SUB: Design Pattern And Framework

TOPIC: JAVASCRIPT

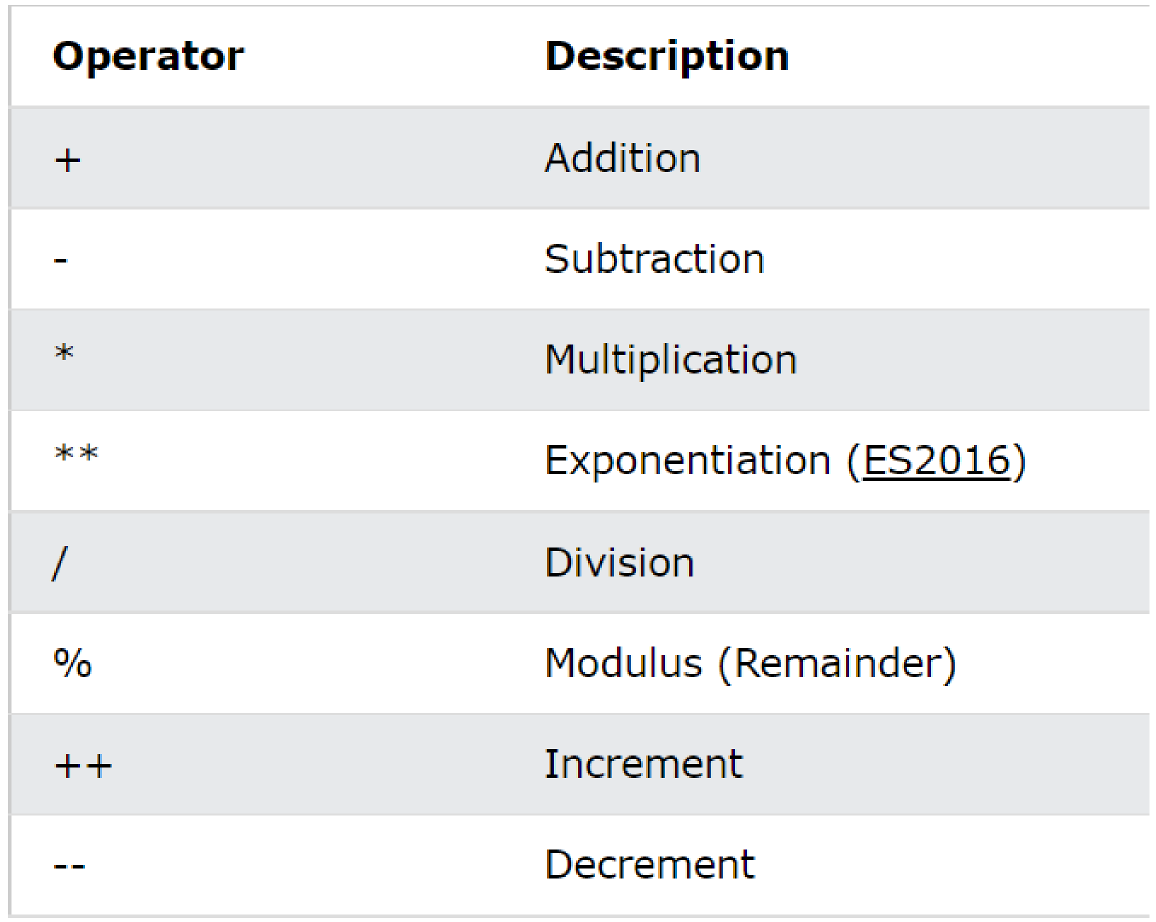
Java Script Operators

What is an Operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators
- 'typeof' Operator

Arithmetic Operator



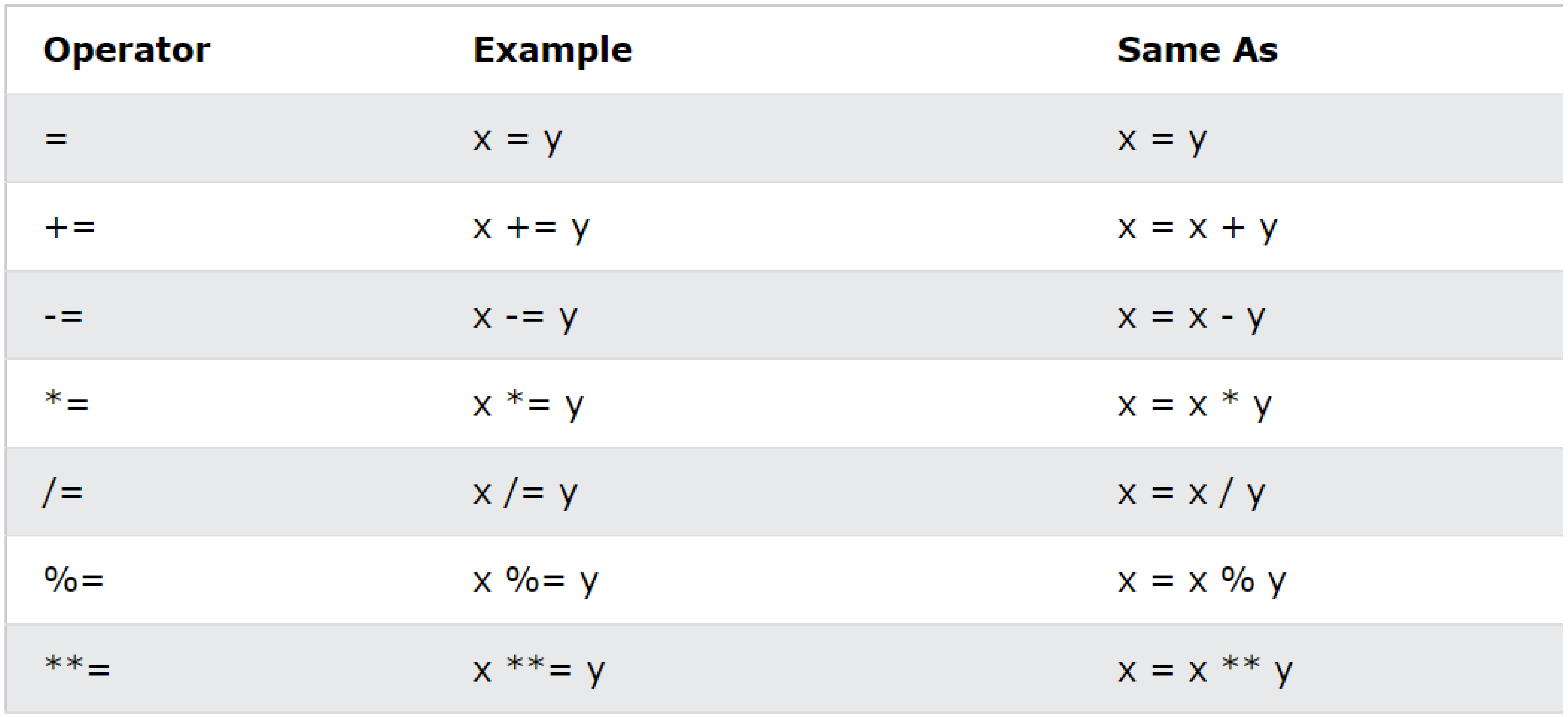
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Java Script Operator

The following code shows how to use arithmetic operators in JavaScript.

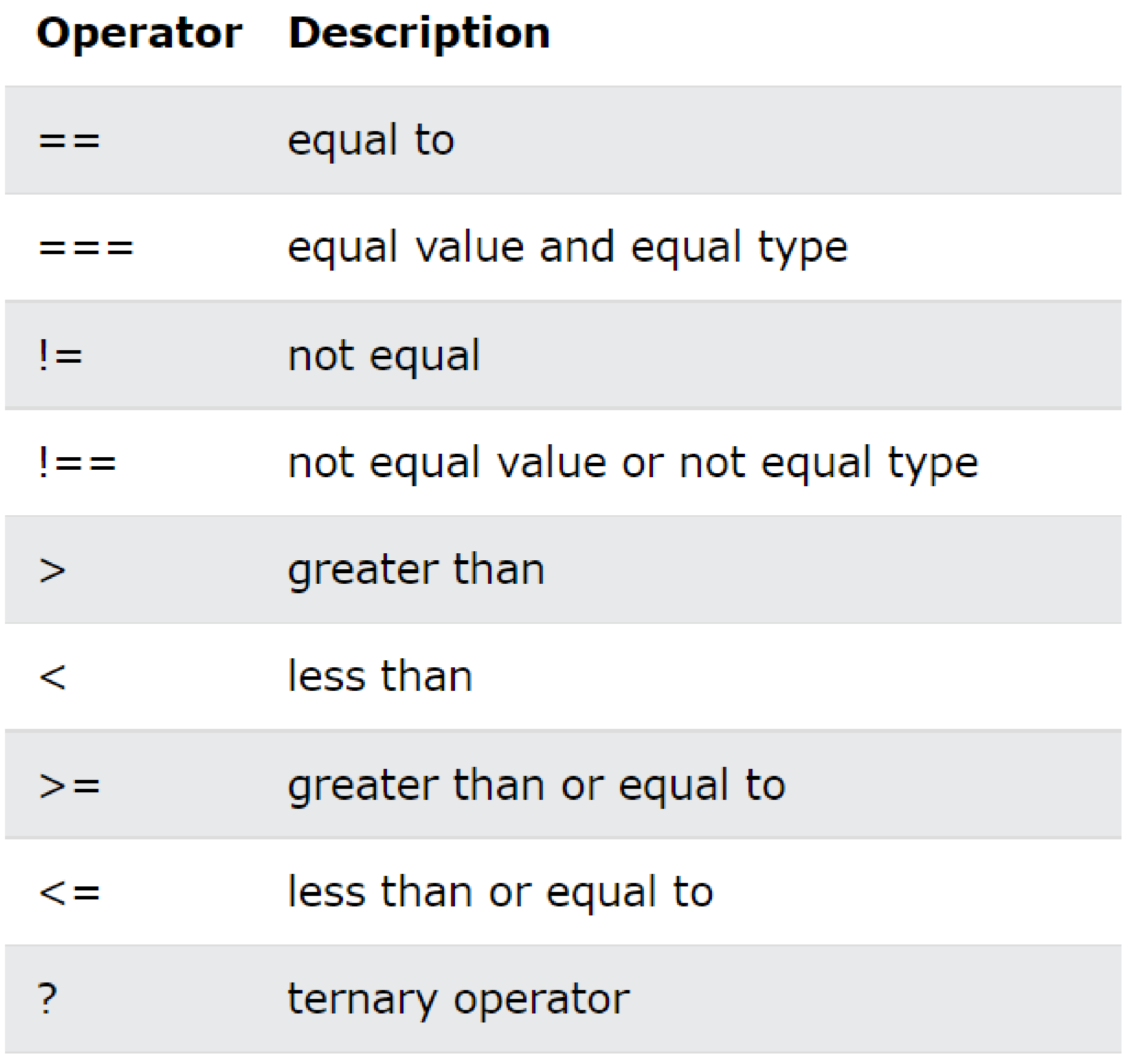
```
<html>
<body>

<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";

document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);
```

Assignment Operator



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

Comparison Operator



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

Java Script Operator

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";

document.write ("((a > b) ? 100 : 200) => ");
result = (a > b) ? 100 : 200;
document.write(result);
document.write(linebreak);
```

Java Script Operator

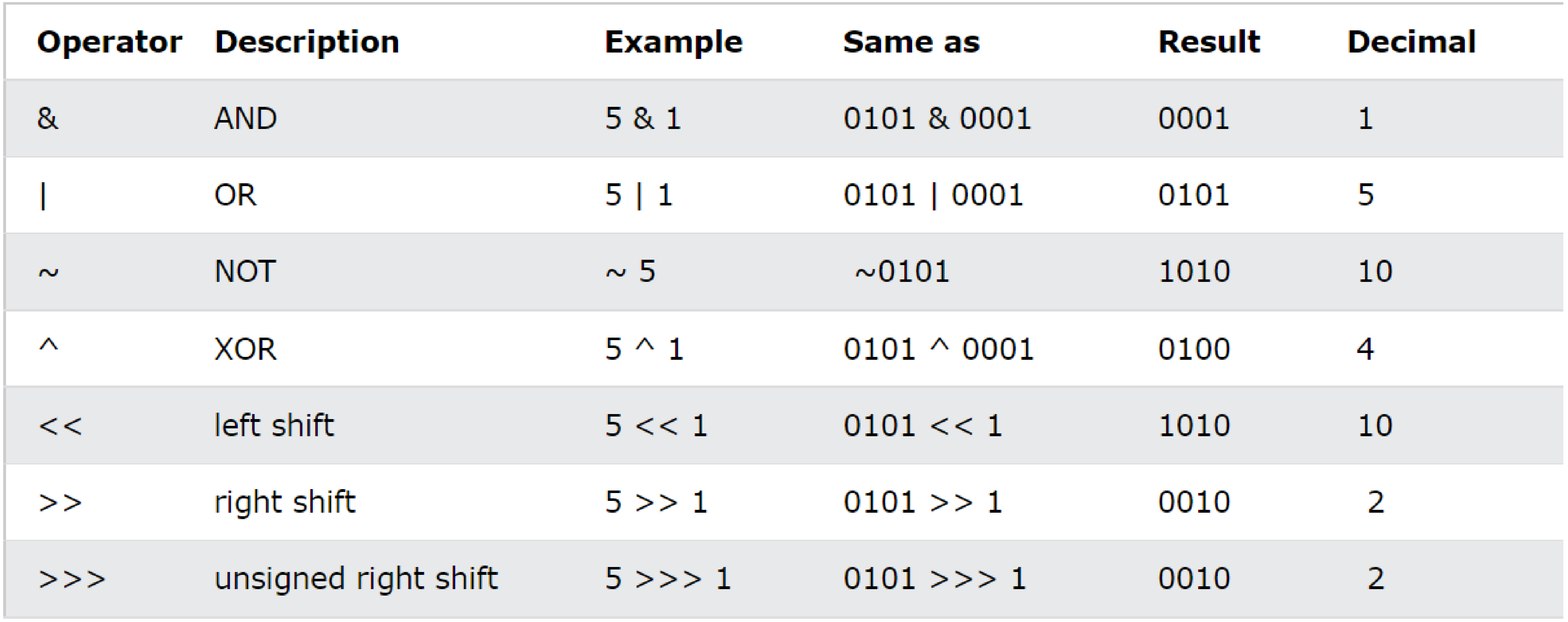
typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

```
<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
```

Bitwise Operator



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
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

Java Script Function

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Java Script Function

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
  statements
}
//-->
</script>
```

Java Script Function

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
  document.write ("Hello there!");
}
</script>
</head>
 <body>
 <p>Click the following button to call the function</p>
 <form>
 <input type="button" onclick="sayHello()" value="Say Hello">
 </form>
 <p>Use different text in write method and then try...</p>
 </body>
```

Java Script Function Parameters

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
   document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>
</body>
</html>
```

Java Script Function Return Statement

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Java Script Function – Return Statement

```
<html>

<head>

<script type="text/javascript">

function concatenate(first, last)

{

   var full;
   full = first + last;

   return  full;

}
function secondFunction()

{

var result;

result = concatenate('Zara', 'Ali');

document.write (result );

}  </script> </head>
```

Java Script Function Return Statement

```
<body>

<p>Click the following button to call the function</p>

<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>


<p>Use different parameters inside the function and then try...</p>

</body>

</html>
```

Java Script Function – Nested Function

```
<html>
<head>

<script type="text/javascript">

<!--

function hypotenuse(a, b) {

   function square(x) { return x*x; }
   return Math.sqrt(square(a) + square(b));

}
function secondFunction(){

   var result;

   result = hypotenuse(1,2);

   document.write ( result );
}

//-->

</script>
```

Java Script Function – Nested Function

```
<body>
<p>Click the following button to call the function</p>


<form>
<input type="button" onclick="secondFunction()" value="Call Function
</form>


<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Java Script Function - Constructor

The *function* statement is not the only way to define a new function; you can define your function dynamically using **Function()** constructor along with the **new** operator.

```
<script type="text/javascript">

<!--

var variablename = new Function(Arg1, Arg2..., "Function Body");

//-->

</script>
```

The **Function()** constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the **Function()** constructor is not passed any argument that specifies a name for the function it creates. The **unnamed** functions created with the **Function()** constructor are called **anonymous** functions.

Java Script Function - Constructor

```
<html>

<head>

<script type="text/javascript">

<!--

var func = new Function("x", "y", "return x*y;");


function secondFunction(){

   var result;

   result = func(10,20);
   document.write ( result );

}

//-->

</script>
```

Java Script Function - Constructor

```
<body>
<p>Click the following button to call the function</p>


<form>
<input type="button" onclick="secondFunction()" value="Call Function
</form>


<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Java Script Function – Literal Function

Function Literals

JavaScript 1.2 introduces the concept of **function literals** which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

```
<script type="text/javascript">
<!--
var variablename = function FunctionName(Argument List){
                      Function Body
       };
//-->
</script>
```

Java Script Function –Literal Function

```
<html>
<head>
<script type="text/javascript">
<!--
var func = function(x,y){ return x*y };
function secondFunction(){
   var result;
   result = func(10,20);
   document.write ( result );
}
//-->
</script>
</head>
```

Java Script Function – Literal Function

```
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```