Tutorial 12

Working with Arrays, Loops, and Conditional Statements

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

- Create an array
- Populate and reference values from an array
- Work with array methods
- Repeat commands using for loops
- Loop through the contents of an array
- Repeat commands with while loops

Objectives

- Work with ECMASScript5 array methods
- Create conditional commands with if and if else statements
- Use arrays, loops, and conditional statements to create a table
- Interrupt loops with break, continue, and label commands

Creating Arrays


```
function calendar() {

   // Date that the monthly calendar is based on
   var calDate = new Date("July 6, 2015")

   document.write("<table id = 'calendar_table'>");

   // Write the header row of the calendar table
   writeCalTitle(calDate);

   document.write("</table>");
}
```

The writeCalTitle() function writes the calendar title.

Arrays can be created using the object constructor

var *arrayName* = new Array(values)
or using an **array literal**

var *arrayName* = [values];

An **array** is a collection of values organized under a single name.

```
function writeCalTitle(calendarDay) {

   /* The calendarDay parameter contains a date object
      that the calendar is based upon */

   // monthName contains array of month names

   var monthName = ["January", "February", "March",
      "April", "May", "June", "July", "August", "September",
      "October", "November", "December"];

   /* The thisMonth variable contains the calendar month number,
      the thisYear variable contains the 4-digit year value */
   var thisMonth = calendarDay.getMonth();
   var thisYear = calendarDay.getFullYear();

   // Write the table header row of the calendar table
   document.write("<tr>");
   document.write("<th id='calendar_head' colspan='7'>");

   document.write(monthName[thisMonth] + " " + thisYear);

   document.write("</th>");
   document.write("</tr>");

}
```

Values within an array are referenced using the format

array[i]

where *array* is the array name and *i* is the index number of the value within the array.

Working with Arrays

- An **array** is a collection of data values organized under a single name
  - Each individual data value is identified by an **index**
- To create an array:
  - `var array = new Array(length);`
- To populate an array:
  - `array[i] = value;`
- To create and populate an array:
  - `var array = new Array(values);`

Specifying Array Length

- To determine the size of an array, use the property:
  - `array.length`
- To add more items to an array, run the command:
  - `array[i] = value;`
- To remove items from an array, run the command:
  - `array.length = value;`

where value is an integer that is smaller than the highest index currently in the array

Methods of JavaScript Arrays

Figure 12-11 Methods of JavaScript arrays

Array Method	Description
`array.concat(array1, array2,...)`	Joins *array* to two or more arrays, creating a single array containing the items from all the arrays.
`array.join(separator)`	Joins all items in *array* into a single text string. The array items are separated using the text in the *separator* parameter. If no separator is specified, a comma is used.
`array.pop()`	Removes the last item from *array*.
`array.push(values)`	Appends *array* with new items, where *values* is a comma-separated list of item values.
`array.reverse()`	Reverses the order of items in *array*.
`array.shift()`	Removes the first item from *array*.
`array.slice(start, stop)`	Extracts the *array* items starting with the *start* index up to the *stop* index, returning a new subarray.
`array.splice(start, size, values)`	Extracts *size* items from *array* starting with the item with the index *start*. To insert new items into the array, specify the array items in a comma-separated *values* list.
`array.sort(function)`	Sorts *array* where *function* is the name of a function that returns a positive, negative, or 0 value. If no function is specified, *array* is sorted in alphabetical order.
`array.toString()`	Converts the contents of *array* to a text string with the array values in a comma-separated list.
`array.unshift(values)`	Inserts new items at the start of *array*, where *values* is a comma-separated list of new values.

Using Program Loops

The days of the week are written using a **program loop** that repeats a set of similar commands until a stopping condition is met.

The *start* expression provides the starting value of the counter variable, *i*.

The *continue* expression is a Boolean expression that must equate to `true` for the loop to continue.

The *update* expression updates the counter variable each time through the loop.

```
function writeDayNames() {

   // Array of weekday abbreviations
   var dayName = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];

   // Start a table row of the weekday abbreviations
   document.write("<tr>");

   // Loop through the dayName array
   for (var i = 0; i < dayName.length; i++) {
      document.write("<th class='calendar_weekdays'> " + dayName[i] + "</th>");
   }

   // End the table row
   document.write("</tr>");

}
```

A `for` **loop** is a programming structure in which a set of commands is repeated based on the changing values of a **counter variable**.

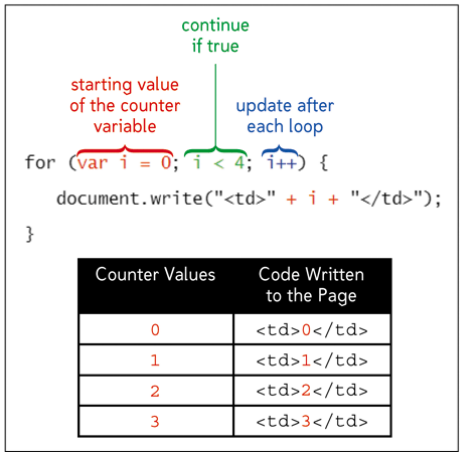
The same code is executed with different values of the counter variable during each iteration of the for loop.

Working with Program Loops

- A **program loop** is a set of commands executed repeatedly until a stopping condition has been met
  - For loop
    - A **counter variable** tracks the number of times a block of commands is run

Working with Program Loops

Figure 12-12 Viewing a for loop



Counter Values	Code Written to the Page
0	<td>0</td>
1	<td>1</td>
2	<td>2</td>
3	<td>3</td>

Figure 12-13 Nested for loops

Nested `for` loop

```
for (var rowNum = 1; rowNum < 4; rowNum++) {
   document.write("<tr>");
   for (var colNum = 1; colNum < 3; colNum++) {
      document.write("<td>" + rowNum + "," + colNum + "</td>");
   }
   document.write("</tr>");
}
```

rowNum Counter	colNum Counter	Code Written to the Page
1		<tr>
1	1	<td>1,1</td>
1	2	<td>1,2</td>
1		</tr>
2		<tr>
2	1	<td>2,1</td>
2	2	<td>2,2</td>
2		</tr>
3		<tr>
3	1	<td>3,1</td>
3	2	<td>3,2</td>
3		</td>

Working with Program Loops

- For loops are often used to cycle through the different values contained within an array

Figure 12-15 Creating the writeDayNames() function

```
function writeDayNames() {

   // Array of weekday abbreviations
   var dayName = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];

   // Start a table row of the weekday abbreviations
   document.write("<tr>");

   // Loop through the dayName array
   for (var i = 0; i < dayName.length; i++) {
      document.write("<th class='calendar_weekdays'> " + dayName[i] + "</th>");
   }

   // End the table row
   document.write("</tr>");

}
```

Working with Program Loops

- A while loop does not depend on the value of a counter variable; it runs as long as a specific condition is met

```
var rowNum = 1;

while (rowNum < 4) {
   document.write("<tr>");
   var colNum = 1;

   while (colNum < 3) {
      document.write("<td>" + rowNum + "," + colNum + "</td>");
      colNum++;
   }

   document.write("</tr>");
   rowNum++;
}
```

Creating Program Loops

- To create a For loop, use the syntax:

```
for (start; continue; update) {
   commands
}
```

- To create a While loop, use the following syntax:

```
while (continue) {
   commands
}
```

Creating Program Loops

- To create a Do/While loop, use the following syntax:

```
do {
   commands
   }
while (continue);
```

- To loop through the contents of an array, enter the For loop:

```
for (var i = 0; i < array.length; i++) {
   commands involving array[i]
}
```

Array Methods with ECMAScript 5

- Program loops are often used with arrays, and starting with ECMAScript 5—the version of JavaScript developed for HTML5—several new methods were introduced to allow programmers to loop through the contents of an array without having to create a program loop structure
- Each of these methods is based on calling a function that will be applied to the array and its contents
  - `array.method(function)`

Array Methods with ECMAScript 5

Figure 12-18 Summary methods for arrays

Method	Description
`every(function)`	Tests whether the condition returned by *function* holds for all items in the array (`true`) or at least one counter-example exists (`false`)
`filter(function)`	Creates a new array populated with the elements of the original array that return a value of `true` from *function*
`forEach(function)`	Applies *function* to each item within the array
`indexOf(value[, start])`	Searches the array, returning the index number of the first element equal to *value*, starting from the optional *start* index
`lastIndexOf(value[, start])`	Searches backward through the array, returning the index number of the first element equal to *value*, starting from the optional *start* index
`map(function)`	Creates a new array by passing the original array items to *function*, which returns the mapped value of the array items
`reduce(function)`	Reduces the array by passing the original array items to *function*, keeping only those items that return a value of `true`

Figure 12-18 Summary methods for arrays (continued)

Method	Description
`reduceRight(function)`	Reduces the array from the last element by passing the original array items to *function*, keeping only those elements that return a value of `true`
`some(function)`	Tests whether the condition expressed in *function* holds for any elements in the array (`true`) or for no elements in the array (`false`)

Conditional Statements

The conditional statement in a simple if statement is either true or false.

In a simple if statement, an expression is tested for being true or false; if true, a specified command is run.

The command in a simple if statement is run if the condition is true.

The first command block in an if else statement is run if the condition is true.

```
for (var i = 1; i <= totalDays; i++) {
   // Move to the next day in the month
   day.setDate(i);
   weekDay = day.getDay();

   if (weekDay == 0) document.write("<tr>");

   if (i == highlightDay) {
      document.write("<td class='calendar_dates'>" + i + "</td>");
   } else {
      document.write("<td class='calendar_today'>" + i + "</td>");
   }

   if (weekDay == 6) document.write("</tr>");
}
```

In an if else statement one command block is run if the statement is true, while a second command block is run if the statement is false.

The equals operator (==) is used to test whether one value equals another.

The second command block in an if else statement is run if the condition is false.

The or operator (||) is used when either of two conditions may be true for the entire conditional expression to be true.

In a nested if structure, one if statement is placed within another; the nested if statement is run only if the conditional expressions of both the outer and inner if statements are true.

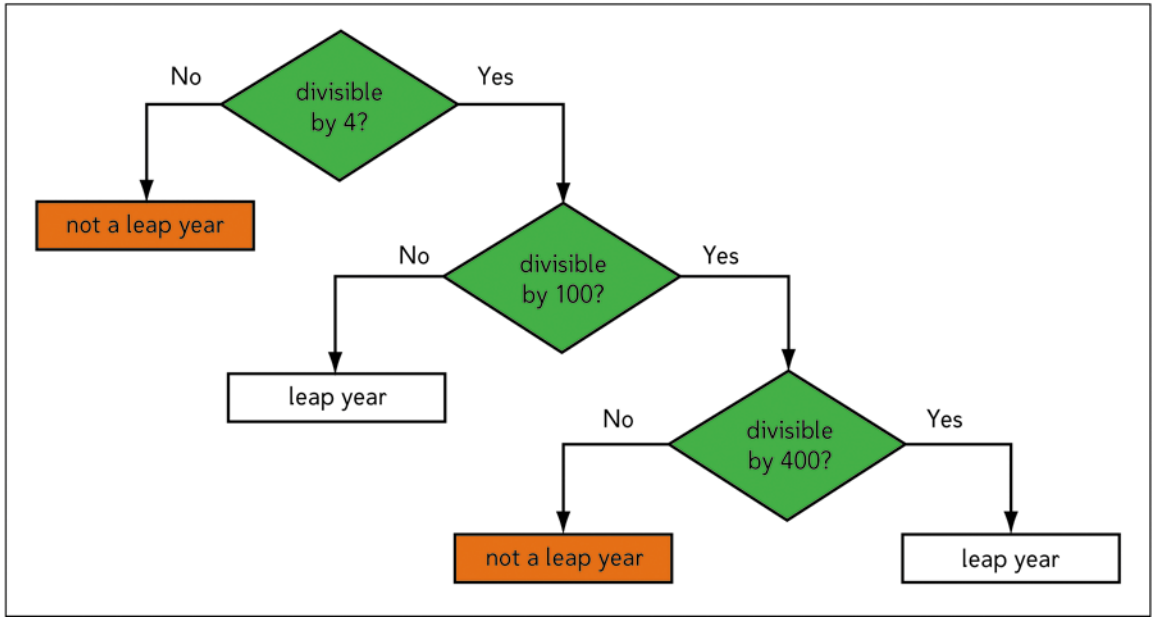
```
function daysInMonth(calendarDay) {

   // Array of days in each month
   var dayCount = [31,28,31,30,31,30,31,31,30,31,30,31];

   // Extract the four digit year value from calendarDay
   var thisYear = calendarDay.getFullYear();

   // Extract the month value from calendarDay
   var thisMonth = calendarDay.getMonth();

   // Revise the days in February for leap years
   if (thisYear % 4 == 0) {
      if ( (thisYear % 100 != 0) || (thisYear % 400 == 0) ) {
         dayCount[1] = 29 ;
      }
   }

   // Return the number of days for the current month
   return dayCount[thisMonth];

}
```

If either of the two conditions joined by an or operator is true, then the entire conditional expression is true.

Working with Conditional Statements

- A **conditional statement** is a statement that runs a command or command block only when certain circumstances are met

Figure 12-21 Process to calculate leap years



Working with Conditional Statements

- To test a single condition, use the construction:

```
if (condition) {
   commands
}
```

- To test between two conditions, use the following construction:

```
if (condition) {
   commands if condition is true
} else {
   commands if otherwise
}
```

Working with Conditional Statements

- To test multiple conditions, use the construction:

```
if (condition 1) {
   first command block
} else if (condition 2) {
   second command block
} else if (condition 3) {
   third command block
} else {
   default command block
}
```

Creating a Switch Statement

- To create a Switch statement to test for different values of an expression, use the structure:

```
switch (expression) {
   case label1: commands1
   break;
   case label2: commands2
   break;
   case label3: commands3
   break;
...
   default: default commands
}
```

Managing Program Loops and Conditional Statements

- The break statement terminates any program loop or conditional statement
- The syntax for the break command is:
  - `break;`
- The continue statement stops processing the commands in the current iteration of the loop and jumps to the next iteration
  - `continue;`

Managing Program Loops and Conditional Statements

- Statement labels are used to identify statements in JavaScript code so that you can reference those statements elsewhere in a program
  - `label: statement`
  - `break label;`
  - `continue label;`