Instructor: Sudad H. Abed / Email: shabed@uoanbar.edu.iq

Lecture Reference: www.w3schools.com

❖ CSS:

- **CSS** stands for **C**ascading **S**tyle **S**heets.
- **CSS describes** how HTML elements are to be **displayed.**
- **CSS saves a lot of work**. It can control the layout of multiple web pages all at once.
- External stylesheets are stored in **CSS files**

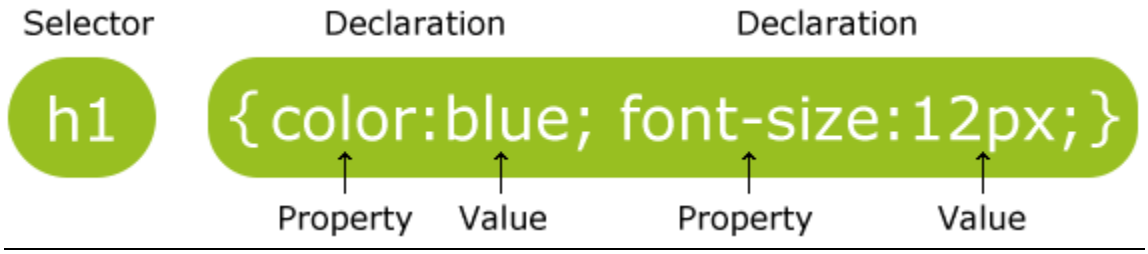
- **CSS Solved a Big Problem (how):**
  - HTML was created to **describe the content** of a web page, like:

    <h1>This is a heading</h1>

    This is a paragraph.
  - HTML was NEVER intended to contain tags for formatting a web page!
  - When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.
  - ✓ To solve this problem, the World Wide Web Consortium (W3C) created CSS. CSS removed the style formatting from the HTML page!

❖ CSS Syntax and Selectors:

A CSS rule-set consists of a selector and a declaration block:



- ✓ 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.
- ✓ A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

❖ CSS Syntax and Selectors:

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

- **The element Selector:**

  The element selector selects elements based on the element name.

  You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

Ex:

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

- **The id Selector:**

  - The id selector uses the id attribute of an HTML element to select a specific element.
  - The id of an element should be 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.
  - The style rule below will be applied to the HTML element with id="para1":

Ex:

```
#para1 {
    text-align: center;
    color: red;
}
```

Note: An id name cannot start with a number!

- **The class Selector:**
  - The class selector selects elements with a specific class attribute.
  - To select elements with a specific class, write a period (.) character, followed by the name of the class.
  - In the example below, all HTML elements with class="center" will be red and center-aligned:

Ex:

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

You can also specify that only specific HTML elements should be affected by a class. In the example below, only <p> elements with class="center" will be center-aligned:

Ex:

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

HTML elements can also refer to more than one class. In the example below, the <p> element will be styled according to class="center" and to class="large":

Ex:

```
<p class="center large">This paragraph refers to two classes.</p>
```

Note: An id name cannot start with a number!

- **Grouping Selectors:**

If you have elements with the same style definitions, It will be better to group the selectors, to minimize the code.

- To group selectors, separate each selector with a comma.

Ex:

```
h1, h2, p {
   text-align: center;
   color: red;
}
```

- **CSS Comments:**

A CSS comment starts with /* and ends with */. Comments can also span multiple lines:

Ex:

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

/* This is
a multi-line
comment */
```

❖ Ways to Insert CSS:

There are three ways of inserting a style sheet:

- ✓ External style sheet
- ✓ Internal style sheet
- ✓ Inline style

- **External Style Sheet:**

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

➢ Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

Ex:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

- An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.
- Here is how the "mystyle.css" looks:

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

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

Note: Do not add a space between the property value and the unit (such as margin-left: 20 px;). The correct way is: margin-left: 20px;

- **Internal Style Sheet:**
  - An internal style sheet may be used if one single page has a unique style.
  - Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

    Ex:

```
<head>
<style>
body {
    background-color: linen;
}

h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

- **Inline Styles:**

  - 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.
  - The example below shows how to change the color and the left margin of a <h1> element:

    Ex:

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>
```

- **Multiple Style Sheets:**

  - If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used.

Ex:

Assume that an external style sheet has the following style for the <h1> element:

```
h1 {
    color: navy;
}
```

then, assume that an internal style sheet also has the following style for the <h1> element:

```
h1 {
    color: orange;
}
```

If the internal style is defined after the link to the external style sheet, the <h1> elements will be "orange":

Ex:

```
<head>
   <link rel="stylesheet" type="text/css" href="mystyle.css">
   <style>
      h1 {
          color: orange;
      }
   </style>
</head>
```

However, if the internal style is defined before the link to the external style sheet, the <h1> elements will be "navy":

Ex:

```
<head>
   <style>
       h1 {
           color: orange;
       }
    </style>
    <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

- **Cascading Order:**

What style will be used when there is more than one style specified for an HTML element?

Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style (inside a specific HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or a browser default value.

❖ CSS Backgrounds:

The CSS background properties are used to define the background effects for elements.

CSS background properties:

- ✓ background-color
- ✓ background-image
- ✓ background-repeat
- ✓ background-attachment
- ✓ background-position

- **Background-Color:**

➢ The background-color property specifies the background color of an element.

The background color of a page is set like this:

Ex:

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

✓ With CSS, a color is most often specified by:

- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

- **Background-Image:**

➢ The background-image property specifies an image to use as the background of an element.

➢ By default, the image is repeated so it covers the entire element.

The background image for a page can be set like this:

Ex:

```
body {
    background-image: url("paper.gif");
}
```

➢ **background-repeat:**

➢ By default, a background-image is repeated both vertically and horizontally.

➢ The background-repeat property sets if/how a background image will be repeated.

```
background-repeat: no-repeat;  /* No repeat */
background-repeat: repeat-y; /* Repeat vertically */
background-repeat: repeat-x; /* Repeat horizontally */
```

Ex:

```
body {
    background-image: url("paper.gif");
    background-repeat: no-repeat;
}
```

- **background- attachment:**

The background-attachment property sets whether a background image is fixed or scrolls with the rest of the page.

Ex:

```
body {
    background-image: url('w3css.gif');
    background-repeat: no-repeat;
    background-attachment: fixed;
}
```

- **background- position:**

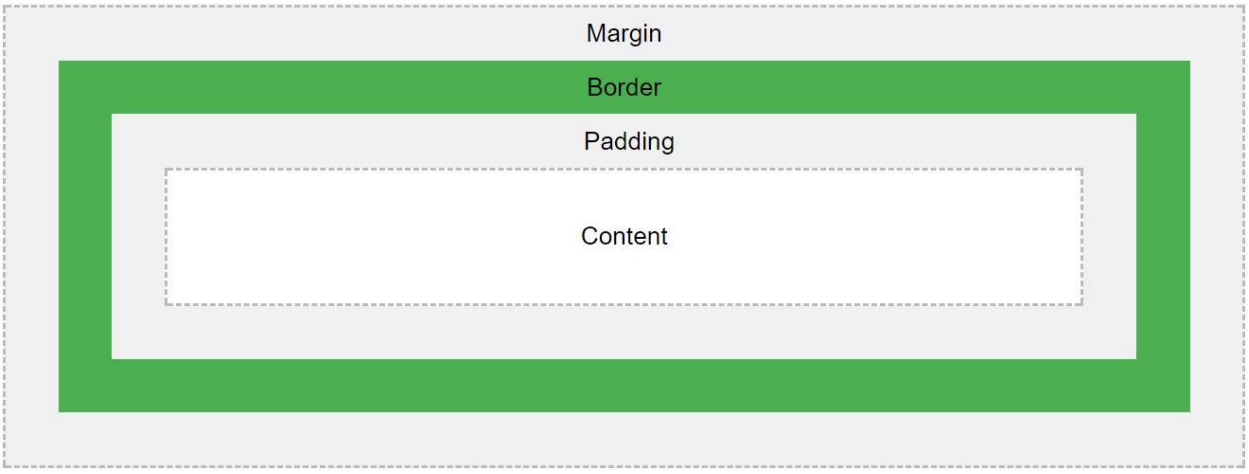
The background-position property sets the starting position of a background image.

Ex:

```
body {
    background-image: url('smiley.gif');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-position: center;
}
```

❖ CSS Box Model:

➢ 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.

Ex:

```
div {
    width: 300px;
    border: 25px solid green;
    padding: 25px;
    margin: 25px;
}
```

❖ CSS Borders:

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

• Border Style:

The border-style property specifies what kind of border to display.

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

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

Ex:

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
```

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

- **Border Width:**

➢ The border-width property specifies the width of the four borders.

➢ The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick.

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

Ex:

```
p.one {
    border-style: solid;
    border-width: 5px;
}
p.two {
    border-style: solid;
    border-width: medium;
}
p.three {
    border-style: solid;
    border-width: 2px 10px 4px 20px;
}
```

- **Border Color:**

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

➢ If border-color is not set, it inherits the color of the element.

Ex:

```
p.one {
    border-style: solid;
    border-color: red;
}
p.three {
    border-style: solid;
    border-color: red green blue yellow;
}
```

You can also specify all the individual border properties for just one side:

Ex:

```
p {
    border-left: 6px solid red;
    background-color: lightgrey;
}
```

❖ CSS Margins:

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

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

- **Margin - Individual Sides:**

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

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

Ex:

```
p {
    margin-top: 100px;
    margin-bottom: 100px;
    margin-right: 150px;
    margin-left: 80px;
}
```

❖ CSS Padding:

- ➢ The CSS padding properties are used to generate space around an element's content, inside of any defined borders.
- ➢ With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

- **Padding - Individual Sides:**

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

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

Ex:

```
div {
    padding-top: 50px;
    padding-right: 30px;
    padding-bottom: 50px;
    padding-left: 80px;
}
```

❖ CSS Font:

The CSS font properties define the **font family**, **boldness**, **size**, and the **style** of a text.

• Font Family:

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

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

Ex:

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

• Font Style:

- The font-style property is mostly used to specify italic text.

Ex:

```
p.normal {
    font-style: normal;
}

p.italic {
    font-style: italic;
}
```

• Font Size:

- The font-size property sets the size of the text.
- Set font size with pixels.

Ex:

```
p {
    font-size: 14px;
}
```

- **Font Weight:**

➢ The font-weight property specifies the weight of a font:

Ex:

```
p.normal {
    font-weight: normal;
}

p.thick {
    font-weight: bold;
}
```

❖ CSS Links:

➢ With CSS, links can be styled in different ways.

- **Styling Links:**

➢ Links can be styled with any CSS property (e.g. color, font family, background, etc.).

Ex:

```
a {
    color: hotpink;
}
```

➢ In addition, links can be styled differently depending on what **state** they are in. The four links states are:

- a:link - a normal, unvisited link.
- a:visited - a link the user has visited.
- a:hover - a link when the user mouses over it.
- a:active - a link the moment it is clicked.

Ex:

```
/* unvisited link */
a:link {
    color: red;
}
/* visited link */
a:visited {
    color: green;
}
/* mouse over link */
a:hover {
    color: hotpink;
}
```

```
/* selected link */
a:active {
    color: blue;
}
```

Note: When setting the style for several link states, there are some order rules:

- ✓ a:hover **MUST** come after a:link and a:visited
- ✓ a:active **MUST** come after a:hover

- **Advanced - Link Buttons:**

➢ This example demonstrates a more advanced example where we combine several CSS properties to display links as boxes/buttons:

Ex:

```
a:link, a:visited {
    background-color: #f44336;
    color: white;
    padding: 14px 25px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
}

a:hover, a:active {
    background-color: red;
}
```

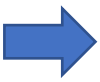
❖ CSS Tables:

➢ The look of an HTML table can be greatly improved with CSS:

- **Table Borders:**
  - ➢ The example below specifies a black border for <table>, <th>, and <td> elements:

Ex:

```
table, th, td {
   border: 1px solid black;
}
```



Firstname	Lastname
Peter	Griffin
Lois	Griffin

- **Table Borders:**
  - ➢ The border-collapse property sets whether the table borders should be collapsed into a single border:

Ex:

```
table {
    border-collapse: collapse;
}
table, th, td {
    border: 1px solid black;
}
```

Firstname	Lastname
Peter	Griffin
Lois	Griffin

- **Table Horizontal Alignment:**

- The text-align property sets the horizontal alignment (like left, right, or center) of the content in <th> or <td>.
- By default, the content of <th> elements are center-aligned and the content of <td> elements are left-aligned.

  Ex:

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

- **Hoverable Table:**

- Use the :hover selector on <tr> to highlight table rows on mouse over:

  Ex:

```
tr:hover {background-color: #f5f5f5;}
```

First Name	Last Name
Peter	Griffin
Lois	Griffin
Joe	Swanson

- **Striped Tables:**

- For zebra-striped tables, use the nth-child() selector and add a background-color to all even (or odd) table rows:

  Ex:

```
tr:nth-child(even) {background-color: #f2f2f2;}
```

First Name	Last Name
Peter	Griffin
Lois	Griffin
Joe	Swanson

- **Responsive Table:**

➢ A responsive table will display a horizontal scroll bar if the screen is too small to display the full content:

➢ Add a container element (like) with overflow-x:auto around the <table> element to make it responsive:

Ex:

```
<div style="overflow-x:auto;">
   <table>
    ... table content ...
   </table>
</div>
```

❖ CSS Position:

➢ The position property specifies the type of positioning method used for an element.

There are five different position values:

- static
- relative
- fixed
- absolute
- sticky

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

position: static;

➢ HTML elements are positioned static by default.

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

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

Ex:

```
div.static {
    position: static;
    border: 3px solid #73AD21;
}
```

position: relative;

- An element with position: relative; is positioned relative to its normal position.
- Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

  Ex:

```
div.relative {
    position: relative;
    left: 30px;
    border: 3px solid #73AD21;
}
```

position: fixed;

- An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.
- A fixed element does not leave a gap in the page where it would normally have been located.

  Ex:

```
div.fixed {
    position: fixed;
    bottom: 0;
    right: 0;
    width: 300px;
    border: 3px solid #73AD21;
}
```

position: absolute;

- An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
- However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.
- **Note:** A "positioned" element is one whose position is anything except static.

Ex:

```
div.relative {
    position: relative;
    width: 400px;
    height: 200px;
    border: 3px solid #73AD21;
}

div.absolute {
    position: absolute;
    top: 80px;
    right: 0;
    width: 200px;
    height: 100px;
    border: 3px solid #73AD21;
}
```

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).
- In this example, the sticky element sticks to the top of the page (top: 0), when you reach its scroll position.

Ex:

```
div.sticky {
    position: -webkit-sticky; /* Safari */
    position: sticky;
    top: 0;
    background-color: green;
    border: 2px solid #4CAF50;
}
```

End of CSS Lectures

❖ HTML Forms:

- **The <form> Element:**

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

```
<form>
.
form elements
.
</form>
```

  - ➢ An HTML form contains form elements.
  - ➢ Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

❖ HTML Form Elements:

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

- **The <text> Element:**
  The **<text>** element defines a one-line input field.

Ex:

```
<form action="#" >
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

First name:

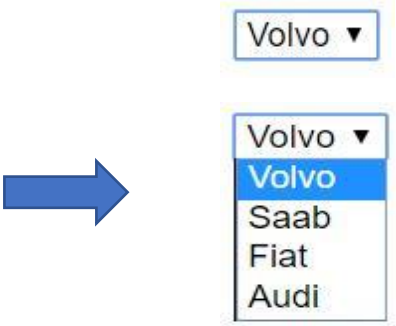
Last name:

- **The <select> Element:**

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

Ex:

```
<form action="#">
  <select name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
</form >
```



- The <option> elements defines an option that can be selected.
- By default, the first item in the drop-down list is selected.
- To define a pre-selected option, add the selected attribute to the option:

Ex:

```
<form action="#">
  <select name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat" selected >Fiat</option>
    <option value="audi">Audi</option>
  </select>
</form >
```

- Visible Values: use the **size** attribute to specify the number of visible values.

Ex:

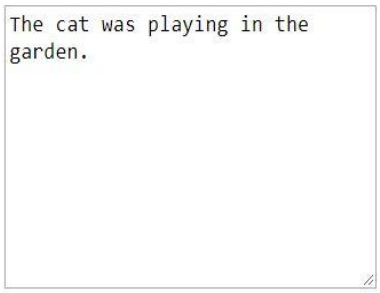
```
<form action="#">
  <select name="cars" size="3">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
</form >
```

- **The <textarea> Element:**

The **<textarea>** element defines a multi-line input field (a text area):

Ex:

```
<form action="#">
  <textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
</form >
```



- The **rows** attribute specifies the visible number of lines in a text area.
- The **cols** attribute specifies the visible width of a text area.

- **The <button> Element:**

The **<button>** element defines a clickable **button**:

Ex:

```
<form action="#">
   <button type="button" onclick="alert('Hello World!')">Click
Me!</button>
</form >
```

❖ HTML Input Types:

1- Input Type Text:

<input type="text"> defines a **one-line text input field**:

Ex:

```
<form action="#" >
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

First name:

Last name:

2- Input Type Password:

<input type="password"> defines **a password field:**

Ex:

```
<form action="#" >
   User name:<br>
  <input type="text" name="username"><br>
  User password:<br>
  <input type="password" name="psw">
</form>
```

User name:

User password:

••••••

3- Input Type Submit:

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

- The **form-handler** is typically a server page with a script for processing input data.
- The **form-handler** is specified in the form's action attribute:

Ex:

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
 <input type="submit" value="Submit">
</form>
```

4- Input Type Reset:

<input type="reset"> defines a **reset button** that will reset all form values to their default values:

Ex:

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
```

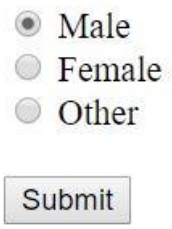
5- Input Type Radio:

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

- Radio buttons let a user select ONLY ONE of a limited number of choices.

Ex:

```
<form action="/action_page.php">
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
  <input type="submit">
</form>
```



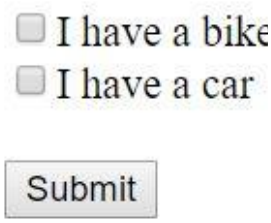
6- Input Type Checkbox:

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

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

Ex:

```
<form action="/action_page.php">
<input type="checkbox" name="vehicle1" value="Bike"> I have a bike<br>
<input type="checkbox" name="vehicle2" value="Car"> I have a car
<input type="submit">
</form>
```



- HTML5 added several new input types:
  - color
  - date
  - email
  - month
  - number
  - range
  - search
  - time
  - url
  - week

1- Input Type Color:

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

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

Ex:

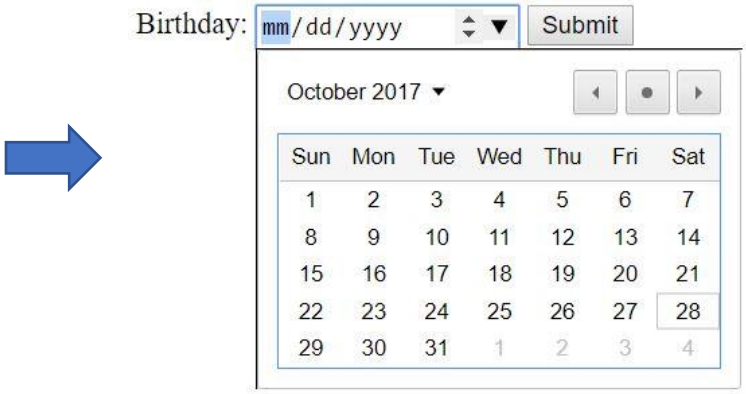
```
<form>
  Select your favorite color:
  <input type="color" name="favcolor">
  <input type="submit">
</form>
```

2- Input Type Date:

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

Ex:

```
<form>
  Birthday:
  <input type="date" name="bday">
  <input type="submit">
</form>
```

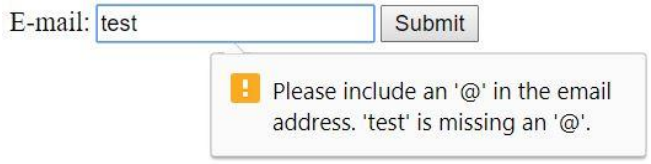


3- Input Type Email:

The **<input type="email">** is used for input fields that should contain an e-mail address.

Ex:

```
<form>
  E-mail:
  <input type="email" name="email">
  <input type="submit">
</form>
```



4- Input Type Number:

The **<input type="number">** defines a **numeric** input field.

- You can also set restrictions on what numbers are accepted.

Ex:

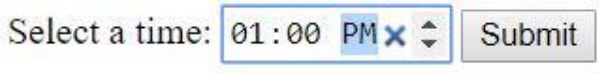
```
<form>
  Quantity (between 1 and 5):
  <input type="number" name="quantity" min="1" max="5">
  <input type="submit">
</form>
```

5- Input Type Time:

The **<input type="time">** allows the user to select a time (no time zone).

Ex:

```
<form>
  Select a time:
  <input type="time" name="usr_time">
   <input type="submit">
</form>
```



❖ HTML Input Attributes:

- **The value Attribute:**

The **value** attribute specifies the initial value for an input field:

Ex:

```
<form action="#">
First name:<br>
<input type="text" name="firstname" value="John">
</form>
```

First name:
John

- **The disabled Attribute:**

The **disabled** attribute specifies that the input field is disabled.

➢ A disabled input field is unusable and un-clickable, and its value will not be sent when submitting the form:

Ex:

```
<form action="#">
First name:<br>
<input type="text" name="firstname" value="John" disabled>
</form>
```

First name:
John

- **The size Attribute:**

The **size** attribute specifies the size (in characters) for the input field:

Ex:

```
<form action="">
First name:<br>
<input type="text" name="fname" value="John" size="40">
Last name:<br>
<input type="text" name="lname">
</form>
```

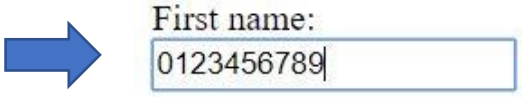
First name:
John
Last name:

- **The maxlength Attribute:**

The **maxlength** attribute specifies the maximum allowed length for the input field:

Ex:

```
<form action="">
First name:<br>
<input type="text" name="firstname" maxlength="10">
</form>
```

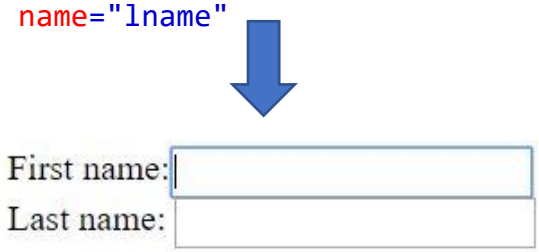


- **The autofocus Attribute:**

The **autofocus** attribute specifies that the input field should automatically get focus when the page loads.

Ex:

```
<form action="#">
First name:<input type="text" name="fname" autofocus><br>
First name:<input type="text" name="lname"
</form>
```



- **The placeholder Attribute:**

The **placeholder** attribute specifies a hint that describes the expected value of an input field (a sample value or a short description of the format).

- The hint is displayed in the input field before the user enters a value.
- The placeholder attribute works with the following input types: text, search, url, tel, email, and password.

Ex:

```
<form action="#">
  <input type="text" name="fname" placeholder="First name"><br>
  <input type="text" name="lname" placeholder="Last name"><br>
  <input type="submit" value="submit">
</form>
```

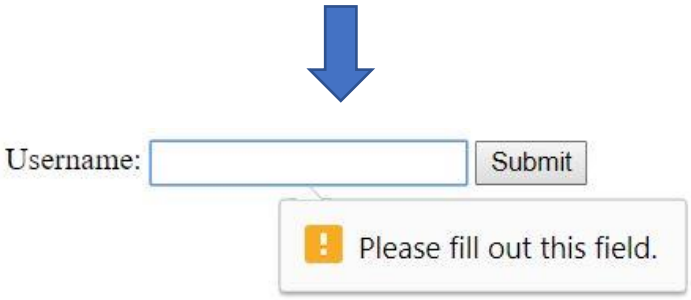
- **The required Attribute:**

The **required** attribute specifies that an input field must be filled out before submitting the form.

- The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

Ex:

```
<form action="#">
 Username: <input type="text" name="usrname" required><br>
 <input type="submit" value="submit">
</form>
```



End of HTML Lectures

JavaScript

JavaScript: Scripting language which is used to enhance the functionality and appearance of web pages.

❖ JavaScript Where To

- ### The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

Ex:

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

➢ **JavaScript Functions and Events**

✓ A JavaScript **function** is a block of JavaScript code, that can be executed when "called" for.

✓ A function can be called when an **event** occurs, like when the user clicks a button.

- ### JavaScript in <head> or <body>

✓ You can place any number of scripts in an HTML document.

✓ Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

Ex:

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Note: placing scripts at the bottom of the <body> element improves the display speed, because script compilation slows down the display.

- **External JavaScript**

Scripts can also be placed in external files:

Ex:

```
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

✓ External scripts are practical when the same code is used in many different web pages.

✓ JavaScript files have the file extension **.js**.

✓ To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

Ex:

```
<!DOCTYPE html>
<html>
<body>
  <script src="myScript.js"></script>
</body>
</html>
```

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page load

❖ What JavaScript Can Do

✓ One of many JavaScript HTML methods is **getElementById()**.

- **Change HTML Content**

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (**innerHTML**) to "Hello JavaScript"

Ex:

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

Note: JavaScript accepts both double and single quotes.

- **Change HTML Attributes**

  This example changes an HTML image by changing the src (source) attribute of an <img> tag:

  Ex:

```
<button
onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn
on the light</button>
<img id="myImage" src="pic_bulboff.gif" style="width:100px">
<button
onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Tur
n off the light</button>
```

- **Change HTML Styles (CSS)**

  Changing the style of an HTML element, is a variant of changing an HTML attribute:

  Ex:

```
document.getElementById("demo").style.fontSize = "35px";
```

- **Hide HTML Elements**

  Hiding HTML elements can be done by changing the display style:

  Ex:

```
document.getElementById("demo").style.display = "none";
```

- **Show HTML Elements**

  Showing hidden HTML elements can also be done by changing the display style:

  Ex:

```
document.getElementById("demo").style.display = "block";
```

❖ JavaScript Output

JavaScript can "display" data in different ways:

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

- **Using innerHTML**

✓ To access an HTML element, JavaScript can use the **document.getElementById (id)** method.

✓ The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

Ex:

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

- **Using document.write()**

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

Ex:

```
<script>
document.write(5 + 6);
</script>
```

- **Using window.alert()**

You can use an alert box to display data:

Ex:

```
<script>
window.alert(5 + 6);
</script>
```

- **Using console.log()**

For debugging purposes, you can use the **console.log()** method to display data.

Ex:

```
<script>
console.log(5 + 6);
</script>
```

❖ JavaScript Comments

- ✓ Single line comments start with //.
- ✓ Multi-line comments start with /* and end with */.

❖ JavaScript Variables

- ✓ All JavaScript **variables** must be **identified** with **unique names**.
- ✓ These unique names are called **identifiers**.
- ✓ A variable declared without a value will have the value undefined.

- **JavaScript Data Types**

JavaScript variables can hold numbers like 100 and text values like "John Doe".

Ex:

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

❖ JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

Ex:

```
var x = 5 + 2 + 3;
```

You can also add strings, but strings will be concatenated:

Ex:

```
var x = "John" + " " + "Doe";
```

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Ex:

```
var x = "5" + 2 + 3
```

But try This:

```
var x = 2 + 3 + "5";
```

- **JavaScript Comparison Operators**

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

❖ JavaScript Functions

✓ A JavaScript function is a block of code designed to perform a particular task.

✓ A JavaScript function is executed when "something" invokes it (calls it).

- **JavaScript Function Syntax**

✓ A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

✓ Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

✓ The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***

✓ The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

✓ Inside the function, the arguments (the parameters) behave as local variables.

❖ JavaScript Objects

✓ In real life, a car is an **object**.
✓ A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

- **JavaScript Objects**

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
var car = "Fiat";
```

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
var car = {type:"Fiat", model:"500", color:"white"};
```

The values are written as **name:value** pairs (name and value separated by a colon).

- **Accessing Object Properties**

You can access object properties in two ways:

objectName.propertyName → *Ex:* `person.lastName;`

Or

objectName["propertyName"] → *Ex:* `person["lastName"];`

- **Accessing Object Methods**

✓ A method is actually a function definition stored as a property value.

You access an object method with the following syntax:

objectName.methodName() → *Ex:* `name = person.fullName();`

❖ JavaScript Scope

Scope determines the accessibility (visibility) of variables.

In JavaScript there are two types of scope:

- Local scope
- Global scope

JavaScript has function scope: Each function creates a new scope.

- **Local JavaScript Variables**
  - ✓ Variables declared within a JavaScript function, become **LOCAL** to the function.
  - ✓ Local variables have **local scope**: They can only be accessed within the function.

Ex:

```
// code here can not use carName

function myFunction() {
    var carName = "Volvo";
    // code here can use carName
}
```

- **Global JavaScript Variables**
  - ✓ A variable declared outside a function, becomes **GLOBAL**.
  - ✓ A global variable has **global scope**: All scripts and functions on a web page can access it.

Ex:

```
var carName = " Volvo";
// code here can use carName

function myFunction() {
    // code here can use carName
}
```

- **The Lifetime of JavaScript Variables**
  - ✓ The lifetime of a JavaScript variable starts when it is declared.
  - ✓ Local variables are deleted when the function is completed.
  - ✓ In a web browser, global variables are deleted when you close the browser window (or tab), but remains available to new pages loaded into the same window.

❖ JavaScript Events

✓ HTML events are "**things**" that happen to HTML elements.

✓ When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.

- **HTML Events**

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

✓ JavaScript lets you execute code when events are detected.

✓ HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

1- With single quotes:

```
<element event='some JavaScript'>
```

2- With double quotes:

```
element event="some JavaScript">
```

In the following example, an onclick attribute (with code), is added to a button element:

Ex:

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">The time is?</button>
```

In the next example, the code changes the content of its own element (using **this**.innerHTML):

Ex:

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

- **Common HTML Events**

Here is a list of some common HTML events:

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

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

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

❖ JavaScript String Methods

String methods help you to work with strings.

Method	Description
length	The length property returns the length of a string.
indexOf()	The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string.
lastIndexOf()	The lastIndexOf() method returns the index of the last occurrence of a specified text in a string.
str.slice(x, y);	• slice() extracts a part of a string and returns the extracted part in a new string. • The method takes 2 parameters: the starting index (position), and the ending index (position). • If you omit the second parameter, the method will slice out the rest of the string.
str.substr(x, y);	substr() is similar to slice(). The difference is that the second parameter specifies the length of the extracted part.
str.replace(x , y)	The replace() method replaces a specified value with another value in a string.

Note:

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

❖ JavaScript Number Methods

Number methods help you work with numbers.

- **Number Properties**

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

✓ Number properties belongs to the JavaScript's number object wrapper called **Number**.

✓ These properties can only be accessed as **Number**.MAX_VALUE.

❖ JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

- **Math Object Methods**

Method	Description
abs(x)	Returns the absolute value of x
ceil(x)	Returns the value of x rounded up to its nearest integer
floor(x)	Returns the value of x rounded down to its nearest integer
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Returns the value of x rounded to its nearest integer

❖ JavaScript If...Else Statements

Conditional statements are used to perform different actions based on different conditions.

- **Conditional Statements**

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.

1- The if Statement

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

Ex:

```
if (hour < 18) {
    greeting = "Good day";
}
```

2- The else Statement

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

Ex:

```
if (hour < 18) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

3- The else if Statement

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

Ex:

```
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

4- The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Ex:

```
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";
}
```

❖ JavaScript For Loop

Loops can execute a block of code a number of times.

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
- **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

1- The For Loop

The for loop is often the tool you will use when you want to create a loop.

Ex:

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

2- The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

Ex:

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

3- The While Loop

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

Ex:

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

4- The Do/While Loop

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

Ex:

```
do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```

❖ JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.
Ex:

```
var cars = ["Saab", "Volvo", "BMW"];
```

- **Creating an Array**

Syntax:

```
var array_name = [item1, item2, ...];
```

- **Access the Elements of an Array**

✓ You refer to an array element by referring to the **index number**.

✓ This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

✓ This statement modifies the first element in cars:

```
cars[0] = "Opel";
```

Ex:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

- **Access the Full Array**

With JavaScript, the full array can be accessed by referring to the array name:

Ex:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

- **Array Properties and Methods**

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

✓ The length Property

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

Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;                          // the length of fruits is 4
```

✓ Looping Array Elements

The best way to loop through an array, is using a "for" loop:

Ex:

```
<p id="demo"></p>
<script>
      var fruits, text, fLen, i;
      fruits = ["Banana", "Orange", "Apple", "Mango"];
      fLen = fruits.length;
      text = "<ul>";
      for (i = 0; i < fLen; i++) {
          text += "<li>" + fruits[i] + "</li>";
      }
      text += "<ul>";
</script>
```

✓ Adding Array Elements

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

Ex:

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

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

Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";
```

- **Associative Arrays**

  ✓ Many programming languages support arrays with named indexes.

  ✓ Arrays with named indexes are called associative arrays (or hashes).

  ✓ JavaScript does **not** support arrays with named indexes.

  ✓ In JavaScript, **arrays** always use **numbered indexes**.

  Ex:

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;          // person.length will return 3
var y = person[0];              // person[0] will return "John"
```

❖ JavaScript Array Methods

- **Converting Arrays to Strings**

  The JavaScript method **toString()** converts an array to a string of (comma separated) array values.

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

  Result: `Banana,Orange,Apple,Mango`

  The **join()** method also joins all array elements into a string.

  It behaves just like toString(), but in addition you can specify the separator:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

  Result: `Banana * Orange * Apple * Mango`

- **Popping and Pushing**

  When you work with arrays, it is easy to remove elements and add new elements.

  Popping: The **pop()** method removes the last element from an array:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();          // Removes the last element ("Mango") from fruit
```

  The pop() method returns the value that was "popped out":

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();      // the value of x is "Mango"
```

  Pushing: The **push()** method adds a new element to an array (at the end):

  Ex:

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

  The push() method returns the new array length:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

- **Changing Elements**

  Array elements are accessed using their **index number**:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";   // Changes the first element of fruits to "Kiwi"
```

- **Deleting Elements**

  Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];

 // Changes the first element in fruits to undefined
```

- **Splicing an Array**

  - The **splice()** method can be used to add new items to an array:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

  Result: Banana,Orange,Lemon,Kiwi,Apple,Mango

  ✓ The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

  ✓ The second parameter (0) defines **how many** elements should be **removed**.

  ✓ The rest of the parameters ("Lemon", "Kiwi") define the new elements to be **added**.

- **Merging (Concatenating) Arrays**

  The **concat()** method creates a new array by merging (concatenating) existing arrays:

  Ex:

```
 var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);

  // Concatenates (joins) myGirls and myBoys
```

- **Sorting an Array**

  The **sort()** method sorts an array alphabetically:

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();            // Sorts the elements of fruits
```

  output**:** Apple,Banana,Mango,Orange

- **Reversing an Array**

  The **reverse()** method reverses the elements in an array.

  Ex:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();            // Sorts the elements of fruits
fruits.reverse();         // Reverses the order of the elements
```

  output: Orange,Mango,Banana,Apple

❖ JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

- **Alert Box**

✓ An alert box is often used if you want to make sure information comes through to the user.

✓ When an alert box pops up, the user will have to click "OK" to proceed.

Syntax:

```
window.alert("sometext");
```

The **window.alert()** method can be written without the window prefix.

Ex:

```
alert("I am an alert box!");
```

✓ To display line breaks inside a popup box, use a back-slash followed by the character n.

Ex:

```
alert("Hello\nHow are you?");
```

- **Confirm Box**

✓ A confirm box is often used if you want the user to verify or accept something.

✓ When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

✓ If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax:

```
window.confirm("sometext");
```

The **window.confirm()** method can be written without the window prefix.

Ex:

```
if (confirm("Press a button!") == true) {
    txt = "You pressed OK!";
} else {
    txt = "You pressed Cancel!";
}
```

- **Prompt Box**

✓ A prompt box is often used if you want the user to input a value before entering a page.

✓ When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

✓ If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax:

```
window.prompt("sometext","defaultText");
```

The **window.prompt()** method can be written without the window prefix.

Ex:

```
var person = prompt("Please enter your name", "Harry Potter");
if (person == null || person == "") {
    txt = "User cancelled the prompt.";
} else {
    txt = "Hello " + person + "! How are you today?";
}
```

❖ Timing Events

- ✓ The window object allows execution of code at specified time intervals.
- ✓ These time intervals are called timing events.
- ✓ The two key methods to use with JavaScript are:
  - ➢ **setTimeout(*function, milliseconds*)**
    Executes a function, after waiting a specified number of milliseconds.
  - ➢ **setInterval(*function, milliseconds*)**
    Same as setTimeout(), but repeats the execution of the function continuously.

- **The setTimeout() Method**

```
window.setTimeout(function, milliseconds);
```

- ✓ The **window.setTimeout()** method can be written without the window prefix.
- ✓ The first parameter is a function to be executed.
- ✓ The second parameter indicates the number of milliseconds before execution.

Ex:

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
    alert('Hello');
}
</script>
```

- **The setInterval() Method**

  ✓ The setInterval() method repeats a given function at every given time-interval.

```
window.setInterval(function, milliseconds);
```

  ✓ The **window.setInterval()** method can be written without the window prefix.

  ✓ The first parameter is the function to be executed.

  ✓ The second parameter indicates the length of the time-interval between each execution.

  Ex:

```
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var d = new Date();
    document.getElementById("demo").innerHTML =
d.toLocaleTimeString();
}
```