Chapter 12 - JavaScript: Objects

Outline

12.3 `Math` Object

Method	Description	Example
`abs( x )`	absolute value of `x`	`abs( 7.2 )` is 7.2 `abs( 0.0 )` is 0.0 `abs( -5.6 )` is 5.6
`ceil( x )`	rounds `x` to the smallest integer not less than `x`	`ceil( 9.2 )` is 10.0 `ceil( -9.8 )` is -9.0
`cos( x )`	trigonometric cosine of `x` (`x` in radians)	`cos( 0.0 )` is 1.0
`exp( x )`	exponential method e^x	`exp( 1.0 )` is 2.71828 `exp( 2.0 )` is 7.38906
`floor( x )`	rounds `x` to the largest integer not greater than `x`	`floor( 9.2 )` is 9.0 `floor( -9.8 )` is -10.0
`log( x )`	natural logarithm of `x` (base e)	`log( 2.718282 )` is 1.0 `log( 7.389056 )` is 2.0
`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( 2.0, 7.0 )` is 128.0 `pow( 9.0, .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( 0.0 )` is 0.0
`sqrt( x )`	square root of `x`	`sqrt( 900.0 )` is 30.0 `sqrt( 9.0 )` is 3.0
`tan( x )`	trigonometric tangent of `x` (`x` in radians)	`tan( 0.0 )` is 0.0

Fig. 12.1 Commonly used `Math` object methods.

12.3 `Math` Object

Constant	Description	Value
`Math.E`	Euler's constant.	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 Euler's constant.	Approximately 1.442.
`Math.LOG10E`	Base 10 logarithm of Euler's constant.	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.

Fig. 12.2 Properties of the `Math` object.

12.4.2 Methods of the `String` Object

Method	Description
`charAt( index )`	Returns a string containing the character at the specified *index*. If there is no character at that *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*. If there is no character at that *index*, `charCodeAt` returns `NaN`.
`concat( string )`	Concatenates its argument to the end of the string that invokes the method. The string invoking this method is not modified; rather 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.
`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 end of the source string.
`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).
`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).

Fig. 12.3 Some methods of the `String` object.

12.4.2 Methods of the `String` Object

Method	Description
`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.
`toString()`	Returns the same string as the source string.
`valueOf()`	Returns the same string as the source string.
Methods that generate XHTML tags	
`anchor(` ***name*** **`)`**	Wraps the source string in an anchor element (<a>) with *name* as the anchor name.
`blink()`	Wraps the source string in a <blink></blink> element.
`fixed()`	Wraps the source string in a <tt></tt> element.
`link(` ***url*** **`)`**	Wraps the source string in an anchor element (<a>) with *url* as the hyperlink location.
`strike()`	Wraps the source string in a <strike></strike> element.
`sub()`	Wraps the source string in a element.
`sup()`	Wraps the source string in a element.

Fig. 12.3 Some methods of the **`String`** object.

Outline

CharacterProcessing.html

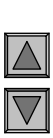
```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Fig. 12.4: CharacterProcessing.html -->
6   <!-- Character Processing Methods        -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>Character Processing Meth
11
12        <script type = "text/javascript">
13           <!--
14           var s = "ZEBRA";
15           var s2 = "AbCdEfG";
16
17           document.writeln( "<p>Character at inde
18              s + "' is " + s.charAt( 0 ) );
19           document.writeln( "<br />Character code at index 0 in '"
20              + s + "' is " + s.charCodeAt( 0 ) + "</p>" );
21
22           document.writeln( "<p>'" +
23              String.fromCharCode( 87, 79, 82, 68 ) +
24              "' contains character codes 87, 79, 82 and 68</p>" )
25
26           document.writeln( "<p
27              s2.toLowerCase() +
28           document.writeln( "<b
29              + s2.toUpperCase() + "</p>" );
30           // -->
31        </script>
32
33     </head><body></body>
34  </html>
```

Method **`charAt`** returns a string containing the character at the specified index (**`0`** in this example).

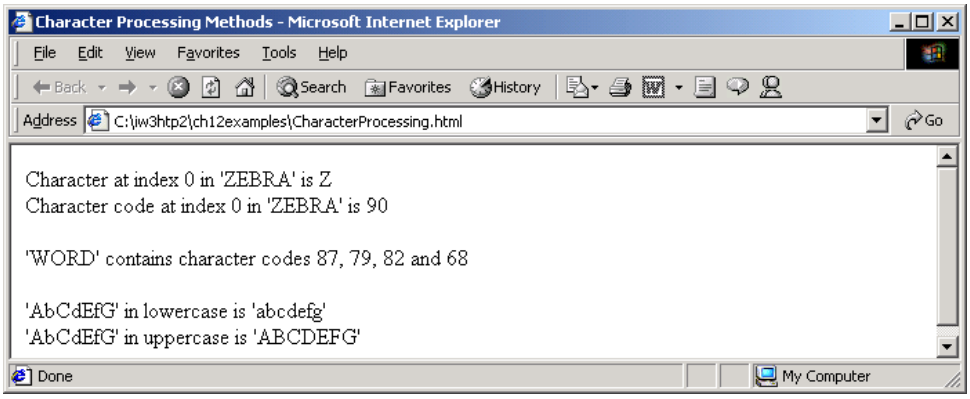
Method **`charCodeAt`** returns the Unicode value of the character at the specified index (**`0`** in this example).

Method **`fromCharCode`** takes a comma-separated list of Unicode values and builds a string containing the character representation of those Unicode values.

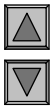
Methods **`toLowerCase`** and **`toUpperCase`** display versions of **`String s2`** in all lowercase and all upper case letters, respectively.



Outline



Program Output



Outline

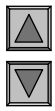
SearchingStrings.html

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Fig. 12.5: SearchingStrings.html -->
6   <!-- Searching Strings                -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>
11           Searching Strings with ind
12        </title>
13
14        <script type = "text/javascri
15           <!--
16           var letters = "abcdefghijklmnopqrstuvwxyzabcdefghijklm";
17
18           function buttonPressed()
19           {
20              searchForm.first.value =
21                 letters.indexOf( search
22              searchForm.last.value =
23                 letters.lastIndexOf( searchForm.inputVal.value );
24              searchForm.first12.value =
25                 letters.indexOf( searchForm.inputVal.value, 12 );
26              searchForm.last12.value =
27                 letters.lastIndexOf(
28                    searchForm.inputVal.value, 12 );
29           }
30           // -->
31        </script>
32
33     </head>
```

Method **indexOf** determines the first occurrence in the string **letters** of the string **searchForm.inputVal.value**.

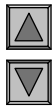
Method **lastIndexOf** determines the location of the last occurrence in **letters** of the string in text field **inputVal**.

```
34      <body>
35         <form name = "searchForm" action = "">
36            <h1>The string to search is:<br />
37                abcdefghijklmnopqrstuvwxyzabcdefghijklm</h1>
38            <p>Enter substring to search for
39            <input name = "inputVal" type = "text" />
40            <input name = "search" type = "button" value = "Search"
41                   onclick = "buttonPressed()" /><br /></p>
42
43            <p>First occurrence located at index
44            <input name = "first" type = "text" size = "5" />
45            <br />Last occurrence located at index
46            <input name = "last" type = "text" size = "5" />
47            <br />First occurrence from index 12 located at index
48            <input name = "first12" type = "text" size = "5" />
49            <br />Last occurrence from index 12 located at index
50            <input name = "last12" type = "text" size = "5" /></p>
51         </form>
52      </body>
53   </html>
```



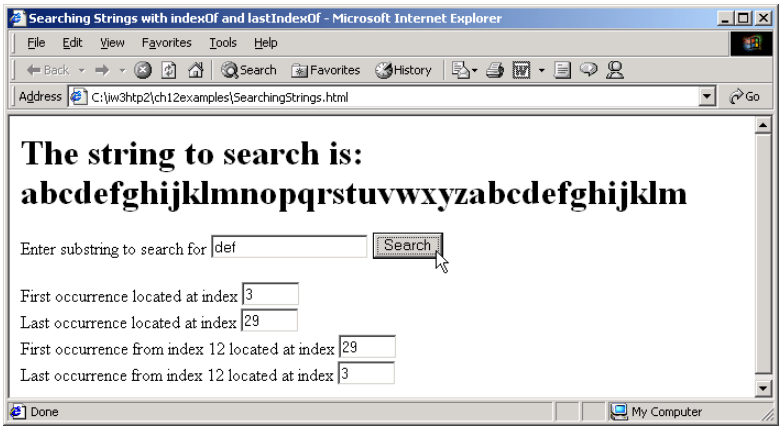
Outline

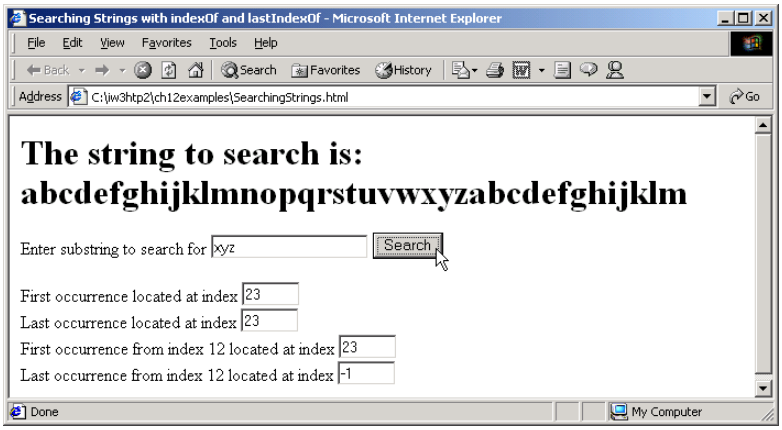
SearchingStrings.html

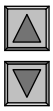


Outline

Program Output







Outline

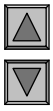
SplitAndSubString.html

```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5    <!-- Fig. 12.6: SplitAndSubString.html -->
6    <!-- String Method split and substring -->
7
8    <html xmlns = "http://www.w3.org/1999
9       <head>
10         <title>String Method split and
11
12         <script type = "text/javascript">
13            <!--
14            function splitButtonPressed()
15            {
16               var strings = myForm.inputVal.value.split( " " );
17               myForm.output.value = strings.join( "\n" );
18
19               myForm.outputSubstring.v
20                  myForm.inputVal.value
21            }
22            // -->
23         </script>
24      </head>
25
26      <body>
27         <form name
28            <p>Enter a sentence to split into words<br />
29            <input name = "inputVal" type = "text" size = "40" />
30            <input name = "splitButton" type = "button" value =
31                  "Split" onclick = "splitButtonPressed()" /></p>
32
33            <p>The sentence split into words is<br />
34            <textarea name = "output" rows = "8" cols = "34">
35            </textarea></p>
```

Method **split** tokenizes the contents of text field **inputVal**.

The argument to method **split** is the delimiter string.

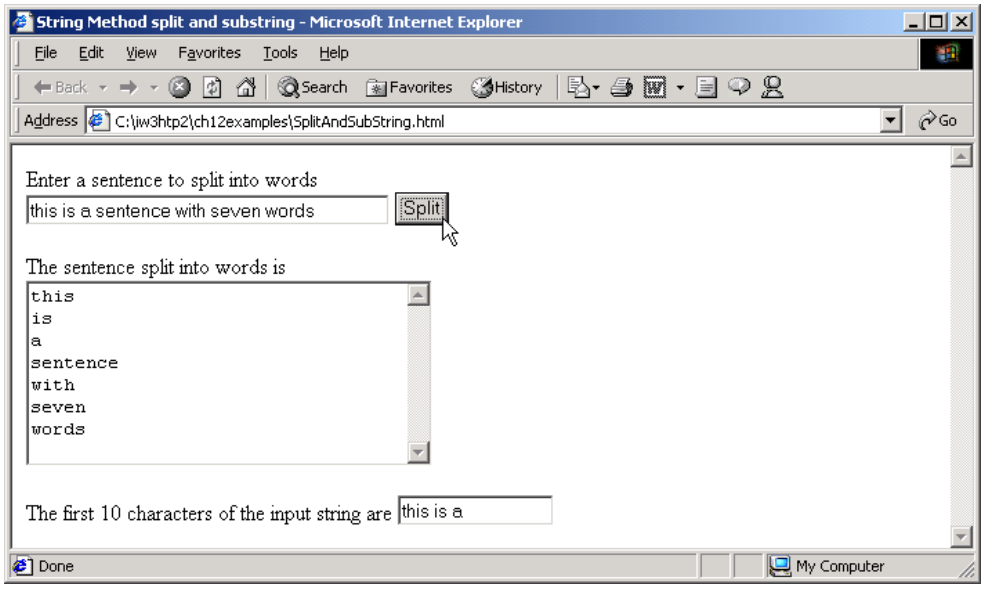
Method **subString** obtains a string containing the first 10 characters of the string the user input in text field **inputVal**.



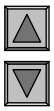
Outline

SplitAndSubString.html

```
36
37            <p>The first 10 characters of the input string are
38            <input name = "outputSubstring" type = "text"
39                  size = "15" /></p>
40         </form>
41      </body>
42   </html>
```



Program Output



Outline

MarkupMethods.html

```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5    <!-- Fig. 12.7: MarkupMethods.html                -->
6    <!-- XHTML markup methods of the String object -->
7
8    <html xmlns = "http://www.w3.org/1999/xhtml">
                                                  ds of the Stri
                                                  script">
                                                  is an anchor",
                                                  s blinking tex
                                                  is monospaced t
17                linkText = "Click here to go to anchorText",
18                strikeText = "This is strike out text",
19                subText = "subscript",
20                supText = "superscript";
21
22             document.writeln( anchorText.anchor( "top" ) );
23             document.writeln( "<br />" + blinkText.blink() );
24             document.writeln( "<br />" + fixedText.fixed() );
25             document.writeln( "<br />" + strikeText.strike() );
26             document.writeln(
27                "<br />This is text with a " + subText.sub() );
28             document.writeln(
29                "<br />This is text with a " + supText.sup() );
30             document.writeln(
31                "<br />" + linkText.link( "#top" ) );
32             // -->
```

Method **strike** displays text with

Method **sub** creates subscript

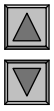
Method **sup** creates superscript

The **link** method creates a hyperlink.

Method **anchor** marks up the text as an anchor.

Method **blink** makes the string blink in the

Method **fixed** displays txt in a fixed-width font.

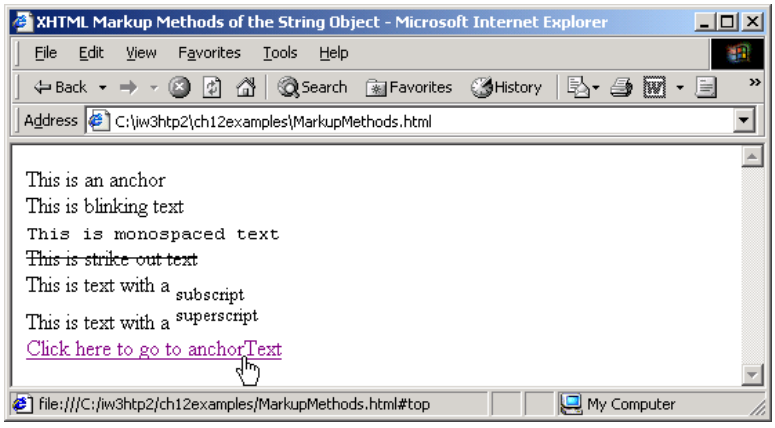


Outline

MarkupMethods.html

```
33          </script>
34
35       </head><body></body>
36    </html>
```

Program Output



12.5 Date Object

Method	Description
`getDate()` `getUTCDate()`	Returns a number from 1 to 31 representing the day of the month in local time or UTC, respectively.
`getDay()` `getUTCDay()`	Returns a number from 0 (Sunday) to 6 (Saturday) representing the day of the week in local time or UTC, respectively.
`getFullYear()` `getUTCFullYear()`	Returns the year as a four-digit number in local time or UTC, respectively.
`getHours()` `getUTCHours()`	Returns a number from 0 to 23 representing hours since midnight in local time or UTC, respectively.
`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, respectively.
`getMonth()` `getUTCMonth()`	Returns a number from 0 (January) to 11 (December) representing the month in local time or UTC, respectively.
`getSeconds()` `getUTCSeconds()`	Returns a number from 0 to 59 representing the seconds for the time in local time or UTC, respectively.
`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—previously known as Greenwich Mean Time (GMT).
`setDate(` *val* `)` `setUTCDate(` *val* `)`	Sets the day of the month (1 to 31) in local time or UTC, respectively.

Fig. 12.8 Methods of the `Date` object.

12.5 Date Object

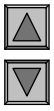
Method	Description
`setFullYear(` *y*`,` *m*`,` *d* `)` `setUTCFullYear(` *y*`,` *m*`,` *d* `)`	Sets the year in local time or UTC, respectively. The second and third arguments representing the month and the date are optional. If an optional argument is not specified, the current value in the `Date` object is used.
`setHours(` *h*`,` *m*`,` *s*`,` *ms* `)` `setUTCHours(` *h*`,` *m*`,` *s*`,` *ms* `)`	Sets the hour in local time or UTC, respectively. 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, respectively.
`setMinutes(` *m*`,` *s*`,` *ms* `)` `setUTCMinutes(` *m*`,` *s*`,` *ms* `)`	Sets the minute in local time or UTC, respectively. 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, respectively. 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 second in local time or UTC, respectively. The second argument representing the milliseconds is optional. If this argument is not specified, the current millisecond value in the `Date` object is used.

Fig. 12.8 Methods of the `Date` object.

12.5 `Date` Object

Method	Description
`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, 2001 at 3:42:22 PM is represented as *09/13/01 15:47:22* in the United States and *13/09/01 15:47:22* in Europe.
`toUTCString()`	Returns a string representation of the date and time in the form: *19 Sep 2001 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 19 15:47:22 EDT 2001* in the United States).
`valueOf()`	The time in number of milliseconds since midnight, January 1, 1970.

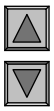
Fig. 12.8 Methods of the **`Date`** object.



Outline

`DateTime.html`

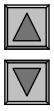
```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5    <!-- Fig. 12.9: DateTime.html -->
6    <!-- Date and Time Methods    -->
7
8    <html xmlns = "http://www.w3.org/1999/xhtml">
9       <head>
10         <title>Date and Time Methods</title>
11
12         <script type = "text/javascript">
13            <!--
14            var current = new Date();
15
16            document.writeln(
17               "<h1>String representations and valueOf</h1>" );
18            document.writeln( "toString: " + current.toString() +
19               "<br />toLocaleString: " + current.toLocaleString() +
20               "<br />toUTCString: " + current.toUTCString() +
21               "<br />valueOf: " + current.valueOf() );
22
23            document.writeln(
24               "<h1>Get methods for local time zone</h1>" );
25            document.writeln( "getDate: " + current.getDate() +
26               "<br />getDay: " + current.getDay() +
27               "<br />getMonth: " + current.getMonth() +
28               "<br />getFullYear: " + current.getFullYear() +
29               "<br />getTime: " + current.getTime() +
30               "<br />getHours: " + current.getHours() +
31               "<br />getMinutes: " + current.getMinutes() +
32               "<br />getSeconds: " + current.getSeconds() +
33               "<br />getMilliseconds: " +
```



Outline

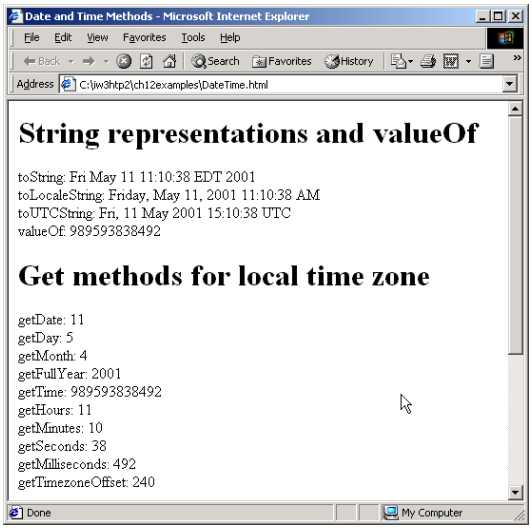
DateTime.html

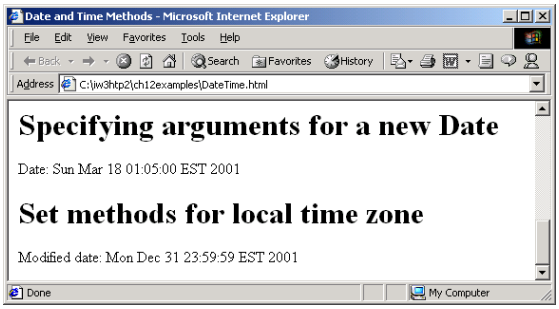
```
34              current.getMilliseconds() +
35              "<br />getTimezoneOffset: " +
36              current.getTimezoneOffset() );
37
38           document.writeln(
39              "<h1>Specifying arguments for a new Date</h1>" );
40           var anotherDate = new Date( 2001, 2, 18, 1, 5, 0, 0 );
41           document.writeln( "Date: " + anotherDate );
42
43           document.writeln(
44              "<h1>Set methods for local time zone</h1>" );
45           anotherDate.setDate( 31 );
46           anotherDate.setMonth( 11 );
47           anotherDate.setFullYear( 2001 );
48           anotherDate.setHours( 23 );
49           anotherDate.setMinutes( 59 );
50           anotherDate.setSeconds( 59 );
51           document.writeln( "Modified date: " + anotherDate );
52           // -->
53        </script>
54
55     </head><body></body>
56  </html>
```

Outline

Program Output





12.6 Boolean and Number Objects

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.

Fig. 12.10 Methods of the **Boolean** object.

12.6 Boolean and Number Objects

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. For example, radix 2 results in the binary representation of the number, 8 results in the octal representation, 10 results in the decimal representation and 16 results in the hexadecimal representation. See Appendix D "Number Systems" for a review of the binary, octal, decimal and hexadecimal number systems.
`valueOf()`	Returns the numeric value.
`Number.MAX_VALUE`	This property represents the largest value that can be stored in a JavaScript program—approximately 1.79E+308
`Number.MIN_VALUE`	This property represents the smallest value that can be stored in a JavaScript program—approximately 2.22E–308
`Number.NaN`	This property represents *not a number*—a value returned from arithmetic expressions that do not result in a number (e.g., the expression `parseInt("hello")` cannot convert the string `"hello"` into 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`	This property represents a value less than `-Number.MAX_VALUE`.
`Number.POSITIVE_INFINITY`	This property represents a value greater than `Number.MAX_VALUE`.

Fig. 12.11 Methods and properties of the **Number** object.