

Introduction to JavaScript

greenwich.edu.vn



Alliance with  Education

What Is JavaScript?

NOTE

*JavaScript was standardized in 1996 by the European Computer Manufacturer's Association (ECMA), which is why you sometimes hear it called **ECMAScript**.*

- It is a client-side scripting language
- runs on the user's machine and not on the server
- That means JavaScript (and the way we use it) is reliant on the browser's capabilities and settings

What JavaScript can do

Whoops! Some errors occurred.

- That username is already in use.
- Email confirmation doesn't match

what can javascript do

what can javascript do

what can javascript be used for

what can javascript do for a website

what can javascript programs do

Confirm Password

- **Embedded script**

`<script>`

... JavaScript code goes here

`</script>`

- **External scripts**

- `<script src="my_script.js"></script>`

- **Script placement**

- The **script** element go anywhere in the document, but the most common places for scripts are in the **head** of the document and at the very end of the **body**

The Anatomy of a Script

- Variable
- Statement
- Data type
- Operator
- Branch
- Loop
- Native functions
- Arguments
- Return of a function

Variables and Data Type

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

Example

```
var x = 5;  
var y = 6;  
var z = x + y;
```

[Try it Yourself >>](#)

Variables and Data Type

JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more:

```
var length = 16;           // Number
var lastName = "Johnson"; // String
var cars = ["Saab", "Volvo", "BMW"]; // Array
var x = {firstName:"John", lastName:"Doe"}; // Object
```

JavaScript Has Dynamic Types

JavaScript has dynamic types. This means that the same variable can be used as different types:

Example

```
var x;           // Now x is undefined
var x = 5;       // Now x is a Number
var x = "John";  // Now x is a String
```


Types of Operators

- Operators help in simplifying expressions.
- JavaScript provides a predefined set of operators that allow you to perform different operations.
 - Arithmetic operators
 - Relational operators
 - Logical operators
 - Assignment operators
 - Bitwise operators
 - Special operators

Few JavaScript Expressions

```
• var basicSal = salPerDay * presentDays;  
• var tax = basicSal * (5 / 100);  
• var salary = basicSal - tax;
```

Binary Operators

```
var addNum = 7267 + 105;  
var subNum = 4523 - 4000;  
if (addNum > subNum)  
    document.writeln("AddNum is greater than SubNum.");
```

Relational Operator

Arithmetic Operator

- Arithmetic operators are binary operators, as they perform basic arithmetic operator on two operand.

Arithmetic Operator	Description	Example
+ (Addition)	Performs addition. In case of string values, it behaves as a string concatenation operator and appends a string at the end of the other.	45 + 56
/ (Division)	Performs division. It divides the first operand by the second operand and returns the quotient.	24 / 8
% (Modulo)	Performs the modulo operation. It divides the first operand by the second operand and returns the remainder.	90 % 20
* (Multiplication)	Performs multiplication. It multiplies the two operands.	98 * 10
- (Subtraction)	Performs subtraction. If a larger value is subtracted from a smaller value, it returns a negative numeric value.	78 - 76

Increment and decrement operators

- The increment operator (++) increases the value by 1, while the decrement operator(--) decreases the value by 1.

Expression	Type	Result
<code>numTwo = ++numOne;</code>	Pre-increment	<code>numTwo = 3</code>
<code>numTwo = numOne++;</code>	Post-increment	<code>numTwo = 2</code>
<code>numTwo = --numOne;</code>	Pre-decrement	<code>numTwo = 1</code>
<code>numTwo = numOne--;</code>	Post-decrement	<code>numTwo = 2</code>
<code>-(numOne)</code>	-	-2

Relational Operators

- Relational operators are binary operators that make compare between two operands. They return a Boolean value namely, true or false.

Relational Operator	Description	Example
<code>==</code> (Equal)	Verifies whether the two operands are equal.	<code>90 == 91</code>
<code>!=</code> (Not Equal)	Verifies whether the two operands are unequal.	<code>99 != 98</code>
<code>===</code> (Strict Equal)	Verifies whether the two operands are equal and whether are of the same type.	<code>3 === 4</code>
<code>!==</code> (Strict Not Equal)	Verifies whether the two operands are unequal and whether are not of the same type.	<code>3 !== "3"</code>
<code>></code> (Greater Than)	Verifies whether the left operand is greater than the right operand.	<code>97 > 95</code>
<code><</code> (Less Than)	Verifies whether the left operand is less than the right operand.	<code>94 < 96</code>
<code>>=</code> (Greater Than or Equal)	Verifies whether the left operand is greater than or equal to the right operand.	<code>92 >= 93</code>
<code><=</code> (Less Than or Equal)	Verifies whether the left operand is less than or equal to the right operand.	<code>99 <= 100</code>

Logical Operators

- Logical operators are binary operators that perform logical operations on two operands.

Logical Operator	Description	Example
<code>&&</code> (AND)	Returns <code>true</code> if both the operands are evaluated to <code>true</code> or returns <code>false</code> . If first operand evaluates to <code>false</code> , it will ignore the second operand and will return <code>false</code> .	<code>(x == 2) && (y == 5)</code> Returns <code>false</code> .
<code>!</code> (NOT)	Returns <code>false</code> if the expression is true and vice-versa.	<code>!(x==3)</code> Returns <code>true</code> .
<code> </code> (OR)	Returns <code>true</code> if either of the operands are evaluated to <code>true</code> . If first operand evaluates to <code>true</code> , it will ignore the second operand and will return <code>true</code> .	<code>(x == 2) (y == 5)</code> Returns <code>true</code> .

Assignment Operators

Expression	Description	Result
<code>numOne += 6;</code>	<code>numOne = numOne + 6</code>	<code>numOne = 12</code>
<code>numOne -= 6;</code>	<code>numOne = numOne - 6</code>	<code>numOne = 0</code>
<code>numOne *= 6;</code>	<code>numOne = numOne * 6</code>	<code>numOne = 36</code>
<code>numOne %= 6;</code>	<code>numOne = numOne % 6</code>	<code>numOne = 0</code>
<code>numOne /= 6;</code>	<code>numOne = numOne / 6</code>	<code>numOne = 1</code>

Bitwise Operators

Bitwise Operator	Description	Example
& (Bitwise AND)	Compares two bits and returns 1 if both of them are 1 or else returns 0.	00111000 & 00011100 Returns 00011000.
~ (Bitwise NOT)	Inverts every bits of the operand and is a unary operator.	~00010101 Returns 11101010.
 (Bitwise OR)	Compares two bits and returns 1 if the corresponding bits of either or both the operands is 1.	00111000 00011100 Returns 00111100.
& (Bitwise AND)	Compares two bits and returns 1 if both of them are 1 or else returns 0.	00111000 & 00011100 Returns 00011000.
~ (Bitwise NOT)	Inverts every bits of the operand and is a unary operator.	~00010101 Returns 11101010.
 (Bitwise OR)	Compares two bits and returns 1 if the corresponding bits of either or both the operands is 1.	00111000 00011100 Returns 00111100.
^ (Bitwise XOR)	Compares two bits and returns 1 if the corresponding bit of either but not both the operands is 1.	00111000 00011100 Returns 00100100.

Special Operators

Special Operator	Description
<code>,</code> comma	Combines multiple expressions into a single expression, operates on them in the left to right order, and returns the value of the expression on the right.
<code>?:</code> conditional	Operates on three operands, where the result depends on a condition. It is a ternary operator and has the form <code>condition ? value1:value2</code> . If the condition is true, the operator obtains <code>value1</code> or else obtains <code>value2</code> .
<code>typeof</code>	Returns a string that indicates the type of the operand. The operand can be a string, variable, keyword, or an object.

String Operators

Operator	Description	Example
+	Concatenates two or more strings into a single string	<pre>var fullname='John' + ' ' + 'Blake'; fullname='John Blake'</pre>
+= (Add-By-Value)	Concatenates two or more strings and assign the result to its left operand	<pre>var flower='Rose'; flower += ' ' + 'flower'; flower ='Rose Flower';</pre>

Shift Operators

Operator	Description	Example
<< (Left Shift Operator)	Shift the bit positions towards the left by shifting the number of bits in the left at the right	8<<2 yield 32,because 00001000 shifted by 2 bits becomes 00100000,which is 32. Here, the two leftmost bits are placed toward the right
>> (Right Shift Operator)	Shift the bit positions towards the right by shifting the number of bits in the right at the left	8>>2 yield 2,because 00001000 shifted by 2 bits becomes 00000010,which is 2. Here, the two rightmost bits are placed toward the left.

Decision – making Statement

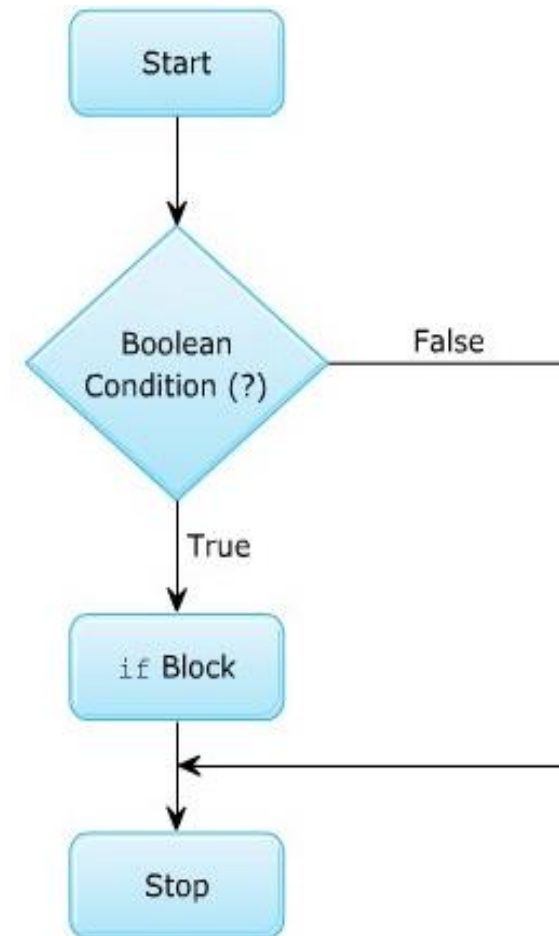
- ❑ Decision-making statements allow implementing logical decision for executing different block to obtain the desired output. They execute a block of statements depending upon a Boolean condition.
- ❑ This condition is an expression that returns either true or false.
- ❑ JavaScript supports four decision-making statements :
 - **If**
 - **If-else**
 - **If-else if else**
 - **switch**

The “if” Statement

- The if statement executes a block of statements based on a logical Boolean condition.

The syntax demonstrates how to use the if statement.

```
if (condition)
{
    //one or more statements;
}
```

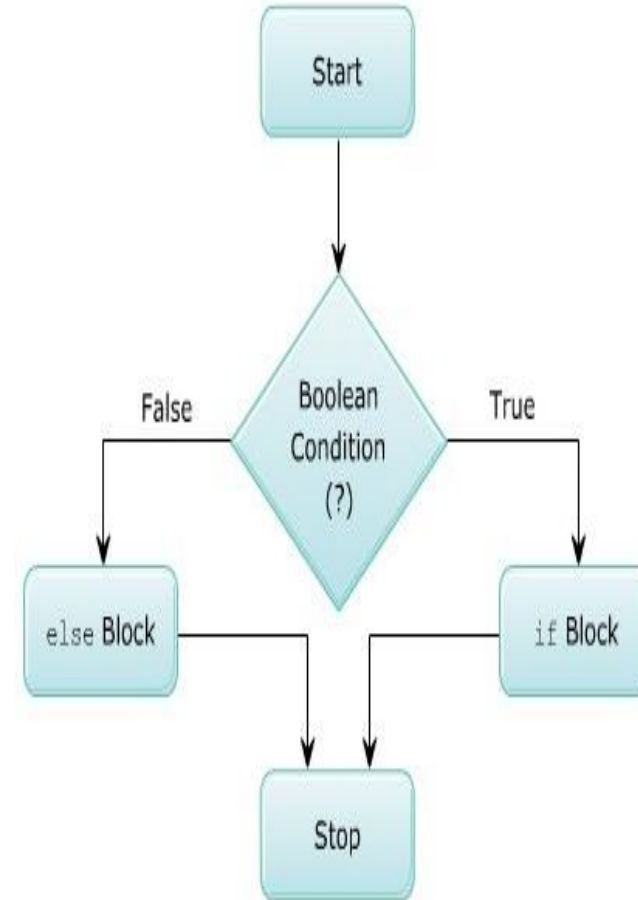


The “if-else” Statement

The **if** statement specifies a block of statement to be executed when the condition in the **if** statement is **true**, and it's requires to define a block of statements to be executed when a conditions evaluated to **false**

The syntax demonstrates how to use the if..else statement.

```
if (condition)
{
    // one or more statements;
}
else
{
    //one or more statements;
}
```

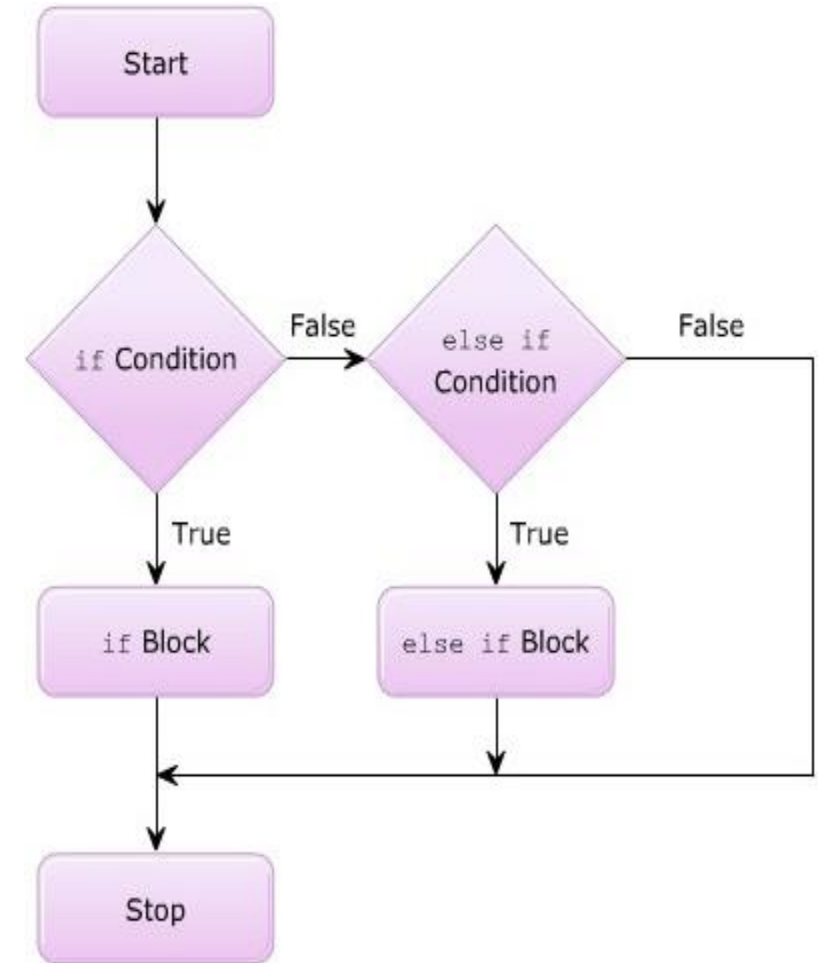


The “if-else if ” Statement

The **if-else if** statements allow you to check multiple conditions and specify a different block to be execute for each condition.

The syntax demonstrates how to use the if-else if statements

```
if (condition)
{
    // one or more statements;
}
else if (condition)
{
    // one or more statements;
}
else
{
    // one or more statements;
}
```

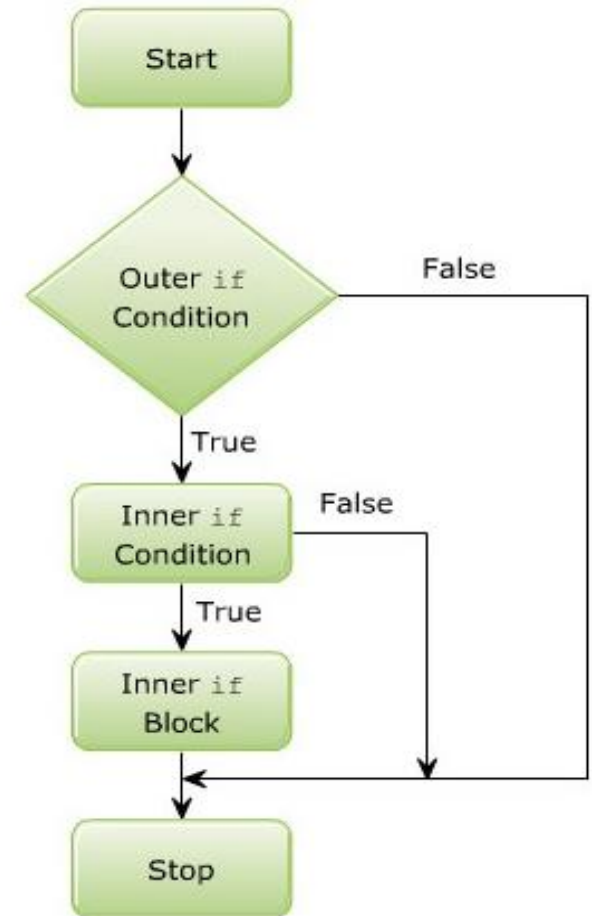


Nested “if” Statement

The nested if statements comprises of multiple if statements within an if statement.

The syntax demonstrates how to use the nested if statements.

```
if (condition)
{
    // one or more statements;
    if (condition)
    {
        // one or more statements;
        if (condition)
        {
            // one or more statements;
        }
    }
}
```



“Switch-case” Statement

To simplify the coding and avoid using multiple if statement, switch-case statement can be used as a different approach to code the same logic

```
switch(expression/variable)
{
    case value1:
        //statements;
        break;
    case value2:
        //statements;
        break;
    .....
    case valueN:
        //statements;
        break;
    default:
        //default statement
}
```


What is a loop?

- Loop is a section of code in a program which is executed repeatedly, until a specific condition is satisfied.
- There are three type of loop structures:

The **while** loop

The **do-while** loop

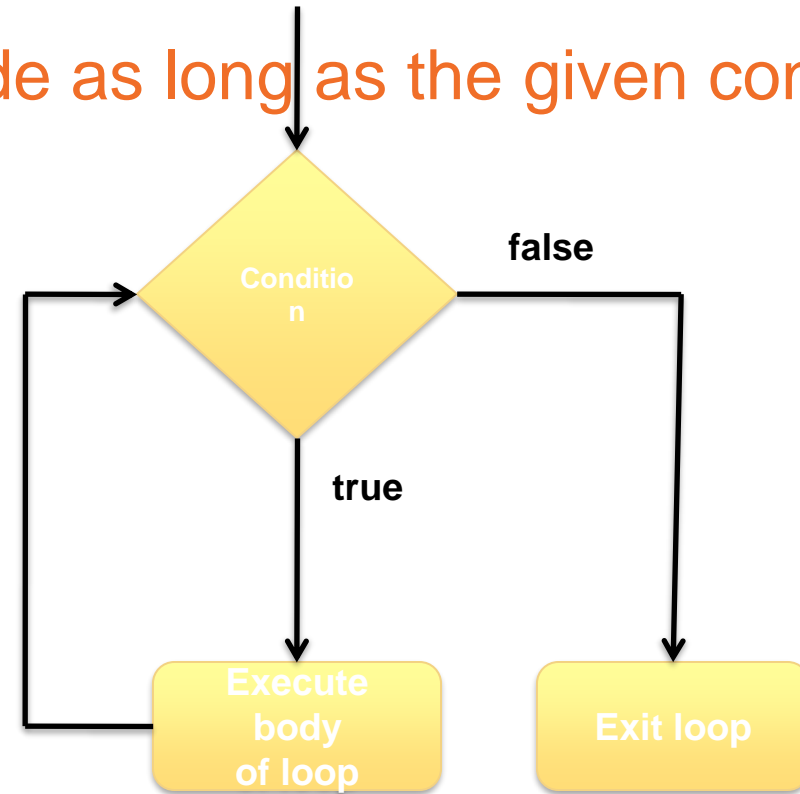
The **for** loop

The **while** loop

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

- Syntax:

```
while (condition)
{
    statement(s) ;
}
```



The **while** loop: Demo

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>while loop demo</title>
    <script language="javascript" type="text/javascript">
      var n=20;
      var i=0;
      document.write("Display the odd number <br>");
      while(i<=n)
      {
        if(i%2 == 0)
          document.write(i + "\t");
        i++;
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

The **do-while** loop

- The **do-while** loop is similar to the while loop, but the **do-while** loop evaluates the condition at the end of the loop. So that, the **do-while** loop executes at least once.
- Syntax:

```
do  
{  
    statement(s) ;  
}while (condition) ;
```

The do-while loop: Demo

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>while loop demo</title>
    <script language="javascript" type="text/javascript">
      var n=20;
      var i=0;
      document.write("Display the even number <br>");
      do
      {
        if(i%2 == 0)
          document.write(i + "\t");
        i++;
      } while(i<=n);
    </script>
  </head>
  <body>
  </body>
</html>
```

The **for** loop

- **Syntax:**

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

- The initialization is an assignment statement that sets the loop control variable, before entering the loop.
- The condition is a relational expression, which determines, when the loop will exit.
- The increment/decrement defines how the loop control variable changes, each time the loop is executed.


The **for** loop: Demo

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>for loop demo</title>
    <script language="javascript" type="text/javascript">
      var n=20;
      var i=0;
      document.write("Display the even number <br>");
      for(i=0; i<n; i++)
      {
        if(i%2 == 0)
          document.write(i + "\t");
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

break statement

- The **break** statement can be used in the **switch-case** and **loop** constructs. It is used to exit the loop without evaluating the specified condition.

```
for(initialization; condition; increment/decrement)
{
    ...
    if(condition)
        break;
    ...
}
```



The diagram illustrates the effect of the `break` statement. A red box encloses the loop body (the code between the opening and closing curly braces). A red arrow originates from the right side of this box and points to the end of the loop, indicating that the `break` statement causes an immediate exit from the loop, bypassing the increment/decrement step and the condition check.

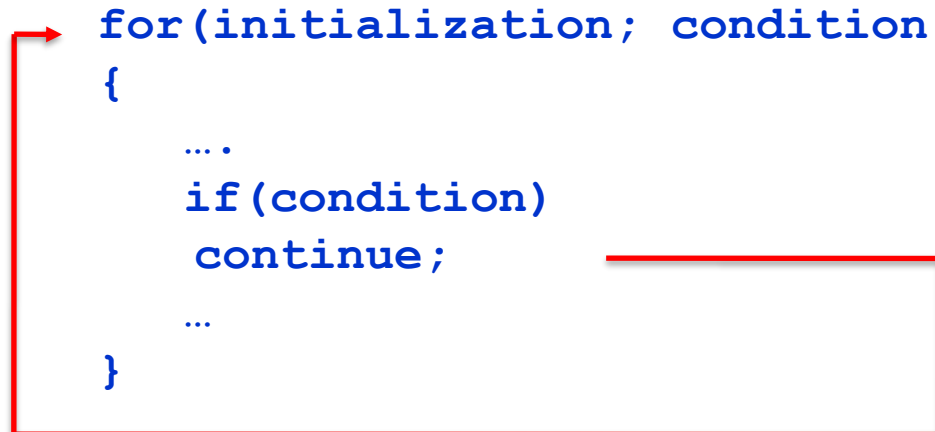
break statement: Demo

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Using break statement</title>
    <script language="javascript" type="text/javascript">
      var n = prompt("Enter the number n:");
      var i=0;
      for(i=2; i<n; i++)
      {
        if(n % i == 0)
        {
          document.write(n + " is not a prime number.");
          break;
        }
      }
      if(i == n)
        document.write(n + " is prime number.");
    </script>
  </head>
</html>
```

continue statement

- The **continue** statement is mostly used in the loop constructs. 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.

```
for(initialization; condition; increment/decrement)
{
    ...
    if(condition)
        continue;
    ...
}
...
```



continue statement: Demo

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>continue statement demo</title>
    <script language="javascript" type="text/javascript">
      var n=20;
      var i=0;
      document.write("Display the even numbers <br>");
      for(i=0; i<=n; i++)
      {
        if(i%2 == 1)
          continue;
        else
          document.write(i + "\t");
      }
    </script>
  </head>
</html>
```

The Browser Object

Property/method	Description
event	Represents the state of an event
history	Contains the URLs the user has visited within a browser window
location	Gives read/write access to the URI in the address bar
status	Sets or returns the text in the status bar of the window
alert()	Displays an alert box with a specified message and an OK button
close()	Closes the current window
confirm()	Displays a dialog box with a specified message and an OK and a Cancel button
focus()	Sets focus on the current window

- Most common event handlers.

Event handler	Event description
onblur	An element loses focus
onchange	The content of a form field changes
onclick	The mouse clicks an object
onerror	An error occurs when the document or an image loads
onfocus	An element gets focus
onkeydown	A key on the keyboard is pressed
onkeypress	A key on the keyboard is pressed or held down
onkeyup	A key on the keyboard is released
onload	A page or an image is finished loading
onmousedown	A mouse button is pressed
onmousemove	The mouse is moved
onmouseout	The mouse is moved off an element
onmouseover	The mouse is moved over an element
onmouseup	A mouse button is released
onsubmit	The submit button is clicked in a form

Summary

- What is JavaScript
- Adding JavaScript to a page
- Anatomy of JavaScript
- Browser Object
- Events