Module 2

Introduction to CSS and JavaScript

Badharudheen P
Asst. Prof. Dept. of CSE,
MESCE, Kuttippuram

Cascading Style Sheets

- CSS is used for formatting and presenting information.
- It allows you to specify the presentation of elements on a web page separately from the document's structure and content.
- This separation of structure from presentation simplifies maintaining and modifying web pages, especially on large-scale websites.

Cascading Style Sheets

- We can declare document styles in
  - **Inline Styles**
  - **Embedded Style Sheets (Internal Style Sheets)**
  - **Separate CSS Files (External Style Sheets)**.

Inline Styles

- Inline styles are used to declare an individual element's format using the HTML5 attribute style.
- Inline styles override any other styles applied (when conflicting styles occurs).
- Example:
  - ***<p style = "font-size: 20pt; color: deepskyblue;">***

- Each CSS property is followed by a colon and a value.
- Two properties are separated by a semi-colon.

Embedded Styles Sheets

- to embed a CSS3 document in document's head section.

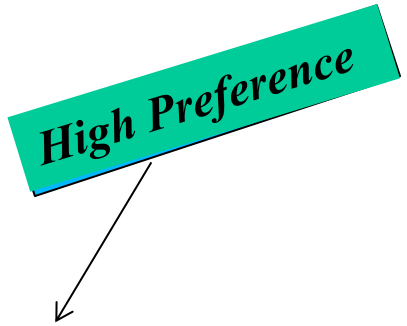
```
<head>
    <title>Embedded Style Sheet</title>
    <style type = "text/css">
        #age { font-weight: bold; color: black; }
        h1 { font-family: tahoma, sans-serif; }
        p { font-size: 12pt;font-family: arial, sans-serif; }
        .special { color: purple; }
    </style>
</head>
<body>
    <h1 class = "special">Deitel & Associates, Inc.</h1>
```

Conflicting Styles

- Most styles defined for parent elements are also inherited by child (nested) elements.
- But there are certain properties that you don't want to be inherited.
  - For example, the *background-image* property applied to body element.
- Properties defined for child elements have a higher precedence than parent elements.
- Conflicts are resolved in favor of properties with a higher precedence, so the child's styles take precedence.

Conflicting Styles

```
<style type = "text/css">
   em { font-weight: bold; color: black; }
   h1 { font-family: tahoma, sans-serif; }
   p { font-size: 12pt;font-family: arial, sans-serif; }
   .special { color: purple; }
</style>
<body>
   <p class = "special"> We are S7 <em> CSE
   </em>students of MESCE</p>
</body>
```



Linking External Style Sheets

- Separate documents that contain only CSS rules.
- With external style sheets you can provide a uniform look and feel to an entire website
- You can reuse the same style sheet across multiple websites.
- When changes to the styles are required, you need to modify only a single CSS file to make style changes across all the pages. This concept is known as **skinning**.

Linking External Style Sheets

```
/* External style sheet */
body { font-family: arial, sans-serif; }
li { font-weight: bold; }
h1, em { text-decoration: underline; }
ul { margin-left: 20px; }
```

External css file named styles.css

Linking external sheet to our page

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

CSS Selectors

- To select and manipulate HTML element(s).
  - based on their id, classes, types, attributes, values of attributes, elements, etc.

CSS Element Selectors

- **Based on id:**
- It uses the id attribute of an HTML element.
  - An id should be unique within a page.
- To find an element with a specific id, write a hash character, followed by the id of the element.
- Example

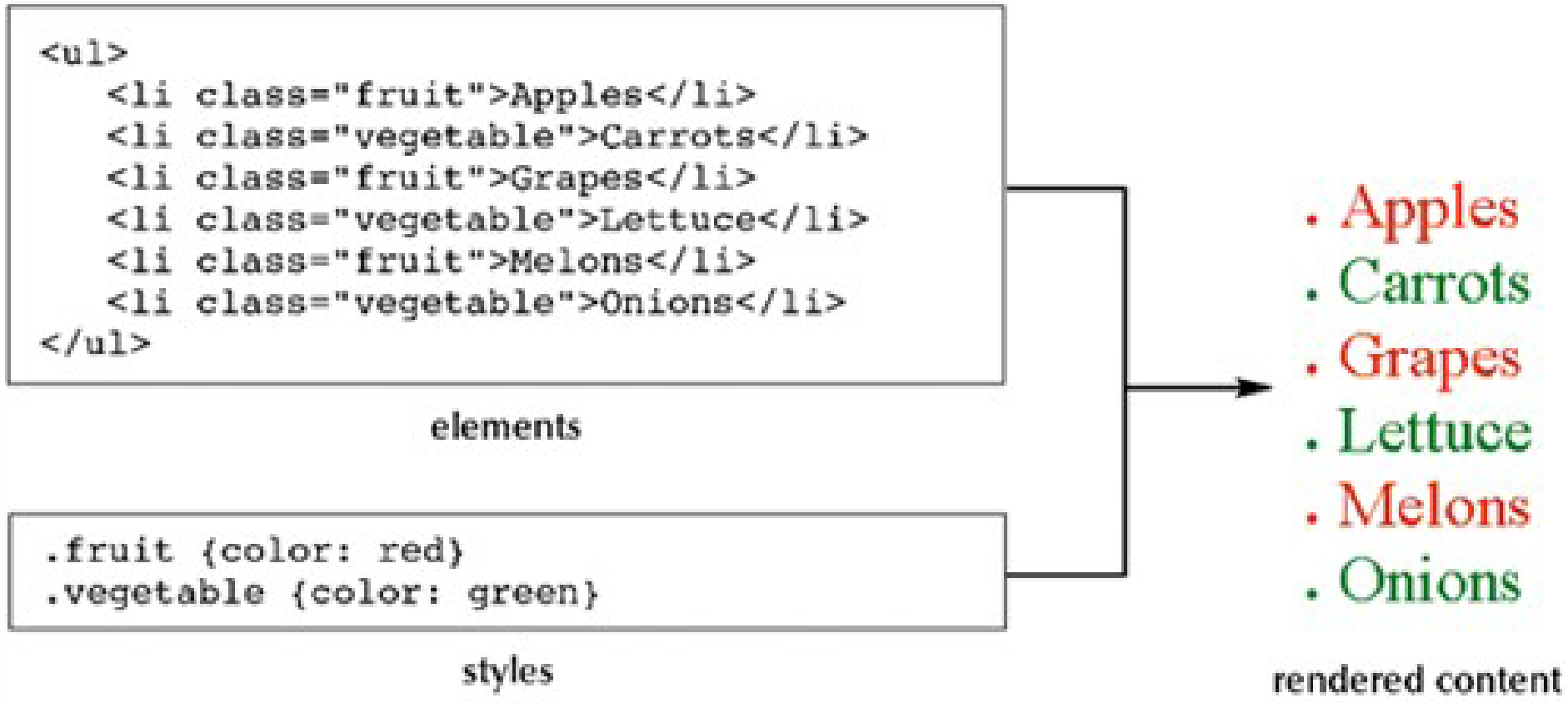
```
#para1
{
   text-align: center;
   color: red;
}
```

CSS Element Selectors

- **Based on class:**
- It uses the HTML class attribute.
- We can mark a group of elements with a common identifier using the class attribute.
- To find elements with a specific class, write a dot character, followed by the name of the class.

CSS Element Selectors

- **Based on class: Example**



CSS Element Selectors

Based on Element

- For eg:

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

- All **<p>** elements will be center-aligned, with a red text color

Grouping Selectors

- Elements with the same style can be grouped together

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

Grouping Selectors

- To minimize the code, you can group selectors.
- To group selectors, separate each selector with a comma.

- For eg:

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

Positioning Elements

- 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.

Positioning Elements

- ***position: static;***
  - 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.

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

Positioning Elements

- ***position: relative;***
  - Elements are 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.

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

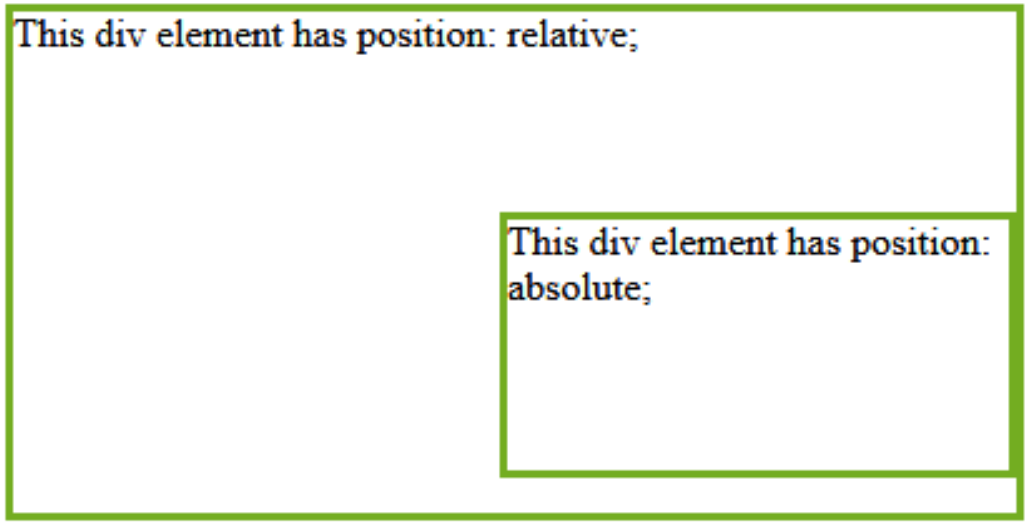
Positioning Elements

- ***position: fixed;***
  - The fixed element stays in the same place even if the page is **scrolled**.
  - The ***top, right, bottom,*** and ***left*** properties are used to position the element.

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

Positioning Elements

- ***position: absolute;***
  - The element is positioned relative to the nearest positioned ancestor.
  - Eg:- Two division tags: One inside another



Positioning Elements

- *position: absolute;*

```
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;
  }
```

Positioning Elements

- ***position: sticky;***
  - Element 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).

```
div.sticky {
    position: sticky;
    top: 0;       //stick at top when scroll
    padding: 5px;
    border: 2px solid #4CAF50;
    }
```

The `z-index` Property

- The ***z-index*** property allows you to layer overlapping elements.
- Elements that have higher z-index values are displayed in front of elements with lower z-index values.
- If you do not specify a z-index or if elements have the same zindex value, the elements are placed from background to foreground in the order in which they're encountered in the document.
- The default z-index value is 0.

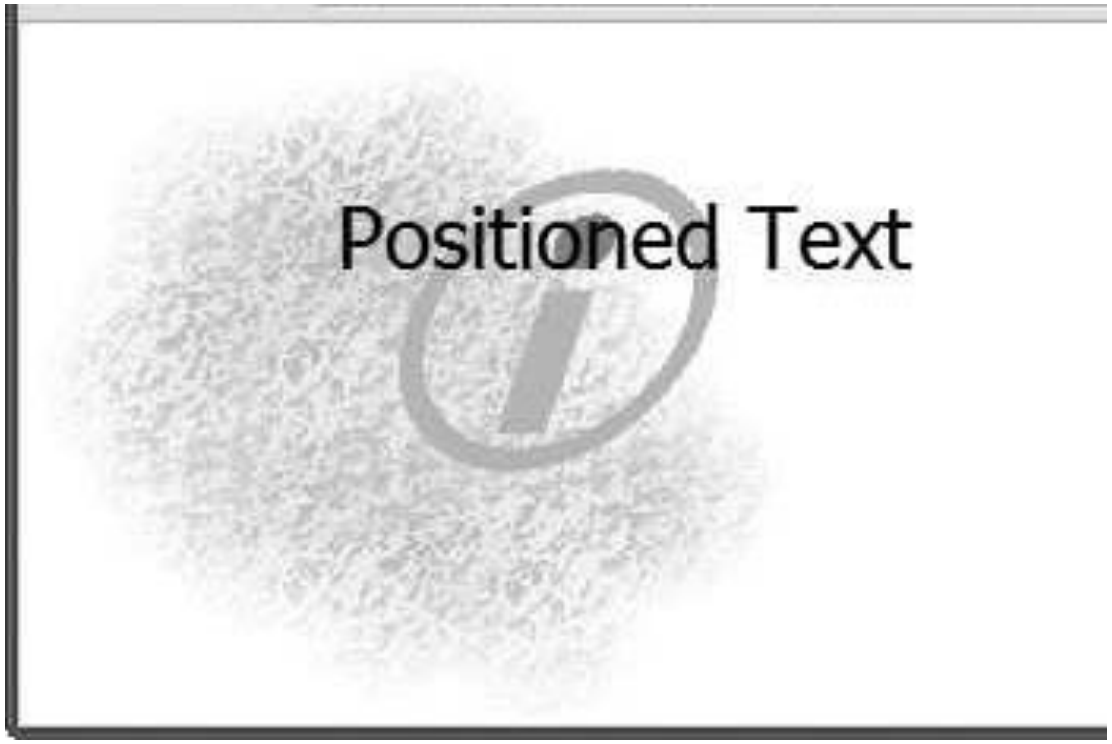
The z-index Property

```
<style type = "text/css">
   .background_image { position: absolute;
            top: 0px;
            left: 0px;
            z-index: 1; }
   .foreground_image { position: absolute;
            top: 25px;
            left: 100px;
            z-index: 2; }
   .text { position: absolute;
            top: 25px;
            left: 100px;
            z-index: 3; }
```

The z-index Property

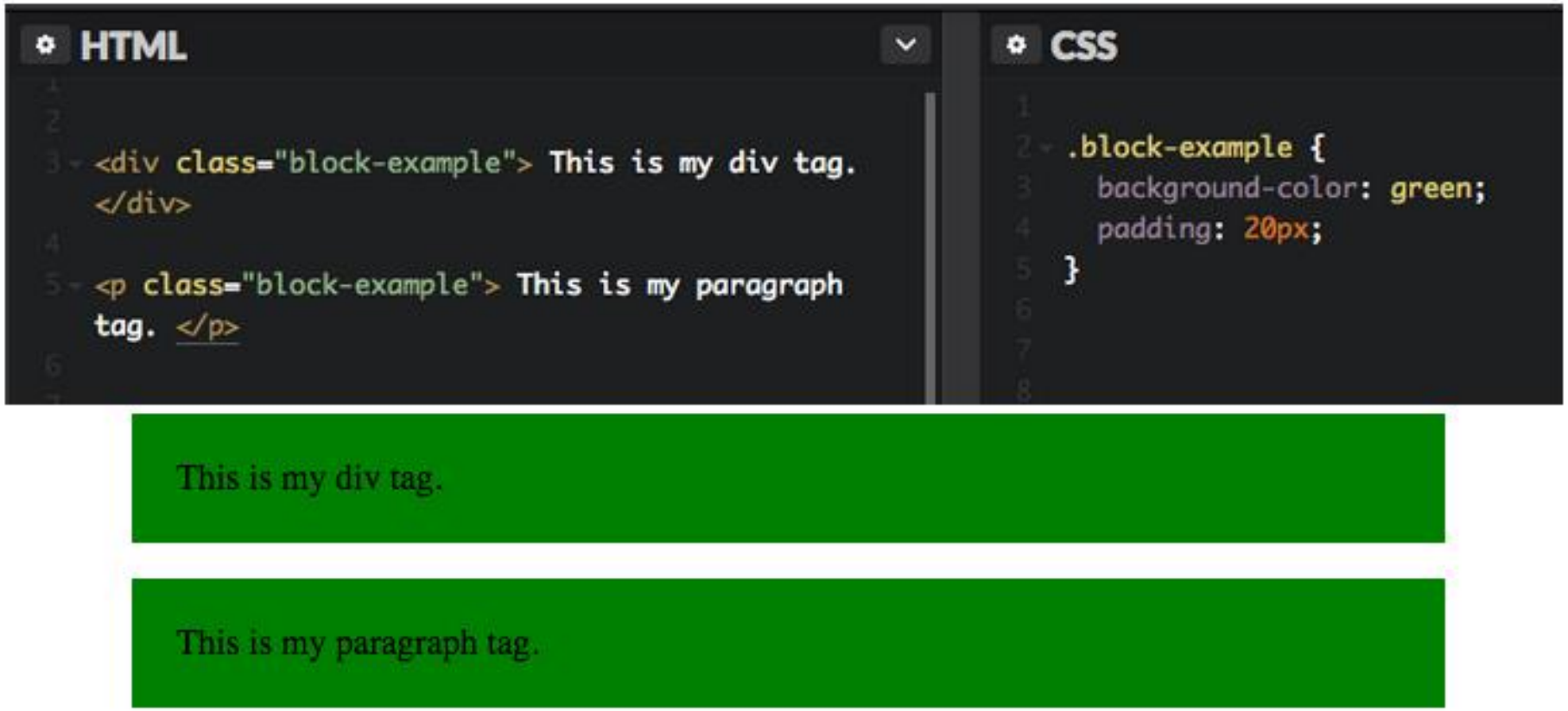
```
<body>
  <p><img src = "background_image.png" class =
  "background_image" alt = "First positioned image" /></p>
  <p><img src = "foreground_image.png" class =
  "foreground_image" alt = "Second positioned image" /></p>
  <p class = "text">Positioned Text</p>
</body>
```

The `z-index` Property



In-line and Block-Level Elements

- A block element always starts on a new line, and fills up the horizontal space left and right on the web page.



In-line and Block-Level Elements

- An in-line element does not cause a line break and does not take up the full width of a page.
- It is usually used within other HTML elements.
- Examples for Inline element: span, img, em, a, strong, etc.
  - Used for grouping elements.
  - Its primary purpose is to apply CSS rules to a section of text.

- The ***div*** element is also a grouping element, but it's a block-level element.

In-line and Block-Level Elements

- Example: In-line Element

HTML

```
<div class="block-example"> This is my <strong>div</strong> tag. </div>

<p class="block-example"> This is my <span>paragraph</span> tag. </p>
```

CSS

```
.block-example {
  background-color: green;
  padding: 20px;
}

strong {
  background-color: white;
  padding-left: 20px;
  padding-right: 20px;
}

span {
   background-color: white;
   padding-left: 20px;
   padding-right: 20px;
}
```

This is my **div** tag.

This is my paragraph tag.

Backgrounds

- CSS can set a background color or add background images to HTML5 elements.
- The ***background-image*** property specifies the image URL.
- You can also set the ***background-color*** property in case the image is not found.
- The ***background-position*** property places the image on the page
- The keywords ***top, bottom, center, left*** and ***right*** are used for vertical and horizontal positioning.
- You can position an image by specifying the horizontal and vertical length.
  - *background-position: 50% 30px;*

Backgrounds

- The ***background-repeat*** property places multiple copies of the image next to each other to fill the background.
  - Here, we can set ***no-repeat*** to display only one copy of the background image.
- ***background-attachment: fixed*** fixes the image in the position specified by background-position.
  - Scrolling the browser window will not move the image from its position.

- The default value, ***scroll***, moves the image as the user scrolls through the document.

Backgrounds

- The ***text-indent*** property is used to indent the first line of text in the element by a specified amount.

- The ***font-style*** property allows you to set text to ***none***, ***italic*** or ***oblique*** - oblique is more slanted than italic.
  - the browser will default to italic if the system or font does not support oblique text

Backgounds

```
<style type = "text/css">
        body
       {
            background-image: url(logo.png);
           background-position: bottom right;
           background-repeat: no-repeat;
           background-attachment: fixed;
          background-color: lightgrey;
       }
</style>
```

Element Dimensions

- CSS can specify the actual dimensions of each page element.

<p style = "width: 20%">

<p style = "width: 80%; text-align: center">

<p style = "width: 20%; height: 150px; overflow: scroll">

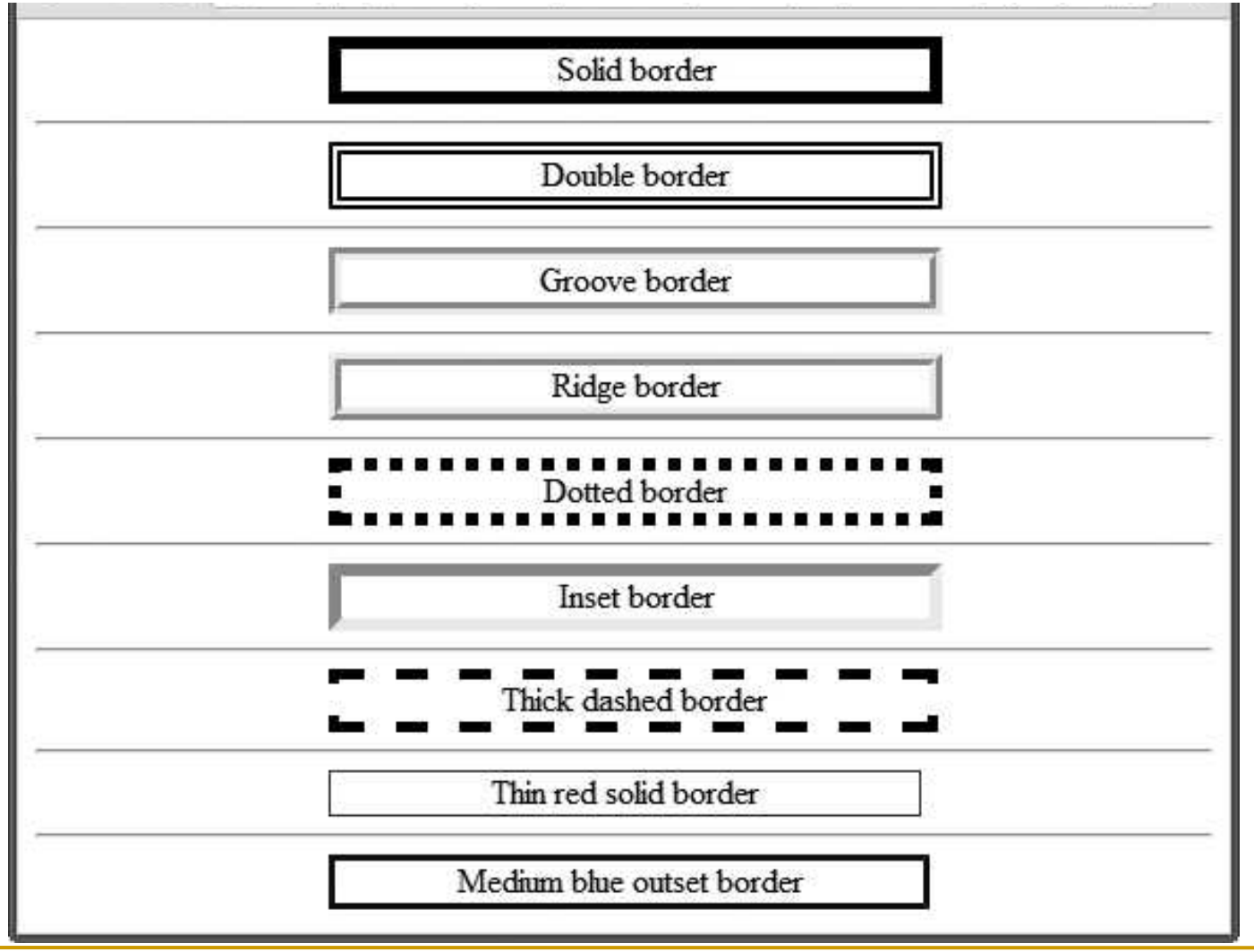
Box Model

- All block-level elements have a virtual box drawn around them.
- CSS controls the border using three properties: ***border-width***, ***border-color*** and ***border-style***.

```
.thin { border-width: thin; }
.solid { border-style: solid; }
.double { border-style: double; }
.groove { border-style: groove; }
.ridge { border-style: ridge; }
.dotted { border-style: dotted; }
.inset { border-style: inset; }
.dashed { border-style: dashed; }
.red { border-color: red; }
```

```
<div class = "double">Double border</div>
```

Box Model


Solid border
Double border
Groove border
Ridge border
Dotted border
Inset border
Thick dashed border
Thin red solid border
Medium blue outset border

Media Types & 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.

- Allows you to decide what a page should look like, depending on the kind of media being used to display the page.

Media Types & Media Queries

- CSS3 Introduced Media Queries
- Media queries in CSS3 extended the CSS2 media types idea
- 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

Media Types

- Example

```
@media all
{
    body { background-color: steelblue; }
    p { font-size: 12pt; color: white; font-family: arial, sans-serif; }
}
@media print
{
    body { background-color: white; }
    p { font-size: 14pt; color: steelblue; font-family: times, serif; }
}
```

Media Queries

- Example

```
@media screen and (min-width: 480px)      //you can give an expression
{
    body
    {
        background-color: lightgreen;
    }
}
```

Drop-Down Menus

- To provide navigation links without using a lot of screen space.
- For example refer *dropdownmenu.html*

Introduction to JavaScript

Introduction to Scripting

- Two types of web pages:
  - **Static** - Content remain the same when you view the URL (unless they were edited by the page creator)
  - **Dynamic** - content is dependent on some conditions or user interaction.
    - Example: Google's search results are dynamic.

Introduction to Scripting

- Three possible scenarios of developing web applications are:
  - Making only the client-side dynamic
    - Using JavaScript, VBScript, Flash, etc.
  - Making only the server-side dynamic
    - Using PHP, ASP, JSP, CGI, etc.
  - Making both the server and client side dynamic
    - Uses both

JavaScript

- It is a client-side scripting language for web-based applications developed by Netscape.
- Highly portable and makes web pages more dynamic and interactive.
- All major web browsers contain JavaScript interpreters, which process the commands written in JavaScript.

JavaScript

- Example:

```
<html>
    <head>
        <meta charset = "utf-8">
        <title>A First Program in JavaScript</title>
        <script type = "text/javascript">

           document.writeln(
              "<h1>Welcome to JavaScript Programming!</h1>" );

        </script>
    </head><body></body>
</html>
```

JavaScript

- The example uses the browser's document object.
- The browser creates a set of objects that allow you to access and manipulate every element of an HTML5 document.
  - ie: Document Object Model (DOM)
- JavaScript code is typically placed in a separate file, then included in the HTML5 document.

JavaScript

- Example: *Displaying a Line of Colored Text*

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing a Line with Multiple Statements</title>
      <script type = "text/javascript">
         <!--
         document.write( "<h1 style = 'color: magenta'>" );
         document.write( "Welcome to JavaScript " +
            "Programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

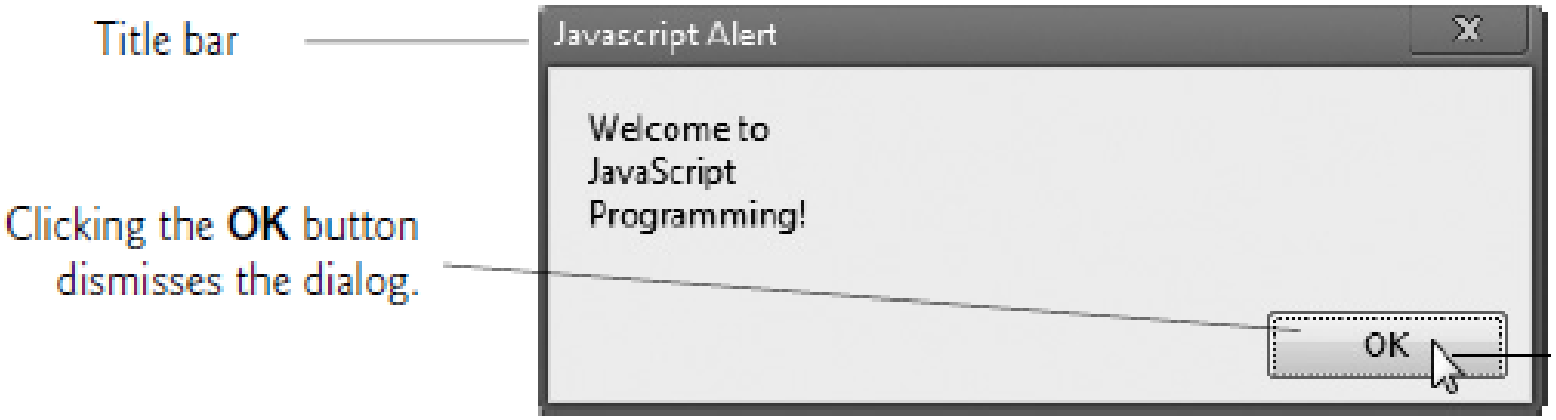
JavaScript

- Example: *Displaying Text in an Alert Dialog*

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing Multiple Lines in a Dialog Box</title>
      <script type = "text/javascript">
         <!--
         window.alert( "Welcome to\nJavaScript\nProgramming!" );
         // -->
      </script>
   </head>
   <body>
      <p>Click Refresh (or Reload) to run this script again.</p>
   </body>
</html>
```

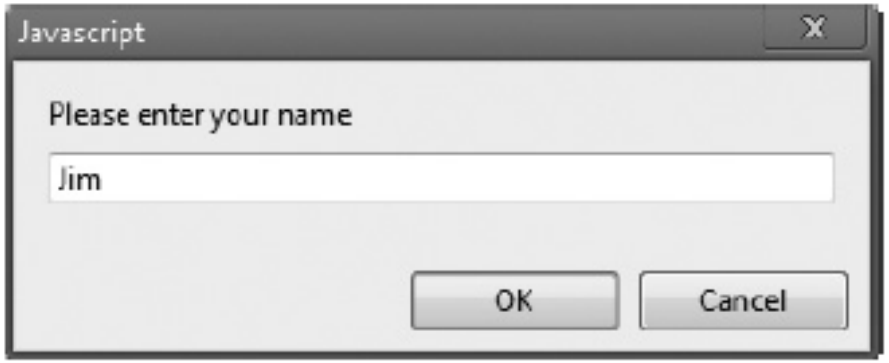
JavaScript

- Example: *Displaying Text in an Alert Dialog*



JavaScript

- Example: *Obtaining user input with prompt dialog*



JavaScript

- Example: *Obtaining user input with prompt dialog*

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>Using Prompt and Alert Boxes</title>
      <script type = "text/javascript">
         <!--
         var name; // string entered by the user

         // read the name from the prompt box as a string
         name = window.prompt( "Please enter your name" );

         document.writeln( "<h1>Hello " + name +
            ", welcome to JavaScript programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

JavaScript

- Example: *Adding Integers*

.....

.....

```
// read in first number from user as a string
firstNumber = window.prompt( "Enter first integer" );

// read in second number from user as a string
secondNumber = window.prompt( "Enter second integer" );

// convert numbers from strings to integers
number1 = parseInt( firstNumber );
number2 = parseInt( secondNumber );

sum = number1 + number2; // add the numbers
```

.....

Questions

- Write a script to evaluate $(a + b)^2$.
- Write a script to find the area of a given circle.

JavaScript Keywords

JavaScript reserved keywords				
break	case	catch	continue	default
delete	do	else	false	finally
for	function	if	in	instanceof
new	null	return	switch	this
throw	true	try	typeof	var
void	while	with		
Keywords that are reserved but not used by JavaScript				
class	const	enum	export	extends
implements	import	interface	let	package
private	protected	public	static	super
yield				

Functions

- Two types:
  - Programmer-defined functions
  - Prepackaged functions (Methods)

- JavaScript provides several objects that have a rich collection of methods for performing common mathematical calculations, string manipulations, date and time manipulations, and manipulations of collections of data called arrays.
  - **Eg:** Math.pow(), Math.sqrt(), dt.getDate(), dt.getHours(), msg.toUpperCase(), txt.length, etc.

Functions

- Programmer-defined functions are defined using the keyword **`function`**

- Format:

```
function function-name(parameter-list)
{
    declarations and statements
}
```

Functions

- Functions can be called by:
  - Specifying the function name

    `fun_name()` or `window.fun_name()`

  - Using the window's **`addEventListener`** method

    ```
    window.addEventListener("load", fun_name);
    ```

  - Using the "**`onX`**" methods on HTML elements like buttons, textfields, lists, etc.
    - **`onClick`**, **`ondblclick`**, **`onmouseover`**, **`onfocus`**, etc..

Functions

```
<script>
  var a, b, sum;
  function add()
  {
     sum = a + b;
     document.writeln("Sum is "+sum);
  }
  a = parseInt(prompt("Enter first number"));
  b = parseInt(prompt("Enter second number"));
  window.addEventListener("load", add);
</script>
```

Functions

```
<script>
  var r, ar;
  function area(r)
  {
     ar = 3.14 * r;
     return ar;
  }
  r = parseFloat(prompt("Enter the radius"));
  document.writeln("Area is "+window.area(r));
</script>
```

Accessing HTML Elements in JavaScript

- By id

  var myElement = document.getElementById(“element_id");

- By tag name

  var y = document.getElementsByTagName("p");

- By Class

  document.getElementsByClassName(“class_name");

- By Object Collections

  var x = document.forms["form_name"];

  x.elements[0].value ;

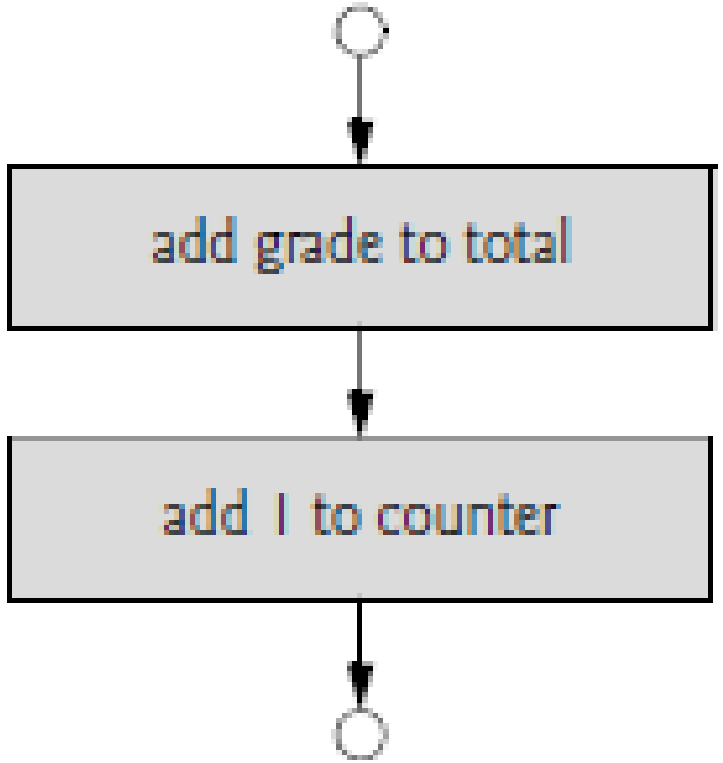
Refer examples…..

Control Statements

- Normally, statements in a script execute sequentially.
- Various JavaScript statements enable you to specify the next statement to execute may not be the next one in sequence.

- All programs could be written in three control structures:
  - sequence structure
  - selection structure - ***if, if-else, switch, conditional operator***
  - repetition structure - ***while, do…while, for*** and ***for…in***

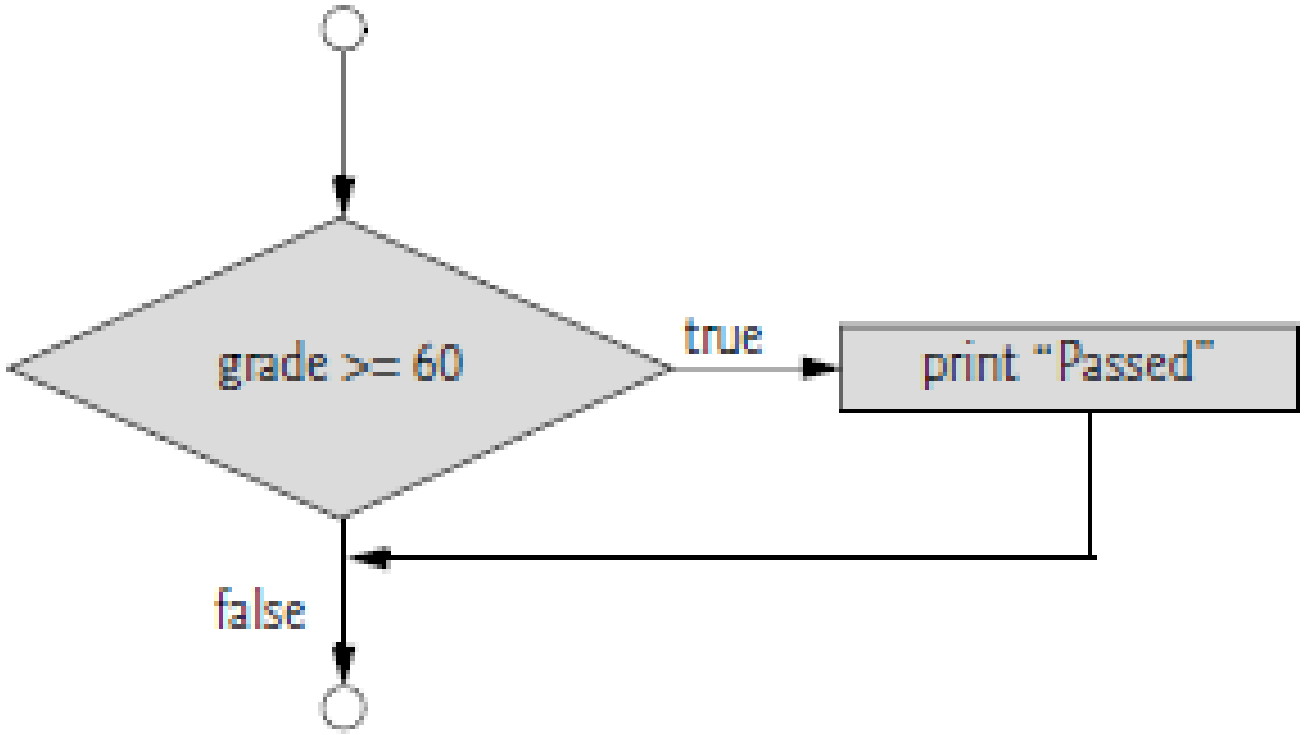
Control Statements




```
total = total + grade;
counter = counter + 1;
```

Sequence Structure

if Selection Statement



Question

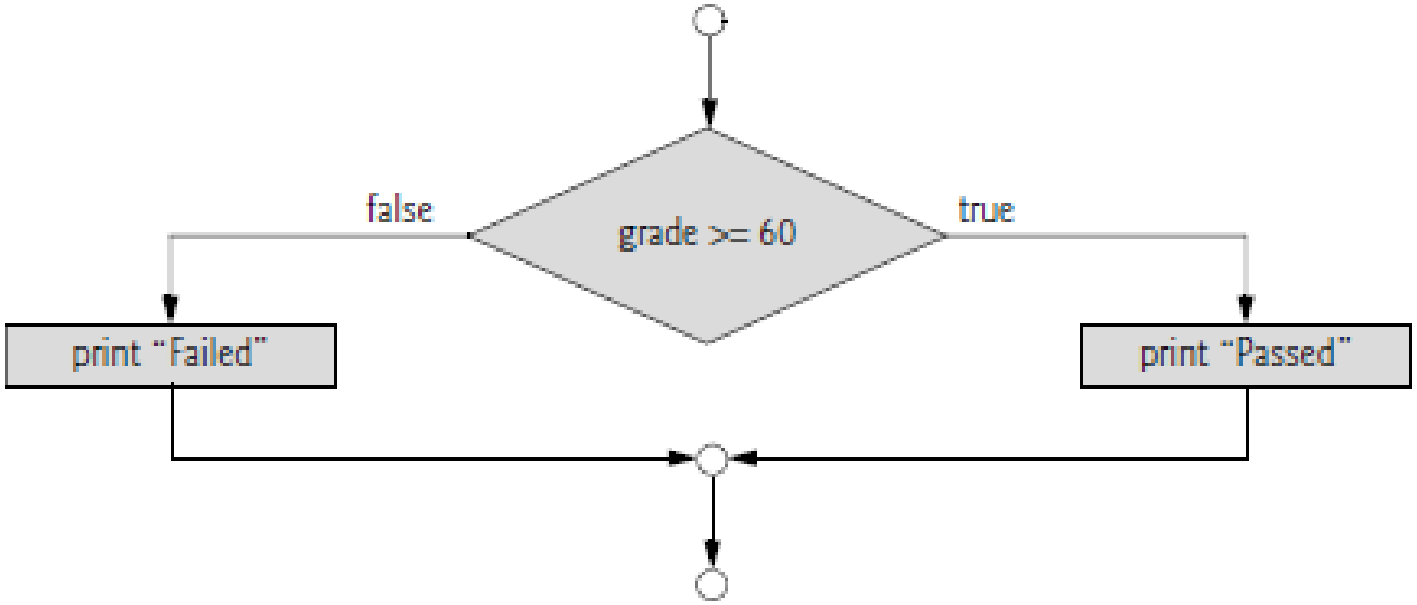
- Write a script to print a greeting message based on the system time.

- Hint:

```
var now = new Date(); // current date and time
var hour = now.getHours(); //current hour(0-23)
```

- Example: Refer ***ifselection.html***

if-else Selection Statement




```
if ( studentGrade >= 60 )
   document.writeln( "<p>Passed</p>" );
else
   document.writeln( "<p>Failed</p>" );
```

Question

- Write a script to find the largest of three numbers.

Conditional Operator (*?* *:*)

- Closely related to the ***if…else*** statement.
- Otherwise called ternary operator
  - It takes three operands.

- Example:

```
document.writeln( studentGrade >= 60 ? "Passed" : "Failed" );
```

Nested *if…else*

```
if ( grade >= 90 )
  document.writeln( "A" );
else if ( grade >= 80 )
  document.writeln( "B" );
else if ( grade >= 70 )
  document.writeln( "C" );
else if ( grade >= 60 )
  document.writeln( "D" );
else
  document.writeln( "F" );
```

Repetition Statement *while*

```
var i = 1;
var sum = 0;
while ( i <= 1000 )
{
   sum = sum + i;
   i = i + 1;
}
```

For example, refer `prime_while.html`

Repetition Statement *for*

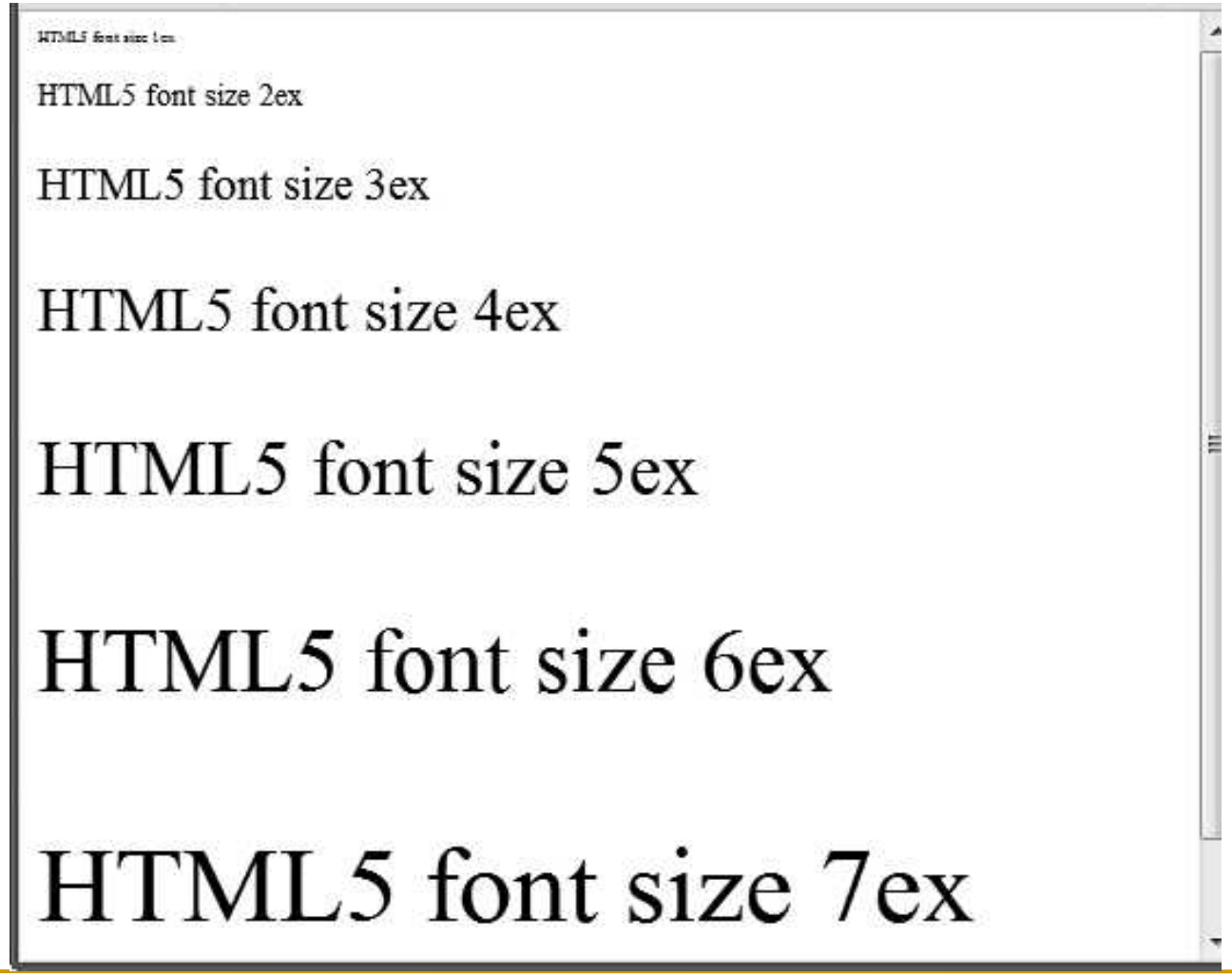
```
for (var i = 1; i < 10; i++)
{
   document.writeln("Hi");
}
```

For example, refer `sumofn_for.html`

Repetition Statement *for*

```
for (var i = 1; i <= 7; i++)
{
   document.writeln("<p   style   =   'font-size:   “+   i
  +"ex'>HTML5 font size:"+ i +"ex </p>");
}
```

Repetition Statement *for*



Repetition Statement *for*

Programming Exercise:

A person invests $1000.00 in a savings account yielding 5 percent interest. Calculate and print the amount of money in the account at the end of each year for 10 years.

Use the following formula to determine the amounts:

$$a = p\,(1 + r)^{n}$$

p = Invested amount (Principal)
r = Annual interest rate
n = Number of years
a = Amount on deposit at the end of the nth year.

Calculating Compound Interest

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

switch Multiple-Selection Statement

```
switch(choice)
{
  case “1”:
      ...
     break;
  case “2”:
      ...
     break;
  default:
      ...
     break;
}
```

For example, refer: *switch_case.html*

do…while Repetition Statement

```
do
{
   .....
   .....
}while(condition);
```

break and *continue* Statement

- The ***break*** causes immediate exit from the loop statements.
  - Execution continues with the first statement after the structure.

```
<script>
   for ( var count = 1; count <= 10; ++count )
   {     if ( count == 5 )
               break;
         document.writeln( count + " " );
   } //end for
   document.writeln("Broke out of loop at count = " + count);
</script>
```

break and *continue* Statement

- The ***continue*** statement skips the remaining statements in the body of the statement and proceeds with the next iteration of the loop.

```
<script>
    for ( var count = 1; count <= 10; ++count )
    {
        if ( count == 5 )
            continue;
        document.writeln( count + " " );
    } //end for
    document.writeln( "<p>Used continue to skip printing 5</p>" );
</script>
```

Functions - Question

- Write down all the methods for performing
  - mathematical calculations,
  - string manipulations,
  - date and time manipulations,
  - manipulations of array data.

- Write a JavaScript to print all the prime numbers between 10 and 100 using a function.

Random Number Generation

- `var rv = Math.random();`
- Method `random` generates a floating-point value from 0.0 up to, but not including, 1.0.
- For example, refer
  - `randomnumbers.html`
  - `rolldice.html`

Scope Rules

- The scope of a variable or function is the portion of the program in which the identifier can be referenced.
- Global variables or script-level variables that are declared in the head element are accessible in any part of a script and are said to have **global scope**.

- Identifiers declared inside a function have **function or local scope** and can be used only in that function.

JavaScript Global Functions

- JavaScript provides some standard global functions.
  - **isNaN:**
    - Takes a numeric argument and returns true if the value of the argument is not a number.
  - **isFinite:**
    - Takes a numeric argument and returns true if the value of the argument is not NaN, Number.POSITIVE_INFINITY or Number.NEGATIVE_INFINITY

JavaScript Global Functions

- **parseInt:**
  - Takes a string argument and attempts to convert the beginning of the string into an integer value. If the conversion is unsuccessful, the function returns NaN; otherwise, it returns the converted value (for example, parseInt("abc123") returns NaN, and parseInt("123abc") returns the integer value 123).

- **parseFloat**
  - Takes a string argument and attempts to convert the beginning of the string into a floating-point value.

Recursion

- A recursive function is a function that calls itself.

```
function factorial( number )
{
  if ( number <= 1 )
      return 1;
  else
      return number * factorial(number - 1);
}
```

Recursion

- One negative aspect of recursion is that function calls require a certain amount of time and memory space. This is known as **function-call overhead**.
- Because recursion uses repeated function calls, this overhead greatly affects the performance of the operation.

- Question:
  - Write a JavaScript to print the factorial of all the numbers from 0 to 10 using recursion.
  - To find the power of a number, ie, x^y

Arrays

- Arrays are data structures consisting of related data items.
- JavaScript arrays are “dynamic” entities in that they can change size after they’re created.
- To refer to a particular location or element in the array, we specify the name of the array and the position number of the element in the array.

Declaring and Allocating Arrays

- An array in JavaScript is an Array object.
- Use the **new** operator to create an array.

```
var c = new Array( 12 );
```

- **Array Initialization**

```
var n = [10, 20, 30, 40, 50];
var n = new Array(10, 20, 30, 40, 50);
var n = [ 10, 20, , 40, 50 ]; //creates a five-element
  array in which the third element has the value undefined.
```

- For example, refer array initialization

Arrays - Questions

- Write a JavaScript to find the average of 10 elements using an array.
- Write a JavaScript to sort the elements of an array.

Passing Arrays to Functions

- To pass an array argument to a function, specify the array's name (a reference to the array).
- For example,

```
var arr = new Array( 24 );
fun( arr );
```

- For a function to receive an array

```
function fun( b )
{
}
```

Searching Arrays with Array Method `indexOf`

- The Array object has built-in methods **`indexOf`** and **`lastIndexOf`** for searching arrays.
- **`indexOf`** searches for the first occurrence of the specified key value, and method **`lastIndexOf`** searches for the last occurrence.
- If the key value is found in the array, it returns the index of that value; otherwise, -1 is returned.
- Refer example: **`array searching`**

Searching Arrays with Array Method `indexOf`

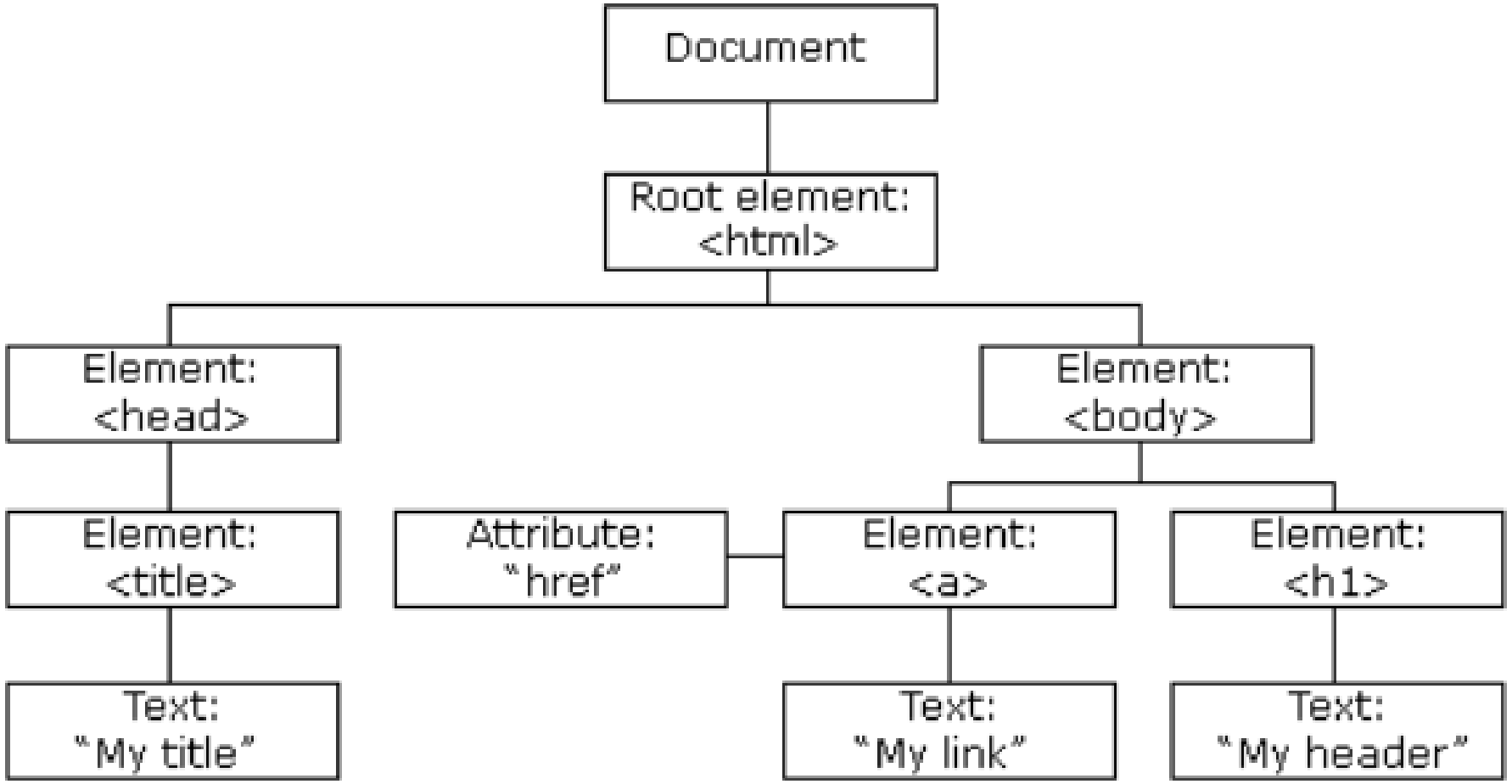
- You can pass an optional second argument to these methods that represents the index from which to start the search.
- By default, this argument's value is 0
- If the argument is greater than or equal to the array's length, the method simply return -1.
- If the argument's value is negative, it's used as an offset from the end of the array.
- For example, the 100-element array has indices from 0 to 99. If we pass -10 as the second argument, the search will begin from index 89.

Document Object Model (DOM)

- The DOM is a W3C standard.
- The DOM defines a standard for accessing documents

- 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:

Document Object Model (DOM)



Document Object Model (DOM)

- With the object model, JavaScript can change:
  - all the HTML elements in the page
  - all the HTML attributes in the page
  - all the CSS styles in the page
  - It can remove existing HTML elements and attributes
  - It can add new HTML elements and attributes
  - It can react to all existing HTML events in the page
  - JavaScript can create new HTML events in the page

Document Object Model (DOM)

- Example:

```
<html>
    <body>
        <p id="demo"></p>

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

    </body>
</html>
```

Finding HTML Elements using DOM

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements using DOM

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.style.*property* = *new style*	Change the style of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding and Deleting Elements using DOM

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new*, *old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers using DOM

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

JavaScript Objects

- In JavaScript, almost "everything" is an object:
  - Boolean, numbers, string, Dates, Maths, Regular Expressions, arrays.

- JavaScript object values are written as name : value pairs (name and value separated by a colon).

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

JavaScript Objects

- A JavaScript object is a collection of named values.
- The named values are called properties.
- Object properties can be both primitive values, other objects, and functions.
- Objects written as name value pairs are similar to:
  - Associative arrays in PHP
  - Dictionaries in Python
  - Hash tables in C

Object Methods

- Methods are actions that can be performed on objects.
- An object method is an object property containing a function definition.

- **Note: JavaScript objects are containers for named values, called properties and methods.**

Creating Objects

- With JavaScript, you can define and create your own objects.
- There are different ways to create new objects:
  - Using an object literal.
  - Using the keyword `new`.
  - Define an object constructor, and then create objects of the constructed type.
  - Create an object using Object.create().

1. Using an Object Literal

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs

- Example:

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

- Note: It is a common practice to declare objects with the **const** keyword.

Using an Object Literal

- Or:

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

2. Using the Keyword `new`

- Example:

```
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

3. Using Object Constructors

- Example:

```
<script>
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
// Create a Person object
const myFather = new Person("John", "Doe", 50, "blue");
```

4. Using Object.create()

- Example:

```
const person = {
  isHuman: false,
  printIntroduction: function() {
      return this.isHuman;
  }
};
const me = Object.create(person);
me.name = 'Mathew'; // "name" is a property set on "me", but
  not on "person"
me.isHuman = true;
```

JavaScript Objects are Mutable

- Objects are mutable: They are addressed by reference.

```
const person = new Object();

const x = person; //Will not create a copy of person.
```

- The object **x** is not a copy of person. It is person. Both **x** and **person** are the same object.
- Any changes to **x** will also change **person**, because **x** and **person** are the same object.

JavaScript Objects are Mutable

```
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50,
  eyeColor:"blue“
}
const x = person;
x.age = 10; // Will change both x.age and person.age
```

JavaScript Objects Properties

- A JavaScript object is a collection of unordered properties.
- Properties are the values associated with a JavaScript object.
- Properties can usually be changed, added, and deleted, but some are read only.
- The syntax for accessing the property of an object is:

```
objectName.property       // person.age
```

or

```
objectName["property"]    // person["age"]
```

or

```
objectName[expression]    //x="age";    person[x]
```

JavaScript Objects Properties

- **Adding Properties**: You can add new properties to an existing object by simply giving it a value.

```
person.nationality = "Indian";
```

- **Deleting Properties**: The `delete` keyword deletes a property from an object.

```
delete person.age;
```

JavaScript Objects Methods

- Objects can also have methods.
- A method is a function stored as a property.

```
const person = {
      fName : "John",
      lName : "Doe",
      id : 5566,
      fullName : function() {
         return this.fName+" "+this.lName;
      }
};
```

Accessing Objects Methods

- You access an object method with the following syntax:

```
objectName.methodName().
```

- Example:

```
name = person.fullName();
```

Nested Objects

- Values in an object can be another object.

```
myObj = {
    name:"John",
    age:30,
    cars: {
        car1:"Ford",
        car2:"BMW",
    }
}
```

To access car2 value, use:

myObj.cars.car2; or

myObj.cars["car2"]; or

myObj["cars"]["car2"];

JavaScript `for...in` Loop

- The JavaScript for...in statement loops through the properties of an object.
- Syntax:

```
for (let variable in object) {
   // code to be executed
}
```

- The block of code inside of the `for...in` loop will be executed once for each property.

JavaScript `for...in` Loop

- Example:

```
const person = {
  fname:" John",
  lname:" Doe",
  age: 25
};
for (let x in person) {
  txt += person[x];
}
```