Neo Rays Software Solutions Pvt. Ltd

Contents

Version Control

Completing the following table makes it easy to come back later and track what changes were made to the requirements at each point in the project, who made them, and why they were made. This is a way of implementing change control on the BRD.

Revision History

Version #	Date	Responsibility (Author)	Description
0.1	December 15th , 2015	Meheboob Nadaf	

Acronyms and Glossary

The following table includes definitions for any unique symbols or notations that are used in the document.

Term	Definition

Table 1: Acronyms and Glossary

JavaScript Introduction

History of JavaScript

- JavaScript was created in 10 days in May 1995 by Brendan Eich, when working at Netscape and now of Mozilla.
- JavaScript was not always known as JavaScript: the original name was Mocha, a name chosen by Marc Andreessen, founder of Netscape.
- In September of 1995 the name was changed to LiveScript, then in December of the same year, upon receiving a trademark license from Sun, the name JavaScript was adopted.

What is JavaScript?

- JavaScript is *an object-based scripting language* that is lightweight and cross-platform.
- JavaScript is the programming language of HTML and the Web.
- It is designed for creating network-centric applications.
- JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.
- JavaScript does not designed as general purpose programming language but it designed to manipulate web pages.
- JavaScript is a major Web technology that provides interactivity and special effects to WebPages.
- JavaScript can be used inside another applications i.e. Web Browser.
- Operating system runs the Web Browser, and the Web Browser is having the implementation of JavaScript engine.

Why JavaScript is used in Web?

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation.
- Dynamic drop-down menus.
- Displaying Date and Time.
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box).
- Dynamic styling,
- Games
- Responses when buttons are pressed or data entered in forms
- animation
- Displaying Clock etc.

Other use of JavaScript

- Applications(Acrobat, Photoshop)
- Server-Side(Node.js, Google Apps Script)

Advantages of JavaScript

- **Less server interaction** − you can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − they don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − you can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

- **Richer interfaces** – you can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Disadvantages of JavaScript

- JavaScript can be disabled at client side (But it will not affect because we can enable it while writing JavaScript).
- Different browser can display web pages differently because of not having the proper implementation of CSS.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features –

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.
- Can’t access database directly.
- Can’t access Hardware (USB etc).

What are the tools to write JavaScript?

- Notepad
- Notepad++
- Dreamweaver
- Xcode
- Visual studios
- Aptana etc

JavaScript Example

```
<html>

<body bgcolor='pink'>

<h2>Welcome to Neo Rays Software Solutions Pvt. Ltd. </h2>

<script>

document.write("Hello this is JavaScript");

</script>

</body>

</html>
```

JavaScript Language

1) Character Set
2) Keywords
3) Identifiers
4) Variable
5) Data Types
6) Literals
7) Operators
8) Control Statement
9) Separators

1) Character Set:

- ➢ To display an HTML page correctly, the browser must know what character set (character encoding) to use.
- ➢ It is list of characters which developer can use for writing JavaScript code. Character set allows the following 3 types.

a) Numbers - 0 to 9.

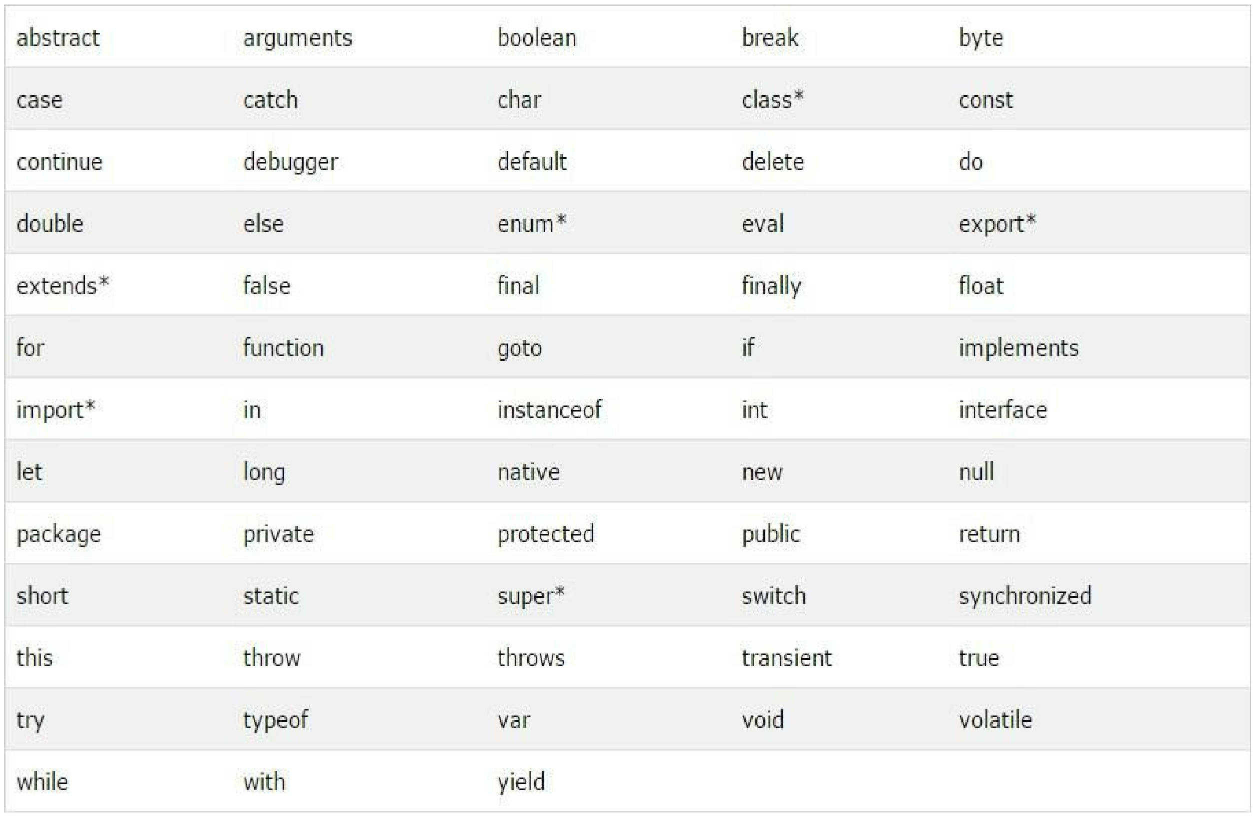
b) Alphabets – A to Z, A to z

c) Special Characters - #, $, etc

- ➢ It uses **UTF-16** (16-bit Unicode Transformation Format).

2) Keywords :

In JavaScript you cannot use these reserved words as variables, labels, or function names:



abstract	arguments	boolean	break	byte
case	catch	char	class*	const
continue	debugger	default	delete	do
double	else	enum*	eval	export*
extends*	false	final	finally	float
for	function	goto	if	implements
import*	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super*	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

Created By	Meheboob Nadaf
Prepared For	Neo Rays Software Solutions Pvt. Ltd.
Version	0.1
Created On	28 December, 2015
Last Updated On	

3) Identifiers :

JavaScript Identifiers are names; names that you give things in JavaScript. These JavaScript "things" include

- ➢ variable
- ➢ functions

- objects
- properties
- methods
- events

Like much of JavaScript, there are rules to be followed.

- Identifiers can only contain letters, numbers, underscore (_) and the dollar sign ($)
- Identifiers cannot start with a number
- Identifiers are case-sensitive
- Identifiers can be any length
- Identifiers cannot be the same as JavaScript reserved words
- Don't use global properties and methods as identifiers (more later)
- Don't use words similar to reserved words

When naming an identifier with two words in it, it's a best practice to use camel case. With this convention, the first letter of each word, excluding the first word, is uppercase. Example:

first**N**ame

my**C**ommon**V**ariable**N**ame

Here are some examples of valid identifier naming conventions:

- firstname
- totalPrice
- cust_1
- click_calculate
- $
- $total

4) Variables :

- Variable is a container used for temporary storage of data. The variables that are used in the program have to be declared before its use.
- 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).

- ➢ There are two types of variables in JavaScript: local variable and global variable.

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 _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

Correct JavaScript variables:

```
var x = 10;

var _value="NeoRays";
```

Incorrect JavaScript variables

```
var  123=30;

var *aa=320;
```

Example of JavaScript variable

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
```

JavaScript local variable:

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc()
```

```
{
var x=10;//local variable
}
</script>
```

JavaScript global variable:

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

5) Data types :

The Concept of Data Types

- ➢ In programming, data types are an important concept.
- ➢ To be able to operate on variables, it is important to know something about the type.
- ➢ Without data types, a computer cannot safely solve this:

JavaScript Data Types

JavaScript variables can hold many **data types as given below**

- Numbers
- Strings
- Arrays
- Objects

JavaScript Numbers

- ✓ It represents numeric values
- ✓ JavaScript has only one type of numbers.
- ✓ Numbers can be written with, or without decimals:
- ✓ Extra large or extra small numbers can be written with scientific (exponential) notation:

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

var y = 123e5; // 12300000( Written with scientific (exponential) notation)
var z = 123e-5; // 0.00123(be written with scientific (exponential) notation)
```

JavaScript Strings

- ✓ JavaScript strings are used for storing and manipulating text.
- ✓ A JavaScript string simply stores a series of characters like "Meheboob Nadaf".
- ✓ A string can be any text inside 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
- ✓ The length of a string is found in the built in property "**length**":
- ✓ The string will be chopped to "We are the so-called ".
- ✓ The solution to avoid this problem is to use the \ **escape character**.
- ✓ The backslash escape character turns special characters into string characters:
- ✓ The escape character (\) can also be used to insert other special characters in a string.
- ✓ 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:
- ✓ You can also break up a code line within a text string with a single backslash:
- ✓ The safest (but a little slower) way to break a long string is to use string addition:
- ✓ You cannot break up a code line with a backslash:
- ✓ Normally, JavaScript strings are primitive values, created from literals: **var firstName = "Meheboob"**
- ✓ But strings can also be defined as objects with the keyword new: **var firstName = new String("Meheboob")**
- ✓ Don't create strings as objects. It slows down execution speed.
- ✓ The new keyword complicates the code. This can produce some unexpected results:
- ✓ When using the == equality operator, equal strings looks equal:
- ✓ When using the === equality operator, equal strings are not equal, because the === operator expects equality in both type and value.
- ✓ JavaScript objects cannot be compared.
- ✓ Never create strings as objects

Example:
```
var name = "Meheboob";
var surname = 'Nadaf';

var ans = "It's JavaScript";
var ans = "He is called 'Meheboob'";
var ans = 'He is called "Meheboob"';
var x = 'It\'s alright';
var y = "We are the best \"Engineer's\" from India."
```

<u>This is the list of special characters that can be added to a text string with the backslash sign:</u>

Code	**Outputs**
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

String Properties and Methods

- ✓ In other languages Primitive values, like "Meheboob Nadaf", cannot have properties or methods (because they are not objects).
- ✓ But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Properties

Property	**Description**
constructor	Returns the function that created the String object's prototype
length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

String Methods

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a start position through a number of characters
substring()	Extracts a part of a string between two specified positions
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

JavaScript Arrays:

- ➢ Array in JavaScript refers to a group of elements sharing a common name.
- ➢ represents group of similar values
- ➢ JavaScript arrays are used to store multiple values in a single variable.

- An array is a special variable, which can hold more than one value at a time.
- An array can hold many values under a single name, and you can access the values by referring to an index number.
- You refer to an array element by referring to the index number.
- JavaScript variables can be objects. Arrays are special kinds of objects.
- Because of this, you can have variables of different types in the same Array.
- You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:
- Arrays are a special type of objects. The **typeof** operator in JavaScript returns "object" for arrays.
- The real strength of JavaScript arrays are the built-in array properties and methods:
- Many programming languages support arrays with named indexes. Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does **not** support arrays with named indexes.
- In JavaScript, **arrays** always use **numbered indexes.**
- Arrays are a special kind of objects, with numbered indexes.
- There is no need to use the JavaScript's built-in array constructor **new** Array() **Use [] instead.**
- The **new** keyword only complicates the code. It can also produce some unexpected results:
- The typeof operator returns object because a JavaScript array is an object. To solve this problem ECMAScript 5 defines a new method **Array.isArray():**

Examples 1:

```
var  numArray = new Array(88,99,45,22,11)  // Don't use new Array() because it slow
down     performance.
```

Examples 2:

```
var points = [40, 100, 1, 5, 25, 10];  // Create Array without using new  keyword its
recommended(Good) .
```

Examples 3:

```
myArray1 [0] = 20; // Values can be stored in an Array by using index position
myArray1 [1] = 15; // Values can be stored in an Array by using index position
myArray1 [2] = 30; // Values can be stored in an Array by using index position
```

Examples 4:

var districts = ["Bengaluru", "Belgaum", "Dharwad", "Hubli"]; //String Values can be stored in an Array

Examples 5:

```
var person = [];
person[0] = "Meheboob"; //Array can store different data types
person[1] = "Nadaf";//Array can store different data types
person[2] = 40000; ;//Array can store different data types
```

Examples 6:

```
myArray[0] = Date.now; //  Array Can Have Different Objects in One Array

myArray[1] = myFunction; //  Array Can Have Different Objects in One Array

myArray[2] = myCars; //  Array Can Have Different Objects in One Array
```

Array Methods :

Method	**Description**
toString()	converts an array to a string of (comma separated) array values.
join()	This method also joins all array elements into a string.
pop()	This method removes the last element from an array:
push()	This method adds a new element to an array (at the end):
shift()	This method removes the first element of an array, and "shifts" all other elements one place up.
unshift()	This method adds a new element to an array (at the beginning), and "unshifts" older elements:
splice()	This method can be used to add new items to an array or to remove an items:

reverse()	This method reverses the elements in an array.
slice()	This method slices out a piece of an array into a new array.
concat()	This method creates a new array by concatenating two arrays:
valueOf()	This method is the default behavior for an array. It converts an array to a primitive value.
sort()	This method sorts an array alphabetically:

JavaScript Objects:

- ➢ An Object, in context of JavaScript can be defined as collection of methods and properties consisting of a set of definable characteristics that one can view or modify.
- ➢ JavaScript objects are containers for **named values.**
- ➢ Data members define the Properties and the methods, which are also known as member functions, act upon the data members.
- ➢ A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- ➢ JavaScript is an object-based language. Everything is an object in JavaScript.
- ➢ JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

object={property1:value1,property2:value2.....propertyN:valueN} //Here property and value is separated by : (colon)

Example :

```
<script>

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);

</script>
```

2) JavaScript Object by creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, new keyword is used to create object.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

3) By using an Object constructor

- Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.
- The this keyword refers to the current object.
- The example of creating object by object constructor is given below.

```
<script>

function emp(id,name,salary){
```

```
this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);



document.write(e.id+" "+e.name+" "+e.salary);

</script>
```

Defining method in JavaScript Object

- We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.
- The example of defining method in object is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;

this.changeSalary=changeSalary;
function changeSalary(otherSalary){
this.salary=otherSalary;
}
}
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script>
```

Do Not Declare Strings, Numbers, and Booleans as Objects!

- ➢ When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

  Examples:

  ```
  var x = new String();        // Declares x as a String object
  var y = new Number();        // Declares y as a Number object
  var z = new Boolean();       // Declares z as a Boolean object
  ```

- ➢ Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
var a=40;//holding number

var b="Rahul";//holding string
```

JavaScript primitive data types:

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents Boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

Syntax :

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

6) Literals :

- It is value.
- It can be one of the following type
  a) Array Literals
  b) Integers literals
  c) Floating number Literals
  d) Boolean Literal
  e) Object Literal
  f) String Literal

a) Array Literals :

In JavaScript an array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [ ] ' . When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified. If no value is supplied it creates an empty array with zero length.

Creating an empty array:
var fruits = [ ];

Creating an array with four elements.
var fruits = ["Orange", "Apple", "Banana", "Mango"]

Comma in Array Literal:

There is no need to specify all elements in an array literal. If we put two commas in a row at any position in an array then an unspecified element will be created in that place.

The following example creates the fruits array :

fruits = ["Orange", , "Mango"]

This array has one empty element in the middle and two elements with values. (fruits[0] is "Orange", fruits[1] is set to undefined, and fruits[2] is "Mango").

If you include a single comma at the end of the elements, the comma is ignored. In the following example, the length of the array is three. There is no fruits[2].

fruits = ["Orange", "Mango",]

In the following example, the length of the array is four, and fruits[0] and fruits[2] are undefined.

fruits = [, 'Apple', , 'Orange'];

b) Integers literals

An **integer** must have at least one digit (0-9).

- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative, if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.

1. Decimal (base 10)
Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.

Example: 123, -20, 12345

2. Hexadecimal (base 16)
Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f. A leading 0x or 0X indicates the number is hexadecimal.

Example: 7b, -14, 3039

3. Octal (base 8)
Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.

Example: 173, -24, 30071

c) Floating number Literals:

A floating number has the following parts.

- A decimal integer.
- A decimal point ('.').
- A fraction.
- An exponent.

The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

Examples**:**

- 8.2935
- -14.72
- 12.4e3 [Equivalent to 12.4 x 10^3]
- 4E-3 [Equivalent to 4 x 10^{-3} => .004]

d) Boolean Literal:

The Boolean type has two literal values:

- true
- false

e) Object Literal:

An object literal is zero or more pairs of comma separated list of property names and associated values, enclosed by a pair of curly braces.

In JavaScript an object literal is declared as follows:

1. An object literal without properties:

```
var userObject = {}
```

2. An object literal with a few properties :

```
var student = {
First-name : "Suresy",
Last-name : "Rayy",
Roll-No : 12
};
```

Syntax rules:

Object literals maintain the following syntax rules:

- There is a colon (:) between property name and value.
- A comma separates each property name/value from the next.
- There will be no comma after the last property name/value pair.

f) String Literal:

JavaScript has its own way to deal with string literals. A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings. The following are the examples of string literals :

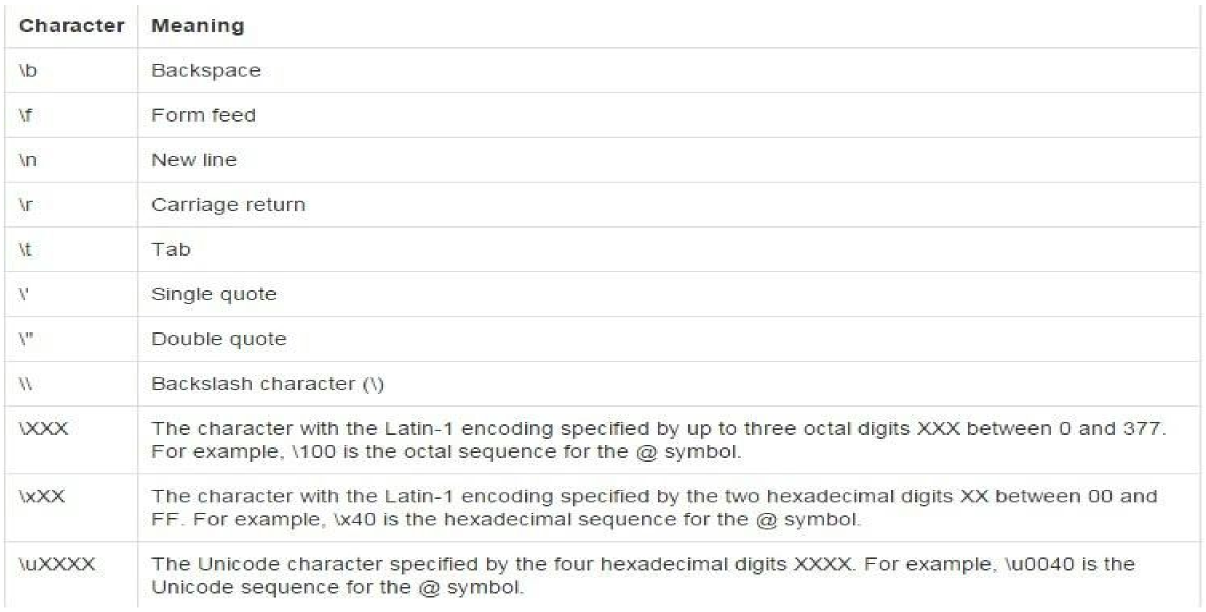
- string1 = "NeoRays.com"
- string1 = ‘NeoRays.com'

- string1 = "1000"
- string1 = "google" + ".com"

In addition to ordinary characters, you can include special characters in strings, as shown in the following table.

string1 = "First line. \n Second line."

List of special characters used in JavaScript:



Character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\'	Single quote
\"	Double quote
\\	Backslash character (\)
\XXX	The character with the Latin-1 encoding specified by up to three octal digits XXX between 0 and 377. For example, \100 is the octal sequence for the @ symbol.
\xXX	The character with the Latin-1 encoding specified by the two hexadecimal digits XX between 00 and FF. For example, \x40 is the hexadecimal sequence for the @ symbol.
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. For example, \u0040 is the Unicode sequence for the @ symbol.

7) Operators:

JavaScript operators are symbols that are used to perform operations on operands. For example:

var sum=10+20;

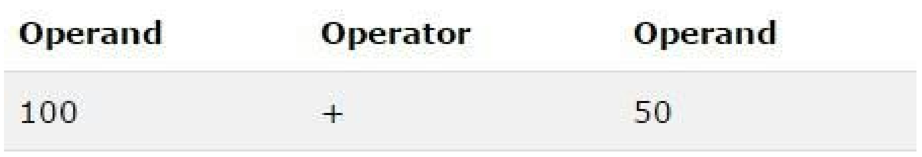
Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

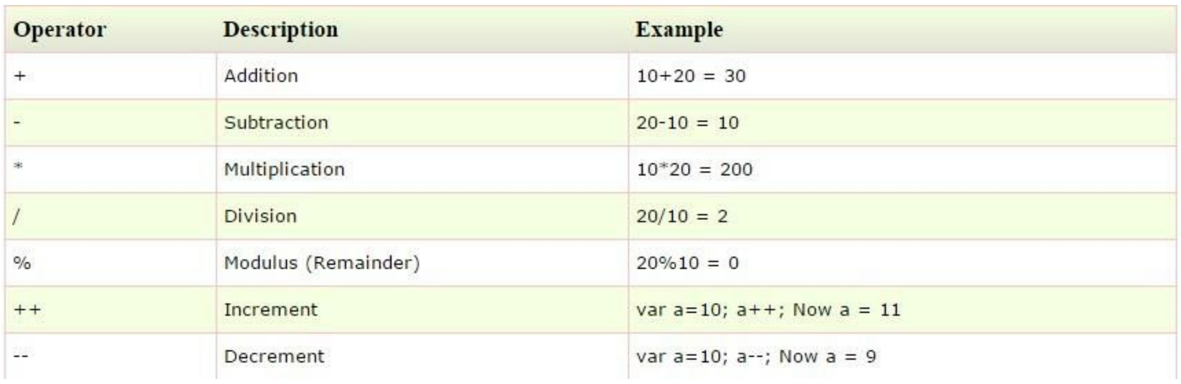
1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

1) Arithmetic Operators:

- ➢ Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.
- ➢ Arithmetic operators perform arithmetic on numbers (literals or variables).
- ➢ The numbers (in an arithmetic operation) are called **operands**.
- ➢ The operation (to be performed between the two operands) is defined by an **operator**.



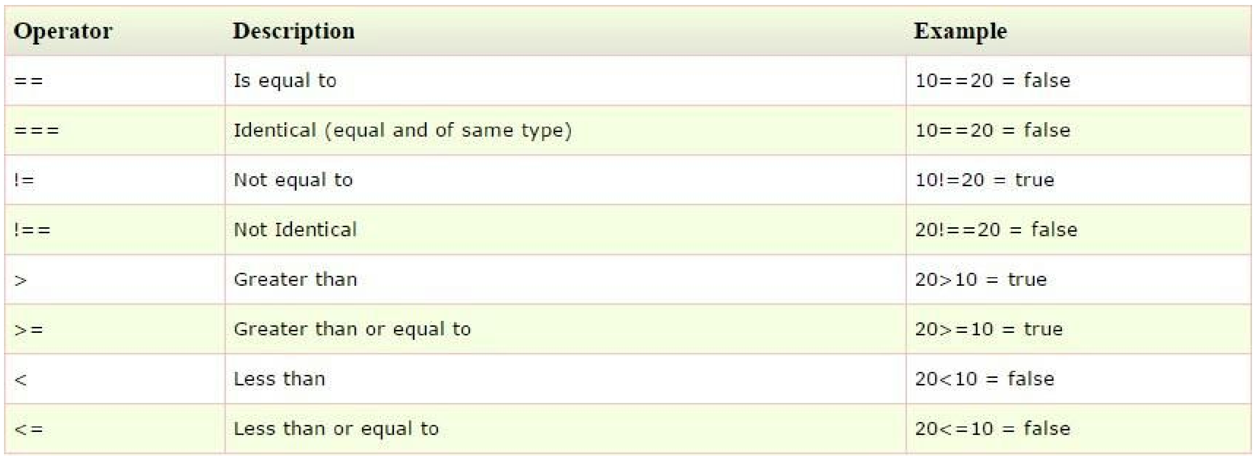
Operand	Operator	Operand
100	+	50



Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

2) JavaScript Comparison Operator:

- The JavaScript comparison operator compares the two operands. The comparison operators are as follows:



Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

3) Bitwise Operators:

- The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Sr.No	Operator and Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
3	**^ (Bitwise XOR)** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. **Ex:** (A ^ B) is 1.
4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
5	**<< (Left Shift)**It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. **Ex:** (A << 1) is 4.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.

	Ex: (A >> 1) is 1.
7	**>>> (Right shift with Zero)** This operator is just like the >> operator, except that the bits shifted in on the left are always zero. **Ex:** (A >>> 1) is 1.

4) Logical Operator:

➢ JavaScript supports the following logical operators −Assume variable A holds 10 and variable B holds 20, then –

Sr.No	Operator and Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.

2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

5) Assignment Operator:

➢ JavaScript supports the following assignment operators –

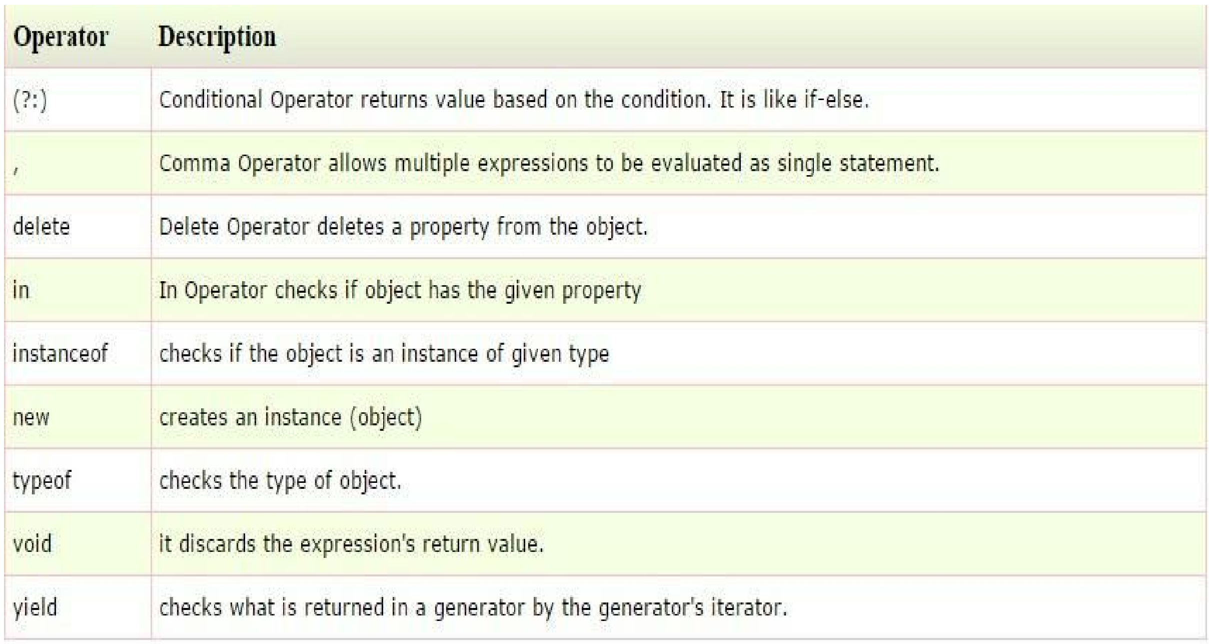
Sr.No	Operator and Description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A

3	**−= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note − same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, \|= and ^=.

6) Special Operators:

➢ The following operators are known as JavaScript special operators.



Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

8) Control Statement:

1. if / else
2. switch / case
3. for
4. while and do / while
5. break
6. continue

1. if / else :

"if statement" is used to perform conditional checks.

a.

```
if (condition)
{
        Statement 1
}
```

Statement X;

In the 1st scenario first condition will be checked. If the condition is true the statement 1 will be executed and then control will be transferred to statement X. If condition is false then control will be transferred to statement X without executing statement **1**.

b.

```
if (condition)
{
        Statement 1
}
else
{
        Statement 2
}
Statement X;
```

In the 2nd scenario if the condition is true then statement 1 will be executed and the control will be transferred to statement X. If the condition is false then statement 2 will be executed and control will be transferred to statement X. Here either if or else condition will execute and both of them will never execute together.

c.

```
if (condition)

{
        Statement 1

}

else if (condition)

{
        Statement 2
}

else
{
        Statement 3
}

Statement X;
```

In the 3rd scenario if the condition is true then statement 1 will be executed and then control will be transferred to statement X. If the condition is false it will come to else if part and checks the condition. If true then Statement 2 will be executed and control will be transferred to statement X. If false then statement 3 will be executed and control will be transferred to statement X.

2. switch / case :

- Switch statement is used to execute a particular set of statements among multiple conditions.
- It is alternative to complicated if else if ladder conditions.
- The expression type in the switch must be any of the following data type.

- break is optional. Every case must end with break statement which will help to terminate and transfer the control outside of switch block. If no break is used then execution will continue to next case.
- default is optional. If present it will execute only if the value of the expression does not match with any of the case and then control goes to statement X. If not present then control will exit switch statement and goes to statement X.

3. for

- for statement is used to execute a set of statements multiple times.

```
      1           2             4
for (initialization;condition;increment/decrement)


{             3
        Statement 1;
}
```

- It is alternative to while loop. In while loop initialization, condition, statements and increment/decrement are not defined in single statement.
- When for statement is encountered 1st initialization statement will be executed and then condition will be verified. If condition is true then statements inside for block will be executed and then increment or decrement operation will be executed. After increment or decrement operation again condition will be verified. If the condition is true then again for block will be executed and same process will be repeated till condition becomes false. If the condition is false then control will come out the loop and executes statement X.

4. while and do/while :

while:

- while is an entry controlled loop. In while statement 1st condition will be verified. If the condition is true loop will be repeated .If the condition is false then control will come out of the loop.

do/while:

- do/while is exit controlled loop. In do while first all the statements inside the block will be executed and then condition will be verified. If the condition is true then loop will be repeated. If condition is false then control will come out of the loop and execute statement X.
- In while when the condition is false for the first time then the statement inside the block will be executed for 0 times, whereas in do while statement will be surely executed for one time.

5. **break :**
    - It helps to transfer control to another part of the program.
    - It can be used in switch statement or in do, while and for loops. It also can be used in labeled blocks.
    - It has the following uses.
        - It helps in terminating a switch statement.
        - It is used to exit a loop or force immediate termination of a loop bypassing the conditional expression and other remaining code in the loop.
6. **continue :**
    - It is used to move the control to the beginning of the loop.
    - When continue is used in while or do-while loop then control is directly transferred to the test condition that controls the loop.
    - When continue is used in for loop control is transferred to iteration portion first then to the test condition.

Note: break is used to move the control to the end of the loop but continue is used to move the control to the beginning of the loop.

7. **return :**

- return keyword terminates the execution in a method and returns the control to the caller method.
- It has two uses:
    a. It is immediately terminates the execution of the callee method and returns the control from callee to the caller method.
    b. It is used to return a value from callee to caller method.

8. goto :

- JavaScript does indeed have a "goto" keyword but is just a reserved keyword.

9) Separators

Symbol	Name	Purpose
( )	parenthesis	Used to contain lists of parameters in method defining and invocation. Also used in control statements for defining conditions.
{}	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block, methods and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates the statements,
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside for statement.
.	Period(Dot)	Used to access members of objet with reference variable and in prototype objects.

JavaScript Object Prototypes

- Every JavaScript object has a prototype. The prototype is also an object.
- All JavaScript objects inherit their properties and methods from their prototype

JavaScript Prototypes

- All JavaScript objects inherit the properties and methods from their prototype.
- Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype.
- Objects created with new Date() inherit the Date.prototype.
- The Object.prototype is on the top of the prototype chain.
- All JavaScript objects (Date, Array, RegExp, Function, ...) inherit from the Object.prototype.

Creating a Prototype

- The standard way to create an object prototype is to use an object constructor function:

Example:

```
function personDetails(first, last, age, eyecolor) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eyecolor;
}
```

- With a constructor function, you can use the **new** keyword to create new objects from the same prototype:

```
<script>
function Player(n,s,r)
        {
                this.name = n;
                this.score = s;
                this.rank = r;

        }
        var nadaf = new Player("Nadaf",1000,1);

                console.log(nadaf.name);
                console.log(nadaf.score);
                console.log(nadaf.rank);
                var sachin = new Player("sachin ",2222,7);
        </script>
```

Note: The constructor function is the prototype for your person objects.

Adding a Property to an Object

- Adding a new property to an existing object :

```
nadaf.prize = "1st";
console.log(nadaf.prize);
```

Note: The property will be added to nadaf. Not to sachin. Not to any other person objects.

Adding a Method to an Object

➢ Adding a new method to an existing object:

```
nadaf.tellName = function()
        {
                console.log("My surName is "+this.name);
        }
        nadaf.tellName();
```

Adding Properties to a Prototype

➢ You cannot add a new property to a prototype the same way as you add a new property to an existing object, because the prototype is not an existing object.

```
Player.position = "2nd";

console.log("Nadaf position is "+nadaf.position);//This will print undefined
```

To add a new property to a constructor, you must add it to the constructor function:

```
function Player(n,s,r)

        {

                this.name = n;

                this.score = s;

                this.rank = r;

                this.natioanlity = "Indian";


        }

        var nadaf = new Player("Nadaf",1000,1);

console.log("Nadaf Nationality is "+nadaf.natioanlity);//prints Indian
```

```
var sachin = new Player("Sachin",7777,421);

console.log("Sachin Nationality is "+ Sachin.natioanlity);//prints Indian
```

Using the **prototype** Property

- The JavaScript prototype property allows you to add new properties to an existing prototype:
- The JavaScript prototype property also allows you to add new methods to an existing prototype:

```
Player.prototype.logInfo = function()

        {

                console.log("I am :  "+this.name);

        }


Player.prototype.promote = function()

        {

                this.rank++;

                console.log("I My new rank is  :  "+this.rank);

        }


        nadaf.logInfo();

        nadaf.promote();

        sachin.logInfo();
sachin.promote();
```

Note: Only modify your **own** prototypes. Never modify the prototypes of standard JavaScript objects.

OPPS Concepts in JavaScript

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** − the capability to store related information, whether data or methods, together in an object.
- **Aggregation** − the capability to store one object inside another object.
- **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Difference between Java and JavaScript

- One thing we should know that Java and JavaScript are both programming languages and these two are completely different with each other and there is nothing to do with each other. In below table describes the difference between Java and JavaScript.
- It is important for us to know the difference between JAVA & JavaScript Because of its naming conflicts.

Sl	Java	JavaScript

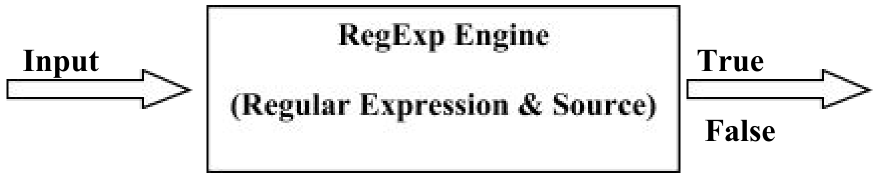
1	Object oriented programming language(encapsulation+abstraction+compile time polymorphism+inheritance+dynamic polymorphism)	Object Based programming language(encapsulation+abstraction+compile time polymorphism+inheritance)
2	It supports all OOP features	It will not supports all OOP features
3	class Based	Prototype based
4	class & Instance are distinct entities	All Objects can inherit from another Object
5	Define a class with class definition, instance a class with constructor method	Define & create a set of Objects with constructor function
6	Create single Object with new Keyword	same
7	Construct an Object hierarchy by using class definitions to define subclass of existing Class	Construct an object hierarchy by assigning an object as the prototype associated with a constructor function.
8	Inherit properties by following the class chain.	Inherit properties by following the prototype chain.
9	Class definition specifies *all* properties of all instances of a class. Cannot add properties dynamically at run time.	Constructor function or prototype specifies an *initial set* of properties. Can add or remove properties dynamically to individual objects or to the entire set of objects.
10	Java developed James Gosling & team. The small team of sun engineers called Green Team in 1995.Befor the name of JAVA it was known by the name of Green talk & Oak. Now JAVA is taken over by Oracle.	JavaScript developed by Brendan Eich, in May 1995 When working at Netscape. Before the name of JavaScript it was known by Mocha & LiveScript. Now JavaScript is taken over by Mozilla

JavaScript Regular Expression

- JavaScript supports regular expression processing.
- Regular expression is a string of characters that describes a character sequence. The abbreviation for regular expression is RegExp.

- A regular expression is generally used for validation, pattern-matching, searching or manipulating text
- In JavaScript, regular expressions are also objects
- A regular expression is an object that describes a pattern of characters.
- A regular expression is a sequence of characters that forms a search pattern.
- The search pattern can be used for text search and text replaces operations.

- When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of text search and text replaces operations.
- Pattern is used to build a regular expression (RegExp).
- Matcher is used to match the pattern (RegExp) built by pattern.
- Any language that supports reggulor expression provides RegExp engine. Java language provides the Matcher to invoke the RegExp engine and do the matching operations.



Syntax:

/pattern/modifiers;

Example:

var pattern1 = /NeoRays/i

Example explained:

- / NeoRays /i is a regular expression.
- NeoRays is a pattern (to be used in a search).
- i is a modifier (modifies the search to be case-insensitive).

Regular Expression:

- ➢ A regular expression contains normal characters, meta characters, quantifiers or characters (set of characters).
- ➢ Meta characters, quantifiers or character (a set of characters) are used for building regular expression to match dynamic values.

1) **Normal Characters:**

- A normal character matches exactly as it is.

Example: var patt = /Meheboob123/i

Example explained: / Meheboob123 /i is a regular expression.

Meheboob123 is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

2) **Metacharacters** :

- Are symbols used to build regular expression to match dynamic values.
- Metacharacters are characters with a special meaning

Metacharacter	**Description**
.	Find a single character, except newline or line terminator
\w	Find a word character

\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

3) **Quantifiers** :

- Are used to specify the quantity or number of occurrences. A quantifier is also known as occurrence indicator.

Quantifier	**Description**
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s

n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

4) Modifiers:

- Modifiers are used to perform case-insensitive and global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

5) Brackets:

- Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any digit between the brackets
[^0-9]	Find any digit NOT between the brackets
(x|y)	Find any of the alternatives specified

6) *RegExp* Object Properties:

Property	Description
constructor	Returns the function that created the RegExp object's prototype
global	Checks whether the "g" modifier is set
ignoreCase	Checks whether the "i" modifier is set
lastIndex	Specifies the index at which to start the next match
multiline	Checks whether the "m" modifier is set

source	Returns the text of the RegExp pattern

7) RegExp Object Methods:

Method	Description
compile()	Deprecated in version 1.5. Compiles a regular expression
exec()	Tests for a match in a string. Returns the first match
test()	Tests for a match in a string. Returns true or false
toString()	Returns the string value of the regular expression

The Document Object Model (DOM)

- The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. The nodes of every document are organized in a tree structure, called the DOM tree.

- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM defines a standard for accessing documents:
- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
- The W3C DOM standard is separated into 3 different parts:
    - Core DOM - standard model for all document types
    - XML DOM - standard model for XML documents
    - HTML DOM - standard model for HTML documents

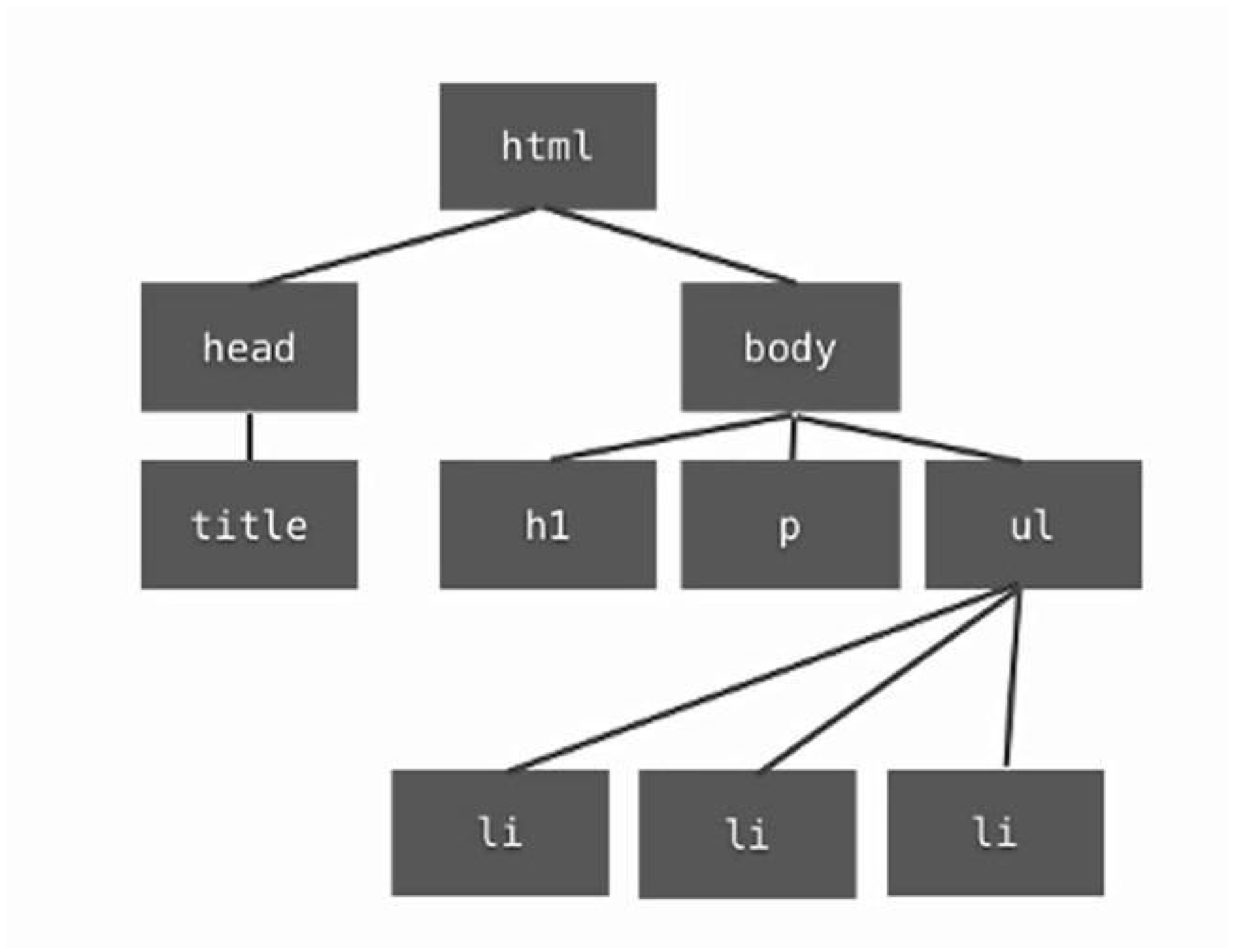
JavaScript HTML DOM

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:
- Here **Document** is :
    - HTML Page
    - Source code
- Here **Object** is :
    - It can be contents of inside the Unordered list
    - It can be contents of inside the body tag
    - It can be contents of inside the HTML tag As shown in figure boxes
- Here **Model** is: Agreed upon set of terms:
- When a web page is loaded, the browser creates a Document Object Model of the page.
- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- The **HTML DOM** model is constructed as a tree of **Objects**:

- With the object model, JavaScript gets all the power it needs to create dynamic HTML:
    - JavaScript can change all the HTML elements in the page
    - JavaScript can change all the HTML attributes in the page
    - JavaScript can change all the CSS styles in the page
    - JavaScript can remove existing HTML elements and attributes
    - JavaScript can add new HTML elements and attributes
    - JavaScript can react to all existing HTML events in the page
    - JavaScript can create new HTML events in the page



What is the HTML DOM?

- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
    - The HTML elements as **objects**
    - The **properties** of all HTML elements
    - The **methods** to access all HTML elements
    - The **events** for all HTML elements
- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

JavaScript - HTML DOM Methods

- HTML DOM methods are **actions** you can perform (on HTML Elements).
- HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
- The programming interface is the properties and methods of each object.
- A **property** is a value that you can get or set (like changing the content of an HTML element).
- A **method** is an action you can do (like add or deleting an HTML element).

The getElementById Method

- The most common way to access an HTML element is to use the id of the element.
- In the example above the getElementById method used id="demo" to find the element.

The innerHTML Property

- The easiest way to get the content of an element is by using the **innerHTML** property.
- The innerHTML property is useful for getting or replacing the content of HTML elements.
-

```
<html>
<body>
```

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

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

</body>
</html>
```

JavaScript HTML DOM Events

- ➢ The event object was interdicted in version 1.2 of JavaScript.

- ➢ It is used for providing additional information about the events.
- ➢ For example the key that was pressed when key press events occurred, the mouse button that was pressed when a mouse Down event occurred and the x, y coordinates of point where the mouse was clicked.

Reacting to Events

- ➢ A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- ➢ To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

Syntax: onclick=JavaScript

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Example:
```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.innerHTML=This is the new Text.'">Click on this text!</h1>

</body></html>
```

HTML Event Attributes

- ➢ To assign events to HTML elements you can use event attributes.

Assign Events Using the HTML DOM

- The HTML DOM allows you to assign events to HTML elements using JavaScript:

The onload and onunload Events

- The onload and onunload events are triggered when the user enters or leaves the page.
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The onload and onunload events can be used to deal with cookies.

The onchange Event

- The onchange event are often used in combination with validation of input fields.
- An example of how to use the onchange is the upperCase() or the lowercase() function will be called when a user changes the content of an input field.

The onmouseover and onmouseout Events

- The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

The onmousedown, onmouseup and onclick Events

- The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

HTML/DOM events for JavaScript

Events	**Description**
Onclick	occurs when element is clicked.

ondblclick	occurs when element is double-clicked.
onfocus	occurs when an element gets focus such as button, input, textarea etc.
Onblur	occurs when form looses the focus from an element.
onsubmit	occurs when form is submitted.
onmouseover	occurs when mouse is moved over an element.
onmouseout	occurs when mouse is moved out from an element (after moved over).
onmousedown	occurs when mouse button is pressed over an element.
onmouseup	occurs when mouse is released from an element (after mouse is pressed).
Onload	occurs when document, object or frameset is loaded.
onunload	occurs when body or frameset is unloaded.
onscroll	occurs when document is scrolled.
onresized	occurs when document is resized.
onreset	occurs when form is reset.
onkeydown	occurs when key is being pressed.
onkeyup	occurs when key is released.

Reasons to avoid JavaScript

JavaScript is not only the most popular programming language in the world but it one of the most badly written. There are many reasons people to avoid JavaScript. Those are:

1. Document.write
2. Browser Sniffing

3. Eval
4. Pseudo-Protocols

1. Document.write

- ➢ Document.write does not work in XHTML and does not understand DOM properly

Example:

```
<!DOCTYPE HTML>

<html>

<head>

<title>This is Error page title</title>

</head>

<body bgcolor="wheat">

<h1>Here I am avoiding JavaSript</h1>

<p>This page is used to avoid Javasript Mistakes</p>

<script >

document.write("This is inside script Tag");

document.write("This is inside script Tag This can include<strong>Strong!!!
Text</strong>");

</script>

</body>

<html>
```

This code will work properly But when you use document.write with other functions it will cause some problems i.e

Example :

```
<!DOCTYPE HTML>

<html>

<head>

<title>This is Error page title</title>

</head>

<body bgcolor="wheat">

<h1>Here I am avoiding JavaSript</h1>

<p>This page is used to avoid Javasript Mistakes</p>

<script >

function simpleFunction()

        {

                document.write("This is inside script Tag");

                document.write("This is inside script Tag This can
include<strong>Strong!!! Text</strong>");

        }


        window.onload = function()

                {

                        setTimeout(simpleFunction,500);
                };

</script>

</body>

<html>
```

In this code as soon as the time gets over, the body which is consisting of heading and paragraph will disappear and only the contents of inside the script will remain in the browser window. This is the one of the reason to avoid using JavaScript.

2. Browser Sniffing :
   - ➢ While Browser detection JavaScript Will not detect browser, instead it detects the features.

   Example: In this example I am asking the browser name and I am running it in chrome but it will show the features.

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

        <p>What is the name(s) of your browser?</p>

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

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

        <script>
        function myFunction() {
        document.getElementById("demo").innerHTML =
        "Name is " + navigator.appName +
        "<br>Code name is " + navigator.appCodeName;
        }
        </script>

        </body>
        </html>
```

Output is :

```
 Name is Netscape
Code name is Mozilla
```

3. Eval :

Eval is a powerful little function take an expression executes it as code. But it will not execute the code properly for example

Example:

```
<html>
<body>
<script>
var a = "alert('";
var b = "hello";
var c = "');";
eval(a+b+c);
</script>
</body>
</html>
```

This program will show only “hello” in dialog box.

4. Pseudo-Protocols:
   - In HTML inside of anchor tag we usually find hyperlink references but, instead of hyperlink we may find JavaScript Objects which doesn’t make any sense. This may cause unexpected results.

Examples : <p>Inside HTML ,You may find :

<a href=”JavaScript:someFunction()”>this</a>

</p>

JavaScript Lint (JSLint):

What It Is

- Many JavaScript implementations do not warn against questionable coding practices. Yes, that's nice for the site that "works best with Internet Explorer" (designed with templates, scripted with snippets copied from forums). But it's a nightmare when you actually want to write quality, maintainable code.
- That's where JavaScript Lint comes in. With JavaScript Lint, you can check all your JavaScript source code for common mistakes without actually running the script or opening the web page.
- JavaScript Lint holds an advantage over competing lints because it is based on the JavaScript engine for the Firefox browser. This provides a robust framework that can not

only check JavaScript syntax but also examine the coding techniques used in the script and warn against questionable practices.

What It Does

Here are some common mistakes that JavaScript Lint looks for:

- Missing semicolons at the end of a line.
- Curly braces without an *if*, *for*, *while*, etc.
- Code that is never run because of a *return*, *throw*, *continue*, *or break*.
- Case statements in a switch that do not have a *break* statement.
- Leading and trailing decimal points on a number.
- A leading zero that turns a number into octal (base 8).
- Comments within comments.
- Ambiguity whether two adjacent lines are part of the same statement.
- Statements that don't do anything.

JavaScript Lint also looks for the following less common mistakes:

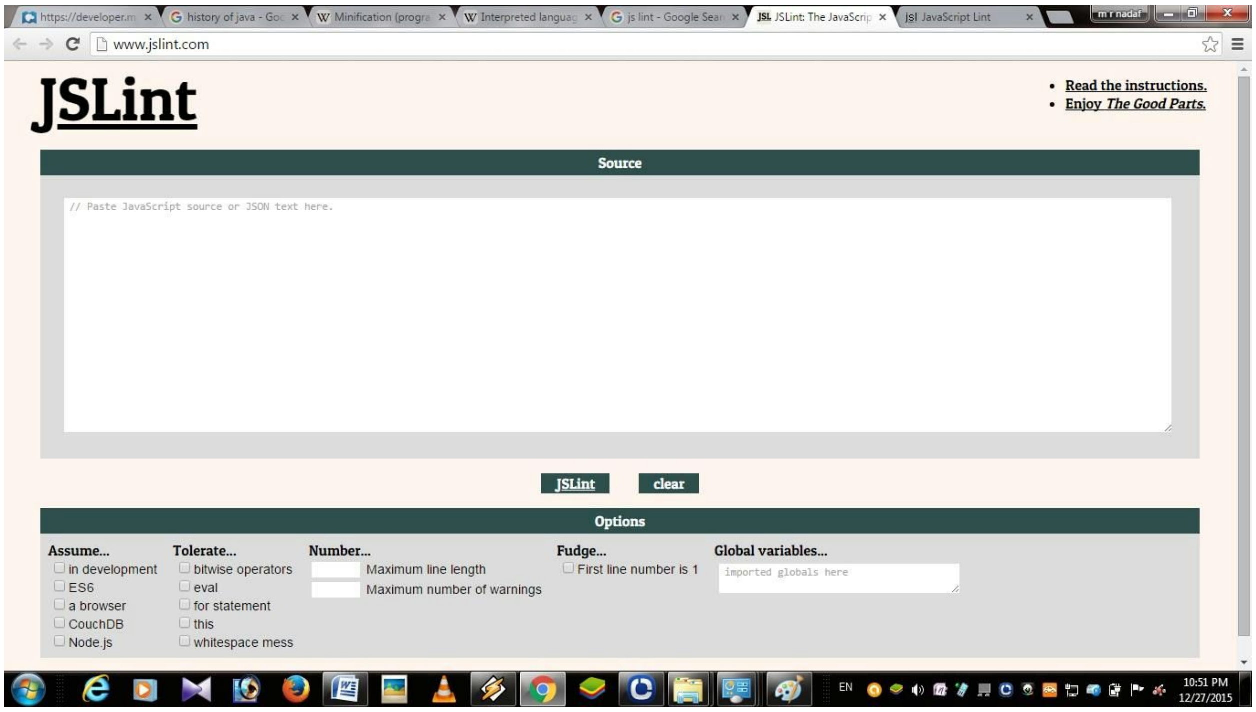
- Regular expressions that are not preceded by a left parenthesis, assignment, colon, or comma.
- Statements that are separated by commas instead of semicolons.
- Use of increment (++) and decrement (--) except for simple statements such as "i++;" or "--i;".
- Use of the *void* type.
- Successive plus (e.g. x+++y) or minus (e.g. x---y) signs.
- Use of labeled *for* and *while* loops.

- *if*, *for*, *while*, etc. without curly braces. (This check is disabled by default.)

Procedure for JSLint:

Step 1: To open JSLint paste this url in browser http://www.jslint.com/

Step 2: You will get this window on your browser as shown in below



Step 3: Paste your script in JSLint and click on JSLint button

Step 4: Follow the instructions given by the JSLint correct your script according to that.

Step 5: After removing all errors copy the code into your script.js file or you can use Minification tools to compress your code for better performance.

Minification :

- Minification (also minimisation or minimization), in computer programming languages and especially **JavaScript**, is the process of removing all unnecessary characters from source code without changing its functionality.
- Minified source code is especially useful for interpreted languages deployed and transmitted on the Internet (such as JavaScript), because it reduces the amount of data that needs to be transferred.
- Minification can be distinguished from the more general concept of data compression in that the minified source can be interpreted immediately without the need for an uncompression step: the same interpreter can work with both the original as well as with the minified source.
- Minification is used in JavaScript for removing unwanted things in Script like (white spaces, comments & long variable names etc).
- Thus browser will take very less time to interpret the script and it will give back the response very soon compared to original script source code.
- There are two advantages of doing Minificaion those are :
    1. Browser will give quick response.
    2. Your script will not understand by others when they do view source on web page but they can copy your script.

Note: The main reason of doing Minification is speeds up the process not for security purpose.

Tools used for JavaScript Minification:

➢ There are many tools and online Minification tools are available. Here is few list given below :

- JSmin
- Packer
- Google Closure compiler
- Microsoft Ajax Minifier
- Pretty Diff
- MinifyPS
- Online JS Minify

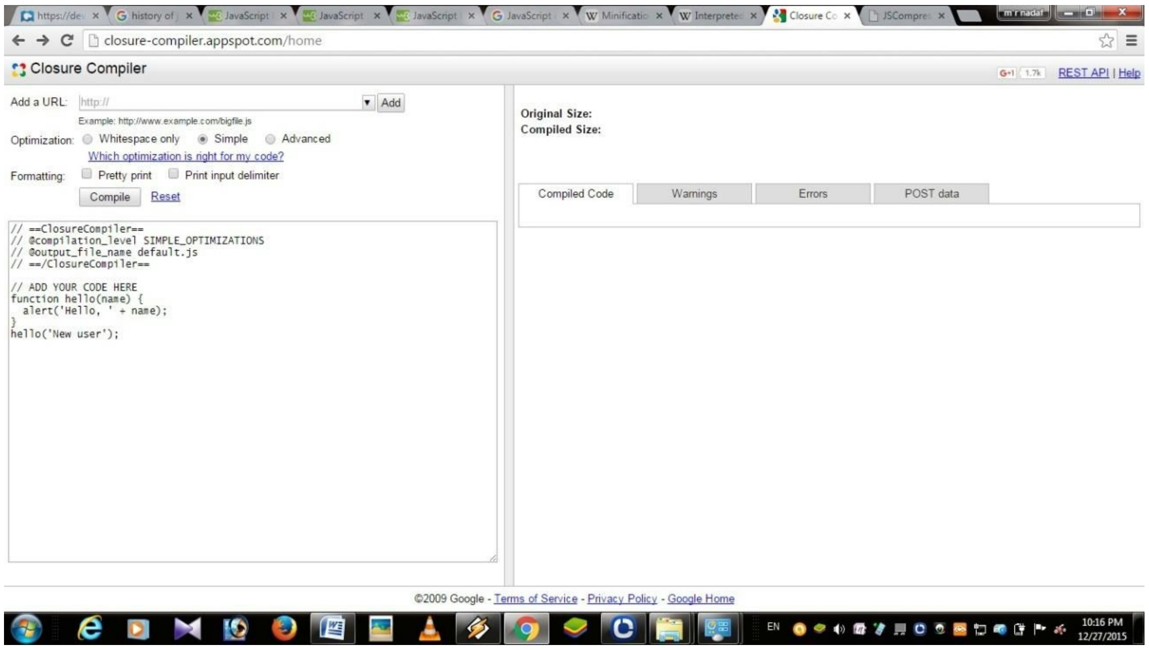
Example:

Google Closure compiler:

➢ Here I shown how to Minify JavaScript using Google Closure compiler.

Steps 1: Type this url in browser http://closure-compiler.appspot.com/home

Step 2: After typing this url you will get this window as shown in below



Add a URL: http://
Example: http://www.example.com/bigfile.js
Optimization: Whitespace only [✓ Simple] Advanced
Formatting: Pretty print, Print input delimiter

```
// ==ClosureCompiler==
// @compilation_level SIMPLE_OPTIMIZATIONS
// @output_file_name default.js
// ==/ClosureCompiler==

// ADD YOUR CODE HERE
function hello(name) {
  alert('Hello, ' + name);
}
hello('New user');
```

Original Size:
Compiled Size:

closure-compiler.appspot.com/home
Closure Compiler
Which optimization is right for my code?
Compile
Reset
REST API | Help
Compiled Code
Warnings
Errors
POST data
©2009 Google - Terms of Service - Privacy Policy - Google Home

Step 3: Copy your script from your editor and paste it in left side of your browser window and before pasting your script click on reset button or clear all default script written by Google as shown in fig.

Step 4: After pasting your script you’re to select radio buttons of optimization. There are three options.

1. Whitespace only: It removes only white spaces (The size of compiled script is little less compared to original script).
2. Simple: It will remove all comments and white spaces (This will reduce the size of the original script more compared to only white space procedure).
3. Advanced: It will remove most of the unwanted things in the script compared to above two options (This will almost reduce the size of the script and the script size also very less compared two above two).

Step 5: After selecting the radio buttons click on Compile button.

Step 6: If there is no errors in your script you will get compiled code on right side copy the script and paste it in your scrpt.js file. If you get any error then debug your script and try once again