CUI Abbottabad

Department of Computer Science

Web Technologies

Lecture -11-[CLO-2]
Javascript-Part6

COMSATS University Islamabad, Abbottabad Campus

Agenda

- Math Object
- String Object
- Date Object
- Number Object
- Document Object

MATH OBJECT

- Math object methods enable you to conveniently perform many common mathematical calculations.
- An object's methods are called by writing the name of the object followed by a dot operator (.) and the name of the method
- In parentheses following the method name are arguments to the method, For example, to calculate the square root of 900 you might write:

```
var result = Math.sqrt( 900 );
```

MATH OBJECT

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`
`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`

`Math` object methods. (Part 1 of 2.)

MATH OBJECT

Method	Description	Examples
`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. (Part 2 of 2.)

MATH OBJECT

Constant	Description	Value
`Math.E`	Base of a natural logarithm (*e*).	Approximately 2.718
`Math.LN2`	Natural logarithm of 2.	Approximately 0.693
`Math.LN10`	Natural logarithm of 10.	Approximately 2.302
`Math.LOG2E`	Base 2 logarithm of *e*.	Approximately 1.442
`Math.LOG10E`	Base 10 logarithm of *e*.	Approximately 0.434
`Math.PI`	π—the ratio of a circle's circumference to its diameter.	Approximately 3.141592653589793
`Math.SQRT1_2`	Square root of 0.5.	Approximately 0.707
`Math.SQRT2`	Square root of 2.0.	Approximately 1.414

Properties of the `Math` object.

STRING OBJECT

- Characters are the building blocks of JavaScript programs
- Every program is composed of a sequence of characters grouped together meaningfully that is interpreted by the computer as a series of instructions used to accomplish a task
- A string is a series of characters treated as a single unit
- A string may include letters, digits and various **special characters**, such as +, -, *, /, and $
- JavaScript supports **Unicode**, which represents a large portion of the world's languages.
- A string is an object of type **String**.
- **String literals** or **string constants** are written as a sequence of characters in double or single quotation marks
- Strings can be compared via:
  - the relational (<, <=, > and >=) and
  - equality operators (==, ===, != and!==).
- The comparisons are based on the Unicode values of the corresponding characters.
- For example, the expression "h" < "H" evaluates to false because lowercase letters have higher Unicode values

METHODS OF STRING OBJECT

- The String object encapsulates the attributes and behaviors of a string of characters.
- It provides many methods (behaviors) that accomplish useful tasks such as:
  - selecting characters from a string,
  - combining strings (called **concatenation**),
  - obtaining *substrings* (portions) of a string,
  - searching for substrings within a string,
  - *tokenizing strings* (i.e., splitting strings into individual words) and
  - converting strings to all uppercase or lowercase letters.
- The String object also provides several methods that generateHTML5 tags.

METHODS OF STRING OBJECT

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.
`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.

Some `String`-object methods. (Part 1 of 3.)

METHODS OF STRING OBJECT

Method	Description
`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).

Some `String`-object methods. (Part 2 of 3.)

METHODS OF STRING OBJECT

Method	Description
`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.

Some `String`-object methods. (Part 3 of 3.)

METHODS OF STRING OBJECT

CHARACTER PROCESSING METHODS

1. String method charAt ()

- Returns the character at a specific position
- Indices for the characters in a string start at 0 (the first character) and go up to (but do not include) the string's length
- If the index is outside the bounds of the string, the method returns an empty string

2. String method charCodeAt()

- Returns the Unicode value of the character at a specific position
- If the index is outside the bounds of the string, the method returns NaN.

3. String method fromCharCode()

- Returns a string created from a series of Unicode values

METHODS OF STRING OBJECT

CHARACTER PROCESSING METHODS

4. **String method toLowerCase()**
   - Returns the lowercase version of a string
5. **String method toUpperCase()**
   - Returns the uppercase version of a string

- In the next example(Listing-11-1 and 11-2),the HTML document (Fig. 11.1) calls the script's start function(Fig. 11.2) to display the results in the results div. [*Note:* Throughout this lecture, we show the CSS style sheets only if there are new features to discuss.

THE HTML FILE

```html
<!DOCTYPE html>

<!-- Fig. 11.1: Listing-11-1-CharacterClassMethods.html -->
 <!-- HTML5 document to demonstrate String methods charAt, charCodeAt,
 fromCharCode, toLowercase and toUpperCase. -->
 <html>
 <head>
    <meta charset = "utf-8">
    <title>Character Processing</title>
    <link rel = "stylesheet" type = "text/css" href = "style.css">
    <script src = "Listing-11-2-CharacterProcessing.js"></script>
 </head>
 <body>
    <div id = "results"></div>
 </body>
 </html>
```

Listing: 11-1—CharacterClassMethods.html

THE JAVASCRIPT FILE

```javascript
// Fig. 11.2: CharacterProcessing.js
 // String methods charAt, charCodeAt, fromCharCode,A=65
 // toLowercase and toUpperCase.
 function start()
 {
 var s = "ZEBRA";//string obj
 var s2 = "AbCdEfG";////
 var result = "";//string

 result = "<p>Character at index 0 in '" + s + "' is " +
 s.charAt( 0 )+ "</p>";
 result += "<p>Character code at index 0 in '" + s + "' is " +
 s.charCodeAt( 0 )+ "</p>";

 result += "<p>'"+String.fromCharCode( 87, 79, 82, 68 ) +
 "' contains character codes 87, 79, 82 and 68</p>";

 result += "<p>'" + s2 + "' in lowercase is '" +
 s2.toLowerCase()+ "'</p>";
 result += "<p>'" + s2 + "' in uppercase is '" +
 s2.toUpperCase()+ "'</p>";

 document.getElementById( "results" ).innerHTML = result;
 } // end function start

 window.addEventListener( "load", start, false );
```

Listing:11-2-CharacterProcessing.js

OUTPUT-LISTING-11-1- CHARACTERCLASSMETHODS.HTML

Character Processing

File | D:/Semesters/Spring-2023/LectureNotes/Programs/Lecture-11-JavaScript-Part6/Listing-11-1-CharacterClassMethods.html

Gmail COMSATS IT Center... CUOnline:: Login (345) Surah Al-Mul... (353) What is the w... (353) surah waqiah... surah kahf - Goog

Character at index 0 in 'ZEBRA' is Z

Character code at index 0 in 'ZEBRA' is 90

'WORD' contains character codes 87, 79, 82 and 68

'AbCdEfG' in lowercase is 'abcdefg'

'AbCdEfG' in uppercase is 'ABCDEFG'

Output-Listing: 11-1—CharacterClassMethods.html

SEARCHING METHODS

1. String method **indexOf()**
   - Determines the location of the first occurrence of its argument in the string used to call the method
   - If the substring is found, the index at which the first occurrence of the substring begins is returned; otherwise, -1 is returned
   - Receives an optional second argument specifying the index from which to begin the search and move to end of the string

2. String method **lastIndexOf()**
   - Determines the location of the last occurrence of its argument in the string used to call the method
   - If the substring is found, the index at which the last occurrence of the substring begins is returned; otherwise, -1 is returned
   - Receives an optional second argument specifying the index from which to begin the search and move to start of string

SEARCHING METHODS

- The example in Figs. 11-3–11-.4 demonstrates the String-object methods **indexOf()** and **lastIndexOf()** that *search* for a specified substring in a string.
- All the searches in this example are performed on a global string named letters in the script (Fig. 11-4, line 3).
- Theuser types a substring in the HTML5 form searchForm's inputField and presses the**Search** button to search for the substring in letters.
- Clicking the **Search** button calls function buttonPressed (lines 5–18) to respond to the click event and perform thesearches.
- The results of each search are displayed in the div named results.
- In the script (Fig. 11-4), lines 10–11 use String method **indexOf()** to determine the location of the *first* occurrence in string letters of the string inputField.value (i.e., the string the user typed in the inputField text field).
- If the substring is found, the index at which the first occurrence of the substring begins is returned; otherwise, –1 is returned.

SEARCHING METHODS

- Lines 12–13 use String method lastIndexOf to determine the location of the *last* occurrence in letters of the string in inputField.
- If the substring is found, the index at which the last occurrence of the substring begins is returned; otherwise, –1 is returned.
- Lines 14–15 use String method **indexOf()** to determine the location of the *first* occurrence in string letters of the string in the inputField text field after the index 12 in letters. If the substring is found, the index at which the first occurrence of the substring (starting from index 12) begins is returned; otherwise, –1 is returned.
- Lines 16–17 use String method **lastIndexOf()** to determine the location of the *last* occurrence in letters of the string in the inputField text field before index 12(means starting from index 12 in letters and moving toward the beginning of the input).
- If the substring is found, the index at which the first occurrence of the substring (if one appears before index 12) begins is returned; otherwise, –1 is returned.

THE HTML FILE

```html
<!DOCTYPE html>
<!-- Fig. 11-3: SearchingStrings.html -->
 <html><head>
    <meta charset="utf-8">
    <title>Searching Strings</title>
    <link rel ="stylesheet" type="text/css" href="style.css">
    <script src="Listing-11-4-SearchingStrings.js"></script>
 </head>
 <body>
    <form id="searchForm" action="#">

        <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>
 </html>
```

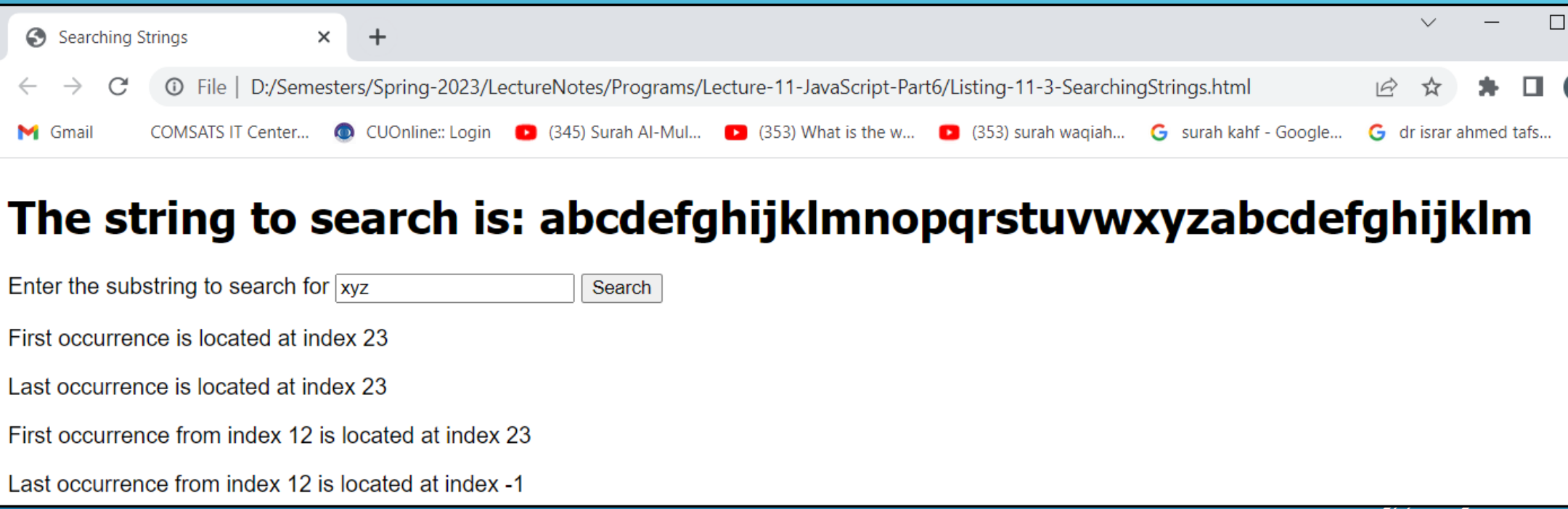
Listing: 11-3—SearchingStrings.html

THE JAVASCRIPT FILE

```javascript
// Fig. 11-4: SearchingStrings.js
 // Searching strings with indexOf and lastIndexOf.
 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>";
 } // end function buttonPressed

// register click event handler for searchButton
 function start()
 {
 var searchButton = document.getElementById( "searchButton" );
 searchButton.addEventListener( "click", buttonPressed, false );
 } // end function start

 window.addEventListener( "load", start, false );
```

Listing: 11-4—SearchingStrings.js

OUTPUT-LISTING-11-3- SEARCHINGSTRING.HTML



Output-Listing: 11-3—SearchingStrings.html

SPLITTING STRINGS AND OBTAINING SUBSTRINGS

- Breaking a string into tokens is called tokenization
- Tokens are separated from one another by delimiters, typically white-space characters such as blank, tab, newline and carriage return
  - Other characters may also be used as delimiters to separate tokens
- **String method split()**
  - Breaks a string into its component tokens
  - Argument is the delimiter string
  - Returns an array of strings containing the tokens
- **String method substring()**
  - Returns the substring from the starting index (its first argument) up to but not including the ending index (its second argument)
  - If the ending index is greater than the length of the string, the substring returned includes the characters from the starting index to the end of the original string

SPLITTING STRINGS AND OBTAINING SUBSTRINGS

- In next example the HTML5 document(Listing:11-5) displays a form containing a text field where the user types a sentence to tokenize.
- The results of the tokenization process are displayed in a div.
- The script(Listing-11-6) also demonstrates String method **substring**, which returns a portion of a string.
- The user types a sentence into the text field with id inputField and presses the **Split** button to tokenize the string.
- Function splitButtonPressed (Fig. 11.6) is called in response to the button's click event.
- In the script (Fig. 11.6), line 5 gets the value of the input field and stores it in variable inputString.
- Line 6 calls String method split to tokenize inputString.
- The argument to method split is the **delimiter string**—the string that determines the end of each token in the original string.
- In this example, the space character delimits the tokens. The delimiter string can contain multiple characters to be used as delimiters.

SPLITTING STRINGS AND OBTAINING SUBSTRINGS

- Method split() returns an array of strings containing the tokens.
- Line 11 uses Array method join to combine the tokens in array tokens and separate each token with to end one paragraph element and start a new one.
- Line 13 uses String method substring to obtain a string containing the first 10 characters of the string the user entered (still stored in inputString).
- The method returns the substring from the **starting index** (0 in this example) up to but not including the **ending index** (10 in this example).
- If the ending index is greater than the length of the string, the substring returned includes the characters from the starting index to the end of the original string.
- The result of the string concatenations in lines 9–13 is displayed in the document's results div.

THE HTML FILE

```html
<!DOCTYPE html>

<!-- Fig. 11.5: SplitAndSubString.html -->
 <!-- HTML document demonstrating String methods split and substring. -->
 <html>
 <head>
    <meta charset = "utf-8">
    <title>split and substring</title>
    <link rel = "stylesheet" type = "text/css" href ="style.css">
    <script src = "Listing-11-6-SplitAndSubString.js"></script>
 </head>
 <body>
    <form action = "#">
        <p>Enter a sentence to split into words:</p>
        <p><input id = "inputField" type = "text" size="50">
        <input id = "splitButton" type = "button" value = "Split"></p>
        <div id = "results"></div>
 </form>
 </body>
 </html>
```

Listing: 11-5—SplitAndSubString.html

THE JAVASCRIPT FILE

```javascript
// Fig. 11.6: SplitAndSubString.js
 // String object methods split and substring.
 function splitButtonPressed()
 {
 var inputString = document.getElementById( "inputField" ).value;
 var tokens = inputString.split( " " );

var results = document.getElementById( "results" );
results.innerHTML = "<p>The sentence split into words is: </p>" +
 "<p class = 'indent'>" +
tokens.join( "</p><p class = 'indent'>" )+ "</p>" +
 "<p>The first 10 characters of the input string are: </p>" +
 "<p class = 'indent'>'"+inputString.substring( 0, 10 )+"'</p>";
 } // end function splitButtonPressed

 // register click event handler for searchButton
 function start()
 {
 var splitButton = document.getElementById( "splitButton" );
 splitButton.addEventListener( "click", splitButtonPressed, false );
 } // end function start

 window.addEventListener( "load", start, false );
```


Listing: 11-6—SplitAndSubString.js

OUTPUT-11-5-SPLITANDSUBSTRING.HTML

split and substring

File | D:/Semesters/Spring-2023/LectureNotes/Programs/Lecture-11-JavaScript-Part6/Listing-11-5-SplitAndSubString.html

Gmail COMSATS IT Center... CUOnline:: Login (345) Surah Al-Mul... (353) What is the w... (353) surah waqiah... surah kahf -

Enter a sentence to split into words:

This is a string containing 7 strings Split

The sentence split into words is:

This

is

a

string

containing

7

strings

The first 10 characters of the input string are:

'This is a '

Output-Listing: 11-5—SplitAndSubString.html

DATE OBJECT

- Date object provides methods for date and time manipulations based:
  - Either on the computer's local time zone or
  - on World Time Standard's **Coordinated Universal Time** (abbreviated UTC)
- Most methods of Date Object have a local time zone and a UTC version
- Next three slides contains the list of Date object's methods

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—Part-1 of 3

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.
`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.

Date-Object Methods—Part-2 of 3

DATE OBJECT METHODS

`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.
`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`.)

Date-Object Methods—Part-3 of 3

DATE OBJECT EXAMPLE

- Next example(Listing-11-7(html file) and Listing-11-8(javascript file) demonstrating many of local time zone methods.
- The html document provides several sections in which the results are displayed.

Date Object Constructor with no-argument:

- Line-5: The Date constructor with no arguments initializes the Date object with the local computer's current date and time.

Methods toString, toLocaleString, toUTCString and valueOf

- Lines 9–12 demonstrate the methods toString, oLocaleString, toUTCString and valueOf.
- Method valueOf returns a large integer value representing the total number of milliseconds between midnight, January 1, 1970, and the date and time stored in Date object current.

DATE OBJECT EXAMPLE

Date-Object get Methods

- Lines 16–25 demonstrate the Date object's *get* methods for the local time zone. The method getFullYear() returns the year as a four-digit number.
- The method getTimeZoneOffset () returns the difference in minutes between the local time zone and UTC time (i.e., a difference of four hours in our time zone when this example was executed).

Date-Object Constructor with Arguments

- Line 28 creates a new Date object and supplies arguments to the Date constructor for year, month, date, hours, minutes, seconds and milliseconds. The hours, minutes, seconds and milliseconds arguments are all optional. If an argument is not specified, 0 is supplied in its place.
- For hours, minutes and seconds, if the argument to the right of any of these is specified, it too must be specified (e.g., if the minutes argument is specified, the hours argument must be specified; if the milliseconds argument is specified, all the arguments must be specified).

DATE OBJECT EXAMPLE

Date-Object set Methods

- Lines 33–38 demonstrate the Date-object *set* methods for the local time zone. Date objects represent the month internally as an integer from 0 to 11. These values are off by one from what you might expect (i.e., 1 for January, 2 for February, …, and 12 for December).
- When creating a Date object, you must specify 0 to indicate January, 1 to indicate February, …, and 11 to indicate December.

Date-Object parse and UTC Methods

- The Date object provides methods **Date.parse** and **Date.UTC** *that can be called without creating a new Date object*.
- Date.parse receives as its argument a string representing a date and time, and returns the number of milliseconds between midnight, January 1, 1970, and

DATE OBJECT EXAMPLE

the specified date and time. This value can be converted to a Date object with the statement

var theDate = **new** Date(*numberOfMilliseconds*);

Method parse converts the string using the following rules:

- Short dates can be specified in the form MM-DD-YY, MM-DD-YYYY, MM/DD/YY or MM/DD/YYYY. The month and day are not required to be two digits.
- Long dates that specify the complete month name (e.g., “January”), date and year can specify the month, date and year in any order.
- Text in parentheses within the string is treated as a comment and ignored. Commas and white-space characters are treated as delimiters.
- All month and day names must have at least two characters. The names are not required to be unique. If the names are identical, the name is resolved as the last match (e.g., “Ju” represents “July” rather than “June”).

DATE OBJECT EXAMPLE

- If the name of the day of the week is supplied, it's ignored.
- All standard time zones (e.g., EST for Eastern Standard Time), Coordinated Universal Time (UTC) and Greenwich Mean Time (GMT) are recognized.
- When specifying hours, minutes and seconds, separate them with colons.
- In 24-hour-clock format, "PM" should not be used for times after 12 noon.

DATE OBJECT EXAMPLE

- Date method UTC returns the number of milliseconds between midnight, January 1,1970, and the date and time specified as its arguments.
- The arguments to the UTC method include the required *year*, *month* and *date*, and the optional *hours*, *minutes*, *seconds* and *milliseconds*.
- If any of the *hours*, *minutes*, *seconds* or *milliseconds* arguments is not specified, a zero is supplied in its place.
- For the *hours*, *minutes* and *seconds* arguments, if the argument to the right of any of these arguments in the argument list is specified, that argument must also be specified (e.g., if the *minutes* argument is specified, the *hours* argument must be specified; if the *milliseconds* argument is specified, all the arguments must be specified).
- As with the result of **Date.parse**, the result of **Date.UTC** can be converted to a Date object by creating a new Date object with the result of **Date.UTC** as its argument.

THE HTML FILE

```html
<!DOCTYPE html>

<!-- Fig. 11.7: DateTime.html -->
 <!-- HTML document to demonstrate Date-object methods. -->
 <html>
 <head>
    <meta charset = "utf-8">
    <title>Date and Time Methods</title>
    <link rel = "stylesheet" type = "text/css" href = "style.css">
    <script src = "Listing-11-8-DateTime.js"></script>
 </head>
 <body>
    <h1>String representations and valueOf</h1>
    <section id = "strings"></section>
    <h1>Get methods for local time zone</h1>
    <section id = "getMethods"></section>
    <h1>Specifying arguments for a new Date</h1>
    <section id = "newArguments"></section>
    <h1>Set methods for local time zone</h1>
    <section id = "setMethods"></section>
 </body>
 </html>
```

Listing: 11-7-DateTime.html

THE JAVASCRIPT FILE

```
1  // Fig. 11.8: DateTime.js
2   // Date and time methods of the Date object.
3   function start()
4   {
5   var current = new Date();
6
7  // string-formatting methods and valueOf
8   document.getElementById( "strings" ).innerHTML =
9   "<p>toString: "+ current.toString()+"</p>" +
10  "<p>toLocaleString: "+ current.toLocaleString()+"</p>" +
11 "<p>toUTCString: "+current.toUTCString() +"</p>" +
12 "<p>valueOf: "+current.valueOf() +"</p>";
13
14 // get methods
15  document.getElementById( "getMethods" ).innerHTML =
16  "<p>getDate: "+current.getDate() +"</p>" +
17  "<p>getDay: "+current.getDay() +"</p>" +
18  "<p>getMonth: "+current.getMonth() +"</p>" +
19  "<p>getFullYear: "+ current.getFullYear()+"</p>" +
20 "<p>getTime: "+current.getTime() +"</p>" +
21  "<p>getHours: "+ current.getHours()+"</p>" +
22  "<p>getMinutes: "+current.getMinutes() +"</p>" +
23  "<p>getSeconds: "+current.getSeconds() +"</p>" +
24  "<p>getMilliseconds: "+current.getMilliseconds() +"</p>" +
25  "<p>getTimezoneOffset: "+current.getTimezoneOffset() +"</p>";
```

Listing: 11-8-DateTime.js

THE JAVASCRIPT FILE

```
26
27  // creating a Date
28  var anotherDate = new Date( 2011, 2, 18, 1, 5, 0, 0 );
29   document.getElementById( "newArguments" ).innerHTML =
30   "<p>Date: " + anotherDate + "</p>";
31
32   // set methods
33  anotherDate.setDate( 31 );
34  anotherDate.setMonth( 11 );
35  anotherDate.setFullYear( 2011 );
36  anotherDate.setHours( 23 );
37  anotherDate.setMinutes( 59 );
38  anotherDate.setSeconds( 59 );
39  document.getElementById( "setMethods" ).innerHTML =
40   "<p>Modified date: " + anotherDate + "</p>";
41   } // end function start
42
43  window.addEventListener( "load", start, false );
```

Listing: 11-8-DateTime.js

THE OUTPUT HTML FILE

Output-Listing: 11-7-DateTime.html

Date and Time Methods

File | D:/Semesters/Spring-2023/LectureNotes/Programs/

Gmail COMSATS IT Center... CUOnline:: Login (345) Surah Al-Mul...

String representations and valueOf

toString: Tue May 09 2023 23:00:29 GMT+0500 (Pakistan Standard Time)

toLocaleString: 5/9/2023, 11:00:29 PM

toUTCString: Tue, 09 May 2023 18:00:29 GMT

valueOf: 1683655229994

Get methods for local time zone

getDate: 9

getDay: 2

getMonth: 4

getFullYear: 2023

getTime: 1683655229994

getHours: 23

getMinutes: 0

getSeconds: 29

getMilliseconds: 994

getTimezoneOffset: -300

Specifying arguments for a new Date

Date: Fri Mar 18 2011 01:05:00 GMT+0500 (Pakistan Standard Time)

Set methods for local time zone

Modified date: Sat Dec 31 2011 23:59:59 GMT+0500 (Pakistan Standard Time)

BOOLEAN OBJECT

- 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.
- JavaScript programmers can create Boolean objects explicitly with the statement

```
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.

BOOLEAN OBJECT

- Otherwise, the newBoolean object contains true. Following Figure summarizes the methods of the Boolean object.

Method	Description
`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`-object methods.

NUMBER OBJECT

- 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.
- Although you can explicitly create Number objects, normally the JavaScript interpreter creates them as needed. Following figure summarizes the methods and properties of the Number object.

NUMBER OBJECT METHODS

Method or property	Description
`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`.

`Number`-object methods and properties.

DOCUMENT OBJECT METHODS

- `document` object
  - Provided by the browser and allows JavaScript code to manipulate the current document in the browser

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 methods.