JavaScript use cases in the application development

1) JavaScript Can Change HTML Content

2) JavaScript Can Change HTML Attribute Values

3) JavaScript Can Change HTML Styles

4) JavaScript Can show/Hide HTML Elements

5) JavaScript can make the calls to the server by using the AJAX.

Writing the JavaScript in the webpage.

1) writing the script tag

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

        js logic

</script>
```

2) loading the external js file using the src attribute. in the src attribute provide the file path for the external js file.

```
<script type="text/javascript" src="project/js/app.js">
```

Script tag can be placed in the <body>, or in the <head> section of an HTML page, or in both.

A JavaScript program is a list of programming statements.

JavaScript statements are composed of Values, Operators, Expressions, Keywords, and Comments.

JavaScript values are of the following types.

1) Integers literals.

in this category we have decimal numbers, floating point numbers.

both positive and negetive values are supported.

2) String literals

in this category we have string of characters.

it can be created by using the single and double quotes.

3) Boolean literals

The Boolean type has two literal values:

true and false.

4) null and underfined

Variables

variables are used to store data values. JavaScript uses the var keyword to declare variables. An equal sign is used to assign values to variables.

variable naming rules: -

>Names can contain letters, digits, underscores, and dollar signs.
>Names must begin with a letter
>Names can also begin with $ and _ (but we will not use it in this tutorial)
>Names are case sensitive (y and Y are different variables)
>Reserved words (like JavaScript keywords) cannot be used as names

```
var i = 100;
```

You can declare many variables in one statement.

```
var i = 10, j = 20, k = 30;
```

Data Type

JS supports the two types of data types.

1) primitive - it represents the value.

String - A series of characters enclosed in quotation marks. A string must be delimited by quotation marks of the same type, either single quotation marks (') or double quotation marks (").
Numbers - Any numeric value. The numbers can be either positive or negative.
Boolean - A logical true or false. Use to evaluate whether a condition is true or false.
null - A special keyword denoting a null value (i.e. empty value or nothing). Since JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant.
undefined - A top-level property whose value is undefined, undefined is also a primitive value.

2) reference - it represents the memory location

Arrays - it used to represents the group values and refered by single variable.

JavaScript arrays are written with square brackets.

Array items are separated by commas.

Objects - it used to represents the key and value pairs.

Data Type Conversion: -

JavaScript is a dynamically typed language. Therefore there is no need to specify the data type of a variable at the time of declaring it. Data types are converted automatically as needed during script execution.

Functions: -

functions are used to define the code once, and use it many times.

JavaScript functions are defined with the function keyword.

we can define the functions in two ways.

1) function declaration

2) function expression

Function Declartaion: -

```
function functionName(parameters) {
  code to be executed
}
```

Function Expressions: -

A JavaScript function can also be defined using an expression.

```
var sum = function (a, b) {

                    return a + b
            };
```

After a function expression has been stored in a variable, the variable can be used as a function:

```
var result = sum(4, 3);
```

The function above is actually an anonymous function.

Function Hoisting: -

moving the declaration to the starting of the program is called hoisting.

Hoisting applies to variable declarations and to function declarations.

Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope.

Function Parameters: -

JavaScript function definitions do not specify data types for parameters.

JavaScript functions do not perform type checking on the passed arguments.

JavaScript functions do not check the number of arguments received.

JavaScript functions have a built-in object called the arguments object. it will be intialized by the js engine.

Operator: -

Operator is used to represents operation.

JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.

Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (division remainder)
++	Increment
--	Decrement

Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

=	x = y
+=	x += y
-=	x -= y
*=	x *= y
/=	x /= y
%=	x %= y

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Conditional (Ternary) Operator

The conditional operator assigns a value to a variable based on a condition.

```
variablename = (condition) ? value1:value2
```

Logical Operators

&&	logical and
||	logical or
!	logical not

Type Operators

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

Bitwise Operators

&	AND
|	OR
~	NOT
^	XOR
<<	Zero fill left shift
>>	Signed right shift
>>>	Zero fill right shift

Scope: -

In JavaScript there are two types of scope:

1) Local scope
2) Global scope

Local JavaScript Variables: -

Variables declared within a JavaScript function, become LOCAL to the function.

Local variables have local scope: They can only be accessed within the function.

```
function m1(){

        var i = 10;

        // code.

}
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Global JavaScript Variables: -

A variable declared outside a function, becomes GLOBAL.

A global variable has global scope: All scripts and functions on a web page can access it.

```
var i = 20;

// code here can use i

function m1() {

        // code here can use i

}

function m2(){

        // code here can use i

}
```

String: -

String represents the group of characters. it can be created by using single quotes or double quotes.

properties

length	length of a string

methods

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 new joined strings
endsWith()	Checks whether a string ends with specified string/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
match()	Searches a string for a match against a regular expression, and returns the matches
repeat()	Returns a new string with a specified number of copies of an existing string
replace()	Searches a string for a specified value and returns a new string where the specified values are replaced
search()	Searches a string for a specified value, or regular expression, 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
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
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

Number: -

JavaScript has only one type of number. Numbers can be written with or without decimals.

Number properties.

MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

Number Functions.

isFinite()	Checks whether a value is a finite number
isInteger()	Checks whether a value is an integer
isNaN()	Checks whether a value is Number.NaN`
toString()	Converts a number to a string
valueOf()	Returns the primitive value of a number

Booleans: -

JavaScript booleans can have one of two values: true or false.

You can use the Boolean() function to find out if an expression is true.

toString()	Converts a boolean value to a string, and returns the result
valueOf()	Returns the primitive value of a boolean

Array: -

The Array object is used to store multiple values in a single variable.

Array indexes are zero-based: The first element in the array is 0, the second is 1, and so on.

length Sets or returns the number of elements in an array.

concat()	Joins two or more arrays, and returns a copy of the joined arrays
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)

reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

Date

Date Represents the date data in the javascript.

getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
getTime()	Returns the number of milliseconds since midnight Jan 1 1970, and a specified date
getTimezoneOffset()	Returns the time difference between UTC time and local time, in minutes
getUTCDate()	Returns the day of the month, according to universal time (from 1-31)
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time
getUTCHours()	Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)
now()	Returns the number of milliseconds since midnight Jan 1, 1970
parse()	Parses a date string and returns the number of milliseconds since January 1, 1970
setDate()	Sets the day of the month of a date object
setFullYear()	Sets the year of a date object
setHours()	Sets the hour of a date object
setMilliseconds()	Sets the milliseconds of a date object
setMinutes()	Set the minutes of a date object

setMonth()	Sets the month of a date object
setSeconds()	Sets the seconds of a date object
setTime()	Sets a date to a specified number of milliseconds after/before January 1, 1970
setUTCDate()	Sets the day of the month of a date object, according to universal time
setUTCFullYear()	Sets the year of a date object, according to universal time
setUTCHours()	Sets the hour of a date object, according to universal time
setUTCMilliseconds()	Sets the milliseconds of a date object, according to universal time
setUTCMinutes()	Set the minutes of a date object, according to universal time
setUTCMonth()	Sets the month of a date object, according to universal time
setUTCSeconds()	Set the seconds of a date object, according to universal time
toDateString()	Converts the date portion of a Date object into a readable string
toISOString()	Returns the date as a string, using the ISO standard
toJSON()	Returns the date as a string, formatted as a JSON date
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string
toTimeString()	Converts the time portion of a Date object to a string
toUTCString()	Converts a Date object to a string, according to universal time
valueOf()	Returns the primitive value of a Date object

JavaScript Dom Manipulations

JavaScript can access and change all the elements of an HTML document.

1. change all the HTML elements in the page
2. change all the HTML attributes in the page
3. change all the CSS styles in the page
4. remove existing HTML elements and attributes
5. add new HTML elements and attributes
6. react to all existing HTML events in the page

To identify the element we use the following methods.

Finding HTML elements by id
Finding HTML elements by tag name
Finding HTML elements by class name
Finding HTML elements by CSS selectors
Finding HTML elements by HTML object collections

element.innerHTML =	new html content	Change the inner HTML of an element
element.attribute =	new value	Change the attribute value of an HTML element
element.setAttribute(attribute, value)		Change the attribute value of an HTML element
element.style.property =	new style	Change the style of an HTML element

document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(element)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Events

JavaScript's interaction with HTML is handled through events.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

JavaScript lets you execute code when events are detected.HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

```
<button onclick="document.getElementById('d1').innerHTML = 'Hello'">Change</button>
```

HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.

Events are normally used in combination with functions, and the function will not be executed before the event occurs.

With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

With capturing, the event is first captured by the outermost element and propagated to the inner elements.

bubbles-- bubbles event property returns a Boolean value that indicates whether or not an event is a bubbling event.

cancelable -- can event cancelable.

createEvent

document.createEvent(type)

ctrlKey

for keyboardevent and mouseevent

currentTarget

returns the element whose event listeners triggered the event.

defaultPrevented

eventPhase

keyCode,which -- keyboardevent,mouseevent

pageX mouseevent

pageY mouseevent

preventDefault()

Cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur

stopPropagation()

target

timeStamp

type

change	content of a selection have changed
click	user clicks on an element
copy	user copies the content of an element
cut	user cuts the content of an element
dblclick	user double-clicks on an element
focus	The event occurs when an element gets focus
keydown	user is pressing a key
load	The event occurs when an object has loaded
message	The event occurs when a message is received
mousedown	user presses a mouse button
mouseover	pointer is moved onto an element

mouseout	user moves the mouse pointer out of an element
paste	user pastes some content in an element
resize	document view is resized
scroll	element's scrollbar is being scrolled
submit	form is submitted
unload	page has unloaded

Objects

objects in JavaScript are maps (dictionaries) from strings to values. A (key, value) entry in an object is called a property.

```
var s1 = {
    name: 'student1',

    describe: function () {
        return 'Person named '+this.name;
    }
};
```

accessing

```
s1.name

s1['name']

s1.describe()

s1['describe']()
```

setting new property

```
s1.id = 1
```

deleting property

```
delete s1.id
```

```
var s1 = {

    name:"s1",
    courses:["HTML5","CSS3","JS6"],
    show:function () {
                this.courses.forEach(
                    function (course) {  // (1)
                        console.log(this.name+' knows '+course);  // (2)
```

```
                                                  }
                                             );
                                        }
                    }
```

hasOwnProperty

Listing Own Property Keys

```
Object.keys(obj)

for (var x in obj)
        console.log(x);

Object.defineProperty(obj, 'foo', { writable: true });
```

[[Value]]

[[Writable]]

[[Enumerable]]

[[Configurable]]

Copying an Object

There are three levels of protecting an object, listed here from weakest to strongest:

```
        Preventing extensions       --          all properties configurable as false
        Sealing                                 --          sealing
        Freezing
```

```
Object.preventExtensions(obj)
```

instanceof

WebSocket

WebSockets are used to get the real time information from server.

server send information to all clients once it have new data.

```
Client          ------>         Server(JAVA .net NodeJS Python)
                      <-------
                      <-------
```

<-------