COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 3
Building Arrays and Controlling Flow

Objectives

In this chapter, you will:

- Store data in arrays
- Use `while` statements, `do/while` statements, and `for` statements to repeatedly execute code
- Use `continue` statements to restart looping statements
- Use `if` statements, `if/else` statements, and `switch` statements to make decisions
- Nest one `if` statement in another

Storing Data in Arrays

- Array
  - Set of data represented by a single variable name

Figure 3-1 Conceptual illustration of an array

Declaring and Initializing Arrays

- Array literal
  - most common way to create an array
  - declares a variable and specifies array as content
- Syntax

  var *name* = [*value1*, *value2*, *value3*, …];
- Example:
  - Create an array named `newsSections` containing 4 strings as elements

```
var newsSections = ["world","local","opinion","sports"]
```


Declaring and Initializing Arrays (cont'd.)

- Element
  - Each piece of data contained in an array
- Index
  - Element's numeric position within the array
  - Array element numbering
    - Starts with index number of zero (0)
- Reference element using its index number
  - Example: to reference the 2nd element in the newsSections array

```
newsSections[1]
```


Declaring and Initializing Arrays (cont'd.)

- Assigning values to individual array elements
  - Include the array index for an individual element
- Example:
  - Add value "entertainment" as fifth element of `newsSections` array

```
newsSections[4] = "entertainment";
```

Declaring and Initializing Arrays (cont'd.)

- Can create an array without any elements
  - Add new elements as necessary
  - Array size can change dynamically

```
var colors = [];
colors[2] = "yellow";
```

- JavaScript values assigned to array elements
  - Can be different data types

Accessing Element Information

- To access an element's value:
  - Include brackets and element index
- Examples:

```
var sec1Head = document.getElementById("section1");
var sec2Head = document.getElementById("section2");
var sec3Head = document.getElementById("section3");
sec1Head.innerHTML = newsSections[0]; // "world"
sec2Head.innerHTML = newsSections[1]; // "local"
sec3Head.innerHTML = newsSections[2]; // "opinion"
```


Modifying Elements

- To modify values in existing array elements
  - Include brackets and element index
- Can change a value assigned to an array element
- Example:

```
newsSections[4] = "living";
```


Determining the Number of Elements in an Array

- `length` property
  - Returns the number of elements in an array
- Syntax

```
name.length;
```


Using the `Array` Object

- JavaScript represents arrays with the `Array` object
  - Contains a special constructor named `Array()`
- Constructor
  - Special function type used as the basis for creating reference variables
- Syntax

```
var newsSections = new Array(6);
```

- Array literals preferred
  - Easier

Referencing Default Collections of Elements

- `getElementsByTagName()` method
  - Can reference web page element by looking up all elements of a certain type in document and referencing one element in that collection
  - Resulting collection uses syntax similar to arrays
- Example:

```
document.getElementsByTagName("li")[2]
```

Repeating Code

- Loop statement
  - Control flow statement repeatedly executing a statement or a series of statements
    - While a specific condition is true or until a specific condition becomes true
- Three types of loop statements
  - `while` statements
  - `do/while` statements
  - `for` statements

`while` Statements

- `while` statement
  - Repeats a statement or series of statements
    - As long as a given conditional expression evaluates to a truthy value
- Syntax

```
while (expression) {
        statements
}
```

- Iteration
  - Each repetition of a looping statement

`while` Statements (cont'd.)

- Counter
  - Variable incremented or decremented with each loop statement iteration
- Examples:
  - `while` statement using an increment operator
  - `while` statement using a decrement operator
  - `while` statement using the `*=` assignment operator

`while` Statements (cont'd.)

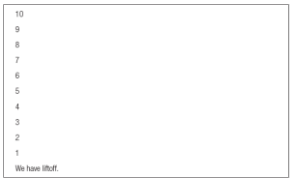
```
var count = 1;
while (count <= 5) {
  document.write(count + "<br />");
  count++;
}
document.write("<p>You have printed 5 numbers.</p>");
```

Result in browser:

`while` Statements (cont'd.)

```
var count = 10;
while (count > 0) {
  document.write(count + "<br />");
  count--;
}
document.write("<p>We have liftoff.</p>");
```

Result in browser:



`while` Statements (cont'd.)

```
var count = 1;
while (count <= 100) {
  document.write(count + "<br />");
  count *= 2;
}
```

Result in browser:

while Statements (cont'd.)

- Infinite loop
  - Loop statement that never ends
    - Conditional expression: never false
  - Example:

```
var count = 1;
while (count <= 10) {
  window.alert("The number is " + count + ".");
}
```


while Statements (cont'd.)

- Example:
  - assigning array element values to table cells:

```
function addColumnHeaders() {
  var i = 0;
  while (i < 7) {
    document.getElementsByTagName("th")
      [i].innerHTML = daysOfWeek[i];
    i++;
  }
}
```


do/while Statements

- do/while statement
  - Executes a statement or statements once
  - Then repeats the execution as long as a given conditional expression evaluates to a truthy value
- Syntax

```
do {
  statements;

} while (expression);
```


do/while Statements (cont'd.)

- Examples:

```
var count = 2;
do {
  document.write("<p>The count is equal to " +
    count + ".</p>");
  count++;
} while (count < 2);
```

```
var count = 2;
while (count < 2) {
  document.write("<p>The count is equal to " +
    count + ".</p>");
  count++;
}
```


do/while Statements (cont'd.)

- Example:
  - adding days of week with a do/while statement instead of a while statement

```
var i = 0;
do {
  document.getElementsByTagName("th")[i].innerHTML =
    daysOfWeek[i];
  i++;
} while (i < 7);
```


for Statements

- for statement
  - Repeats a statement or series of statements
    - As long as a given conditional expression evaluates to a truthy value
  - Can also include code that initializes a counter and changes its value with each iteration
- Syntax

```
for (counter_declaration; condition;
    counter_operation) {
  statements
}
```

for Statements (cont'd.)

- Steps when JavaScript interpreter encounters a for loop
  1. Counter variable declared and initialized
  2. for loop condition evaluated
  3. If condition evaluation in Step 2 returns truthy value:
     - for loop statements execute, Step 4 occurs, and the process starts over again with Step 2

     If condition evaluation in Step 2 returns falsy value:
     - for statement ends
     - Next statement following the for statement executes
  4. Update statement in the for statement executed

for Statements (cont' d.)

```
var brightestStars =
   ["Sirius", "Canopus", "Arcturus", "Rigel", "Vega"];
for (var count = 0; count < brightestStars.length; count++) {
   document.write(brightestStars[count] + "<br />");
}
```

Result in browser:

Sirius
Canopus
Arcturus
Rigel
Vega

for Statements (cont'd.)

- for statement
  - More efficient than a while statement
- Examples:

```
var count = 1;
while (count < brightestStars.length) {
   document.write(count + "<br />");
   count++;
}
```

```
for (var count = 1; count < brightestStars.length; count++) {
   document.write(count + "<br />");
}
```


for Statements (cont'd.)

- Example:
  - addColumnHeaders() function with a for statement instead of a do/while statement

```
function addColumnHeaders() {
   for (var i = 0; i < 7; i++) {
      document.getElementsByTagName("th")[i].innerHTML
         = daysOfWeek[i];
   }
}
```


Using continue Statements to Restart Execution

- continue statement
  - Halts a looping statement
    - Restarts the loop with a new iteration
  - Used to stop a loop for the current iteration
    - Have the loop to continue with a new iteration
- Examples:
  - for loop with a continue statement

Using continue Statements to Restart Execution (cont' d.)

```
for (var count = 1; count <= 5; count++) {
   if (count === 3) {
      continue;
   }
   document.write("<p>" + count + "</p>");
}
```

Result in browser:

1
2
4
5

Making Decisions

- Decision making
  - Process of determining the order in which statements execute in a program
- Decision-making statements, decision-making structures, or conditional statements
  - Special types of JavaScript statements used for making decisions
- `if` statement
  - Most common type of decision-making statement

`if` Statements

- Used to execute specific programming code
  - If conditional expression evaluation returns truthy value
- Syntax

```
if (condition) {
  statements
}
```

- After the `if` statement executes:
  - Any subsequent code executes normally

`if` Statements (cont'd.)

- Use a command block to construct a decision-making structure containing multiple statements
- Command block
  - Set of statements contained within a set of braces

`if/else` Statements

- Executes one action if the condition is true
  - And a different action if the condition is false
- Syntax for an `if . . . else` statement

```
if (expression) {
  statements
}
else {
  statements
}
```


`if/else` Statements (cont'd.)

- Example:

```
var today = "Tuesday"
if (today === "Monday") {
  document.write("<p>Today is Monday</p>");
}
else {
  document.write("<p>Today is not Monday</p>");
}
```


Nested `if` and `if/else` Statements

- Nested decision-making structures
  - One decision-making statement contains another decision-making statement
- Nested `if` statement
  - An `if` statement contained within an `if` statement or within an `if/else` statement
- Nested `if/else` statement
  - An `if/else` statement contained within an `if` statement or within an `if/else` statement

Nested `if` and `if/else` Statements (cont'd.)

- Example:

```
var salesTotal = 75;
if (salesTotal > 50) {
  if (salesTotal < 100) {
    document.write("<p>The sales total is
      between 50 and 100.</p>");
  }
}
```


`else if` constructions

- Compact version of nested `if/else` statements
  - combine an `else` statement with its nested `if` statement
  - requires fewer characters
  - easier to read

`else if` constructions (cont'd.)

nested `if/else` version

```
if (gameLocation[i] === "away") {
  paragraphs[1].innerHTML = "@ ";
}
else {
  if (gameLocation[i] === "home") {
    paragraphs[1].innerHTML = "vs ";
  }
}
```

`else if` version

```
if (gameLocation[i] === "away") {
  paragraphs[1].innerHTML = "@ ";
}
else if (gameLocation[i] === "home") {
  paragraphs[1].innerHTML = "vs ";
}
```


`else if` constructions (cont'd)

- Used to create backward-compatible event listeners:

```
var submitButton = document.getElementById("button");
if (submitButton.addEventListener) {
  submitButton.addEventListener("click", submitForm,
    false);
}
else if (submitButton.attachEvent) {
  submitButton.attachEvent("onclick", submitForm);
}
```


`switch` Statements

- Controls program flow by executing a specific set of statements
  - Dependent on an expression value
- Compares expression value to value contained within a `case` label
- `case` label
  - Represents a specific value
  - Contains one or more statements that execute:
    - If `case` label value matches the switch statement's expression value

`switch` Statements (cont'd.)

- Syntax

```
switch (expression) {
  case label:
    statements;
    break;
  case label:
    statements;
    break;
  ...
  default:
    statements;
    break;
}
```

`switch` Statements (cont'd.)

- `default` label
  - Executes when the value returned by the `switch` statement expression does not match a case label
- When a `switch` statement executes:
  - Value returned by the expression is compared to each `case` label
    - In the order in which it is encountered
- `break` statement
  - Ends execution of a `switch` statement
  - Should be final statement after each case label

`switch` Statements (cont' d.)

```
function city_location(americanCity) {
  switch (americanCity) {
    case "Boston":
      return "Massachusetts";
      break;
    case "Chicago":
      return "Illinois";
      break;
    case "Los Angeles":
      return "California";
      break;
    case "Miami":
      return "Florida";
      break;
    case "New York":
      return "New York";
      break;
    default:
      return "United States";
      break;
  }
}
document.write("<p>" + city_location("Boston") + "</p>");
```


Summary

- Array
  - Set of data represented by a single variable name
  - Index: element's numeric position within the array
  - Can access and modify array elements
  - `length` property
    - number of elements in an array

Summary (cont'd.)

- Loop statements
  - `while` statements, `do/while` statements, and `for` statements
  - Iteration: each repetition of a looping statement
  - Counter: variable
    - Incremented or decremented with each iteration of a loop statement
  - `continue` statement
    - Restarts a loop with a new iteration

Summary (cont'd.)

- Decision making
  - Determining the order in which statements execute in a program
- May execute in a linear fashion
  - `if` statement,`if/else` statement, `else if` construction
    - Nested decision-making structures
  - `switch` statement and `case` labels
  - `break` statement: used to exit control statements
  - Command block
    - Set of statements contained within a set of braces
    - May repeat the same statement, function, or code section