Unit - III

JAVASCRIPT Introduction :

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage..

JavaScript was developed by Brendan Eich in 1995, which appeared in Netscape, a popular browser of that time.

The language was initially called LiveScript and was later renamed JavaScript.

Core features of JavaScript :

some important features of JavaScript.

Light Weight Scripting language

Dynamic Typing

Object-oriented programming support

Functional Style

Platform Independent

Prototype-based

Interpreted Language

Async Processing

Client-Side Validation

More control in the browser

JavaScript Datatypes:

There are 8 basic data types in JavaScript.

number for numbers of any kind: integer or floating-point, integers are limited by ±(253-1).

bigint is for integer numbers of arbitrary length.

string for strings. A string may have zero or more characters, there's no separate single-character type.

boolean for true/false.

null for unknown values – a standalone type that has a single value null.

undefined for unassigned values – a standalone type that has a single value undefined.

object for more complex data structures.

symbol for unique identifiers.

JavaScript Operators :

JavaScript supports the following types of operators.

Arithmetic Operators

Comparison Operators

Logical (or Relational) Operators

Assignment Operators

Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No. Operator & Description

1. + (Addition)

Adds two operands

Ex: A + B will give 30

2. - (Subtraction)

Subtracts the second operand from the first

Ex: A - B will give -10

3. * (Multiplication)

Multiply both operands

Ex: A * B will give 200

4. / (Division)

Divide the numerator by the denominator

Ex: B / A will give 2

5. % (Modulus)

Outputs the remainder of an integer division

Ex: B % A will give 0

6. ++ (Increment)

Increases an integer value by one

Ex: A++ will give 11

7. -- (Decrement)

Decreases an integer value by one

Ex: A-- will give 9

Comparison Operators

JavaScript supports the following comparison operators −

Assume variable A holds 10 and variable B holds 20, then

1. = = (Equal)

Checks if the value of two operands are equal or not, if yes, then the condition becomes true.

Ex: (A == B) is not true.

2. != (Not Equal)

Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.

Ex: (A != B) is true.

3. > (Greater than)

Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.

Ex: (A > B) is not true.

4. < (Less than)

Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.

Ex: (A < B) is true.

5. >= (Greater than or Equal to)

Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.

Ex: (A >= B) is not true.

6. <= (Less than or Equal to)

Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.

Ex: (A <= B) is true.

Logical Operators

JavaScript supports the following logical operators −

Assume variable A holds 10 and variable B holds 20, then −

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.

Bitwise Operators

JavaScript supports the following bitwise operators −

Assume variable A holds 2 and variable B holds 3, then −

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.

Assignment Operators

JavaScript supports the following assignment operators −

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

Miscellaneous Operator

We will discuss two operators here that are quite useful in JavaScript: the conditional operator (? :) and the typeof operator.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

1.? : (Conditional)

If Condition is true? Then value X : Otherwise value Y

t**ypeof Operator**

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

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

Here is a list of the return values for the typeof Operator.

Type	String Returned by typeof
Number	"number"
String.	"string"
Boolean	"boolean"
Object	"object"
Function	"function"

Undefined "undefined"

Null "object"

JavaScript Variable

A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : l**ocal variable and global variable**.

There are some rules while declaring a JavaScript variable (also known as identifiers).

Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.

After first letter we can use digits (0 to 9), for example value1.

JavaScript variables are case sensitive, for example x and X are different variables.

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>

JavaScript Expressions

Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation. As per the MDN documentation, JavaScript has the following expression categories.

Arithmetic Expressions:

Arithmetic expressions evaluate to a numeric value. Examples include the following

```
10;     // Here 10 is an expression that is evaluated to the numeric value 10 by the JS interpreter

10+13; // This is another expression that is evaluated to produce the numeric value 23
```

String Expressions:

String expressions are expressions that evaluate to a string. Examples include the following

```
'hello';

'hello' + 'world'; // evaluates to the string 'hello world'
```

Logical Expressions:

Expressions that evaluate to the boolean value true or false are considered to be logical expressions. This set of expressions often involve the usage of logical operators && (AND), ||(OR) and !(NOT). Examples include

```
10 > 9;   // evaluates to boolean value true

10 < 20;  // evaluates to boolean value false

true;     //evaluates to boolean value true

a===20 && b===30; // evaluates to true or false based on the values of a and b
```

Primary Expressions:

Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values. Examples include the following

```
'hello world'; // A string literal

23;          // A numeric literal

true;         // Boolean value true

sum;          // Value of variable sum

this;         // A keyword that evaluates to the current object
```

Left-hand-side Expressions:

Also known as lvalues, left-hand-side expressions are those that can appear on the left side of an assignment expression. Examples of left-hand-side expressions include the following

// variables such as i and total

i = 10;

total = 0;

// properties of objects

var obj = {}; // an empty object with no properties

obj.x = 10; // an assignment expression

// elements of arrays

array[0] = 20;

array[1] = 'hello';

// Invalid left-hand-side errors

++(a+1); // SyntaxError. Attempting to increment or decrement an expression that is not an lvalue will lead to errors.

Now that we have covered the basics of expressions, let's dive a bit deeper into expressions.

Assignment Expressions:

When expressions use the = operator to assign a value to a variable, it is called an assignment expression. Examples include

average = 55;

var b = (a = 1); // here the assignment expression (a = 1) evaluates to a value that is assigned to the variable b. b = (a = 1) is another assignment expression. var is not part of the expression.

The = operator expects an lvalue as its left-side operand. The value of an assignment expression is the value of the right-side operand such as 55 in the above example. As a side effect, the = operator assigns the value on the right side to the value on the left side.

Expressions with side effects:

As we just saw with assignment expressions, expressions with side effects are those that result in a change or a side effect such as setting or modifying the value of a variable through the assignment operator =, function call, incrementing or decrementing the value of a variable.

sum = 20; // here sum is assigned the value of 20

sum++; // increments the value of sum by 1

```
function modify(){
  a *= 10;
}
var a = 10;
modify(); // modifies the value of a to 100.
```

JavaScript Statements

A statement is an instruction to perform a specific action. Such actions include creating a variable or a function, looping through an array of elements, evaluating code based on a specific condition etc. JavaScript programs are actually a sequence of statements.

Statements in JavaScript can be classified into the following categories

Declaration Statements:

Such type of statements create variables and functions by using the var and function statements respectively. Examples include

```
var sum;
var average;
// In the following example, var total is the statement and total = 0 is an assignment expression
var total = 0;
// A function declaration statement
function greet(message) {
  console.log(message);
}
```

Expression Statements:

Wherever JavaScript expects a statement, you can also write an expression. Such statements are referred to as expression statements. But the reverse does not hold. You cannot use a statement in the place of an expression.

```
var a = var b; // leads to an error cause you cannot use a statement in the place of an expression
var a = (b = 1); // since (b = 1) is an assignment expression and not a statement, this is a perfectly acceptable line of code
console.log(var a); // results in error as you can pass only expressions as a function argument
```

Stand alone primary expressions such as variable values can also pass off as statements depending on the context. Examples of expression statements includes the following

// In the following example, sum is an expression as it evaluates to the value held by sum but it can also pass off as a valid statement.

sum;

// An expression statement that evaluates an expression with side effects

b = 4+38;

Conditional Statements:

Conditional statements execute statements based on the value of an expression. Examples of conditional statements includes the if..else and switch statements.

// Syntax of an if statement. If the expression following the if statement evaluates to a truthy value, statement 1 is executed else statement 2 is executed.

```
if (expression)
   statement 1
else
   statement 2
```

Loops and Jumps :

Looping statements includes the following statements: while, do/while, for and for/in. Jump statements are used to make the JavaScript interpreter jump to a specific location within the program. Examples of jump statements includes break, continue, return and throw.

Defining functions in JavaScript :

Function declarations

A function definition (also called a function declaration, or function statement) consists of the function keyword, followed by:

The name of the function.

A list of parameters to the function, enclosed in parentheses and separated by commas.

The JavaScript statements that define the function, enclosed in curly brackets, {...}.

For example, the following code defines a simple function named square:

```
function square(number) {
  return number * number;
```

}

The function square takes one parameter, called number. The function consists of one statement that says to return the parameter of the function (that is, number) multiplied by itself. The statement return specifies the value returned by the function:

return number * number;

Primitive parameters (such as a number) are passed to functions by value; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.

Objects in JavaScript

In JavaScript, almost "everything" is an object.

Booleans can be objects (if defined with the new keyword)

Numbers can be objects (if defined with the new keyword)

Strings can be objects (if defined with the new keyword)

Dates are always objects

Maths are always objects

Regular expressions are always objects

Arrays are always objects

Functions are always objects

Objects are always objects

All JavaScript values, except primitives, are objects

Arrays in JavaScript

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

var car1 = "Saab";

var car2 = "Volvo";

var car3 = "BMW";

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

var array_name = [item1, item2, ...];

Example

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

Dates in JavaScript :

Creating Date Objects

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

new Date()

new Date(year, month, day, hours, minutes, seconds, milliseconds)

new Date(milliseconds)

new Date(date string)

new Date()

new Date() creates a new date object with the current date and time:

Example

var d = new Date();

new Date(year, month, ...)

new Date(year, month, ...) creates a new date object with a specified date and time.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

var d = new Date(2018, 11, 24, 10, 33, 30, 0);

new Date(dateString)

new Date(dateString) creates a new date object from a date string:

Example

var d = new Date("October 13, 2014 11:13:00");

new Date(milliseconds)

new Date(milliseconds) creates a new date object as zero time plus milliseconds:

Example

```
var d = new Date(0);
```

Math object JavaScript :

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Example

```
Math.PI;          // returns 3.141592653589793
```

Math.round()

Math.round(x) returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);    // returns 5

Math.round(4.4);    // returns 4
```

Math.pow()

Math.pow(x, y) returns the value of x to the power of y:

Example

```
Math.pow(8, 2);      // returns 64
```

Math.sqrt()

Math.sqrt(x) returns the square root of x:

Example

```
Math.sqrt(64);      // returns 8
```

Math.abs()

Math.abs(x) returns the absolute (positive) value of x:

Example

```
Math.abs(-4.7);     // returns 4.7
```

Math.ceil()

Math.ceil(x) returns the value of x rounded up to its nearest integer:

Example

```
Math.ceil(4.4);     // returns 5
```

Math.min() and Math.max()

Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:

Example

```
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
```

Math.sin()

Math.sin(x) returns the sine (a value between -1 and 1) of the angle x (given in radians).

If you want to use degrees instead of radians, you have to convert degrees to radians:

Angle in radians = Angle in degrees x PI / 180.

Example

```
Math.sin(90 * Math.PI / 180);     // returns 1 (the sine of 90 degrees
```

Math.cos()

Math.cos(x) returns the cosine (a value between -1 and 1) of the angle x (given in radians).

If you want to use degrees instead of radians, you have to convert degrees to radians:

Angle in radians = Angle in degrees x PI / 180.

Example

```
Math.cos(0 * Math.PI / 180);     // returns 1 (the cos of 0 degrees)
```

Documents object model :

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 Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:

The HTML DOM Tree of Objects

DOM HTML tree

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

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

Event Handling :

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:

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

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>
</body>
</html>
```

In this example, a function is called from the event handler:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="changeText(this)">Click on this text!</h1>
<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>
</body>
</html>
```

HTML | DOM Window frames Properties

The Window frames property in HTML DOM is used to return the frame element in the form of an array object. This property represents all <iframe> elements in the current window. DOM Windowframe is a read-only property.

Syntax:

window.frames

Return Value: It returns a reference to the Window object, representing all the frames in the current window.

Properties:

length property: It returns the number of iframe elements in the current window.

Syntax:

window.length

Example:

```
<!DOCTYPE html>
<html>
<head>
   <title>
      HTML | DOM Window frames Property
   </title>
</head>
<body>
   <iframe src = "https://ide.geeksforgeeks.org/tryit.php">
   </iframe><br>
    <p>
      Click on the button to display    the number of iframes
   </p>
   <button onclick="myGeeks()">
      Click Here!
   </button>
   <p id = "GFG"></p>
   <!-- script to count iframes -->
   <script>
      function myGeeks() {
```

```
        var x = window.length;
        document.getElementById("GFG").innerHTML = x;
    }
  </script>
</body>
</html>
```

Form validation :

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()"
method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```