Session: 14

Loops and Arrays

Objectives

- Explain while loop
- Explain for loop
- Explain do..while loop
- Explain break and continue statement
- Explain single-dimensional arrays
- Explain multi-dimensional arrays
- Explain for..in loop

Introduction

Loops allow you to execute a single statement or a block of statements multiple times.

They are widely used when you want to display a series of numbers and accept repetitive input.

A loop construct consists of a condition that instructs the compiler the number of times a specific block of code will be executed.

If the condition is not specified within the construct, the loop continues infinitely. Such loop constructs are referred to as infinite loops.

- JavaScript supports three types of loops that are as follows:
  - `while` Loop
  - `for` Loop
  - `do-while` Loop

while Loop 1-4

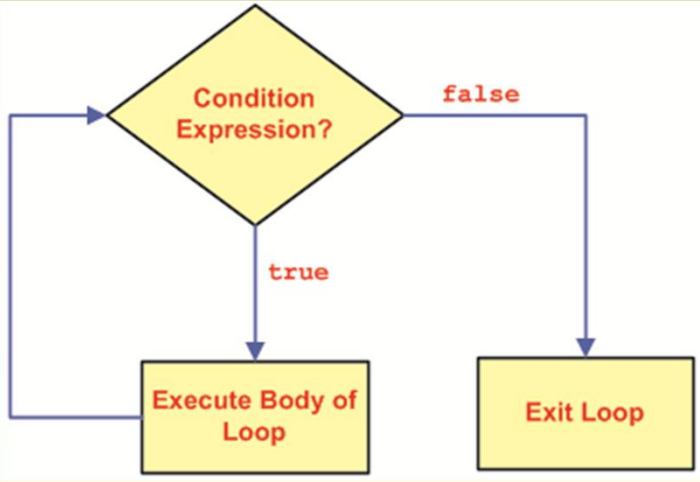
The `while` loop executes a block of code as long as the given condition remains `true`.

The while loop begins with the `while` keyword, which is followed by parentheses containing a `boolean` condition.

If this condition returns `true`, the block of statements within the while loop are executed.

Once the condition becomes `false`, the while statement stops the execution of loop and transfers the control to next statement appearing after the block.

- Following figure shows the flow of execution of the `while` loop.



while Loop 2-4

- The syntax for the `while` loop is as follows:

Syntax:

```
while (condition)
{
// statements;
}
```

where,

- `condition`: Is a boolean expression.

- The Code Snippet displays the sum of numbers from 1 to 10 by using the `while` loop.

```
<script>
var i = 0;
var sum = 0;
```

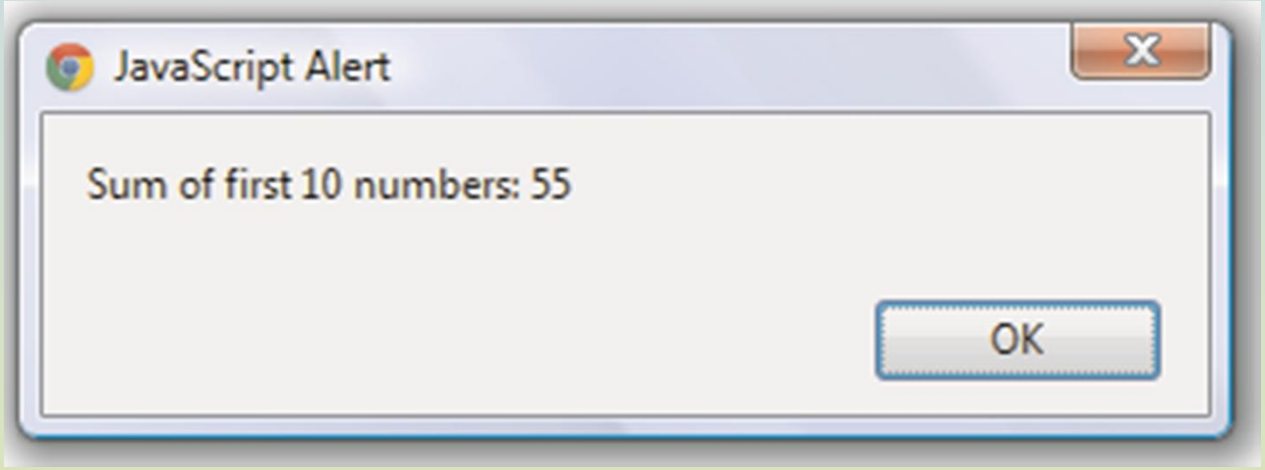
while Loop 3-4

```
while(i<=10)
{
sum = sum + i;
i = i + 1;
}
alert('Sum of first 10 numbers: ' + sum);
</script>
```

- The code declares two variables, `i` and `sum`, which are initialized to value `0`.
- The variable, `i`, is a counter variable, whose value increases for every execution of loop.
- The condition in the while loop checks that the value of the counter variable, `i`, is less than or equal to `10`.
- If this condition is `true`, the value of the sum variable is added to the value of `i` variable.
- The value of the variable `i` is incremented by `1`.
- Then, the program control is passed to the `while` statement to check the condition again.
- When the value of `i` becomes `11`, the while loop terminates as the loop condition becomes `false`.

while Loop 4-4

- Following figure shows the output.



for Loop 1-4

The `for` loop is similar to the `while` loop as it executes the statements within the loop as long as the given condition is `true`.

Unlike the `while` loop, the `for` loop specifies the loop control statements at the top instead in the body of the loop.

The `for` loop begins with the `for` keyword, which is followed by parentheses containing three expressions, each of which are separated by a semicolon.

The three expressions are referred to as **initialization expression**, **condition expression**, and **increment/decrement expression** respectively.

- The syntax for the `for` loop is as follows:

Syntax:

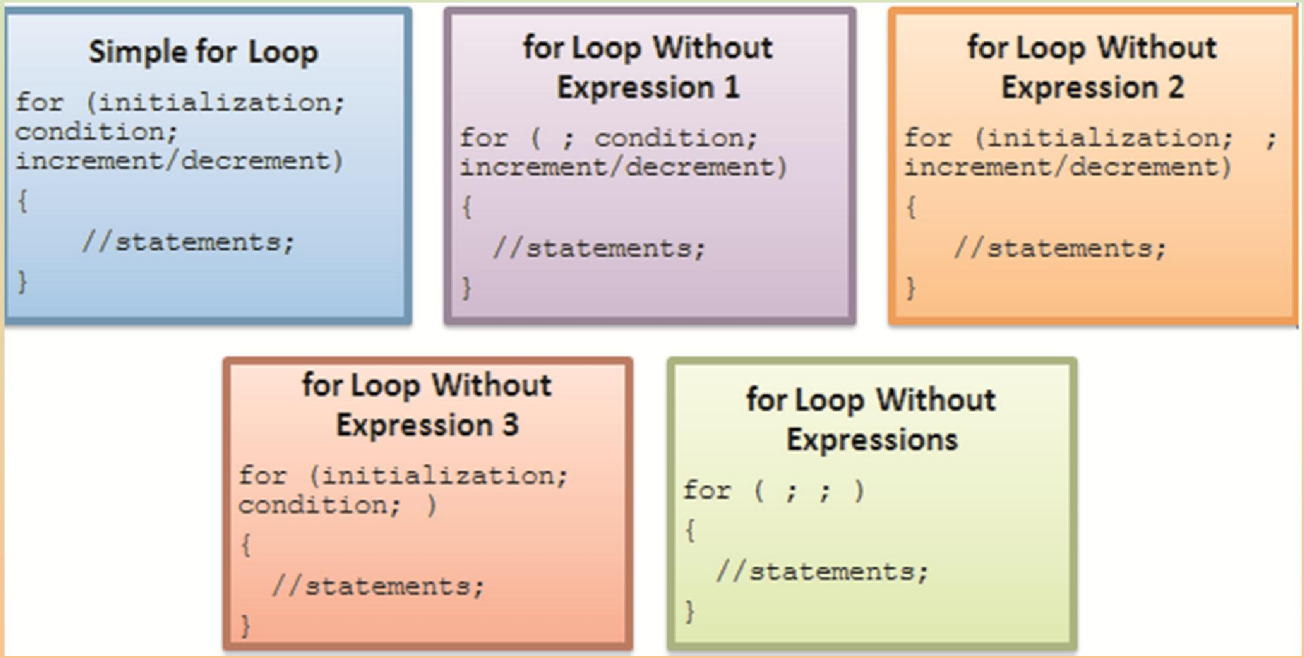
```
for (initialization; condition; increment/decrement)
{
// statements;
}
```

for Loop 2-4

where,

- `initialization`: Initializes the variable(s) that will be used in the condition.
- `condition`: Comprises the condition that is checked before the statements in the loop are executed.
- `increment/decrement`: Comprises the statement that changes the value of the variable(s) on every successful execution of the loop to ensure that the condition specified in the condition section is reached.

- Following figure shows the `for` loop.



for Loop 3-4

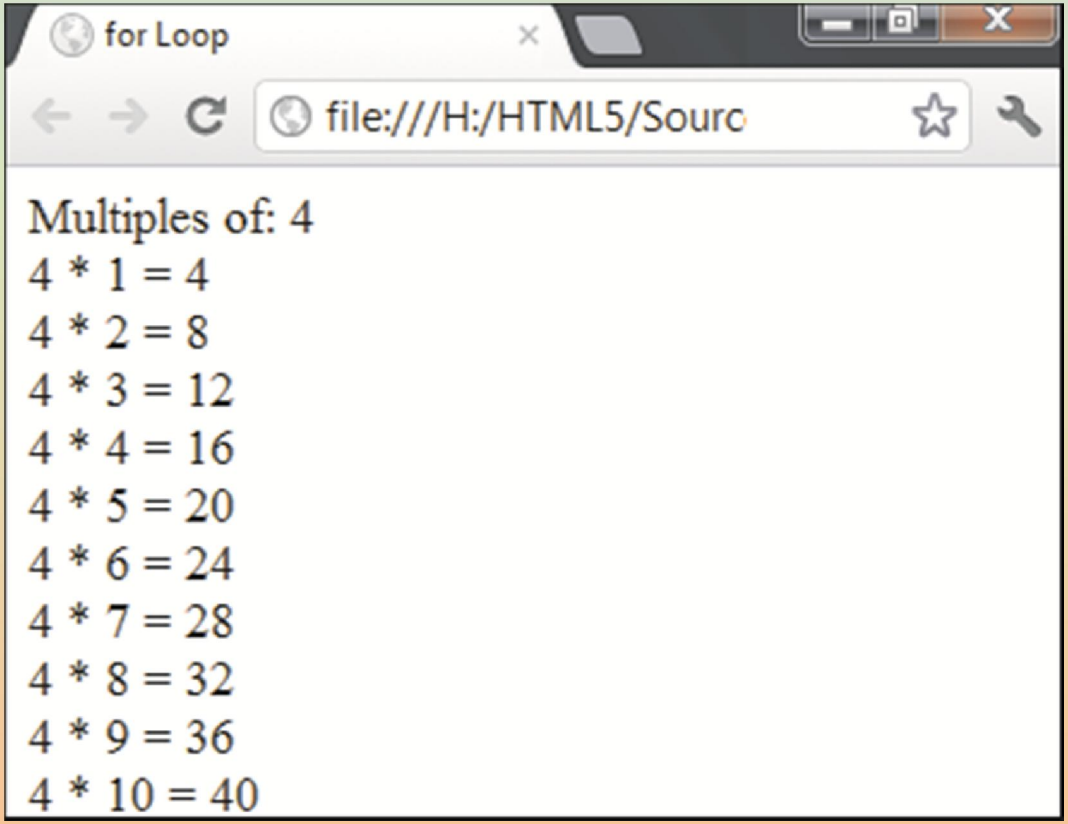
- The Code Snippet demonstrates the script that accepts a number from the user and displays the first ten multiples of that number.

```
<script>
var inputNum = prompt('Enter any number:');
var result = 0;
document.write ('Multiples of: ' + inputNum + '<br />');
for (var i=1; i<=10; i++)
{
   result = inputNum * i ;
   document.write (inputNum + '  *  ' + i + '  =  ' +
        result + '<br />');
}
</script>
```

- In the code, a variable, `inputNum`, is created and initialized to the value specified by the user in the prompt box.
- The `for` loop declares a variable, `i`, and initializes it to the value `1`.
- If the condition is `true`, the number specified by the user is multiplied to the value of `i` variable and the result is appended to the `result` variable.

for Loop 4-4

- The program control is again passed to `for` statement, where the value of `i` is incremented.
- The incremented value is again checked with the specified condition and it is multiplied to the number specified by the user.
- This process continues till the value of `i` becomes `11`.
- Following figure shows the multiples of a number.



do-while Loop 1-4

The do-while loop is similar to the while loop. This is because both the do-while and while loops execute until the condition becomes false.

However, the do-while loop differs by executing the body of the loop at least once before evaluating the condition even if the condition is false.

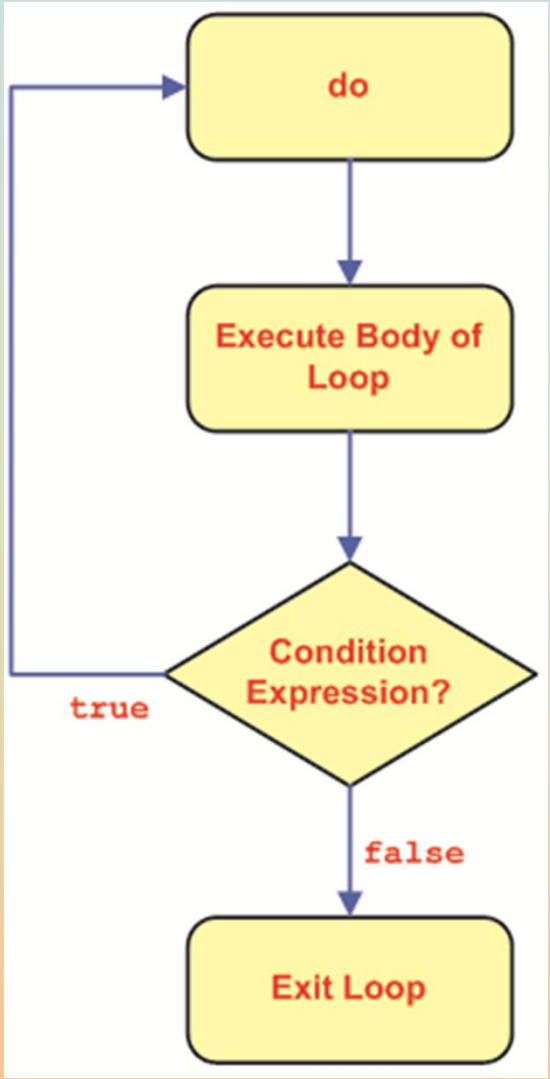
The do-while loop starts with the do keyword and is followed by a block of statements.

At the end of the block, the while keyword is specified that is followed by parentheses containing the condition.

When the condition returns false, the block of statements after the do keyword are ignored and the next statement following the while statement is executed.

do-while Loop 2-4

- Following figure shows the `do-while` loop.



do-while Loop 3-4

- The syntax for the `do-while` loop is as follows:

Syntax:

```
do
{
...
statements;
...
}while(condition);
```

where,

- `condition`: Is a boolean expression.

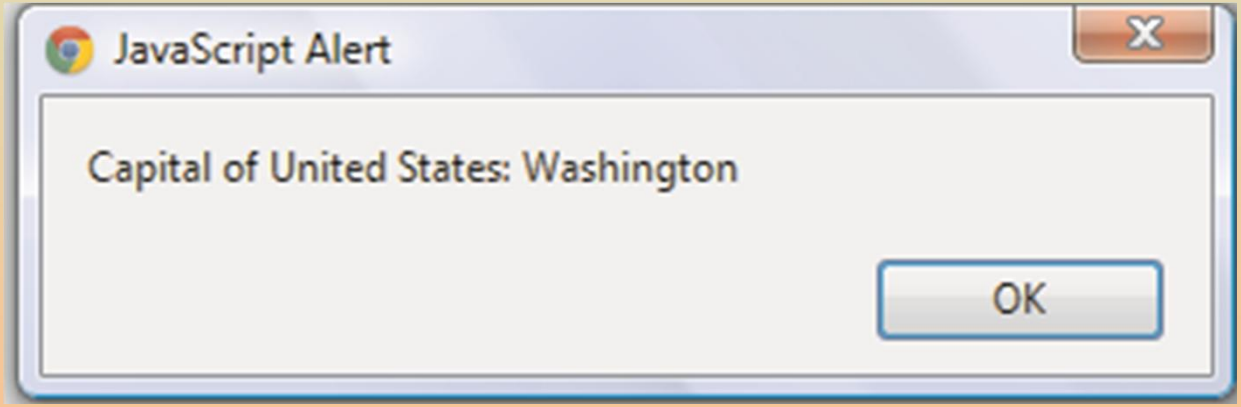
- The Code Snippet demonstrates the script to accept the capital of United States from the user using the `do-while` loop.

```
<script>
  var answer = '';
  do
  {
      answer = prompt('Capital of United States:', '');
  }while(answer!='Washington');
```

do-while Loop 4-4

```
alert('Capital of United States: ' + answer);
</script>
```

- The code declares a variable, `answer`, which stores the string entered by the user.
- The `do` block displays a prompt box without checking any condition.
- The prompt box accepts the capital of United States and stores this string in the variable, `answer`.
- The condition is specified in the `while` block that checks if the user has entered the string **Washington**.
- If this condition is `true`, prompt box is closed; else the prompt box is again displayed to accept the user input.
- Following figure shows the output of capital of United States.



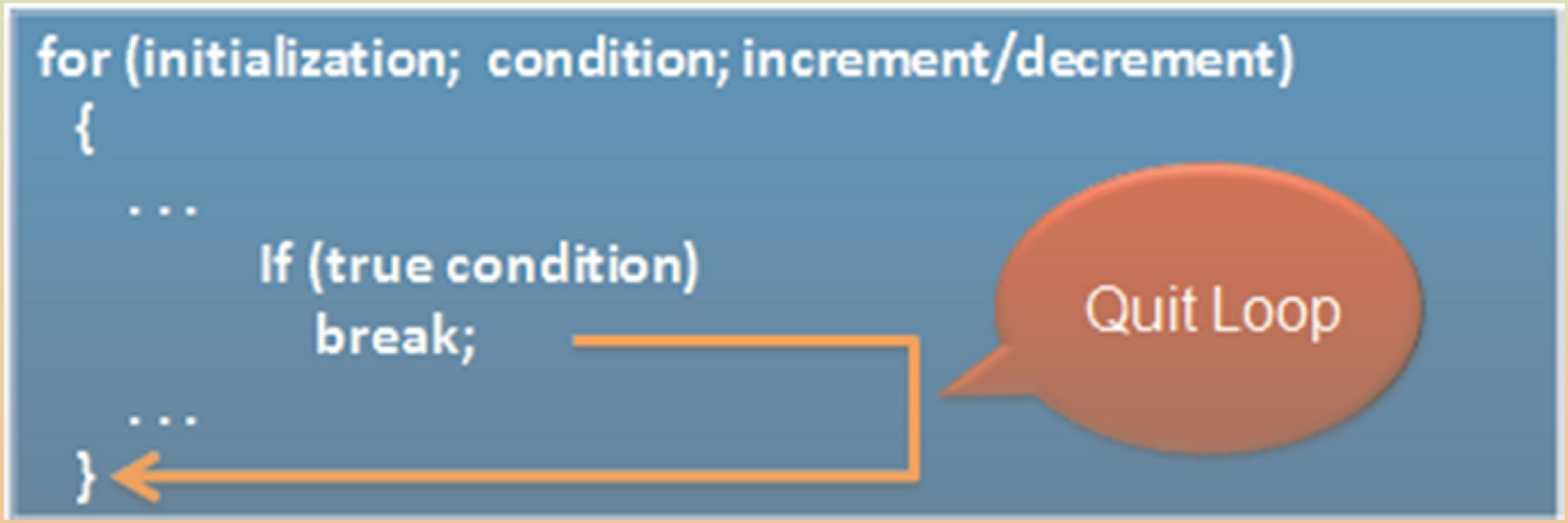
break Statement 1-3

The `break` statement can be used with decision-making such as `switch-case` and loop constructs such as `for` and `while` loops.

The `break` statement is denoted by using the `break` keyword. It is used to exit the loop without evaluating the specified condition.

The control is then passed to the next statement immediately after the loop.

- Following figure shows the flow of execution of the `break` statement.



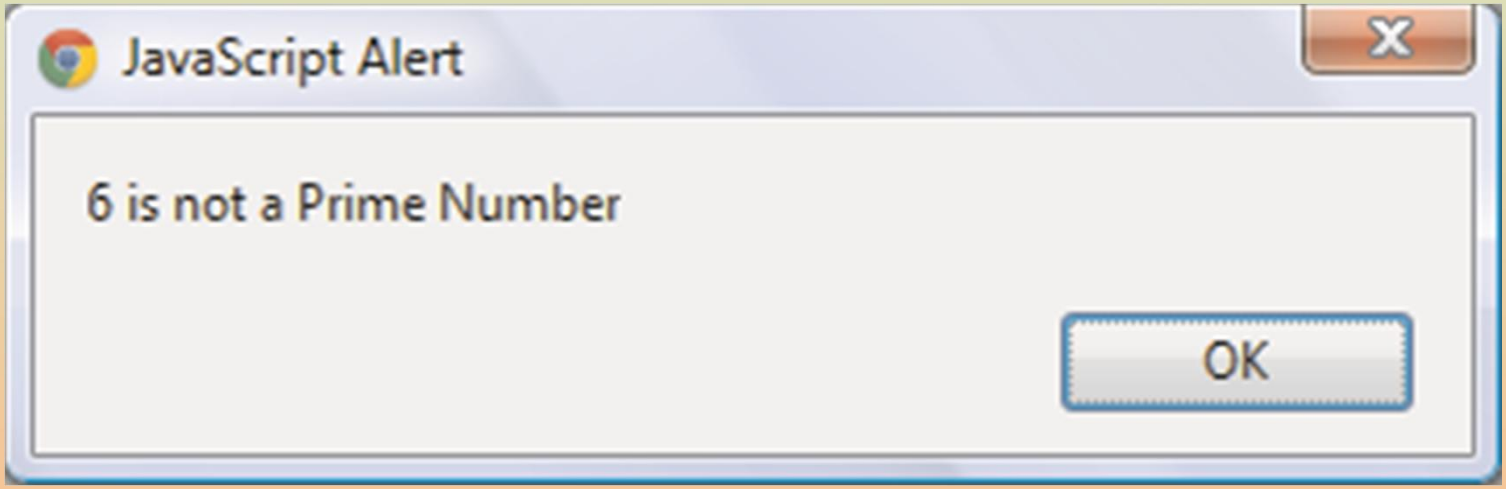
break Statement 2-3

- The Code Snippet demonstrates the script that accepts a number from the user and determines if it is a prime number or not.

```
<script>
var inputNum = parseInt(prompt('Enter number: ',''));
var num = 2;
  while(num <= inputNum-1)
  {
    if(inputNum % num == 0)
       {
      alert(inputNum + ' is not a Prime Number');
        break;
    }
      num++;
  }
  if(num == inputNum)
  {
      alert(inputNum + ' is a Prime Number');
  }
</script>
```

break Statement 3-3

- The code creates a variable, `inputNum`, which is initialized to the number entered by the user.
- The variable `num` is declared and initialized to `2`.
- If the while condition returns `true`, the inner if statement is checked.
- If this condition returns `true`, an alert box is displayed stating that the number is not a prime number.
- The `break` statement is used to exit the entire while loop.
- If the condition evaluates to `false`, the program control is passed to if statement outside the while loop.
- Following figure shows the output of the prime number on accepting number, `6` from the user in the prompt box.



continue Statement 1-3

The `continue` statement is mostly used in the loop constructs and is denoted by the `continue` keyword.

It is used to terminate the current execution of the loop and continue with the next repetition by returning the control to the beginning of the loop.

This means, the `continue` statement will not terminate the loop entirely, but terminates the current execution.

- Following figure shows the flow of execution of the `continue` statement.

continue Statement 2-3

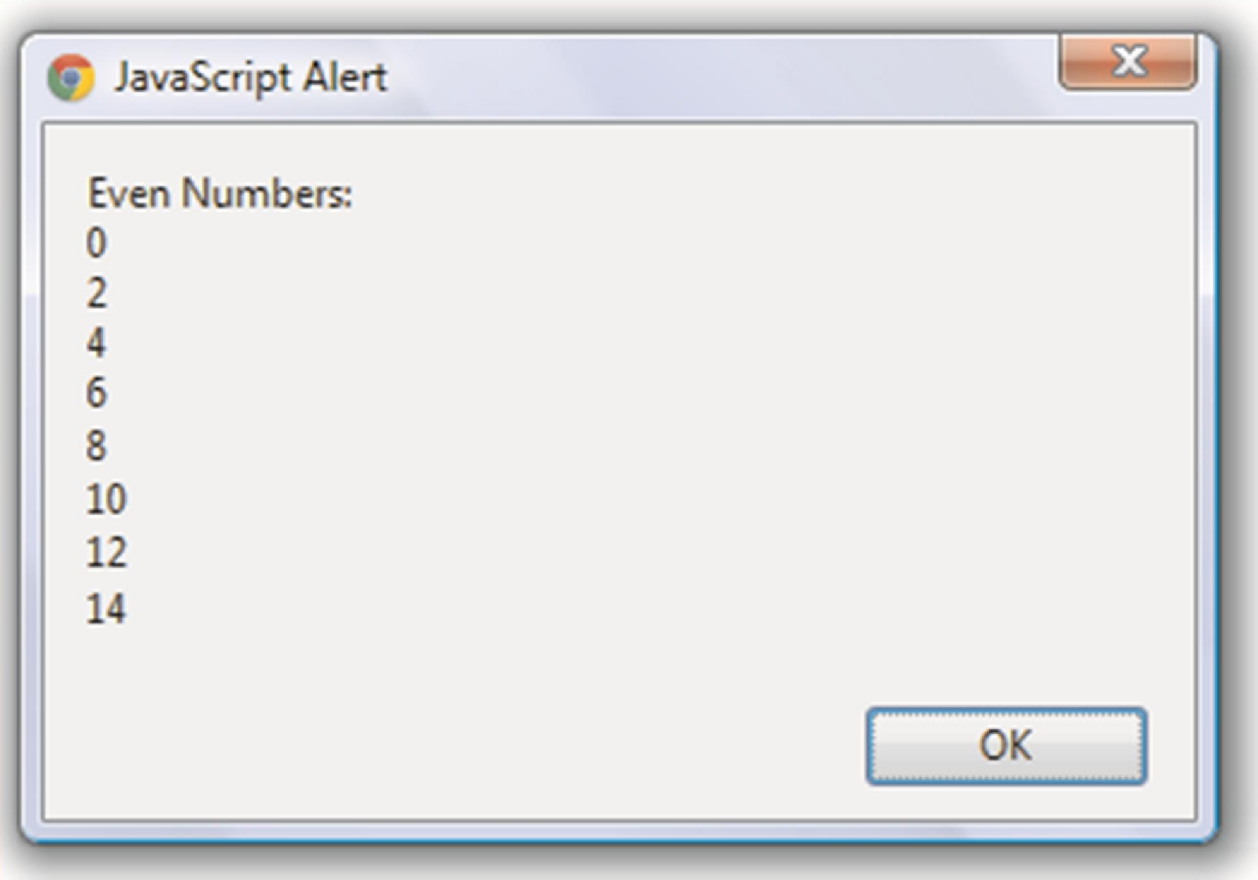
- The Code Snippet displays even numbers from 0 to 15.

```
<script>
  var result = '';
  for (var i = 0; i <= 15; i++)
  {
      if((i%2) != 0)
      {
        continue;
      }
   result = result + i + '\n';
  }
  alert('Even Numbers:\n' + result);
</script>
```

- The code declares a variable, `i`, in the for loop definition and initializes it to value `1`.
- When the value of `i` is divided by zero, the `if` statement checks whether the remainder is equal to zero.

continue Statement 3-3

- If the remainder is zero, the value of `i` is displayed as the value is an even number.
- If the remainder is not equal to `0`, the continue statement is executed.
- It transfers the program control to the beginning of the `for` loop.
- Following figure shows the output of the continue statement.



Arrays

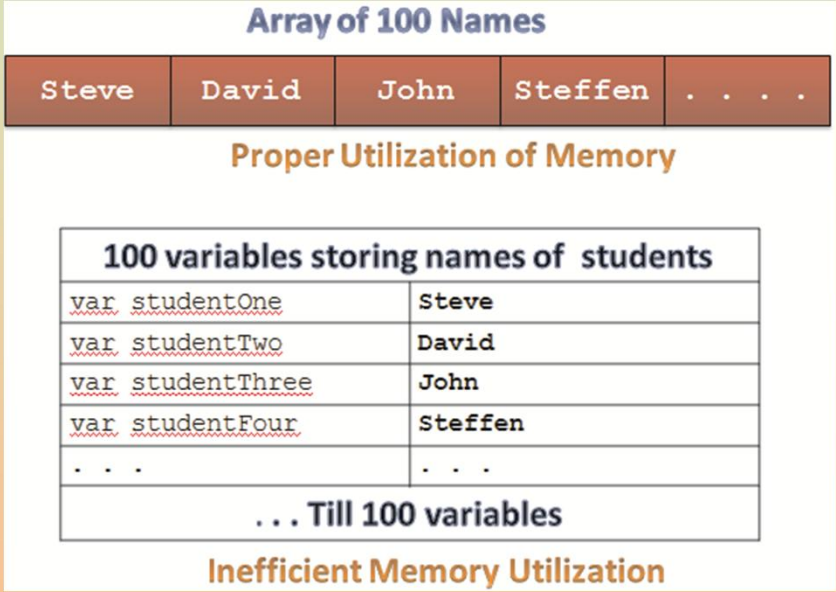
An array is a collection of values stored in adjacent memory locations.

These array values are referenced using a common array name. The values of an array variable must be of the same data type.

These values that are also referred to as elements can be accessed by using subscript or index numbers.

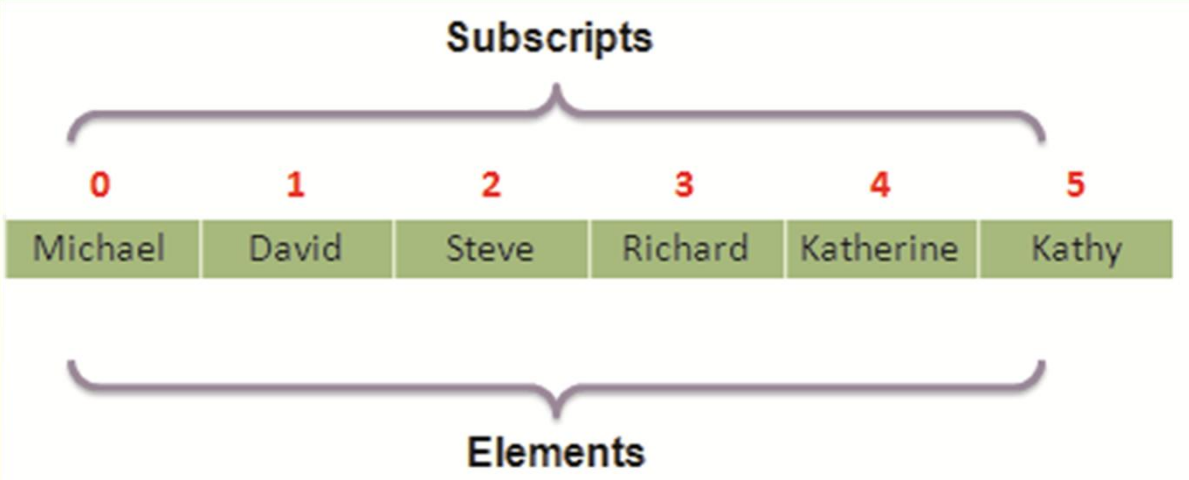
The subscript number determines the position of an element in an array list.

- Following figure shows the effective usage of memory achieved using an array.



Single-dimensional Array 1-3

- JavaScript supports two types of arrays that are as follows:
  - Single-dimensional array
  - Multi-dimensional array
- In a single-dimensional array, the elements are stored in a single row in the allocated memory.
- Following figure shows the allocation of single-dimensional array.



- As shown in the figure, the first array element has the index number zero.
- The last array element has an index number one less than the total number of elements.
- This arrangement helps in efficient storage of data.
- It also helps to sort data easily and track the data length.

Single-dimensional Array 2-3

- The array variable can be created using the `Array` object and `new` keyword along with the size of the array element.
- The syntax to declare and initialize a single-dimensional array is as follows:

Syntax:

```
var variable_name = new Array(size); //Declaration

variable_name[index] = 'value';
```

where,

- `variable_name`: Is the name of the variable.
- `size`: Is the number of elements to be declared in the array.
- `index`: Is the index position.

Single-dimensional Array 3-3

- The Code Snippet shows the different ways to declare and initialize a single-dimensional array.

```
<script>

//Declaration using Array Object and then Initialization
   var marital_status = new Array(3);
   marital_status[0] = 'Single';
   marital_status[1] = 'Married';
   marital_status[2] = 'Divorced';

//Declaration and Initialization
   var marital_status = new
     Array('Single','Married','Divorced');

//Declaration and Initialization Without Array
   var marital_status = ['Single','Married','Divorced'];

</script>
```

Accessing Single-dimensional Arrays 1-4

- Array elements can be accessed by using the array name followed by the index number specified in square brackets.

Accessing Array Elements Without Loops

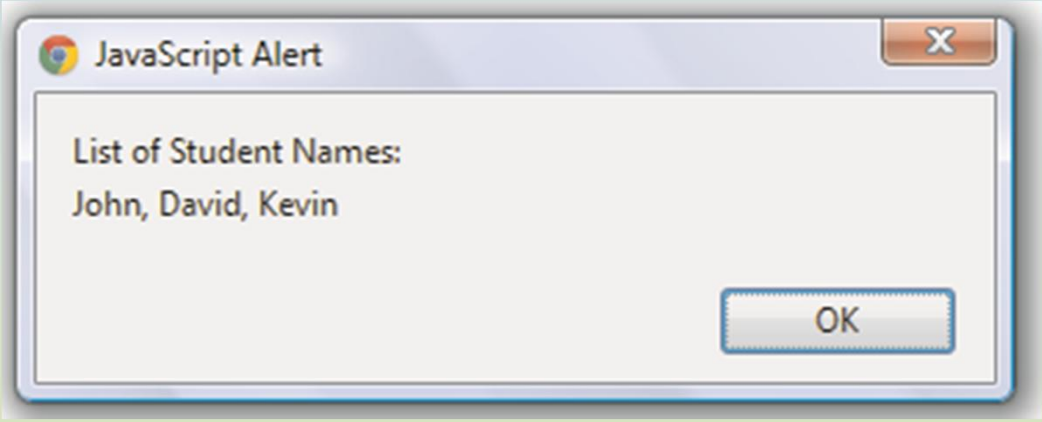
- An array element can be accessed without using loops by specifying the array name followed by the square brackets containing the index number.
- The Code Snippet demonstrates a script that stores and displays names of the students using a single-dimensional array.

```
<script>
  var names = new Array("John", "David", "Kevin");
  alert('List of Student Names:\n' + names[0] + ',' + '
  ' + names[1] + ',' + ' ' + names[2]);
</script>
```

- As shown in the code, `var names = new Array("John","David","Kevin");` declares and initializes an array.
- The `names[0]` accesses the first array element, which is `John`.
- The `names[1]` accesses the second array element, which is `David`.
- The `names[2]` accesses the third array element, which is `Kevin`.

Accessing Single-dimensional Arrays 2-4

- Following figure displays the names of the students.



➢ Accessing Array Elements With Loops

- JavaScript allows you to access array elements by using different loops.
- Thus, you can access each array element by putting a counter variable of the loop as the index of an element.
- However, this requires the count of the elements in an array.
- The `length` property can be used to determine the number of elements in an array.

Accessing Single-dimensional Arrays 3-4

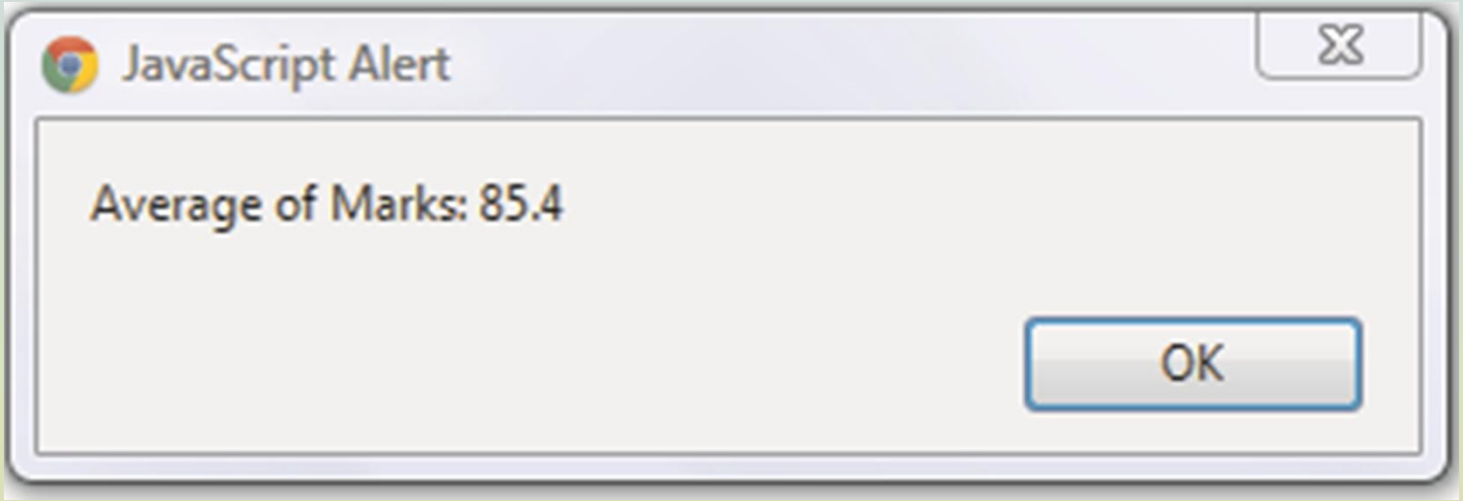
- The Code Snippet demonstrates the script that creates an array to accept the marks of five subjects and display the average.

```
<script>
  var sum = 0;
  var marks = new Array(5);
  for(var i=0; i<marks.length; i++)
  {
    marks[i] = parseInt(prompt('Enter Marks:', ''));
    sum = sum + marks[i];
  }
  alert('Average of Marks: ' + (sum/marks.length));
</script>
```

- In the code, `var marks = new Array(5);` declares an array of size `5`.
- It displays a prompt box that accepts the marks for a subject in each iteration.
- Then, the code calculates and displays the average marks.

Accessing Single-dimensional Arrays 4-4

- Following figure displays the average of the marks, 90, 75, 85, 95, and 82 accepted from the user in the prompt box.



Multi-dimensional Array 1-2

- A multi-dimensional array stores a combination of values of a single type in two or more dimensions.
- These dimensions are represented as rows and columns similar to those of a Microsoft Excel sheet.
- A two-dimensional array is an example of the multi-dimensional array.
- Following figure shows the representation of a multi-dimensional array.

Employee Salaries → ↓	0 BASIC	1 HRA	2 ALLOWANCE	3 TOTAL
0	14350	10500	1500	26350
1	34350	4050	1000	39400
2	6150	4500	3250	13900
3	4920	4500	2250	11670
4	12300	9000	2000	23300

- A two-dimensional array is an array of arrays.
- This means, for a two-dimensional array, first a main array is declared and then, an array is created for each element of the main array.

Multi-dimensional Array 2-2

- The syntax to declare a two-dimensional array is as follows:

Syntax:

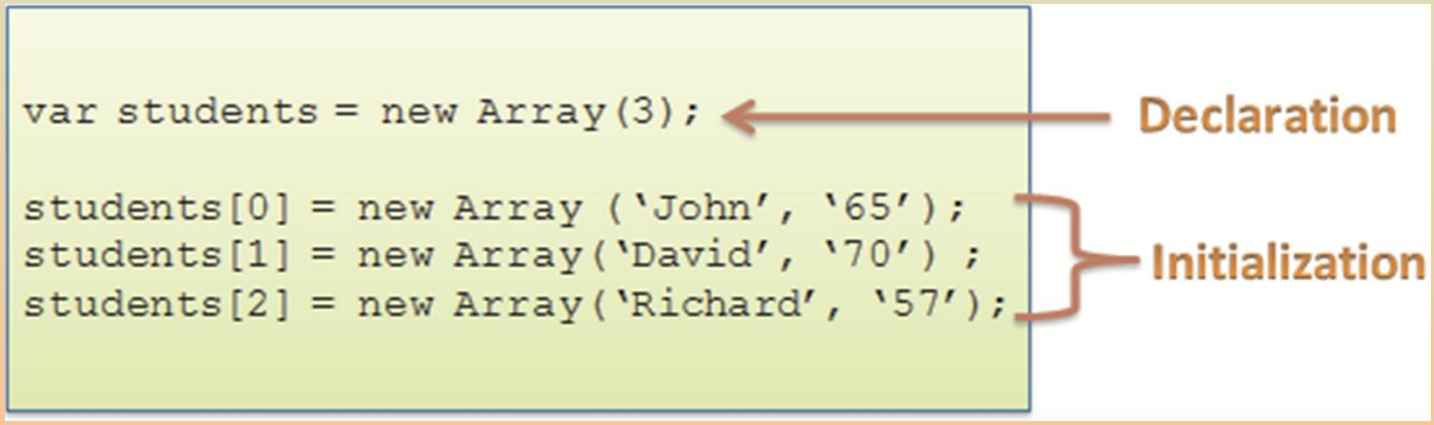
```
var variable_name = new Array(size); //Declaration

variable_name[index] = new Array('value1','value2'..);
```

where,

- `variable_name`: Is the name of the array.
- `index`: Is the index position.
- `value1`: Is the value at the first column.
- `value2`: Is the value at the second column.

- Following figure shows the declaration of a two-dimensional array.



Accessing Two-dimensional Arrays 1-4

- Multi-dimensional arrays can be accessed by using the index of main array variable along with index of sub-array.

➢ Accessing Array Elements Without Loops

- The Code Snippet creates a two-dimensional array that displays the employee details.

```
<script>
  var employees = new Array(3);
  employees[0] = new Array('John', '25', 'New Jersey');
  employees[1] = new Array('David', '21', 'California');
  document.write('<H3> Employee Details </H3>');
  document.write('<P><B>Name: </B>' + employees[0][0] +
  '</P>');
  document.write('<P><B>Location: </B>' + employees[0][2]
  + '</P>');
  document.write('<P><B>Name: </B>' + employees[1][0]
  + '</P>');
  document.write('<P><B>Location: </B>' + employees[1][2]
  + '</P>');
</script>
```

Accessing Two-dimensional Arrays 2-4

- In the code, `var employees = new Array(3)` creates an array of size `3`.
- The declaration `employees[0] = new Array('John','23','New Jersey')` creates an array at the `0`th row of the employees array.
- Similarly, `employees[1] = new Array('David','21','California')` creates an array at the first row of the employees array.
- Following figure displays the employee details.

Accessing Two-dimensional Arrays 3-4

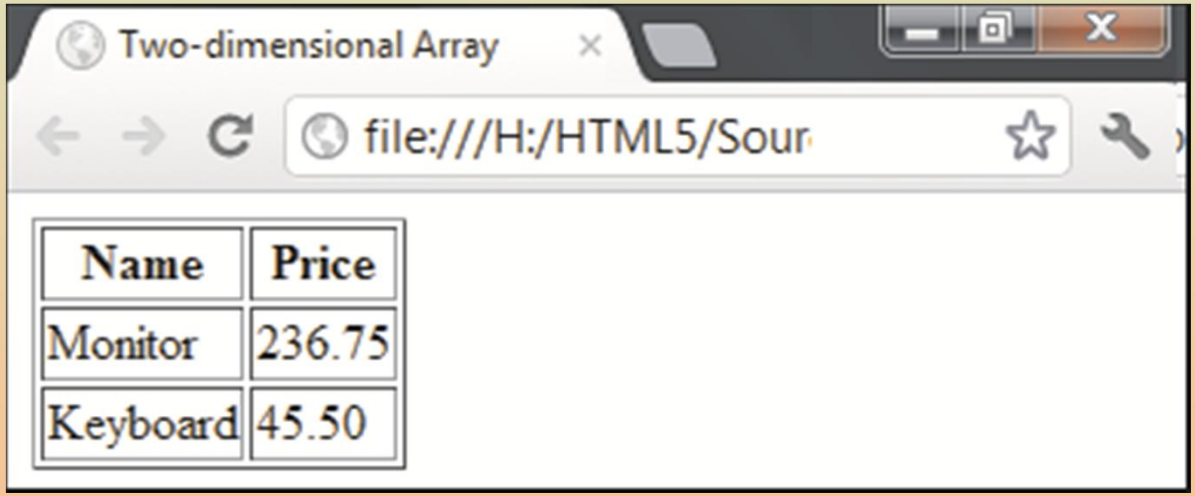
Accessing Array Elements With Loops

- The Code Snippet creates a two-dimensional array to display the product details.

```
<script>
  var products = new Array(2);
  products[0] = new Array('Monitor', '236.75');
  products[1] = new Array('Keyboard', '45.50');
  document.write('<TABLE border=1><TR><TH>Name</TH>
   <TH>Price</TH></TR>');
  for(var i=0; i<products.length; i++)
  {
    document.write('<TR>');
    for(var j=0; j<products[i].length; j++)
    {
      document.write('<TD>' + products[i][j] + '</TD>');
    }
    document.write('</TR>');
  }
    document.write('</TABLE>');
</script>
```

Accessing Two-dimensional Arrays 4-4

- In the code, `products[0] = new Array('Monitor','236.75')` creates an array at the `0`th row of the `products` array.
- Similarly, `products[1] = new Array('Keyboard','45.50')` creates an array at the first row of the `products` array.
- The condition, `i < products.length`, specifies that the counter variable, `i`, should be less than the number of rows in the array variable, `products`.
- For each row in the array, the condition, `j < products[i].length` specifies that the counter variable, `j`, should be less than the number of columns specified the `i`th row of the array variable, `products`.
- Finally, `document.write("<TD>" + products[i][j] + "</TD>")` displays the values at the `i`th row and `j`th column of array variable, `products`.
- Following figure displays the product details in a table.



Array Methods 1-3

An array is a set of values grouped together and identified by a single name. In JavaScript, the `Array` object allows you to create arrays.

It provides the `length` property that allows you to determine the number of elements in an array.

The various methods of the `Array` object allow to access and manipulate the array elements.

- Following table lists the most commonly used methods of the object.

Data Type	Description
`concat`	Combines one or more array variables.
`join`	Joins all the array elements into a string.
`pop`	Retrieves the last element of an array.
`push`	Appends one or more elements to the end of an array.
`sort`	Sorts the array elements in an alphabetical order.

Array Methods 2-3

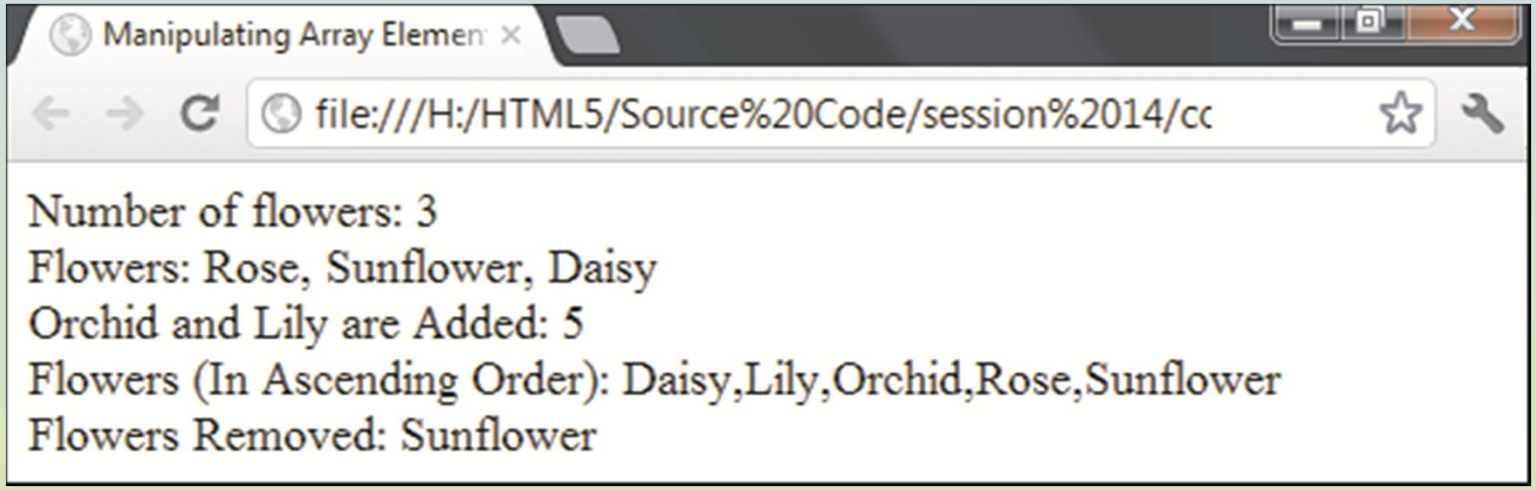
- The Code Snippet demonstrates how to access and manipulate the array elements.

```
<script>
  var flowers = new Array('Rose', 'Sunflower', 'Daisy');
  document.write('Number of flowers: ' + flowers.length +
  '<br/>');
  document.write('Flowers: ' + flowers.join(', ') +
  '<br/>');
  document.write('Orchid and Lily are Added: ' +
  flowers.push("Orchid", "Lily") + '<br/>');
  document.write('Flowers (In Ascending Order): ' +
  flowers.sort() + '<br/>');
  document.write('Flowers Removed: ' + flowers.pop()
  +'<br/>');

</script>
```

Array Methods 3-3

- Following figure displays the corresponding output of the script.



- In the code, an array variable, `flowers`, is created that stores the names of three flowers.
- The `length` property is used to display the number of flowers in the array variable.
- The `join()` method joins the flower names and separates them with a comma.
- The `push()` method adds orchid and lily at the end of the array and the total number of flowers in the array list is displayed as `5`.
- The `sort()` method sorts the flowers in alphabetical order.
- The `pop()` method retrieves the last element that is `Sunflower`, from the array list.

for..in Loop 1-3

The `for..in` loop is an extension of the `for` loop. It enables to perform specific actions on the arrays of objects.

The loop reads every element in the specified array and executes a block of code only once for each element in the array.

Syntax:

```
for (variable_name in array_name)
{
     //statements;
}
```

where,

- `variable_name`: Is the name of the variable.
- `array_name`: Is the array name.

for..in Loop 2-3

- The Code Snippet demonstrates how to display elements from an array using the `for..in` loop.

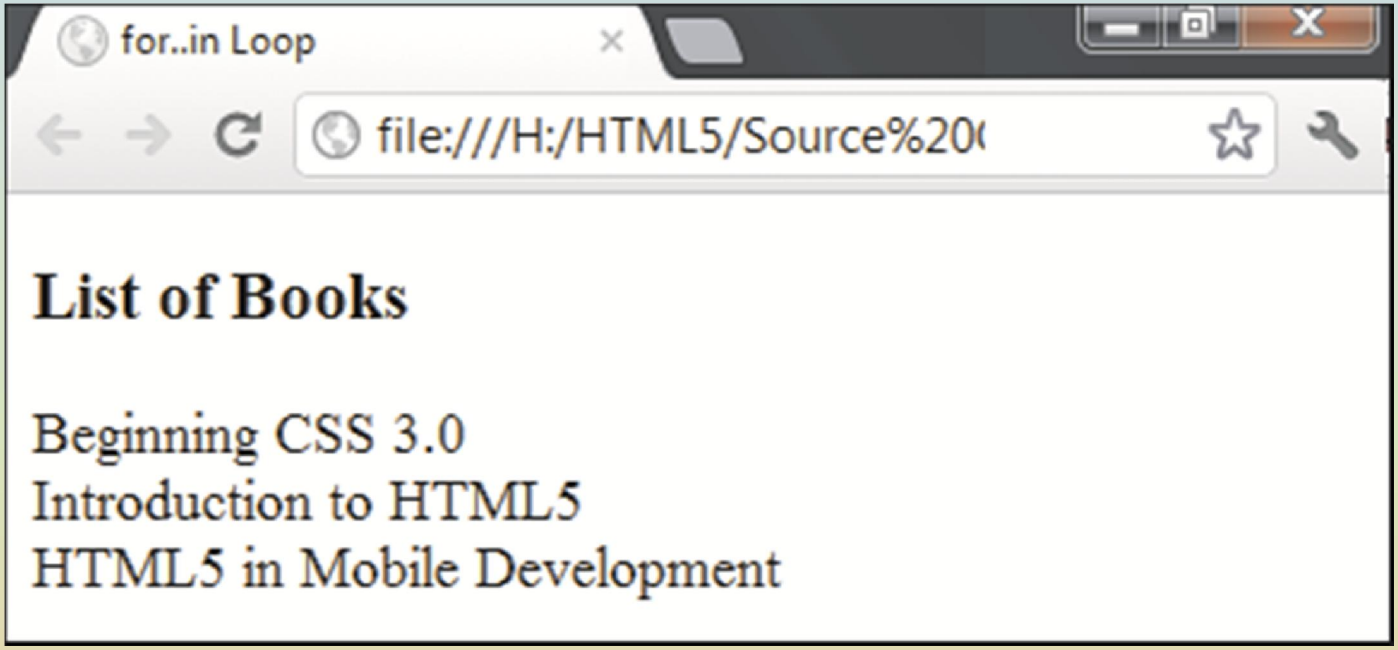
```
<script>
  var books = new Array('Beginning CSS 3.0',
    'Introduction to HTML5', 'HTML5 in Mobile
     Development');

  document.write('<H3> List of Books </H3>');

  for(var i in books)
  {
     document.write(books[i] + '<br/>');
  }
</script>
```

for..in Loop 3-3

- Following figure displays the corresponding output of the script.



Summary

- A loop construct consists of a condition that instructs the compiler the number of times a specific block of code will be executed.
- JavaScript supports three types of loops that include: while loop, for loop, and do-while loop.
- The break statement is used to exit the loop without evaluating the specified condition.
- The continue statement terminates the current execution of the loop and continue with the next repetition by returning the control to the beginning of the loop.
- JavaScript supports two types of arrays namely, Single-dimensional array and Multi-dimensional array.
- The for..in loop is an extension of the for loop that enables to perform specific actions on the arrays of objects.