

GE-161L

Introduction to Information and Communication Technologies

Laboratory 14

**Introduction to Web Development – III
(CSS & JavaScript)**

Version: 1.0.0

Release Date: 07-05-2022

**Department of Information Technology
University of the Punjab
Lahore, Pakistan**

Contents:

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
 - JavaScript
 - What JavaScript do?
- Activities
 - Pre-Lab Activity
 - CSS Flexbox
 - CSS Flex Container
 - Flex Direction Property
 - Flex wrap Property
 - Justify Content Property
 - Align Items Property
 - Task 01 (Develop Web Page using Flex Box)
 - In-Lab Activity
 - CSS Media Queries
 - Min Width to Max Width
 - Media Queries for Menus
 - Media Queries for Columns
 - Flex Box Responsive
 - JavaScript HTML DOM
 - Including JavaScript in Your Page
 - The HTML DOM Document Object
 - getElementById Method
 - The innerHTML Property
 - Changing HTML Content
 - Changing the Value of an Attribute
 - Changing HTML Style
 - Using Events
 - Task 01: HTML DOM (I)
 - Task 02: HTML DOM (II)
 - Task 03: HTML DOM (III)
 - Task 04: HTML DOM (IV)
 - Post-Lab Activity
 - Task 01: Develop a Static Website
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

Learning Objectives:

- Learn to develop web pages
- Learn HTML, CSS and JavaScript
- Learn Web page formatting concepts

Required Resources:

- Open Desktop or Laptop PC
- Text Editor
- Browser

General Instructions:

- This is an individual lab, you are **NOT** allowed to discuss your solution with your colleagues, not even allowed to ask how is he/she doing, this may result into negative marking. You can **ONLY** discuss with your TAs or with course instructor.
- Your TAs will be available in the lab for your help. Alternatively, you can send your queries via email.

Lab Instructors:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants (TAs)	Usman Ali	bitf19m007@pucit.edu.pk
	Saad Rahman	bsef19m021@pucit.edu.pk
	Mahreen Asama	bsef19m030@pucit.edu.pk

Background and Overview:

JavaScript:

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.

What is JavaScript doing on your page?

Here we'll actually start looking at some code, and while doing so, explore what actually happens when you run some JavaScript in your page.

Let's briefly recap the story of what happens when you load a web page in a browser (first talked about in our How CSS works article). When you load a web page in your browser, you are running your code (the HTML, CSS, and JavaScript) inside an execution environment (the browser tab). This is like a factory that takes in raw materials (the code) and outputs a product (the web page).

A very common use of JavaScript is to dynamically modify HTML and CSS to update a user interface, via the Document Object Model API (as mentioned above). Note that the code in your web documents is generally loaded and executed in the order it appears on the page. Errors may occur if JavaScript is loaded and run before the HTML and CSS that it is intended to modify. You will learn ways around this later in the article, in the Script loading strategies section.

Activities:

Pre-Lab Activities:

CSS Flexbox:

What is CSS Flexbox?

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

Basics and Terminology

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

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

CSS Flex Container:

Parent Element (Container)

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



Fig. 1 (Flex Container)

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

Example:

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

The flex container properties are:

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

The flex-direction Property:

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



Fig. 2 (Flex Container)

Example

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

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

The flex-direction Property:

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

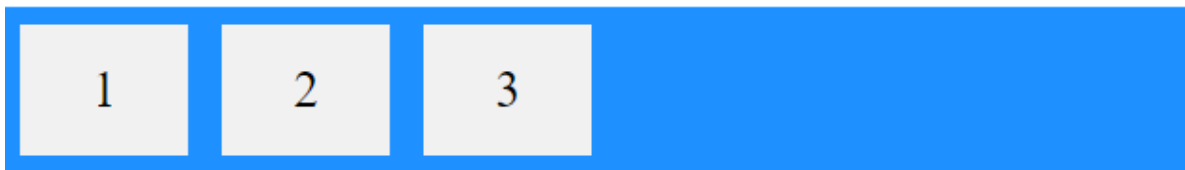


Fig. 3 (Flex Container)

Example

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

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

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

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

The flex-wrap Property

The flex-wrap property specifies whether the flex items should wrap or not. The flex-wrap property is a sub-property of the Flexible Box Layout module. It defines whether the flex items are forced in a

single line or can be flowed into multiple lines. If set to multiple lines, it also defines the cross-axis which determines the direction new lines are stacked in.

Example:

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

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

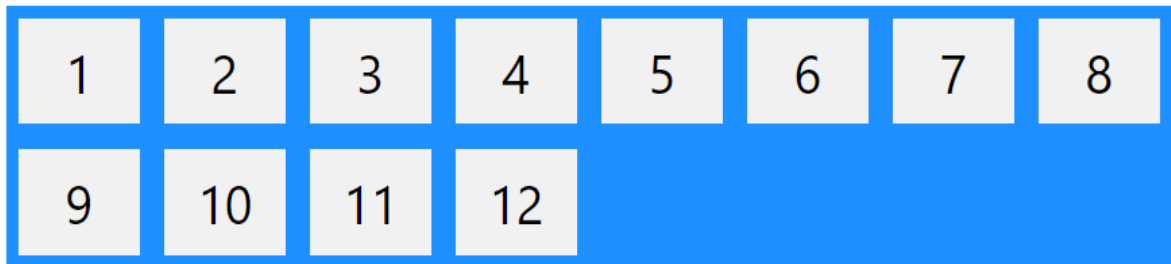


Fig. 4 (Flex Container)

The flex-wrap Property

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

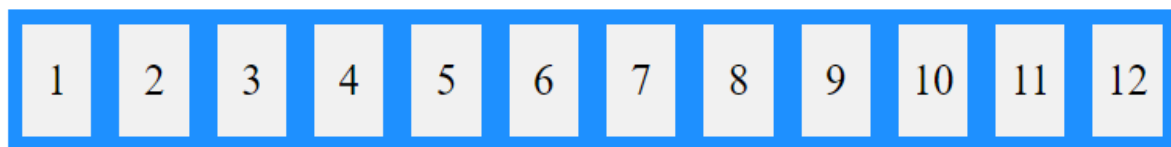


Fig. 5 (Flex Container)

All items are in one row.

The justify-content Property:

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

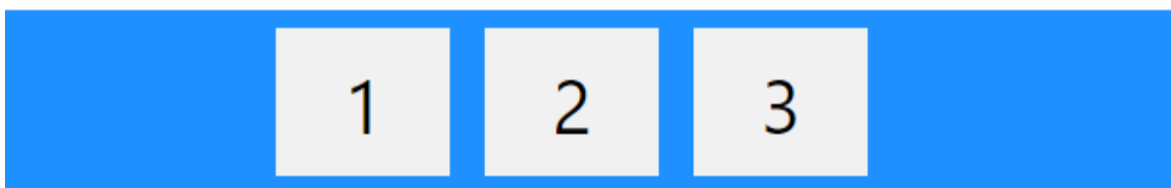


Fig. 6 (Flex Container)

Example

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

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

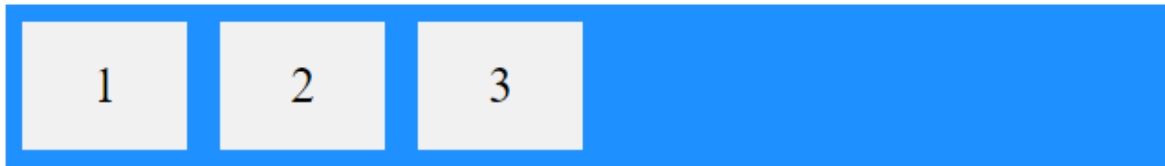


Fig. 7 (Flex Container)

Example

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

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

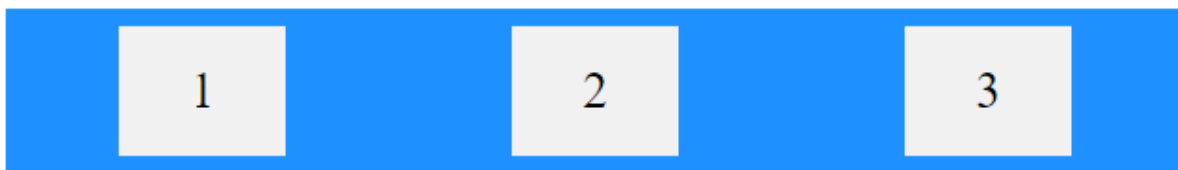


Fig. 8 (Flex Container)

Example:

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

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

The align-items Property:

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



Fig. 9 (Flex Container)

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

Example:

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

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

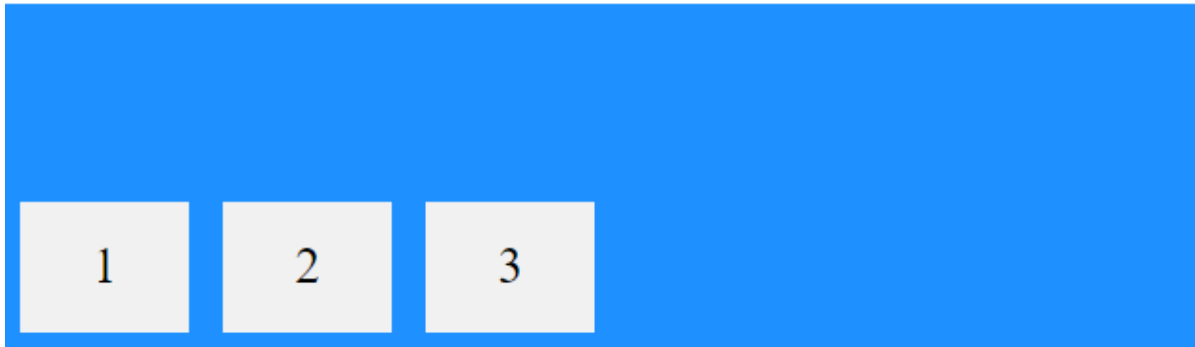



Fig. 10 (Flex Container)

Example :

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

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

The CSS Flexbox Container Properties :

The following table lists all the CSS Flexbox Container properties:

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

Task 01: Web Page using Flex Box

[40 minutes / 40 marks]

See the templates. Make the web page of like this:

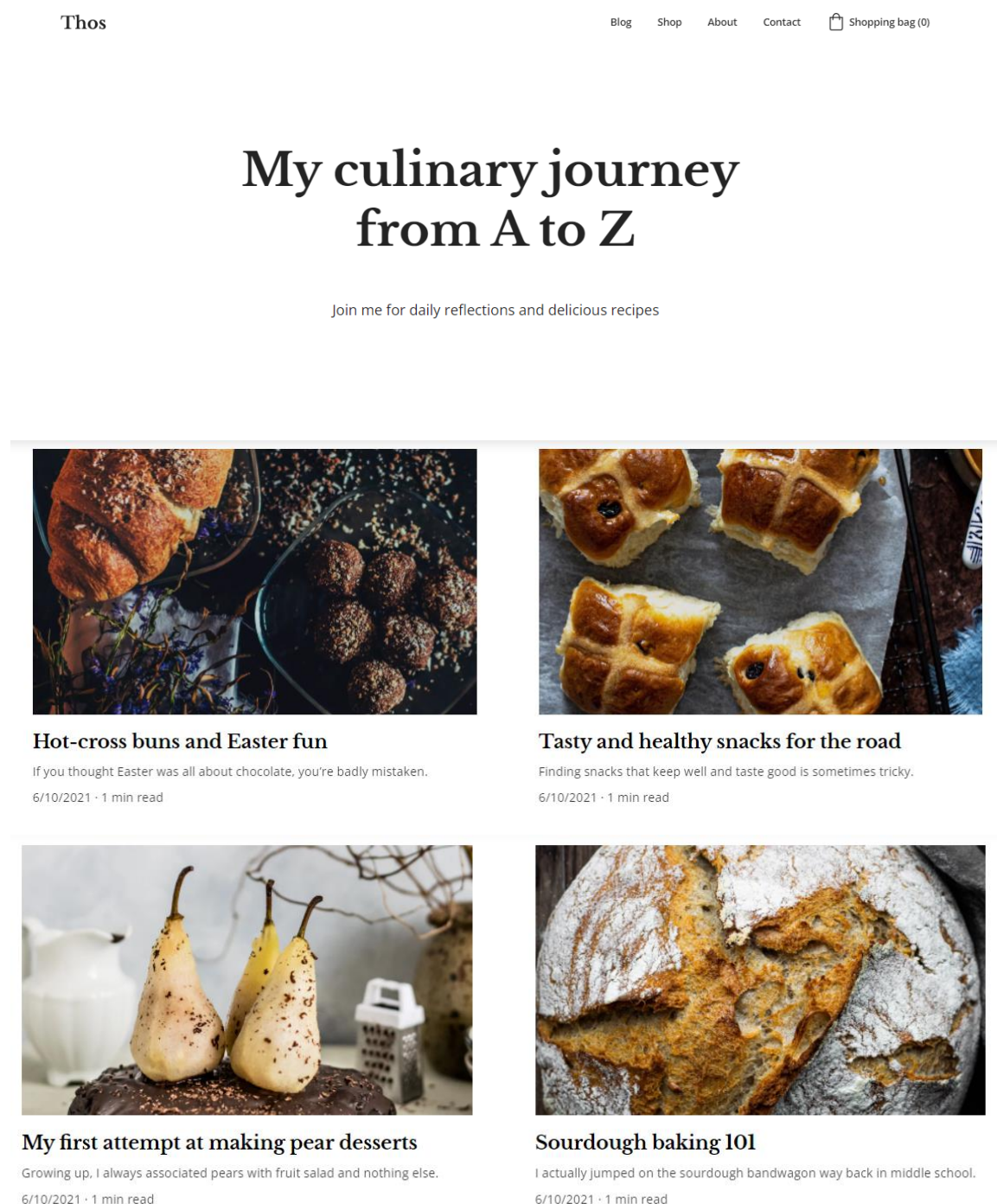


Fig. 11 (Pre-Lab Task 01)

Open Text editor, starting coding this document. Check the output step by step in browser by saving file with .html extension.

Submit the file of name “flexbox_rollno.html”.

In-Lab Activities:

CSS Media Queries

CSS2 Introduced Media Types:

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

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

CSS3 Introduced Media Queries:

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

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

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

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

Min Width to Max Width:

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

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

Media Queries Simple Examples:

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

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

Example:

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

Media Queries for Menus:

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



Fig. 12 (Media Query)

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

```
/* Navbar links */
.topnav a {
  float: left;
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
```

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

Media Queries for Columns:

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

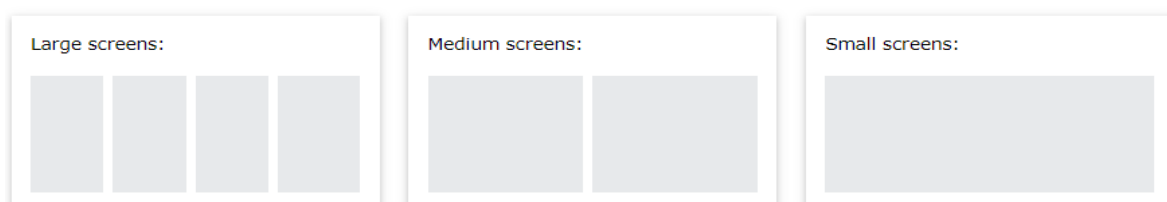


Fig. 13 (Media Query)

Example:

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

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

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

Flex Box Responsive:

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

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

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

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

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

JavaScript HTML DOM:

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model), When a web page is loaded, the browser creates a Document Object Model of the page.

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

What You Will Learn

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

What is the HTML DOM?

The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

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

JavaScript - HTML DOM Methods:

HTML DOM methods are actions you can perform (on HTML Elements).

HTML DOM properties are values (of HTML Elements) that you can set or change.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Including JavaScript in Your Page:

Including JavaScript in your page is a fairly simple process.

You can include JavaScript in your HTML in two ways:

- Writing the code in your HTML
- Including it as a link to an external file

For the most part, you will include the JavaScript as an external file.

The Script Tag

The `<script>` tag is what we use to includes our JavaScript. It's a lot like the `<link>` tag you've already been using to include your CSS files.

Here's a very basic snippet of JavaScript using the script tag. This JavaScript is written directly into our HTML page. It will call and alert box as soon as the page loads.

```
<script type="text/javascript">
    alert("This alert box was called with the onload event");
</script>
```

When using the script tag, we must always use the attribute name and value of `type="text/javascript"`.

Using the script tag to include an external JavaScript file

To include an external JavaScript file, we can use the script tag with the attribute `src`. You've already used the `src` attribute when using images. The value for the `src` attribute should be the path to your JavaScript file.

```
<script type="text/javascript" src="path-to-javascript-file.js"></script>
```

This script tag should be included between the `<head>` tags in your HTML document.

Example Code:

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

<h2>My First Page</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

My First Page

Hello World!

Fig. 14 (Script Tag)

In the example above, `getElementById` is a method, while `innerHTML` is a property.

The `getElementById` Method

The most common way to access an HTML element is to use the id of the element. In the example above the `getElementById` method used `id="demo"` to find the element.

The `innerHTML` Property:

The easiest way to get the content of an element is by using the `innerHTML` property. The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The HTML DOM Document Object

The document object represents your web page. If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
<code>document.getElementById(<i>id</i>)</code>	Find an element by element id
<code>document.getElementsByTagName(<i>name</i>)</code>	Find elements by tag name
<code>document.getElementsByClassName(<i>name</i>)</code>	Find elements by class name

Changing HTML Elements:

Property	Description
<code>element.innerHTML = new html content</code>	Change the inner HTML of an element
<code>element.attribute = new value</code>	Change the attribute value of an HTML element
<code>element.style.property = new style</code>	Change the style of an HTML element
Method	Description
<code>element.setAttribute(<i>attribute</i>, <i>value</i>)</code>	Change the attribute value of an HTML element

Adding Events Handlers:

Method	Description
<code>document.getElementById(<i>id</i>).onclick = function(){<i>code</i>}</code>	Adding event handler code to an onclick event

JavaScript HTML DOM - Changing HTML:

The HTML DOM allows JavaScript to change the content of HTML elements.

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the `innerHTML` property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = new HTML
```

This example changes the content of a `<p>` element:


```

<!DOCTYPE html>
<html>
<body>

<h2>JavaScript can Change HTML</h2>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>

```

JavaScript can Change HTML

New text!

The paragraph above was changed by a script.

Fig. 15 (HTML Element)

Example explained:

- The HTML document above contains a `<p>` element with `id="p1"`
- We use the HTML DOM to get the element with `id="p1"`
- A JavaScript changes the content (innerHTML) of that element to "New text!"

This example changes the content of an `<h1>` element:

```

<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old Heading</h1>

<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>

<p>JavaScript changed "Old Heading" to "New Heading".</p>

</body>
</html>

```

New Heading

JavaScript changed "Old Heading" to "New Heading".

Fig. 16 (HTML DOM)

Example explained:

- The HTML document above contains an `<h1>` element with `id="id01"`
- We use the HTML DOM to get the element with `id="id01"`
- A JavaScript changes the content (innerHTML) of that element to "New Heading"

Changing the Value of an Attribute :

To change the value of an HTML attribute, use this syntax:

document.getElementById(id).attribute = new value

This example changes the value of the `src` attribute of an `` element:

```

<!DOCTYPE html>
<html>
<body>

```

```


<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an element with id="myImage"
- We use the HTML DOM to get the element with id="myImage"
- A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

JavaScript HTML DOM - Changing CSS:

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

The following example changes the style of a <p> element:

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

</body>
</html>
```

Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

Before Clicked:

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

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

My Heading 1

Click Me!

Fig. 17 (On Click)

After Clicked:

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

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

My Heading 1

Click Me!

Fig. 18 (On Click)

Task 01: HTML DOM (I)

[10 minutes / 10 marks]

See the templates. Make the web page of like this.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8 />
    <title>JS DOM paragraph style</title>
  </head>
  <body>
    <p id='text'>JavaScript Exercises - w3resource</p>
    <div>
      <button id="jsstyle">Style</button>
    </div>
  </body>
</html>
```

Here is a sample html file with a submit button. Now modify the style of the paragraph text through javascript code.

Clicking on the button the font, font size, and color of the paragraph text should be changed.

Task 02: HTML DOM (II)

[10 minutes / 10 marks]

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8 />
    <title>Return first and last name from a form - w3resource</title>
  </head>
  <body>
    <h1 id= "first"></h1>
    <h1 id= "last"></h1>
    <form id="form1">
      First name: <input type="text" name="fname" value="David"><br>
      Last name: <input type="text" name="lname" value="Beckham"><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Write a JavaScript code to get the values of First and Last name of the following form and write in following h1 tags.

Task 03: HTML DOM (III)

[5 minutes / 5 marks]

Write a JavaScript program to set the background color of a paragraph in Task 01.

Task 04: HTML DOM (IV)

[10 minutes / 10 marks]

Write a JavaScript code to highlight the bold words of the following paragraph, on mouse over a certain link. Go to the editor

Sample text: [On mouse over here bold words of the following paragraph will be highlighted by changing color of text]

“Be willing to be a **beginner** every single morning.” “The **master** has **failed** more times than the **beginner** has even tried.” “Every **beginner** possesses a **great potential** to be an **expert** in his or her **chosen field**.”

Post-Lab Activities:**Task 01: Develop a Static Website****[300 minutes / 100 marks]**

Develop a Static Website. It must have at least 3 Pages. There should be a links to navigate through pages. Make header and footer for pages. Try to use all concepts and element in your website.

It must have separate CSS and JS files.

Submissions:

- For Pre-Lab Activity:
 - Perform Pre-Lab as mentioned above.
 - Save respective document in folder “RollNo_Pre-Lab_14”.
 - Then zip whole folder (RollNo_Pre-Lab_14.zip), and email to your respective TA.
- For In-Lab:
 - Perform mentioned tasks of In-Lab activity.
 - Make a folder on Desktop by name “RollNo_In-Lab_14”.
 - Then save each document in folder “RollNo_In-Lab_14”.
- For Post-Lab Activity:
 - Perform Post-Lab as mentioned above.
 - Save respective document in folder “RollNo_Post-Lab_14”.
 - Then zip whole folder (RollNo_Pre-Lab_14.zip), and email to your respective TA.

Evaluations Metric:

- All the Lab tasks will be evaluated offline by TA's.
- Division of Pre-Lab tasks: [40 marks]
 - Task 01 Develop Web Page using Flex Box [40 marks]
- Division of In-Lab tasks: [35 marks]
 - Task 01: HTML DOM (I) [10 marks]
 - Task 02: HTML DOM (II) [10 marks]
 - Task 03: HTML DOM (III) [05 marks]
 - Task 04: HTML DOM (IV) [10 marks]
- Division of Post-Lab tasks: [100 marks]
 - Task 01: Develop a Static Website [100 marks]

References and Additional Material:

- Jennifer Niederst Robbins, Learning, Web Design A Beginner's Guide To HTML, CSS, JavaScript, and Web Graphics, 5th Edition, ISBN: 978-1-491-96020-2.
https://drive.google.com/drive/u/5/folders/1V9nh8WIKOIQvi_ig98_YCaP7Vvei-tQz
- Jeremy Osborn, Jennifer Smith, Web Design with HTML and CSS Digital Classroom, 2011, ISBN: 978-0-470-58360-9.
https://drive.google.com/drive/u/5/folders/1V9nh8WIKOIQvi_ig98_YCaP7Vvei-tQz
- J. M. Gustafson - HTML5 Web Application Development by Example Beginner's Guide-Packt Publishing, 2013, ISBN 978-1-84969-594-7.
https://drive.google.com/drive/u/5/folders/1V9nh8WIKOIQvi_ig98_YCaP7Vvei-tQz
- Learn to Code
www.w3schools.com
- Learn HTML:
<https://html.com>
- Learn CSS:
<https://web.dev/learn/css/>

Lab Time and Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Settlement and attendance
- Slot – 02 – 00:15 – 00:30: Discussion on topics, some nouns and context
- Slot – 03 – 00:30 – 00:45: Demonstration on screen (Flex Box CSS)
- Slot – 04 – 00:45 – 01:00: Demonstration on screen (Flex Box CSS)
- Slot – 11 – 01:00 – 01:15: Demonstration on screen (JavaScript)
- Slot – 06 – 01:15 – 01:30: Demonstration on screen (JavaScript)
- Slot – 07 – 01:30 – 01:45: Demonstration on screen (JavaScript)
- Slot – 08 – 01:45 – 02:00: Give Tasks and discussion on each task
- Slot – 09 – 02:00 – 02:15: Activity time slot (Task 01 & Task 02)
- Slot – 10 – 02:15 – 02:30: Activity time slot (Task 03 & Task 04)
- Slot – 11 – 02:30 – 02:45: Evaluate In-Lab
- Slot – 12 – 02:45 – 03:00: Evaluation and Next Instructions