JavaScript

* JavaScript is the world's most popular programming language.
* JavaScript is the programming language of the Web.
* JavaScript is easy to learn.

JavaScript Introduction:

## **JavaScript Can Change HTML Content**

* One of many JavaScript HTML methods is getElementById().
* The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":
* Example:

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>

</body>

</html>

* JavaScript accepts both double and single quotes

## **JavaScript Can Change HTML Attribute Values**

* **Example**:

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic\_bulbon.gif'">Turn on the light

</button>

<img id="myImage" src="pic\_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic\_bulboff.gif'">Turn off the light</button>

</body>

</html>

## **JavaScript Can Change HTML Styles (CSS)**

* Changing the style of an HTML element, is a variant of changing an HTML attribute:
* Syntax:

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>

</body>

</html>

* **JavaScript Where To:**

## **The <script> Tag**

* In HTML, JavaScript code is inserted between  <script>  and  </script> tags.
* Syntax:

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

## **JavaScript Functions and Events**

* A JavaScript function is a block of JavaScript code, that can be executed when "called" for.
* For example, a function can be called when an **event** occurs, like when the user clicks a button.

## **JavaScript in <head> or <body>**

* 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.

## **JavaScript in <head>**

* In this example, a JavaScript function is placed in the <head> section of an HTML page.
* The function is invoked (called) when a button is clicked
* Syntax:

<!DOCTYPE html>  
<html>

<head>  
<script>  
function myFunction()

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

<h1>A Web Page</h1>  
<p id="demo">A Paragraph</p>  
<button type="button" onclick="myFunction()">Try it</button>

</body>  
</html>

## **JavaScript in <body>**

* In this example, a JavaScript function is placed in the <body> section of an HTML page.
* The function is invoked (called) when a button is clicked:
* Syntax:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>A Web Page</h1>  
<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
* Syntax:

function myFunction()

{  
  document.getElementById("demo").innerHTML = "Paragraph changed.";  
}

* 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:
* Syntax:

<!DOCTYPE html>

<html>

<body>

<h2>External JavaScript</h2>

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

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

<p>(myFunction is stored in an external file called "myScript.js")</p>

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

</body>

</html>

# JavaScript Output

## **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:
* Syntax:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<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():
* Syntax:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
document.write(5 + 6);  
</script>  
  
</body>  
</html>

* Never call document.write after the document has finished loading. It will overwrite the whole document.
* Syntax:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<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:
* Syntax:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
window.alert(5 + 6);  
</script>  
  
</body>  
</html>

* You can skip the window keyword.
* In JavaScript, the window object is the global scope object, that means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

## **Using console.log()**

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

<!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.
* Syntax:

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

# JavaScript Statements

## **JavaScript Programs**

* 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**.

## **JavaScript Statements**

* JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.
* This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":
* Example:

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

## **Semicolons ;**

* 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:
* Example:

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

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

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

## **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:
* Example:

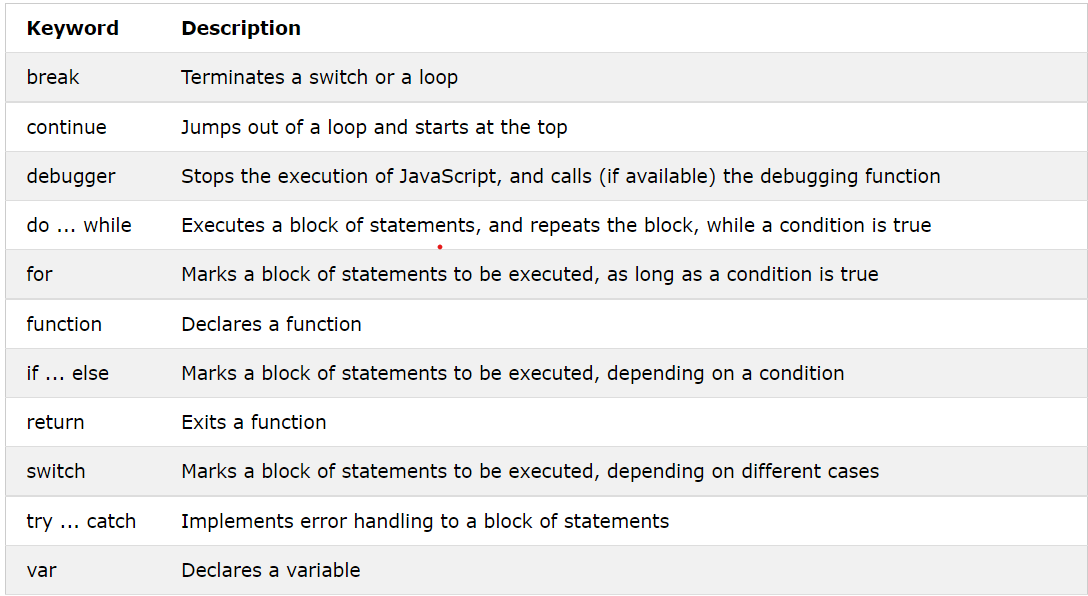
document.getElementById("demo").innerHTML =  
"Hello Dolly!";

## **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:
* Example:

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

## **JavaScript Keywords**

* JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

# JavaScript Syntax

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

## **JavaScript Values**

* The JavaScript syntax defines two types of values:
* Fixed values
* Variable values
* Fixed values are called **Literals**.
* Variable values are called **Variables**.

## **JavaScript Literals**

* The two most important syntax rules for fixed values are:

1. **Numbers** are written with or without decimals:

# 10.50 1001

# 2. **Strings** are text, written within double or single quotes:

# "John Doe" 'John Doe'

## **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:
* Example:

var x;  
  
x = 6;

## **JavaScript Operators**

* JavaScript uses **arithmetic operators** ( + - \* / ) to **compute** values:

# (5 + 6) \* 10

# JavaScript uses an **assignment operator** ( = ) to **assign** values to variables:

# var x, y; x = 5; y = 6;

## **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:

# var x = 5;   // I will be executed // var x = 6;   I will NOT be executed

## **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.

## **JavaScript is Case Sensitive**

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

# var lastname, lastName; lastName = "Doe"; lastname = "Peterson";

## **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.

**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.

# JavaScript Comments

## **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).
* This example uses a single-line comment before each code line:
* Example:

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

## **Multi-line Comments**

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

/\*  
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.";

## **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.
* This example uses // to prevent execution of one of the code lines:
* Example:

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

# This example uses a comment block to prevent execution of multiple lines:

* Example:

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

# JavaScript Variables

* JavaScript variables are containers for storing data values.
* In this example, x, y, and z, are variables, declared with the var keyword:
* Example:

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

## **Using**let**and**const**(ES6)**

* Before 2015, using the var keyword was the only way to declare a JavaScript variable.
* The 2015 version of [JavaScript (ES6)](https://www.w3schools.com/js/js_es6.asp) allows the use of the const keyword to define a variable that cannot be reassigned, and the let keyword to define a variable with restricted scope.
* Because it is a little complicated to describe the difference between these keywords, and because they are not supported in older browsers, the first part of this tutorial will most often use var.

## **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
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case sensitive (y and Y are different variables)
* Reserved words (like JavaScript keywords) cannot be used as names

## **The Assignment Operator**

* In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.
* This is different from algebra. The following does not make sense in algebra:

# x = x + 5

* In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.
* (It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

## **JavaScript Data Types**

* JavaScript variables can hold numbers like 100 and text values like "John Doe".
* In programming, text values are called text strings.
* JavaScript can handle many types of data, but for now, just think of numbers and strings.
* Strings are written inside double or single quotes. Numbers are written without quotes.
* If you put a number in quotes, it will be treated as a text string.
* Example:

var pi = 3.14;  
var person = "John Doe";  
var answer = 'Yes I am!';

## **Declaring (Creating) JavaScript Variables**

* Creating a variable in JavaScript is called "declaring" a variable.
* You declare a JavaScript variable with the var keyword:

# var carName;

* After the declaration, the variable has no value (technically it has the value of undefined).
* To **assign** a value to the variable, use the equal sign:

# carName = "Volvo";

# You can also assign a value to the variable when you declare it:

# var carName = "Volvo";

* In the example below, we create a variable called carName and assign the value "Volvo" to it.
* Then we "output" the value inside an HTML paragraph with id="demo":
* In the example below, we create a variable called carName and assign the value "Volvo" to it.
* Then we "output" the value inside an HTML paragraph with id="demo":
* Example:

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

## **One Statement, Many Variables**

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

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

# A declaration can span multiple lines:

* Example:

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

## **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.
* The variable carName will have the value undefined after the execution of this statement:

## **Re-Declaring JavaScript Variables**

* If you re-declare a JavaScript variable, 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;

# If you re-declare a JavaScript variable, it will not lose its value.

## **JavaScript Dollar Sign $**

* Remember that JavaScript identifiers (names) must begin with:
* A letter (A-Z or a-z)
* A dollar sign ($)
* Or an underscore (\_)
* Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:
* Example:

var $$$ = "Hello World";  
var $ = 2;  
var $myMoney = 5;

## **JavaScript Underscore (\_)**

Since JavaScript treats underscore as a letter, identifiers containing \_ are valid variable names:

* Example:

var \_lastName = "Johnson";  
var \_x = 2;  
var \_100 = 5;

# JavaScript Let

* ES2015 introduced two important new JavaScript keywords: let and const.
* These two keywords provide **Block Scope** variables (and constants) in JavaScript.
* Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.

## **Global Scope**

* Variables declared **Globally** (outside any function) have **Global Scope**.
* Example:

var carName = "Volvo";  
  
// code here can use carName  
  
function myFunction() {  
  // code here can also use carName  
}

## **Function Scope**

* Variables declared **Locally** (inside a function) have **Function Scope**.
* Example:

// code here can NOT use carName  
  
function myFunction() {  
  var carName = "Volvo";  
  // code here CAN use carName  
}  
  
// code here can NOT use carName

## **JavaScript Block Scope**

* Variables declared with the var keyword cannot have **Block Scope**.
* Variables declared inside a block **{}** can be accessed from outside the block.
* Example:

{  
  var x = 2;  
}  
// x CAN be used here

* Before ES2015 JavaScript did not have **Block Scope**.
* Variables declared with the let keyword can have Block Scope.
* Variables declared inside a block **{}** cannot be accessed from outside the block:
* Example:

{  
  let x = 2;  
}  
// x can NOT be used here

## **Redeclaring Variables**

* Redeclaring a variable using the var keyword can impose problems.
* Redeclaring a variable inside a block will also redeclare the variable outside the block:
* Example:

var x = 10;  
// Here x is 10  
{  
  var x = 2;  
  // Here x is 2  
}  
// Here x is 2

* Redeclaring a variable using the let keyword can solve this problem.
* Redeclaring a variable inside a block will not redeclare the variable outside the block:
* Example:

var x = 10;  
// Here x is 10  
{  
  let x = 2;  
  // Here x is 2  
}  
// Here x is 10

## **Loop Scope**

* Using var in a loop:
* Example:

var i = 5;  
for (var i = 0; i < 10; i++) {  
  // some statements  
}  
// Here i is 10

* Using let in a loop:
* Example:

let i = 5;  
for (let i = 0; i < 10; i++) {  
  // some statements  
}  
// Here i is 5

* In the first example, using var, the variable declared in the loop redeclares the variable outside the loop.
* In the second example, using let, the variable declared in the loop does not redeclare the variable outside the loop.
* When let is used to declare the i variable in a loop, the i variable will only be visible within the loop.

## **Function Scope**

* Variables declared with var and let are quite similar when declared inside a function.
* They will both have **Function Scope**:
* Example:

function myFunction() {  
  var carName = "Volvo";   // Function Scope  
}

function myFunction() {  
  let carName = "Volvo";   // Function Scope  
}

## **Global Scope**

* Variables declared with var and let are quite similar when declared outside a block.
* They will both have **Global Scope**:
* Example:

var x = 2;       // Global scope

let x = 2;       // Global scope

## **Redeclaring**

* Redeclaring a JavaScript variable with var is allowed anywhere in a program:
* Example:

var x = 2;  
// Now x is 2  
   
var x = 3;  
// Now x is 3

* Redeclaring a var variable with let, in the same scope, or in the same block, is not allowed:
* Example:

var x = 2;       // Allowed  
let x = 3;       // Not allowed  
  
{  
  var x = 4;   // Allowed  
  let x = 5   // Not allowed  
}

* Redeclaring a let variable with let, in the same scope, or in the same block, is not allowed:
* Example:

let x = 2;       // Allowed  
let x = 3;       // Not allowed  
  
{  
  let x = 4;   // Allowed  
  let x = 5;   // Not allowed  
}

* Redeclaring a let variable with var, in the same scope, or in the same block, is not allowed:
* Example:

let x = 2;       // Allowed  
var x = 3;       // Not allowed  
  
{  
  let x = 4;   // Allowed  
  var x = 5;   // Not allowed  
}

* Redeclaring a variable with let, in another scope, or in another block, is allowed:
* Example:

let x = 2;       // Allowed  
  
{  
  let x = 3;   // Allowed  
}  
  
{  
  let x = 4;   // Allowed  
}

## **Let Hoisting**

* Variables defined with var are **hoisted** to the top and can be initialized at any time.
* Meaning: You can use the variable before it is declared:
* Example:

carName = "Volvo";  
alert(carName);  
var carName;

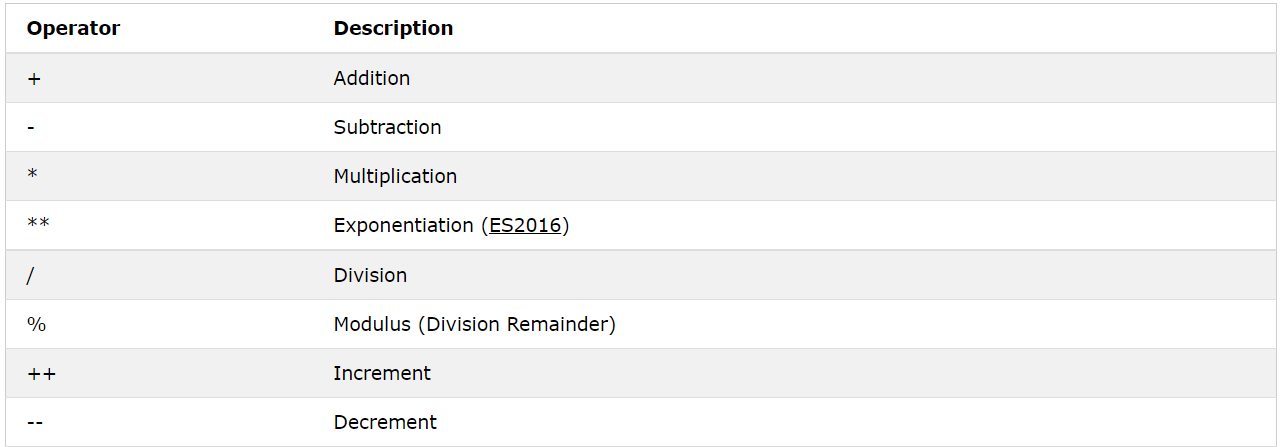
* Variables defined with let are also hoisted to the top of the block, but not initialized.
* Meaning: Using a let variable before it is declared will result in a ReferenceError:
* Example:

carName = "Saab";  
let carName = "Volvo";

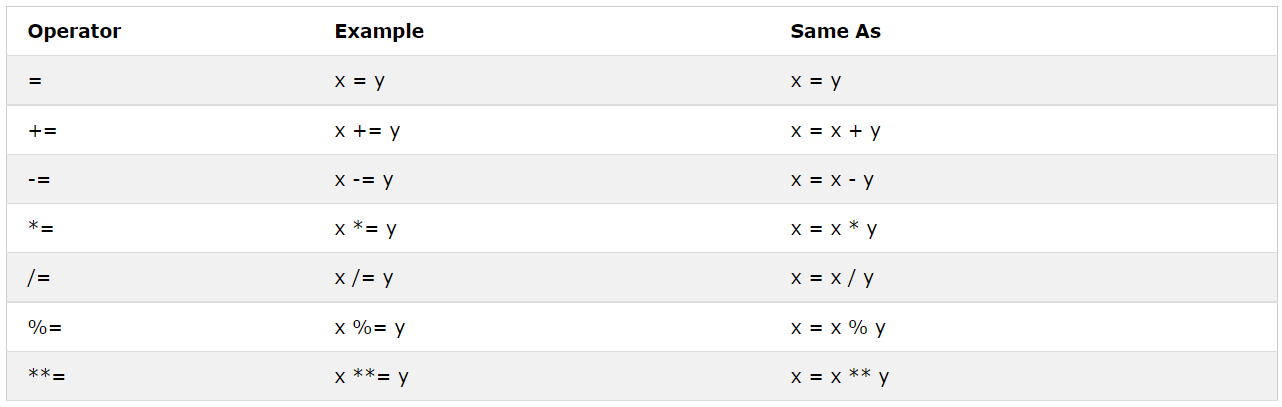
# JavaScript Operators

## **JavaScript Arithmetic Operators**

* Arithmetic operators are used to perform arithmetic on numbers:



## **JavaScript Assignment Operators**

* Assignment operators assign values to JavaScript variables.

## **JavaScript String Operators**

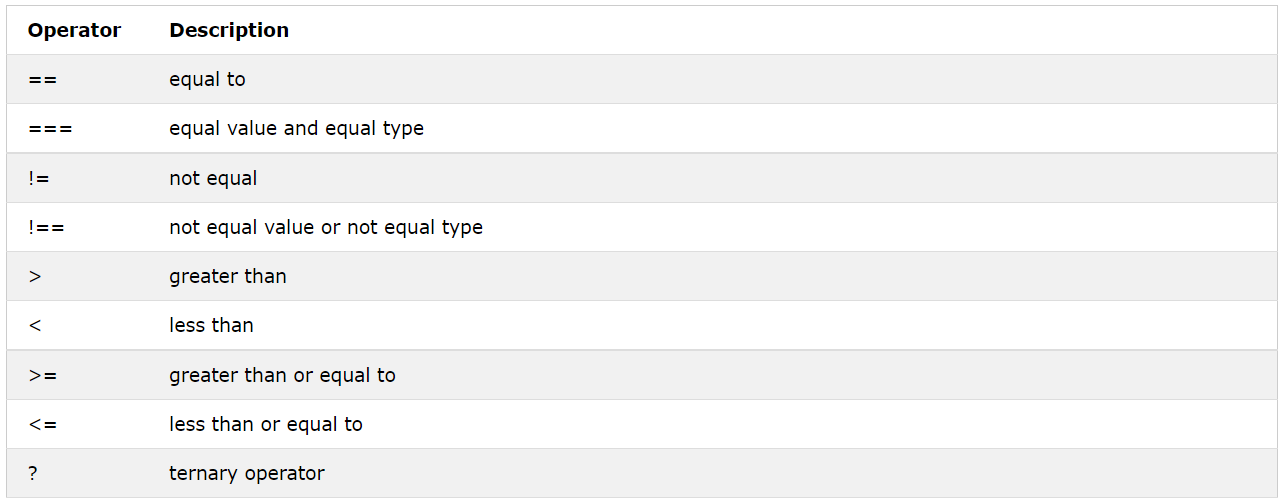
* The + operator can also be used to add (concatenate) strings.
* Example:

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

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

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

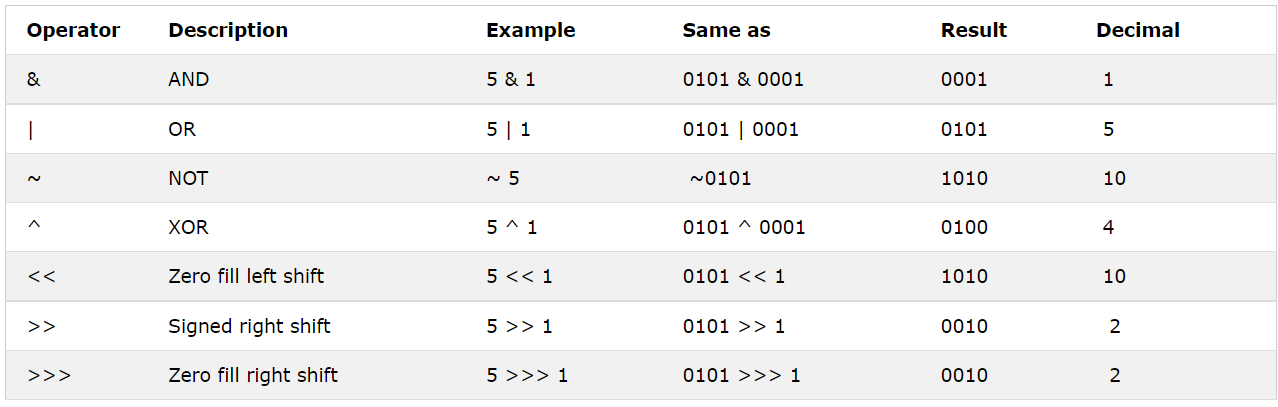
## **JavaScript Comparison Operators**



## **JavaScript Logical Operators**

## **JavaScript Type Operators**

## **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.