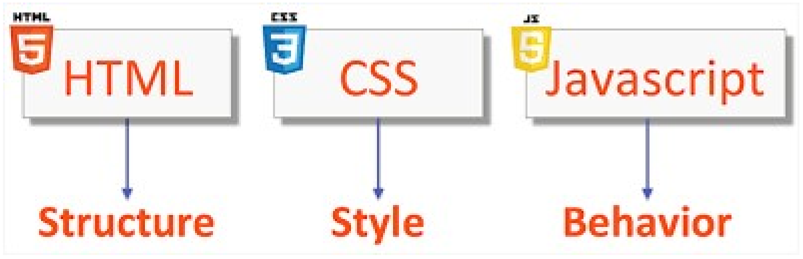
1 COURSE OVERVIEW

- Introduction to JavaScript.
- Basics.
- Variables and Values.
- Control and Looping.
- Objects and Properties.
- Functions.
- Events.
- Ajax.
- Jquery.
- OOP - Object Oriented Programing.

1.1 WHAT IS JAVASCRIPT?

- Initially created to **“make webpages alive”**
- A **lightweight** programming language that runs in a Web browser
- Interpreted, not compiled
- Java Scripts is **NOT JAVA**
- https://insights.stackoverflow.com/survey/2018



1.2 WHAT CAN JAVASCRIPT DO?

- With JavaScript web pages are dynamic.
- JavaScript can put dynamic text into an HTML page.
- JavaScript can **react to events**.
- JavaScript can be used to **validate input data**.
- This allows you to make parts of your web pages appear or disappear or move around on the page.
- JavaScript can change HTML content.
- JavaScript can change HTML attribute values.
- JavaScript can change HTML styles (CSS).
- JavaScript can hide HTML elements.
- JavaScript can show HTML elements.

2 WRITING A JAVASCRIPT PROGRAM

A **computer program** is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called **statements**. A **JavaScript program** is a list of programming **statements**.

❖ *In HTML, JavaScript programs are executed by the web browser.*

- The Web browser runs a JavaScript program when the Web page is first loaded, or in response to an event.
- JavaScript programs can either be placed directly into the HTML file or they can be saved in external files.
- Placing a program in an external file allows you to hide the program code from the user.
- Source code placed directly in the HTML file can be viewed by anyone.

❖ *The <script> Tag: In HTML, JavaScript code must be inserted between <script> and </script> tags.*

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

2.1 IMPLEMENTING JAVASCRIPT

You can place any number of scripts in an HTML document. Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

Scripts can also be placed in external files. External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension `.js`. To use an external script, put the name of the script file in the `src` (source) attribute of a `<script>` tag.

You can place an external script reference in `<head>` or `<body>` as you like. The script will behave as if it was located exactly where the <script> tag is located.

2.1.1 External References

- External scripts can be referenced with a full URL or with a path relative to the current web page. This example uses a full URL to link to a script:

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

- This example uses a script located in a specified folder on the current web site:

```
<script src="/js/myScript1.js"></script>
```

- This example links to a script located in the same folder as the current page:

```
<script src="myScript1.js"></script>
```

2.1.2 External JavaScript Advantages

- Placing scripts in external files has some advantages:
  - It separates HTML and code.
  - It makes HTML and JavaScript easier to read and maintain.
  - Cached JavaScript files can speed up page loads.

- To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

3 STATEMENTS

JavaScript statements are composed of: **Values**, **Operators**, **Expressions**, **Keywords**, and **Comments**.

Most JavaScript programs contain many JavaScript statements. The statements are executed, one by one, in the same order as they are written.

❖ *JavaScript code: Is the JavaScript programs (and JavaScript statements).*

3.1 SEMICOLONS (;)

- Semicolons separate JavaScript statements. Add a semicolon at the end of each executable statement:

```
var a, b, c;       // Declare 3 variables
a = 5;             // Assign the value 5 to a
b = 6;             // Assign the value 6 to b
c = a + b;         // Assign the sum of a and b to c
```

- When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

❖ *On the web, you might see examples without semicolons. Ending statements with semicolon is not required, but highly recommended.*

3.2 JAVASCRIPT WHITE SPACE

- JavaScript ignores multiple spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

- A good practice is to put spaces around operators (= + - * /):

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

3.3 JAVASCRIPT LINE LENGTH AND LINE BREAKS

For best readability, programmers often like to avoid code lines longer than 80 characters. If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

3.4 JAVASCRIPT CODE BLOCKS

JavaScript statements can be grouped together in code blocks, inside **curly brackets {...}**. The purpose of code blocks is to define statements to be executed together. One place you will find statements grouped together in blocks, is in **JavaScript functions**:

```
function myFunction() {
    document.getElementById("demo1").innerHTML = "Hello Dolly!";
    document.getElementById("demo2").innerHTML = "How are you?";
}
```

3.5 KEYWORDS IN JAVASCRIPT

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed. Here is a list of some of the keywords you will learn about:

❖ *JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.*

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable
case	
default	
delete	
finally	
in	
instanceof	
new	
this	
throw	
typeof	
void	
with	

4 SYNTAX

- JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
var x, y;          // How to declare variables
x = 5; y = 6;      // How to assign values
z = x + y;         // How to compute values
```

4.1 JAVASCRIPT VALUES

- The JavaScript syntax defines two types of values: Fixed values and variable values. Fixed values are called ***literals***. Variable values are called ***variables***.

4.1.1 JavaScript Literals

- The most important rules for writing fixed values are:
  - Numbers are written with or without decimals: `10.50 , 1001`
  - **Strings** are text, written within double or single quotes: `"John Doe" , 'John Doe'`

4.1.2 JavaScript Variables

- In a programming language, **variables** are used to **store** data values.
- JavaScript uses the **var** keyword to **declare** variables.
- An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
var x;
x = 6;
```

4.2 JAVASCRIPT OPERATORS

- JavaScript uses **arithmetic operators** (+ - * /) to **compute** values.
- JavaScript uses an **assignment operator** (=) to **assign** values to variables.

4.3 JAVASCRIPT EXPRESSIONS

- An expression is a combination of values, variables, and operators, which computes to a value. The computation is called an evaluation. For example, 5 * 10 evaluates to 50
- Expressions can also contain variable values: x * 10
- The values can be of various types, such as numbers and strings. For example, the next statement evaluates to "John Doe":

```
"John" + " " + "Doe"
```

4.4 JAVASCRIPT KEYWORDS

- JavaScript **keywords** are used to identify actions to be performed.

4.5 JAVASCRIPT COMMENTS

- Not all JavaScript statements are "executed". Code after double slashes **//** or between **/*** and ***/** is treated as a **comment**. Comments are ignored, and will not be executed

4.6 JAVASCRIPT IDENTIFIERS

- Identifiers are names. In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels). The rules for legal names are much the same in most programming languages.
- In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($). Subsequent characters may be letters, digits, underscores, or dollar signs.

- *Numbers are not allowed as the first character. This way JavaScript can easily distinguish identifiers from numbers.*
- *All JavaScript identifiers are* ***case sensitive****. The variables* ***lastName*** *and* ***lastname****, are two different variables. JavaScript does not interpret* ***VAR*** *or* ***Var*** *as the keyword* ***var****.*

4.7 JAVASCRIPT AND CAMEL CASE

- Historically, programmers have used different ways of joining multiple words into one variable name:
  - ✓ **Hyphens:** first-name, last-name, master-card, inter-city.

- *Hyphens are not allowed in JavaScript. They are reserved for subtractions.*

  - ✓ **Underscore:** first_name, last_name, master_card, inter_city.
  - ✓ **Upper Camel Case (Pascal Case):** FirstName, LastName, MasterCard, InterCity.
  - ✓ **Lower Camel Case:** JavaScript programmers tend to use camel case that starts with a lowercase letter: firstName, lastName, masterCard, interCity.

4.7.1 JavaScript Character Set

JavaScript uses the **Unicode** character set. Unicode covers (almost) all the characters, punctuations, and symbols in the world.

5 COMMENTS

JavaScript comments can be used to explain JavaScript code, and to make it more readable. JavaScript comments can also be used to prevent execution, when testing alternative code.

5.1 SINGLE LINE COMMENTS

Single line comments start with **//**. Any text between **//** and the end of the line will be ignored by JavaScript (will not be executed).

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

```
var y = x + 2;  // Declare y, give it the value of x + 2
```

5.2 Multi-line Comments

Multi-line comments start with **/*** and end with ***/**. Any text between them will be ignored by JavaScript. This example uses a multi-line comment (a comment block) to explain the code:

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

5.3 Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing. Adding **//** in front of a code line changes the code lines from an executable line to a comment.

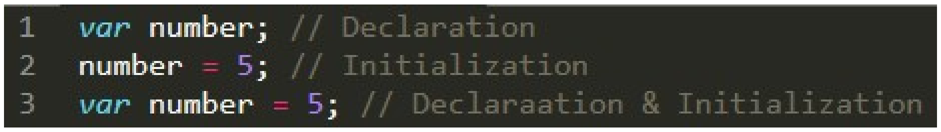
```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

6 Variables

- Variables are used to store data in memory.
  - Ex. It's much easier to write **pi** instead of 3.141592653589793
- Two steps required in order to use a variables. You need to:
  - **Declare** the variable.
  - **Initialize** it, that is, give it a value.

❖ *PRO TIP: Always prefer first declare, then use variables.*



```
var number; // Declaration
number = 5; // Initialization
var number = 5; // Declaraation & Initialization
```

6.1 JAVASCRIPT IDENTIFIERS

All JavaScript **variables** must be **identified** with **unique names**. These unique names are called **identifiers**. Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume). The general rules for constructing names for variables (unique identifiers) are:

- Names **can** contain letters, digits, underscores, and dollar signs.
- Names **must** begin with a letter, $ or _.
- Names are case sensitive (**y** and **Y** are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.
- Variables are un-typed (VALUE IS THE TYPE!).

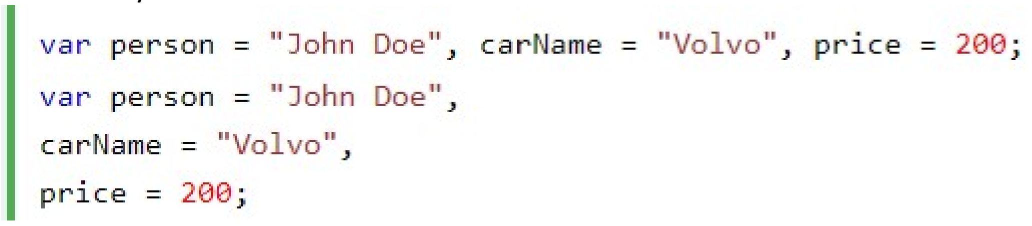
- *JavaScript identifiers are case-sensitive.*
- *Try VALIDATING some names yourself: https://mothereff.in/js-variables*

6.2 THE ASSIGNMENT OPERATOR

In JavaScript, the equal sign (=) is an "**assignment**" operator, not an "equal to" operator. The "**equal to**" operator is written like == in JavaScript.

6.3 ONE STATEMENT, MANY VARIABLES

You can declare many variables in one statement. Start the statement with `var` and separate the variables by **comma**:



```
var person = "John Doe", carName = "Volvo", price = 200;
var person = "John Doe",
carName = "Volvo",
price = 200;
```

6.4 VALUE = UNDEFINED

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

- *A variable declared without a value will have the value **undefined**.*
- *If you re-declare a JavaScript variable, it will not lose its value.*

6.5 JAVASCRIPT ARITHMETIC

- As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:
```
var x = 5 + 2 + 3;    // the result will be 10
```
- You can also add strings, but strings will be concatenated:
```
var x = "John" + " " + "Doe";          // the result will be John Doe
```
- If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.
```
var x = "5" + 2 + 3;  // the result will be 523
var x = 2 + 3 + "5";  // the result will be 55
```

7 DATA TYPES

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

```
var length = 16;                              // Number
var lastName = "Johnson";                     // String
var x = {firstName:"John", lastName:"Doe"};  // Object
```

- *When adding a number and a string, JavaScript will treat the number as a string.*

7.1 JAVASCRIPT TYPES ARE DYNAMIC

- JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

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

7.2 JAVASCRIPT STRINGS

- A string (or a text string) is a series of characters like "John Doe". Strings are written with quotes. You can use single or double quotes.
- You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
var answer = "It's alright";              // Single quote inside double quotes
var answer = "He is called 'Johnny' ";   // Single quotes inside double quotes
var answer = 'He is called "Johnny" ';   // Double quotes inside single quotes
```

7.3 JAVASCRIPT NUMBERS

- JavaScript has only one type of numbers. Numbers can be written with, or without decimals:

```
var x1 = 34.00;        // Written with decimals
var x2 = 34;           // Written without decimals
```

- Extra-large or extra small numbers can be written with scientific (exponential) notation:

```
var y = 123e5;       // 12300000
var z = 123e-5;      // 0.00123
```

7.4 JAVASCRIPT BOOLEANS

- Booleans can only have two values: true or false.
- Booleans are often used in conditional testing.

```
var x = 5;
var y = 5;
var z = 6;
(x == y)        // Returns true
(x == z)        // Returns false
```

7.5 JAVASCRIPT ARRAYS

- JavaScript arrays are written with **square brackets**. Array items are separated by commas.

- The following code declares (creates) an array called cars, containing three items (car names):

```
var cars = ["Saab", "Volvo", "BMW"];
```

❖ *Array indexes are zero-based, which means the first item is [0], second is [1], and so on.*

7.6 JAVASCRIPT OBJECTS

- JavaScript objects are written with **curly braces**.
- Object properties are written as **name:value** pairs, separated by commas.
- The object (person) in the example bellow has 4 properties: firstName, lastName, age, and eyeColor.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

7.7 THE TYPEOF OPERATOR

- You can use the JavaScript **typeof** operator to find the type of a JavaScript variable. The **typeof** operator returns the type of a variable or an expression:

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string"
typeof "John Doe"     // Returns "string"
typeof 0              // Returns "number"
typeof 314            // Returns "number"
typeof 3.14           // Returns "number"
typeof (3)            // Returns "number"
typeof (3 + 4)        // Returns "number"
```

7.8 UNDEFINED

- In JavaScript, a variable without a value, has the value **undefined**. The type is also **undefined**.

```
var car;    // Value is undefined, type is undefined
```

- Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

```
car = undefined;    // Value is undefined, type is undefined
```

7.9 EMPTY VALUES

- An empty value has nothing to do with undefined. An empty string has both a legal value and a type.

```
var car = "";  // The value is "", the typeof is "string"
```

7.10 NULL

- In JavaScript **null is "nothing"**. It is supposed to be something that doesn't exist.
- Unfortunately, in JavaScript, the **data type of null** is an **object**.
- You can consider it a bug in JavaScript that typeof null is an object. It should be null. You can empty an object by setting it to null:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;          // Now value is null, but type is still an object
```

- You can also empty an object by setting it to undefined:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = undefined; // Now both value and type is undefined
```

7.11 DIFFERENCE BETWEEN UNDEFINED AND NULL

- In JavaScript, **UNDEFINED** means a variable has been declared but has not yet been assigned a value, such as:

```
var TestVar;
alert(TestVar); //shows undefined
alert(typeof TestVar); //shows undefined
```

- NULL is an assignment value. It can be assigned to a variable as a representation of no value:

```
var TestVar = null;
alert(TestVar); //shows null
alert(typeof TestVar); //shows object
```

- From the preceding examples, it is clear that undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.
- Undefined and null are equal in value but different in type:

```
typeof undefined          // undefined
typeof null               // object
null === undefined        // false
null == undefined         // true
```

7.12 PRIMITIVE DATA TYPES

- A primitive data value is a single simple data value with no additional properties and methods. The **typeof** operator can return one of these primitive types:
  - Strings.
  - Numbers.
  - Undefined.
  - Boolean.
  - NULL.

7.13 COMPLEX DATA

- The **typeof** operator can return one of two complex types:
  - Function.
  - Object.
- The typeof operator returns **object** for both *objects*, *arrays*, and *null*. It will return **function** for *functions*.

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]  // Returns "object" (not "array", see note below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

❖ *The **typeof** operator returns "object" for arrays because in JavaScript arrays are objects.*

8 OPERATORS OF JAVASCRIPT

8.1 ARITHMETIC OPERATORS:

Arithmetic operators are used to perform arithmetic on numbers.

Operator	Description	Example	Result
+	Addition	5+2	7
-	Subtraction	5-2	3
*	Multiplication	5*2	10
/	Division	5/2	2.5
%	Modulus (Division Remainder)	5%2	1
++	Increment	5++	6
--	Decrement	5--	4

8.1.1 Arithmetic Operations

- A typical arithmetic operation operates on two numbers.

The two numbers can be **literals**: `var x = 100 + 50;`
or **variables**: `var x = a + b;`
or **expressions**: `var x = (100 + 50) * a;`

8.1.2 Operators and Operands

- The numbers (in an arithmetic operation) are called **operands**.
- The operation (to be performed between the two operands) is defined by an **operator**.

Operand	Operator	Operand
100	+	50

8.1.3 Remainder

The **modulus** operator (%) returns the division remainder.

❖ *In arithmetic, the division of two integers produces a* ***quotient*** *and a* ***remainder****. In mathematics, the result of a* ***modulo operation*** *is the* ***remainder*** *of an arithmetic division.*

8.1.4 Operator Precedence Values

Operator precedence describes the order in which operations are performed in an arithmetic expression.

Value	Operator	Description	Example
20	( )	Expression grouping	(3 + 4)
19	.	Member	person.name
19	[]	Member	person["name"]
19	()	Function call	myFunction()

Value	Operator	Description	Example
19	new	Create	new Date()
17	++	Postfix Increment	i++
17	--	Postfix Decrement	i--
16	++	Prefix Increment	++i
16	--	Prefix Decrement	--i
16	!	Logical not	!(x==y)
16	typeof	Type	typeof x
15	**	Exponentiation (ES7)	10 ** 2
14	*	Multiplication	10 * 5
14	/	Division	10 / 5
14	%	Division Remainder	10 % 5
13	+	Addition	10 + 5
13	-	Subtraction	10 - 5
12	<<	Shift left	x << 2
12	>>	Shift right	x >> 2
12	>>>	Shift right (unsigned)	x >>> 2
11	<	Less than	x < y
11	<=	Less than or equal	x <= y
11	>	Greater than	x > y
11	>=	Greater than or equal	x >= y
11	in	Property in Object	"PI" in Math
11	instanceof	Instance of Object	instanceof Array
10	==	Equal	x == y
10	===	Strict equal	x === y
10	!=	Unequal	x != y
10	!==	Strict unequal	x !== y
9	&	Bitwise AND	x & y
8	^	Bitwise XOR	x ^ y
7	|	Bitwise OR	x | y
6	&&	Logical AND	x && y
5	||	Logical OR	x || y
4	? :	Condition	? "Yes" : "No"
3	+=	Assignment	x += y
3	+=	Assignment	x += y
3	-=	Assignment	x -= y
3	*=	Assignment	x *= y
3	%=	Assignment	x %= y
3	<<=	Assignment	x <<= y
3	>>=	Assignment	x >>= y
3	>>>=	Assignment	x >>>= y
3	&=	Assignment	x &= y
3	^=	Assignment	x ^= y

Value	Operator	Description	Example
3	|=	Assignment	x |= y
2	yield	Pause Function	yield x
1	,	Comma	5 , 6

❖ *Expressions in parentheses are fully computed before the value is used in the rest of the expression.*

8.2 ASSIGNMENT OPERATORS:

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

❖ *The **= operator is an experimental part of the ECMAScript 2016 proposal (ES7). It is not stable across browsers. Do not use it.*

8.3 JAVASCRIPT STRING OPERATORS

The + operator can also be used to add (concatenate) strings.

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be: John Doe

The += assignment operator can also be used to add (concatenate) strings:

```
var txt1 = "What a very ";
txt1 += "nice day";
```

The result of txt1 will be: `What a very nice day`

❖ *When used on strings, the + operator is called the **concatenation** operator.*
❖ *If you add a number and a string, the result will be a string!*

8.4 COMPARISON OPERATORS:

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

8.5 LOGICAL OPERATORS:

Operator	Description
&&	logical and
||	logical or
!	logical not

8.6 TYPE OPERATORS:

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

8.7 JAVASCRIPT BITWISE OPERATORS

Bit operators work on 32 bits numbers. Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers. Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.

~00000000000000000000000000000101 will return 11111111111111111111111111111010