ST. JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CST463 - WEB PROGRAMMING

September 5, 2024

CST463 - WEB PROGRAMMING

PROF.SMITHA JACOB

St . Joseph's College of Engineering and Technology, Palai

Module 2 (CO2, Cognitive Knowledge Level: Apply)

- (CSS, JavaScript) Introduction to Stylesheets : Introduction to CSS-Basic syntax and structure-Inline Styles, Embedded Style Sheets, Conflict Resolution, Linking External Style Sheets-Exploring CSS Selectors-Properties, values, Positioning Elements: Absolute Positioning, Relative Positioning - Backgrounds-List Styles-Element Dimensions- Table Layouts-Box Model and Text Flow-div and span -Basics of Responsive CSS, Media port & Media Queries.
- Introduction to JavaScript : Introduction to Scripting- Programming fundamentals of JavaScript -Obtaining User Input with prompt Dialogs-Arithmetic-Decision Making -Control Statements - Functions -Arrays - Objects -Document Object Model (DOM) -Form processing

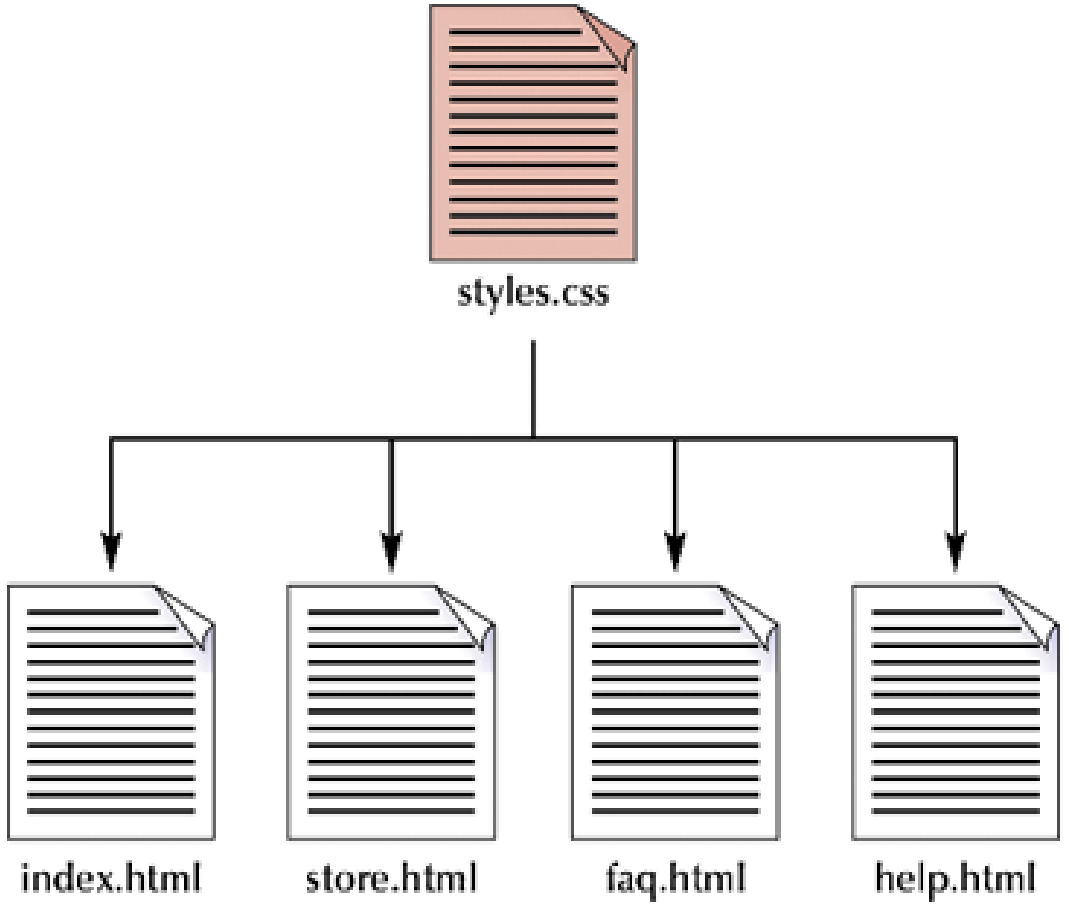
Introduction to Stylesheets

Introduction to CSS3

- Cascading Style Sheets 3 (CSS3) that allows you to specify the presentation of elements on a web page (e.g., fonts, spacing, sizes, colors, positioning) separately from the document's structure and content (section headers, body text, links, etc.).
- **CSS is a set of rules for displaying markup content**.
- CSS made the Separation of structure from presentation
  - multiple style sheets can be applied to the same Web page
  - Same style sheet can be applied to the multiple Web page
- Cascading:
  - Display rules "cascade" down
  - The most specific rule is used
- Styles Sheet: consists of Rules which are created as styles

Advantages of CSS3

- Saves time
- Easy to change
- Keep consistency(always behave in the same way)
- more control over layout
- Use styles with JavaScript => DHTML
- create a common format for all the Web pages



Types of CSS

- Inline style sheet
  - The style properties are added within the HTML tag itself
- Embedded style sheet
  - The set of style properties are embedded within the HTML document
- External style sheet
  - Common set of style properties are applied to all the web pages from an external .css file.

Each definition contains:

- A property
- A colon
- A value
- A semicolon to separate two or more values
- Can include one or more values

Selector{property1:value1; property2:value2;…}

```
h1 {font-size:12pt; color:red}
```

Order of Precedence of Style Sheet

- Inline style sheets(lowest level)
- Embedded/Document Level sheets
- External style sheets(highest Level)
- Inline Style sheets have precedence over document style sheets which have precedence over external style sheet.
- When using multiple styles that conflict, which will be displayed?
- Order:
  - Inline style sheet
  - Embedded style sheet
  - External style sheet
  - Browser default

Prof.Smitha Jacob, Department of Computer Science and Engineering, SJCET Palai

Style Specification Format

- Inline Style

  Style="property0:value0;property1:value1;

  ……..

  propertyZ:valueZ;"

- **Document Level**

  <style type="text/css">

  rule_list

  </style>

  - Each style rule in a rule list has two parts
  - Selector—the tag or tags affected by the rule.
  - List of property/value pairs
  - Selector { property_1: value_1; property_2: value_2:… property_n: value_n;}

Style Specification Format

- External Style sheet

  <link href=URL rel="relation_type" type="link_type">
- URL is the file.css
  - Relation_type="stylesheet"
  - Link_type="text/css"
- Do not include <style> tags
- Save the document as filename.css

Inline styles

- Add styles to each tag within the HTML file
- Use it when you need to format just a single section in a web page
- style information is directly attached to the HTML elements they affect
- Attribute style,followed by a colon and a value
- <h1 style="color:red; font-family: sans-sarif">India is my country</h1>

```
<!DOCTYPE html>
<html>
  <head>    <title> sjc</title>
    <meta charset = "utf-8" /></head>
  <body>
<p>Hello world from the web Master</p>
<p style="font-size:50; font-family: Arial;font-weight:bold;color:green" >
        Welcome to HTML CSS Effects
</p>
</body>
</html>
```

Inline styles

```
<!DOCTYPE html>
<!-- Using inline styles -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Inline Styles</title>
   </head>
   <body>
      <p>This text does not have any style applied to it.</p>
      <p style = "font-size: 20pt;">This text has the
         <em>font-size</em> style applied to it, making it 20pt.
      </p>

      <p style = "font-size: 20pt; color: deepskyblue;">
         This text has the <em>font-size</em> and
         <em>color</em> styles applied to it, making it
         20pt and deep sky blue.</p>
   </body>
</html>
```

Inline styles

- Attribute style specifies an element's style. Each CSS property (font-size in this case) is followed by a colon and a value. the two properties, font-size and color are separated by a semicolon.
- set the given paragraph's color to deepskyblue. Hexadecimal codes may be used in place of color names.
- Figure contains the HTML standard color set.
- A complete list of HTML standard and extended colors at www.w3.org/TR/ css3-color/

Color name	Value	Color name	Value
aqua	#00FFFF	navy	#000080
black	#000000	olive	#808000
blue	#0000FF	purple	#800080
fuchsia	#FF00FF	red	#FF0000
gray	#808080	silver	#C0C0C0
green	#008000	teal	#008080
lime	#00FF00	yellow	#FFFF00
maroon	#800000	white	#FFFFFF

HTML standard colors and hexadecimal RGB values.

Embedded / internal/Document level

- A style is applied to the entire HTML file.Use it when you need to modify all instances of particular element (e.g., h1) in a web page
- Example

```
<style type="text/css">
        h1 {color:red; font-size:20;font-family:monospace}
</style>
<head>
<title>Embedded Example</title>
<style type="text/css">
        Style declarations
</style>
</head>
```

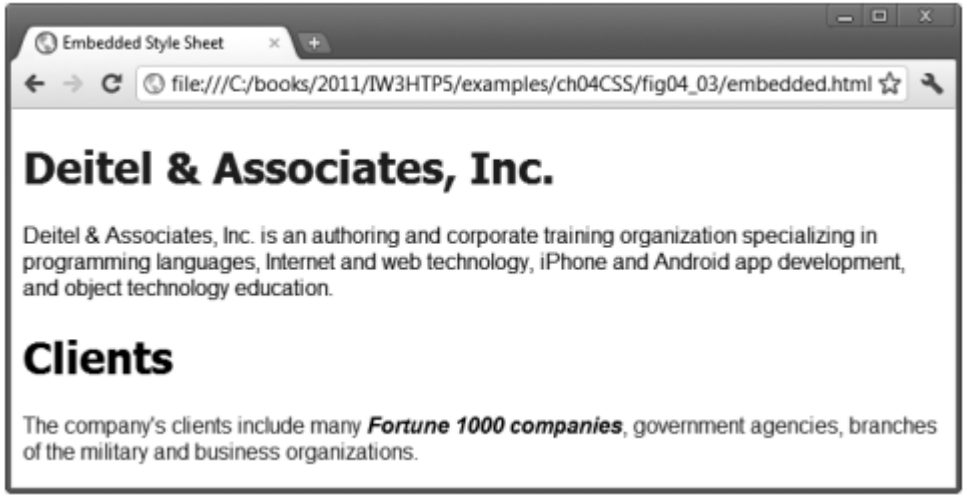
A style declaration:

Selector {attribute1:value1; attribute2:value2; ...}

Selector = an element in a document (e.g., a header or paragraph)

Embedded / internal/Document level

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Embedded Style Sheet</title>

      <!-- this begins the style sheet section -->
      <style type = "text/css">
         em       { font-weight: bold;
                    color: black; }
         h1       { font-family: tahoma, helvetica, sans-serif; }
         p        { font-size: 12pt;
                    font-family: arial, sans-serif; }
         .special { color: purple; }
      </style>
   </head>
   <body>
      <!-- this attribute applies the .special style class -->
      <h1 class = "special">Deitel & Associates, Inc.</h1>

      <p>Deitel & Associates, Inc. is an authoring and
         corporate training organization specializing in
         programming languages, Internet and web technology,
         iPhone and Android app development, and object
         technology education.</p>

         <h1>Clients</h1>
         <p class = "special"> The company's clients include many
            <em>Fortune 1000 companies</em>, government agencies,
            branches of the military and business organizations.</p>
      </body>
   </html>
```



Embedded / internal/Document level

- Styles placed in the head apply to matching elements wherever they appear in the body.
- The style element's type attribute specifies the MIME (Multipurpose Internet Mail Extensions) type that describes the style element's content.
- CSS documents use the MIME type text/css

MIME type	Description
text/css	CSS documents
image/png	PNG images
text/javascript	JavaScript markup
text/plain	Plain text
image/jpeg	JPEG image
text/html	HTML markup

A few common MIME types.

```
<!DOCTYPE html>
<html >
<head>
<title>Getting Started</title>
<style type="text/css">
   h1 {font-family: sans-serif; color: organge}
   p{font-size:15;font-style:italic;background-color:grey}
</style>
</head><body><h1>Hello world using CSS</h1>
<p> Embedded Style is really interesting<p>
</body>
</html>
```

External style sheets

- Style sheets are a convenient way to create a document with a uniform theme.
- With external style sheets (i.e., separate documents that contain only CSS rules), you can provide a uniform look and feel to an entire website (or to a portion of one).
- You can also reuse the same external style sheet across multiple websites.
- Different pages on a site can all use the same style sheet. When changes to the styles are required, you need to modify only a single CSS file to make style changes across all the pages that use those styles. This concept is sometimes known as **skinning**.
- While embedded style sheets separate content from presentation, both are still contained in a single file, preventing a web designer and a content author from conveniently working in parallel.
- External style sheets solve this problem by separating the content and style into separate files

External style sheets

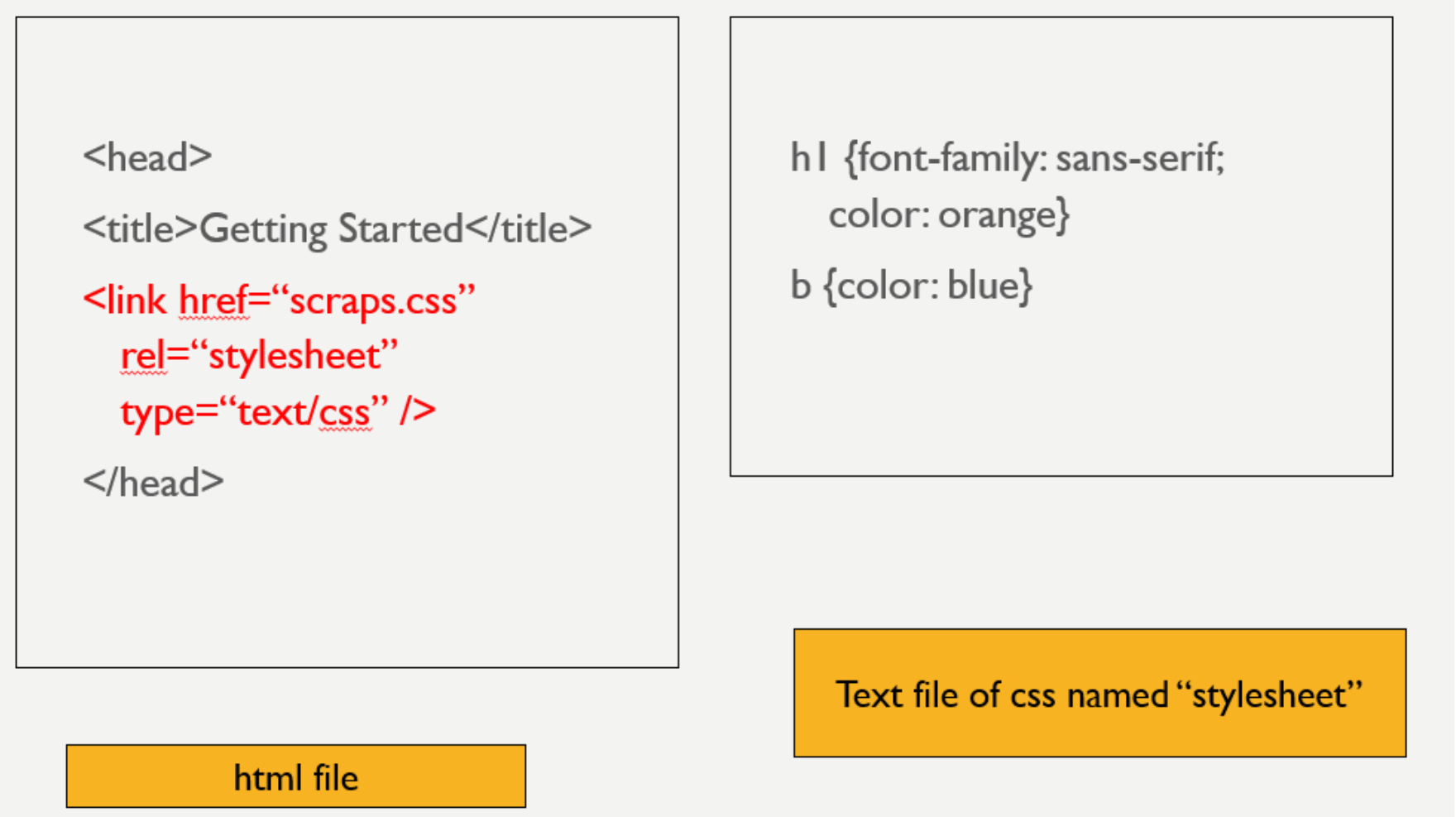
- An external style sheet is a .css file containing the style definition (declaration)
- Use it when you need to control the style for an entire web site
- Example
- h1, h2, h3, h4, h5, h6 {color : red; font-family : sans-serif}
- Save this in a new document using a .css extension

- Creating an External Style Sheet
  - Open a new blank document in Notepad
  - Type style declarations

    h1 {color:red; font-family:sans-serif;}
  - Do not include <style> tags
  - Save the document as filename.css
- Linking to Style Sheets
  - Open an HTML file
  - Between <head> and </head> add

    <link href=URL rel="relation_type" type="link_type">
  - URL is the file.css
  - Relation_type="stylesheet" and Link_type="text/css"
  - Save this file and the .css file in the same web server directory

Linking External style sheets

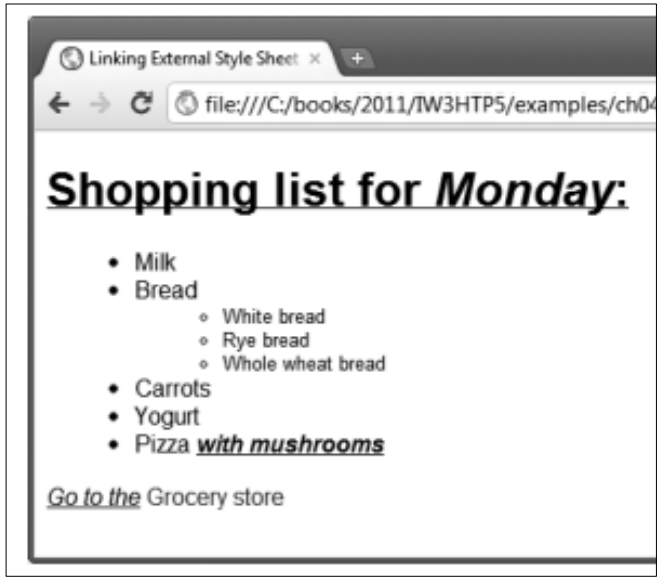


Linking External style sheets

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Linking External Style Sheets</title>
      <link rel = "stylesheet" type = "text/css"
         href = "styles.css">
   </head>
   <body>
      <h1>Shopping list for <em>Monday</em>:</h1>
      <ul>
         <li>Milk</li>
         <li>Bread
            <ul>
               <li>white bread</li>
               <li>Rye bread</li>
               <li>Whole wheat bread</li>
            </ul>
         </li>
         <li>Carrots</li>
         <li>Yogurt</li>
         <li>Pizza <em>with mushrooms</em></li>
      </ul>

      <p><em>Go to the</em>
         <a class = "nodec" href = "http://www.deitel.com">
            Grocery store</a>
      </p>
   </body>
</html>
```

```
body      { font-family: arial, helvetica, sans-serif; }
a.nodec   { text-decoration: none; }
a:hover   { text-decoration: underline; }
li em     { font-weight: bold; }
h1, em    { text-decoration: underline; }
ul        { margin-left: 20px; }
ul ul     { font-size: .8em; }
```

External style sheet.



Conflict Resolution

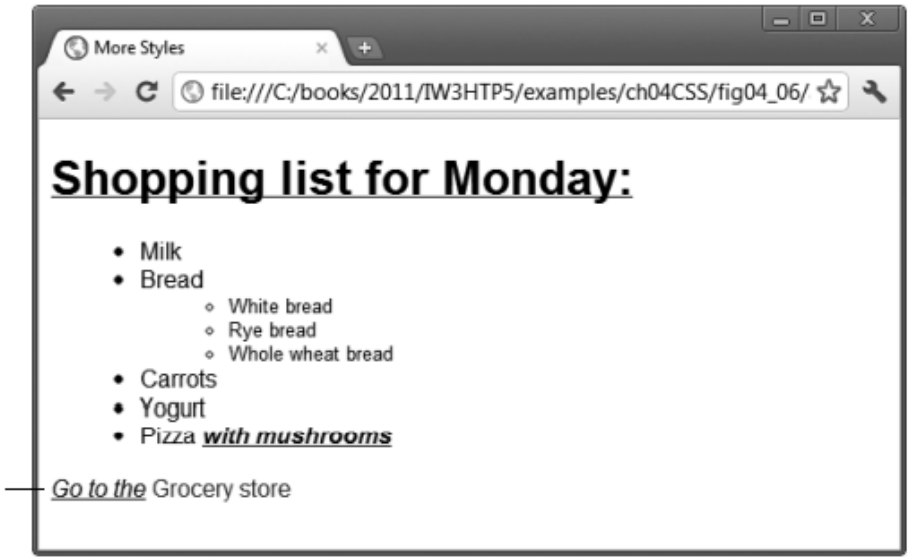
Conflicting styles

- Styles may be defined by a user, an author or a user agent.
  - User →User is a person viewing your web page,
  - Author → Author is the person who writes the document and the
  - User Agent →It is the program used to render and display the document (e.g., a web browser).
- Styles cascade (and hence the term “Cascading Style Sheets”), or flow together, such that the ultimate appearance of elements on a page results from combining styles defined in several ways.
- ***Styles defined by the user take precedence over styles defined by the user agent.***
- ***Styles defined by authors take precedence over styles defined by the user.***
- Most styles defined for parent elements are also inherited by child (nested) elements

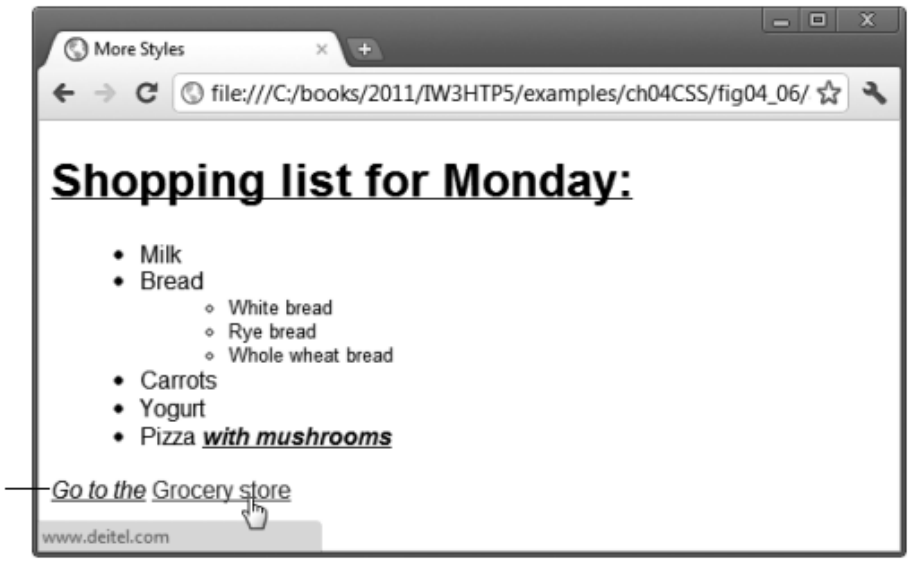
Conflict resolution

- There are certain properties that you don't want to be inherited.
- For example, the background-image property allows you to set an image as the background of an element.
- If the body element is assigned a background image, we don't want the same image to be in the background of every element in the body of our page. Instead, the background-image property of all child elements retains its default value of none.
- There are some rules for resolving conflicts between styles defined for elements and styles inherited from parent and ancestor elements
- Properties defined for child and descendant elements have a higher specificity than properties defined for parent and ancestor elements.
- Conflicts are resolved in favor of properties with a higher specificity, so the child's styles take precedence

```
<!DOCTYPE html>
<html> <head>
        <meta charset = "utf-8">
        <title>More Styles</title>
        <style type = "text/css">
           body     { font-family: arial, helvetica, sans-serif; }
           a.nodec  { text-decoration: none; }
           a:hover  { text-decoration: underline; }
           li em    { font-weight: bold; }
           h1, em   { text-decoration: underline; }
           ul       { margin-left: 20px; }
           ul ul    { font-size: .8em; }
        </style>
     </head>
    <body>
       <h1>Shopping list for Monday:</h1>
       <ul>
          <li>Milk</li>
          <li>Bread
             <ul> <li>white bread</li>
                  <li>Rye bread</li>
                  <li>Whole wheat bread</li> </ul> </li>
          <li>Carrots</li>
          <li>Yogurt</li>
          <li>Pizza <em>with mushrooms</em></li>
       </ul>
       <p><em>Go to the</em>
          <a class = "nodec" href = "http://www.deitel.com">
             Grocery store</a> </p> </body> </html>
```



Grocery store link with no decoration



Grocery store link with underline displayed when user hovers over the link

Style Sheets-Exploring CSS Selectors

Exploring CSS Selectors

- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- There are six different selector forms
  - Simple selector
  - Class selector
  - Generic selector
  - Id selector
  - Universal selector
  - Pseudo classes

Exploring CSS Selectors- Simple selector

- Simple selector form is a list of style rules, and property values in the rule apply to all occurrences of the all of the named element
- The selector is a tag name or a list of tag names, separated by commas
- Ex) h1, h3 { font-size: 24pt ;}
  h2 { font-size: 20pt ;}
- Contextual selectors: Selectors can also specify that the style should apply only to elements in certain positions in the document
- Ex) selector applies its style to the content of italic elements that are descendants of bold elements in the body of the document.
  body b i {font-size: 24pt ;}
- Also called as descendant selectors.

Exploring CSS Selectors- class selector

- Used to allow different occurrences of the same tag to use different style specifications.
- HTML and XHTML require each id be unique– therefore an id value can only be used once in a document.
- You can mark a group of elements with a common identifier using the class attribute.
  `<element class="class"> … </element>`
- A style class has a name, which is attached to the tag's name with a period.

  p.ques{property-value list}

  p.ans {property-value list}

   What is WWW?

   WWW is World wide web,an information repository which can be accessed over internet

Exploring CSS Selectors- class selector - Example

```
<!DOCTYPE html>
<html lang = ″en″> <head> <title> sjc</title> <meta charset = ″utf-8″ />
<style type = "text/css">
p.regtext {font-family: Times; font-size: 14pt; width: 800px}
p.abstext { position: absolute;top: 125px; left: 50px; font-family: Times; font-size: 24pt; font-style: italic; width: 500px}
</style> </head> <body>
<p class = "abstext"> APPLES ARE GOOD FOR YOU </p>
<p class = "regtext"> Apple is the common name for any tree of the genus Malus, of the family Rosaceae. Apple trees grow in any of the temperate areas of the world. Some apple blossoms are white, but most have stripes or tints of rose. Some apple blossoms are bright red. Apples have a firm and fleshy structure that grows from the blossom. The colors of apples range from green to very dark red. The wood of apple trees is fine-grained and hard. </p>
<p class = "abstext"> ORANGES ARE GOOD IN SUMMER </p>
<p class = "regtext"> Oranges the common name for any tree of Orange. trees grow in any of the temperate areas of the world. Some orangaes blossoms are white, but most have stripes or tints of rose. </p>
</body>
 </html>
```

Generic Selector

- A generic class can be defined if you want a **style to apply to more than one kind of tag**.
- A generic class must be named, and the name must begin with a period without a tag
- Example: .really-big { … }

```
<h1 class = "really-big"> SJCET ADMISSION</h1>...
<p class = "really-big"> SJCET BTECH ADMISSION 2022… </p>
<h2 class = "really-big"> SJCET BHRM ADMISSION 2022… </h2>
<h3 class = "really-big"> SJCET PG ADMISSION 2022… </h3>
```

Generic Selector

```
<!DOCTYPE html><html lang = ″en″>
<head> <title> sjc</title>     <meta charset = ″utf-8″ />
<style type = "text/css">
.regtext {font-family: Times; font-size: 14pt; width: 800px}
.admstext { position: absolute;top: 125px; left: 50px; font-family: Times; font-size: 24pt; font-style: italic; width: 500px}
</style>
 </head>
 <body>
      <h1 class = “admstext"> SJCET ADMISSION</h1>.
    <p class = "regtext">SJCET structure that grows from the blossom. Minority catholic institution .provides Btech and MTECh courses</p>
    <p class = “admstext"> SJCET BTECH ADMISSION 2022… </p>
    <p class = "regtext">SJHMCT hotel management institution . Minority catholic institution .provides BHRM courses</p>
    <h2 class = “admstext"> SJCET BHRM ADMISSION 2022… </h2>
    <h3 class = “admstext"> SJCET PG ADMISSION 2022… </h3>
    <p class = "admstext"> PG courses MTECH,MBA,MTECH available at SJCET</p>
</body>  </html>
```

Id Selector

- An id selector allow the application of a style to one specific element.
- Style specified in the id selector applies to the element with the given id.
- To create an ID for a specific tag, use the property:

  <tag id="id_name">
- To apply a style to a specific ID, use:

  #id_name {style attributes and values}
- The general form of an id selector is as follows :

  #specific-id {property-value list}

Example:

#section14 {font-size: 20} specifies a font size of 20 points to the element

<h2 id ="section14"> Alice in wonderland</h2>

Id Selector

```
<!DOCTYPE html>
<html><head>
<style>
#para1 {
  text-align: center;
  color: green;
  font-weight:bold;
}
</style>
</head><body>
<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>
</body>
</html>
```

Hello World!

This paragraph is not affected by the style.

Universal Selector

- The universal selector, denoted by an asterisk(*), which applies style to all elements in the document.
- For example:
  *{color: red;}
  makes all elements in the document red. Is not often useful.

Pseudo Classes

- A pseudo-class is used to define a special state of an element.
- For example, it can be used to:
  - Style an element when a user moves the mouse over it
  - Style visited and unvisited links differently
  - Style an element when it gets focus

```
selector:pseudo-class {
  property: value;
}
```

- Pseudo classes are styles that apply when something happens, dynamically instead of simply displaying the target element
- Names of pseudo classes begin with colons hover classes apply when the mouse cursor is over the element focus classes apply when an element has focus i.e. the mouse cursor is over the element and the left mouse button is clicked
- These two pseudo classes are supported by FX2 but IE

```
div:hover {
  background-color: blue;
}
```

Pseudo classes

```
<html >
<head> <title> Style for Input Type  Text </title>
<style type = "text/css">
        input:hover {color: red;border-top:dotted;}
        input:focus {color: green;border-color:yellow;}
</style>
</head>
<body>
<form action = "">
    <p> Your name: <input type = "text" /> </p>
</form>
</body>
 </html>
```

Anchor Pseudo Classes

- Links can be displayed in different ways.
- We can modify the way hyperlinks appear by creating style rules modifying the <a> tag with the following “pseudo-classes”:
- link
- visited
- hover
- active

a:link {color:#0000ff}

a:visited {color: #00ff00}

a:hover {color:fuschia; font-weight:bold}

a:active {font-size:30pt}

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}

/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

Anchor Pseudo Classes

```
<!DOCTYPE html>
<html><head>
<style>
    /* 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;}
</style></head>
<body>
    <h2>Styling a link depending on state</h2>
    <p><b><a href="default.asp" target="_blank">This is a link</a></b></p>
</body>
</html>
```

CSS Properties - values

- CSS properties are the styles used on specified selectors.
- They are written before values in the CSS ruleset and are separated from property values by a colon.
- Different HTML selectors and elements have different properties.
- Some properties are universal and can be used on every selector.
- Others work only on specific selectors and under particular conditions
- There are many properties and their values for HTML selectors.
- There are *520 distinct property names from 66 technical reports and 66 editor's drafts*.
- These properties are common because they are frequently used in all CSS documents and can be applied to different selectors.
- One unique thing about properties is that they have more than one value attached to them.

CSS Property

- CSS properties are the styles used on specified selectors.
- five common properties to work with
  - List properties
  - Font properties
  - Border properties
  - Text properties
  - Color Properties
- CSS VALUES
- Values are written immediately after the colon that separates them from CSS properties.
- CSS values aren't just text; they come in different forms
  - Text
  - URLs, units, measurements, integers, strings, inherit, auto, none, etc.

CSS VALUES

- Values are written immediately after the colon that separates them from CSS properties.
- CSS values aren't just text; they come in different forms
- Keywords property values are used when there are only a few possible values and they are predefined (not case sensitive)
  - Eg: small, large, medium.
- Number values can be integer or sequence of digits with decimal points and a + or – sign.
- Length value are specified as number values that are followed immediately by a two character abbreviation of a unit name.
  - There can be no space between the number and the unit name.
    px for pixels, pt for points, pc for picas (12 points) , in for inches, cm for centimeters, mm for millimeters,

CSS VALUES

- Percentage - just a number followed immediately by a percent sign: eg: font size set to 85% means new font size will be 85% of the previous font size value.
- URL values: URL property values use a form that is slightly different from references to URLs in links.
  - url(protocol://server/pathname)
  - No space should be left between URL and the left parenthesis.
- Colors : Color name rgb(n1, n2, n3).

  Hex form: #B0E0E6 stands for powder blue color.

CSS Property- values-TEXT properties

Properties	Description	Values
color	Sets the color of a text	*Hex, RGB, keyword*
text-transform	Sets the capitalization of the text	*uppercase, lowercase, capitalize, none*
text-align	Sets the alignment of the text on the screen	*right, left, center, justify*
letter-spacing	Sets the spacing between text characters	*normal, length*
text-decoration	Sets the decoration added to the text	*none, underline, line-through, overline*

CSS Property- values-Border properties

Properties	Description	Values
border	Sets the shorthand combination for border-width, border-style and border-color	*border-width, border-style, border-color*
border-color	Sets the color for the border	*Keyword, RGB, Hex, transparent, inherit*
border-radius	Sets the radius of the four corners of an element's border	*length, percentage, initial, inherit*
border-style	Sets the style for an element's border	*none, hidden, dotted, solid, dashed, double, groove, inset, outset, ridge, initial, inherit*
border-image	Sets an image as an element's border	*border-image-source, border-image-width, border-image-slice, border-image-repeat, border-image-outset, initial, inherit*

CSS Property- values-Font properties

Properties	Description	Values
font	Sets the shorthand for all the font specifications	*font-style, font-variant, font-weight, font-size/line-height, font-family, caption, icon, menu, message-box, small-caption, status-bar, inherit*
font-weight	Sets the weight of a font	*normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900, inherit*
font-style	Sets the style of a font	*Normal, italic, oblique, initial, inherit*

- The font-family property is used to specify a list of font name.The browser will use the first font in the list that it supports.
- font-family: Arial, Helvetica, Courier
- Generic fonts: They can be specified as the font family value for example :serif, sans-serif, cursive, fantasy, and monospace (defined in CSS).
- If a font name that has more than one word, it should be single-quoted Eg: font-family: 'Times New Roman'

CSS Property- values-Font properties

- Font Shorthand
- If more than one font property is to be specified than the values may be stated in a list as the value of the font property .

  Eg: font: bold 24pt 'Times New Roman' Palatino Helvetica
- The order which browser follows is last must be font name, second last font size and then the font style, font variant and font weight etc can be in any order but before the font size and names.
- ***Font-size*** **-**Possible values: a length number or a name, such as smaller, xx-large, medium , large etc.
- ***Font-variant*** **-**The default value of the font-variant property is normal, can be set to small-caps to specify small capital characters.
- ***Font-style*** **-**The property is most commonly used to specify italic,
- ***Font-weight*** **-**The property is used to specify the degree of boldness.
- ***Text Decoration*****-**line-through , overline , underline, none
- **Text Spacing:** letter spacing , word spacing , line-height

CSS Property- values-List properties

Properties	Description	Values
list-style	Shorthand combination for list-style-type, list-style-position, and list-style-image	*list-style-type, list-style-position, list-style-image, inherit*
list-image	Sets an image as the list-item marker	*none, url, initial, inherit*
list-type	This sets the type of list-item marker	*none, disc, circle, square, decimal, decimal-leading-zero, armenian, georgian, lower-alpha, upper-alpha, lower-greek, lower-latin, upper-latin, lower-roman, upper-roman, inherit*

CSS Property- values-List properties

- It is used to specify style of bullets or sequencing values in list items.
- The list-style-type of Unordered lists can be set to disc,circle,square or none.
- Bullet can be a disc (default), a square, or a circle

```
<li style = "list-style-type: circle">BTECH </li>
```

- Could use an image for the bullets in an unordered list.
- **Example:**

```
<li style = "list-style-image: url(book.jpg)">
```

CSS Property- values-List properties

- When ordered lists are nested, it is best to use different kinds of sequence values for the different levels of nesting
- Decimal Arabic numerals 1, 2, 3, 4
- upper-alpha Uc letters A, B, C, D
- lower-alpha Lc letters a, b, c, d
- upper-roman Uc Roman I, II, III, IV
- lower-roman Lc Roman i, ii, iii, iv

```
<style type = "text/css">
    ol {list-style-type: upper-roman;}
    ol ol {list-style-type: upper-alpha;}
    ol ol ol {list-style-type: decimal;}
 </style>
```

CSS Property- values-color properties

- color –specify the foreground color of html elements

  color:red

- background-color→set the background color of an element

  background-color:grey

- Web Palette contains , one of 16 million different colors
- A set of 16 generic colors that are guaranteed to be displayable
  - Names for some:
    - blue, red, green, pink
  - Hexadecimal
    - #0000FF, #FF0000, #00FF00, #FF3399
  - RGB
    - rgb(0,0,255), rgb(255,0,0), rgb(0,255,0)
  - RGB%
    - rgb(0%,0%,100%), rgb(100%,0%,0%)

CSS Property- values-other properties

1.text-align property has the possible values, **left (the default), center, right, or justify.**

- we want text to flow around another element - the float property.

2. float property has the possible values, **left, right, and none (the default)**.

- Float property is used to specify that text should flow around some element.

3. text-indent property used to indent the first line of a paragraph

CSS Property- values-Background

- CSS provides control over the backgrounds of block-level elements.
- CSS can set a background color or add background images to HTML5 element
- background-color→set the background color of an element

  background-color:grey
- The **background-image** property is used to place an image in the background of an element Repetition can be controlled.
- Background image can be replicated to fill the area of the element. This is known as tiling.

  background-repeat property possible values:repeat (default), no-repeat, repeat-x, or repeat-y

  background-position property. Possible values: top, center, bottom, left, or right.

CSS Property- values-Example

```
<html><head>    <title> sjc</title><meta charset = “utf-8” />
<style type = "text/css">
body { background-color:grey;background-
image:url("agriculture.jfif");background-repeat:repeat-y;
        background-position:center;      }
Img  {float:left; }
</style> </head>
<body>
    <h1> SJCET ADMISSION</h1>.
   <img src="alogo.jfif"/><p>SJCET structure that grows from the blossom.
   Minority catholic institution .provides Btech and MTECh courses</p>
   <p > SJCET BTECH ADMISSION 2022... </p>
   <p >SJHMCT hotel management institution . Minority catholic institution
   .provides BHRM courses</p> <h2 > SJCET BHRM ADMISSION 2022... </h2>
   <h3 > SJCET BHRM ADMISSION 2022... </h2><h3 > SJCET PG ADMISSION
   2022... </h3>
   <p> PG courses MTECH,MBA,MTECH available at SJCET</p>
</body>  </html>
```

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Background Images</title>
      <style type = "text/css">
         body  { background-image: url(logo.png);
                 background-position: bottom right;
                 background-repeat: no-repeat;
                 background-attachment: fixed;
                 background-color: lightgrey; }
         p     { font-size: 18pt;
                 color: Darkblue;
                 text-indent: 1em;
                 font-family: arial, sans-serif; }
         .dark { font-weight: bold; }
      </style>
   </head>
   <body>
      <p>
         This example uses the background-image,
         background-position and background-attachment
         styles to place the <span class = "dark">Deitel
         & Associates, Inc.</span> logo in the
         bottom-right corner of the page. Notice how the logo
         stays in the proper position when you resize the
         browser window. The background-color fills in where
         there is no image.
      </p>
   </body>
</html>
```

CSS Property- values-Background

- background-image Property
  - The background-image property specifies the image URL for the image logo.png in the format url(fileLocation).
  - You can also set the background-color property in case the image is not found
- background-position Property
  - The background-position property places the image on the page. The keywords top, bottom, center, left and right are used individually or in combination for vertical and horizontal positioning.
  - For example, to position the image as horizontally centered (positioned at 50 percent of the distance across the screen) and 30 pixels from the top, use

```
background-position: 50% 30px;
```

CSS Property- values-Background

- background-attachment
  - fixed Property The next property setting, 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.
- background-repeat Property
  - The background-repeat property controls background image tiling, which places multiple copies of the image next to each other to fill the background.
  - Here, we set the tiling to no-repeat to display only one copy of the background image.
  - Other values include repeat (the default) to tile the image vertically and horizontally, repeat-x to tile the image only horizontally or repeat-y to tile the image only vertically.
- text-indent property to indent the first line of text in the element by a specified amount, in this case 1em.
- font-style property formats text is the font-style property, which allows you to set text to none, italic or oblique

Positioning Elements: Absolute Positioning, z-index

- The CSS position property is used to set the position for an element
- CSS introduced the position property, which gives you greater control over how document elements are displayed
- The position property specifies the type of positioning method used for an element.
- 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.

Positioning Elements: Absolute Positioning, z-index

- **Absolute Position**: An element with position: absolute; will cause it to adjust its position with respect to its parent. If no parent is present, then it uses the document body as parent.

  position: absolute;

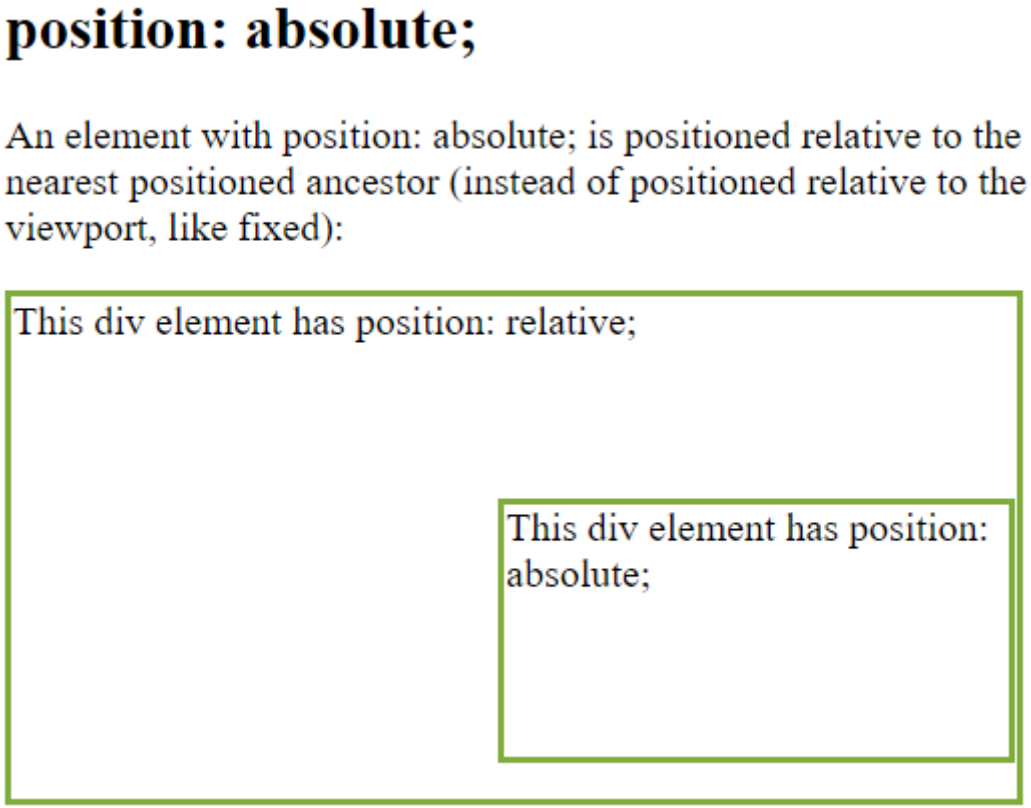
- **Relative Position**: Setting the top, right, bottom, and left properties of an element with **position: relative;** property will cause it to adjust from its normal position. The other objects or elements will not fill the gap
- The element that is set to relative position can be shifted with respect to other elements in the document. The element is shifted using top, right, bottom, and left properties

Positioning Elements: Absolute Positioning, z-index

- An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
- Note: Absolute posi he normal flow, and can overlap element

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

```
<!DOCTYPE html>
<html>
<head>
<style>
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;
}
</style>
</head>
<body>
<h2>position: absolute;</h2>
<p>An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):</p>
<div class="relative">This div element has position: relative;
  <div class="absolute">This div element has position: absolute;</div>
</div>
</body>
</html>
```



```
<!DOCTYPE html> <html> <head>
<meta charset = "utf-8">
<title>Relative Positioning</title>
<style type = "text/css">
       p { font-size: 1.3em;
       font-family: verdana, arial, sans-serif; }
       span { color: red;font-size: .6em;height: 1em; }
       .super { position: relative;top: -1ex; }
      .sub { position: relative;bottom: -1ex; }
      .shiftleft { position: relative;left: -1ex; }
      .shiftright {position: relative;right: -1ex; }
</style> </head>
<body>
   <p>The text at the end of this sentence <span class = "super">is in
  superscript</span></p>
   <p>The text at the end of this sentence <span class = "sub">is in
  subscript</span></p>
   <p>The text at the end of this sentence<span class = "shiftleft">is shifted
  left</span></p>
   <p>The text at the end of this sentence<span class = "shiftright">is
  shifted right</span></p> </body> </html>
```

Positioning Elements:Relative Positioning

The text at the end of this sentence is in superscript

The text at the end of this sentence is in subscript

The text at the end of this sentence is shifted left

The text at the end of this sentence is shifted right

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Absolute Positioning</title>
      <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;
                             font-size: 20pt;
                             font-family: tahoma, geneva, sans-serif; }
      </style>
   </head>
   <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>
</html>
```

Absolute positioning of elements. (Part 1 of 2.)

Absolute positioning of elements. (Part 2 of 2.)

Positioning Elements: Absolute Positioning, z-index

- 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.
- In this example, .background_image has the lowest z-index (1), so it displays in the background. The .foreground_image CSS rule (lines 14–17) gives the circle image (foreground_image.png, in lines 30–31) a z-index of 2, so it displays in front of background_image.png.
- The p element in line 33 is given a z-index of 3 in line 21, so its content (Positioned Text) displays in front of the other two.
- If you do not specify a z-index or if elements have the same z-index 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.

Positioning Elements: Relative Positioning

- In relative positioning, elements are positioned relative to other elements
- 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.
- Setting the position property to relative, as in class super (lines 15–16), lays out the element on the page and offsets it by the specified top, bottom, left or right value.
- relative positioning keeps elements in the general flow of elements on the page, so positioning is relative to other elements in the flow.

Positioning Elements: Relative Positioning

- Setting the position property to relative, as in class super (lines 15–16), lays out the element on the page and offsets it by the specified top, bottom, left or right value.
- Class super (lines 15–16) lays out the text at the end of the sentence as superscript, and
- class sub (lines 17–18) lays out the text as subscript relative to the other text.
- Class shiftleft (lines 19–20) shifts the text at the end of the sentence left and class shiftright (lines 21–22) shifts the text right

Span and Div

- and are tags that let you select a group of elements and apply styles to them
- is an inline tag
  - no breaks are added before or after
- is a block tag
  - a break is usually added by the browser before and after the tags
- The `<span>` tag is similar to other HTML tags,
- they can be nested and they have `id` and `class` attributes
- Another tag that is useful for style specifications:

`<div>` - Used to create document sections (or divisions) for which style can be specified

Span and Div

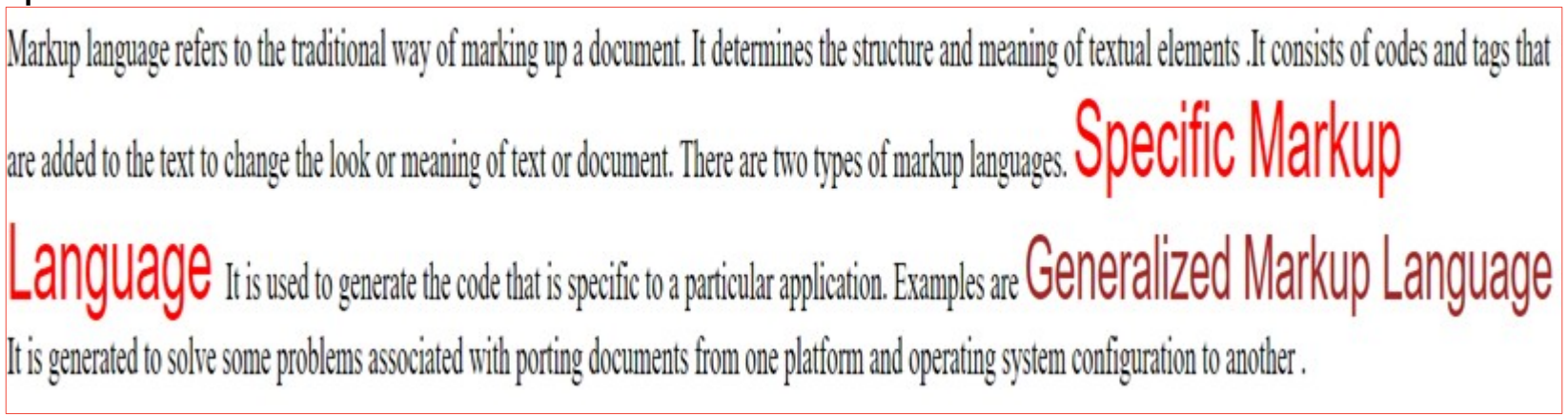
```
<html><head>
<style>
div {   line-height: 20px;
    margin: 30px;
    padding-bottom: 20px;
    text-align: justify;
    width: 140px;
    color: red; }
</style></head>
<body><p>A div element is displayed like this:
<div>This is some text in a div element.This is some text
    in a div element.This is some text in a div element.This
    is some text in a div element.This is some text in a div
    element.This is some text in a div element.</div>
Change the default CSS settings to see the effect.
</p></body></html>
```

A div element is displayed like this:

This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.

Change the default CSS settings to see the effect.

```
<!DOCTYPE html >
<html >
<head> <title>Selectors.html</title>
        <style type="text/css">
                .spanred {font-size:24pt;font-family:Arial;color:red;}
                .spanbrown {font-size:20pt;font-family:Arial;color:brown;}
        </style>
</head>
<body>
        <p>Markup language refers to the traditional way of marking up a document. It determines the structure and meaning of textual elements . There are two types of markup languages.
<span class="spanred">Specific Markup Language </span>
It is used to generate the code that is specific to a particular application. Examples are
<span class="spanbrown">Generalized  Markup Language</span>
It is generated to solve some problems associated with porting documents from one platform and operating system configuration to another</p>
</body></html>
```



Span and Div

```
<html><head>
<style>
div {
        line-height: 20px;
         margin: 30px;
         text-align: justify;
         width: 140px;
         color: red;
        background-color:green;
         }
</style></head><body><p>A div element is displayed like this:
<div>This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.This is some text in a div element.</div>Change the default CSS settings to see the effect.</p></body></html>
```

A div element is displayed like this:

This is some text in a div element.This is some text in a div element. This is some text in a div element.This is some text in a div element. This is some text in a div element. This is some text in a div element.

Change the default CSS settings to see the effect.

Differentiate between block tags and inline tags. Illustrate with suitable examples?

- In HTML, elements are categorized as either **block-level tags** or **inline tags** based on how they behave in the document's structure.
- A block-level element always starts on a new line and takes up the full width available
- Block-level tags create a new block of content and can contain other block-level or inline elements.ex: ,<h1>,<ul>,<p>etc
- An inline element does not start on a new line and it only takes up as much width as necessary
- The element is a block-level and is often used as a container for other HTML elements
- The element is an inline container used to mark up a part of a text, or a part of a document.ex:<img>,<a>, etc

Prof.Smitha Jacob, Department of Computer Science and Engineering, SJCET Palai

Box Model

Box Model

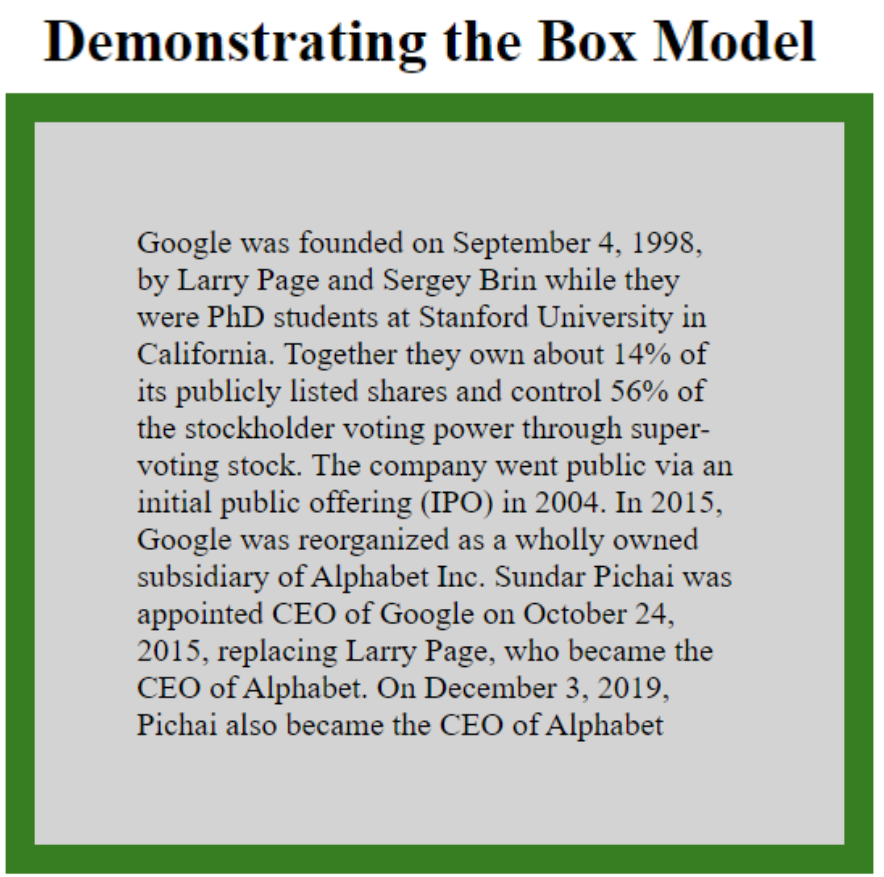
When the browser renders an element using the box model, the content is surrounded by padding, a border and a margin

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.
- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The amount of space between the content of an element and its border , known as padding. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The space between the border and an adjacent element known as margin. The margin is transparent
- The box model allows us to add a border around elements, and to define space between elements.

Box model

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  background-color: lightgrey;
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
</style>
</head>
<body>
<h2>Demonstrating the Box Model</h2>
<div>Google was founded on September 4, 1998, by Larry Page and Sergey Brin while they were PhD students at Stanford University in California. Together they own about 14% of its publicly listed shares and control 56% of the stockholder voting power through super-voting stock. The company went public via an initial public offering (IPO) in 2004. In 2015, Google was reorganized as a wholly owned subsidiary of Alphabet Inc. Sundar Pichai was appointed CEO of Google on October 24, 2015, replacing Larry Page, who became the CEO of Alphabet. On December 3, 2019, Pichai also became the CEO of Alphabet</div>
</body>
</html>
```



Box model-controlling the margin

- To define the margins of an element, use:
  - margin : *value*
  - where *value* = a length value (“em” is often used), a percentage (a margin proportional to the element’s width, or auto
- To set margins on a side, use:
  - **margin-top**
  - **margin-right**
  - **margin-bottom**
  - **margin-left**
- E.g., body {margin-left:12px; margin-right:3%; margin-top:10px; margin-bottom:1em}
  - (“em” is often used), a margin proportional to the element’s width, or auto

Box model-setting the padding

- To define padding, use:
  - padding: *value*
  - where *value* = a length value or a percentage (a padding proportional to the element's width)
- To set padding on a side, use:
  - **padding-top**
  - **padding-right**
  - **padding-bottom**
  - **padding-left**

Box model-formatting the border

- Border can be set in three ways:
  - **border-width(thin,thick,medium or length in pixels)**
  - **border-style (dotted,dashed,solid,double,default,none)**
  - **border-color**
  - **Short hands for setting border**
  - **Ex: border:5px solid blue ;**
  - td,th { border:thin double black ; }
- To set the border, use:
  - border : *width_value style color*
- To set borders on a side, use:
  - border-top
  - border-bottom
  - border-left
  - border-right

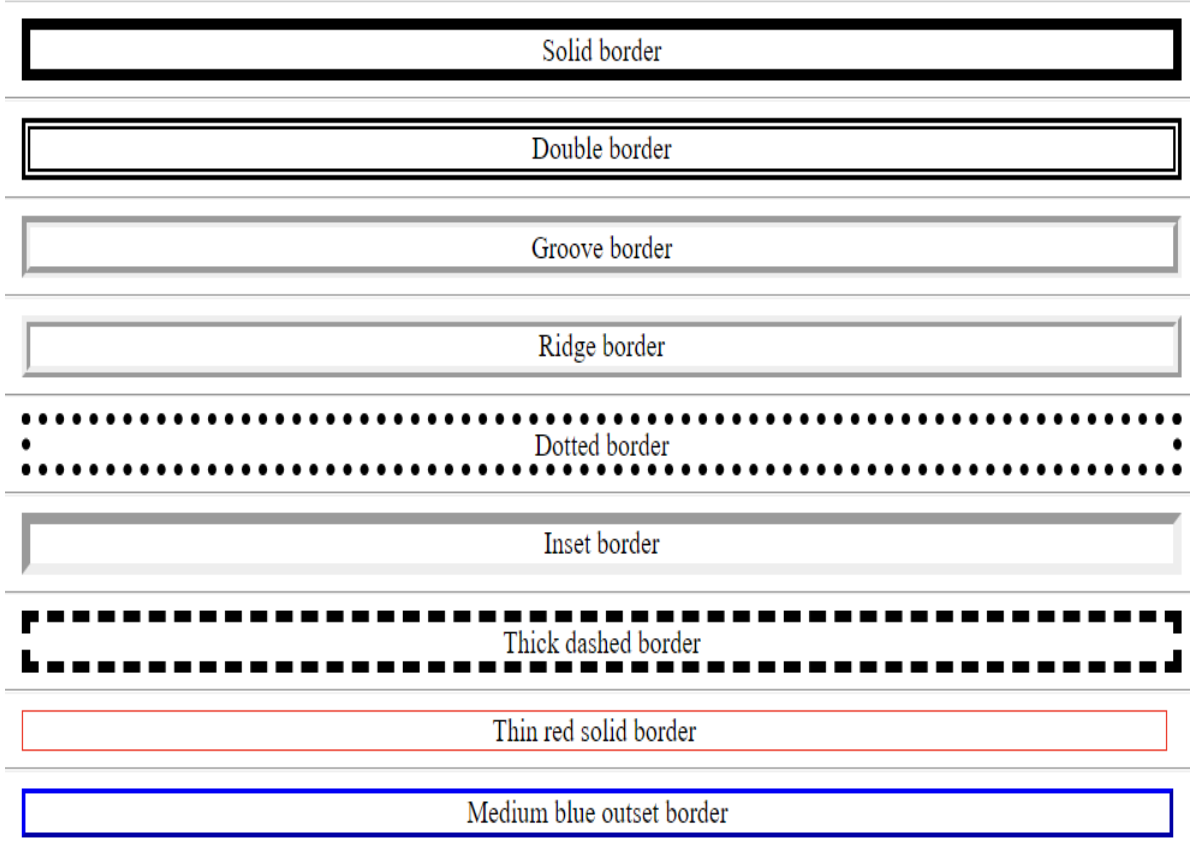
Box model-formatting the border

- Border can be set in three ways:
  - **border-width(thin,thick,medium or length in pixels)**
  - **border-style (dotted,dashed,solid,double.default-none)**
  - **border-color**
  - **Short hands for setting border**
  - **Ex: border:5px solid blue ;**
  - td,th { border:thin double black ; }
- To set the border, use:
  - border : *width_value style color*
- To set borders on a side, use:
  - border-top
  - border-bottom
  - border-left
  - border-right

Box model-formatting the border

```
<style  type="text/css">
table{
        padding-left:20px;
        padding-right:10pt;
        padding-top:3%;
        padding-bottom:1%;
        border-left:dotted;
        border-right:double;
        border-top:oblique;
        border-bottom:solid;
        border-width:10px;
        border-color:red

}
</style>
```

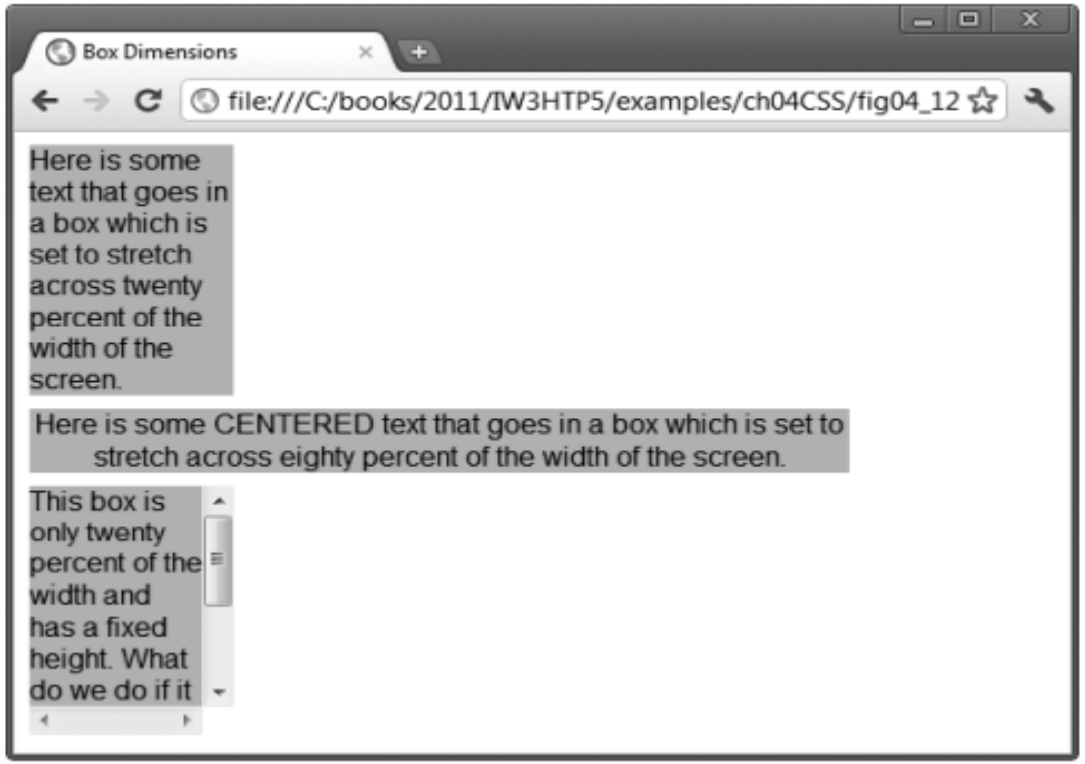
```html
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Borders</title>
      <style type = "text/css">
         div     { text-align: center;
                   width: 50%;
                   position: relative;
                   left: 25%;
                   border-width: 6px; }
         .thick  { border-width: thick; }
         .medium { border-width: medium; }
         .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; }
         .outset { border-style: outset; }
         .dashed { border-style: dashed; }
         .red    { border-color: red; }
         .blue   { border-color: blue; }
      </style>
   </head>
   <body>
      <div class = "solid">Solid border</div><hr>
      <div class = "double">Double border</div><hr>
      <div class = "groove">Groove border</div><hr>
      <div class = "ridge">Ridge border</div><hr>
      <div class = "dotted">Dotted border</div><hr>
      <div class = "inset">Inset border</div><hr>
      <div class = "thick dashed">Thick dashed border</div><hr>
      <div class = "thin red solid">Thin red solid border</div><hr>
      <div class = "medium blue outset">Medium blue outset border</div>
   </body>
</html>
```



Element Dimensions

- In addition to positioning elements, CSS rules can specify the actual dimensions of each

```
<!DOCTYPE html>



<html>
   <head>
      <meta charset = "utf-8">
      <title>Box Dimensions</title>
      <style type = "text/css">
         p { background-color: lightskyblue;
               margin-bottom: .5em;
               font-family: arial, helvetica, sans-serif; }
      </style>
   </head>
   <body>
      <p style = "width: 20%">Here is some
         text that goes in a box which is
         set to stretch across twenty percent
         of the width of the screen.</p>

      <p style = "width: 80%; text-align: center">
         Here is some CENTERED text that goes in a box
         which is set to stretch across eighty percent of
         the width of the screen.</section>

      <p style = "width: 20%; height: 150px; overflow: scroll">
         This box is only twenty percent of
         the width and has a fixed height.
         What do we do if it overflows? Set the
         overflow property to scroll!</p>
   </body>
</html>
```



Media Types and media queries

Responsive web design

- Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops
- Responsive web design provides an optimal experience, easy reading and easy navigation with a minimum of resizing on different devices such as desktops, mobiles and tabs).
- Responsive web design makes your web page look good on all devices.
- Responsive web design uses only HTML and CSS.
- Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:
- It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

Responsive web design

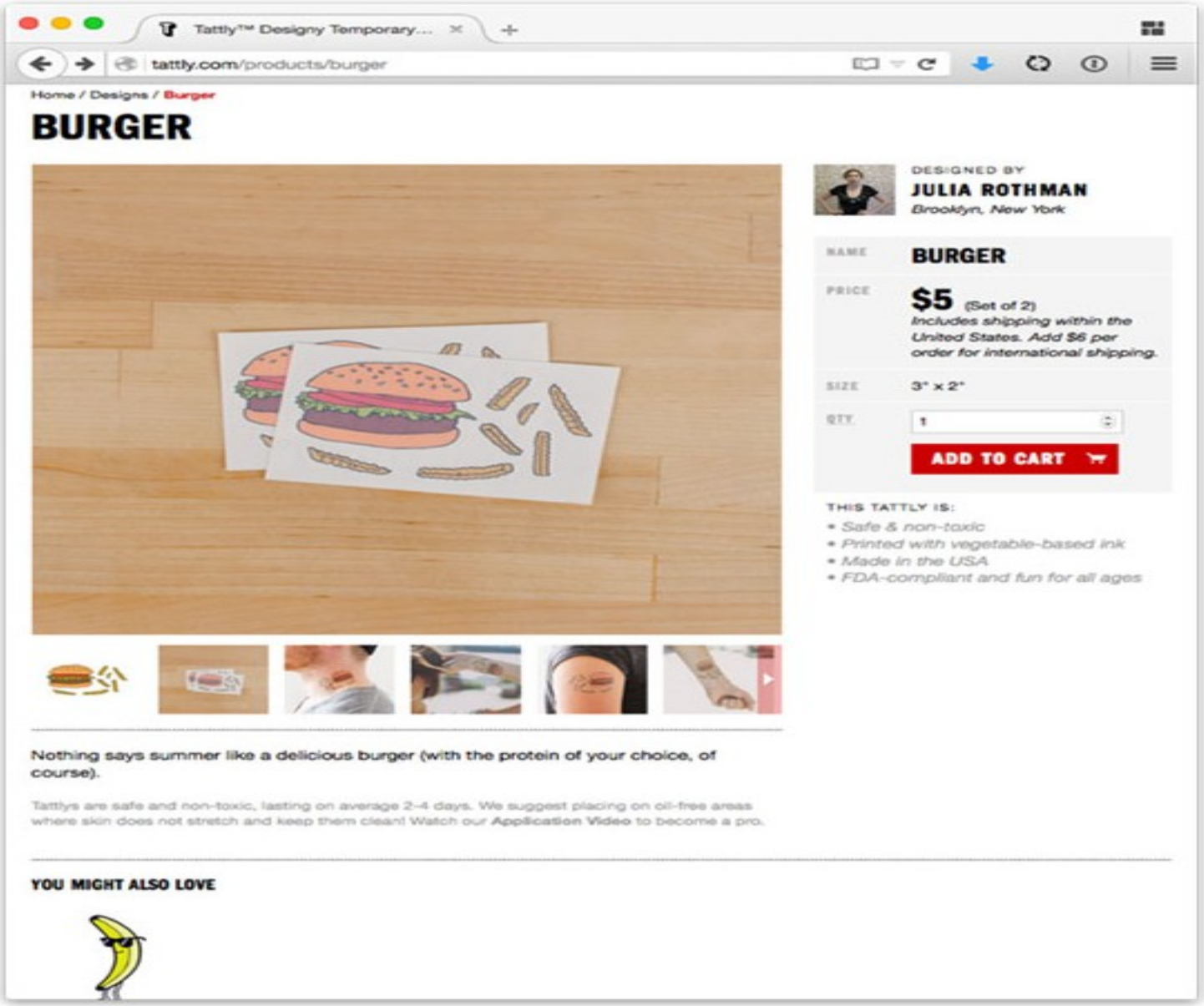
- **Breakpoints** are the building blocks of responsive design. Use them to control when your layout can be adapted at a particular viewport or device size.
- media queries are used to architect the CSS by breakpoint.
- Media queries are a feature of CSS that allow you to conditionally apply styles based on a set of browser and operating system parameters.
- We most commonly use **min-width** in our media queries.
- Mobile first, responsive design is the goal.
- Bootstrap's CSS aims to apply the bare minimum of styles to make a layout work at the smallest breakpoint, and then layers on styles to adjust that design for larger devices.

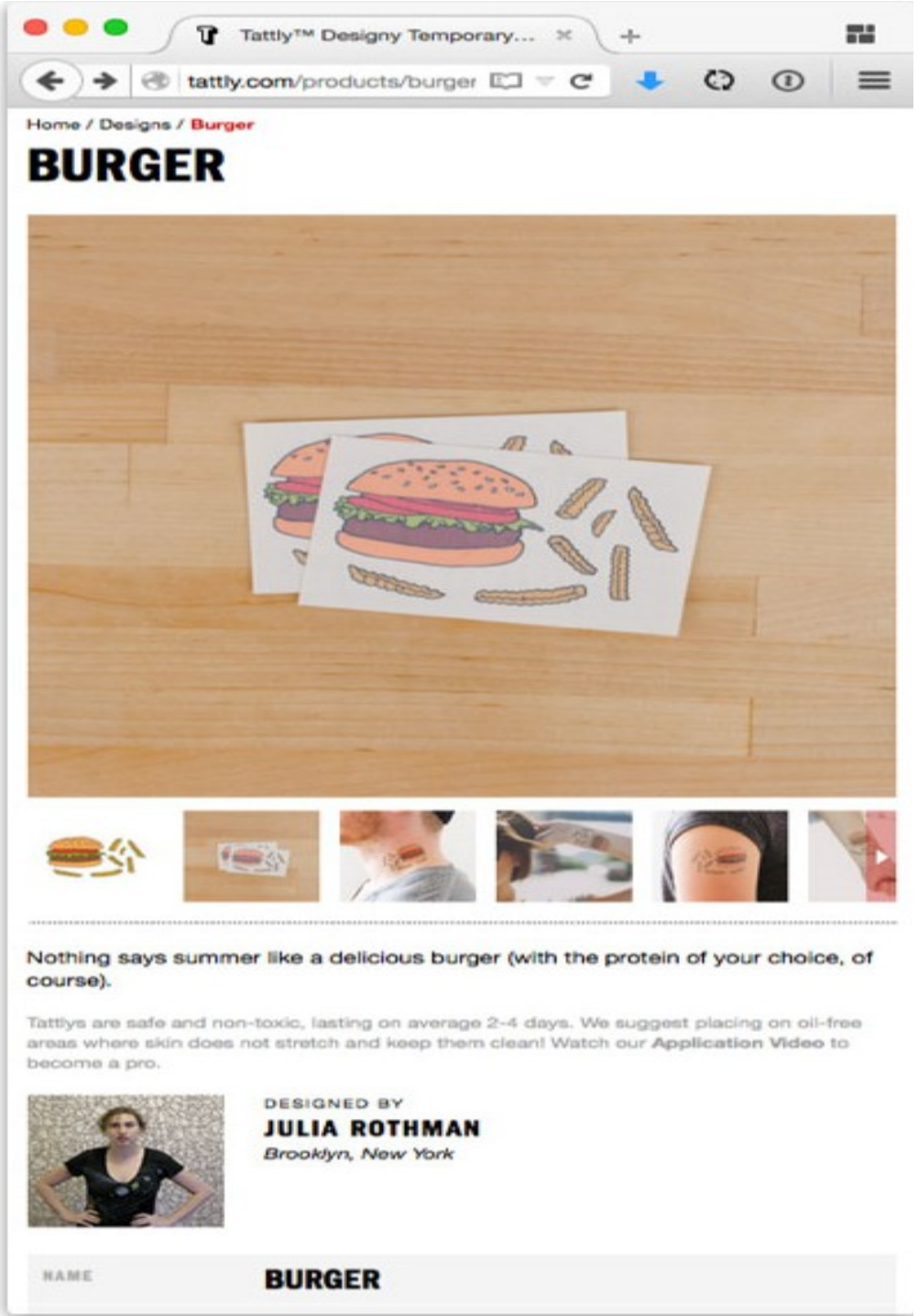
Available breakpoints

- **Breakpoints** are the building blocks of responsive design.
- media queries are used to architect the CSS by breakpoint.

Breakpoint	Class infix	Dimensions
X-Small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Responsive web design





Media types and media queries

- CSS media types allow you to decide what a page should look like, depending on the kind of media being used to display the page.
- Media queries are a key part of responsive web design, as they **allow you to create different layouts depending on the size of the viewport**
- 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.
- The most common media type for a web page is the screen media type, which is a standard computer screen.
- Other media types in CSS include handheld, braille, speech and print

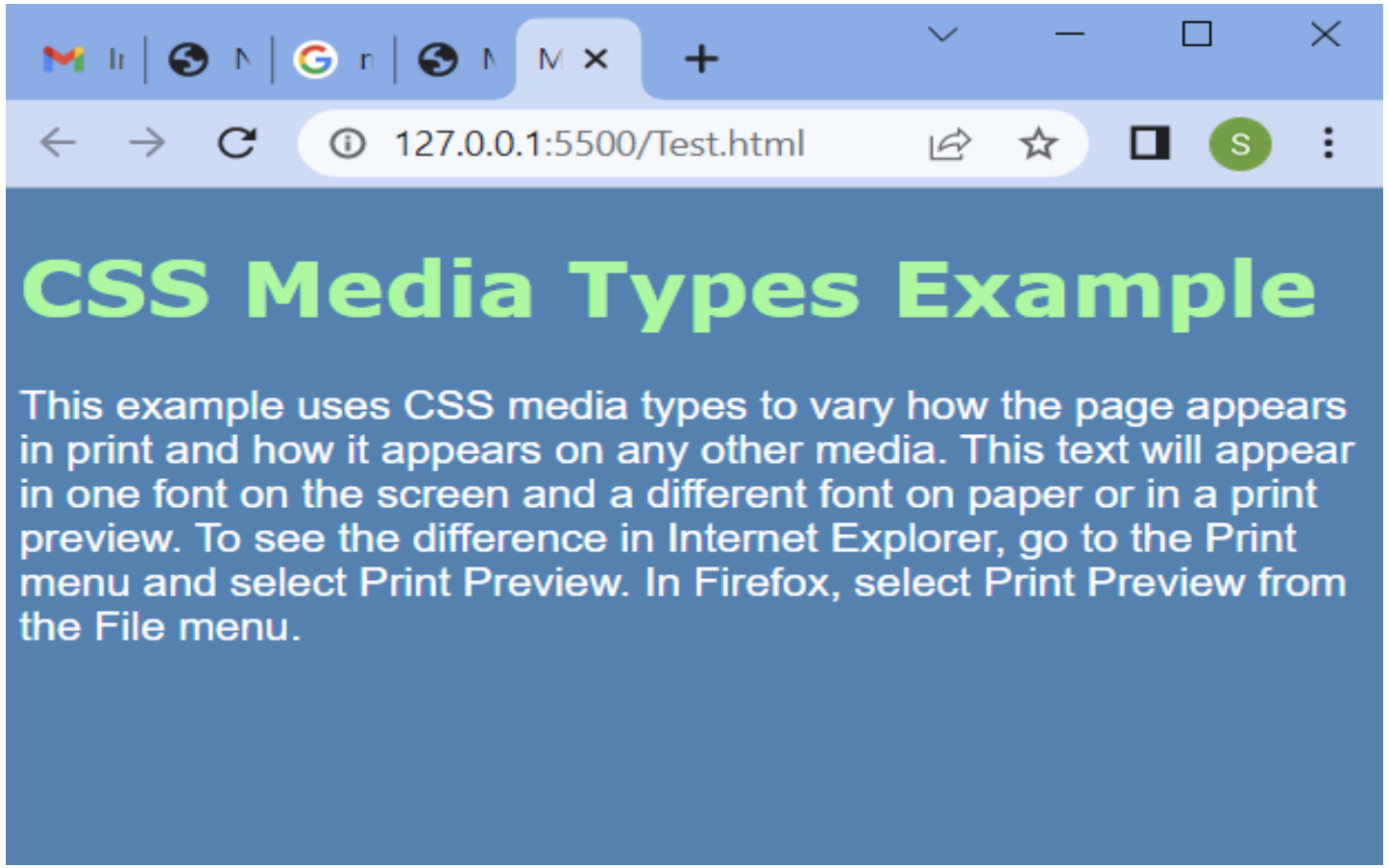
Media types and media queries

- The handheld medium is designed for mobile Internet devices such as smartphones, while braille is for machines that can read or print web pages in braille.
- speech styles allow you to give a speech-synthesizing web browser more information about the content of a page.
- The print media type affects a web page's appearance when it's printed.
- For a complete list of CSS media types, see

```
http://www.w3.org/TR/REC-CSS2/media.html#media-types
```

- Media types allow you to decide how a page should be presented on any one of these media without affecting the others
- Example that applies one set of styles when the document is viewed on all media (including screens) other than a printer, and another when the document is printed.

```
<!DOCTYPE html><html><head>
    <meta charset="utf-8">
    <title>Media Types</title>
</head>
<style type="text/css">
    @media all{
      body {
          background-color: steelblue;
      }

      h1 {
          font-family: verdana, helvetica, sans-serif;
          color: palegreen;
      }

      p {
          font-size: 12pt;
          color: white;
          font-family: arial, sans-serif;
      }
    }
    /* End @media all declaration. */
    @media print
        {
        body {background-color: white;}
        h1 {color: seagreen;}
        p {font-size: 14pt;
           color: steelblue;
           font-family: "times new roman", times, serif;}
    }
    /* End @media print declaration. */
</style>
<body><h1>CSS Media Types Example</h1>
    <p> This example uses CSS media types to vary how the page
        appears in print and how it appears on any other media.
        This text will appear in one font on the screen and a
        different font on paper or in a print preview. To see
        the difference in Internet Explorer, go to the Print
        menu and select Print Preview. In Firefox, select Print
        Preview from the File menu.
    </p>
</body>
</html>
```



CSS Media Types Example

This example uses CSS media types to vary how the page appears in print and how it appears on any other media. This text will appear in one font on the screen and a different font on paper or in a print preview. To see the difference in Internet Explorer, go to the Print menu and select Print Preview. In Firefox, select Print Preview from the File menu.

Media queries

- Media query is a CSS technique introduced in CSS3.
- Media queries can be used to check many things, such as:
  - **width and height** of the viewport
  - **width and height of the device**
  - **orientation** (is the tablet/phone in landscape or portrait mode?)
  - **resolution**
- Using media queries is a popular technique for ***delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones***
- A media query consists of a media type and can contain one or more expressions, which resolve to either true or false.
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.

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

Media queries

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

- meaning of the **not**, **only** and **and** keywords:
- **not:** The not keyword inverts the meaning of an entire media query.
- **only:** The only keyword prevents older browsers that do not support media queries with media features from applying the specified styles. **It has no effect on modern browsers.**
- **and:** The and keyword combines a media feature with a media type or other media features.
- They are all optional. However, if you use **not** or **only**, you must also specify a media type.
- Media Types are:

Value	Description
all	Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

Media queries

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

- If the browser window is 600px or smaller, the background color will be lightblue:

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

Media queries

- Use mediaqueries to set the background-color to lavender if the viewport is 800 pixels wide or wider, to lightgreen if the viewport is between 400 and 799 pixels wide. If the viewport is smaller than 400 pixels, the background-color is lightblue

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

@media screen and (min-width: 400px) {
  body {
    background-color: lightgreen;
  }
}

@media screen and (min-width: 800px) {
  body {
    background-color: lavender;
  }
}
```

Media queries

- Media queries allow you to format your content to specific output devices. Media queries include a media type and expressions.
- Some of the common media features include:
- width—the width of the part of the screen on which the document is rendered, including any scrollbars
- height—the height of the part of the screen on which the document is rendered, including any scrollbars
- device-width—the width of the screen of the output device
- device-height—the height of the screen of the output device
- orientation—if the height is greater than the width, orientation is portrait, and if the width is greater than the height, orientation is landscape
- aspect-ratio—the ratio of width to height
- device-aspect-ratio—the ratio of device-width to device-height

Media queries

- Media queries are useful when you want to modify your site or app depending on a device's general type (such as print vs. screen) or specific characteristics and parameters (such as screen resolution or browser viewport width).
- Media queries are used for the following:
  - To conditionally apply styles with the CSS @media and @import at-rules.
  - To target specific media for the <style>, <link>, <source>, and other HTML elements with the media= attribute.

Media queries

```
<!DOCTYPE html><html lang="en">
<head><meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
* {
  box-sizing: border-box;
}
/* Style the top navigation bar */
.topnav {
  overflow: hidden;
  background-color: aqua;
}
/* Style the topnav links */
.topnav a {
  float: left;
  display: block;
  color: #671212;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
```

SJCET PALAI

Resize the browser window to see the effect: When the screen is less than 600px, the navigation menu will be displayed vertically instead of horizontally.

Home

About SJCET

Contact Us

```
/* Change color on hover */
.topnav a:hover {
  background-color: #ddd;  color: black;
}
/* On screens that are 600px wide or less, make the menu links
stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .topnav a {  float: none;   width: 100%; }
}
</style></head><body>
<h2>SJCET PALAI</h2>
<p>Resize the browser window to see the effect: When the screen is less
    than 600px, the navigation menu will be displayed vertically instead of horizontally.</p>
<div class="topnav">
  <a href="#">Home</a>
  <a href="#">About SJCET</a>
  <a href="#">Contact Us</a>
</div>
</body>
</html>
```

SJCET PALAI

Resize the browser window to see the effect: When the screen is less than 600px, the navigation menu will be displayed vertically instead of horizontally.

Home About SJCET Contact Us

JAVASCRIPT

MODULE 2

- (CSS, JavaScript) Introduction to Stylesheets : Introduction to CSS-Basic syntax and structure-Inline Styles, Embedded Style Sheets, Conflict Resolution, Linking External Style Sheets-Exploring CSS Selectors-Properties, values, Positioning Elements: Absolute Positioning, Relative Positioning - Backgrounds-List Styles-Element Dimensions- Table Layouts-Box Model and Text Flow-div and span -Basics of Responsive CSS, Media port & Media Queries.
- Introduction to JavaScript : Introduction to Scripting- Programming fundamentals of JavaScript -Obtaining User Input with prompt Dialogs-Arithmetic-Decision Making - Control Statements - Functions -Arrays -Objects -Document Object Model (DOM) - Form processing

JAVASCRIPT

- JavaScript is a scripting language, which is used to enhance the functionality and appearance of web pages
- JavaScript is a Client side scripting tool
- JavaScript is used in Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.
- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, Opera
- JavaScript and Java are only related through syntax →JavaScript is dynamically typed →JavaScript's support for objects is very different

Java & JAVASCRIPT

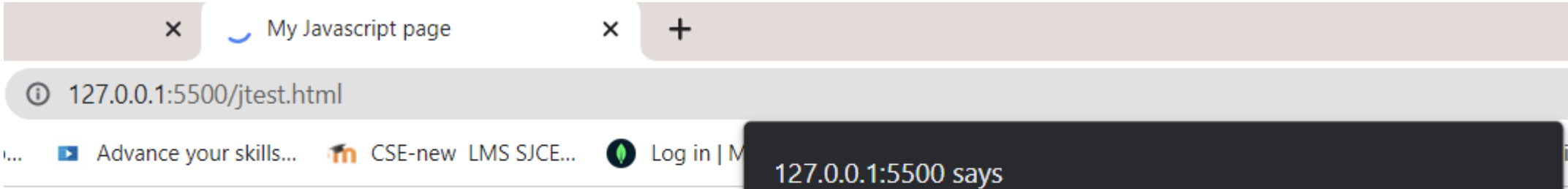
- Java and JavaScript are two completely different languages in both concept and design!
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.
- Object model of javascript is quite different from java & C++
- Java is strongly typed and Javascript is Dynamically typed
- In Java ,types are all known at compile time, where as compile time type checking impossible
- Objects in Java are static (variables or function is shared between all instances of class)where as objects are dynamic

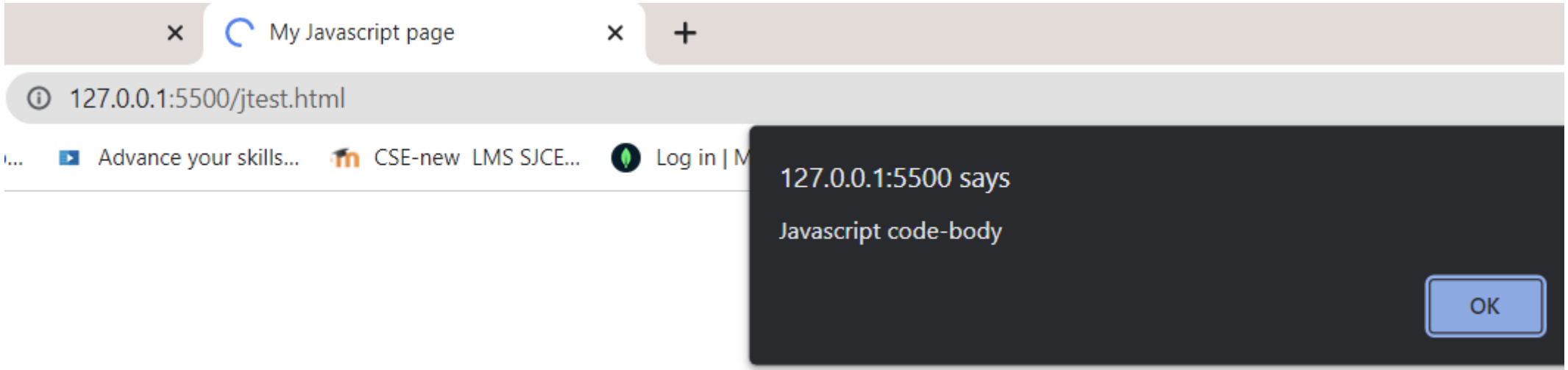
Uses of JAVASCRIPT

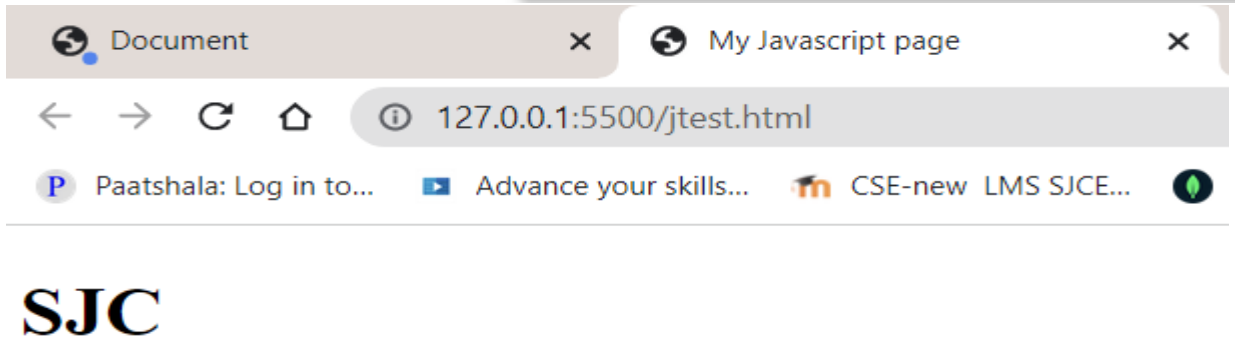
- JavaScript was designed to add interactivity to HTML
- JavaScript is a scripting language
- It is usually embedded directly into HTML pages
- JavaScript is an interpreted language
- provide programming capability at both the server and the client ends of a Web connection.
- Improve appearance especially graphics
- Site navigation
- Perform calculations
- Validation of input
- DOM allows Javascript to access &modify the style

How to put JAVASCRIPT in HTML code?

```
<!DOCTYPE html>
<html lang="EN">
<head>
<title>My Javascript page</title>
<script type="text/javascript">
<!--
alert("Javascript code-head");
// -->
</script>
</head>
<body>
<script type="text/javascript">
    alert("Javascript code-body");
</script>
<h1>SJC</h1>
</body>
</html>
```


My Javascript page
127.0.0.1:5500/jtest.html
Advance your skills...
CSE-new LMS SJCE...
Log in | M
127.0.0.1:5500 says
Javascript code-head
OK


My Javascript page
127.0.0.1:5500/jtest.html
Advance your skills...
CSE-new LMS SJCE...
Log in | M
127.0.0.1:5500 says
Javascript code-body
OK


Document
My Javascript page
127.0.0.1:5500/jtest.html
Paatshala: Log in to...
Advance your skills...
CSE-new LMS SJCE...
SJC

How to embed JavaScript in an HTML document

- When a JavaScript script is encountered in the HTML document, the browser uses its JavaScript interpreter to “execute” the script.
- Output from the script becomes the next markup to be rendered.
- When the end of the script is reached, the browser goes back to reading the HTML document and displaying its content.
- There are two different ways to embed JavaScript in an HTML document:
  - implicitly and explicitly.
- In explicit embedding, the JavaScript code physically resides in the XHTML document.
- In implicit embedding the JavaScript can be placed in its own file(.js file), separate from the XHTML document.
- When JavaScript scripts are explicitly embedded, they can appear in either part of an XHTML document—the head or the body
- The interpreter notes the existence of scripts that appear in the head of a document, but it does not interpret them while processing the head.
- Scripts that are found in the body of a document are interpreted as they are found.

How to embed JavaScript in an HTML document

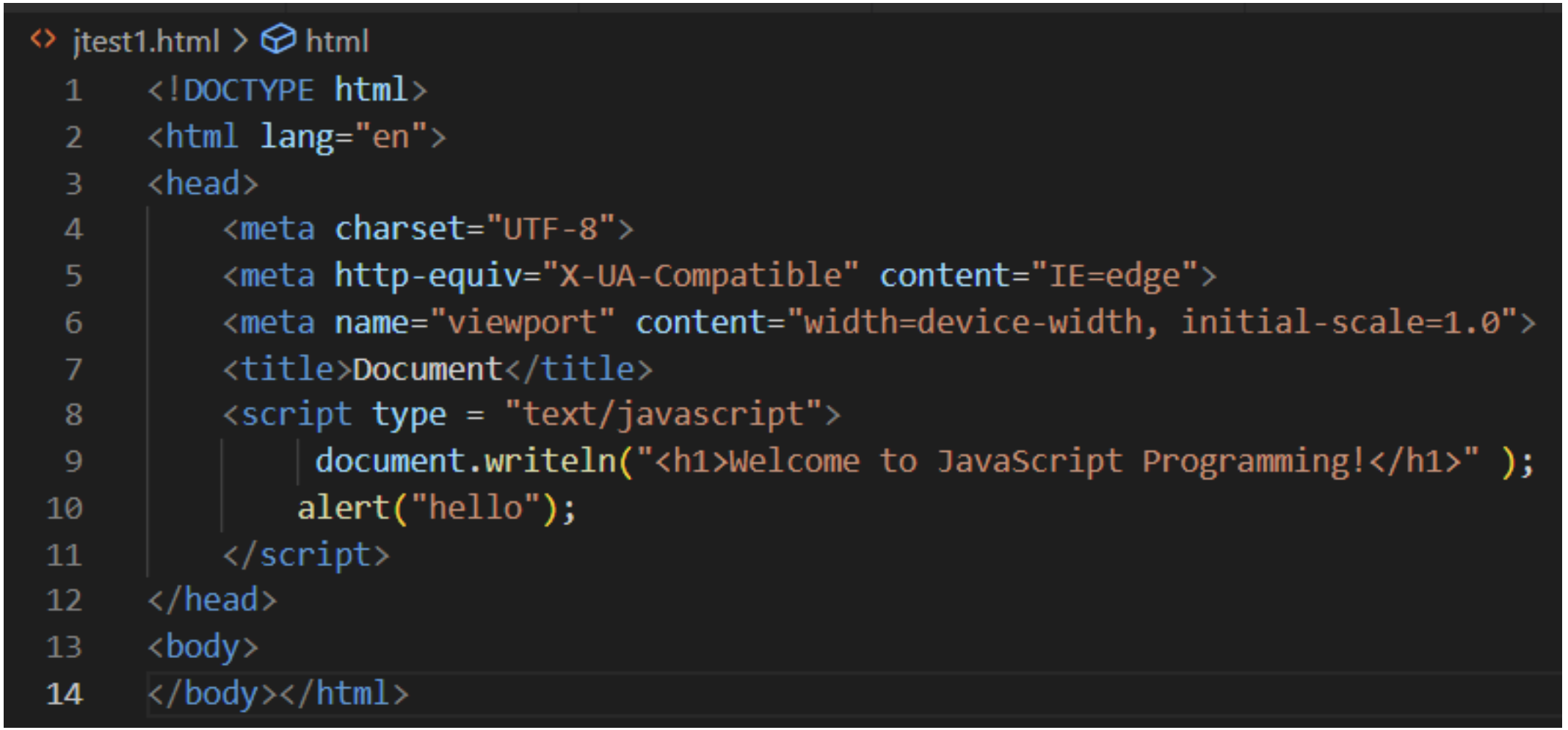
- JavaScript are embedded in HTML documents , Either directly, as in • -

```
<script type = "text/javaScript">
        -- JavaScript script –
  </script>
```

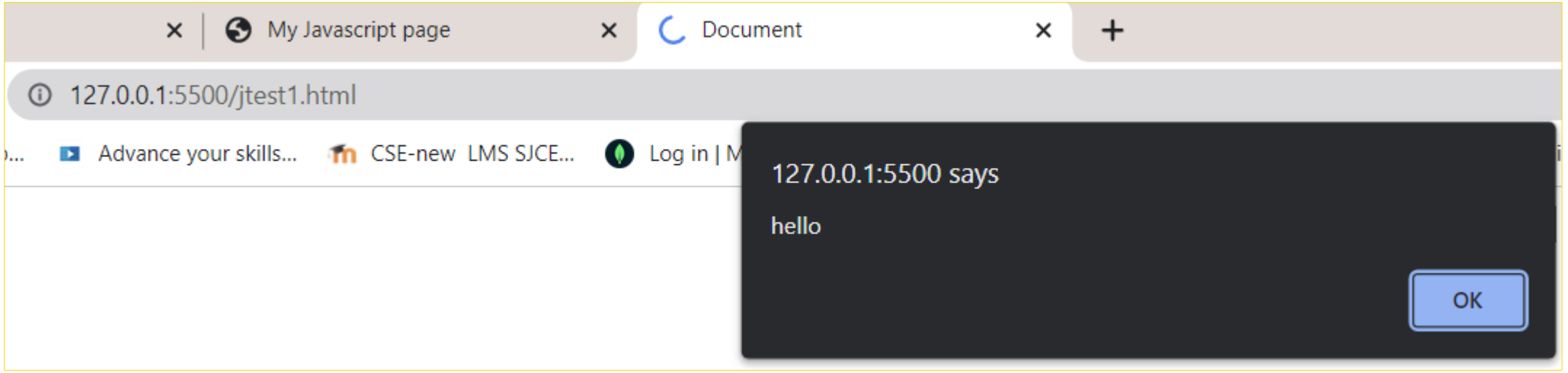
- Or indirectly, as a file specified in the src attribute of

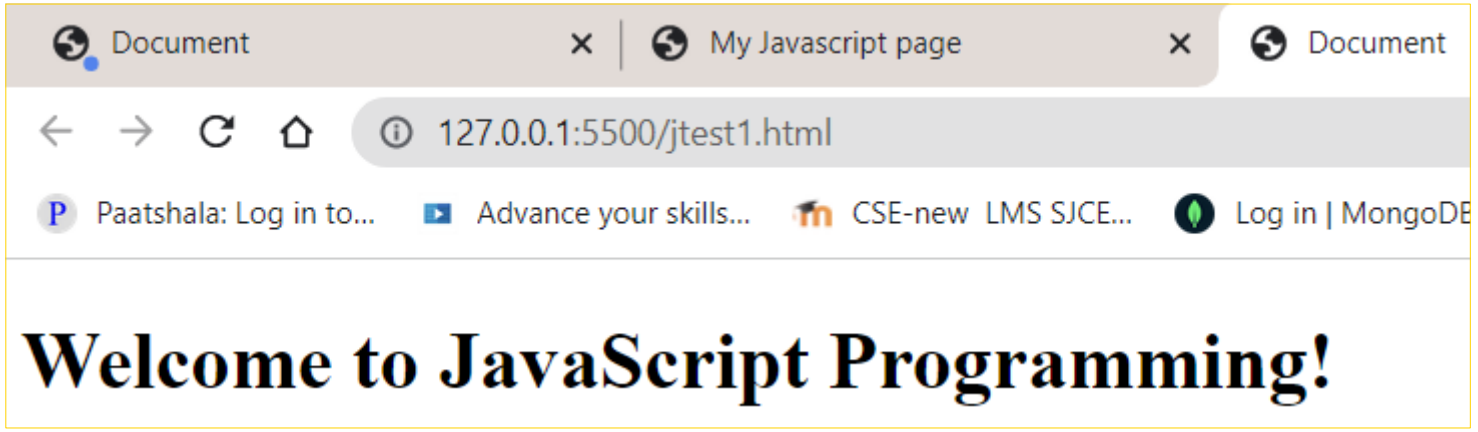
```
<script type = "text/javaScript"
          src = "myScript.js">
  </script>
```

JavaScript “hello” example in an HTML document



JavaScript “hello” example in an HTML document



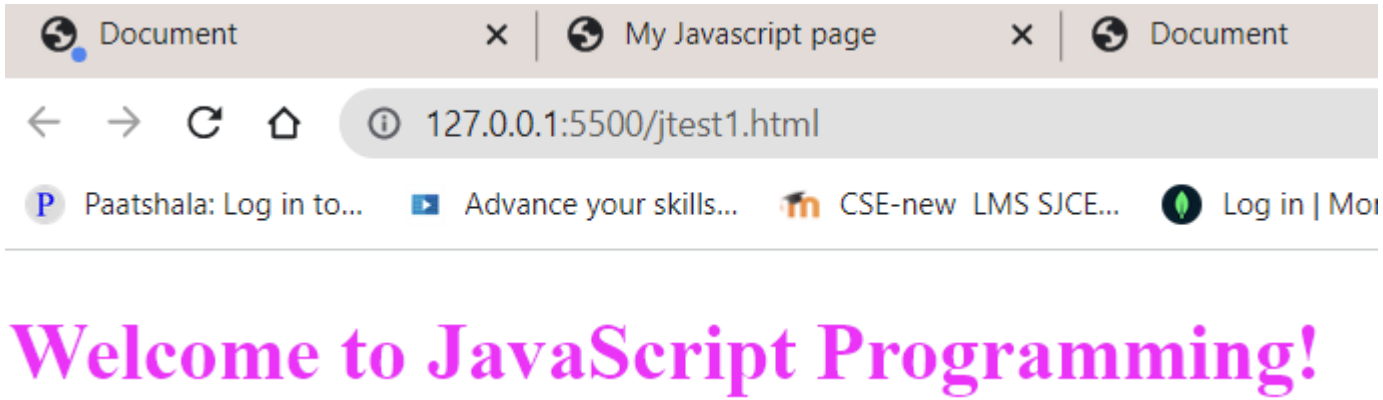


How to embed JavaScript in an HTML document

- <script> tag to indicate to the browser that the text which follows is part of a script.
- The type attribute specifies the MIME type of the script as well as the scripting language used in the script—in this case, a text file written in javascript. In HTML5,
- the default MIME type for a <script> is "text/html", so you can omit the type attribute from your <script> tags
- Individual white-space characters between words in a string are not ignored by the browser.
- browsers ignore leading white-space characters (i.e., white space at the beginning of a string).
- Strings in JavaScript can be enclosed in either double quotation marks (") or single quotation marks (‘)
- document object, which represents the HTML5 document the browser is currently displaying. This object allows you to specify text to display in the HTML5 document.
- document object’s writeln method is used to write a line of HTML5 markup in the HTML5 document.
- Method writeln instructs the browser to write the argument string into the web page for rendering.
- If the string contains HTML5 elements, the browser interprets these elements and renders them on the screen

JavaScript “hello” example with inline style

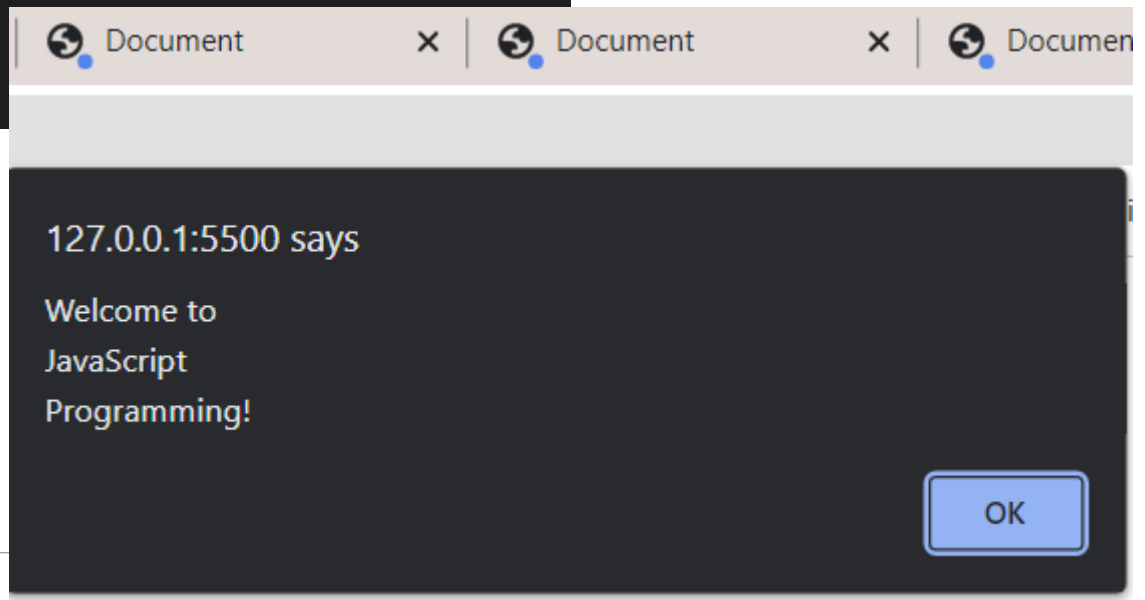
```
jtest1.html > ...
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>···
9   </head>
10  <body>
11      <script type = "text/javascript">
12          document.writeln("<h1 style = 'color: magenta'>Welcome to JavaScript Programming!</h1>" );
13      </script>
14  </body></html>
```



JavaScript example with alert

jtest1.html > html > body > script

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>···
9   </head>
10  <body>
11      <script type = "text/javascript">
12          window.alert( "Welcome to\nJavaScript\nProgramming!" );
13      </script>
14  </body></html>
```



JavaScript example with alert-Usage of Escape Sequence

- The backslash (\) in a string is an escape character.
- It indicates that a "special" character is to be used in the string.
- When a backslash is encountered in a string, the next character is combined with the backslash to form an escape sequence.

Escape sequence	Description
`\n`	*New line*—position the screen cursor at the beginning of the next line.
`\t`	*Horizontal tab*—move the screen cursor to the next tab stop.
`\\`	*Backslash*—used to represent a backslash character in a string.
`\"`	*Double quote*—used to represent a double-quote character in a string contained in double quotes. For example, `window.alert( "\"in double quotes\"" );` displays `"in double quotes"` in an `alert` dialog.
`\'`	*Single quote*—used to represent a single-quote character in a string. For example, `window.alert( '\'in single quotes\'' );` displays `'in single quotes'` in an `alert` dialog.

Some common escape sequences.

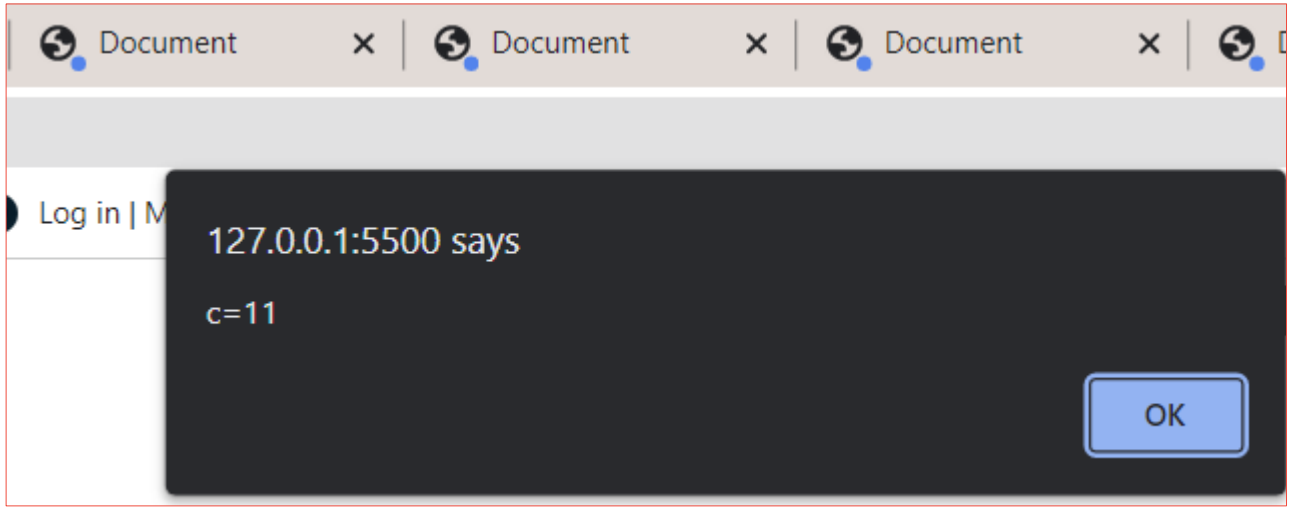
Programming fundamentals of JavaScript

GENERAL SYNTACTIC CHARACTERISTICS

- Identifier form: begin with a letter or underscore, followed by any number of
- letters, underscores, and digits
- Case sensitive
- 25 reserved words, plus future reserved words
- Comments: both // and /* … */
- Scripts are usually hidden from browsers that do not include JavaScript
- interpreters by putting them in special comments
- <!--
- -- JavaScript script –
- //-->
- Semicolons can be a problem. They are “somewhat” optional.
- Problem : When the end of the line can be the end of a statement ,JavaScript puts a semicolon there
- When separated by semicolons, multiple statements on one line are allowed: a = 5; b = 6; c = a + b;

JavaScript example with alert, No semicolon

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>···
9   </head>
10  <body>
11      <script type="text/javascript">
12          var a, b, c;
13          a = 5
14          b = 6
15          c = a + b
16          alert("c=" + c);
17      </script>
18  </body>
19  </html>
```



PRIMITIVES

- JavaScript provides different data types to hold different types of values.
  1. Primitive Types
     - five primitive types:
       - Number,
       - String
       - Boolean,
       - Undefined,
       - Null
  2. Non-Primitive Types
     - Objects such as functions and arrays are referred to as non-primitive values.

PRIMITIVES

- Primitive Types
- five primitive types: Number, String, Boolean, Undefined, or Null
- Number, String, and Boolean have wrapper objects (Number, String, and Boolean).
- wrapper objects
  - Wrapper objects are used to provide properties and methods that are convenient to use with the values of primitive types.
  - Anything that doesn't belong to any of these five primitive types is considered an object.
  - BUT each of these five primitive data types has a corresponding object constructor..
  - First as a primitive data type:

    var word = "hello";
  - And then as an object:

    var word = new String("hello");

PRIMITIVES and NonPrimitive Values

- The fundamental difference between primitives and non-primitives is that primitives are immutable and non-primitives are mutable.
- Primitives are known as being immutable data types because there is no way to change a primitive value once it gets created.

```
var s = 'This is a string.';
s[1] = 'H';
alert(s)    // 'This is a string.'
```

- Non-primitive values are mutable data types.
- The value of an object can be changed after it gets created.

```
var arr = [ 'one', 'two', 'three', 'four', 'five' ];
arr[1] = 'TWO';
alert(arr)    // [ 'one', 'TWO', 'three', 'four', 'five' ];
```

Numeric & String Literals,Other Primitives

- Numeric & String Literals
  - Numeric literals – like Java All numeric values are stored in double-precision floating point 72,7.2,.72,7E2,7e2 etc are valid
  - String literals are delimited by either ' or "
  - Can include escape sequences (e.g., \t)
  - All String literals are primitive values
- Other Primitive Types
- Boolean values are true and false.Used for evaluating Boolean Expressions
- The only Null value is null-means no value variable has not been explicitly declared or assigned a value.
- The only Undefined value is undefined-variable has been explicitly declared, but not assigned a value
- A variable declared without a value will have the value undefined. ex)var carName;

Declaring variables

- JavaScript is dynamically typed – any variable can be used for anything
- The interpreter determines the type of a particular occurrence of a variable
- Keywords are words that have special meaning in JavaScript.
- The keyword var at the beginning of the statement indicates that the word name is a variable.
- A variable is a location in the computer's memory where a value can be stored for use by a script.
- All variables have a name and value, and should be declared with a var statement before they're used in a script.
- Variables can be implicitly or explicitly declared

  var sum =0,today ="Monday", flag = false; var counter;

  - Variable names are case sensitive , They must begin with a letter or the underscore character

Identifiers and Case Sensitivity

- The name of a variable can be any valid identifier.
- An identifier is a series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and is not a reserved JavaScript keyword.
- Identifiers may not contain spaces.
- Some valid identifiers are Welcome, $value, _value, m_inputField1 and button7.
- The name 7button is not a valid identifier, because it begins with a digit, and the name ‘input field’ is not valid, because it contains a space.
- Remember that JavaScript is case sensitive—uppercase and lowercase letters are considered to be different characters, so name, Name and NAME are different identifiers.

Obtaining User Input with prompt Dialogs

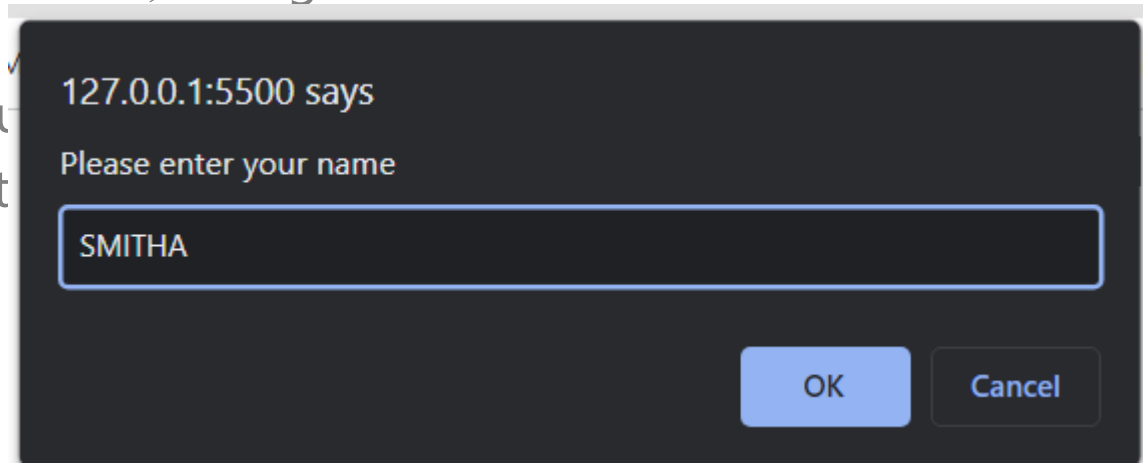
Obtaining User Input with prompt Dialogs

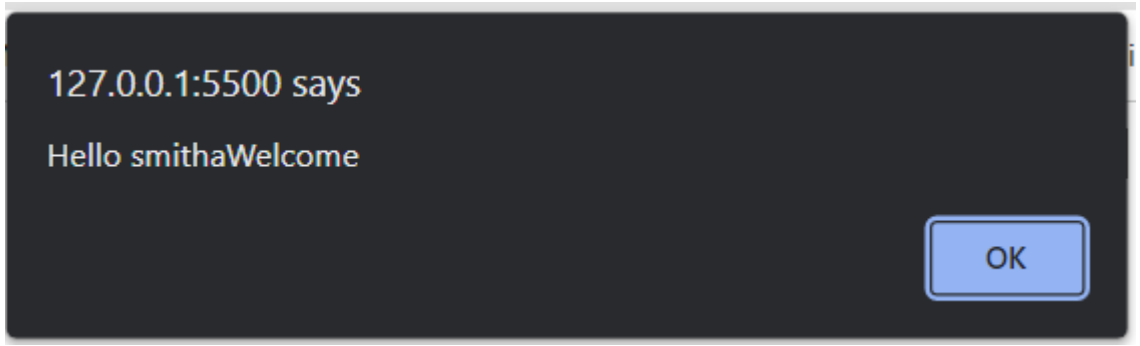
- Scripting gives you the ability to generate part or all of a web page's content at the time it's shown to the user.
- A script can adapt the content based on input from the user or other variables, such as the time of day or the type of browser used by the client. Such web pages are said to be dynamic, as opposed to static, since their content has the ability to change.
- The script uses another predefined dialog box from the window object—a prompt dialog—which allows the user to enter a value that the script can use
- The JavaScript model for the HTML document is the Document object
- The model for the browser display window is the Window object
- The Document object has a method, write, which dynamically creates content
- The parameter is a string, often catenated from parts, some of which are variables e.g., document.write("Answer: " + result);
- The Window object has three methods for creating dialog boxes, **alert, confirm, & prompt**

Obtaining User Input with prompt Dialogs

- prompt("What is your name?", "");
- Opens a dialog box and displays its string parameter, along with a text box and two buttons, OK and Cancel
- An empty string is returned for the default inpu
- Null will be returned if we press the cancel but

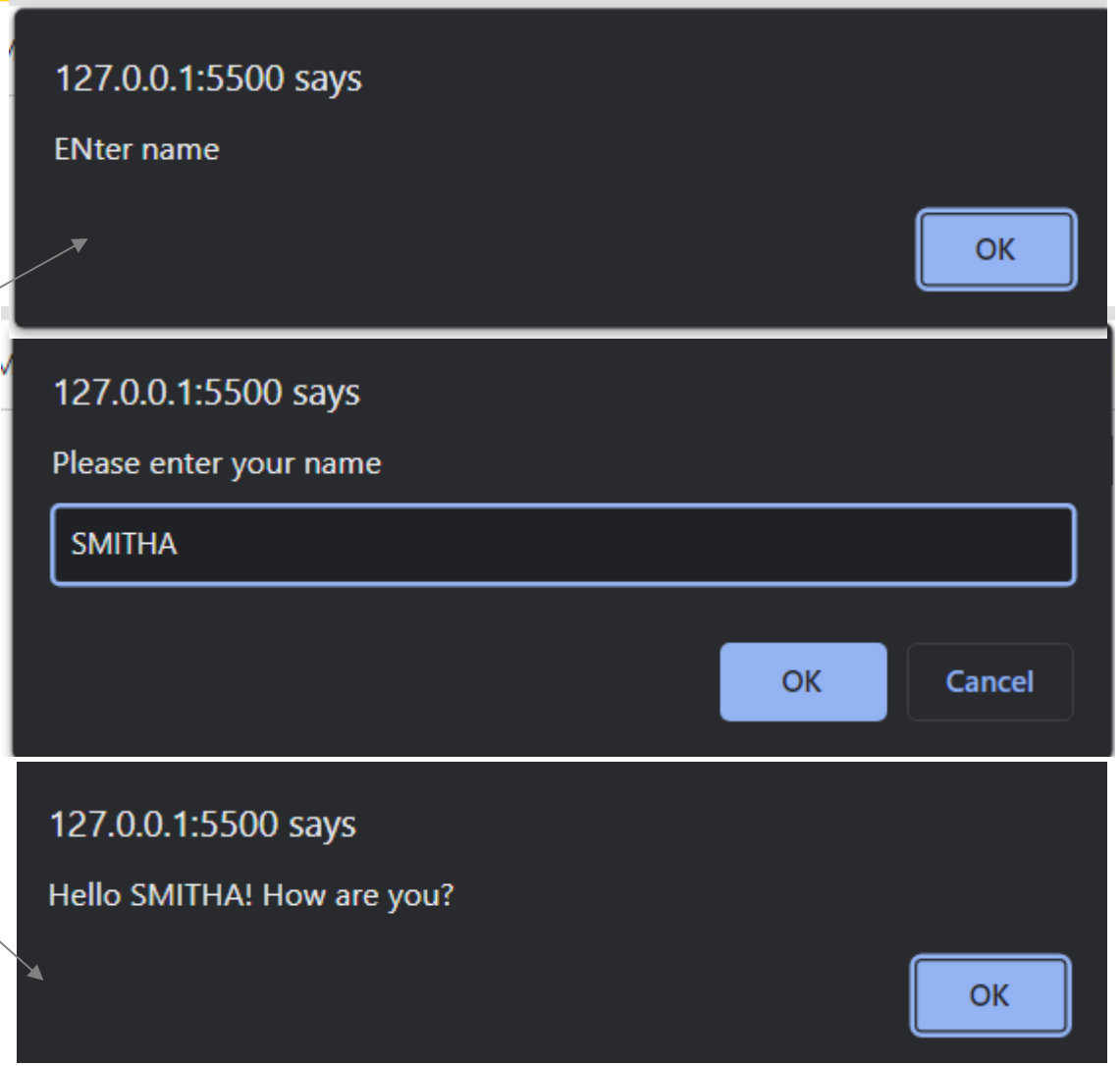
```
<head>
    <title>My Javascript page</title>
    <script type="text/javascript">
        var person = prompt("Please enter your name", "");

            txt = "Hello " + person + "Welcome";

        window.alert(txt);
    </script>
</head>
```





Obtaining User Input with prompt Dialogs

```
<head>
    <title>My Javascript page</title>
    <script type="text/javascript">
        var person = prompt("Please enter your name", "");
        //person returns null when you pressed cancel
        //person returns " " when you failed to provide input
        if (person == null)
        {
            txt = "You pressed cancel";
        }
        else if (person == "")
        { txt = "ENter name"; }
        else
        { txt = "Hello " + person + "! How are you?"; }
        window.alert(txt);

    </script>
</head>
```



Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Arithmetic Operators

JavaScript operation	Arithmetic operator	Algebraic expression	JavaScript expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x/y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Operator(s)	Operation(s)	Order of evaluation (precedence)
*, / or %	Multiplication Division Remainder	Evaluated first. If there are several such operations, they're evaluated from left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several such operations, they're evaluated from left to right.

Precedence of arithmetic operators.

Arithmetic Operators

Operator	Example	Called	Explanation
++	++a	preincrement	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	postincrement	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	predecrement	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	postdecrement	Use the current value of b in the expression in which b resides, then decrement b by 1.

Increment and decrement operators.

Operator Precedence

- JavaScript applies the operators in arithmetic expressions uses the following rules of operator precedence
- Multiplication, division and remainder operations are applied first. If an expression contains several multiplication, division and remainder operations, operators are applied from left to right.
- Multiplication, division and remainder operations are said to have the same level of precedence.
- Addition and subtraction operations are applied next. If an expression contains several addition and subtraction operations, operators are applied from left to right.
- Addition and subtraction operations have the same level of precedence.

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Assignment Operators

Assignment operator	Initial value of variable	Sample expression	Explanation	Assigns
+=	c = 3	c += 7	c = c + 7	10 to c
-=	d = 5	d -= 4	d = d - 4	1 to d
*=	e = 4	e *= 5	e = e * 5	20 to e
/=	f = 6	f /= 3	f = f / 3	2 to f
%=	g = 12	g %= 9	g = g % 9	3 to g

Arithmetic assignment operators.

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Logical Operators

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

Truth table for the || (logical OR) operator.

- The && operator has a higher precedence than the || operator.
- Both operators associate from left to right. An expression containing && or || operators is evaluated only until truth or falsity is known.
- the || operator immediately returns true if the first operand is true. This performance feature for evaluation of logical AND and logical OR expressions is called short-circuit evaluation.

String Concatenation Operator

- JavaScript has a version of the + operator for string concatenation that enables a string and a value of another data type (including another string) to be combined. The result of this operation is a new (and normally longer) string
- String catenation operator +

```
txt1 = “Ben";
txt2 = “Alex";
txt3 = txt1 + " " + txt2;
o/p→ Ben Alex
txt1 = "What a very ";
txt1 += "nice day";
o/p→What a very nice day
```

String Concatenation Operator

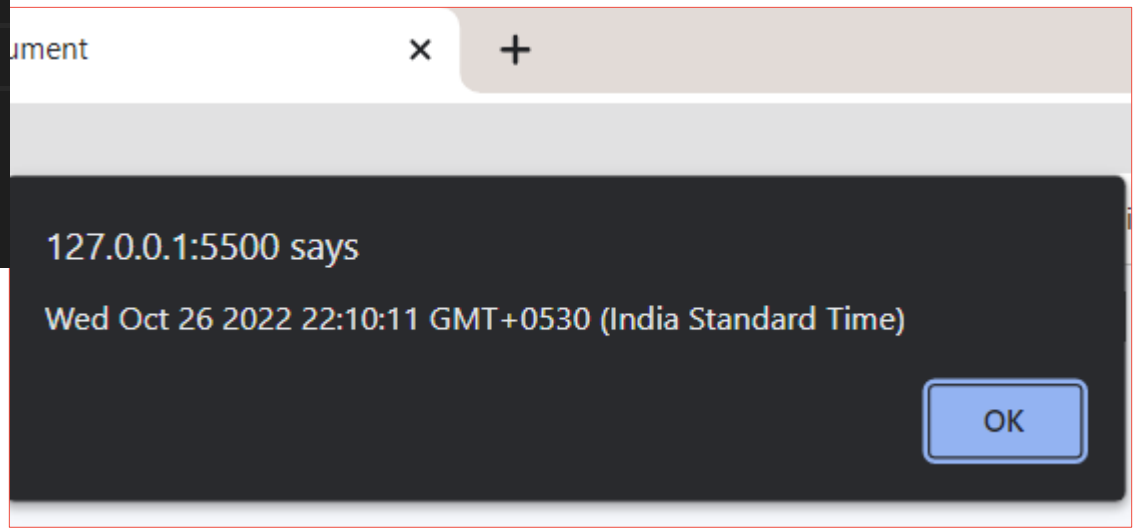
- the + operator used for string concatenation can convert other variable types to strings if necessary.
- Because string concatenation occurs between two strings, JavaScript must convert other variable types to strings before it can proceed with the operation.
- For example, if a variable age has an integer value equal to 21, then the expression "my age is " + age evaluates to the string "my age is 21".
- JavaScript converts the value of age to a string and concatenates it with the existing string literal "my age is ".

Creating and Using a New Date Object

- Create one with the new operator & Date constructor (no params)
- Local time methods of Date:
- toLocaleString– returns a string of the date
- getDate – returns the day of the month
- getMonth – returns the month of the year (0 – 11)
- getDay – returns the day of the week (0 – 6)
- getFullYear– returns the year
- getTime – returns the number of milliseconds since January 1, 1970
- getHours – returns the hour (0 – 23)
- getMinutes– returns the minutes (0 – 59)
- getMilliseconds– returns the milliseconds (0 – 999)

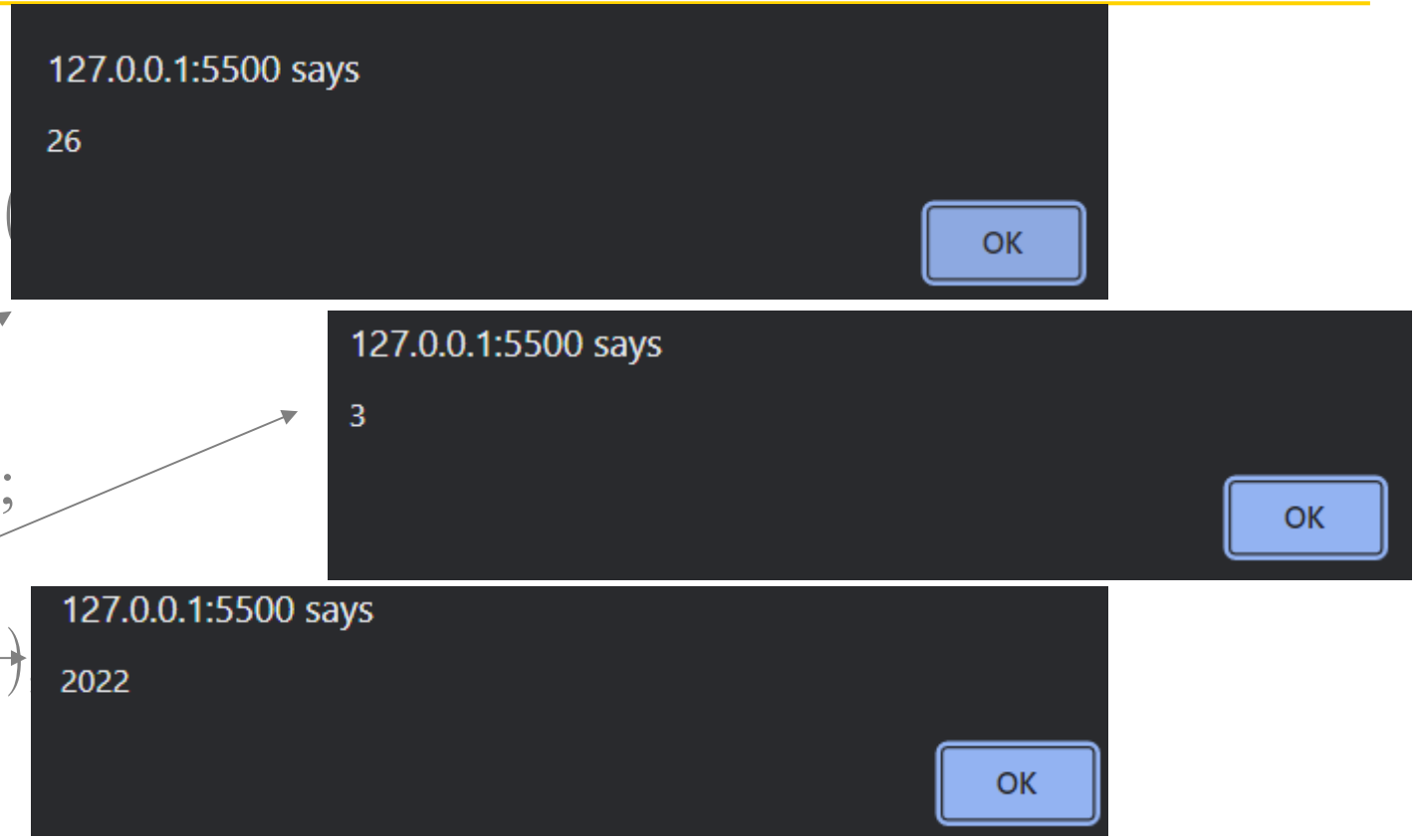
Creating and Using a New Date Object

```
1   <!DOCTYPE html>
2   <html lang="en">
3   > <head>···
8   </head>
9   <body>
10      <script type="text/javascript">
11          var today=new Date();
12          alert(today);
13          </script>
14  </body>
15  </html>
```



Creating and Using a New Date Object

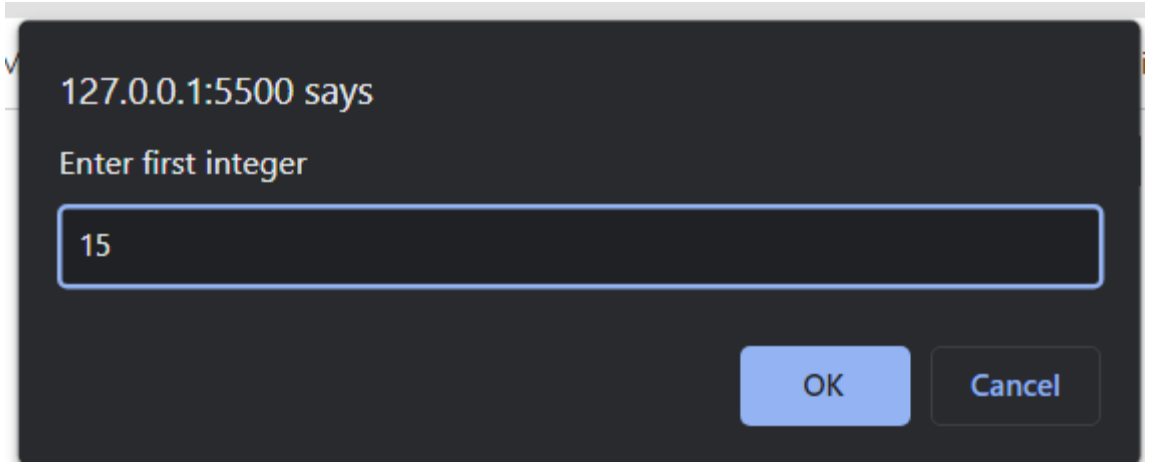
```
<script type="text/javascript">
        var today = new Date(
        alert(today);
        alert(today.getDate());
        alert(today.getMonth());
        alert(today.getDay());
        alert(today.getFullYear())
        alert(today.getTime())
</script>
```

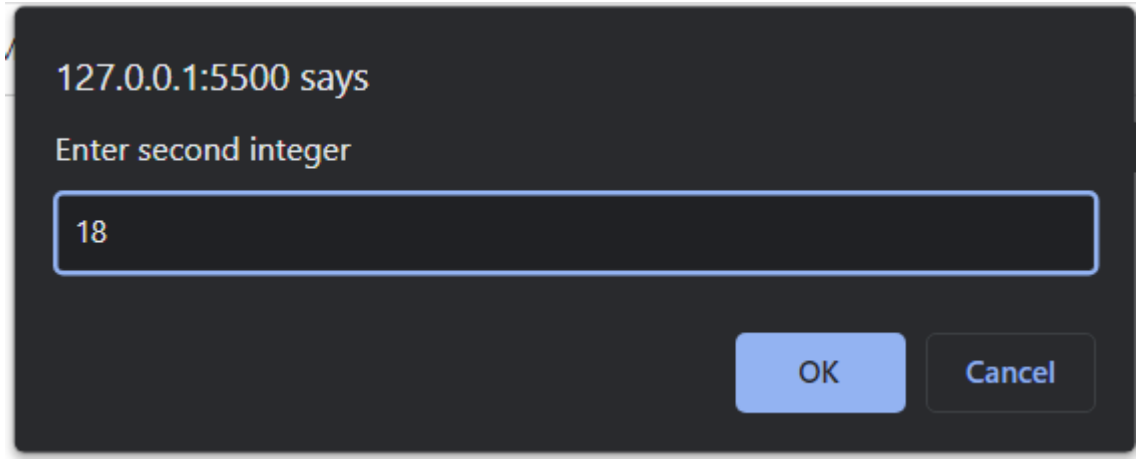


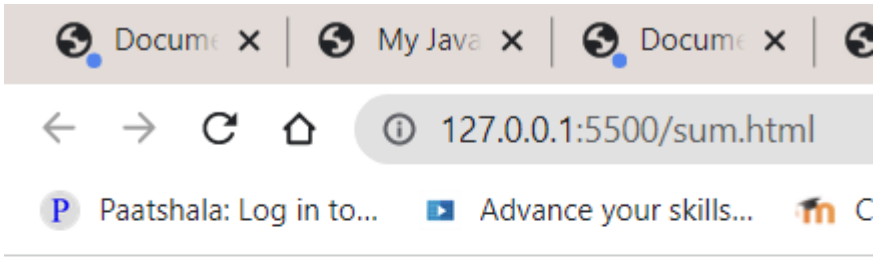
Sum of two numbers

```
<body>
    <h1>Sum of Two numbers</h1>
    <SCRIPT TYPE="text/javascript">
        var firstNumber, secondNumber, number1, number2, sum;
        firstNumber = window.prompt("Enter first integer", "");
        secondNumber = window.prompt("Enter second integer", "");
        // convert numbers from strings to integers
        number1 = parseInt(firstNumber);
        number2 = parseInt(secondNumber);
        sum = number1 + number2;
        document.writeln("<H1>The sum is " + sum + "</H1>");
    </SCRIPT>

</body>
```

Sum of two numbers





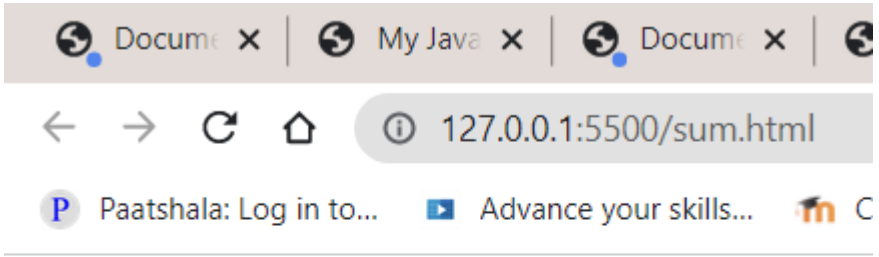


Sum of Two numbers

The sum is 33

Sum of two numbers

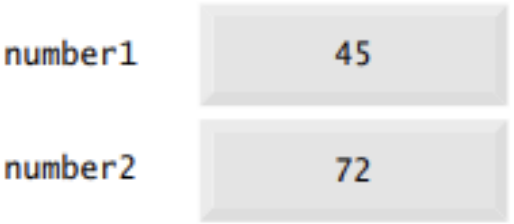
```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>···
8   </head>
9   <body>
10      <h1>Sum of Two numbers</h1>
11      <SCRIPT TYPE="text/javascript">
12          var firstNumber, secondNumber, number1, number2, sum;
13          firstNumber = window.prompt("Enter first integer", "");
14          secondNumber = window.prompt("Enter second integer", "");
15          // convert numbers from strings to integers
16          number1 = Number(firstNumber);
17          number2 = Number(secondNumber);
18          sum = number1 + number2;
19          document.writeln("<H1>The sum is " + sum + "</H1>");
20      </SCRIPT>
21  </body>
22
23  </html>
```



Memory Concepts

- Variable names such as number1, number2 and sum actually correspond to locations in the computer's memory.
- Every variable has a name, a type and a value
- Suppose the user entered the string 45 as the value for firstNumber. The script converts firstNumber to an integer, and the computer places the integer value 45 into location number1
- Whenever a value is placed in a memory location, the value replac the previous value in that locatior

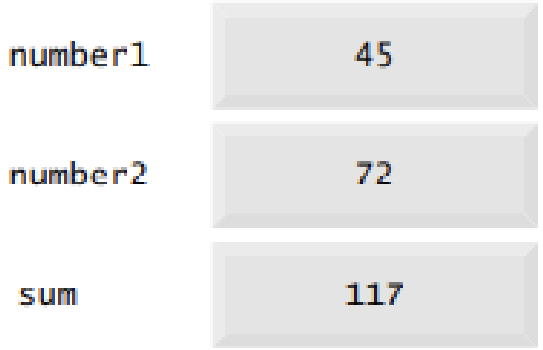
The previous value is lost.



Memory locations after inputting values for variables **number1** and **number2**

Memory Concepts

- Once the script has obtained values for number1 and number2, it adds the values and places the sum into variable sum.
- JavaScript does not require variables to have a declared type before they can be used in a script.
- A variable in JavaScript can contain a value of any data type, and in many situations JavaScript automatically converts between values of different types for you.
- For this reason, JavaScript is referred to as a loosely typed language



Memory locations after calculating the sum of number1 and number2.

Memory Concepts

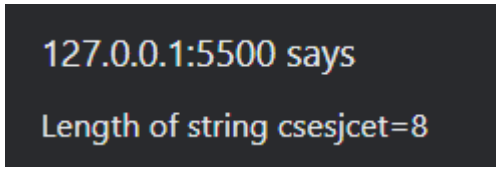
- Unlike its predecessor languages C, C++ and Java, JavaScript does not require variables to have a declared type before they can be used in a script
- When a variable is declared in JavaScript, but is not given a value, the variable has an undefined value.
- Attempting to use the value of such a variable is normally a logic error.
- When variables are declared, they're not assigned values unless you specify them.
- Assigning the value null to a variable indicates that it does not contain a value.

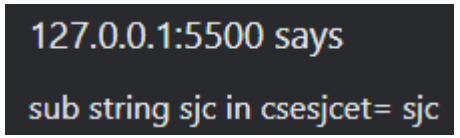
String properties and Methods

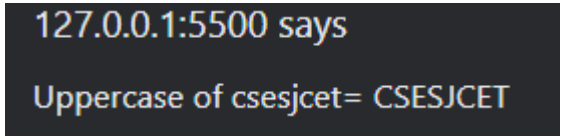
- String properties & methods:
- length e.g., var len = str1.length;
- var str="George";
- var len = str1.length; //len→6
- charAt(position) e.g., str.charAt(3) //r
- indexOf(string) e.g., str.indexOf('o') //2
- substring(from, to) e.g., str.substring(2,4) //org
- toLowerCase() e.g., str.toLowerCase() //george

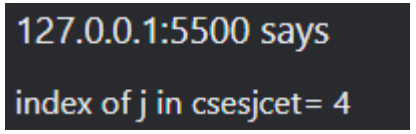
String properties and Methods

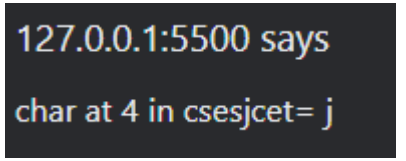
```
<script type="text/javascript">

    var str = "csesjcet";
    alert("Length of string csesjcet=" + str.length);
    alert("sub string sjc in csesjcet= " + str.substring(3, 6));
    alert("Uppercase of csesjcet= " + str.toUpperCase());
    alert("index of j in csesjcet= " + str.indexOf('j'));
    alert("char at 4 in csesjcet= " + str.charAt(4));

</script>
```











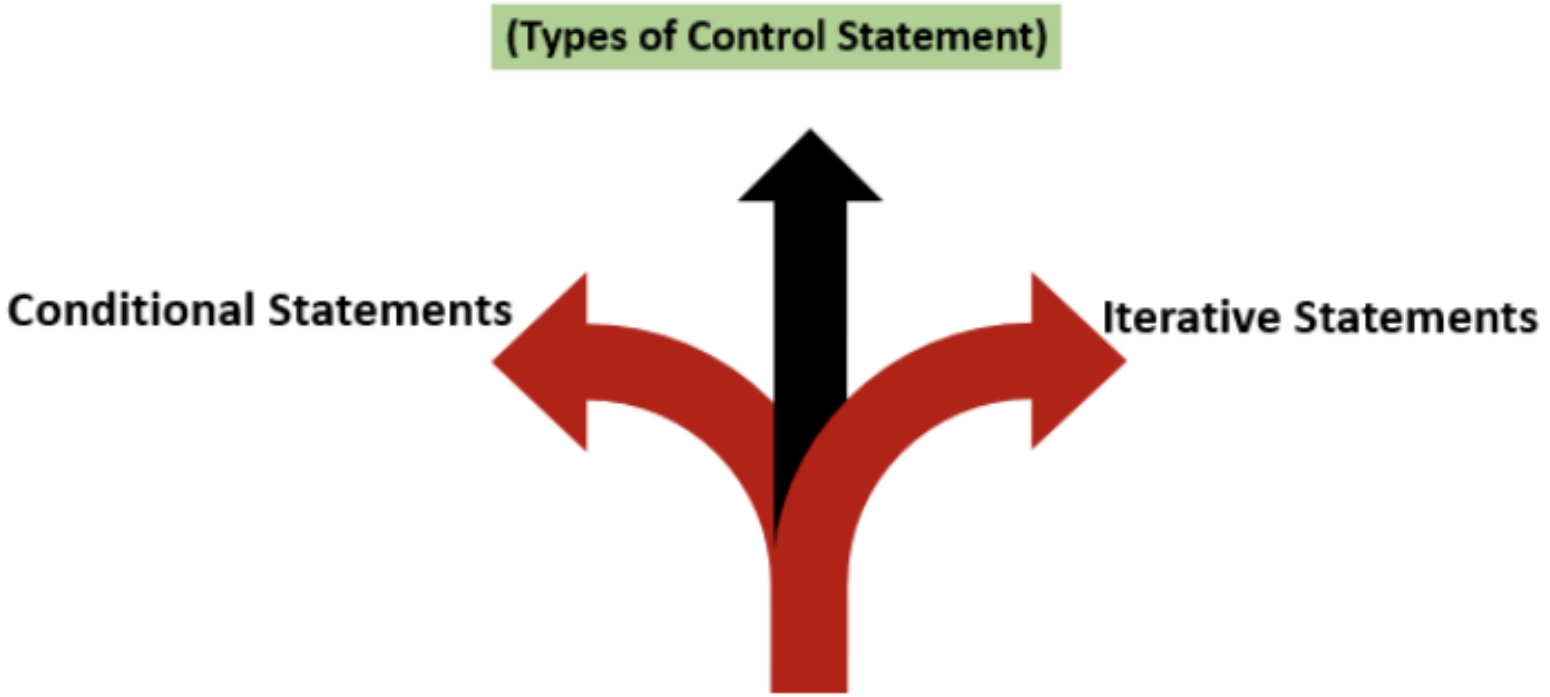
Precedence and Associativity

Operator	Associativity	Type
++ -- !	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== != === !==	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Precedence and associativity of the operators discussed so far.

Control Statements

Control statement



Decision Making

Conditional statement

- if statement - use this statement if you want to execute some code only if a specified condition is true
- if...else statement - use this statement if you want to execute some code if the condition is true and another code if the condition is false
- if...else if....else statement - use this statement if you want to select one of many blocks of code to be executed
- switch statement - use this statement if you want to select one of many blocks of code to be executed .

Conditional statement

- if (condition)
  {
  code to be executed if condition is true
  }
- JavaScript's if statement that allows a script to make a decision based on the truth or falsity of a condition.
- If the condition is met (i.e., the condition is true), the statement in the body of the if statement is executed.
- If the condition is not met (i.e., the condition is false), the statement in the body of the if statement is not executed.
- conditions in if statements can be formed by using the equality operators and relational operators

Equality & Relational Operators

Standard algebraic equality operator or relational operator	JavaScript equality or relational operator	Sample JavaScript condition	Meaning of JavaScript condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

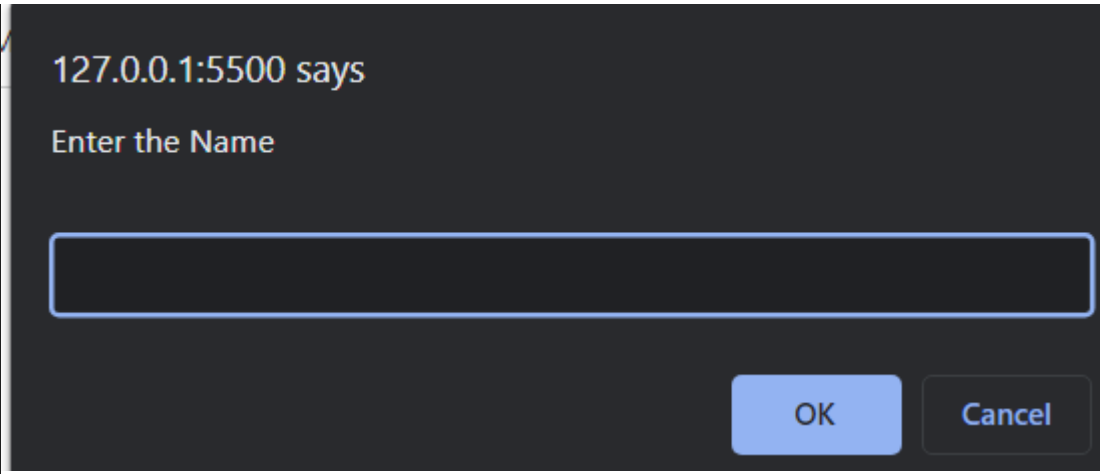
Equality and relational operators.

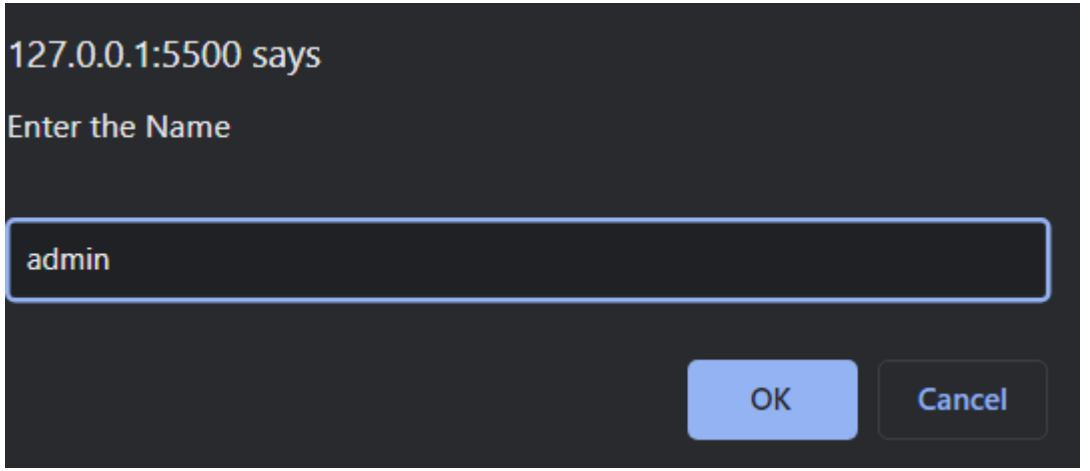
Conditional statement

- if (condition)
  {
  code to be executed if condition is true
  }
- JavaScript's if statement that allows a script to make a decision based on the truth or falsity of a condition.
- If the condition is met (i.e., the condition is true), the statement in the body of the if statement is executed.
- If the condition is not met (i.e., the condition is false), the statement in the body of the if statement is not executed.
- conditions in if statements can be formed by using the equality operators and relational operators

IF statement

```
<script type="text/javascript">
    var x = prompt("Enter the Name \n", "");
    if(x=="admin")
    {
    alert ("Welcome Administrator");
    }
    </script>
```



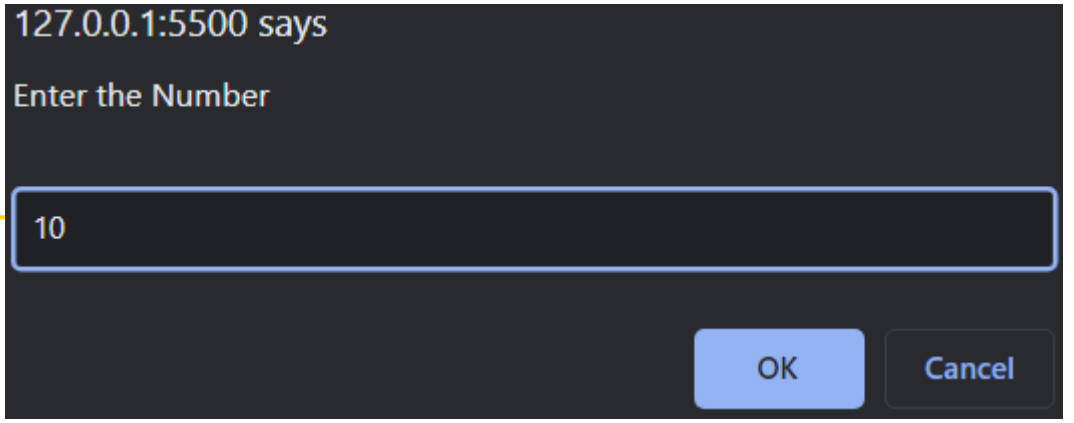


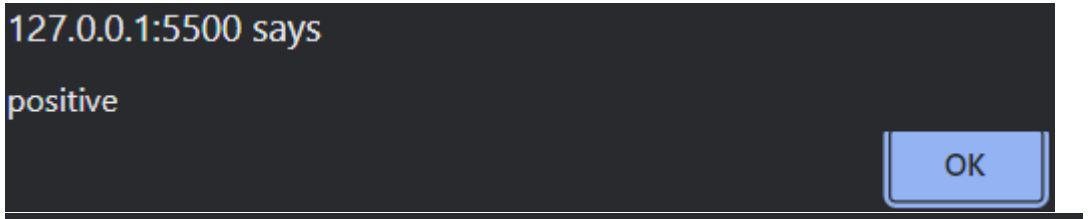
IF ELSE

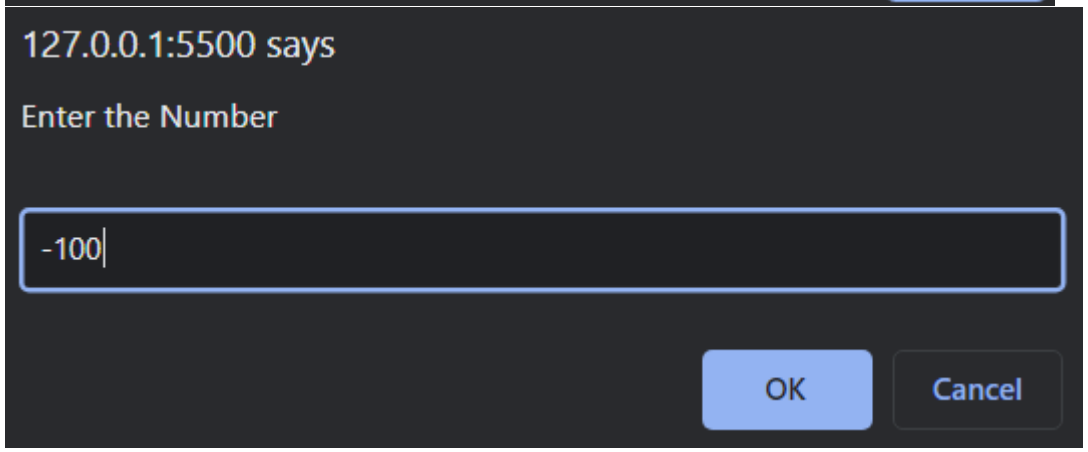
- if (condition)
  { code to be executed if condition is true
  }
  else
  { code to be executed if condition is not true
  }
- JavaScript's if statement that allows a script to make a decision based on the truth or falsity of a condition.
- If the condition is met (i.e., the condition is true), the statement in the body of the if statement is executed.
- If the condition is not met (i.e., the condition is false), the statement in the body of the else block is executed.

IF ELSE

```
<script type="text/javascript">
    var x = prompt("Enter the Number \n", "");
    if (x < 0) {
        alert("negative");
    }
    else {
        alert("positive");
    }
</script>
```







ELSE IF STATEMENT

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

```
if (condition1)
  { block of code to be executed if condition1 is true }
else if (condition2)
  { block of code to be executed if the condition1 is false and condition2 is true }
else
  { block of code to be executed if the condition1 is false and condition2 is false }
```

Conditional Operator (?:)

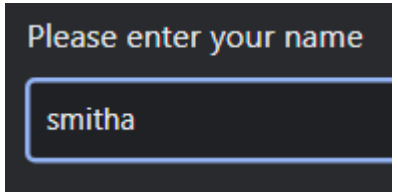
- JavaScript provides an operator, called the conditional operator (?:), that's closely related to the if…else statement.
- The operator ?: is JavaScript's only ternary operator—it takes three operands. The operands together with the ?: form a conditional expression.
- The first operand is a Boolean expression, the second is the value for the conditional expression if the expression evaluates to true and the third is the value for the conditional expression if the expression evaluates to false.

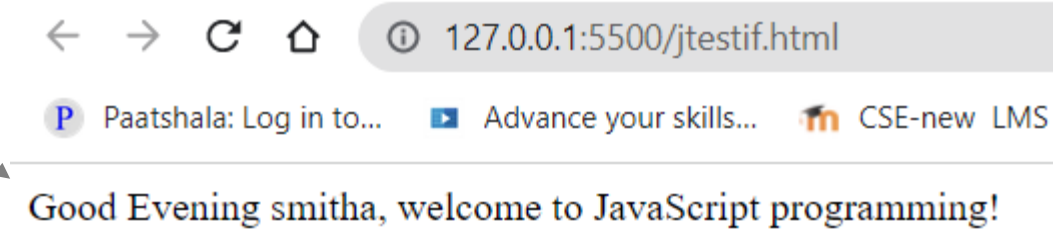
```
document.writeln( studentGrade >= 60 ? "Passed" : "Failed" );
```

- A conditional expression that evaluates to the string "Passed" if the condition studentGrade >= 60 is true and evaluates to the string "Failed" if the condition is false

ELSE IF STATEMENT

```
<script type="text/javascript">
    var name;
    var now = new Date();
    var hour = now.getHours()
    alert(hour);
    name = window.prompt("Please enter your name");
    if (hour < 12) {
        greet = 'Good Morning';

    }
    else if (hour >= 12 && hour <= 17) {
        greet = 'Good Afternoon';

    }
    else if (hour >= 17 && hour <= 24)
        greet = 'Good Evening';
    document.writeln(greet + " " + name + ", welcome to JavaScript programming!");
```





Nested if else

```
If student's grade is greater than or equal to 90
    Print "A"
Else
    If student's grade is greater than or equal to 80
        Print "B"
    Else
        If student's grade is greater than or equal to 70
            Print "C"
        Else
            If student's grade is greater than or equal to 60
                Print "D"
            Else
                Print "F"
```

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

Nested if else

```
<!-- Arranging 3 numbers in order -->
<!doctype html>
<html lang="en">
<head>
 <title>nested if statement</title>
</head>
<body>
 <script language="javascript" type="text/javascript" >
 var a = parseInt(window.prompt("Enter Marks Sub1"));
 var b= parseInt(prompt("Enter Marks Sub2"));
 var c= parseInt(prompt("Enter Marks Sub3"));
 var avg=(a+b+c)/3;
 if(avg >= 70)
    alert("First With Distinction");
 else if(avg >=60 && avg <70)
    alert("First Class");
 else if(avg >=50 && avg <60)
    alert("Second Class");
 else if(avg >=40 && avg <50)
    alert("Third Class");
 else
    alert("Fail");
 </script>
</body>
</html
```

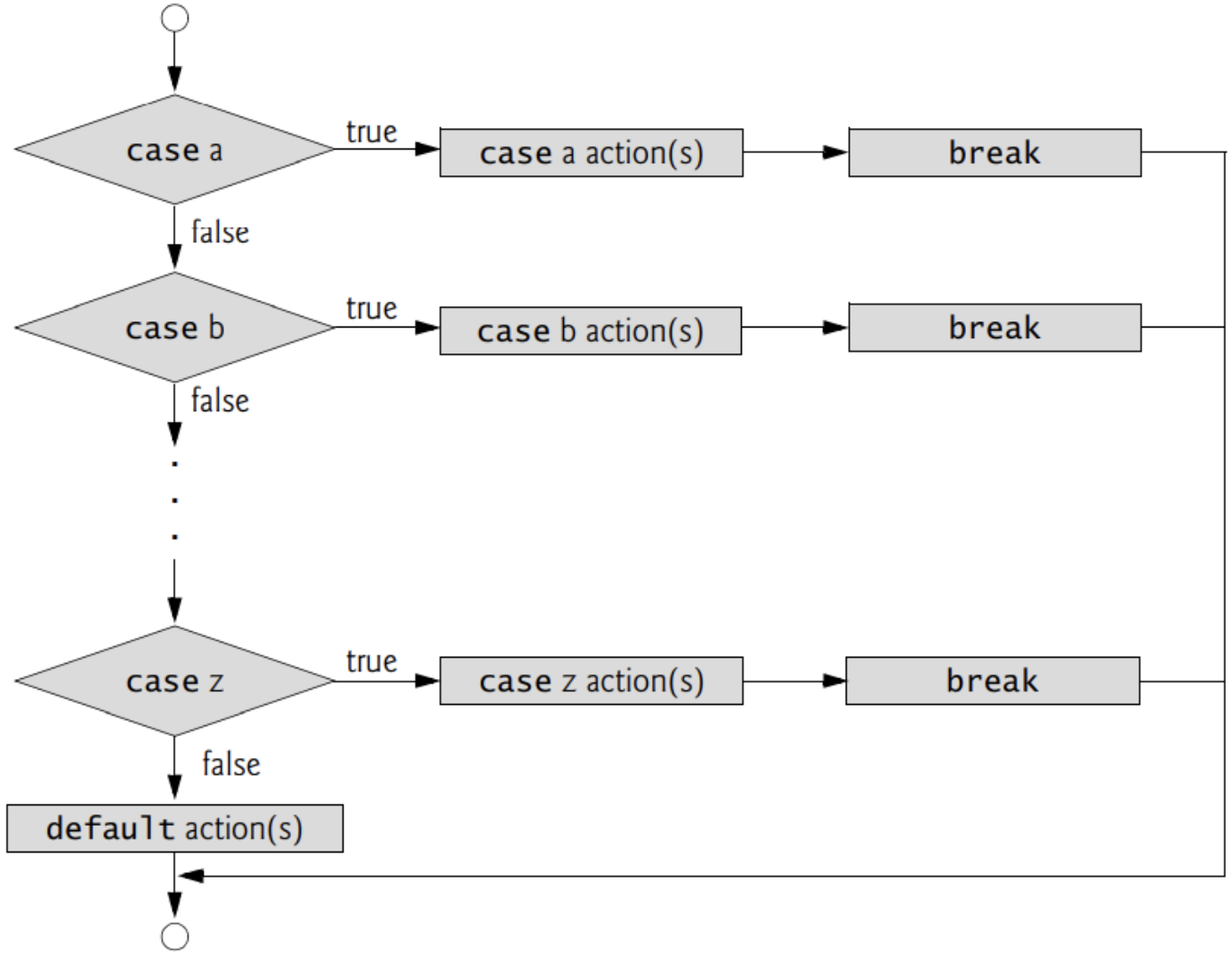
Switch statements

- Use the switch statement to select one of many blocks of code to be executed. Syntax switch(expression)

```
{ case 1: code block
        break; ……
  case n: code block
        break;
  default: code block
}
```

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed

Switch statements

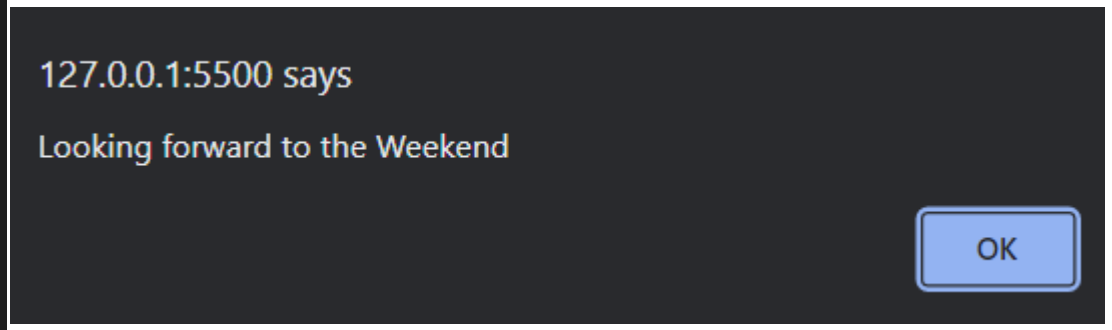


Switch statements


```
<script>
    var day = new Date().getDay();
    switch (day) {
        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";
    }
    alert(day);
</script>
```

Switch statements

```
<script>
    var day = new Date().getDay();
    switch (day) {
    case 4:
    case 5:
    text = "Soon it is Weekend";
    break;
    case 0:
    case 6:
    text = "It is Weekend";
    break;
    default:
    text = "Looking forward to the Weekend";
    }
    alert(text);
    </script>
```



Switch statements

```
<script language="javascript" type="text/javascript">
var n1 = parseInt(window.prompt("Enter the marks in Sub1"));
var n2 = parseInt(window.prompt("Enter the marks in Sub2"));
var n3 = parseInt(window.prompt("Enter the marks in Sub3"));
var avg= (n1+n2+n3)/3;

var ch = parseInt(avg/10); //parse to int

switch(ch)
{
case 10:
case 9:
case 8:
case 7: window.alert("Distinction");break;
case 6: window.alert("First Class");break;
case 5: window.alert("Second Class");break;
case 4: window.alert("Third Class ");break;
default: window.alert("Fail ");
}</script>
```

Loop Statements

Loop statements

- Different Kinds of Loops
- JavaScript supports different kinds of loops:
- for - loops through a block of code a number of times
  - For a loop, when you execute a loop for a fixed number of times, the loop does three specified tasks (Statement 1, Statement 2, Statement 3)
- 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
  - It will execute at least once even if the condition is to be not satisfied. If the condition is true, it will continue executing and once the condition becomes false, it stops the code execution.

For Loop statements

- for (statement 1; statement 2; statement 3)

```
{
code block to be executed
}
```

- Statement 1 is executed before the loop (the code block) starts.
- Statement 2 defines the condition for running the loop (the code block).
- Statement 3 is executed each time after the loop (the code block) has been executed. Example

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

Examples using For Loop statements

a) Vary the control variable from 1 to 100 in increments of 1.

```
for ( var i = 1; i <= 100; ++i )
```

b) Vary the control variable from 100 to 1 in increments of -1 (i.e., *decrements* of 1).

```
for ( var i = 100; i >= 1; --i )
```

c) Vary the control variable from 7 to 77 in steps of 7.

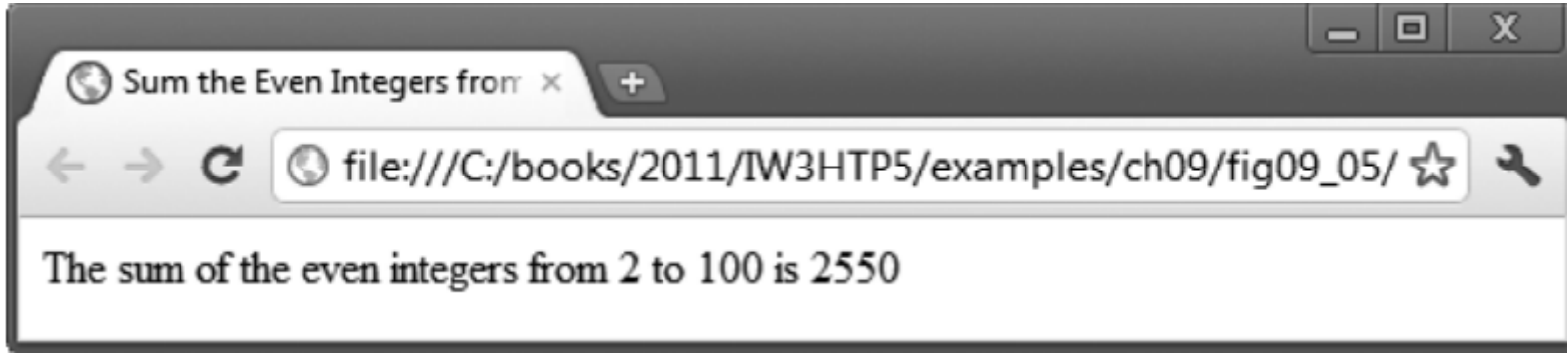
```
for ( var i = 7; i <= 77; i += 7 )
```

d) Vary the control variable from 20 to 2 in steps of -2.

```
for ( var i = 20; i >= 2; i -= 2 )
```

Examples using For Loop statements

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Sum the Even Integers from 2 to 100</title>
      <script>

         var sum = 0;

         for ( var number = 2; number <= 100; number += 2 )
            sum += number;

         document.writeln( "The sum of the even integers " +
            "from 2 to 100 is " + sum );
      </script>
   </head><body></body>
</html>
```



For..in statements

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

```
<script type="text/javascript">
    var x;
    var cars =["Maruthi","Toyoto", "BMW"];
    for (x in cars)
    {
    alert(x+ "="+ cars[x]);
    }
</script>
```

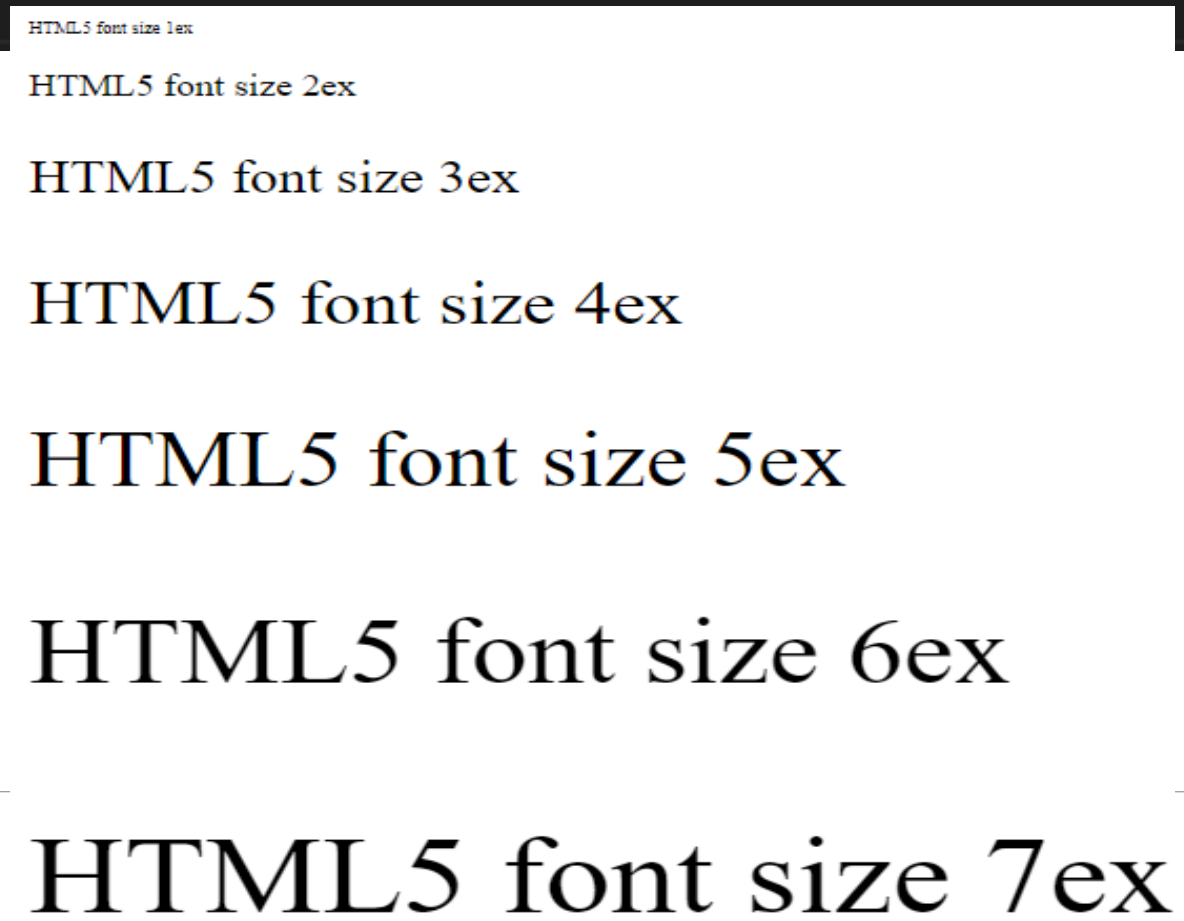
While loop statements

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

  { code block to be executed }

```
<script type="text/javascript">
  var i=0;
  while (i < 10)
  {
  alert("i="+i);
  i++;
  }
</script>
```

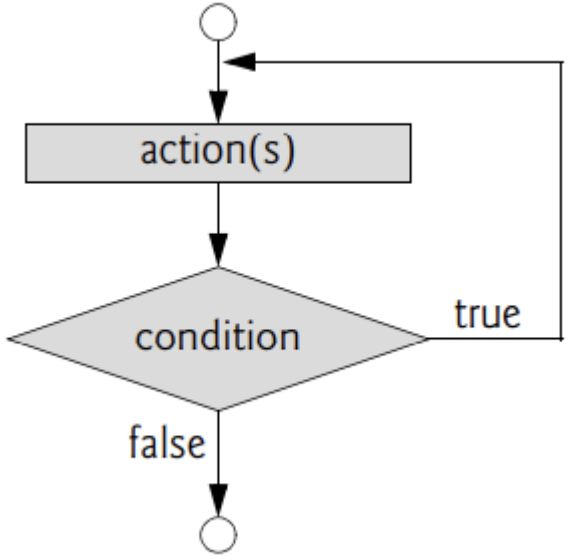
```
<script type="text/javascript">

    var counter = 1;
    // initialization
    while (counter <= 7) // repetition condition
    {
        document.writeln("<p style = 'font-size: " + counter + "ex'>HTML5 font size " + counter + "ex</p>");
        ++counter; // incremen
    }
</script>
```



Do ..While loop statements

- 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.
- Syntax

```
do
{ code block to be executed
}
while (condition);
```




```
<script>
var i=0;
do
{
alert("i="+i);
i++;
}while (i <= 3)
</script>
```

Break statements

- The break statement "jumps out" of a loop.
- It was used to "jump out" of a switch() statement
- break statement also be used to jump out of a loop.
- The break statement breaks the loop and continues executing the code after the loop (if any):
- Example

```
<script type="text/javascript">
var i,text="";
for (i = 0; i < 10; i++)
{
if (i === 3)
        { break; }
text += "The number is " + i;
        alert(text);
}
</script>
```

continue statement

- The continue statement "jumps over" one iteration in the loop.
- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 3

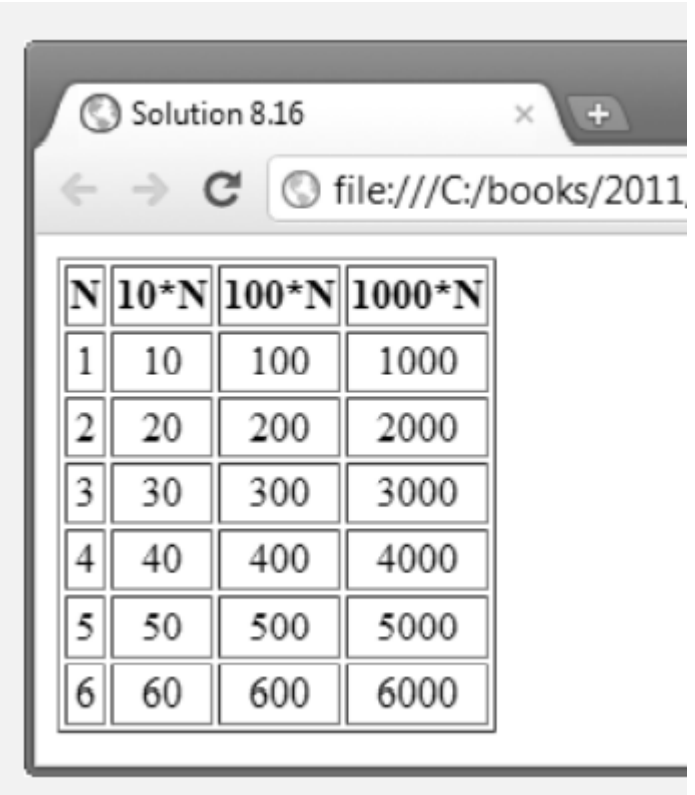
```
<script type="text/javascript">
var i,text="";
for (i = 0; i < 10; i++)
{
        if (i === 3)
         { continue; }
        text += "The number is " + i;
}
        alert(text);
</script>
```

Tutorial Questions

- A palindrome is a number or a text phrase that reads the same backward and forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a script that reads in a five-digit integer and determines whether it's a palindrome. If the number is not five digits long, display an alert dialog indicating the problem to the user. Allow the user to enter a new value after dismissing the alert dialog.
- Develop a script that will determine the gross pay for each of three employees. The company pays "straight time" for the first 40 hours worked by each employee and pays "time and a half" for all hours worked in excess of 40 hours. You're given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your script should input this information for each employee, determine the employee's gross pay and output HTML5 text that displays the employee's gross pay. Use prompt dialogs to input the data.

Tutorial Questions

- Write a script that uses looping to print the following table of values. Output the results in an HTML5 table. Use CSS to center the data in each column.



N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000
6	60	600	6000

Tutorial Questions

```
<!DOCTYPE html>

<html>
   <head>
      <meta charset = "utf-8">
      <title>Mystery Script</title>
      <script type = "text/javascript">
      <!--
         var y;
         var x = 1;
         var total = 0;

         while ( x <= 15 )
         {
            y = x * x * x;
            document.writeln( "<p>" + y + "</p>" );
            total += y;
            ++x;
         } // end while

         document.writeln( "<p>Total is " + total + "</p>" );
         //-->
      </script>
   </head><body></body>
</html>
```

Predict the output??

Functions

Functions

- function function_name([formal_parameters])

  { -- body – }

- Return value is the parameter of return ,If there is no return, or if the end of the function is reached, undefined is returned .If return has no parameter, undefined is returned
- Function call -Functions are invoked by writing the name of the function, followed by a left parenthesis, followed by a comma-separated list of zero or more arguments, followed by a right parenthesis
- If fun is the name of a function,

```
var a=fun();
    or
fun();      /* A call to fun */
```

Functions-example

```
// function returns a value
<script>
function myFunction(a, b)
 {
    return a * b;
}
var x = myFunction(4, 3);
alert(x);
</script>
```

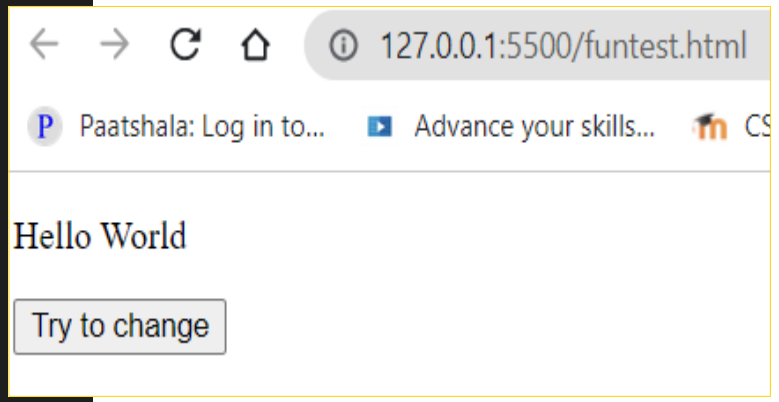
```
//invoking a function as a
function
<script>
function myFunction(a, b)
                {
            return a * b;
                }
alert(myFunction(10, 2));
</script>
```

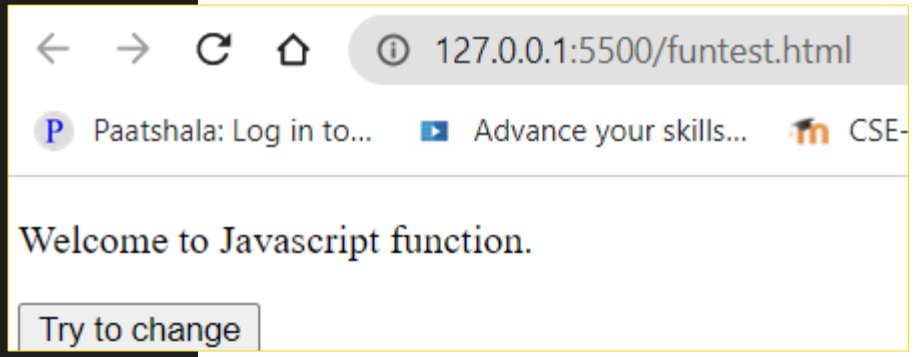
Functions

- The parameter values that appear in a call to a function are called actual parameters.
- The parameter names that appear in the header of a function definition, which correspond to the actual parameters in calls to the function, are called formal parameters.
- JavaScript uses the pass-by-value parameter-passing method.
- When a function is called, the values of the actual parameters specified in the call are copied into their corresponding formal parameters, which behave exactly like local variables.

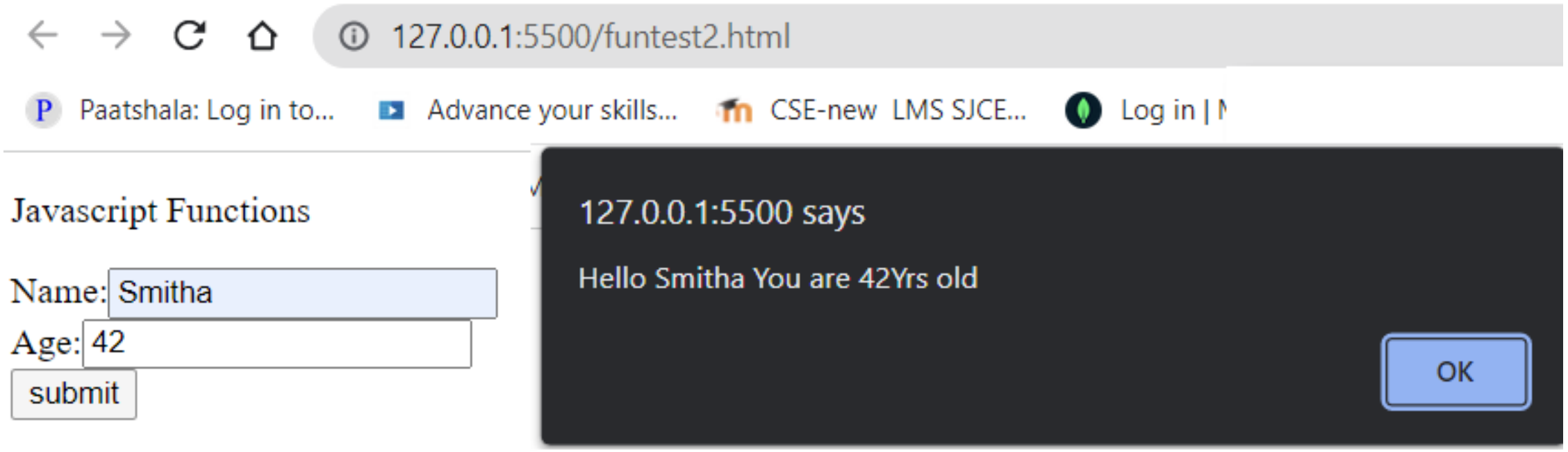
Functions

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function test() {
            document.getElementById("hello").innerHTML = "Welcome to Javascript function.";
        }
    </script>
</head>
<body>
    <p id="hello">Hello World</p>
    <input type="button" onclick="test()" value="Try to change" />
</body>
</html>
```





Functions


127.0.0.1:5500/funtest2.html
Paatshala: Log in to...
Advance your skills...
CSE-new LMS SJCE...
Log in | M
Javascript Functions
Name: Smitha
Age: 42
submit
127.0.0.1:5500 says
Hello Smitha You are 42Yrs old
OK

Functions

```
<body>
    <p>Javascript Functions</p>
    <form name="form1" method="post" action="funtest2.html" >
        Name:<input type="text" name="fname" />
        <br />Age:<input type="text" name="age" />
        <br /><input type="submit" name="submit" value="submit" onclick="greeting()" />
        <br />
    </form>
    <script type="text/javascript">
        function greeting()
        {//document.getElementById("fname").value or var
            var name = document.form1.fname.value;
             var age = document.form1.age.value;
              alert('Hello ' + name + " You are " + age + "Yrs old");
              document.write('Hello ' + name + " You are " + age + "Yrs old")
        }
    </script>
</body>
```

Functions

- Scripts can also be placed in external files:
- External file:// myScript.js

```
function test()
{
document.getElementById("demo").innerHTML = “welcome to Javascript.";
}
```

- 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 and embed in the html doc:

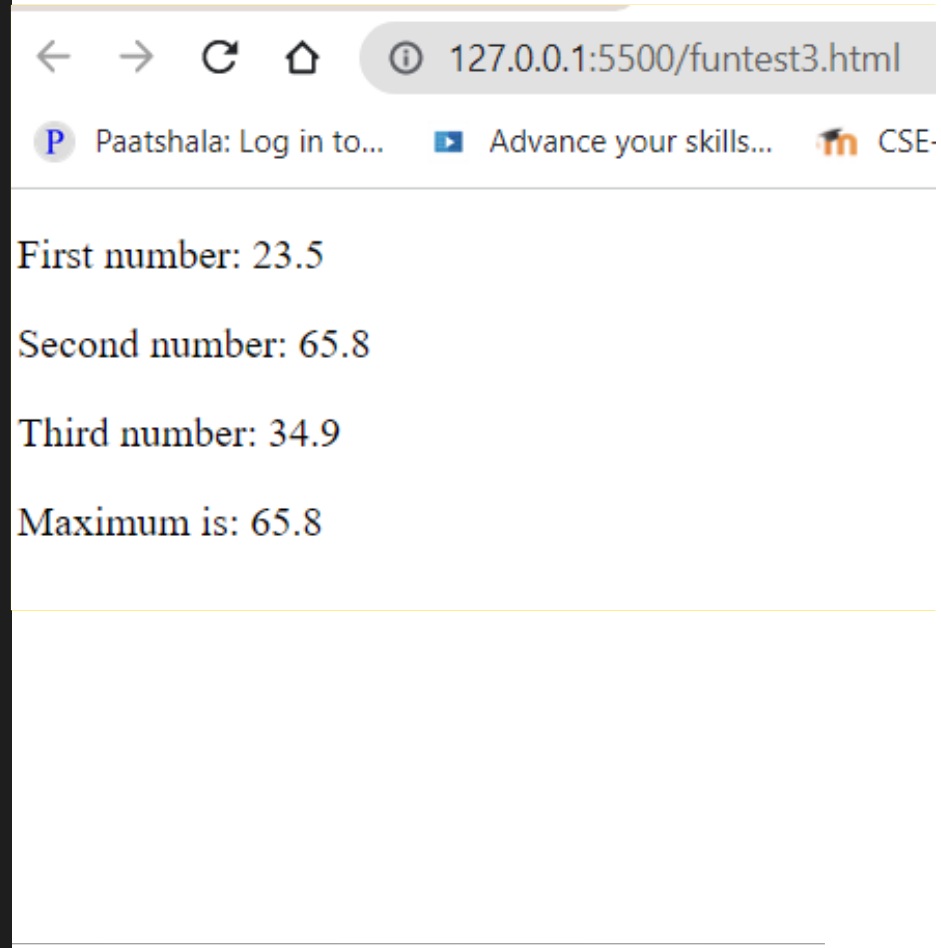
```
<script src="myScript.js"></script>
```

Compute the roots of a quadratic equation - example

```
//If the roots are imaginary, this script displays NaN
var a = prompt("Find the solution for ax^2 + bx +c \n What is the value of 'a'? \n", "");
var b = prompt("What is the value of 'b'? \n", "");
var c = prompt("What is the value of 'c'? \n", "");
document.write("a="+ a+"<br />");
document.write("b="+b+"<br />");
document.write("c="+c+"<br />");
// Compute the square root and denominator of the result
var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;
var root1 = (-b + root_part) / denom;
var root2 = (-b - root_part) / denom;
document.write("The first root is: ", root1, "<br />");
document.write("The second root is: ", root2, "<br />");
```

Example-Maximum of three numbers using user defined function

```
<body>
    <script type="text/javascript">

      var no1 = window.prompt("No 1","");
      var no2 = window.prompt("No 2","");
      var no3 = window.prompt("No 3","");
      var x=parseFloat(no1);
      var y=parseFloat(no2);
      var z=parseFloat(no3);
      var maxValue = maximum( x, y, z);
          document.writeln( "<p>First number: " + x + "</p>" +
          "<p>Second number: " + y + "</p>" +"<p>Third number: " + z + "</p>"
          +  "<p>Maximum is: " + maxValue + "</p>" )
    function maximum( x, y, z )
    {
     return Math.max( x, Math.max( y, z ) );
    }
</script>
</body>
```



Example-Random Number Generation

- Consider the following statement:

```
var randomValue = Math.random();
```

- Method random generates a floating-point value from 0.0 up to, but not including, 1.0.
- If random truly produces values at random, then every value in that range has an equal chance (or probability) of being chosen each time random is called.
- Math.floor determine the closest integer not greater than the argument's value—for example, Math.floor(1.75) is 1 and Math.floor(6.75) is 6

```
Math.floor( 1 + Math.random() * 6 )
```

- The preceding expression multiplies the result of a call to Math.random() by 6 to produce a value from 0.0 up to, but not including, 6.0. This is called scaling the range of the random numbers.

Example-Random Number Generation

```
<script type="text/javascript">
    var value;
    document.writeln("<p>Random Numbers</p><ol>");
    for (var i = 1; i <= 10; ++i) {
        value = Math.floor(1 + Math.random() * 6)

        document.writeln("<li>" + value + "</li>");
    } // end for
    document.writeln("</ol>");
</script>
```

Random Numbers

1. 3
2. 1
3. 4
4. 4
5. 2
6. 2
7. 5
8. 2
9. 1
10. 3

Example-Linear search

```
<script type="text/javascript">
    function Search()
    {  var a = new Array(50); // create an array
       // fill array with even integer values from 0 to 48
       for (var i = 0; i < a.length; ++i) {
        a[i] = 2 * i;
        document.getElementById("list1").innerHTML+=a[i]+" ";
       }
       var inputVal = document.getElementById("inputVal");
        var searchKey = parseInt(inputVal.value);
        var element = a.indexOf(searchKey);
        //The indexOf() method returns the first index (position) of a specified value.
        // get the result paragraph
        var result = document.getElementById("result");
        // get the search key from the input text field then perform the search
        if (element != -1) {
            result.innerHTML = "Found value in element " + element;
        } // end if
        else {
            result.innerHTML = "Value not found";
        }
      }  </script></head><body>
<form action="#" onsubmit="Search()">
    <label>Enter integer search key:
        <input id="inputVal" type="number"></label>
        <input id="searchButton" type="submit" value="Search">
        <p id="list1"></p>
    <p id="result"></p>
</form></body></html>
```

Enter integer search key: 24 Search

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42

44 46 48 50 52 54 56 58 60 62 64 66 68 70 72

74 76 78 80 82 84 86 88 90 92 94 96 98

Found value in element 12

Global functions

- JavaScript provides several standard global functions that are commonly used in programming.
- These functions are part of the global JavaScript object and can be used without the need for importing or including additional libraries
- The Global object contains all the global variables in the script, all the user-defined functions in the script, and all the functions.
- Because global functions and user-defined functions are part of the Global object, some JavaScript programmers refer to these functions as methods.
- You do not need to use the Global object directly—JavaScript references it for you

Global functions

- parseInt(): Converts a string to an integer and returns the integer value.
- parseFloat(): Converts a string to a floating-point number and returns the number.
- isNaN(): Checks if a value is NaN (Not-a-Number).
- isFinite(): Checks if a value is a finite number.
- decodeURI() and encodeURI(): These functions are used for decoding and encoding Uniform Resource Identifiers (URIs), respectively.
- decodeURIComponent() and encodeURIComponent(): These functions are used for decoding and encoding URI components.
- eval(): Evaluates a string as JavaScript code.
- alert(), prompt(), and confirm(): Functions for interacting with the user via pop-up dialog boxes.

Global functions

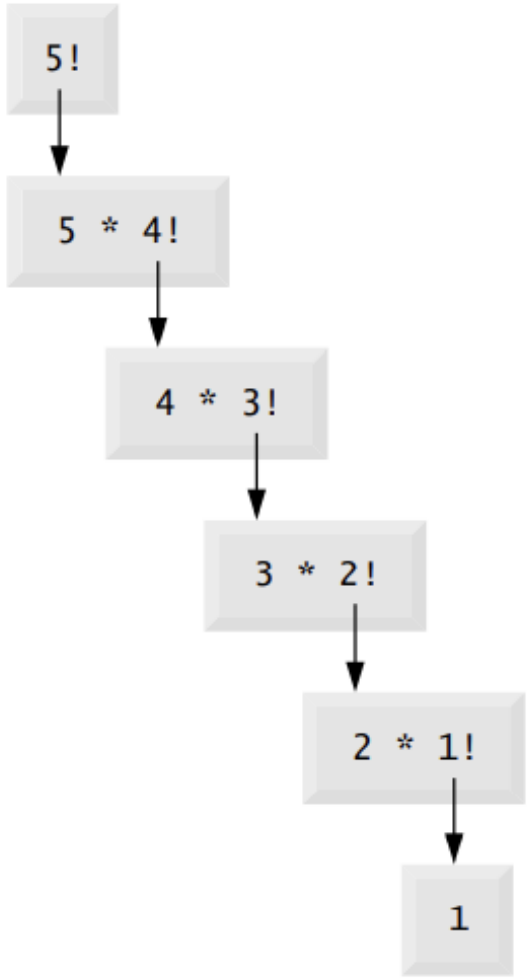
Global function	Description
`isFinite`	Takes a numeric argument and returns `true` if the value of the argument is not `NaN`, `Number.POSITIVE_INFINITY` or `Number.NEGATIVE_INFINITY` (values that are not numbers or numbers outside the range that JavaScript supports)—otherwise, the function returns `false`.
`isNaN`	Takes a numeric argument and returns `true` if the value of the argument is not a number; otherwise, it returns `false`. The function is commonly used with the return value of `parseInt` or `parseFloat` to determine whether the result is a proper numeric value.
`parseFloat`	Takes a string argument and attempts to convert the *beginning* of the string into a floating-point value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (e.g., `parseFloat( "abc123.45" )` returns `NaN`, and `parseFloat( "123.45abc" )` returns the value `123.45`).
`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`). This function takes an optional second argument, from 2 to 36, specifying the **radix** (or **base**) of the number. Base 2 indicates that the first argument string is in **binary** format, base 8 that it's in **octal** format and base 16 that it's in **hexadecimal** format. See Appendix E, for more information on binary, octal and hexadecimal numbers.

```
<body >
    <script type="text/javascript">
    function printme()
     {
        var numericString = "42";
        var parsedIntValue = parseInt(numericString);
        alert("Parsed Integer: " + parsedIntValue);
        var floatString = "3.14";
        var parsedFloatValue = parseFloat(floatString);
        alert("Parsed Float: " + parsedFloatValue);
        var notANumber = "I am not a Number";
        var numericValue = 42;
        alert("Is NaN: " + isNaN(notANumber));
        // This will show a pop-up alert with "Is NaN: true"
        alert("Is NaN: " + isNaN(numericValue));
        // This will show a pop-up alert with "Is NaN: false"
    }
</script>
<input type="button" name="button" value="Print Me" onclick="printme()"/>
</body>
```

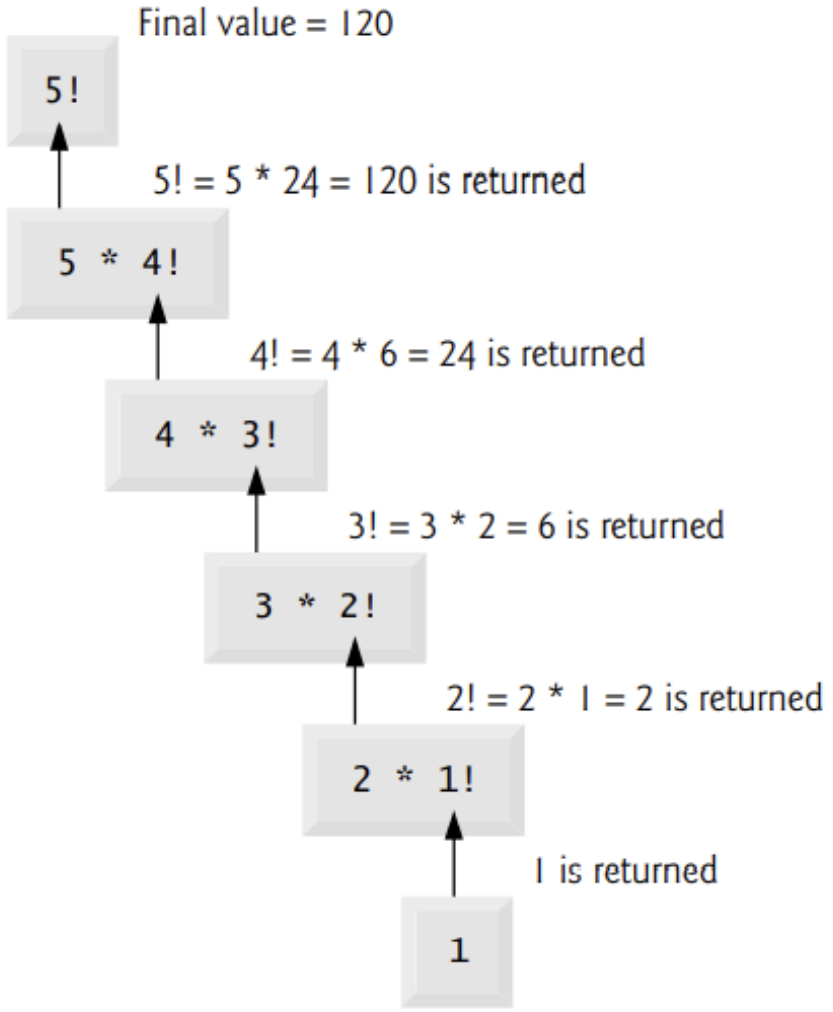
Recursion

- A recursive function is a function that calls itself, either directly, or indirectly through another function.
- The recursion step executes while the original call to the function is still open (i.e., it has not finished executing).
- The recursion step can result in many more recursive calls as the function divides each new subproblem into two conceptual pieces.
- For the recursion eventually to terminate, each time the function calls itself with a simpler version of the original problem, the sequence of smaller and smaller problems must converge on the base case.

Recursive Function -Factorial



(a) Sequence of recursive calls.

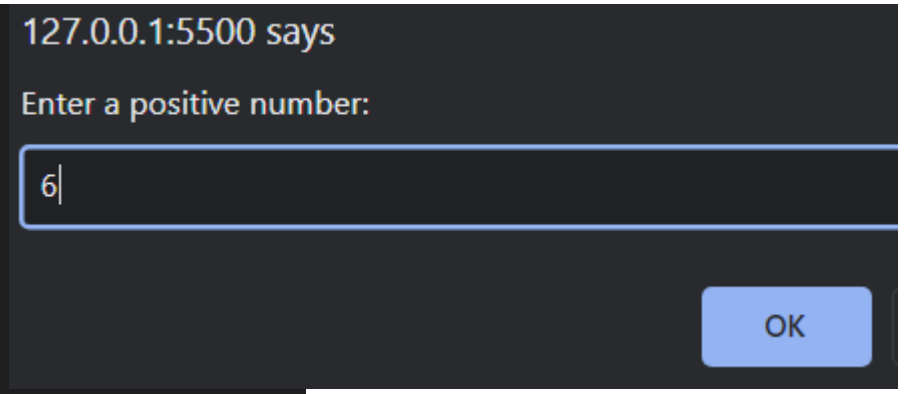


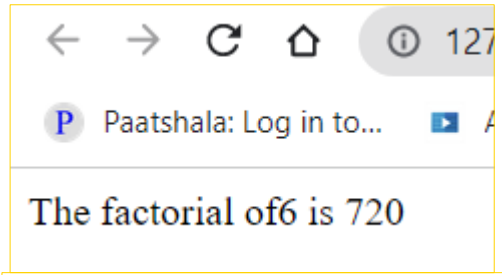
(b) Values returned from each recursive call.

Recursive evaluation of 5!.

Recursive Function -Factorial

```
12      <script type="text/javascript">
13          function factorial(x) {
14              // if number is 0
15              if (x == 0) {
16                  return 1;
17              }
18              // if number is positive
19              else {
20                  return x * factorial(x - 1);
21              }
22          }
23          const num = prompt('Enter a positive number: ');
24          if (num >= 0) {
25              const result = factorial(num);
26              document.write("The factorial of" + num + " is " + result);
27          }
28      </script>
```





var and let create variables that can be reassigned to another value. Const creates “constant” variables that cannot be reassigned to another value

Recursion Vs Iteration

- Both iteration and recursion are based on a control statement:
  - Iteration uses a repetition statement (e.g., for, while or do…while);
  - recursion uses a selection statement (e.g., if, if…else or switch).
- Both iteration and recursion involve repetition:
  - Iteration explicitly uses a repetition statement;
  - recursion achieves repetition through repeated function calls.
- Iteration and recursion each involve a termination test:
  - Iteration terminates when the loop-continuation condition fails;
  - recursion terminates when a base case is recognized.
- Both iteration and recursion can occur infinitely:
  - An infinite loop occurs with iteration if the loop-continuation test never becomes false;
  - infinite recursion occurs if the recursion step does not reduce the problem each time via a sequence that converges on the base case or if the base case is incorrect.

ARRAYS

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.
- An array is a group of memory locations that all have the same name and normally are of the same type (although this attribute is not required in JavaScript).
- To refer to a particular location or element in the array, we specify the name of the array and the position number of the particular element in the array
- The first element in every array is the zeroth element.
- Thus, the first element of array c is referred to as c[0], the second element as c[1].
- The position number in square brackets is called an index and must be an integer or an integer expression.
- If a program uses an expression as an index, then the expression is evaluated to determine the value of the index.

ARRAYS

- JavaScript arrays are used to store multiple values in a single variable.
- Syntax: **var array_name = [item1, item2, ...];**
- Example: var cars = [“Maruthi",“Toyoto","BMW"];
- An array can hold many values under a single name,
- Access the values of array by referring to an index number.
- we can create a JavaScript Array Using an array literal.

  Syntax: **var array_name = new Array(item1, item2, ...);**
- The following example also creates an Array, and assigns values to it:
- Example

```
var s = new Array(“AA", ”BB",“CC”);
var list = new Array(2, 4, 6, 8);
```

ARRAYS

```
var n = [ 10, 20, 30, 40, 50 ];
```

- Uses a comma-separated initializer list enclosed in square brackets ([and]) to create a five element array with indices of 0, 1, 2, 3 and 4.
- The array size is determined by the number of values in the initializer list.

```
var n = new Array( 10, 20, 30, 40, 50 );
```

- Also creates a five-element array with indices of 0, 1, 2, 3 and 4.
- In this case, the initial values of the array elements are specified as arguments in the parentheses following new Array.
- The size of the array is determined by the number of values in parentheses

```
var n = [ 10, 20, , 40, 50 ];
```

- creates a five-element array in which the third element (n[2]) has the value undefined

Accessing elements of ARRAYS

- An array element can be accessed by referring to the index number.
- To access the value of the first element in s:

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

```
<script>
    var s = ["AA", "BB", "CC"];
    alert(s[2]+" "+s[1]+" "+s[0]);
</script>
```

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

```
var s = new Array("AA", "BB","CC");
alert(s.length);
//return 3
```

Example of function and array

```
<body>
    <script type="text/javascript">
     function printme( unknown )
     {   document.writeln("<br/>we got ", unknown);
         }
         var array1 = ["cat", "mouse", "dog"];
         document.writeln("<br/>Array1= ", array1);
         var array2 = ["bird", "fish"];
         document.writeln("<br/>Array2= ", array2);
         var x = 1;
         printme (array1);
         printme (array2[1]);
         printme (x);
         array1[x] = "lion";
         printme (array1);
    </script>
</body>
```

Array1= cat,mouse,dog
Array2= bird,fish
we got cat,mouse,dog
we got fish
we got 1
we got cat,lion,dog

ARRAY properties and methods

- Adding Array Elements
  - To add a new element to an array is using the push method: •

s.push("XX");

```
<script type="text/javascript">

    var s = ["AA", "BB", "CC"];
      s.push("XX");
      for (i = 0; i < s.length; i++) {
      alert(s[i])
    }

</script>
```

ARRAY properties and methods

- Adding Array Elements
- New element can also be added to an array using the length property:

```
var s = ["BB", "OO", "AA", "MM"];
s[s.length] = "LL"; //add as last element
```

- Adding elements with high indexes can create undefined "holes" in an array:

```
var s = ["BB", "OO", "AA", "MM"];
s[6] = "PP“
```

```
<script>
var s = ["AA","BB","CC"];
alert(s[0]+" "+s[1]+" "+s[2]);
alert(s.length);
s[s.length] = "LL";
s[6]="XX"
for (x in s)
 {
alert(“Element”+x+ "="+ s[x]);
}
</script>
```

ARRAY properties and methods

- Popping Array Elements
- The pop() method removes the last element from an array:
- Example

```
var s = ["AA", "BB", "CC"];
s.pop();
// Removes the last element (“CC") from array s
```

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

```
var s = ["BB","OO","AA",“MM"];
var x = s.pop();
// the value of x is "MM"
```

ARRAY properties and methods

- Shifting is equivalent to popping, working on the first element instead of the last.
- The shift() method removes the first array element and "shifts" all other elements to a lower index.
- Example

```
var s = ["BB", "OO", "AA", “MM"];
 s.shift();
 // Removes the first element "BB" from ‘ s’
```

- The shift() method returns the string that was "shifted out":

Example

```
var s = ["BB", "OO", "AA", “MM"];
 alert(s.shift());  //dislays "BB"
```

ARRAY properties and methods

- The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:
- Example

```
var s = ["BB", "OO", "AA", “MM"];
s.unshift(“PP"); // Adds a new element “PP" to s
```

- The unshift() method returns the new array length.
- Example

```
var s = ["BB", "OO", "AA", “MM"];
alert(s.unshift(“PP");
// Returns 5
```

```
<script>
var s = ["BB", "OO", "AA", "MM"];
s.unshift("PP");
//add PP to s in the first position
for (x in s)
 {
alert("Element"+x+ "="+ s[x]);
}
</script>
```

ARRAY properties and methods

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

```
var s = ["BB", "OO", "AA",“MM"];
delete s[0];
// Changes the first element in s to undefined
```

- Using delete may leave undefined holes in the array. Use pop() or shift() instead.

ARRAY properties and methods

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

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

- 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.

ARRAY properties and methods

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
for (x in fruits)
 {
document.write(fruits[x]+"<br/>");
}
</script>
```

Banana
Orange
Lemon
Kiwi
Apple
Mango

ARRAY properties and methods

- Merging an Array
- concat() method creates a new array by merging (concatenating) existing arrays:
- Example (Merging Two Arrays)

```
var girls = [“ammu",“annu"];
var boys = ["Emil",“richu","Linu"];
var mystds = girls.concat(boys);          //Concatenates (joins) girls and boys
```

- The concat() method does not change the existing arrays. It always returns a new array.The concat() method can take any number of array arguments:
- Example (Merging Three Arrays)

```
var arr1 = ["cilie","Lona"]; var arr2 = ["Emil","Tobias","Linus"];
var arr3 = ["Robin","Morgan"];
var myChildren = arr1.concat(arr2, arr3);          // Concatenates arr1 with arr2 and arr3
```

ARRAY properties and methods

- The concat() method can also take values as arguments:
- Example (Merging an Array with Values)

```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "asi", "Linu"]);
```

- slice() method slices out a piece of an array into a new array.
- slices out a part of an array starting from array element 1 ("Orange"):

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

- slice() method creates a new array. It does not remove any elements from the source array.

ARRAY properties and methods

- This example slices out a part of an array starting from array element 3 ("Apple"):

```
var fruits = ["Banana","Orange","Lemon","Apple","Mango"];
var citrus = fruits.slice(3);
```

- The slice() method can take two arguments like slice(1, 3).
- The method then selects elements from the start argument, and up to (but not including) the end argument.

```
var fruits = ["Banana","Orange","Lemon","Apple","Mango"];
var citrus = fruits.slice(1,3);
```

- The sort() method sorts an array alphabetically:

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

ARRAY properties and methods

- Numeric Sort
- **By default, the sort() function sorts values as strings.**
- This works well for strings .
- However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".
- Because of this, the sort() method will produce incorrect result when sorting numbers
- Use compare function for numeric sorting.
- A function that defines an alternative sort order. • The function should return a negative, zero, or positive value, depending on the arguments, like:

```
function(a, b)
{
return a-b;
}
```

ARRAY properties and methods

- When the sort() method compares two values, it sends the values to the compare function , and sorts the values according to the returned (negative, zero, positive) value.
- Example:
- When comparing 40 and 100, the sort() method calls the compare function(40,100).The function calculates 40-100, and returns -60 (a negative value).The sort function will sort 40 as a value lower than 100.

```
    <script type="text/javascript">
        // Define an array of numbers
        var numbers = [40, 100, 1, 5, 25, 10];
        // Use the built-in sort() method
        //to perform a numerical sort
        numbers.sort(function(a, b)
        {  return a - b;
        });
// Display the sorted array
alert(numbers);
</script>
```

Reverse Arrays

- The reverse() method reverses the elements in an array.
- use it to sort an array in descending order:

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

Associate Arrays

- 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.
- Example

```
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"
```

Associate Arrays

- If you use named indexes, JavaScript will redefine the array to a standard object.
- After that, some array methods and properties will produce incorrect results.
- Example:

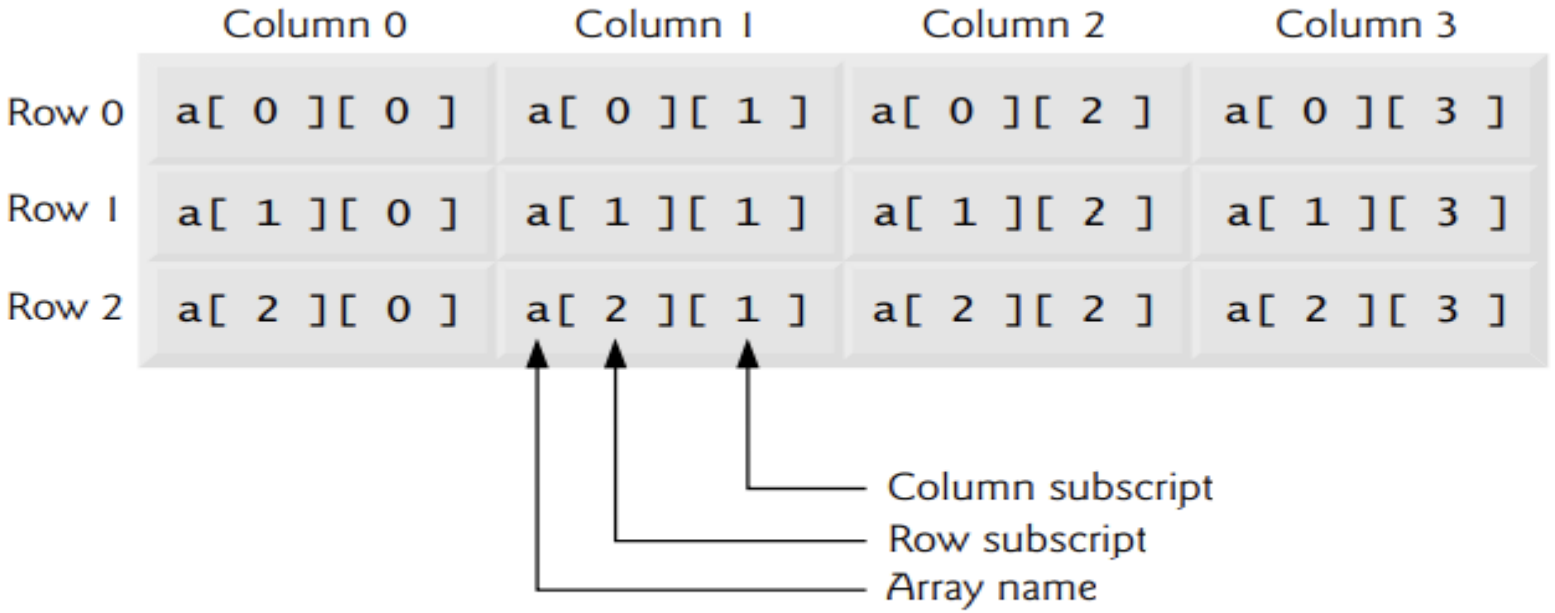
```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;
// person.length will return 0
var y = person[0];
// person[0] will return undefined
```

Multi Dimensional Arrays

- Multidimensional arrays with two indices are often used to represent tables of values consisting of information arranged in rows and columns
- Arrays that require two indices to identify a particular element are called two-dimensional arrays.
- Multidimensional arrays can have more than two dimensions.
- JavaScript does not support multidimensional arrays directly, but it does allow you to specify arrays whose elements are also arrays.
- An array with m rows and n columns is called an m-by-n array.

Multi Dimensional Arrays

- Every element in array a is identified by an element name of the form a[row][column]—a is the name of the array, and row and column are the indices that uniquely identify the row and column, respectively, of each element in a.
- The element names in row 0 all have a first index of 0; the element names in column 3 all have a second index of 3.



Two-dimensional array with three rows and four columns.

Arrays of One dimensional Arrays

- Multidimensional arrays can be initialized in declarations like a one-dimensional array. Array b with two rows and two columns could be declared and initialized with the statement .Two-dimensional array with three rows and four columns.

```
var b = [ [ 1, 2 ], [ 3, 4 ] ];
```

- The values are grouped by row in square brackets.
- The array [1, 2] initializes element b[0], and the array [3, 4] initializes element b[1]. So 1 and 2 initialize b[0][0] and b[0][1], respectively. Similarly, 3 and 4 initialize b[1][0] and b[1][1], respectively.
- interpreter determines the number of rows by counting the number of subinitializer lists— arrays nested within the outermost array. The interpreter determines the number of columns in each row by counting the number of values in the subarray that initializes the

Two-Dimensional Arrays with Rows of Different Lengths and new

- The rows of a two-dimensional array can vary in length. The declaration var b = [[1, 2], [3, 4, 5]];
- creates array b with row 0 containing two elements (1 and 2) and row 1 containing three elements (3, 4 and 5).
- Creating Two-Dimensional Arrays with new
- A multidimensional array in which each row has a different number of columns can be allocated dynamically, as follows:

```
var b;
b = new Array( 2 ); // allocate two rows
b[ 0 ] = new Array( 5 ); // allocate columns for row 0
b[ 1 ] = new Array( 3 ); // allocate columns for row 1
```

- The preceding code creates a two-dimensional array with two rows. Row 0 has five columns, and row 1 has three columns.

Two-Dimensional Arrays with Rows of Different Lengths and new

```
<script>
    var myNumbers = [[1, 2], // row 0
                     [3], // row 1
                     [4, 5, 6]]; // row 2
    for (var i = 0; i < myNumbers.length; ++i)
    {
        for (var j = 0; j < myNumbers[i].length; ++j)
        {
            document.writeln(myNumbers[i][j]);
        }
        document.writeln("<br/>");
    }
</script>
```

```
1 2
3
4 5 6
```

OBJECTS

OBJECTS

- In JavaScript, objects are a fundamental data type that allow you to store and manipulate data in a structured way.
- Almost "everything" is an object.
- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- All JavaScript values, except primitives, are objects.

OBJECTS

```
<script type="text/javascript">
    const primitiveNumber = 42; // Primitive number
    const objectNumber = new Number(42); // Number object
    alert("typeof primitiveNumber "+typeof primitiveNumber); // Output: "number"
    alert("typeof primitiveNumber "+typeof objectNumber);    // Output: "object"
    const primitiveString = "Hello, World!"; // Primitive string
    const objectString = new String("Hello, World!"); // String object
    alert("typeof primitiveString "+typeof primitiveString); // Output: "string"
    alert("typeof objectString "+typeof objectString);    // Output: "object"
    const currentDate = new Date(); // Date object
    alert("typeof current Date "+typeof currentDate); // Output: "object"
    alert("typeof Math "+typeof Math); // Output: "object"

</script>
```

OBJECTS

- With JavaScript, you can define and create your own objects.
- There are different ways to create new objects:
  1)Define and create a single object, using an object literal.
  2)Define and create a single object**, with the keyword new.**
- JavaScript objects are written with curly braces. Object properties are written as name:value pairs, separated by commas.
- The following example creates a new JavaScript object with four properties:

```
var person = {firstName:“xyz”, lastName:“pqr", age:50, eyeColor:"blue"};
```

- The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.
- A JavaScript object is a collection of named values
- The named values, in JavaScript objects, are called properties.

OBJECT Creation using new

- The following example also creates a new JavaScript object with four properties:
- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only
- Example

```
var person = new Object();
 person.firstName = “xyz";
person.lastName = “pqr";
person.age = 50;
person.eyeColor = "blue";
```

OBJECT Creation using new

- The following example also creates a new JavaScript object with four properties:
- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only
- Example

```
var person = new Object();
 person.firstName = “xyz";
person.lastName = “pqr";
person.age = 50;
person.eyeColor = "blue";
```

Add new OBJECT properties & Accessing OBJECT properties

- You can add new properties to an existing object by simply giving it a value.
- Assume that the person object already exists - you can then give it new properties:
- Example person.city = ”kochi";
- 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]

Accessing OBJECT properties

```
person.firstname + " is " + person.age + " years old.";
        or
person["firstname"] + " is " + person["age"] + " years old.";
```

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

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

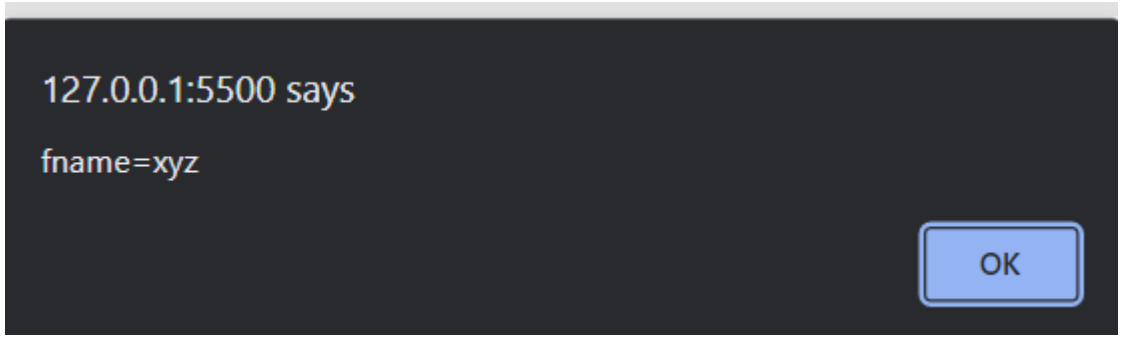
- The block of code inside of the for...in loop will be executed once for each property.

Looping through the OBJECT

JavaScript for...in Loop

- Looping through the properties of an object:
- Example

```
<script type="text/javascript">
    var person = {fname:"xyz", lname:"pqr", age:25};
    for (x in person)
    {
    alert(x+ "="+ person[x]);
    }
    </script>
```



Delete the properties of OBJECT

- The delete keyword deletes a property from an object:
- Example

```
var person = {fname:“xyz", lname:“pqr", age:50, city:“kochi"};
delete person.age;
        or
delete person["age"];
```

- The delete keyword deletes both the value of the property and the property itself.
- After deletion, the property cannot be used before it is added back again.
- The delete operator is designed to be used on object properties.
- It has no effect on variables or functions

STRINGS, NUMBERS, AND BOOLEANS AS OBJECTS

- When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();
 // Declares x as a String object
var y = new Number();
 // Declares y as a Number object
var z = new Boolean();
// Declares z as a Boolean object
```

- Avoid String, Number, and Boolean objects.

Math Object methods

Method	Description	Examples

`abs( x )`	Absolute value of x.	`abs( 7.2 )` is `7.2`
		`abs( 0 )` is `0`
		`abs( -5.6 )` is `5.6`

`ceil( x )`	Rounds x to the smallest integer not less than x.	`ceil( 9.2 )` is `10` `ceil( -9.8 )` is `-9.0`
`cos( x )`	Trigonometric cosine of x (x in radians).	`cos( 0 )` is `1`
`exp( x )`	Exponential method e^x.	`exp( 1 )` is `2.71828` `exp( 2 )` is `7.38906`
`floor( x )`	Rounds x to the largest integer not greater than x.	`floor( 9.2 )` is `9` `floor( -9.8 )` is `-10.0`
`log( x )`	Natural logarithm of x (base *e*).	`log( 2.718282 )` is `1` `log( 7.389056 )` is `2`

Math Object methods

`max( x, y )`	Larger value of x and y.	`max( 2.3, 12.7 )` is 12.7 `max( -2.3, -12.7 )` is -2.3
`min( x, y )`	Smaller value of x and y.	`min( 2.3, 12.7 )` is 2.3 `min( -2.3, -12.7 )` is -12.7
`pow( x, y )`	x raised to power y (x^y).	`pow( 2, 7 )` is 128 `pow( 9, .5 )` is 3.0
`round( x )`	Rounds x to the closest integer.	`round( 9.75 )` is 10 `round( 9.25 )` is 9
`sin( x )`	Trigonometric sine of x (x in radians).	`sin( 0 )` is 0
`sqrt( x )`	Square root of x.	`sqrt( 900 )` is 30 `sqrt( 9 )` is 3
`tan( x )`	Trigonometric tangent of x (x in radians).	`tan( 0 )` is 0

Math Object methods

```
alert(Math.round(3.7));
alert(Math.abs(-5));
alert(Math.floor(3.7));
alert(Math.ceil(3.2));
alert(Math.min(5, 2, 8, 1));
alert(Math.max(5, 2, 8, 1));
alert(Math.pow(2, 3));
alert(Math.sin(45));
alert(Math.sqrt(625));
```

String Object methods

Method	Description
`charAt( index )`	Returns a string containing the character at the specified *index*. If there's no character at the *index*, `charAt` returns an empty string. The first character is located at *index* 0.
`charCodeAt( index )`	Returns the Unicode value of the character at the specified *index*, or `NaN` (not a number) if there's no character at that *index*.
`concat( string )`	Concatenates its argument to the end of the string on which the method is invoked. The original string is not modified; instead a new `String` is returned. This method is the same as adding two strings with the string-concatenation operator + (e.g., `s1.concat(s2)` is the same as `s1 + s2`).
`fromCharCode( value1, value2, ...)`	Converts a list of Unicode values into a string containing the corresponding characters.

String Object methods

`indexOf(` *substring*, *index* `)`	Searches for the *first* occurrence of *substring* starting from position *index* in the string that invokes the method. The method returns the starting index of *substring* in the source string or –1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from index 0 in the source string.
`lastIndexOf(` *substring*, *index* `)`	Searches for the *last* occurrence of *substring* starting from position *index* and searching toward the beginning of the string that invokes the method. The method returns the starting index of *substring* in the source string or –1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from the *end* of the source string.
`replace(` *searchString*, *replaceString* `)`	Searches for the substring *searchString*, replaces the first occurrence with *replaceString* and returns the modified string, or returns the original string if no replacement was made.
`slice(` *start*, *end* `)`	Returns a string containing the portion of the string from index *start* through index *end*. If the *end* index is not specified, the method returns a string from the *start* index to the end of the source string. A negative *end* index specifies an offset from the end of the string, starting from a position one past the end of the last character (so –1 indicates the last character position in the string).

String Object methods

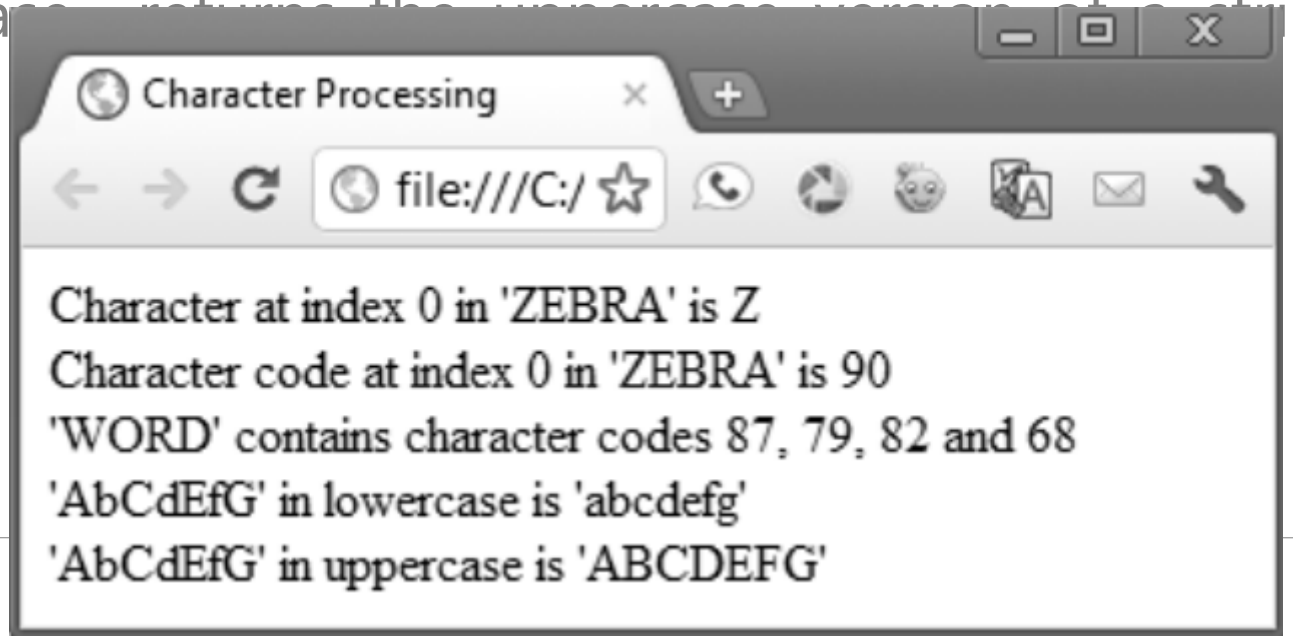
`split( string )`	Splits the source string into an array of strings (tokens), where its *string* argument specifies the delimiter (i.e., the characters that indicate the end of each token in the source string).
`substr( start, length )`	Returns a string containing *length* characters starting from index *start* in the source string. If *length* is not specified, a string containing characters from *start* to the end of the source string is returned.
`substring( start, end )`	Returns a string containing the characters from index *start* up to but not including index *end* in the source string.
`toLowerCase()`	Returns a string in which all uppercase letters are converted to lowercase letters. Non-letter characters are not changed.
`toUpperCase()`	Returns a string in which all lowercase letters are converted to uppercase letters. Non-letter characters are not changed.

String Object methods

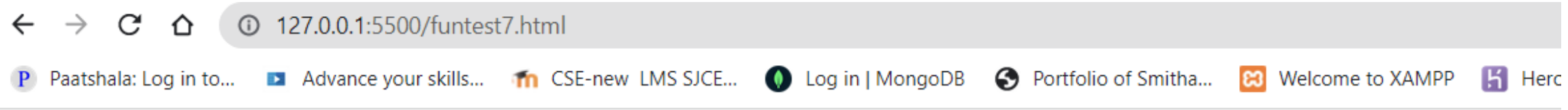
```
<script type="text/javascript">
    var str = "Hello, World!";
    alert(str.length); // Output: 13
    var str = "Hello";
    alert(str.charAt(1)); // Output: "e"
    var str = "Hello, World!";
    alert(str.substring(7, 12)); // Output: "World"
    var str = "Hello, World!";
    alert(str.toUpperCase()); // Output: "HELLO, WORLD!"
    alert(str.toLowerCase()); // Output: "hello, world!"
    var str = "Hello, World!";
    alert(str.replace("World", "Universe")); // Output: "Hello, Universe!"
    var str = "  Hello, World!  ";
    alert(str.trim()); // Output: "Hello, World!"
</script>
```

Character processing methods

- String object's character-processing methods, including:
  - charAt—returns the character at a specific position
  - charCodeAt—returns the Unicode value of the character at a specific position
  - fromCharCode—returns a string created from a series of Unicode values
  - toLowerCase—returns the lowercase version of a string
  - toUpperCase—returns the uppercase version of a string



Character processing methods



The string to search is: abcdefghijklmnopqrstuvwxyzabcdefghijklm

Enter the substring to search for def Search

First occurrence is located at index 3

Last occurrence is located at index 29

First occurrence from index 12 is located at index 29

Last occurrence from index 12 is located at index 3

Character processing methods

```
<script>
    var letters = "abcdefghijklmnopqrstuvwxyzabcdefghijklm";
    function buttonPressed() {
        var inputField = document.getElementById("inputField");
        document.getElementById("results").innerHTML =
            "<p>First occurrence is located at index " + letters.indexOf(inputField.value)
            + "</p>" + "<p>Last occurrence is located at index " + letters.lastIndexOf(inputField.value)
            + "</p>" + "<p>First occurrence from index 12 is located at index " + letters.indexOf(inputField.value, 12)
            + "</p>" + "<p>Last occurrence from index 12 is located at index " + letters.lastIndexOf(inputField.value, 12)
            + "</p>";
    }
    function start() {
        var searchButton = document.getElementById("searchButton");
        searchButton.addEventListener("click", buttonPressed, false);
    } // end function start
    window.addEventListener("load", start, false);
</script>
<body>
    <form id="searchForm">
        <h1>The string to search is:
            abcdefghijklmnopqrstuvwxyzabcdefghijklm</h1>
        <p>Enter the substring to search for
            <input id="inputField" type="search">
            <input id="searchButton" type="button" value="Search">
        </p>
        <div id="results"></div>
    </form>
</body>
```

Date Object Methods

Method	Description
`getDate()` `getUTCDate()`	Returns a number from 1 to 31 representing the day of the month in local time or UTC.
`getDay()` `getUTCDay()`	Returns a number from 0 (Sunday) to 6 (Saturday) representing the day of the week in local time or UTC.
`getFullYear()` `getUTCFullYear()`	Returns the year as a four-digit number in local time or UTC.
`getHours()` `getUTCHours()`	Returns a number from 0 to 23 representing hours since midnight in local time or UTC.
`getMilliseconds()` `getUTCMilliSeconds()`	Returns a number from 0 to 999 representing the number of milliseconds in local time or UTC, respectively. The time is stored in hours, minutes, seconds and milliseconds.
`getMinutes()` `getUTCMinutes()`	Returns a number from 0 to 59 representing the minutes for the time in local time or UTC.
`getMonth()` `getUTCMonth()`	Returns a number from 0 (January) to 11 (December) representing the month in local time or UTC.
`getSeconds()` `getUTCSeconds()`	Returns a number from 0 to 59 representing the seconds for the time in local time or UTC.

Date Object Methods

Method	Description
`getTime()`	Returns the number of milliseconds between January 1, 1970, and the time in the `Date` object.
`getTimezoneOffset()`	Returns the difference in minutes between the current time on the local computer and UTC (Coordinated Universal Time).
`setDate( val )` `setUTCDate( val )`	Sets the day of the month (1 to 31) in local time or UTC.
`setFullYear( y, m, d )` `setUTCFullYear( y, m, d )`	Sets the year in local time or UTC. The second and third arguments representing the month and the date are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setHours( h, m, s, ms )` `setUTCHours( h, m, s, ms )`	Sets the hour in local time or UTC. The second, third and fourth arguments, representing the minutes, seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMilliSeconds( ms )` `setUTCMilliseconds( ms )`	Sets the number of milliseconds in local time or UTC.

Date Object Methods

Method	Description
`getTime()`	Returns the number of milliseconds between January 1, 1970, and the time in the `Date` object.
`getTimezoneOffset()`	Returns the difference in minutes between the current time on the local computer and UTC (Coordinated Universal Time).
`setDate( val )` `setUTCDate( val )`	Sets the day of the month (1 to 31) in local time or UTC.
`setFullYear( y, m, d )` `setUTCFullYear( y, m, d )`	Sets the year in local time or UTC. The second and third arguments representing the month and the date are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setHours( h, m, s, ms )` `setUTCHours( h, m, s, ms )`	Sets the hour in local time or UTC. The second, third and fourth arguments, representing the minutes, seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMilliSeconds( ms )` `setUTCMilliseconds( ms )`	Sets the number of milliseconds in local time or UTC.

Date Object Methods

`setUTCHours( h, m, s, ms )`	fourth arguments, representing the minutes, seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMilliSeconds( ms )` `setUTCMilliseconds( ms )`	Sets the number of milliseconds in local time or UTC.
`setMinutes( m, s, ms )` `setUTCMinutes( m, s, ms )`	Sets the minute in local time or UTC. The second and third arguments, representing the seconds and milliseconds, are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setMonth( m, d )` `setUTCMonth( m, d )`	Sets the month in local time or UTC. The second argument, representing the date, is optional. If the optional argument is not specified, the current date value in the `Date` object is used.
`setSeconds( s, ms )` `setUTCSeconds( s, ms )`	Sets the seconds in local time or UTC. The second argument, representing the milliseconds, is optional. If this argument is not specified, the current milliseconds value in the `Date` object is used.
`setTime( ms )`	Sets the time based on its argument—the number of elapsed milliseconds since January 1, 1970.

Date Object Methods

`toLocaleString()`	Returns a string representation of the date and time in a form specific to the computer's locale. For example, September 13, 2007, at 3:42:22 PM is represented as *09/13/07 15:47:22* in the United States and *13/09/07 15:47:22* in Europe.
`toUTCString()`	Returns a string representation of the date and time in the form: *15 Sep 2007 15:47:22 UTC.*
`toString()`	Returns a string representation of the date and time in a form specific to the locale of the computer (*Mon Sep 17 15:47:22 EDT 2007* in the United States).
`valueOf()`	The time in number of milliseconds since midnight, January 1, 1970. (Same as `getTime`.)

Boolean and Number Objects

- JavaScript provides the Boolean and Number objects as object wrappers for boolean true/ false values and numbers, respectively.
- These wrappers define methods and properties useful in manipulating boolean values and numbers.
- When a JavaScript program requires a boolean value, JavaScript automatically creates a Boolean object to store the value.

  var b = new Boolean(booleanValue);
- The booleanValue specifies whether the Boolean object should contain true or false.
- If booleanValue is false, 0, null, Number.NaN or an empty string (""), or if no argument is supplied, the new Boolean object contains false.

`toString()`	Returns the string "`true`" if the value of the `Boolean` object is `true`; otherwise, returns the string "`false`".
`valueOf()`	Returns the value `true` if the `Boolean` object is `true`; otherwise, returns `false`.

Boolean and Number Objects

- JavaScript automatically creates Number objects to store numeric values in a script. You can create a Number object with the statement

  var n = new Number(numericValue);

- The constructor argument numericValue is the number to store in the object.
- We can explicitly create Number objects, normally the JavaScript interpreter creates them as needed

Boolean and Number Objects

`toString( radix )`	Returns the string representation of the number. The optional *radix* argument (a number from 2 to 36) specifies the number's base. Radix 2 results in the *binary* representation, 8 in the *octal* representation, 10 in the *decimal* representation and 16 in the *hexadecimal* representation. See Appendix E, Number Systems, for an explanation of the binary, octal, decimal and hexadecimal number systems.
`valueOf()`	Returns the numeric value.
`Number.MAX_VALUE`	The largest value that can be stored in a JavaScript program.
`Number.MIN_VALUE`	The smallest value that can be stored in a JavaScript program.
`Number.NaN`	*Not a number*—a value returned from an arithmetic expression that doesn't result in a number (e.g., `parseInt("hello")` cannot convert the string `"hello"` to a number, so `parseInt` would return `Number.NaN`.) To determine whether a value is `NaN`, test the result with function `isNaN`, which returns `true` if the value is `NaN`; otherwise, it returns `false`.
`Number.NEGATIVE_INFINITY`	A value less than `-Number.MAX_VALUE`.
`Number.POSITIVE_INFINITY`	A value greater than `Number.MAX_VALUE`.

Document Objects

- The document object, which is provided by the browser and allows JavaScript code to manipulate the current document in the browser.
- The document object has several properties and methods, such as method document.getElementByID

Method	Description
`getElementById( id )`	Returns the HTML5 element whose `id` attribute matches *id*.
`getElementByTagName( tagName )`	Returns an array of the HTML5 elements with the specified *tagName*.

DOCUMENT OBJECT MODEL

The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

DOM

- DOM is ***a model for describing HTML documents***
- A common set of properties/methods to access everything in a web 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:
- Objects are in a hierarchy
- The window is the parent for a given web page
- Document is the child with the objects that are most commonly manipulated
- The HTML *DOM is a standard* for how to get, change, add, or delete HTML elements.
- HTML DOM methods are actions you can perform (on HTML Elements).
- HTML DOM properties are values (of HTML Elements) that you can set or change.

Document Object Model

- The DOM gives you scripting access to all the elements on a web page.
- Inside the browser, the whole web page—paragraphs, forms, tables, etc.—is represented in an object hierarchy.
- Using DOM, we can dynamically create, modify, and remove elements on the page.
- JavaScript can change all the HTML elements on the page
- JavaScript can change all the HTML attributes on the page
- JavaScript can change all the CSS styles on the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events on the page
- JavaScript can create new HTML events on the page

DOM nodes and Trees

- The document's getElementById method is the simplest way to access a specific element in a page. The method returns objects called DOM nodes.
- Every piece of an HTML5 page (elements, attributes, text, etc.) is modeled in the web browser by a DOM node.
- All the nodes in a document make up the page's DOM tree, which describes the relationships among elements.
- Nodes are related to each other through child-parent relationships.
- An HTML5 element inside another element is said to be its child—the containing element is known as the parent.
- A node can have multiple children but only one parent.
- Nodes with the same parent node are referred to as siblings.

DOM nodes and Trees

- The head and body nodes are siblings since they're both children of the html node.
- The head contains the meta and title nodes
- The body node contains nodes representing each of the elements in the document's body element.
- The li nodes are children of the ul node since they're nested inside it.

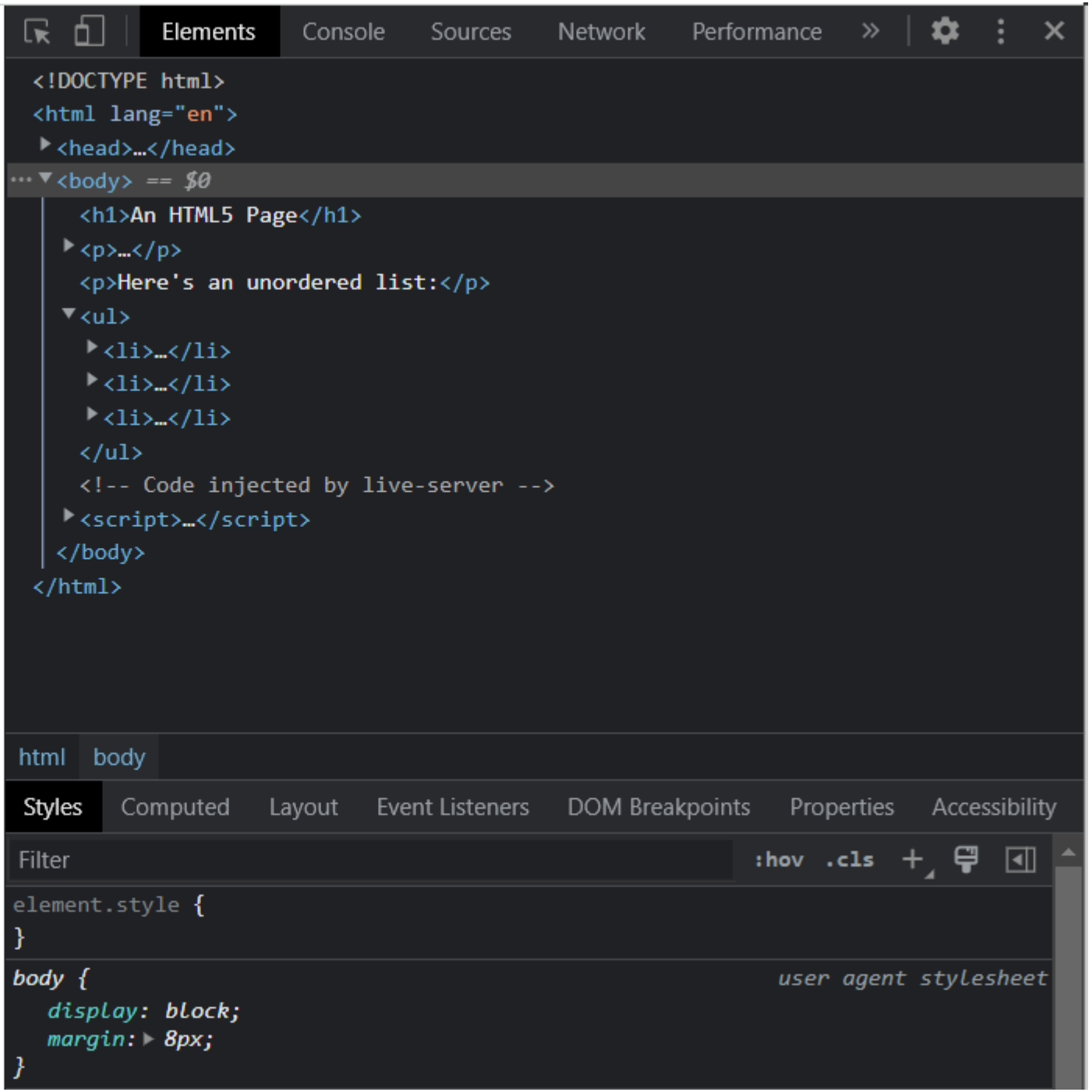
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>An HTML5 Page</h1>
 <p>This page contains some basic HTML5 elements. The DOM tree
 for the document contains a DOM node for every element</p>
 <p>Here's an unordered list:</p>
<ul>
 <li>One</li>
 <li>Two</li>
 <li>Three</li>
 </ul>
</body>
</html>
```

An HTML5 Page

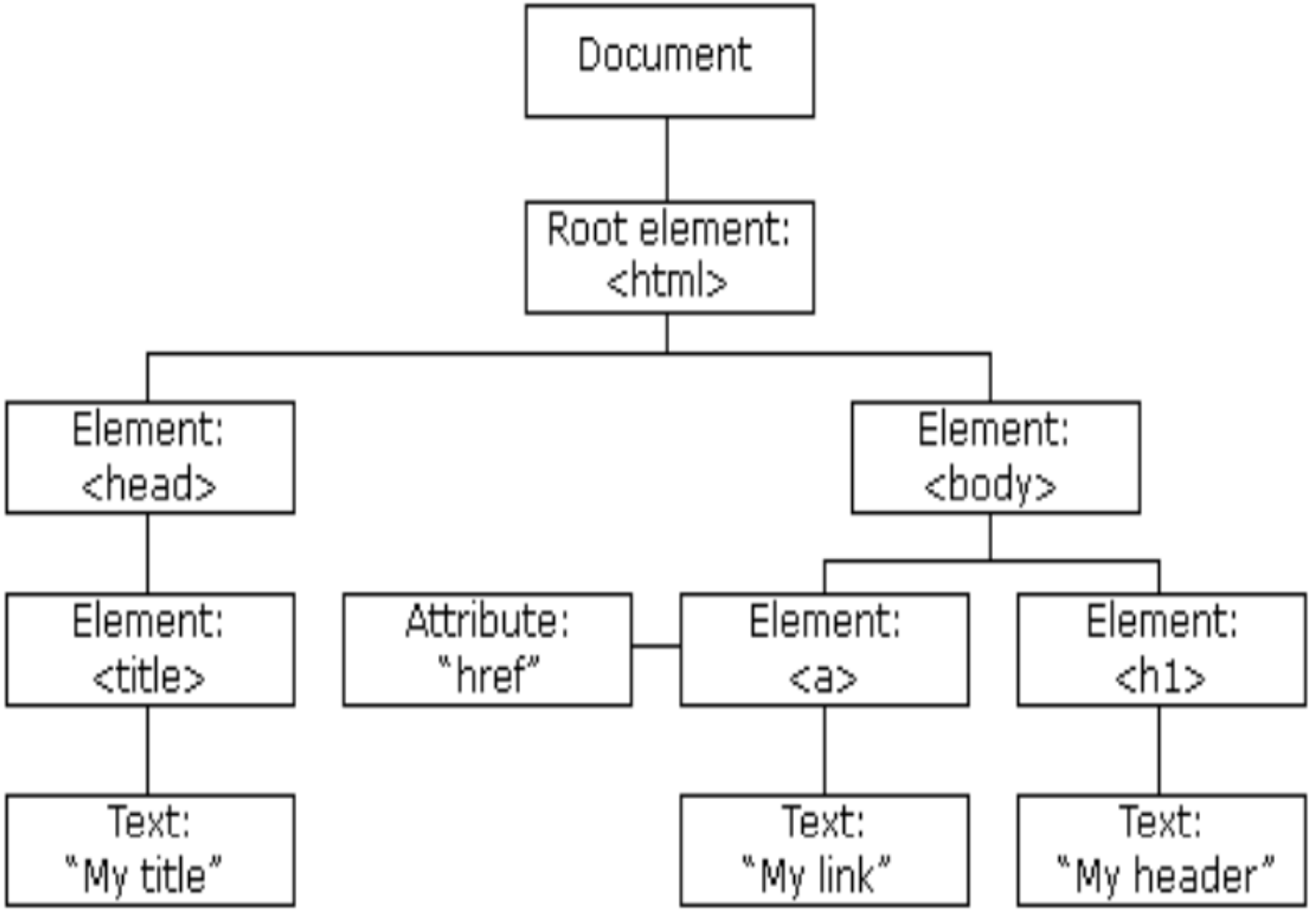
This page contains some basic HTML5 elements. The DOM tree for the document contains a DOM node for every element

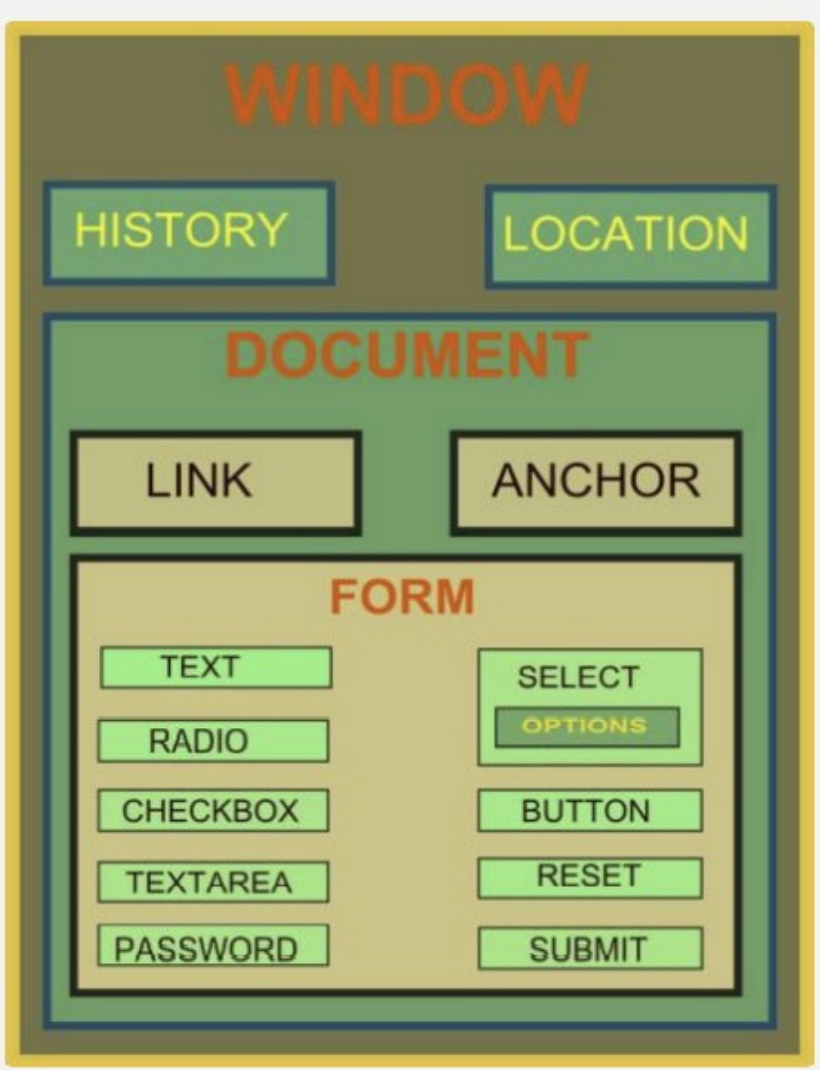
Here's an unordered list:

- One
- Two
- Three



DOM nodes and Trees





FINDING & CHANGING HTML ELEMENTS

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

ADDING AND DELETING HTML ELEMENTS

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

CHANGING THE VALUE OF AN ATTRIBUTE

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

  document.getElementById(id).attribute = new value

```
<!DOCTYPE html>
<html><body>
<img id="clg" src="sjc.jpg">
<script>
    document.getElementById("clg").src ="sjclogo.jpg";
</script>
</body>
</html>
```

CHANGING THE HTML Style

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

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

```
<html>
<body>
<p id="p1">Hello World!</p>
<script>
    document.getElementById("p1").style.color = "blue";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

DOM EVENTS

- HTML DOM allows JavaScript to react to HTML events:
- JavaScript can execute a statement (typically, call a function)
- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:
- when an event occurs

```
<… oneventname="javascript stmt;">
<BODY … onload="func();">
<INPUT type="submit“ name=“submit” onsubmit="fun();">
```

DOM EVENTS

- onsubmit - call when submit button is clicked
- onclick - call when this button is clicked
- onreset - call when the reset button is clicked
- onload - call after page loads
- onmouseover - call when mouse pointer enters image area
- onmouseout - call when mouse pointer leaves image area
- onfocus - call when control receives focus
- onblur - call when a control loses focus
- onchange - call when a control loses focus and the value of its contents has changed
- many more

DOM EVENTS

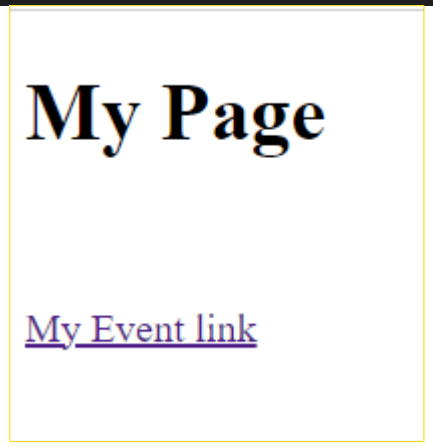
Event Name	Description
`click`	Triggered when the element is clicked with the mouse or activated via the keyboard.
`dblclick`	Fired when the element is double-clicked.
`mousedown`	Occurs when the mouse button is pressed down over the element.
`mouseup`	Occurs when the mouse button is released after a `mousedown` event.
`mousemove`	Fired when the mouse pointer moves over the element.
`mouseenter`	Triggered when the mouse pointer enters the boundaries of the element.
`mouseleave`	Fired when the mouse pointer leaves the boundaries of the element.
`mouseover`	Triggered when the mouse pointer enters the element or one of its child elements.
`mouseout`	Fired when the mouse pointer leaves the element or one of its child elements.

DOM EVENTS

`keydown`	Occurs when a key on the keyboard is pressed down.
`keyup`	Occurs when a key on the keyboard is released after a `keydown` event.
`keypress`	Fired when a key on the keyboard is pressed and released.
`focus`	Triggered when an element receives focus, such as through keyboard navigation or mouse click.
`blur`	Occurs when an element loses focus.
`change`	Fired when the value of an input, select, or textarea element is changed by the user.
`submit`	Occurs when a form is submitted, either through a button click or by pressing Enter while in a form field.
`load`	Fired when an element, such as an image or an iframe, has finished loading.
`unload`	Occurs when a page is being unloaded or refreshed.
`resize`	Fired when the size of the browser window is changed.
`scroll`	Triggered when the user scrolls an element or the page.

Event handling using DOM –onclick and onMouseover

```
<!DOCTYPE html>
<html lang="en">
<head>···
</head>
<body>
    <h1 onclick="this.innerHTML = 'Ooops!'">My Page</h1>
    <br /><br />
    <a href="jtest1.html" onMouseover="window.alert('Hello');">My Event link</a>
</body>
```



Event handling using DOM –onload()

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta http-equiv="X-UA-Compatible" content="IE=edge">
6       <meta name="viewport" content="width=device-width, initial-scale=1.0">
7       <title>Document</title>
8
9   <script>
10      function test()
11      {
12      document.getElementById("s2").style.color = "red";
13      document.getElementById("s3").style.color= "green";
14      }
15      </script></head><body onload="test()">
16      <h1 id="s1">SJCET</h1><h2 id="s2">SJCET BTECH</h1>
17      <h3 id="s3">SJCET MTECH</h1>
18      <h4 id="s4">SJCET MBA</h1>
19  </body>
20  </html>
```

SJCET

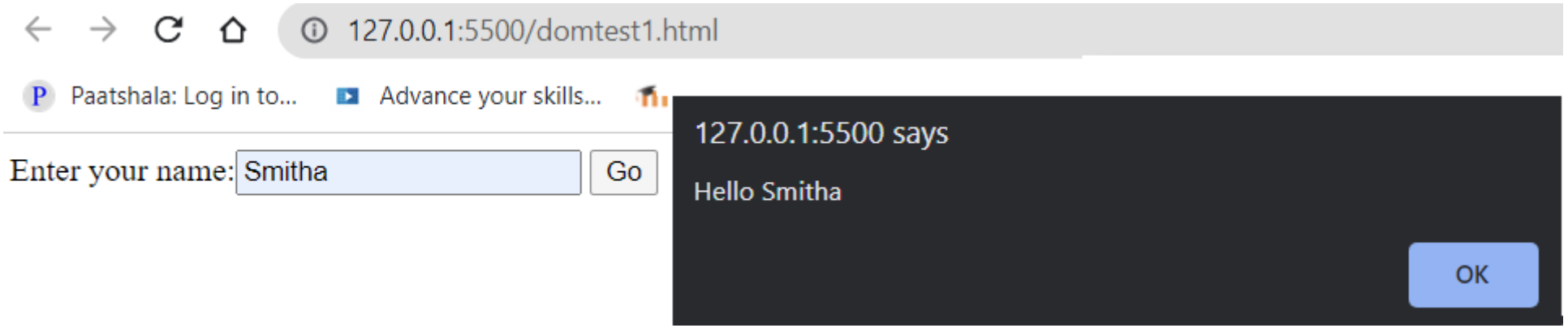
SJCET BTECH

SJCET MTECH

SJCET MBA

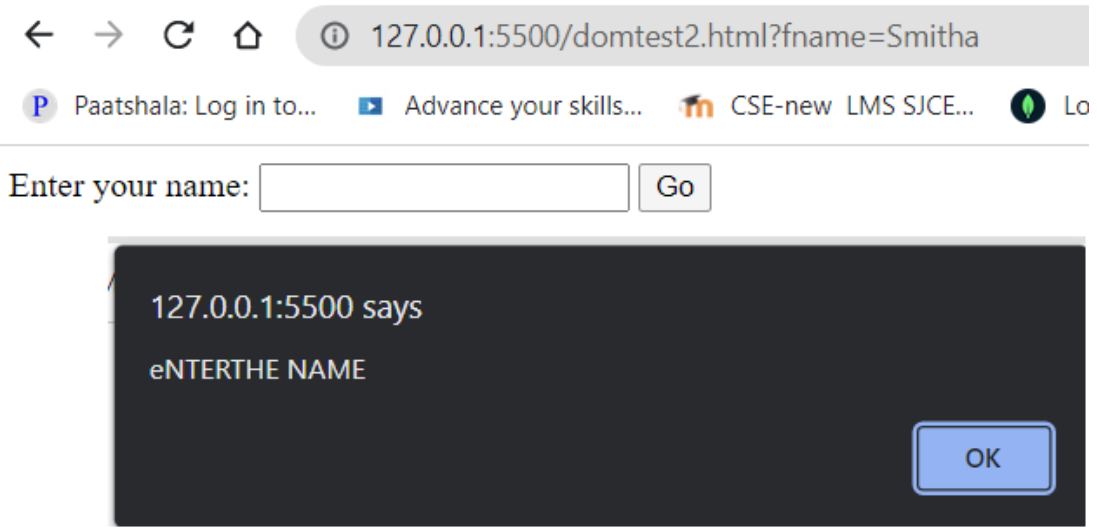
Form Handling –onclick event

```
<!DOCTYPE html>
<html lang="en">
<head>…
</head>
<body>
    <form name="f1">
        Enter your name:<input type="text" name="fname">
        <input type="button" value= "Go" onClick="window.alert ('Hello ' + document.f1.fname.value);">
        </form>
    </body> </html>
```

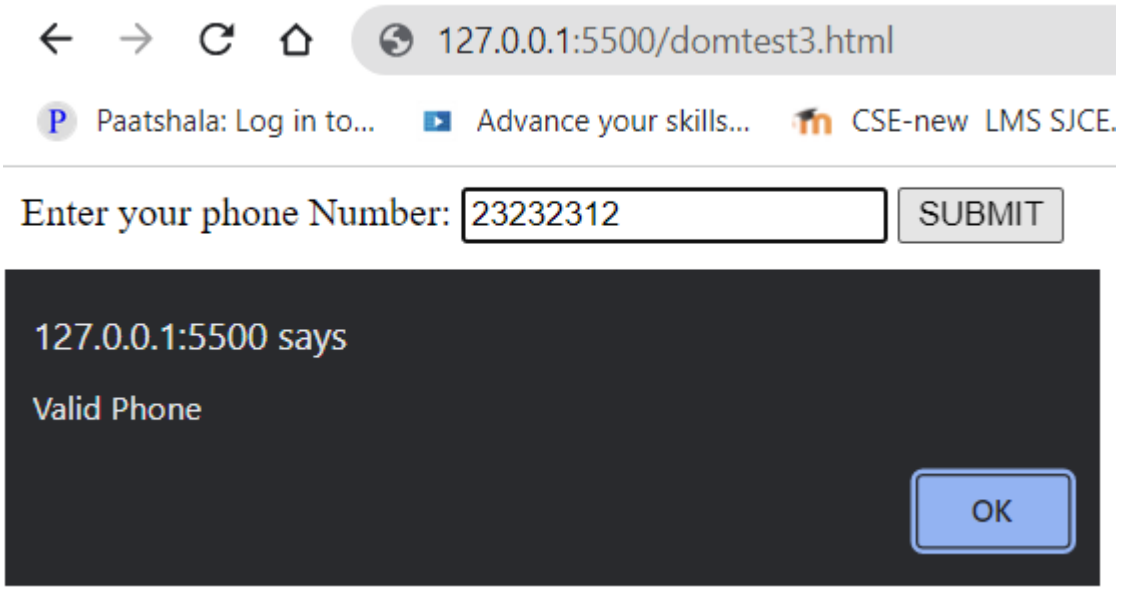


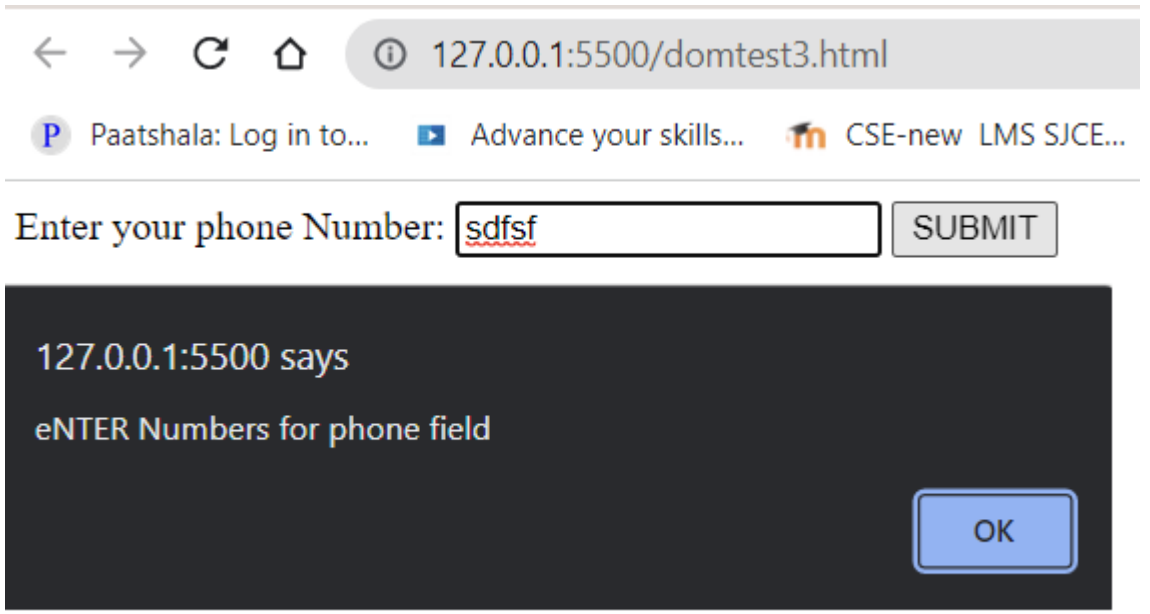
Form Handling –onclick event

```
<script>
    function valname() {
    if(document.f1.fname.value=="")
    {window.alert('eNTERTHE NAME');
    }
    else
    {window.alert('Hello ' +document.f1.fname.value); }
    }
    </script></head>
    <body><form name="f1" onSubmit="valname()">
    Enter your name: <input type="text" name="fname">
    <input type="submit" value= "Go" >
    </form></body></html>
```



Form Handling –onchange event





Form Handling –onchange event

```
<script>
    function valphone()
    {
        if ((document.f1.phone.value != ""))
        {
            if (isNaN(document.f1.phone.value))
            { window.alert('eNTER Numbers for phone field'); }
            else {
                window.alert("Valid Phone "); }
        }
        else
        {
            window.alert("Enter Phone ");
        }
    }
</script></head><body>
<form name="f1">
    Enter your phone Number:
    <input type="text" name="phone" onchange="valphone()">
    <input type="submit" value="SUBMIT">
</form>
</body>
```

Node in DOM

- Any element on a page including text & whitespaces of a DOM structure is known as “NODE.”
- Nodes can be between HTML Tags.
- Nodes available in DOM:
  - node.childNodes
  - node.firstChild
  - node.lastChild
  - node.parentNode
  - node.nextSibling
  - node.previousSibling

Adding and Removing Nodes (HTML Elements) **in DOM**

- To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<body ><h2>JavaScript HTML DOM</h2>
    <p>Add a new HTML Element.</p>

    <div id="div1">
    <p id="p1">This is a paragraph.</p>
    <p id="p2">This is another paragraph.</p>
    </div>

    <script>
    const para = document.createElement("p");
    const node = document.createTextNode("This is new PARA.");
    para.appendChild(node);
    const element = document.getElementById("div1");
    element.appendChild(para);
    </script>
</body>
```

JavaScript HTML DOM

Add a new HTML Element.

This is a paragraph.

This is another paragraph.

This is new PARA.

Adding and Removing Nodes (HTML Elements) **in DOM**

- This code creates a new <p> element:

  const para = document.createElement("p");

- To add text to the <p> element, you must create a text node first. This code creates a text node: const node = document.createTextNode("This is a new paragraph.");
- Then you must append the text node to the <p> element:

  para.appendChild(node);

- Finally you must append the new element to an existing element.This code finds an existing element:

  const element = document.getElementById("div1");

- This code appends the new element to the existing element:

  element.appendChild(para);

Insert before in Node

- This code appends the new element to the existing element:

element.appendChild(para);

JavaScript HTML DOM

Add a new HTML Element.

This is new.

This is a paragraph.

This is another paragraph.

```
<h2>JavaScript HTML DOM</h2>
<p>Add a new HTML Element.</p>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

Remove element in Node

JavaScript HTML DOM

Add a new HTML Element.

This is new.

This is a paragraph.

This is another paragraph.

JavaScript HTML DOM

Remove an HTML Element.

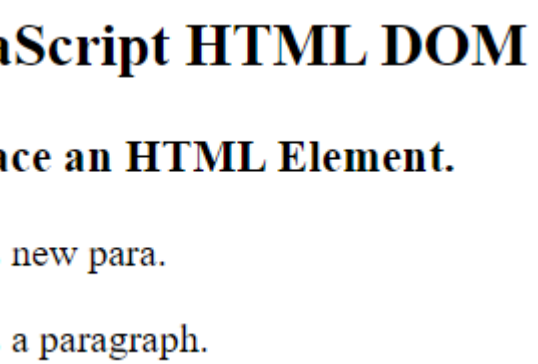
This is another paragraph.

Remove Element

```
<body>
    <h2>JavaScript HTML DOM</h2>
<h3>Remove an HTML Element.</h3>
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<button onclick="myFunction()">Remove Element</button>

<script>
function myFunction() {
document.getElementById("p1").remove();
}
</script>
</body>
```

Replacing child element in Node

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

<h2>JavaScript HTML DOM</h2>
<h3>Replace an HTML Element.</h3>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is a paragraph.</p>
</div>

<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
const para = document.createElement("p");
const node = document.createTextNode("This is new para.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
</html>
```



DOM Collection

- The Document Object Model contains several collections, which are groups of related objects on a page.
- DOM collections are accessed as properties of DOM objects such as the document object or a DOM node.
- The document object has properties containing the
  - images collection
  - links collection
  - forms collection
  - anchors collection
- These collections contain all the elements of the corresponding type on the page

```html
<!DOCTYPE html>
<html><head><title>Links Collection length</title></head>
<body><form><fieldset>
<legend>Links-Collection-length</legend>
<a href="https://www.google.com">Google</a>   
<a href="https://www.bing.com">Bing</a>   
<a href="https://www.ebay.com">E-bay</a>   
<a href="https://www.sjcetpalai.ac.in">SJCET</a><br/><br/>
<input type="button" value="Get Links" onclick="getLinks()">
<div id="divDisplay"></div></fieldset></form>
<script>
   var divDisplay = document.getElementById("divDisplay");
   function getLinks(){
      var links = document.links.length;
      divDisplay.textContent = 'Total Links: '+links;
      for (var i = 0; i < links; i++) {
         divDisplay.textContent += '--> '+document.links[i]+' ';
      }
   }
</script></body></html>
```

Links-Collection-length

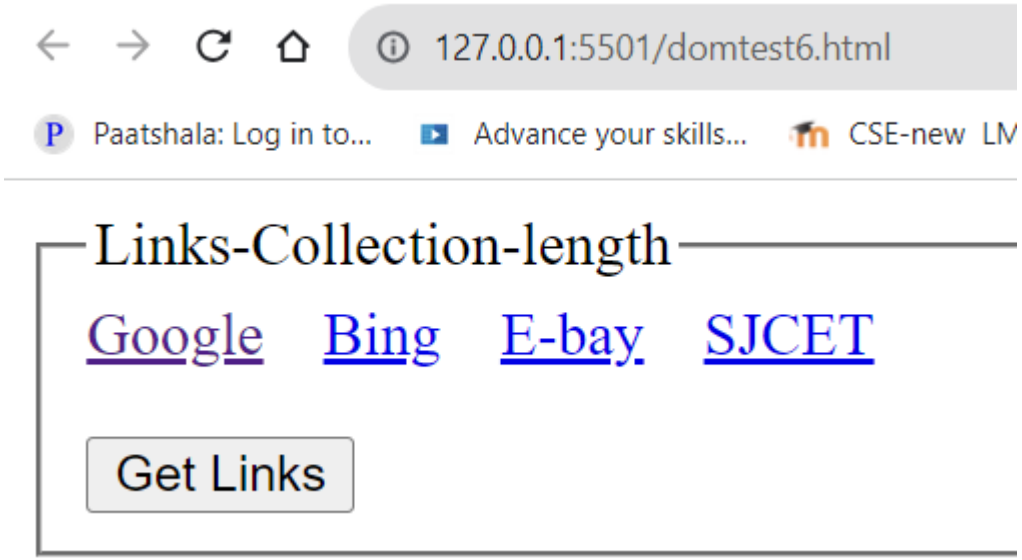
Google Bing E-bay SJCET

Get Links

DOM Collection

```
3     <form>
4         <fieldset>
5             <legend>Links-Collection-length</legend>
6             <a href="https://www.google.com">Google</a>   
7             <a href="https://www.bing.com">Bing</a>   
8             <a href="https://www.ebay.com">E-bay</a>   
9             <a href="https://www.sjcetpalai.ac.in">SJCET</a><br/><br/>
10            <input type="button" value="Get Links" onclick="getLinks()">
11            <div id="divDisplay"></div>
12        </fieldset>
13    </form>
14    <script>
15        var divDisplay = document.getElementById("divDisplay");
16        function getLinks() {
17            // Get the links collection from the document object
18            const linksCollection = document.links;
19            var links = document.links.length;
20            divDisplay.innerHTML = 'Total Links: ' + links + '<br>';
21            for (let i = 0; i < linksCollection.length; i++) {
22                divDisplay.innerHTML += "Link URL: " + linksCollection[i].href + '<br>';
23                divDisplay.innerHTML += "Link Text: " + linksCollection[i].textContent + '<br>';
24            }
25        }
26    </script></body></html>
```

DOM Collection



Links-Collection-length

Google Bing E-bay SJCET

Get Links

Total Links: 4
Link URL: https://www.google.com/
Link Text: Google
Link URL: https://www.bing.com/
Link Text: Bing
Link URL: https://www.ebay.com/
Link Text: E-bay
Link URL: https://www.sjcetpalai.ac.in/
Link Text: SJCET

Thank You