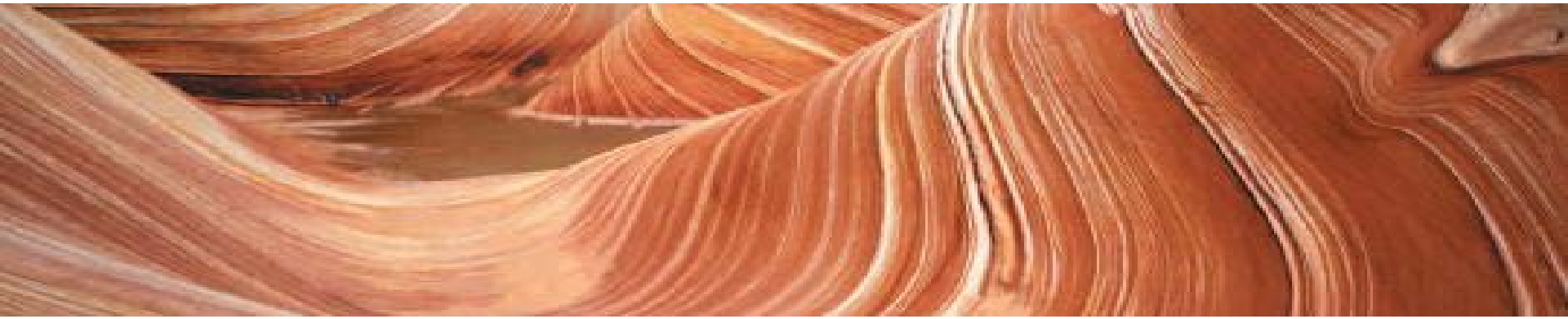
Tutorial 11

Working with Operators and Expressions

HTML, CSS, and Dynamic HTML

5TH EDITION



Carey, P. (2013). New perspectives on HTML, CSS, and Dynamic HTML: Comprehensive (5th ed.). Australia ; United Kingdom: Course Technology Cengage Learning.

Objectives

- Insert a value into a Web form field
- Work with event handlers
- Create and work with Date objects
- Extract information from Date objects
- Work with arithmetic and unary operators

Objectives

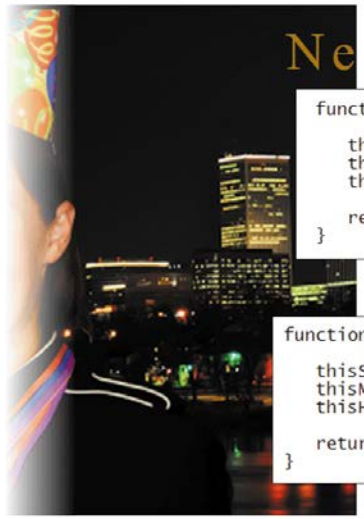
- Understand the properties and methods of the Math object
- Control how JavaScript works with numeric values
- Explore conditional, comparison, and logical operators
- Run time-delayed and timed-interval commands

Using DATE Objects and Methods

```
<body onload="NYClock()">
   <form name="clockform" id="clockform" action="">

   <header>
      <h1>New Year's Eve<br />Bash</h1>
```

The onload event handler specifies a JavaScript statement to run when the page is loaded by a browser.



value of the dateNow element

value of the timeNow element

The getDate() method returns the day of the month, the getMonth() method returns the month number starting with 0 for January, and the getFullYear() method returns the four-digit year value.

```
function showDate(dateObj) {

   thisDate = dateObj.getDate();
   thisMonth = dateObj.getMonth() + 1;
   thisYear = dateObj.getFullYear();

   return thisMonth + "/" + thisDate + "/" + thisYear;
}
```

```
function showTime(dateObj) {

   thisSecond=dateObj.getSeconds();
   thisMinute=dateObj.getMinutes();
   thisHour=dateObj.getHours();

   return thisHour + ":" + thisMinute + ":" + thisSecond;
}
```

The getSeconds() method returns the seconds value, the getMinutes() method returns the minutes value, and the getHours() method returns the hours value in 24-hour time.

The new Date() object constructor creates a Date object containing information about a specified date and time.

```
function NYClock() {

   var today = new Date("September 14, 2015 17:23:45");

   document.clockform.dateNow.value = showDate(today);
   document.clockform.timeNow.value = showTime(today);
}
```

To set or retrieve the value of a Web form element, use the object reference

document.form.element.value

where *form* is the name of the Web form and *element* is the name of the element.

Understanding Events and Event Handlers

- An **event** is an action that occurs within a Web browser or Web document.
- An **event handler** is a statement that tells browsers what code to run in response to the specified event.
  - Script

```
<input type = "button" value = "Total Cost" onclick = "calcTotal()" />
```

Inserting an Event Handler as an HTML Attribute

- To insert an event handler as an HTML attribute, apply the HTML code

```
<element onevent = "script">
...
```

where *element* is the Web page element, *event* is the name of an event associated with the element, and *script* is a command to be run in response to the event.

JavaScript Event Handlers

Figure 11-4 JavaScript event handlers

Category	Event Handler	Occurs
Window and document event handlers	`onload`	The browser has completed loading the document.
	`onunload`	The browser has completed unloading the document.
	`onerror`	An error has occurred in the JavaScript program.
	`onmove`	The user has moved the browser window.
	`onresize`	The user has resized the browser window.
	`onscroll`	The user has moved the scroll bar within the browser window.
Form event handlers	`onfocus`	The user has entered an input field.
	`onblur`	The user has exited an input field.
	`onchange`	The content of an input field has changed.
	`onselect`	The user has selected text within an input or text area field.
	`onsubmit`	The user has submitted the Web form.
	`onreset`	The user has reset the Web form.
Mouse and key-board event handlers	`onkeydown`	The user has pressed a key.
	`onkeypress`	The user has pressed and released a key.
	`onclick`	The user has clicked the mouse button.
	`ondblclick`	The user has double-clicked the mouse button.
	`onmousedown`	The user has pressed the mouse button.
	`onmouseup`	The user has released the mouse button.
	`onmousemove`	The user has moved the pointer while within the object's boundaries.
	`onmouseout`	The user has moved the pointer beyond the object's boundaries.
	`onmouseover`	The user has moved the pointer into the object's boundaries.

Working with `Date` Objects

- **Date objects** contain information about a specified date and time
- To store a date and time in a variable, use the JavaScript command

```
variable = new Date("month day, year
  hours:minutes:seconds")
```

where variable is the name of the variable that contains the date object, and month, day, year, hours, minutes, and seconds indicate the date and time to be stored in the object.

- Time values are entered in 24-hour time

Working with Date Objects

- To store a date and time using numeric values, use the JavaScript command

  ```
  variable = new Date(year, month, day, hours, minutes, seconds)
  ```

  where year, month, day, hours, minutes, and seconds are the values of the date and time, and the month value is an integer from 0 to 11, where 0 = January, 1 = February, and so forth.

- Time values are entered in 24-hour time.

Working with Date Objects

- To create a date object containing the current date and time, use the following object constructor:

```
new Date()
```

Working with Date Objects

- **Date methods** can be used to retrieve information from a date object or to change a date object's value

Figure 11-8 Retrieving values from a Date object

```
var thisDate = new Date("July 22, 2015 14:35:28")
```

Method	Retrieves	Value
`thisDate.getSeconds()`	the seconds value	28
`thisDate.getMinutes()`	the minutes value	35
`thisDate.getHours()`	the hours value (in 24-hour time)	14
`thisDate.getDate()`	the day of the month value	22
`thisDate.getDay()`	the day of the week value (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday)	3
`thisDate.getMonth()`	the month value (0 = January, 1 = February, 2 = March, etc.)	6
`thisDate.getFullYear()`	the four-digit year value	2015
`thisDate.getTime()`	the time value, as expressed in milliseconds, since January 1, 1970 at 12:00 a.m.	1,437,593,728,000

Working with Date Objects

Figure 11-9 Setting values within a Date object

Method	Sets
dateObject.setSeconds(*value*)	the seconds value of *dateObject* to *value*
dateObject.setMinutes(*value*)	the minutes value of *dateObject* to *value*
dateObject.setHours(*value*)	the hours value of *dateObject* to *value*
dateObject.setDate(*value*)	the day of the month value of *dateObject* to *value*
dateObject.setMonth(*value*)	the month number of *dateObject* (0 = January, 1 = February, etc.) to *value*
dateObject.setFullYear(*value*)	the four-digit year value of *dateObject* to *value*
dateObject.setTime(*value*)	the time of *dateObject* in milliseconds since January 1, 1970 at 12:00 a.m. to *value*

Using Operators and Expressions

The **setInterval() method** is used to repeat a JavaScript command at a specified number of milliseconds.

```
<body onload="setInterval('NYClock()',1000) " >
   <form name="clockform" id="clockform" action="">

   <header>
      <h1>New Year's Eve<br />Bash</h1>
```

A **conditional operator** compares one value to another using a logical operator. If the condition is true, the conditional operator returns the first value; otherwise, it returns the second value.

```
function showTime(dateObj) {

   thisSecond=dateObj.getSeconds();
   thisMinute=dateObj.getMinutes();
   thisHour=dateObj.getHours();

   var ampm = (thisHour < 12) ? " a.m." : " p.m.";

   thisHour = (thisHour > 12) ? thisHour - 12 : thisHour;
   thisHour = (thisHour == 0) ? 12 : thisHour;

   thisMinute = (thisMinute < 10) ? "0" + thisMinute : thisMinute;
   thisSecond = (thisSecond < 10) ? "0" + thisSecond : thisSecond;

   return thisHour + ":" + thisMinute + ":" + thisSecond + ampm;
}
```

When no date or time is specified, the `new Date()` object constructor returns a `Date` object with the current date and time.

```
function NYClock() {

   var today = new Date();

   document.clockform.dateNow.value = showDate(today);
   document.clockform.timeNow.value = showTime(today);

   var days = calcDays(today);
   document.clockform.daysLeft.value = Math.floor(days) + " day(s)";

   var hours = (days - Math.floor(days))*24;
   document.clockform.hrsLeft.value = Math.floor(hours) + " hour(s)";

   var minutes = (hours - Math.floor(hours))*60;
   document.clockform.minsLeft.value = Math.floor(minutes) + " minute(s)";

   var seconds = (minutes - Math.floor(minutes))*60;
   document.clockform.secsLeft.value = Math.floor(seconds) + " second(s)";

}
```

The **Math object** is used to create objects for mathematical functions; the `floor()` method is a mathematical method used to round a number down to the next closest integer.

altafulla/Shutterstock.com; jbdphotography/Shutterstock.com

Working with Operators and Operands

- An **operator** is a symbol used to act upon an item or a variable within a JavaScript expression.
- The variables or expressions that operators act upon are called **operands.**

Figure 11-13 Arithmetic operators

Operator	Description	Example

+	Combines or adds two items	`Men = 20;`
		`Women = 25;`
		`Total = Men + Women;`

–	Subtracts one item from another	`Income = 1000;`
		`Expense = 750;`
		`Profit = Income - Expense;`

*	Multiplies two items	`Width = 50;`
		`Length = 20;`
		`Area = Width * Length;`

/	Divides one item by another	`Persons = 50;`
		`Cost = 200;`
		`CostPerPerson = Cost / Persons;`

%	Calculates the remainder after dividing one value by another	`TotalEggs = 64;`
		`CartonSize = 12;`
		`EggsLeft = TotalEggs % CartonSize;`

Working with Operators and Operands

- **Binary operators** work with two operands in an expression.
- **Unary operators** work with one operand.
- **Increment operators** increase the value of the operand by 1.
  - x++;
- **Decrement operators** decrease the value of the operand by 1.
  - x--;
- **Negation operators** change the sign of an item's value

Working with Operators and Operands

- **Assignment operators** assign values to items.
  - Equal sign (=)
    - x = x + y
  - A common use of the += operator is to concatenate strings or add a value to an existing value of a variable

```
quote = "To be or not to be: ";
quote += "That is the question. ";
quote += "Whether 'tis nobler in the mind to suffer ";
quote += "the slings and arrows of outrageous fortune, ";
quote += "Or to take arms against a sea of troubles";
quote += "And by opposing end them.";
...
```

Working with Operators and Operands

Figure 11-15 Assignment operators

Operator	Description	Example	Equivalent To
=	Assigns the value of the expression on the right to the expression on the left	`x = y`	`x = y`
+=	Adds two expressions and assigns the value to a variable	`x += y`	`x = x + y`
–=	Subtracts the expression on the right from the expression on the left and assigns the value to a variable	`x -= y`	`x = x - y`
*=	Multiplies two expressions and assigns the value to a variable	`x *= y`	`x = x * y`
%=	Calculates the remainder from dividing the expression on the left by the expression on the right and assigns the value to a variable	`x %= y`	`x = x % y`

Working with the Math Object

- The **Math object** is an object that can be used for performing mathematical tasks and storing mathematical values.
  - **Math methods** store functions used for performing advanced calculations and mathematical operations such as:
    - Generating random numbers
    - Extracting square roots
    - Calculating trigonometric values

Working with the Math Object

Figure 11-20 Methods of the Math object

`Math` Method	Returns
`Math.abs(x)`	the absolute value of x
`Math.acos(x)`	the arc cosine of x in radians
`Math.asin(x)`	the arc sine of x in radians
`Math.atan(x)`	the arc tangent of x in radians
`Math.atan2(x, y)`	the angle between the x-axis and the point (x, y)
`Math.ceil(x)`	x rounded up to the next highest integer
`Math.cos(x)`	the cosine of x
`Math.exp(x)`	e^x
`Math.floor(x)`	x rounded down to the next lowest integer
`Math.log(x)`	the natural logarithm of x
`Math.max(x, y)`	the larger of x and y
`Math.min(x, y)`	the smaller of x and y
`Math.pow(x, y)`	x^y
`Math.random()`	a random number between 0 and 1
`Math.round(x)`	x rounded to the nearest integer
`Math.sin(x)`	the sine of x
`Math.sqrt(x)`	the square root of x
`Math.tan(x)`	the tangent of x

Working with the Math Object

- The Math object also stores numeric values for mathematical constants.

Figure 11-21 Properties of the Math object

Math Constant	Description
Math.E	The natural logarithm base, *e* (approximately 2.7183)
Math.LN10	The natural logarithm of 10 (approximately 2.3026)
Math.LN2	The natural logarithm of 2 (approximately 0.6931)
Math.LOG10E	The base 10 logarithm of *e* (approximately 0.4343)
Math.LOG2E	The base 2 logarithm of *e* (approximately 1.4427)
Math.PI	The value π (approximately 3.1416)
Math.SQRT1_2	The value of 1 divided by the square root of 2 (approximately 0.7071)
Math.SQRT2	The value of the square root of 2 (approximately 1.4142)

Controlling How JavaScript Works with Numeric Values

- Some mathematical operations can return results that are not numeric values.
  - You cannot divide a number by a text string
    - Returns “NaN”
  - You cannot divide a number by zero
    - Returns “Infinity”
- The isNaN function is a Boolean function that tests a value to see whether it is numeric or not.
- The isFinite function is a Boolean function that checks for the value of Infinity.

Controlling How JavaScript Works with Numeric Values

Figure 11-28 Numeric functions and methods

Function or Method	Description
`isFinite(value)`	Returns a Boolean value indicating whether *value* is finite and a legal number
`isNaN(value)`	Returns a Boolean value, which has the value true if *value* is not a number
`parseFloat(string)`	Extracts the first numeric value from a text string, *string*
`parseInt(string)`	Extracts the first integer value from a text string, *string*
`value.toExponential(n)`	Returns a text string displaying *value* in exponential notation with *n* digits to the right of the decimal point
`value.toFixed(n)`	Returns a text string displaying *value* to *n* decimal places
`value.toPrecision(n)`	Returns a text string displaying *value* to *n* significant digits either to the left or to the right of the decimal point

Working with Conditional, Comparison, and Logical Operators

- A **conditional operator** is a ternary operator that tests whether a certain condition is true.
- A **comparison operator** is an operator that compares the value of one expression to another.

Figure 11-29 Comparison operators

Operator	Definition	Expression	Description
==	equal to	x == y	Returns true if x equals y
!=	not equal to	x != y	Returns true if x does not equal y
>	greater than	x > y	Returns true if x is greater than y
<	less than	x < y	Returns true if x is less than y
>=	greater than or equal to	x >= y	Returns true if x is greater than or equal to y
<=	less than or equal to	x <= y	Returns true if x is less than or equal to y

Working with Conditional, Comparison, and Logical Operators

- **Logical operators** allow you to connect several expressions

Figure 11-30 Logical operators

Operator	Definition	Expression	Description
`&&`	and	`(x == 20) && (y == 25)`	Returns `true` if x equals 20 and y equals 25
`||`	or	`(x == 20) || (y < 10)`	Returns `true` if x equals 20 or y is less than 10
`!`	not	`!(x == 20)`	Returns `true` if x does not equal 20

Running Timed Commands

- A **time-delayed command** is a JavaScript command that is run after a specified amount of time has passed.
  - setTimeout("command", delay);
  - clearTimeout();
- A **time-interval command** instructs the browser to run the same command repeatedly at specified intervals.
  - setInterval ("command", interval);
  - clearInterval();