Why Study JavaScript?

- JavaScript is one of the **3 languages** all web developers **must** learn:
- 1. **HTML** to define the content of web pages
- 2. **CSS** to specify the layout of web pages
- 3. **JavaScript** to program the behaviour of web pages

The <script> Tag

- In HTML, JavaScript code is inserted between <script> and </script> tags.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

<script> Tag: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

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

</body>
</html>
```

JavaScript in <head>

- A JavaScript function is placed in the <head> section of an HTML page.
- The function is invoked (called) when a button is clicked.

JavaScript in <head>: Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

- A JavaScript function is placed in the <body> section of an HTML page.
- The function is invoked (called) when a button is clicked.
- Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

JavaScript in <body>

```
<!DOCTYPE html>
<html>
<body>

<h2>Demo JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

External JavaScript

- Scripts can also be placed in external files.
- External file: myScript.js
- 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:

<script src="myScript.js"></script>

- 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.
- External scripts cannot contain <script> tags.

External References

- An external script can be referenced in 3 different ways:

- With a full URL (a full web address)

- With a file path (like /js/)

- Without any path

full URL: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Click Me</button>

<p>This example uses a full web URL to link to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

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

</body>
</html>
```

file path: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>This example uses a file path to link to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

<script src="/js/myScript.js"></script>

</body>
</html>
```

Without path: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Demo External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>This example links to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element,
  using innerHTML.
- Writing into the HTML output
  using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console,
  using console.log().

Using innerHTML

- To access an HTML element, JavaScript can use
  the document.getElementById(id) method.
- The id attribute defines the HTML element. The innerHTML property defines the HTML content.

Using innerHTML: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My First Paragraph.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Using document.write()

- For testing purposes, it is convenient to use document.write().

Using document.write(): Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<p>Never call document.write after the document has finished loading.
It will overwrite the whole document.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Using document.write() continue...

- Using document.write() after an HTML document is loaded, will **delete all existing HTML.**
- The document.write() method should only be used for testing.

Continue....

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try
    it</button>

</body>
</html>
```

Using window.alert()

- You can use an alert box to display data.

Using window.alert(): Example

```
<!DOCTYPE html>
<html>
<body>

<h2>My First Web Page</h2>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

skip the window keyword

- You can skip the window keyword.
- In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional.
- **It represents the browser's window**. All global JavaScript objects, functions, and variables automatically become members of the window object.

skip the window keyword: Example

```
<!DOCTYPE html>
  <html>
  <body>

  <h1>My First Web Page</h1>
  <p>My first paragraph.</p>

  <script>
  alert(5 + 6);
  </script>

  </body>
  </html>
```

Using console.log()

- For debugging purposes, you can call the console.log() method in the browser to display data.

Using console.log(): Example

```
<!DOCTYPE html>
<html>
 <body>
  <script>
   console.log(5+6);
  </script>
 </body>
</html>
```

JavaScript Print

- JavaScript does not have any print object or print methods.
- You cannot access output devices from JavaScript.
- The only exception is that you can call the window.print() method in the browser to print the content of the current window.

JavaScript Print: Example

```
<!DOCTYPE html>
<html>
 <body>
  <button onclick="window.print()"> Print this
  page</button>
 </body>
 </html>
```

JavaScript program

- A **JavaScript program** is a list of programming **statements**.
- In HTML, JavaScript programs are executed by the web browser.

JavaScript Statements

- JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.
- The statements are executed, one by one, in the same order as they are written.
- Semicolons separate JavaScript statements.
- Add a semicolon at the end of each executable statement.
- When separated by semicolons, multiple statements on one line are allowed.

JavaScript Statements: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello Dolly.";
</script>

</body>
</html>
```

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:

```
let person = "Hege";
let person="Hege";
```

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

```
let x = y + z;
```

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.

JavaScript Line Length and Line Breaks: Example

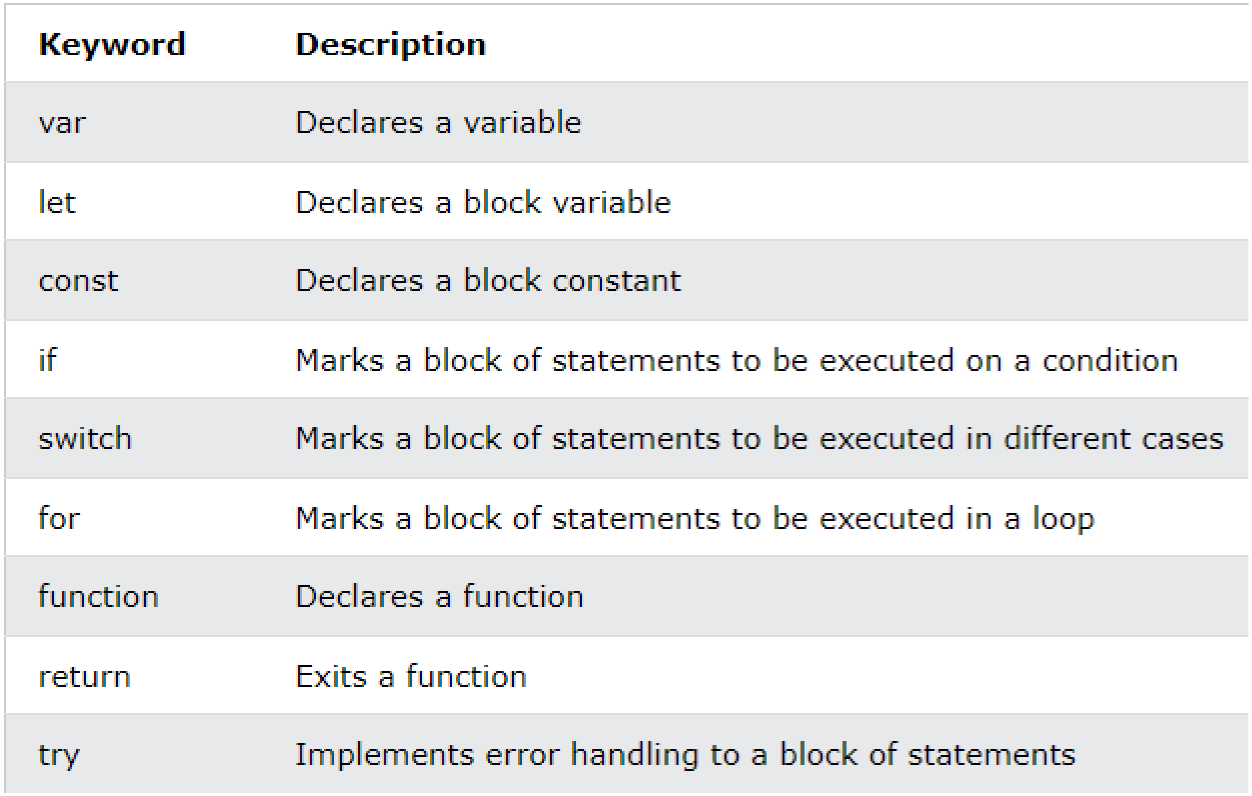
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>
The best place to break a code line is after an operator or a comma.
</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Hello Dolly!";
</script>

</body>
</html>
```

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.
- Example Functions

JavaScript Keywords

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



Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values:- Fixed values are called **Literals.**
- Variable values:- Variable values are called **Variables**.

JavaScript Literals

- **Numbers** are written with or without decimals.

10.50

1001

- **Strings** are text, written within double or single quotes:

"John Doe“

'John Doe‘

variables

- **variables** are used to **store** data values. JavaScript uses the keywords var, let and const to **declare** variables. An **equal sign** is used to **assign values** to variables.

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

```
let x;
x = 6;
```

JavaScript Variables

4 Ways to Declare a JavaScript Variable:

- Using var
- Using let
- Using const
- Using nothing

declared with the var keyword

- With var you can:
- Example:-

  var x = "John Doe";

  var x = 0;
- Variables defined with let have Block Scope.

declared with the var keyword:Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are variables.</p>

<p id="demo"></p>

<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

declared with the let keyword

- Variables defined with let cannot be **redeclared**.
- You cannot accidentally redeclare a variable.
- With let you can not do this:
- let x = "John Doe";

  let x = 0;

  // SyntaxError: 'x' has already been declared

declared with the let keyword: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>In this example, x, y, and z are variables.</p>

<p id="demo"></p>

<script>
let x = 5;
let y = 6;
let z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

undeclared variables: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are undeclared variables.</p>

<p id="demo"></p>

<script>
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>

</body>
</html>
```

Declaring variables: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>Create a variable, assign a value to it, and display it:</p>

<p id="demo"></p>

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

One Statement, Many Variables

- You can declare many variables in one statement.
- Start the statement with let and separate the variables by **comma.**
- let person = "John Doe", carName = "Volvo", price = 200;

OR

- let person = "John Doe",
  carName = "Volvo",
  price = 200;

One Statement, Many Variables: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Variables</h1>

<p>You can declare many variables in one statement.</p>

<p id="demo"></p>

<script>
let person = "John Doe", carName = "Volvo", price = 200;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```

Re-Declaring JavaScript Variables

- If you re-declare a JavaScript variable declared with var, it will not lose its value.
- The variable carName will still have the value "Volvo" after the execution of these statements:
- Example:-

  ```
  var carName = "Volvo";
  var carName;
  ```

- You cannot re-declare a variable declared with let or const.

  This will not work:

  ```
  let carName = "Volvo";
  let carName;
  ```

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.

JavaScript Keywords

- JavaScript **keywords** are used to identify actions to be performed.
- The let/var keyword tells the browser to create variables.

JavaScript Comments

- Not all JavaScript statements are "executed".
- Code after double slashes // or
  between /* and */ is treated as a **comment**.

JavaScript Identifiers / Names

- Identifiers are JavaScript names.
- Identifiers are used to name variables and keywords, and functions.
- The rules for legal names are the same in most programming languages.
- A JavaScript name must begin with:

  -A letter (A-Z or a-z)

  -A dollar sign ($)

  -Or an underscore (_)
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- Numbers are not allowed as the first character in names.

Case Sensitive

- JavaScript is Case Sensitive
- All JavaScript identifiers are **case sensitive**.
- The variables lastName and lastname, are two different variables:-

```
let lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

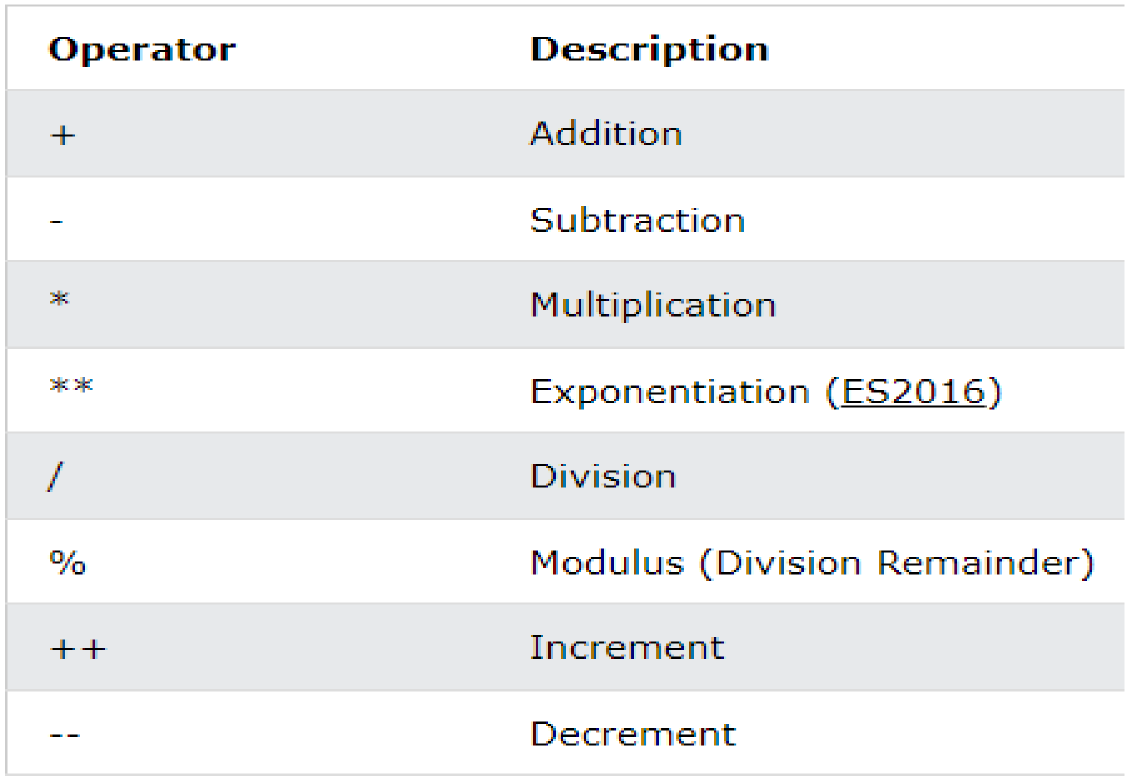
Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

Arithmetic Operators

- **Arithmetic Operators** are used to perform arithmetic on numbers.



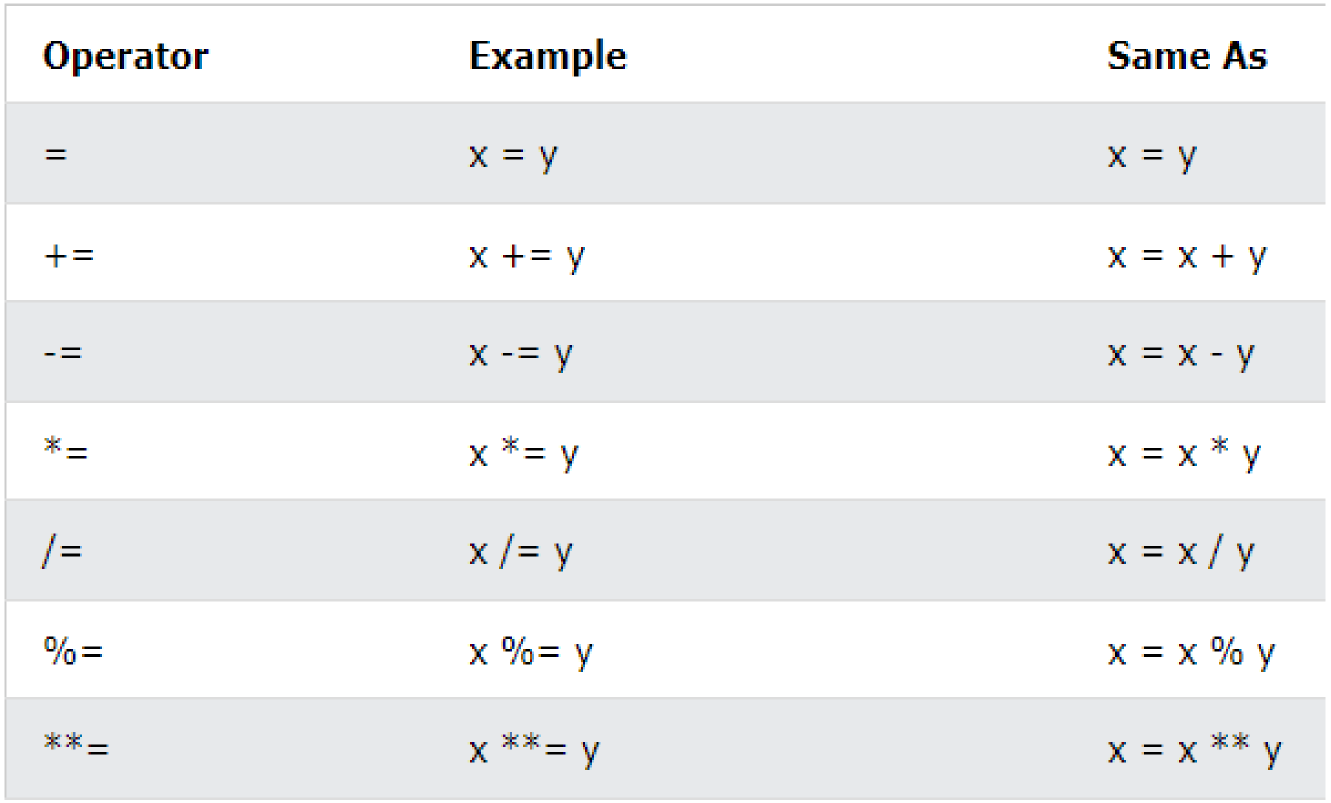
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Arithmetic Operators: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Arithmetic</h1>
<h2>Arithmetic Operations</h2>
<p>A typical arithmetic operation takes two numbers (or expressions) and produces a new number.</p>

<p id="demo"></p>

<script>
let a = 3;
let x = (100 + 50) * a;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

JavaScript Assignment Operators

- Assignment operators assign values to JavaScript variables.
- The **Addition Assignment Operator** (+=) adds a value to a variable.



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

Assignment Operators: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Arithmetic</h1>
<h2>The += Operator</h2>

<p id="demo"></p>

<script>
var x = 10;
x += 5;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Adding JavaScript Strings

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

JavaScript Strings: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Operators</h1>
<h2>The + Operator</h2>

<p>The + operator concatenates (adds) strings.</p>

<p id="demo"></p>

<script>
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
document.getElementById("demo").innerHTML = text3;
</script>

</body>
</html>
```

Adding JavaScript Strings

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

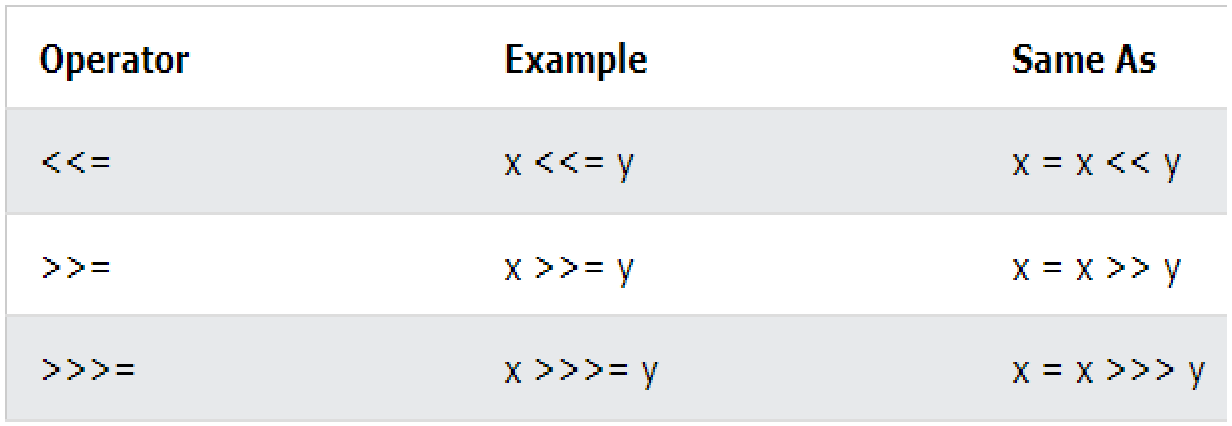
Adding JavaScript Strings: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Operators</h1>

<p>The assignment operator += can concatenate strings.</p>

<p id="demo"></p>

<script>
let text1 = "What a very ";
text1 += "nice day";
document.getElementById("demo").innerHTML = text1;
</script>

</body>
</html>
```

Adding Strings and Numbers

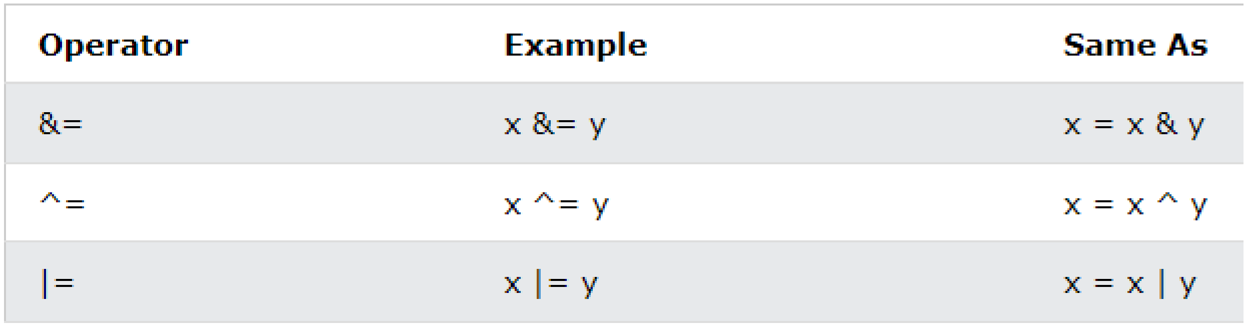
```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Operators</h1>

<p>Adding a number and a string, returns a string.</p>

<p id="demo"></p>

<script>
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
document.getElementById("demo").innerHTML =
x + "<br>" + y + "<br>" + z;
</script>

</body>
</html>
```

Shift Assignment Operators



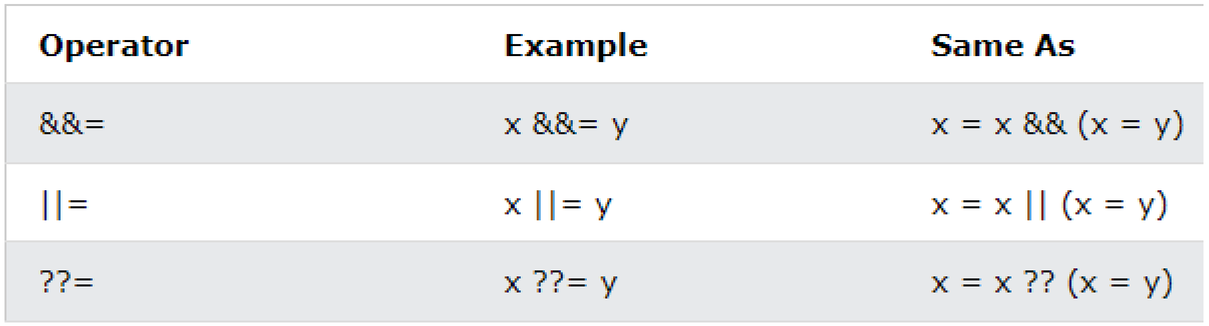
Operator	Example	Same As
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

Bitwise Assignment Operators



Operator	Example	Same As
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

Logical Assignment Operators

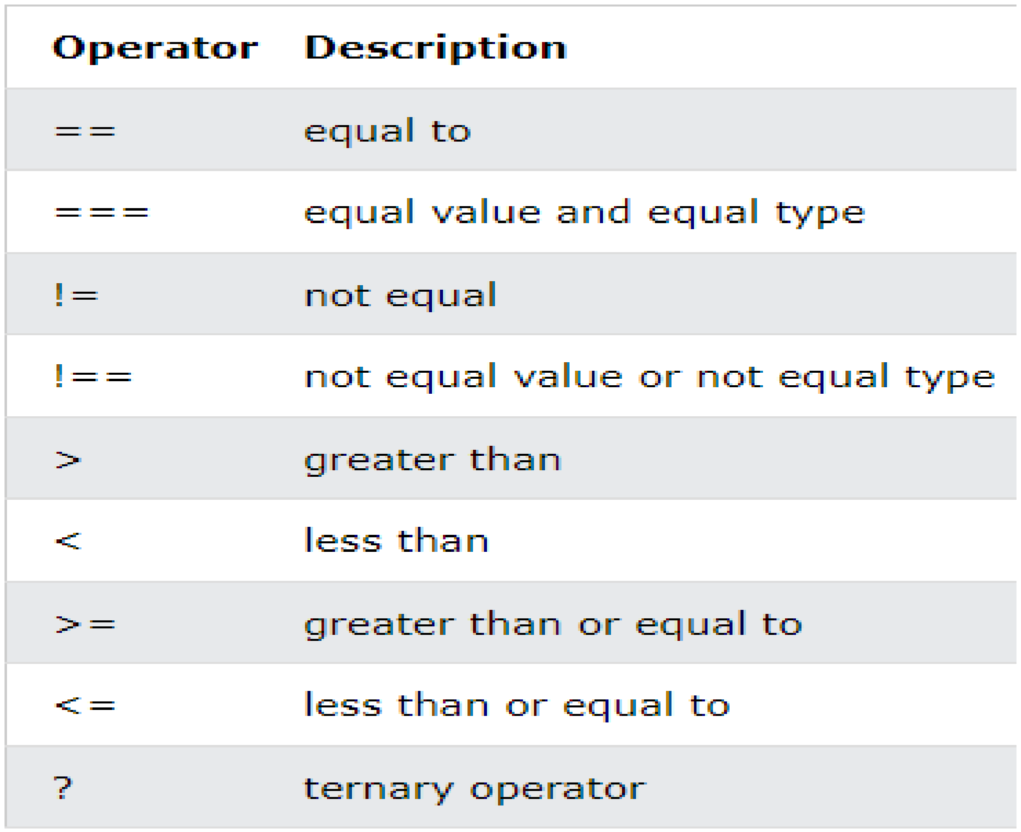


Operator	Example	Same As
&&=	x &&= y	x = x && (x = y)
||=	x ||= y	x = x || (x = y)
??=	x ??= y	x = x ?? (x = y)

The = Operator

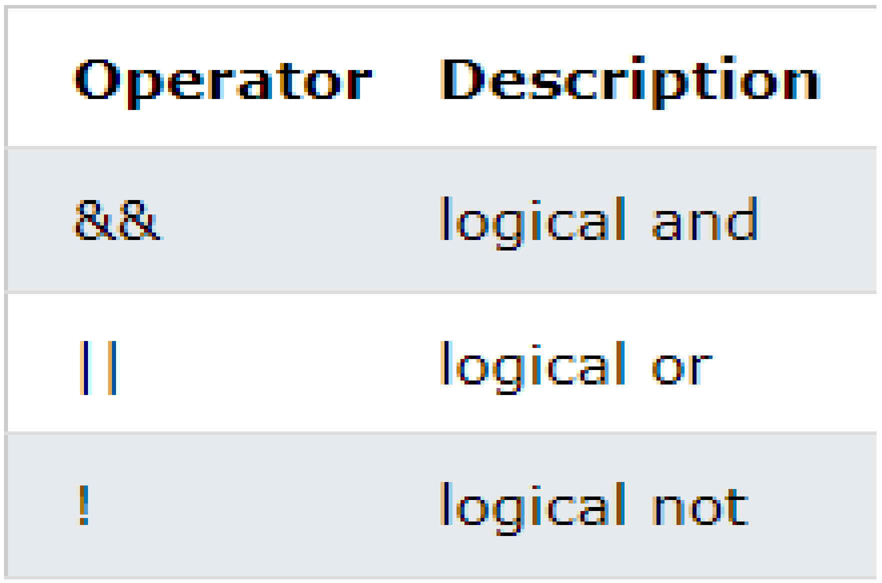
- The **Simple Assignment Operator** assigns a value to a variable.

JavaScript Comparison Operators



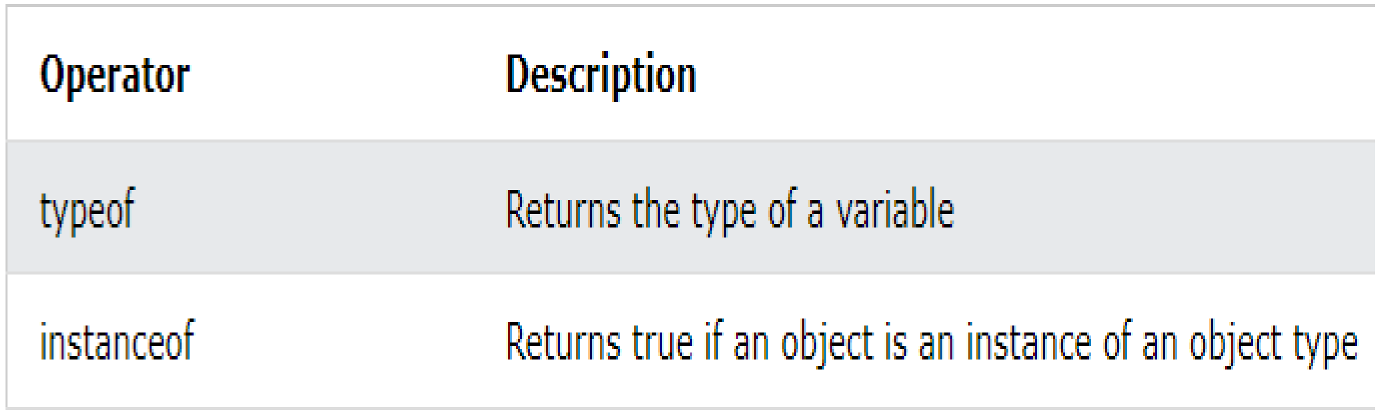
Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators



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

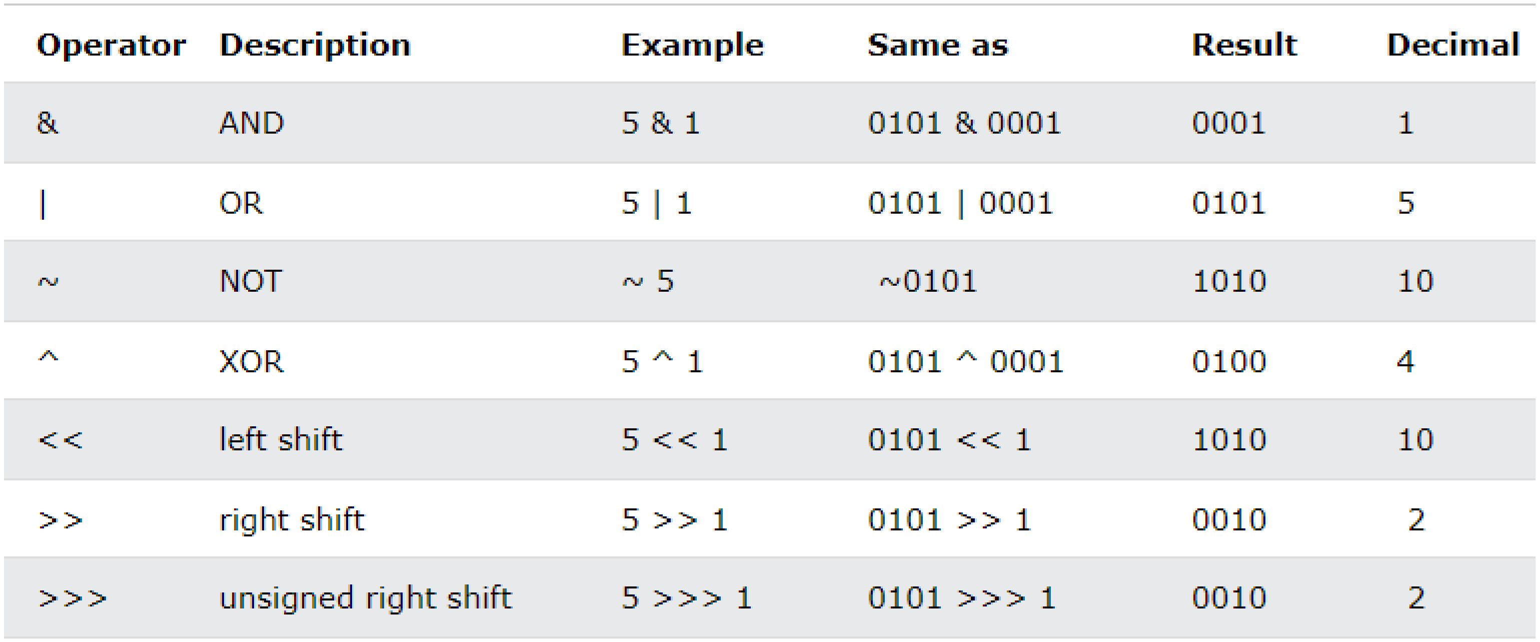
JavaScript Type Operators



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

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
<<	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

Operator Precedence

- Operator precedence describes the order in which operations are performed in an arithmetic expression.
- Multiplication (*) and division (/) have higher **precedence** than addition (+) and subtraction (-).
- The precedence can be changed by using parentheses.
- When many operations have the same precedence (like addition and subtraction), they are computed from left to right.

Data Types

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

Example

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

- 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:

Example

```
let carName1 = "Volvo XC60";   // Using double quotes
let carName2 = 'Volvo XC60';   // Using single quotes
```

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

Example

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

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

Example

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

- Booleans can only have two values: true or false.

Example

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

- 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):

Example:

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

- JavaScript objects are written with curly braces {}. Object properties are written as name:value pairs, separated by commas.

Example

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

- The typeof operator returns the type of a variable or an expression:
- Example:

```
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"
  ```

Undefined

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

Example

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

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

Example

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

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

Example

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