UNIT-1 Basics of JavaScript

JavaScript basics – Syntax, Data types, Variables, Operators, User interaction through dialog boxes, Built-in functions, Flow control statements: Decision making and Looping, Arrays, User-defined functions

USING JAVASCRIPT ON A WEB PAGE:

JavaScript syntax is embedded into an HTML file. A browser reads HTML file and interprets the tags. Since all Javascript need to be included as an integral part of an HTML document when required, the browser needs to be informed that specific sections of HTML code is JavaScript. The browser will then use its built-in Javascript engine to interpret this code.

The browser is given this information using the HTML tags <script>.....</script>. This tag marks the beginning and ending of a snippet of scripting code. It has an optional attribute **language** which indicates the scripting language used for writing the snippet of scripting code. If left undefined, Netscape communicator assumes JavaScript and Internet Explorer assumes VB script. It also includes a MIME type attribute **type** which defines the application and language of the script so written.

SYNTAX:

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

The Scripts can be written in <head> and <body> or both.

In <head>:

The <head> is an ideal place because the head of an HTML document is always processed before the body. Placing Javascript memory variables, constants and user-defined Javascript function in this section of an HTML document will cause them to be defined by the Javascript interpreter before being used. This is important because any attempt to use a variable or any other JavaScript object before it is defined, results in an error.

```
<html>
<head>
<title>some title here</title>
<script language="javascript" type="text/javascript">
:
</script>
</head>
<body>
:
</body>
</html>
```

In <body>:

The scripts that needs to be run as the page loads so that the script generates some dynamic content in the page must go in the <body>.

```
<html>
<head>
</head>
<body>
:
<script language="javascript" type="text/javascript">
:
</script>
:
```

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

Any number of <script> tag set can be used in a document. Generally those scripts which are event driven are written in the <head>. Also there may be <script> tags both in the <head> and <body> at the same time.

DATATYPES:

JavaScript is a weakly-typed language and a lot forgiving about how you used different types of data. It does not allow the datatype of the variable to be declared when a variable is created. The same variable may be used to hold different types of data at different times when the JavaScript code runs. When you are using different types of data at the same time, JavaScript will work out behind the scenes what it is you are trying to do. There are 4 primitive (basic) types of data (Number, Boolean, String, Null) and complex types (Arrays, Objects). Primitive types can be assigned a single literal value such as Number, String or Boolean value. Literals are fixed values which literally provide a fixed value in a program.

The Primitive datatypes that Javascript supports are:

1. **Number:**
   It consists of Integer and Floating point numbers and the special value **NaN** (**N**ot **a** **N**umber). These numbers can be positive or negative and can span a very wide range i.e. -2^{53} to $+2^{53}$ for integers.
   e.g. a=10;
   b=123.45;

2. **Boolean:**
   It consists of logical values (truth values) **true** and **false** and are generally used in condition building.
   e.g. a=true;
3. **String:**
   It consists of string values that are enclosed in single or double quotes. If a string begins with a double quote, it must end with a double quote.
   e.g. a=”Fybca”;
   b=’first’;

VARIABLES:

Data can be stored either permanently or temporarily. The variables are created to hold any type of data. They are held in the computer’s memory temporarily and have a limited lifetime. In order to use a variable, it may or may not be declared. Declaring a variable tells Javascript that a variable of a given name exists so that the interpreter can understand references to that variable name throughout the rest of the script. Although it is possible to declare variables by simply using them, declaring variables helps to ensure that programs are well organized and helps keep track of the scope of the variables. When the web page is cleared or the window closed, these variables are discarded.

The following rules apply when creating variables:

1. A variable must be 31 characters long. (New versions support 255 length)
   e.g. var Employee_registered_name VALID (24 character long name)
2. The variable names are case sensitive.
   e.g var a,A; VALID (both are different variables)
3. Reserved names cannot be used as variable names. Names that cannot be used are called reserved words, which Javascript keeps for its own use e.g if, for, var
   e.g var for=0; INVALID
4. A variable name cannot contain spaces.
   e.g var ab cd; INVALID
5. No punctuation or special symbols except (_) underscore is allowed.

e.g var ab+cd; INVALID

6. First character must be a letter.
   e.g var 123abc; INVALID

Declaring variables:
Variables are declared using the **var** keyword. This informs the computer that it needs to reserve some memory for your data to be stored in later.

SYNTAX: var some_var1, some_var2,..., some_varN;
e.g. var rollno;
var rollno_marks;

Assigning values to variables:
The variables may be assigned values using the (=) assignment operator.

SYNTAX: some_var1=some_value;
e.g var rollno;
rollno=12;

Initializing variables:
The variables may be assigned values when they are declared using **var**, called as initializing variables.

SYNTAX: var some_var1=some_value;
e.g var rollno=12;

LITERALS:

Literals are the fixed or constant values used in the program which do not change.

String Literal example: a=”FYBCA”;
A=’FYBCA’;
Number Literal example: a=123.45;
A=123;

TYPE CASTING:

In JavaScript, variables are loosely cast. i.e. the type of a variable is implicitly defined based on the values that are assigned to it from time to time.
e.g. Combining the string literal “Total amount is “ with the integer literal 1000 results in a string with the value “Total amount is 1000”.
e.g. Adding a numeric literal 10.5 with the string literal “20” results in a floating-point numeric literal 30.5. This process is called **type casting**.

OPERATORS:

An operator is used to transform one or more values into a single resultant value. The values to which the operator is applied is referred to as **operands**. A combination of an operator and its operands is referred to as an **expression**. Expressions are evaluated to determine the value of the expression. For some operators, the order of the operands matters. An operator which requires one operand is called **Unary** operator. An operator which requires two operands is called **Binary** operator. An operator which requires three operands is called **Ternary** operator.

SYNTAX: Operand1 Operator Operand2 e.g. (a + b)

1. **Arithmetic operators:**
   These operators are used in math calculations.

Operator	Type	Name	Example	Output
+	Unary	Positive sign of a number	a=+5;	5
	Binary	Addition	a=10; b=a+1;	11
-	Unary	Negative sign of a number	a=-5;	-5

	Binary	Subtraction	a=10; b=a-1;	9
*	Binary	Multiplication	a=5; b=a*10;	50
/	Binary	Division	a=10; b=a/2;	5
%	Binary	Modulus	a=10; b=a%3;	1

+ → Adds the value of the two operands
- → Subtracts the value of the operand on the right from the operand on the left
* → Multiplies the values of the two operands
/ → Divides the left operand's value with that of the right operand's value
% → Divides the left operand's value with that of the right operand's value and returns the remainder

2. Increment / Decrement operators:

These operators are used to increase or decrease the value of a variable by 1. They can be used in two different ways: Before the operand or after the operand.

Operator	Type	Name	Example	Output

++	Unary	Increment
		(Pre-increment)
		
		(Post-increment)
			a=10;
			b=++a;
			
			b=a++;
				11
				
				10

--	Unary	Decrement
		(Pre-decrement)
		
		(Post-decrement)
			a=10;
			b=--a;
			
			b=a--;
				9
				
				10

If **++a** then it increments **a** by one and returns the result, while **a++** returns **a** and then increments the value of **a** by one. Similarly, **--a** decreases the value of **a** by one and returns the result, while **a—** returns **a** and then decrements the value of **a** by one.

3. Assignment operator:

The assignment operator is used to update the value of a variable. Some assignment operators are combined with other operators to perform a computation on the value contained in a variable and then update the variable with the new value.

Operator	Type	Name	Example	Output
=	Binary	Assignment	a=10; b=a;	10

= → Sets the variable on the left of the operator to the value of the expression on its right.

4. Relational / Comparison operators:

They are used to compare two operands. They result in the Boolean values **true** or **false**.

Operator	Type	Name	Example	Output
==	Binary	Equality	a=10; b=20; if (a==b)	false
!=	Binary	Not Equal	a=10; b=20; if (a!=b)	true
<	Binary	Less than	a=10; b=20; if (a<b)	true
>	Binary	Greater than	a=10; b=20; if (a>b)	false
<=	Binary	Less than or equal to	a=10; b=20; if (a<=b)	true
>=	Binary	Greater than or equal	a=10; b=20;	

		to	if (a>=b)	false

== → Checks whether the values on both sides are same or not.
!= → Checks whether the values on both sides are not same or not.
< → Checks whether the value on the left is less than the value on the right side or not.
> → Checks whether the value on the left is greater than the value on the right side or not.
<= → Checks whether value on the left is less than or equal to value on right side or not.
>= → Checks whether value on the left is greater than or equal to value on right side or not.

5. Logical operators:

These operators are used to connect to conditions and form a single condition.

Operator	Type	Name	Example	Output
&&	Binary	Logical AND	a=10; b=20; c=30; if((a>=b)&&(a<c))	false
||	Binary	Logical OR	a=10; b=20; c=30; if ((a>=b)||(a<c))	true
!	Binary	Logical NOT	a=10; b=20; c=30; if (!(a>=b))	true

&& → It is used to compare the truth values on both sides and return a true if both are true otherwise a false.
|| → It is used to compare the truth values on both sides and return a true if either one is true.
! → It is used to negate the truth value of the resultant truth value.

6. String Concatenation operator:

It is used to perform operation on the strings. This operator is used to join two strings.

Operator	Type	Name	Example	Output
+	Binary	String concatenation	a=”In”; b=”dia”; c=a+b;	India

Note: If the operands are Number and Number string, then is will perform addition of the numbers.

USER INTERACTION THROUGH DIALOG BOXES:

Javascript provides the ability to take user input or display small amounts of text to the user by using dialog boxes. These dialog boxes appear as separate windows and their content depends on the information provided by the user. This content is independent of the text in the HTML page containing the Javascript code and does not affect the content of the page in any way. All the three types of dialog boxes are **modal** type which means that unless you discard the dialog box by clicking the button, the script and HTML in the page will not be executed further.

1. Alert dialog box:

This method takes a string as an argument and displays an alert dialog box in the browser window when invoked. The alert dialog box displays the string passed to the method, as well as an **OK** button. The Javascript and the HTML program, in which this code snippet is held, will not continue processing until the **OK** button is clicked. It requires only one parameter, i.e. the string/value to be displayed.

SYNTAX:

```
alert(“your message here”);
OR  window.alert(“your message here”);
```

Example:

```
var a=”Hello”, b=100;
alert(a);
alert(b);
```

OUTPUT:Hello 100

2. Prompt dialog box:

This method, in addition to displaying a specified message, also provides a single data entry field, which accepts user input. It also displays a **OK** and **CANCEL** button. The prompt dialog box also causes program execution to halt until user action takes place. If the **OK** button is clicked then whatever is entered in the text box is returned. If the **CANCEL** button is clicked then **NULL** value is returned. It requires two parameters, where the first indicates the message to be displayed and the second indicates the default value that will be displayed in the textbox (this is optional).

SYNTAX:

some_var=prompt("your message here", "some default value");

OR some_var=window.prompt("your message here", "default value");

Example:

```
var a;
a=prompt("Enter your name", " ");
alert(a);                    OUTPUT: Mahatma Gandhi
```

BUILT-IN FUNCTIONS:

Functions are blocks of Javascript code that perform a specific task and often return a value. A function can take zero or more parameters. Javascript provides several built-in functions that can be used to perform explicit type conversion.

1. parseInt()

This function is used to parse a string value to an integer. It returns the first integer contained in a string or NaN if the string doesn't begin with a digit. It actually parses each character of the string to see if it's a valid digit, if not then it stops converting and returns the number it has converted so far.

SYNTAX:

```
parseInt(var)
```

Example:

```
var a=123, b=123.45, c="123.45", d="123abc", e="abc123";
alert(parseInt(a));
alert(parseInt(b));
alert(parseInt(c));
alert(parseInt(d));
alert(parseInt(e));      OUTPUT: 123  123  123  123   NaN
```

2. parseFloat()

This function returns the first floating point number contained in a string, or NaN if the string does not begin with a valid floating point number. While converting, it considers the decimal point and the fractional part also as a part of the allowable character. When it sees an invalid character, it stops and returns the number converted so far.

SYNTAX:

```
parseFloat(var)
```

Example:

```
var a=123, b=123.45, c="123.45", d="123.45abc", e="abc123";
alert(parseFloat(a));
alert(parseFloat(b));
alert(parseFloat(c));
alert(parseFloat(d));
alert(parseFloat(e));      OUTPUT: 123  123.45  123.45  123.45   NaN
```

3. Number()

This function converts the whole string value into a numeric value. This function is used when your script is not concerned with the precision of the value and prefers to let the source string govern whether the returned value is a floating-point number or an integer. Here it is expected that the string contains only allowable number characters. If there is a non-numeric character in the string then it returns NaN.

SYNTAX:

Number(var)

Example:

```
var a="123", b="123.45", c="123a";
alert(Number(a));
alert(Number(b));
alert(Number(c));          OUTPUT: 123   123.45   NaN
```

4. isNaN()

When you design an application that requests user input or retrieves data from a server-side database, you cannot be guaranteed that a value you need is numeric, or can be converted to a number. So you need to see if the value is a number before performing some math operations on it. Also for those instances in which a calculation relies on data coming from a text field or other string-oriented source, you frequently need to check whether the value is a number. If the value is not a number, the calculation may result in a script error. Some strings are not convertible to numbers. When parseInt, parseFloat or Number function is used on such a string, it doesn't return a number, but returns a special value - **NaN**. To check whether the returned value is a **NaN**, you must use **isNaN()** function. It returns a true if the value is NaN and returns a false it the value is a Number.

SYNTAX:

isNaN(var)

Example:

```
var a="Hello", b;
b=Number(a);
if (isNaN(b))
        alert("Variable is not a number");
                        OUTPUT: Variable is not a number
```

FLOW CONTROL STATEMENTS:

These statements control the flow of execution of the lines of code. Depending on a certain condition, it may deviate from the regular flow and execute additional statements also.

A. DECISION MAKING STATEMENTS:

The conditional construct in Javascript offers a simple way to make a decision in a Javascript program. It returns either **true** or **false** depending upon how the condition is evaluated.

if Statement:

If the condition evaluation returns **true**, the block is executed, if the evaluation returns **false** then this section is skipped. The curly braces { } marks out a block of code and is treated as a single piece of code. This entire block will be executed if the condition is **true.** The code outside the block will always be executed. This structure is useful when a code is to be executed if a certain condition is **true**. It does not do anything if the condition is **false**.

SYNTAX:

```
if (condition)
{           // code to be executed          }
```

Example:

```
var a=10;
if (a>0)
{     a=a+10;          }
alert(a);                                  OUTPUT:   20
```

if–else Statement:

A situation where you want some code to execute if a certain condition is **true** and some other code to execute if it is **false**, can be solved by writing two independent if-statements. The other way to solve this is using the **else** part. This structure is useful when there are only two alternatives to choose from.

SYNTAX:

```
if (condition)
{
        //code1 to be executed if condition is true
}
else
{
        //code2 to be executed if condition is false
}
```

Example:

```
var a=10;
if (a>0)
        alert("Positive no");
else
        alert("Negative no");
```

OUTPUT: Positive no

<u>else-if Statement:</u>

There are situations where a variable/expression needs to be evaluated against multiple values/conditions. Depending on which condition is satisfied, appropriate code will be executed. This is possible using the else-if ladder. If none of the conditions are correct, then some code can be executed by using the **else** part, which is optional.

SYNTAX:

```
if (condition1)
{
        //code to be executed if this condition is true
}
else  if (condition2)
{
        //code to be executed if this condition is true
}
else  if...
:
else
{
        //code to be executed if none of the above conditions is true
}
```

Example:

```
var a=2;
if (a==1)
        alert("One");
else if (a==2)
        alert("Two");
else if (a==3)
        alert("Three");
else
        alert("No Prize");
```

OUTPUT: Two

<u>switch Statement:</u>

When you want to check the value of a particular variable for a large number of possible discrete values, there is a more efficient alternative, the switch-statement. The switch has four important elements: The test expression, the case statements, the break statements and the default statement.

The **test expression** is what is to be tested. The **case statements** do the condition checking. Each case statement specifies a value. If only the block under a particular case must be executed and not all the other blocks below it, then the **break statement** must be used to stop

the execution of the remaining cases and exit the switch code. The **default statement** is optional and will be executed only if none of the above case statements are evaluated as **true**. The value specified in the case must be followed by a colon sign and must match the type of variable being tested.

SYNTAX:

```
switch (var/expression)
{
        case value1:
                //code1 to be executed if var==value1
                        break;
        :
        case valueN:
                //codeN to be executed if var==valueN
                        break;
        default:
                //code to be executed if var is not value1 nor ... valueN
}
```

Example:

```
var a=2;
switch (a)
{
        case 1:         alert("One");
                        break;
        case 2:         alert("Two");
                        break;
        case 3:         alert("Three");
                        break;
        default:        alert("Incorrect input");
}                                                   OUTPUT: Two
```

B. LOOPING STATEMENTS:

Looping refers to the ability of a block of code to repeat itself when the condition is **true**. This repetition can be for a predefined number of times (for) or it can go until certain conditions are met (while). There are two looping structures: **for** and **while**. Both are entry-controlled loops, meaning that the loop is entered if and only if the condition is **true** in the beginning itself.

for Looping statement:

When you know how many times the code must be executed, then for-Loop is used. This statement repeats a block of code a certain number of times. There are three sections in the for-Loop header, each separated by a semicolon(**;**), the **expression1** is used to set up a counter variable and assign the initial value (initialization), **condition** specifies the final value for the loop to stop at, **expression2** specifies how the initial value in expression1 will be incremented. All the three sections are optional.

To keep track of how many times you have looped through the code, you need a variable to keep count. This counting variable is initialized in the initialization part. Following the semicolon, you have the test condition part. The code inside the for-statement will keep executing for as long as this test condition evaluates to **true**. After the code is looped through each time, this condition is tested. The number of times a loop is performed is often called the **number of iterations**. After the second semicolon, you have the final increment part of the loop, where variables in the loop's test condition have their values incremented. This part is executed with every loop of the code. The block of code is contained within curly braces. If the condition is never true, even at the first test of the loop condition, then the code inside the for-loop will be skipped and never executed.

SYNTAX:

```
for (expression1 ; condition ; expression2)
{
```

```
        // code block to be repeated
    }
```

Example:

```
    var i,a=1;
    for (i=1; i<=5; i++)
    {
        a=a+1;
    }
    alert(a);
```

OUTPUT: 6

while Looping Statement

JavaScript while Loop check condition and execute while block for a specify number of times, until while condition become false. In JavaScript while loop check first condition. If condition becomes true then execute block of code.

Syntax:

```
while (condition)
{
    statements;          // Do stuff if while true
    increment;           // Update condition variable value
}
```

Example:

```
<script>
    number = 1;
    while (number <= 10)
    {
        alert(number + " times");
        number++;
    }
</script>
```

Output: 1 times 2 times 3 times 4 times 5 times 6 times 7 times 8 times 9 times 10 times

Break Statement

JavaScript break is special statement that can be used inside the loops, JavaScript break keyword terminates the current loop and execution control transfer after the end of loop.

Syntax: break;

Example:

```
<script>
    var i = 0;
    while(i <= 10)
    {
        if (i == 5)
        {
            alert("terminated, i = 5");
            break;
        }
        alert(i);
        i++;
    }
</script>
```

Output: 0 1 2 3 4 terminated, i = 5

Continue Statement

JavaScript continue statement used inside the loops, continue statement skip the current iteration and execute to the next iterations of the loop.

Syntax: continue;
Example:

```
<script>
	var i = 0;
	while(i < 10)
	{
		i++;
		if (i == 5)
		{
			alert("skipped, i = 5");
			continue;
		}
		alert(i);
	}
</script>
```

Output: 1 2 3 4 skipped, i = 5 6 7 8 9 10

ARRAY:

Arrays are Javascript objects that are capable of storing a sequence of values. These values are stored in indexed locations within the array. The length of an array is the number of elements that an array contains. The individual elements of an array are accessed by using the name of the array followed by the index value of the array element enclosed in square brackets. The array element index starts with 0. Hence the last array element index number is one less than that of the length of the array. Javascript automatically extends the length of any array when new array elements are initialized. An array can store any kind of data in any element. This means that all the elements can be of different datatypes.

Declare an array:

An array must be declared before it is used using the following techniques:

var array_name = new Array(size);
var array_name = new Array();

where array_name = name of the array variable
size = number of elements to be reserved space for.

The first method will create an array with length **size**. The second array will be created with 0 size. A zero-length array or fixed length array is extended by referencing elements outside the current size of the array. The browser will automatically change the length to incorporate the new element.

e.g
```
var roll=new Array(5);
roll[10]=50;
```

Here, the array was created with total 5 elements. But, with the second statement, we are trying to insert 50 at index 10 i.e. 11th position. So automatically the array size will become 11.

Initialize / Dense array:

A dense array is an array that has been created with each of its elements being assigned a specific value. They are declared and initialized at the same time.

var array_name = new Array(val0, val1, ..., valn);

In this array, since the element count starts from 0 to n, the array length is n+1;

e.g. var a = new Array(10, 10.5, “India”);

This will create an Array **a** with the values 10, 10.5 and India in a[0], a[1] and a[2].

Assign values to an array:

An array can be assigned values using the = operator. Each element must be referenced using its index value.

array_name[index] = value;

e.g.
```
var a = new Array(3);
a[0] = 10;
```

```
a[1] = 10.50;
a[2] = “India”;
```
This will assign 10 to the 1st element with index 0, 10.5 to the second element with the index 1 and India to the third element with the index 2.

USER-DEFINED FUNCTION

Functions group together program code that performs a specific task into a single unit that can be used repeatedly whenever required in a Javascript program. A user defined function first needs to be declared and coded. Once this is done the function can be invoked by calling it using the name given to the function when it is declared. Functions can accept information in the form of arguments and can return a result. Functions come in two forms: Built-in functions like parseInt() and user-defined functions like abc(). These user-defined functions can be written anywhere in the HTML file i.e. in the HEAD or in the BODY but within <script> and </script>

Function Definition

Functions are declared using the **function** keyword. A function definition consists of a function name, list of parameters/arguments that will accept values passed to the function when called and a block of Javascript code that defines what the function does. The function name follows the same rules as of variable creation like it starts with a letter and is case-sensitive. The list of parameters passed to the function appears in parentheses and is separated using comma.

SYNTAX:

```
function function_name(parameter1, parameter2, ..., parameterN)
{
        //block of JavaScript code here
}
```

Example:

```
function abc(a, b)
{               alert(a+b);                     }
```

Function Body

The body of the function is written between curly parentheses. This encloses a block of code which performs a certain action and may return a value.

Parameter Passing

Values can be passed to function parameters when a parameterized function is called. Values are passed to the function by listing them in the parentheses following the function name. Multiple values can be passed, separated by commas provided that the function has been coded to accept multiple parameters. Parameters of the function and the variables declared inside the functions are considered local variables and cannot be accessed outside that function. The parameter values can be variables or literals or mix. If the function does not accept or need parameters, then the parentheses will be left empty.

SYNTAX:

```
function_name(value1,value2,...,valueN);
```

Example:

```
function abc(a, b)
{       alert(a+b);     }
var x=5, y=10;
abc(x, y);
```

Function Call

The function created can be called be using the function name, followed by the parameter list, if defined. If the function has a return statement then a receiving variable must be defined. If the function does not require parameters then the function call with empty parentheses is used.

SYNTAX:

```
function_name(parameters);
```

OR

```
some_var=function_name(parameters);
```

Example:

```
function abc(a, b)
{       alert(a+b);    }
function xyz(a, b)
{       return (a-b);  }
var x=5, y=10, z;
abc(x, y);
z=xyz(a, b);
alert(z);
```

Returning a Value

The functions can return a value. For returning a value, the **return** statement must be used. The return value must be a single value which can be of any type.

SYNTAX:

```
return variable/expression;
```

Example:

```
function abc(a,b)
{
        var x=a+b;
        return x;      }
var x=5, y=10, z;
z=abc(x, y);
alert(z);
```