   // A new Array object  
new RegExp()  // A new RegExp object  
new Function() // A new Function object  
new Date()    // A new Date object
```



## **EcmaScript 6 (ES6)**

Object Orientation is a software development paradigm that follows real-world modeling. Object Orientation, considers a program as a collection of objects that communicates with each other via mechanisms called methods. ES6 supports these object-oriented components too.

### **Object-Oriented Programming Concepts**

To begin with, let us understand

Object - An object is a real-time representation of any entity. According to Grady Brooch, every object is said to have 3 features -

- State - Described by the attributes of an object.
- Behavior - Describes how the object will act.
- Identity - A unique value that distinguishes an object from a set of similar such objects.

Class - A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object.

Method - Methods facilitate communication between objects.

Let us translate these Object-Oriented concepts to the ones in the real world. For example: A car is an object that has data (make, model, number of doors, Vehicle Number, etc.) and functionality (accelerate, shift, open doors, turn on headlights, etc.)

Prior to ES6, creating a class was a fussy affair. Classes can be created using the class keyword in ES6.

Classes can be included in the code either by declaring them or by using class expressions.

**Syntax:** Declaring a Class

```
class Class_name {  
}
```

**Syntax:** Class Expressions

```
var var_name = new Class_name {  
}
```

The class keyword is followed by the class name. The rules for identifiers (already discussed) must be considered while naming a class.

A class definition can include the following -

Constructors - Responsible for allocating memory for the objects of the class.

Functions - Functions represent actions an object can take. They are also at times referred to as methods.

These components put together are termed as the data members of the class.

Note - A class body can only contain methods, but not data properties.

**Example: Declaring a class**

```
class Polygon {  
    constructor(height, width) {  
        this.height = height;  
        this.width = width;  
    }  
}
```

**Example: Class Expression**

```
var Polygon = class {  
  constructor(height, width) {  
    this.height = height;  
    this.width = width;  
  }  
}
```

The above code snippet represents an unnamed class expression. A named class expression can be written as.

```
var Polygon = class Polygon {  
  constructor(height, width) {  
    this.height = height;  
    this.width = width;  
  }  
}
```

## Creating Objects

To create an instance of the class, use the new keyword followed by the class name. Following is the syntax for the same.

```
var object_name= new class_name([ arguments ])
```

Where, The **new** keyword is responsible for instantiation.

The right hand side of the expression invokes the constructor. The constructor should be passed values if it is parameterized.

Example: Instantiating a class

```
var obj = new Polygon(10,12)
```

## Accessing Functions

A class's attributes and functions can be accessed through the object. Use the '.' dot notation (called as the period) to access the data members of a class.

```
//accessing a function
```

```
obj.function_name()
```

Example: Putting them together

```
'use strict'
```

```
class Polygon {
```

```
  constructor(height, width) {
```

```
    this.h = height;
```

```
    this.w = width;
```

```
  }
```

```
  test() {
```

```
    console.log("The height of the polygon: ", this.h)
```

```
    console.log("The width of the polygon: ",this. w)
```

```
  }
```

```
}
```

```
//creating an instance
```

```
var polyObj = new Polygon(10,20);
```

```
polyObj.test();
```

The Example given above declares a class 'Polygon'. The class's constructor takes two arguments - height and width respectively. The 'this' keyword refers to the current instance of the class. In other words, the constructor above initializes two variables h and w with the parameter values passed to the constructor. The test () function in the class, prints the values of the height and width.

To make the script functional, an object of the class Polygon is created. The object is referred to by the polyObj variable. The function is then called via this object.

The following output is displayed on successful execution of the above code.

The height of the polygon: 10

The width of the polygon: 20

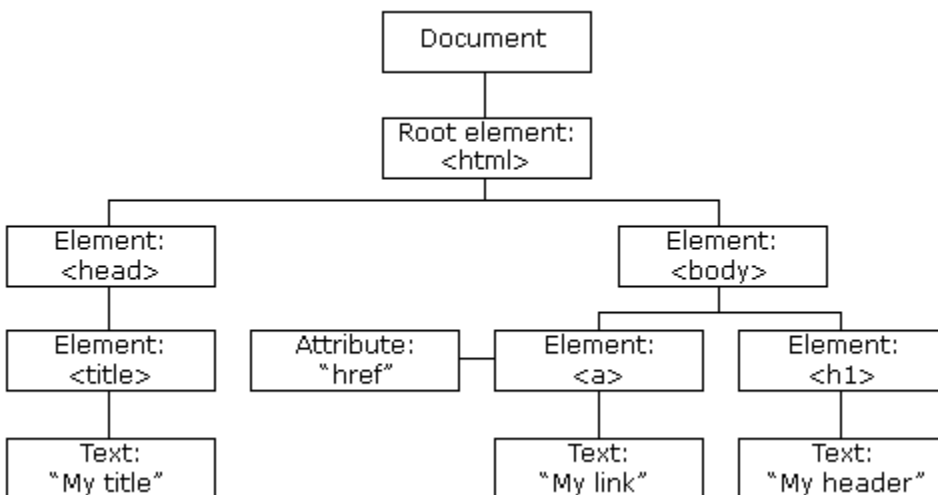
## DOM (Document Object Model)

Document Object Model or DOM in short is nothing but the hierarchy representation of all the HTML elements like a tree. DOM is created by the browser for the HTML page so that it can easily navigate and make changes to the elements.

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

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:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

JavaScript can change all the HTML elements in the page  
JavaScript can change all the HTML attributes in the page  
JavaScript can change all the CSS styles in the page  
JavaScript can remove existing HTML elements and attributes  
JavaScript can add new HTML elements and attributes  
JavaScript can react to all existing HTML events in the page  
JavaScript can create new HTML events in the page

JavaScript can modify the DOM elements. By adding, removing and reading the elements from the DOM.

### **Read a Value of an Element**

You can read any element from the DOM using JavaScript functions:

1. `getElementById(idName)`
2. `getElementsByTagName(tagName)`
3. `getElementByName(name)`
4. `getElementByClassName(className)`

```
<!DOCTYPE html>
<html>

<head>
  <title>Inline JavaScript</title>
  <script>
    function show() {
      var txtValue = document.getElementById('txtName').value; alert(txtValue);
    }
  </script>
</head>

<body>
  <h1>Access the DOM Elements</h1>
  <form>
    <input type="text" id="txtName" />
```

```
    <input type="button" onclick="show();" Value="Show Value" />
  </form>
</body>

</html>
```

## Update the Data

Update the Element Data innerHTML property can be used to update data for any element.

Adding data to this attribute will modify the DOM element and it will reflect immediately on the page.

```
<!DOCTYPE html>
<html>
<head>
  <title>Inline JavaScript</title>
  <script>
    function update() {
      document.getElementById("txtValue").innerHTML =
document.getElementById("txtName").value;
    }
  </script>
</head>
<body>
  <h1>Update the DOM Elements</h1>
  <p id="txtValue"></p>
  <form>
    <input type="text" id="txtName" onkeypress="update();" />
  </form>
</body>
</html>
```



## Access the Form Elements

You can access all the form elements using the DOM. This is very helpful when you want to check what user has entered the data before sending the data to the server to save it. It is called Client Side Validation.

```
<!DOCTYPE html>
<html>
<head>
  <script>
    function checkFormData() {
      //Query all elements that tag name as input var inputs =
      document.getElementsByTagName("input");
      var message = "Form Elements\n\n";
      for (var i = 0; i < inputs.length; i++) {
        if (inputs[i].getAttribute('type') == 'text') {
          message += inputs[i].getAttribute('name') + ": "; message += inputs[i].value + "\n";
        }
      } alert(message);
    }
  </script>
</head>

<body>
  <form name="user" id="userfrm" action="#">
    Your Name: <input type="text" name="name" id="txt_name" />
    Your Email: <input type="text" name="email" id="txt_email" />
    <input type="button" name="submit" value="Submit" onclick="checkFormData();" />
  </form>
</body>

</html>
```

## **Window Object**

The Window interface represents a window containing a DOM document; the document property points to the DOM document loaded in that window.

A window for a given document can be obtained using the document.defaultView property.

A global variable, window, representing the window in which the script is running, is exposed to JavaScript code.

The window object represents an open window in a browser.

If a document contains frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

## JavaScript Events

HTML events are "things" that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can "react" to these events.

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

An HTML web page has finished loading

An HTML input field was changed

An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

Eg:

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time  
is?</button>
```

In the example above, the JavaScript code changes the content of the element with `id="demo"`.

### **Common HTML Events**

Here is a list of some common HTML events:

**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

## Access the Form Elements

You can access all the form elements using the DOM. This is very helpful when you want to check what user has entered the data before sending the data to the server to save it. It is called Client Side Validation.

Eg:

```
<!DOCTYPE html>
<html>

<head>
  <script>
    function checkFormData() {
      //Query all elements that tag name as input
      var inputs = document.getElementsByTagName("input");
      var message = "Form Elements\n\n";
      for (var i = 0; i < inputs.length; i++) {
        if (inputs[i].getAttribute('type') == 'text') {
          message += inputs[i].getAttribute('name') + ": "; message += inputs[i].value + "\n";
        }
      } alert(message);
    }
  </script>
</head>

<body>
  <form name="user" id="userfrm" action="#">
    Your Name: <input type="text" name="name" id="txt_name" />
    Your Email: <input type="text" name="email" id="txt_email" />
    <input type="button" name="submit" value="Submit" onclick="checkFormData();" />
  </form>
</body>

</html>
```

www.tutorialspoint.com says

Form Elements

name: Test Name

email: test-name@gmail.com

OK

mail: test-name@gmail.com

Submit