Chapter 8. JavaScript: Control Structures I

JavaScript supports the usual control structures.

JavaScript supports abbreviated assignment operators of the form:

\[ op= \]

Example:

\[ x += y; \]

JavaScript supports the increment and decrement (both pre- and post-) operators of C++.

Example:

\[ x ++ ; \]

When a variable is declared but not given a value, it has an \textit{undefined} value.

To indicate that a variable doesn’t contain a value, you can assign the value \texttt{null} to it.
Chapter 9. JavaScript: Control Structures II
You can use control structures to get display effects.

Example:

```javascript
<script type = "text/javascript">
    for ( var counter = 1; counter <= 5; ++ counter )
        document.writeln( "<p><font size = "" + counter + 
                          
                          
                          "">HTML font size " + counter + 
                        
                        "</font></p>" );
</script>
```

For each value `counter` 1-5, this code sends `document` the string

```html
<p><font size = "counter">HTML font size `counter`</font></p>
```
JavaScript has the C++ logical operators (!, &&, ||) and constants true and false.

Example:

```html
<script type = "text/javascript">
var choice,
    startTag,
    endTag,
    validInput = true,
    listType;

choice  =  window.prompt( "Select a list style:\n" +
    "1 (bullet), 2 (numbered), 3 (lettered)", "1" );

switch ( choice ) {
  case "1":
    startTag =  "<ul>";
    endTag   =  "</ul>";
    listType =  "<h2>Bullet List</h2>";
    break;
  case "2":
    startTag =  "<ol>";
    endTag   =  "</ol>";
    listType =  "<h2>Ordered List: Numbered</h2>";
    break;
  case "3":
    startTag =  "<ol type = 'A'>";
    endTag   =  "</ol>";
    listType =  "<h2>Ordered List: Lettered</h2>";
    break;
  default:
    validInput = false;
}

if ( validInput ) {
  document.writeln( listType + startTag );
  for ( var i = 1; i <= 3; ++ i )
    document.writeln("<li>List item " + i + "</li>" );
  document.writeln( endTag );
}
else
  document.writeln( "Invalid choice: " + choice );
</script>
```
When 1 is entered in the prompt dialog box, this is rendered as:

**Bullet List**

- List item 1
- List item 2
- List item 3

Click Refresh to run this script again.

In this case, the text rendered by the browser was:

```html
<h2>Bullet List</h2>
<ul>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ul>
```
Chapter 10. JavaScript: Functions

The syntactic form of a function definition is:

function function-name (parameter-list)
{
   declarations and statements
}

where

parameter-list is a comma-separated list of formal parameters, and
function-name and the formal parameters are valid identifiers.

If the function is to return a value, it must execute a statement

return expression;

where the value of expression is what is returned.

If the function does not return a value, control is returned either by executing a statement

return;

or by control reaching the closing ‘}’.

A function definition can go anywhere in the script (even surrounded by the main code).

But it is conventional to put function definitions at the end of the script.

The body of a function may access global variables.

The usual scoping rules apply.
Example

```
<script type = "text/javascript">
    var
        input1 = parseInt( window.prompt(
            "Enter an integer", "0" )),
        input3;

    document.writeln( "<p>" + f1( input1 ) +
        " is the sum of " + input1 +
        " and some other number." </p> );

    document.writeln( "<p>The value of input3 is 
        + input3 + "</p>" );

    function f1( i1 )
    {
        var
            input2 = parseInt( window.prompt(
                "Enter another integer", "0"));

            input3 = parseInt( window.prompt(
                "Enter a third integer", "0"));

            return  i1 + input2;
    }
</script>
```

The following is the display when the integers 5, 10, and 20 are entered in that order:

15 is the sum of 5 and some other number.

The value of input3 is 20

Click Refresh to run this script again.
The following has exactly the same behavior.

Here we have moved the last statement in the top-level code so that it occurs after the definition of function f1.

(In fact, the function definition could appear anywhere in the script.)

```javascript
<script type= "text/javascript">
var
  input1 = parseInt( window.prompt(  
    "Enter an integer", "0" )),

input3;

document.writeln("<p>" + f1( input1 ) +  
  " is the sum of " + input1 + 
  " and some other number.</p>" );

function f1( i1 )
{
  var
    input2 = parseInt( window.prompt(  
      "Enter another integer", "0"));

    input3 = parseInt( window.prompt(  
      "Enter a third integer", "0"));

    return i1 + input2;
}

document.writeln("<p>The value of input3 is "  
  + input3 + "</p>" );

</script>
```
We can write script that consists only of functions by associating one function with an *event* as the *event handler* for that event.

When that event happens, its event handler is called.

The *onload* event happens when the page is loaded.

So, if \( f() \) is a function we want to use as the (dynamically) top-level function, we define \( f() \) in the header and have a body of the following form:

\[
\text{<body onload = "f()"} > \\
\text{</body>}
\]

Note that clicking **Refresh** does not invoke the function again – it isn’t an *onload* event.
Example:

```html
<head>
<title>Using ONLOAD</title>

<script type = "text/javascript">

  function first() {
    document.writeln("<p>In function first().</p>");
    second();
  }

  function second() {
    document.writeln("<p>In function second().</p>");
  }

</script>

</head>

<body onload = "first()">
</body>
```

In function first().
In function second().
Chapter 11. JavaScript: Arrays

An array is an object.

`new` is the dynamic memory allocation operator for instantiating an object.

Use

```
new Array(size)
```

where `size` is a positive integer.

The array will have indices from 0 to `size-1`.

Example:

```
var ar = new Array( 10 );
```

or

```
var ar;
ar = new Array( 10 );
```

Where `ar` is an array, `ar.length` is the length of `ar`, that is, one more than its largest index value.
Two ways to initialize an array:

- Supply a comma-separated list of \( n \) arguments to the constructor `Array`.
  This creates an array of those \( n \) elements, in the order supplied.
  Example:
  ```javascript
  var vals = new Array(1, 2, 3, 4, 5);
  ```

- Assign to a variable an initializer list constructed from a comma-separated list of values enclosed in “[ … ]”.
  Example:
  ```javascript
  var vals = [1, 2, 3, 4, 5];
  ```
  This allows place holders, for example,
  ```javascript
  var vals = [1, 2, , 4, 5];
  ```
  An unset value is output as `undefined`.

All objects (including arrays) are passed by reference.

But array elements are passed by value.
JavaScript arrays are dynamic: the size of an array can be changed at run time.

If \( \text{ar} \) is an array and \( n \) is a natural number, then
\[
\text{ar}.\text{length} = n;
\]
makes the length of \( \text{ar} \) \( n \).

This could increase (if \( n > \text{ar}.\text{length} \)) or decrease (is \( n < \text{ar}.\text{length} \)) the length of \( \text{ar} \).

If it increases the length, then the new elements are undefined.

We can also increase the length of an array by assigning to an element whose index is greater than or equal to its length.

If \( n > \text{ar}.\text{length} \), then
\[
\text{ar}[n] = x;
\]
does the following. It

- increases \( \text{ar}.\text{length} \) to \( n+1 \),
- assigns the value of \( x \) to the (new) last element, and
- creates new, undefined elements between the old and new last elements.

The statement
\[
\text{ar}[\text{ar}.\text{length}] = x;
\]
creates a new last element of \( \text{ar} \), assigning it the value of \( x \).

It increments \( \text{ar}.\text{length} \) by one.
Example

```javascript
<script type = "text/Javascript">
  var vals = new Array( 0, 2, 4 );
  vals[ 4 ] = 8;

  for ( var i = 0; i < vals.length; i++ )
    document.write( vals[i] + " " );
  document.writeln( "<br>" );

  vals[ 3 ] = 6;
  vals[ vals.length ] = 10;

  for ( var i = 0; i < vals.length; i++ )
    document.write( vals[i] + " " );
  document.writeln( "<br>" );

  vals.length = 4;

  for ( var i = 0; i < vals.length; i++ )
    document.write( vals[i] + " " );
  document.writeln( "<br>" );
</script>
```

0 2 4 undefined 8
0 2 4 6 8 10
0 2 4 6
The \textbf{for/in} Control Structure

The statement

\begin{verbatim}
  for ( variable in array )
    body
\end{verbatim}

iterates \textit{variable} over the indices of \textit{array}.

For example,

\begin{verbatim}
  sum = 0;
  for ( var i in values)
    sum += values[ i ];
\end{verbatim}

computes \textit{sum} as the sum of all the elements in \textit{values}.
Multiple-Subscripted Arrays

\[ \text{ar} [ \text{row} ] [ \text{col} ] \]

A 2D array is an array of (row) arrays.

Allocate an array for the rows.

Set each row to a new array with enough cells for each column.

For example, for a \(2 \times 3\) array \(b\), do

```javascript
var b = new Array(2);
b[0] = new Array(3);
  // b[0][0], b[0][1], b[0][2]
b[1] = new Array(3);
```

An array can also be dimensioned by an initialization, e.g.:

```javascript
var ar = [ [1, 2, 3], [4, 5, 6] ];
```
Example:

```html
<html>
<head>
<title>Double-subscripted Array Example</title>
<script type = "text/javascript">

    function start()
    {
        var grades = [ [ 77, 68, 86, 73 ],
                        [ 96, 87, 89, 81 ],
                        [ 70, 90, 86, 81 ] ];

        outputArray( grades );
        document.writeln( "<pre>" );
        for ( var i in grades )
            document.write( "\nAverage for student " + i + " is " + average( grades[i] ) );
        document.writeln( "</pre>" );
    }

    function average( rowOfGrades )
    {
        var total = 0;
        for ( var i in rowOfGrades )
            total += rowOfGrades[i];
        return total / rowOfGrades.length;
    }

</script>
</head>
<body>
</body>
</html>
```

Continued
function outputArray( grades )
{
    document.writeln("<table border = '1'" +
        " align = 'center' width = '60%'">");
    document.writeln("<caption>Student Grades" +
        " on Exams</caption>" );

document.writeln("<thead>");
document.write("<tr>")
document.write("<th></th>");
    for ( var i in grades[ 0 ] )
        document.write("<th>Exam " + i + "</th>" );

document.writeln("</tr>");
document.writeln("</thead>");

document.writeln("<tbody align = 'center'>" );
    for ( var i in grades ) {
        document.write("<tr><td><strong>Student " +
            i + "</strong></td>");
            for ( var j in grades[ i ] )
                document.write("<td>" + grades[ i ][ j ] +
                    "</td>" );
            document.writeln("</td>");
    }

document.writeln("</tbody>");
document.writeln("</table>" );
}
</script>
</head>
<body onload = "start()">
</body>
</html>
## Student Grades on Exams

<table>
<thead>
<tr>
<th></th>
<th>Exam 0</th>
<th>Exam 1</th>
<th>Exam 2</th>
<th>Exam 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 0</td>
<td>77</td>
<td>68</td>
<td>86</td>
<td>73</td>
</tr>
<tr>
<td>Student 1</td>
<td>96</td>
<td>87</td>
<td>89</td>
<td>81</td>
</tr>
<tr>
<td>Student 2</td>
<td>70</td>
<td>90</td>
<td>86</td>
<td>81</td>
</tr>
</tbody>
</table>

Average for student 0 is 76  
Average for student 1 is 88.25  
Average for student 2 is 81.75